CONSTRUYA SU PROPIO SUPERCOMPUTADOR CON RASPBERRY PI

SERGIO ISERTE AGUT · SANDRA CATALÁN PALLARÉS
ROCÍO CARRATALÁ SÁEZ · SERGIO LÓPEZ HUGUET

CONTENIDOS WEB

Marcombo

Construya su propio supercomputador con Raspberry Pi

Un manual sencillo para iniciarse en la supercomputación desde el entretenimiento

Sergio Iserte Agut, Sandra Catalán Pallarés, Rocío Carratalá Sáez, Sergio López Huguet

Marcombo

Construya su propio supercomputador con Raspberry Pi

Primera edición, 2021

www.marcombo.com

Diseño de cubierta: ENEDENÚ DISEÑO GRÁFICO

Directora de producción: M.*a* Rosa Castillo

Revisión técnica: Ferran Fàbregas

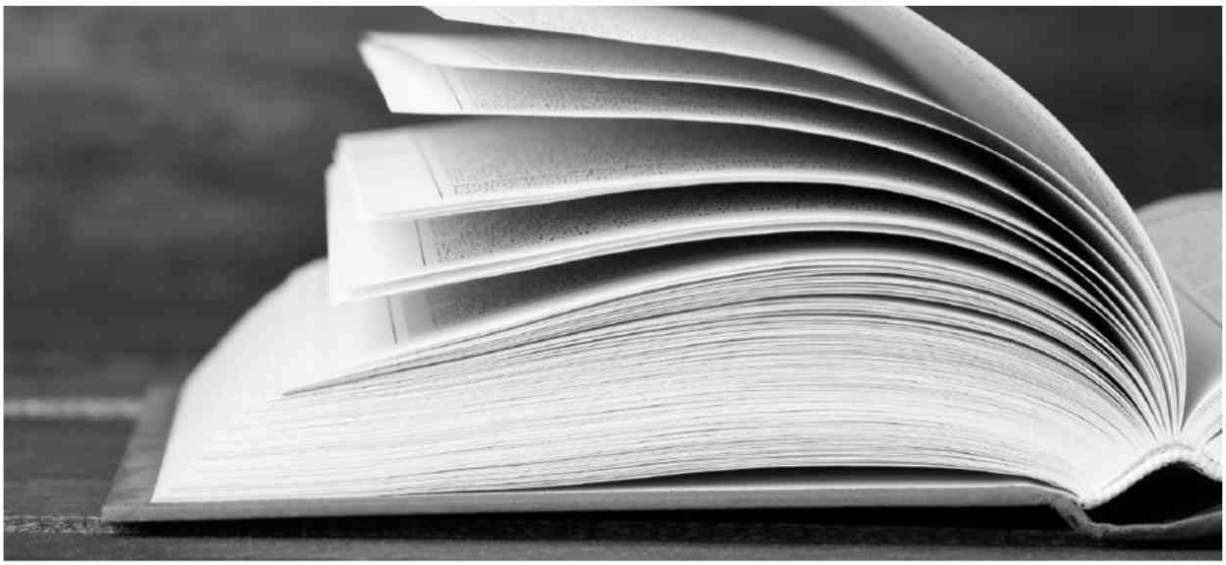
Corrección: Haizea Beitia

ISBN: 978-84-267-3333-7

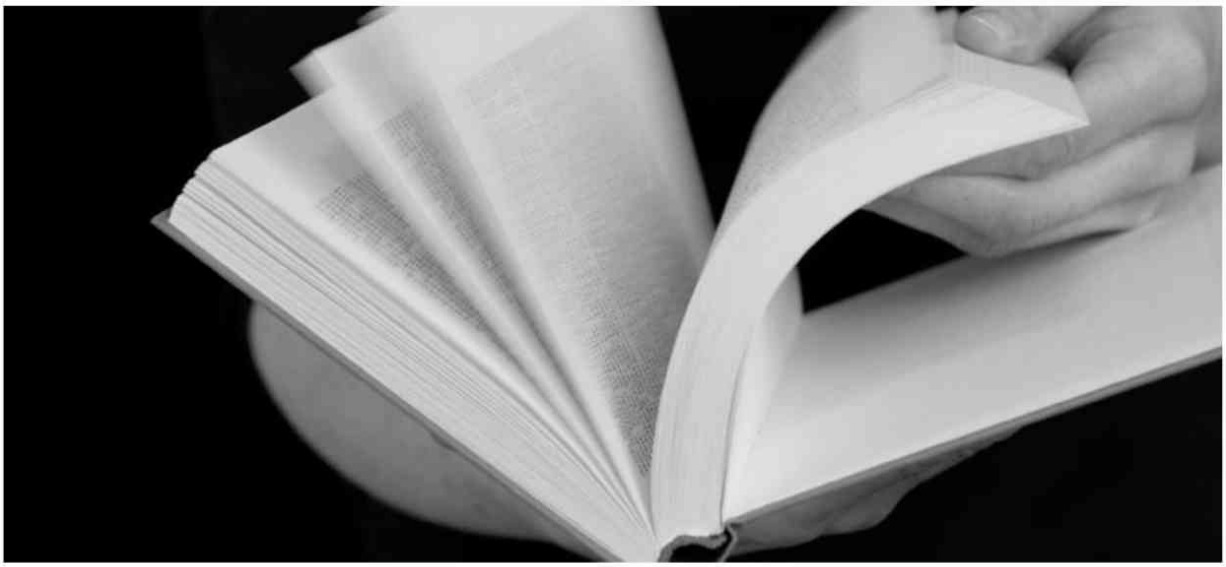
D.L.: B 6919-2021

Producción del ePub: booqlab

Para Enrique Quintana.



Índice general



¿Qué estoy leyendo?

Estimado/a lector/a:

Este libro pretende recoger una visión general e introductoria a la computación de altas prestaciones (*High Performance Computing*, en inglés, en adelante referida como HPC). A diferencia de otras obras, la presente no está centrada en la programación paralela o sus paradigmas, sino que profundiza en la administración, configuración y uso de sistemas HPC. Desde nuestra perspectiva, especialmente como alumnos, siempre echamos de menos una visión global del área que nos permitiera entender cómo cooperan los distintos componentes, tanto a nivel hardware como software, para crear entornos de alto rendimiento en los que ejecutar las aplicaciones pertinentes. Con la intención de proporcionar esa visión general y gracias a la experiencia que hemos ido adquiriendo al utilizar y formar parte del desarrollo de distintas infraestructuras, herramientas y aplicaciones, hemos elaborado el presente libro.

El contenido de este libro, además, incluye un componente práctico que detalla cómo construir, configurar y utilizar un "supercomputador" empleando hardware no específico para tal fin (en concreto, dispositivos

Raspberry Pi 4 Model B) e instalando software habitual en sistemas HPC. Debido a que el entorno hardware creado es un prototipo, en algunos casos se usan alternativas software que difieren de las comúnmente presentes en entornos reales, lo cual está debidamente especificado cuando ocurre. Esta propuesta práctica surge a partir de un curso no obligatorio creado e impartido a nivel universitario para dar a conocer la HPC, cuyos resultados siempre han sido valorados de forma muy positiva por los estudiantes asistentes.

Finalmente, dada la visión general y práctica que se presenta, este libro no pretende ser un curso de formación específica de ninguna de las distintas disciplinas que forman parte de la HPC, así como tampoco persigue sustituir ninguno de los manuales que se puedan encontrar con una finalidad similar. Consideramos que este es un buen punto de partida para quien quiera iniciarse en la HPC y lo animamos a seguir aprendiendo y descubriendo el área con la lectura de otros libros y manuales más específicos. Para tal fin, también hemos incluido referencias bibliográficas a lo largo del manual.

Sin más, agradecemos su interés en este libro. Deseamos que sea de su agrado y que le sirva como motivación para profundizar en el campo de la HPC.

Sergio Iserte Agut, Sandra Catalán Pallarés, Rocío Carratalá Sáez y Sergio López Huguet.

Sobre los autores de este libro

Sergio Iserte Agut es investigador postdoctoral (APOSTD20) del grupo de fluidos multifásicos (GFM) de la Universitat Jaume I (UJI), donde obtuvo su Doctorado en Informática (2018), Máster en Sistemas Inteligentes (2014) y título de Ingeniería Informática (2011). Además, Sergio combina la investigación con la docencia en el área de la computación de altas prestaciones (HPC) de la Universitat Oberta de Catalunya (UOC). Actualmente participa en proyectos de computación distribuida, gestión de recursos y modelado de cargas de trabajo en supercomputadores, a la vez que centra sus intereses en la investigación del aprendizaje automático en problemas físicos.

Sandra Catalán Pallarés es profesora de la Universidad Complutense de Madrid (UCM) y miembro del grupo de investigación *Architecture and Technology of Computing Systems* (ArTeCS) de la misma. Obtuvo su Doctorado en Informática en 2018 en la Universitat Jaume I (UJI), donde también se tituló en el Máster de Sistemas Inteligentes en 2013 e Ingeniería Informática en 2012. Ha colaborado con instituciones como el Barcelona Supercomputing Center (BSC), The University of Texas at Austin o IBM Research Zurich. Sus líneas de investigación incluyen la computación de altas prestaciones, los modelos de programación paralela, el consumo energético y las bibliotecas de álgebra lineal.

Rocío Carratalá Sáez es investigadora postdoctoral en la Universitat Jaume I (UJI) y miembro del grupo de investigación *High Performance Computing and Architectures* (HPC&A). En 2015 se graduó en Matemática Computacional por la UJI, en 2016 obtuvo el Máster en Computación Paralela y Distribuida por la Universitat Politècnica de València (UPV) y en 2021 se doctoró en informática por la UJI. Ha colaborado con instituciones como el BSC, Inria (Francia) o la Christian-Albrechts-Universität zu Kiel (Alemania). Sus líneas de investigación giran en torno a la computación de altas prestaciones y se centran en el desarrollo de bibliotecas de álgebra lineal. Además, tiene experiencia docente universitaria y colabora en proyectos de innovación relacionados.

Sergio López Huguet es estudiante de Doctorado en Informática en la UPV y miembro del grupo de investigación de Grid y Computación de Altas Prestaciones (GRyCAP) del Instituto de Instrumentación para Imagen Molecular (I3M) en la UPV desde 2016. En 2015 se graduó en Matemática Computacional por la UJI y en 2017 obtuvo el Máster en Ingeniería Informática por la UPV. Desde 2016 ha trabajado en distintos proyectos internacionales que le han permitido centrar sus líneas de investigación en la computación en la nube, la computación de altas prestaciones, el procesamiento de imagen médica y los contenedores.

Parte I

Introducción

Bienvenido/a a *Construya su propio supercomputador con Raspberry Pi*, un libro surgido de una experiencia universitaria y pensado para que quien lo lea disfrute aprendiendo sobre un tema poco conocido, aunque presente en muchos aspectos del día a día: la HPC.

¿A quién va dirigido este libro?

Este libro va dirigido a todas las personas que sienten interés por la ciencia en general y la computación en particular. Concretamente, el hilo conductor de este libro es la HPC. Basándose en la experiencia adquirida como docentes, los autores consideran importante disponer de conocimientos previos básicos sobre la terminal de Linux antes de profundizar en la lectura de este manual. Si adicionalmente se dispone también de formación básica en redes y programación, la comprensión de esta obra será más sencilla. No obstante, este libro trata de ser autocontenido, es decir, se proporcionan explicaciones detalladas sobre cada uno de los términos, comandos, protocolos y códigos mencionados y utilizados a lo largo del texto, respaldadas por un glosario y listado de acrónimos al final del mismo. El objetivo principal de esta obra es describir paso a paso el montaje, configuración y evaluación de un clúster de computadores formado por dispositivos Raspberry Pi. Esto no significa que el contenido explicado en este libro solo pueda ponerse en práctica mediante esos dispositivos y no con otros. Salvando las distancias en lo que a hardware, software y sus limitaciones o diferencias respecta, los desarrollos y actividades propuestos deben poder realizarse en sistemas distintos al descrito en este manual.

Por lo tanto, los autores consideran que, además de cualquier otra persona interesada, esta obra (o parte de ella) puede servir como guía básica para llevar a cabo proyectos sobre HPC enfocados tanto a estudiantes de bachillerato como de formación profesional y universitarios.

¿Qué se aprenderá con este libro?

El objetivo principal de este libro es que el lector entienda de forma global qué es, cómo funciona y para qué se utiliza la supercomputación. Además, presenta algunas de las herramientas comúnmente usadas en este campo, lo que facilita la iniciación en el uso de las mismas.

Por otro lado, a lo largo de todo el libro se presentan las instrucciones necesarias para construir paso a paso un clúster con dispositivos Raspberry Pi 4 Model B. Este prototipo de supercomputador servirá para poner en práctica todo lo explicado: desde el montaje de los componentes hardware hasta llegar a ejecutar software común en la HPC.

¿Cómo leer este libro?

Este libro está diseñado para poder poner en práctica lo aprendido a medida que se va leyendo, por lo tanto se recomienda una lectura ordenada y consecutiva de los capítulos. Aunque los capítulos son independientes unos de otros, están planteados para construir un supercomputador con dispositivos Raspberry Pi de manera ordenada, es decir, se parte de los elementos físicos hasta llegar a la ejecución de herramientas y aplicaciones, pasando por la configuración software de las mismas. Por tanto, si se pretende llevar a cabo la implementación física del supercomputador propuesto, no recomendamos alterar el orden de lectura ni obviar ningún capítulo.

Asimismo, si únicamente se está interesado en el contenido teórico, se recomienda el mismo modo de lectura, puesto que los conceptos se van tratando desde los más elementales hasta los más complejos. Cada capítulo incluirá una breve descripción de los conceptos y herramientas presentados, que siempre podrá ampliarse consultando la bibliografía propuesta para profundizar en ese tema concreto o para resolver dudas específicas.

Sobre la estructura del libro

El libro está estructurado en cinco partes que, a su vez, contienen uno o más capítulos. El contenido de cada una de estas partes se detalla a continuación:

- Parte I. Es la bienvenida al libro, donde se presenta el mismo y su contexto. En esta parte del libro se encuentran las instrucciones básicas sobre cómo moverse por el libro, cómo interpretar los cuadros que aparecen a lo largo de la lectura y cómo resolver las dudas que puedan surgir. Es la parte que se está leyendo en este momento.
- Parte II. Está formada por dos capítulos. En el Capítulo 1 se describen distintos aspectos relacionados con la HPC, mientras que en el Capítulo 2 se detallan los componentes hardware que formarán el clúster basado en Raspberry Pi.
- Parte III. Constituye la parte práctica del libro y está compuesta de varios capítulos que abordan desde la configuración básica del clúster hasta la ejecución de herramientas HPC. El Capítulo 3 presenta el sistema operativo, una breve descripción de los comandos que se usarán a lo largo del libro y el ensamblado del hardware que compone el supercomputador Raspberry Pi. El Capítulo 4 explica la configuración software básica del clúster, también paso a paso. El Capítulo 5 introduce los modelos de programación más comunes en la HPC, basados en metodologías de computación de memoria compartida y distribuida. El Capítulo 6 introduce el uso de los gestores de trabajos y recursos en los supercomputadores. El Capítulo 7 recoge una serie de herramientas y aplicaciones científicas habitualmente utilizadas en entornos HPC. Finalmente, el Capítulo 8 describe los contenedores como herramientas de virtualización.
- Parte IV. El Capítulo 9 es el último del libro y en él se incluye una serie de actividades (y propuestas de soluciones) referentes a todo el software presentado a lo largo del libro. El objetivo es profundizar en la utilización

y análisis de sistemas HPC.

- Parte V. Incluye dos anexos. En el Anexo A se detallan los tiempos aproximados para completar distintas configuraciones, compilaciones y ejecuciones sobre el clúster propuesto en el manual (basado en Raspberry Pi 4 Model B). En el momento de la redacción de este libro, Raspberry Pi 4 Model B es la versión más reciente de Raspberry Pi. De forma complementaria, el Anexo B contiene los detalles para construir el clúster usando el modelo anterior Raspberry Pi 3 Model B+.

Nótese que las estimaciones temporales incluidas en los anexos no tienen en cuenta el tiempo correspondiente a la lectura y comprensión de los capítulos, ni tampoco el necesario para completar las actividades propuestas.

Tras los anexos, se han incluido una lista de acrónimos y un glosario para facilitar la consulta de los principales términos utilizados a lo largo del libro.

Sobre fragmentos de código

A lo largo del libro se utilizan cuadros de texto para presentar conjuntos de comandos y sus resultados de ejecución, así como el contenido de ficheros.

Para facilitar la escritura de los códigos más largos, así como la de los ficheros de configuración de determinadas aplicaciones, se puede acceder al repositorio del libro[2].

A continuación, se describe el formato de cada uno de ellos con la finalidad de que sean fácilmente identificables.

Comandos y resultados de ejecuciones

La configuración de cada uno de los dispositivos Raspberry Pi que componen el clúster se presenta, tal como se ha indicado previamente, de forma

gradual, agrupando bloques de comandos según su finalidad. Dichos conjuntos de comandos y las correspondientes ejecuciones se muestran en figuras que simulan la consola del sistema operativo, como la siguiente:

```
COMENTARIO: Esto es un ejemplo de comentario
usuario@nodo:~ $ Esto es un ejemplo de comando a ejecutar
Esto es un ejemplo de resultado de ejecución
```

Los comandos a ejecutar se presentan precedidos por el *prompt* de la consola con el formato usuario@nodo :~ $ y los resultados de ejecución de los comandos aparecerán justo a continuación. Por otro lado, los comentarios respecto a ciertos comandos o conjuntos de ellos que se incluyan dentro de estas figuras para hacer alguna aclaración se muestran precedidos por la palabra COMENTARIO:. Cabe añadir que los comentarios que se incluyan dentro de estos cuadros no deberán ser trasladados a la consola; en ella solamente deberán escribirse los comandos y podrán observarse, tras su ejecución, los resultados correspondientes.

Ficheros

Cuando sea necesario modificar el contenido de un fichero, se mostrará un cuadro de texto similar a este:

Fichero: /lugar_de_almacenamiento/hola_mundo.c

```
#include <stdio.h>
int main() {
    // Esto es un ejemplo en el que una línea del fichero↩
         sigue en la línea inferior
    printf("Hola, mundo!\n");
    return 0;
}
```

En este tipo de cuadros, en la cabecera (en color gris oscuro) se muestra el nombre del fichero a modificar y dónde se encuentra almacenado. El contenido que debe tener el mismo se muestra a continuación, en el cuerpo del cuadro (en color gris claro). Si una línea dentro del fichero es muy larga, esta se divide en varias líneas. El carácter - es el indicador de que una determinada línea continúa en la línea siguiente y, por tanto, al seguir los pasos del manual, deben escribirse como una única línea todas aquellas relacionadas con dicho símbolo.

Aplicaciones recomendadas

Los desarrollos y configuraciones presentados en este libro se realizan directamente sobre los nodos Raspberry Pi, utilizando las aplicaciones preinstaladas en los mismos, sin que sea necesario ningún entorno de desarrollo integrado (*Integrated Development Environment* [IDE], en inglés).

Si bien pueden utilizarse (previa instalación si fuera necesario) otras aplicaciones similares, los autores harán referencia a las siguientes aplicaciones, ya preinstaladas en el sistema operativo escogido:

- Terminal del sistema (consola) para la ejecución de comandos.
- Editor de textos (por ejemplo, Mousepad si la interfaz gráfica está habilitada, o Nano en caso contrario).

Invitaciones a la reflexión

En la conclusión de cada capítulo se incluye un apartado titulado "Para profundizar..." en el que se plantean distintas cuestiones con la intención de promover la reflexión por parte del lector. En algunas de ellas se proporciona información relacionada, si bien en ninguna se da una respuesta cerrada.

Sobre el material recomendado

Al final de cada apartado se proporcionará una serie de recursos

bibliográficos, como por ejemplo:

- Libros y capítulos de libro.
- Artículos científicos, páginas web o artículos periodísticos.
- Enlaces a la documentación oficial de las herramientas o interfaces de programación de aplicaciones (*Application Programming Interface* [API], en inglés).

En el caso de que alguno de los enlaces web indicados para ampliar información sobre un determinado recurso no esté disponible cuando se realice la consulta, una búsqueda rápida del término en cuestión en cualquier buscador web debería servir para encontrar la información deseada.

¿Cómo conseguir ayuda?

Si con los recursos al final de cada tema no es suficiente para resolver las dudas, en general, los autores recomiendan hacer uso de los siguientes sitios de la plataforma Stack Exchange:

- https://raspberrypi.stackexchange.com : preguntas y respuestas de usuarios de Raspberry Pi.
- https://serverfault.com : preguntas y respuestas de administradores de sistemas y redes.
- https://stackoverflow.com : preguntas y respuestas de profesionales y desarrolladores de software.
- https://unix.stackexchange.com : preguntas y respuestas de usuarios de Linux y otros sistemas operativos de la familia Unix.
- https://superuser.com : preguntas y respuestas de entusiastas de los ordenadores con permisos de administración.

Si quedan cuestiones por resolver o se detecta algún error, se puede contactar con los autores vía correo electrónico:

- Sergio Iserte Agut: siserte@uji.es

- Sandra Catalán Pallarés: scatalan@ucm.es
- Rocío Carratalá Sáez: rcarrata@uji.es
- Sergio López Huguet: serlohu@upv.es

Recomendamos que, aunque se dirija a un autor en concreto, ponga al resto en copia para poder resolver la duda más rápidamente.

Llegados a este punto, ¡es momento de empezar esta aventura!

1 R. Carratalá-Sáez, S. Iserte and S. Catalán, "Teaching on Demand: an HPC Experience", 2019 IEEE/ACM Workshop on Education for High-Performance Computing (EduHPC), Denver, CO, USA, 2019, pp. 32-41.

2 Entre en www.marcombo.info con el código promocional PI3 para acceder al repositorio del libro.

Parte II

Fundamentos

¿Qué es la supercomputación? ¿Por qué se necesitan supercomputadores? ¿Qué marca la diferencia entre un supercomputador y un ordenador personal? ¿Qué se necesita saber para poder utilizar un supercomputador? Estas son solo algunas de las preguntas que se podría estar haciendo.

En el primer capítulo de esta parte (Capítulo 1) se dará respuesta a las preguntas planteadas anteriormente mediante la explicación de distintos conceptos. Este capítulo pretende poner en contexto al/a la lector/a. Por este motivo se tratarán aspectos muy diversos, pero todos ellos clave para iniciarse en el mundo de la HPC.

En el Capítulo 2 se describen los principales componentes de la Raspberry Pi 4 Model B. Puesto que este dispositivo constituye la base para la construcción del supercomputador propuesto en este libro, es necesario conocer los elementos que lo componen. Además, se establecerán paralelismos entre los componentes de la Raspberry Pi y sus equivalentes en un supercomputador real.

1. ¿Qué es la supercomputación?

1.1 Introducción

Ingentes cantidades de datos (comúnmente referidas como Big Data), provenientes de campos muy diversos, deben ser evaluadas a diario en cualquier contexto: en los análisis estadísticos económicos o sociales de las empresas privadas, en el tratamiento de los datos sensibles que manejan las entidades públicas, en la selección de los anuncios que se muestran a cada usuario mientras navega por Internet, en las sugerencias de contactos en las redes sociales y en un larguísimo etcétera de circunstancias muy diversas.

Para que esto sea posible, se dispone de clústeres HPC o supercomputadores capaces de procesar toda esa información que resulta crucial para empresas, instituciones, agrupaciones o administraciones.

Ahora bien, ¿en qué se debe pensar cuando se habla de supercomputadores? Un supercomputador puede definirse como aquel computador dotado de una arquitectura, recursos y componentes que, operando coordinados, permiten alcanzar una potencia computacional masiva. Los supercomputadores en producción están formados por miles de procesadores que pueden trabajar conjuntamente sobre un problema concreto.

¿Cómo de "masiva" es esa potencia computacional y cómo puede saberse que no se está ante un ordenador común, sino que se ha pasado a la escala de la supercomputación? Para resolver esta cuestión presentamos la HPC.

1.2 Computación de altas prestaciones (HPC)

El término HPC hace referencia a la resolución de problemas científicos y de ingeniería computacionalmente complejos y costosos (tanto a nivel de recursos como a nivel temporal), apoyados en el procesamiento paralelo y distribuido. El rendimiento necesario en la HPC requiere de miles de nodos potentes trabajando simultáneamente con un gran ancho de banda y una latencia baja en la red que los comunica, gracias a los cuales se obtiene un resultado derivado de cálculos complejos en un tiempo razonable, notablemente inferior al que conllevaría realizar dichos cálculos en un ordenador con prestaciones comunes.

De entre las innumerables aplicaciones donde la HPC es esencial (desde el ámbito de la investigación universitaria hasta el mundo empresarial), pueden destacarse las predicciones meteorológicas como uno de los casos de uso que evidencia la necesidad de inmediatez en la obtención de resultados. Con el fin de predecir *el tiempo que va a hacer mañana*, se realizan miles y miles de simulaciones de cálculo numérico y análisis estadísticos que, individualmente, podrían llegar a tardar horas o incluso días en completarse si se ejecutaran en computadores comunes, lo que implicaría no tenerlos disponibles a tiempo y que, consecuentemente, perdieran toda su utilidad y carecieran de sentido.

Es en los supercomputadores donde puede desarrollarse la HPC. Por supuesto, según el presupuesto disponible tanto para su configuración como para su mantenimiento (consumo energético, administración, actualización de componentes, etc.), distintos supercomputadores presentan diferencias en términos de rendimiento.

1.3 Evaluación del rendimiento en la HPC

Si hay algo indispensable en la HPC es determinar el rendimiento de un sistema para evaluar cómo se comporta y si se está explotando al máximo. Para ello, es común la utilización de *benchmarks*. En computación, un *benchmark* es un algoritmo (o conjunto de ellos) que se utiliza para medir el rendimiento de un determinado sistema y, si es necesario, poder realizar comparativas. Tanto para realizar las comparativas como para conocer el funcionamiento del propio sistema, a menudo se analizan la escalabilidad y la eficiencia de esos algoritmos.

1.3.1 FLOPS

La aritmética utilizada en computación para representar números reales se denomina coma flotante. En ella, los valores se expresan de manera aproximada, dado que el número de decimales representables físicamente en un computador es limitado. Aquellas operaciones matemáticas que el procesador es capaz de realizar teniendo en cuenta dicha aritmética se denominan operaciones en coma flotante (*Floating Point Operation* (FLOP), en inglés).

Las operaciones en coma flotante por segundo (*Floating Point Operations per Second* [FLOPS], en inglés) son la unidad de medida para el rendimiento de cualquier computador. La unidad FLOPS puede ir precedida de un prefijo que facilita la lectura del rendimiento de un computador, puesto que este puede variar mucho dependiendo de si se trata de un dispositivo móvil, un ordenador personal o un supercomputador. Para ello, se hace uso de los prefijos del Sistema Internacional recogidos en la Tabla 1.1.

Medida	Unidad	FLOPS
Kilo-FLOPS	KFLOPS	103

Mega-FLOPS	MFLOPS	106
Giga-FLOPS	GFLOPS	109
Tera-FLOPS	TFLOPS	1012
Peta-FLOPS	PFLOPS	1015
Exa-FLOPS	EFLOPS	1018
Zetta-FLOPS	ZFLOPS	1021
Yotta-FLOPS	YFLOPS	1024

Tabla 1.1 Prefijos del Sistema Internacional combinados con FLOPS.

1.3.2 Análisis de escalabilidad

La escalabilidad de un sistema es la propiedad que indica cuán bien responde el mismo al cambiar su configuración o capacidad. Por ejemplo, en un sistema con dos procesadores, sería deseable que cualquier programa ejecutado en él (en paralelo) tardase la mitad de tiempo que si el mismo sistema dispusiese únicamente de un procesador. Es decir, en este caso esperamos que:

$$\frac{T_{secuencial}}{T_{paralelo}} = 2,$$

o lo que es lo mismo, esperamos que el sistema con dos procesadores sea capaz de hacer el doble de operaciones por unidad de tiempo:

$$\frac{FLOPS_{paralelo}}{FLOPS_{secuencial}} = 2.$$

En la práctica, este valor es teórico, ya que existen diversos motivos tanto a nivel hardware como software que impiden una escalabilidad idealmente

lineal. En cualquier caso, la escalabilidad teórica es una buena referencia para estimar el buen funcionamiento de un sistema.

Para calcular la escalabilidad teórica, habitualmente se utiliza la Ley de Amdahl, que permite hacer una estimación más precisa. Esto se debe a que tiene en cuenta durante qué fracción de tiempo se puede utilizar el sistema mejorado. Supongamos que el 25% de un programa puede ser paralelizado; si se dispone de dos procesadores, esto significa que durante el 25% del tiempo de la ejecución secuencial podrían aprovecharse los dos procesadores para la ejecución paralela y así reducir el tiempo total. Para hacer un cálculo más preciso de la mejora a obtener, se aplica la Ley de Amdahl:

$$Escalabilidad = \frac{1}{(1-F)+\frac{F}{M}}$$

donde F es la fracción de tiempo del sistema original que puede beneficiarse de la mejora del sistema y M es la mejora aplicada. En el ejemplo, la fracción de tiempo es el 25%, que expresado en tanto por uno es 0,25; la mejora es 2, puesto que el sistema base tendría un único procesador y el mejorado dos. En este caso la escalabilidad obtenida sería la siguiente:

$$\frac{1}{(1-0{,}25)+\frac{0{,}25}{2}} \approx 1,14$$

Es decir, puede esperarse que la ejecución paralela que aprovecha los dos procesadores disponibles durante el 25% del tiempo de ejecución sea un 14% más rápida que la equivalente secuencial.

1.3.3 Análisis de eficiencia energética

Con la evolución de los sistemas ha aumentado su complejidad, a la par que su consumo energético. La electricidad necesaria para mantener un sistema funcionando ha ido aumentando a lo largo del tiempo, por lo que tener

en cuenta este parámetro a la hora de valorar el rendimiento es algo cada vez más común. La eficiencia energética (E) permite calcular el rendimiento que obtiene el sistema en función de la energía consumida. Para ello, se calculan los FLOPS que se computan por vatio (W):

$$E = \frac{FLOPS}{W}$$

Tanta importancia ha ido cobrando la evaluación del consumo energético en entornos HPC que no solo se clasifican los supercomputadores en función de su rendimiento, sino también en base a su eficiencia energética.

Para más material visita:
https://dogramcode.com/bloglibros/libros-electronica

1.4 El TOP500

En la comunidad científica, empresarial o cualquier otra donde la HPC sea de interés, también lo es conocer cuáles son los rendimientos de los supercomputadores más potentes existentes, así como su configuración, localización y aplicaciones para las que se utilizan.

Desde 1993, con el fin de facilitar a la comunidad científica una clasificación en la que basarse para establecer colaboraciones y para tener una referencia propia respecto a la actualidad de la HPC a nivel global, se elabora una lista que recoge los 500 sistemas computacionales más potentes a nivel mundial. Esta lista se actualiza bianualmente desde entonces en junio y noviembre. Recibe el nombre de TOP500[1] y la participación en ella es voluntaria.

Con el propósito de decidir qué supercomputadores optan a formar parte de la clasificación del TOP500 y qué posiciones ocupan en la misma, se utiliza LINPACK para evaluar el rendimiento en términos de FLOPS. LINPACK[2] es un *benchmark* ampliamente utilizado en computación científica, compuesto por una colección de subrutinas escritas en lenguaje Fortran que analizan y resuelven sistemas de ecuaciones lineales y problemas de mínimos cuadrados. Dispone, además, de una versión más reciente enfocada a la HPC llamada HPL[3] (*High Performance* LINPACK). Sobre la instalación y uso de este *benchmark* se profundizará más adelante (Apartado 7.1.6), cuando se instale en el clúster propuesto en este manual.

1.5 El Green500

La comunidad científica no es ajena a la realidad ni a los grandes problemas que preocupan en la actualidad. El cambio climático y los efectos del alto consumo energético que los humanos llevan a cabo es una de las cuestiones más importantes a analizar y resolver en cualquier contexto. Por ello, en abril de 2005 el Dr. Wu-chun Feng introdujo la idea de elaborar un listado con los superordenadores más potentes ordenados según su eficiencia energética[4], en paralelo a la lista TOP500. La presentación de esta misma idea de nuevo en 2006 ayudó a incrementar el interés de la comunidad por hacerla realidad y en 2007 se publicó la primera lista Green500[5]. En definitiva, el propósito último del Green500, tal como se manifiesta en su sito web, es "alentar a la comunidad HPC a garantizar que los supercomputadores solo simulen el cambio climático y no contribuyan a él".

Del mismo modo que el TOP500, el Green500 también se actualiza bianualmente (en junio y en noviembre) y un histórico completo de los listados puede encontrarse actualmente en la web del TOP500[6]. En este listado se ordenan los supercomputadores que integran la lista del TOP500 según su eficiencia energética, la cual se mide en GFLOPS/vatio; es decir, se evalúa cuántas veces se realizan 10⁹ operaciones en coma flotante por segundo por cada vatio consumido.

1.6 Estructura clúster

Actualmente, la HPC se lleva a cabo en clústeres de computadores. Un clúster es un grupo de computadores independientes (llamados nodos) que pueden trabajar conjuntamente para resolver un problema. Las principales características de un clúster son las siguientes:

- Por lo general, los nodos del clúster tienen las mismas características hardware, aunque también se pueden encontrar clústeres heterogéneos.
- Los nodos están conectados a una o varias redes de comunicaciones privadas.
- Los nodos comparten espacio de almacenamiento.
- El acceso a los nodos se realiza mediante mecanismos de autenticación inherentes.

La Figura 1.1 muestra un esquema de conexión de los distintos actores que forman parte de un clúster básico de computación: los nodos de cómputo y el *front-end*.

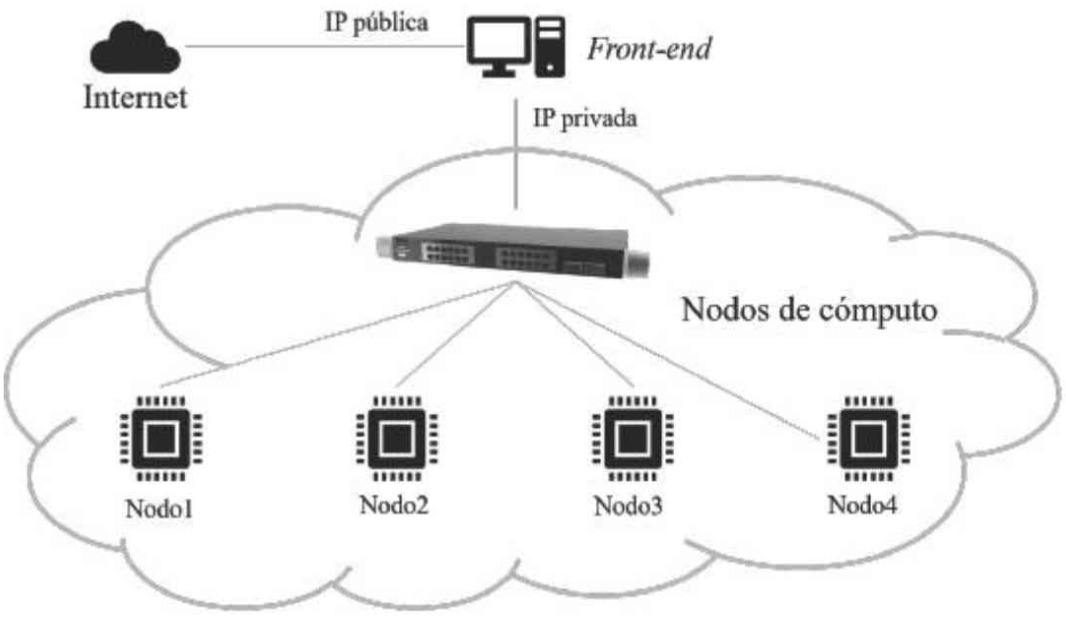


Figura 1.1 Esquema de la estructura de computación clúster y la conexión entre sus elementos.

El modo de trabajo habitual de los usuarios en estos sistemas es el siguiente:

1. Los usuarios, mediante su conexión a Internet, se conectan al *front-end* a través de la dirección IP pública de este.

2. Una vez se conectan, se autentican en el *front-end* , adquiriendo así acceso al clúster de computación.

3. Una vez autenticados, los usuarios interactúan con el *front-end* para utilizar el clúster, pues este es el nodo responsable de coordinar los nodos de cómputo a través de su red privada.

Normalmente, cada usuario tiene asignada una limitación de uso del clúster tanto en términos de nodos accesibles como en lo referente a tiempo y recursos computacionales disponibles. Además, los nodos de cómputo no tienen acceso a Internet, ya que comparten los directorios con el software necesario para los usuarios. Tampoco es habitual que los usuarios se conecten directamente a los nodos de cómputo, aunque en sesiones interactivas donde sí lo hacen, es siempre a través de la gestión del *front-end*.

En este apartado se ha descrito una estructura clúster genérica y su modo de operación. Como se puede apreciar, se usan continuamente términos como "habitualmente", "normalmente", "predominantemente", etc. Esto es debido a que existen muchas configuraciones alternativas, variantes y casos especiales que quedan fuera del alcance de este libro.

1.7 Consumo energético y refrigeración

Desde que surgió el primer procesador hasta la actualidad, los sistemas informáticos han incrementado su consumo energético notablemente debido a la complejidad de los nuevos componentes. No en vano, han ido surgiendo sistemas cuyo objetivo es mantener bajo control el consumo energético a la vez que garantizar un rendimiento adecuado a las necesidades, desde el segmento doméstico hasta la HPC. Este es el caso, por ejemplo, de Raspberry Pi, un dispositivo diseñado para tener un bajo coste y consumo.

Vinculado a ese consumo energético surge la necesidad de refrigerar el sistema. Cualquier sistema de cómputo, independientemente de su finalidad y rendimiento, suele disponer de un sistema de refrigeración que garantiza que sus componentes se mantienen a una temperatura adecuada para su correcto funcionamiento. El sistema más común de refrigeración es por aire, puesto que resulta más económico y su instalación suele ser más sencilla. Sin embargo, en el caso de los supercomputadores, los sistemas de refrigeración por aire no suelen ser convenientes, ya que elevan notablemente el coste de refrigeración. Como alternativa, se usa habitualmente refrigeración líquida, o una refrigeración híbrida que combina refrigeración líquida y por aire. La refrigeración líquida consiste en enfriar un líquido (agua, aceite mineral o fluoroplásticos líquidos) que se hace llegar a las distintas partes de la máquina a través de tubos o capilares.

Habitualmente, refrigerar el supercomputador supone un gran coste en términos económicos, que puede alcanzar el 50% del coste total de la energía empleada por la infraestructura[7]. Por este motivo, disponer de un sistema de refrigeración adecuado es indispensable para mantener el sistema en funcionamiento y hacerlo con un coste razonable.

1.8 Sistemas multiprocesador y multinúcleo

En la actualidad la gran mayoría de sistemas cuentan con más de una unidad de procesamiento, es decir, uno o más procesadores, también conocidos como microprocesadores o unidades centrales de procesamiento (*Central Processing Unit* [CPU], en inglés), compuestos a su vez de uno o más núcleos (*cores*, en inglés). Por simplicidad, no se tratarán en este manual otros tipos de unidades de procesamiento adicionales, como pueden ser las unidades de procesamiento gráfico (*Graphics Processing Unit* [GPU], en inglés).

En pocas palabras, un computador puede albergar uno o varios procesadores que, a su vez, pueden estar compuestos por uno o varios núcleos. Los núcleos pueden contar con más de un conjunto de recursos hardware (*Simultaneous MultiThreading* [SMT], en inglés), lo que se conoce como procesador lógico (también se le llama comúnmente *hyper-threading*, aunque este nombre pertenece a la implementación SMT propia de Intel).

A continuación, se proporcionan algunas definiciones útiles de cara a comprender y diferenciar correctamente los términos, no solamente en las explicaciones o aclaraciones presentadas a lo largo de este manual, sino también en la documentación propia del contexto general de la HPC.

- Procesador: conjunto de una o más unidades de procesamiento que comparten la memoria principal y los periféricos.
- Núcleo: conjunto completo de registros y unidades de ejecución dentro de un procesador.
- Procesador lógico: contexto hardware dentro de un núcleo.

La Figura 1.2 muestra la jerarquía de unidades de procesamiento en un computador con dos procesadores de cuatro núcleos, cada uno de ellos con dos unidades simultáneas de procesado, lo que ofrece un total de ocho núcleos y dieciséis procesadores lógicos.

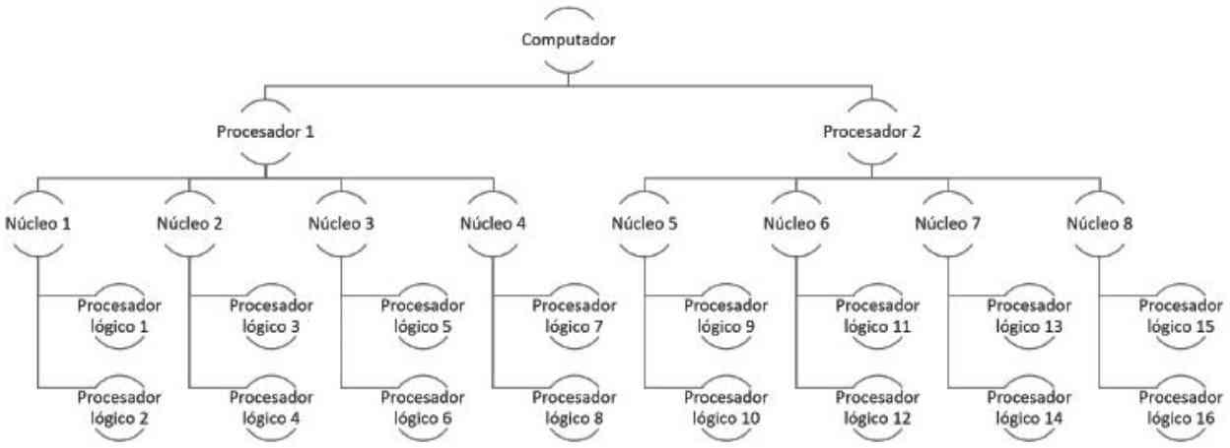


Figura 1.2 Ejemplo de jerarquía de unidades de procesamiento en un computador.

Nótese que en entornos de HPC es habitual utilizar el núcleo como unidad básica de procesamiento, aunque el procesador soporte SMT.

1.9 Concurrencia y paralelismo

Ligados a los términos *procesador lógico* y *núcleo*, a nivel software encontramos los conceptos de *proceso* e *hilo*. Un proceso es una entidad independiente de ejecución, gestionada por el sistema operativo, que se ejecuta en un momento dado en un núcleo del procesador. Un proceso está compuesto por:

- las instrucciones de programa a ejecutarse;
- su estado de ejecución en un momento dado (información que guarda la CPU sobre ese proceso);
- la memoria en la que se almacenan los datos con los que trabaja el proceso; y
- otra información que permite al sistema operativo decidir qué proceso se ejecuta en cada momento.

El sistema operativo se encarga de crear y destruir procesos, así como de planificar su ejecución y gestionar la comunicación entre los mismos. Es importante no confundir "programa" con "proceso"; un programa puede ejecutarse en un único proceso, pero también puede estar compuesto de varios procesos que se ejecutan simultáneamente, aunque no empiecen y terminen necesariamente a la vez. Esto es lo que se conoce como *concurrencia*.

Los hilos, también denominados en la literatura como hebras (*threads*, en inglés), son entidades de ejecución independientes que viven dentro de los procesos. Por lo tanto, comparten el código, los datos almacenados en memoria y aquellos recursos del sistema operativo de que disponga el proceso, por ejemplo los ficheros abiertos. Esta característica de los hilos hace que su comunicación sea mucho más sencilla, puesto que pueden comunicarse usando el espacio de memoria que comparten sin necesidad de involucrar al sistema operativo. Además, la creación de un nuevo hilo en un proceso existente por parte del sistema operativo (así como su terminación o su conmutación

en un determinado núcleo del procesador) puede hacerse de forma mucho más rápida que si se tratara de un proceso, dado que la información específica que el sistema operativo debe manejar es mucho más reducida.

Por otro lado, los hilos se pueden ejecutar concurrentemente dentro de un proceso, lo que da lugar al concepto de *paralelismo*. El paralelismo es un caso concreto de concurrencia en el que se plantea un escenario donde la resolución de un problema puede acelerarse mediante la filosofía "divide y vencerás". De esta forma, cada hilo procesa una parte del problema y existe una última etapa en la que los resultados parciales calculados en cada hilo son combinados para dar el resultado final. Por ejemplo: imagine que necesita calcular la suma de 100 números; si se dispone de cuatro hilos para hacer el cálculo, se puede dividir el cálculo total en cuatro partes, de manera que cada hilo se encargará de hacer la suma parcial de 25 elementos y, una vez calculadas las sumas parciales, uno de los hilos será el encargado de combinar los cuatro resultados parciales para calcular el final.

Dada la posibilidad de que un computador sea capaz de trabajar con varios procesos e hilos, existen cuatro escenarios distintos posibles (ver Figura 1.3). El más común actualmente es el uso de varios procesos multihilo, lo que permite una mayor flexibilidad y control a la hora de mejorar el rendimiento de las aplicaciones.

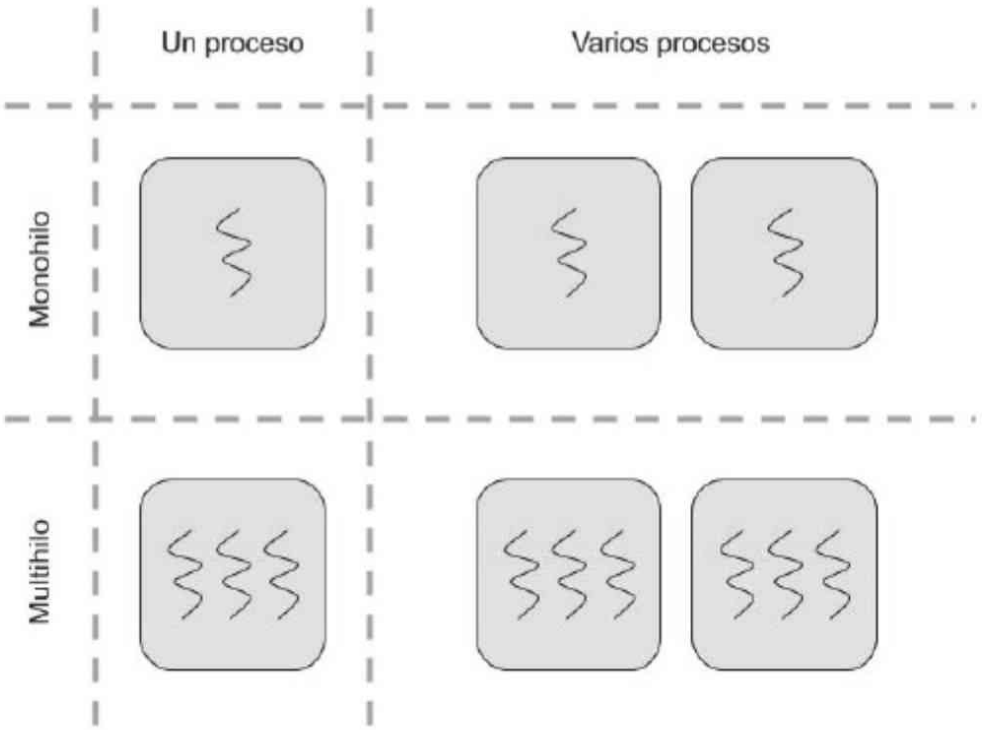


Figura 1.3 Posibles combinaciones de procesos e hilos en un computador.

1.10 Virtualización

La virtualización es una tecnología que permite la emulación de un entorno informático (de manera completa o parcial) dentro de otro entorno informático físico.

La virtualización puede involucrar uno o distintos aspectos: entornos totalmente configurados, red, almacenamiento o solamente algunas aplicaciones. Un tipo de virtualización son las máquinas virtuales, las cuales proporcionan al usuario un entorno informático prácticamente completo e indistinguible de un sistema físico gracias a la emulación tanto del software como de los recursos físicos o hardware. Las máquinas virtuales son controladas por un software llamado hipervisor, el cual es el encargado de asignar y proporcionar acceso a los distintos recursos físicos (por ejemplo, la CPU o la red) a las máquinas virtuales.

Otra herramienta de virtualización son los contenedores. Estos son programas ejecutables que, al contrario que las máquinas virtuales, solamente contienen las aplicaciones y sus dependencias. Los contenedores no son una emulación tan compleja de un sistema como las ofrecidas por las máquinas virtuales, sino que, aprovechando ciertas tecnologías disponibles, permiten ejecutar aplicaciones en la máquina donde se hospedan de manera aislada (sin poder interactuar con otras aplicaciones externas al contenedor), por lo que virtualizan solamente una parte del entorno informático y no su totalidad.

Los contenedores se han convertido en una tecnología muy popular para la portabilidad de aplicaciones o herramientas entre los usuarios. Esto es debido a que los contenedores son más ligeros que las máquinas virtuales y pueden ser generados a partir de una receta (*recipe*, en inglés) que recoge las especificaciones mediante las cuales se define y configura el contenedor. Más detalles sobre esto se proporcionarán en el Capítulo 8.

El uso de máquinas virtuales en los supercomputadores no está muy extendido por el elevado consumo de recursos que ocasiona el hipervisor. Sin embargo, debido a que los contenedores no son controlados por el hipervisor, sino que son controlados de la misma forma que el resto de procesos nativos de la máquina que los hospeda, estos han comenzado a popularizarse en entornos HPC. Un ejemplo de ello[8] es el supercomputador Fugaku, el cual es el número 1 de la lista TOP500 de noviembre de 2020[9].

1.11 Lenguajes de programación para HPC

La mayoría de programas que se ejecutaban en contextos HPC al comienzo se basaban en algoritmos matemáticos implementados en lenguaje Fortran[10]. No obstante, dada su versatilidad y herramientas para gestionar el almacenamiento y acceso a los datos en memoria, C[11] pasó a ser un lenguaje habitualmente presente en contextos HPC desde casi sus inicios. Desde hace algún tiempo, gracias a las similitudes entre C y C++[12], este último se ha convertido en un lenguaje de programación ampliamente extendido hoy en día en entornos HPC.

Alternativamente, gracias a la flexibilidad de los modelos de programación paralela que han ido surgiendo, enfocados a "facilitar" la adaptación de códigos secuenciales a una ejecución donde se aproveche la concurrencia y el paralelismo, diferentes lenguajes como Java[13] o Python[14] también han encontrado su hueco en la comunidad HPC y hoy en día existen muchas aplicaciones escritas en dichos lenguajes que logran una buena eficiencia paralela.

Lenguajes emergentes como X10[15], Chapel[16] o Fortress[17], entre otros, persiguen lograr altos niveles de productividad y rendimiento, sin perder programabilidad, portabilidad y robustez, basándose en los lenguajes clásicos utilizados en la HPC.

Asimismo, la aparición de nuevos sistemas HPC también genera nuevos lenguajes, entornos de programación y estándares que se ajustan a los mismos y que persiguen aprovechar su rendimiento al máximo. Por ejemplo, con la incorporación de las GPU surgieron CUDA[18] y OpenCL[19], cuya finalidad es aprovechar la capacidad de cómputo de estos componentes para acelerar la resolución de ciertas operaciones computacionalmente costosas.

En este libro, el principal lenguaje de programación empleado es C.

1.12 Para profundizar...

1. ¿En qué sistemas electrónicos utilizados en el día a día se puede apreciar concurrencia?

   Piense en los dispositivos electrónicos que usa a diario en los que se realizan varias tareas a la vez.

2. ¿Se le ocurren aplicaciones de la supercomputación en contextos que formen parte del día a día?

   Por ejemplo, en los escenarios en los que el tiempo de respuesta (es decir, lo que cuesta lograr el resultado de ejecución) es clave: ¿serviría de algo predecir mañana el tiempo de hoy?

3. Con la virtualización es posible emular desde un entorno completo hasta solo una aplicación informática. Pueden encontrarse, por tanto, distintos softwares de virtualización según su propósito.

   Por ejemplo, las aplicaciones desarrolladas en Java se ejecutan sobre un entorno virtualizado que previamente se tiene que instalar en la computadora. Por su parte, Wine[20] es un software que permite la ejecución de aplicaciones Windows dentro de sistemas Linux. En el caso inverso, WSL[21] es una capa de compatibilidad que permite ejecutar programas para Linux en Windows.

4. Explore la evolución de núcleos por *socket* en los supercomputadores del TOP500.
   - ¿Cuál es el número de núcleos por *socket* más popular

actualmente?

- ¿Hay alguna tendencia en el número de núcleos por *socket* que en los últimos años va cogiendo fuerza?

Puede hacerlo desde www.top500.org ⇒ "Statistics" ⇒ "Development over time" y en "Category" elegir "Cores per socket" para el tipo "Systems share".

1.13 Material recomendado

Las siguientes referencias bibliográficas fundamentan lo que se ha explicado en este capítulo y presentan más detalles e ilustraciones al respecto. Particularmente, en el artículo de investigación [1] se explica por primera vez la intención de elaborar un listado con los superordenadores más potentes ordenados según su eficiencia energética, en paralelo a la lista TOP500. Esta idea se materializa dos años después y se presenta en el artículo de investigación [2]. Así surge el Green500, como ya se ha explicado.

Por otro lado, este capítulo es meramente introductorio, por lo que se recomienda la lectura de determinados materiales para poder profundizar en las distintas áreas que componen la HPC. Por ejemplo, [3] es un libro sobre arquitectura de computadores que explica cómo aumentar el rendimiento de los sistemas teniendo en cuenta diversos aspectos tecnológicos y estudiando en profundidad el paralelismo. De forma similar, el libro [4] ofrece una detallada introducción a la computación paralela y cómo utilizarla. Además, también se recomienda el libro [5], que ahonda en la computación distribuida desde varias perspectivas.

Finalmente, el libro [6] proporciona al lector una guía para el estudio de la tecnología de contenedores Docker, así como indicaciones para el uso del orquestador de contenedores Kubernetes.

[1] Chung-Hsing Hsu, Wu-chun Feng, and Jeremy S. Archuleta. Towards Efficient Supercomputing: A Quest for the Right Metric. In *19th International Parallel and Distributed Processing Symposium (IPDPS 2005), CD-ROM / Abstracts Proceedings, 4-8 April 2005, Denver, CO, USA* . IEEE Computer Society, 2005.

[2] Wu-chun Feng and Kirk W. Cameron. The Green500 List: Encouraging

Sustainable Supercomputing. *IEEE Computer* , 40(12):50–55, 2007.

[3] J.O. Lopera, M.A. López, and A.P. Espinosa. *Arquitectura de computadores* . Thomson, 2005.

[4] A. Grama, V. Kumar, A. Gupta, and G. Karypis. *Introduction to Parallel Computing* . Pearson Education. Addison-Wesley, 2003.

[5] K. Hwang, J. Dongarra, and G.C. Fox. *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things* . Elsevier Science, 2013.

[6] G.N. Schenker. *Learn Docker – Fundamentals of Docker 19.x: Build, test, ship, and run containers with Docker and Kubernetes, 2nd Edition* . Packt Publishing, 2020.

1.14 Resumen

En este capítulo introductorio se han presentado los conceptos básicos que deben conocerse para entender todo lo que se desarrollará en los capítulos que siguen. En concreto, deben haber quedado claros (y en caso contrario, recomendamos su revisión antes de continuar la lectura) los siguientes términos:

- HPC, supercomputador, clúster y *front-end* .
- FLOPS, escalabilidad y eficiencia energética.
- Procesador, núcleo e hilo.
- Sistemas multiprocesador y multinúcleo, concurrencia y paralelismo.
- Virtualización, máquinas virtuales y contenedores.

Además, se han presentado dos de los ránkings más importantes en lo que a supercomputadores se refiere: el TOP500 y el Green500, que respectivamente se basan en la potencia computacional y la eficiencia energética para evaluar los distintos sistemas y listarlos de manera ordenada según el resultado obtenido.

Por último, se han introducido los principales lenguajes de programación que hoy en día se utilizan en contextos HPC.

1.14.1 ¿Qué viene a continuación?

En el Capítulo 2, se describen detalladamente los componentes del clúster basado en Raspberry Pi que se desarrollará a lo largo de este manual, junto con las definiciones introductorias y conceptos básicos asociados al mismo.

1 https://www.top500.org

2 http://www.netlib.org/linpack/

3 http://www.netlib.org/benchmark/hpl

4 https://research.cs.vt.edu/synergy/pubs/talks/feng-ipdp-s2005-powe-re_talk.pdfpdf

5 http://www.green500.org/

6 https://www.top500.org/lists/green500/

7 https://www.researchgate.net/publication/287392545_-Supercompu-s_Keeping_People_Warm_in_the_Winter

8 https://sylabs.io/2020/08/worlds-fastest-super-com-puter-ssingula-rity-?s=09?s=09

9 https://www.top500.org/lists/top500/list/2020/11

10 https://fortran-lang.org

11 https://www.iso.org/standard/74528.html

12 https://www.iso.org/standard/79358.html

13 https://www.oracle.com/java/

14 https://www.python.org/

15 http://x10-lang.org

16 http://chapel.cray.com/

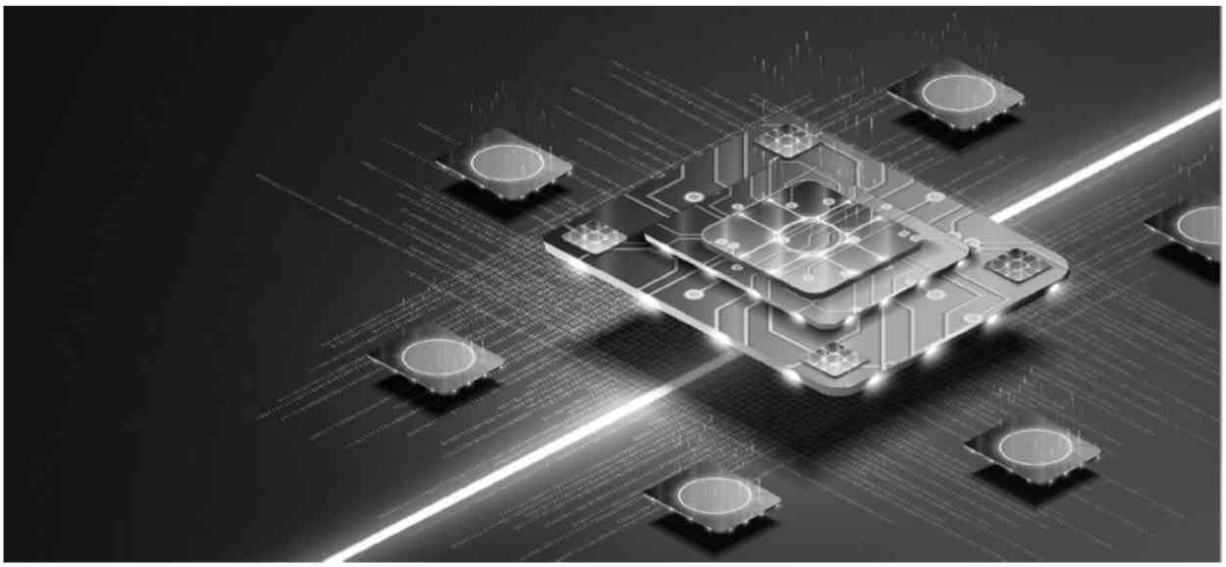
17 http://projectfortress.java.net/

18 http://developer.nvidia.com/category/zone/cuda-zone

19 http://www.khronos.org/opencl/

20 https://www.winehq.org/

21 https://docs.microsoft.com/en-us/windows/wsl/install-win10



2. Clúster con Raspberry Pi

2.1 ¿Por qué Raspberry Pi?

El primer modelo de Raspberry Pi se lanzó el 29 de febrero de 2012. Salió al mercado como un ordenador "simple y pequeño" de bajo coste, especialmente enfocado al público joven y a los estudiantes, con la intención de motivarlos a aprender a programar y también a descubrir los distintos componentes de un ordenador. Hoy en día la Raspberry Pi es también la protagonista de muchos proyectos de domótica, robótica e ingeniería informática a pequeña escala, así como en distintos contextos educativos.

La Raspberry Pi puede definirse como un pequeño ordenador cuya placa base (al descubierto) integra el procesador, la memoria RAM, distintos puertos de conexión (red, USB, audio, vídeo, cámara...), una ranura para insertar una memoria que haga las funciones de disco duro y un conector para la fuente de alimentación. Algunos modelos están también provistos de receptor Wi-Fi y Bluetooth.

Dadas sus prestaciones, versatibilidad y bajo coste, hemos eligido la Raspberry Pi como elemento principal del pequeño supercomputador que se configurará a lo largo del libro.

2.1.1 El procesador

La Raspberry Pi cuenta con un procesador con arquitectura ARM (Acorn RISC Machine). La principal característica de los procesadores ARM es que siguen un diseño de CPU con un conjunto de instrucciones reducido (*Reduced Instruction Set Computer* [RISC], en inglés). Esto hace que los procesadores ARM sean más sencillos que los procesadores Intel o AMD, habitualmente presentes en los ordenadores personales, que se caracterizan por tener un conjunto de instrucciones complejo (*Complex Instruction Set Computer* [CISC], en inglés). Los programas que ejecuta un ordenador se traducen en

instrucciones a ejecutar por el procesador, pero ambas arquitecturas tienen distintos enfoques: para una misma operación, RISC ejecutará más instrucciones, que serán más simples; sin embargo, CISC utilizará un menor número de instrucciones, aunque más complejas.

El hecho de que ARM sea una arquitectura RISC tiene varias ventajas. Si bien es cierto que el conjunto de instrucciones es más limitado, lo que supone que la ejecución de un programa necesite más memoria para almacenar un mayor numero de instrucciones, esto también tiene su parte positiva, ya que ejecutar una menor variedad de instrucciones hace que el diseño del procesador sea más sencillo. Como consecuencia, serán necesarios menos transistores para su construcción, lo cual reduce el coste de fabricación, el tamaño y también la energía necesaria para alimentar el procesador.

Así pues, los procesadores ARM son idóneos para entornos en los que se requiere una buena relación entre potencia de cálculo y consumo de energía, como por ejemplo *smartphones, tablets,* reproductores multimedia, sistemas de control empotrados o incluso lavadoras, entre otros.

Dadas las mencionadas características de los procesadores ARM y puesto que Raspberry Pi incluye este tipo de procesador, la elección de este sistema se considera adecuada para poder llevar a cabo el desarrollo hardware propuesto a lo largo de este libro.

2.2 Raspberry Pi 4 Model B

En concreto, a lo largo de este libro se va a utilizar la Raspberry Pi 4 Model B, en adelante referida como RPi.

Las especificaciones técnicas de la RPi[1] proporcionadas por el fabricante se resumen en la Tabla 2.1. Los componentes presentados en esta tabla, que serán analizados más en detalle a lo largo de este capítulo, se disponen en la placa de RPi como se muestra en la Figura 2.1.

Procesador	Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC 1,5 GHz
Memoria RAM	2 GB LPDDR4-3200 SDRAM
Wi-Fi	LAN inalámbrico de 2,4 GHz y 5,0 GHz IEEE 802.11.b/g/n/ac
Bluetooth	Bluetooth 5.0, BLE
Puertos USB	2 × USB 2.0, 2 × USB 3.0
Puerto para cámara	Puerto MIPI CSI de dos pistas
Puerto para monitor	2 Puertos micro-HDMI (soporta resolución hasta 4Kp60)
Puerto para vídeo/audio	Puerto de 4 polos para audio estéreo y vídeo compuesto
Puerto para alimentación	5V/3A DC mediante USB-C
Puerto Ethernet	Gigabit Ethernet
Otros	Puerto MIPI DSI de dos pistas, 40 Pines GPIO

Tabla 2.1 Especificaciones técnicas de la Raspberry Pi 4 Model B.

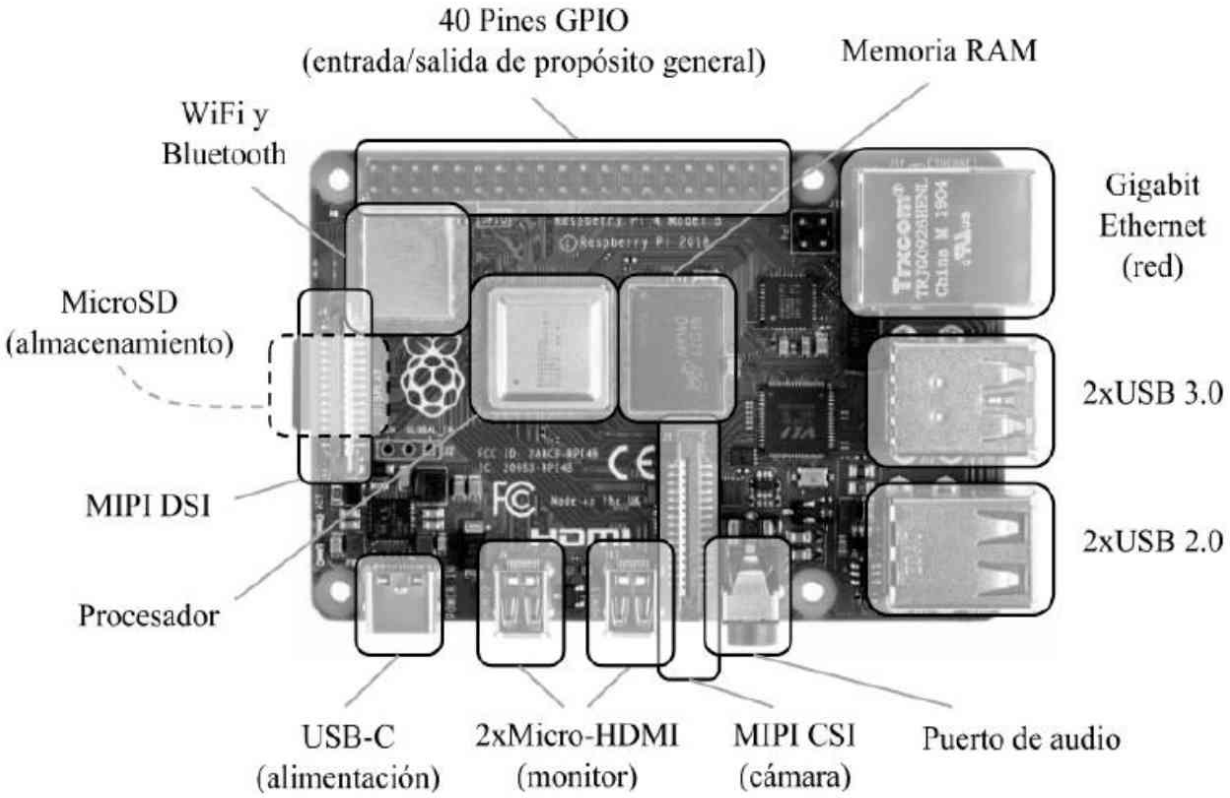


Figura 2.1 Esquema de los componentes que integran una Raspberry Pi 4 Model B.

2.2.1 Sensor de temperatura incorporado

Cada nodo RPi incorpora un termómetro o sensor de temperatura, cuyo valor en un momento puntual puede consultarse mediante el comando que se muestra a continuación:

```
pi@raspberry:~ $ cat /sys/class/thermal/thermal_zone0/temp 43470
```

Nótese que el comando mostrado devuelve el valor de temperatura en grados centígrados multiplicado por mil. Por tanto, en el ejemplo anterior, la temperatura es de 43,47 °C.

Debido a la disposición de los componentes en la RPi, las consultas de

temperatura realizadas en este dispositivo mediante el comando anterior pueden presentar cierto grado de inexactitud. Alternativamente, existe un comando que puede proporcionar valores más precisos de la temperatura instantánea, gracias a que directamente toma el valor a partir de la GPU integrada en el microprocesador:

```
pi@raspberry:~ $ /opt/vc/bin/vcgencmd measure_temp temp=44.5'C
```

Resulta muy interesante comprobar la temperatura en distintos momentos (en reposo, mientras se descarga e instala un paquete software, cuando se realiza una ejecución secuencial, al ejecutar un código en paralelo, etc.). En el Capítulo 9 se puede encontrar un ejemplo práctico sobre el uso del sensor de temperatura.

2.2.2 Interpretación de los indicadores LED

La RPi lleva incorporados dos LED, situados en una esquina de la placa, entre el puerto DSI y el USB-C, tal como puede observarse en la Figura 2.2. Concretamente, estos dos LED están etiqueados como "ACT" y "PWR".

A partir del análisis de cuándo y cómo se iluminan estos LED, puede obtenerse información útil sobre lo que está ocurriendo en la RPi desde el momento en que se inicia, lo cual resulta especialmente importante cuando se produce algún error durante el arranque. A continuación, se describen los principales patrones que reproducen los LED y qué indican.

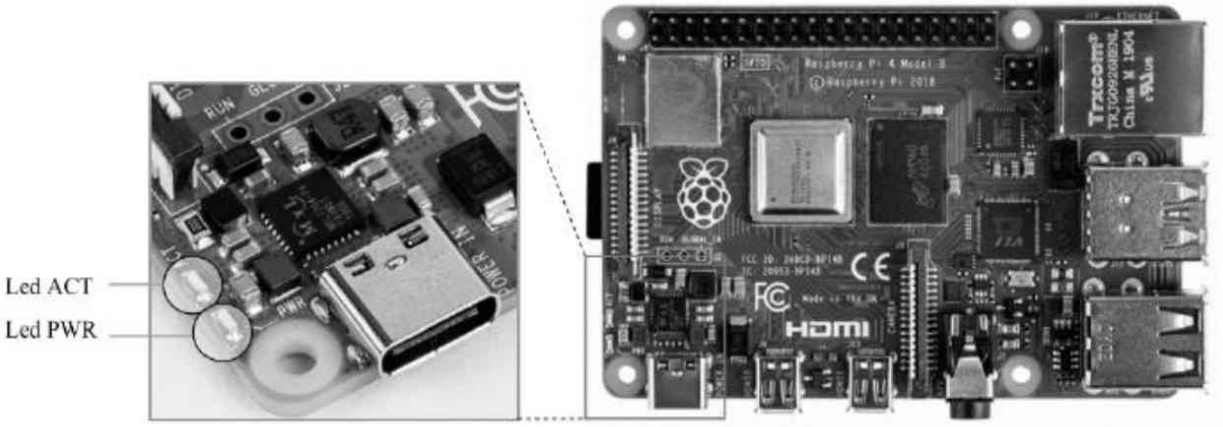


Figura 2.2 Detalle de los LED que incorpora la RPi 4B.

LED ACT

El LED ACT es el que emite una luz de color verde y está directamente relacionado con la capacidad de la RPi para leer del dispositivo conectado en el zócalo Micro SD.

Al arrancar la RPi, si la lectura es correcta y normal, podrá observarse un parpadeo irregular durante algunos segundos a partir del momento en el que se inicia. En caso contrario, se estará produciendo algún error y este LED reproducirá un patrón asociado al problema. La Tabla 2.2 muestra el significado de los distintos patrones que se pueden observar. Habitualmente, el patrón se repetirá dejando un intervalo de dos segundos entre cada emisión completa del mismo.

LED PWR

El comportamiento del indicador LED PWR de las RPi, el cual emite una luz roja, está asociado a la cantidad de energía que la placa está recibiendo.

Muchos de los problemas derivados del uso de RPi se originan por una mala alimentación del dispositivo, que si es insuficiente puede ocasionar problemas de arranque o corromper la tarjeta SD. Si en algún caso (mientras

está arrancada la RPi) este indicador LED no está encendido de manera constante, sino que parpadea o incluso se apaga, entonces es necesario revisar de inmediato la fuente de alimentación, ya que muy posiblemente exista algún problema.

Es importante tener en cuenta, respecto al suministro de energía que se haga a las RPi, que estas no contienen batería. Por tanto, emplear cargadores en lugar de fuentes de alimentación al uso está totalmente desaconsejado, ya que no proporcionan un voltaje estable, lo cual es necesario para el correcto funcionamiento del dispositivo. Además, un cable demasiado largo o demasiado delgado desde la fuente de alimentación al dispositivo puede deteriorar la señal, y esto se vería igualmente reflejado en el comportamiento del LED, que parpadearía o incluso podría llegar a apagarse por completo.

Parpadeos largos	**Parpadeos cortos**	**Significado**
0	3	Fallo genérico al arrancar
0	4	No se ha encontrado (o está corrupto) el fichero *start*.elf* al arrancar
0	7	No se ha encontrado la imagen del núcleo del SO
0	8	No se reconoce la SDRAM
0	9	La SDRAM disponible es insuficiente
0	10	El dispositivo se encuentra en estado *HALT* (parada)
2	1	La partición del sistema de ficheros no es de tipo FAT

2	2	No se ha podido leer desde la partición del sistema de ficheros
2	3	La partición extendida del sistema de ficheros no es de tipo FAT
2	4	Archivo de firma/hash erróneo (sin coincidencia)
3	1	Error en el SPI EEPROM2
3	2	El SPI EEPROM está protegido contra escritura
4	4	No se soporta el tipo de placa
4	5	Se ha producido un error fatal de *firmware*
4	6	ó 7 Fallo de potencia

Tabla 2.2 Significados asociados al indicador LED ACT según el número y tipo de parpadeos emitidos.

Las RPi presentan unos requisitos de alimentación que deben cumplirse estrictamente para garantizar su correcto funcionamiento. Particularmente, el modelo sugerido en este manual requiere 5V y 3A.

Alternativamente, el hecho de utilizar un software obsoleto o no soportado por la RPi también puede provocar que este indicador LED refleje problemas. En tal caso, se observarían también parpadeos en lugar de una luz roja constante.

2.2.3 Otros indicadores y ayudas ante problemas

Además de los indicadores LED ya descritos, la RPi puede proporcionar al usuario información ante fallos o deficiencias tanto durante el tiempo de ejecución (bajo determinadas circunstancias) como al arrancar incorrectamente.

Iconos *firmware* de aviso

La Tabla 2.3 refleja los tres iconos que el usuario puede observar, superpuestos generalmente en una de las esquinas, en la pantalla donde esté proyectando el dispositivo.

		
El voltaje que está siendo suministrado al dispositivo es inferior a 4,63 V.	La temperatura del dispositivo es demasiado alta (se encuentra entre 80 °C y 85 °C).	La temperatura del dispositivo es demasiado alta (es superior a 85 °C).

Tabla 2.3 Iconos mostrados por el dispositivo durante su ejecución para alertar de una situación inapropiada para el correcto funcionamiento.

Diagnóstico de arranque del sistema

Desde la versión 2020-04-16 del sistema de arranque de la Raspberry Pi 4 (en cualquiera de sus modelos), en el momento en el que se inicia el dispositivo se puede mostrar información relativa a dicho proceso en un monitor (si lo hay conectado). Para ver esta información, a la que se le denomina *diagnóstico de arranque*, debe retirarse la tarjeta SD (teniendo el dispositivo apagado) y, a continuación, iniciar el sistema. Además, si el dispositivo no

pudiera realizar correctamente el arranque (teniendo la SD insertada), también aparecería esta información al iniciar el dispositivo.

La información que proporciona el diagnóstico de arranque se detalla línea a línea en la Tabla 2.4.

Línea	Información
bootloader	Versión del sistema de arranque y fecha
board	Versión de la placa, número de serie y dirección MAC Ethernet
boot	Información asociada al arranque (modo, BOOT_ORDER [configuración EEPROM] y registro PM_RSTS)
SD CID	Identificador de la tarjeta SD (proporcionado por el fabricante)
part	Tipo de particiones primarias del registro de arranque maestro
fw	Nombre de archivo para *start.elf* y *fixup.dat* (si está presente)
net	Información relativa a la red: estado (up/down), dirección IP (ip), subred (sn) y puerta de enlace predeterminada (gw)
tftp	Dirección IP del servidor TFTP

Tabla 2.4 Información proporcionada en cada línea del diagnóstico de arranque.

Más información sobre estos elementos y documentación al respecto puede encontrarse en la web oficial de Raspberry Pi3.

2.3 Componentes del clúster

En este apartado se profundiza en las características más destacables de un clúster de computación. Concretamente, utilizando los componentes del clúster RPi de este libro se establecerán (siempre que sea posible) paralelismos entre un clúster de supercomputación en producción y el entorno experimental propuesto.

2.3.1 Nodo de cómputo

Como se ha descrito anteriormente en el Apartado 1.6, en un clúster de computación existen dos roles básicos asociados a los nodos: *front-end* y cómputo. Aunque estos roles pueden ser asumidos por computadores con distintas especificaciones técnicas, es habitual que los nodos sean prácticamente iguales, a excepción, por ejemplo, de los contextos en los que el clúster utiliza aceleradores, en cuyo caso estos podrían encontrarse solamente en los nodos de cómputo. Sin embargo, los aceleradores están fuera del alcance de este libro.

En nuestro clúster RPi contamos con una serie de dispositivos idénticos, de los cuales uno de ellos será el *front-end*, y el resto, nodos de cómputo. En grandes entornos de supercomputación, se pueden encontrar varios *front-end* (para disponer de redundancia, manteniendo así la disponibilidad del servicio) y del orden de miles de nodos de cómputo.

2.3.2 Memoria principal

Existe una amplia variedad de tipos de memoria principal en dispositivos electrónicos, según el propósito de los mismos. En la gran mayoría de computadores, como la RPi, la memoria principal permite ser leída y escrita, y es de tipo volátil. En otras palabras, si la memoria se desconecta de la red

eléctrica, los datos almacenados desaparecen. En esta memoria principal de los nodos, también conocida como memoria de acceso aleatorio (*Random Access Memory* [RAM], en inglés), es donde se cargan las instrucciones y datos que el procesador utilizará.

Hay que tener en cuenta que la memoria principal disponible nunca se corresponde con la memoria total existente en el nodo, dado que el sistema operativo utiliza y reserva parte de esta memoria para sus operaciones internas. Esto se puede comprobar al acceder al fichero /proc/meminfo, donde se encuentra el desglose de la utilización de la memoria principal.

Tal y como se ha descrito en la Tabla 2.1, la memoria principal de la RPi es RAM dinámica síncrona (SDRAM) de bajo consumo energético (LPDDR4 SDRAM) y de 2 GB de capacidad. En entornos de supercomputación, el concepto de memoria principal se mantiene tal y como se ha descrito, si bien se utilizan versiones más rápidas y de mayor capacidad, como por ejemplo la SDRAM DDR4.

2.3.3 Almacenamiento

El almacenamiento, o memoria secundaria, es un tipo de memoria no volátil que, aunque prescindible en un ordenador, resulta necesaria para un funcionamiento adecuado. Este tipo de memoria es mucho más económica, pero presenta un acceso (tanto de escritura como de lectura) más lento que el de la memoria principal. Por este motivo, es habitual que el espacio de almacenamiento sea mayor (en varios órdenes de magnitud) que el espacio ofrecido por la memoria principal. Además, muchos sistemas operativos establecen un espacio de intercambio (*swap*, en inglés) en la memoria secundaria, utilizado para almacenar datos cuando estos ya no caben en la memoria principal.

En el caso de la RPi, y en concreto en este libro, se ha optado por utilizar

una memoria secundaria de tipo SDHC conectada al Bus SD mediante el zócalo Micro SD que la RPi incorpora para este propósito. Particularmente, se ha utilizado una tarjeta Micro SDHC de Clase 10 (es decir, capaz de realizar escritura secuencial a una velocidad de hasta 10 Mbps) con 16 GB de capacidad.

Mientras que para la memoria principal el concepto explicado en el apartado anterior es fácilmente extrapolable a supercomputadores, para el almacenamiento es distinto. Existe una clara diferencia entre el almacenamiento en el clúster RPi y el que se daría en un supercomputador. Los grandes centros de supercomputación dedican nodos en exclusiva al sistema de ficheros distribuido. Estos sistemas permiten hacer las operaciones de lectura y escritura en paralelo, lo que posibilita multiplicar el ancho de banda con el que estas operaciones pueden llevarse a cabo. No solo eso, sino que además, para reducir el tráfico de red, estos sistemas cuentan con una red de comunicaciones exclusiva para tal efecto. Dado que añadir una segunda red dedicada al almacenamiento está fuera del alcance de este libro, para simular el funcionamiento de un clúster de supercomputación a este respecto, se aprovechará la red de interconexión con el fin de configurar un sistema de ficheros compartidos entre todos los nodos, lo cual será debidamente presentado posteriormente en el libro. A modo ilustrativo, soluciones como PanFS[4], BeeGFS[5], Lustre[6] o IBM Spectrum Scale (anterior GPFS)[7] ofrecen sistemas consistentes de alto rendimiento para la gestión distribuida de ficheros.

2.3.4 Red de comunicación

La red de comunicación del clúster es lo que va a determinar la capacidad de colaboración de los distintos nodos que lo componen a la hora de ejecutar un mismo trabajo. Concretamente, el rendimiento de la red vendrá dado por el ancho de banda, es decir, la cantidad de datos que se pueden transportar

por unidad de tiempo, también conocido de manera informal como *velocidad de red*. En entornos de supercomputación también se tiene muy en cuenta la latencia o, en otras palabras, el tiempo de respuesta entre nodos.

Para aumentar la productividad de la red de interconexión es habitual que se establezcan múltiples enlaces por nodo para multiplicar el ancho de banda total. Además, existen topologías de red que permiten comunicaciones más rápidas y fiables que otras. La Figura 2.3 ilustra algunas de las topologías que se pueden utilizar para la comunicación de dispositivos. Por ejemplo, la topología "doble anillo" utiliza dos enlaces para duplicar su ancho de banda, la topología "malla" ofrece redundancia, ya que no solo existe un camino entre dos nodos, y la "totalmente conexa" permite la comunicación directa con cualquier nodo de la red sin tener que pasar por ningún intermediario.

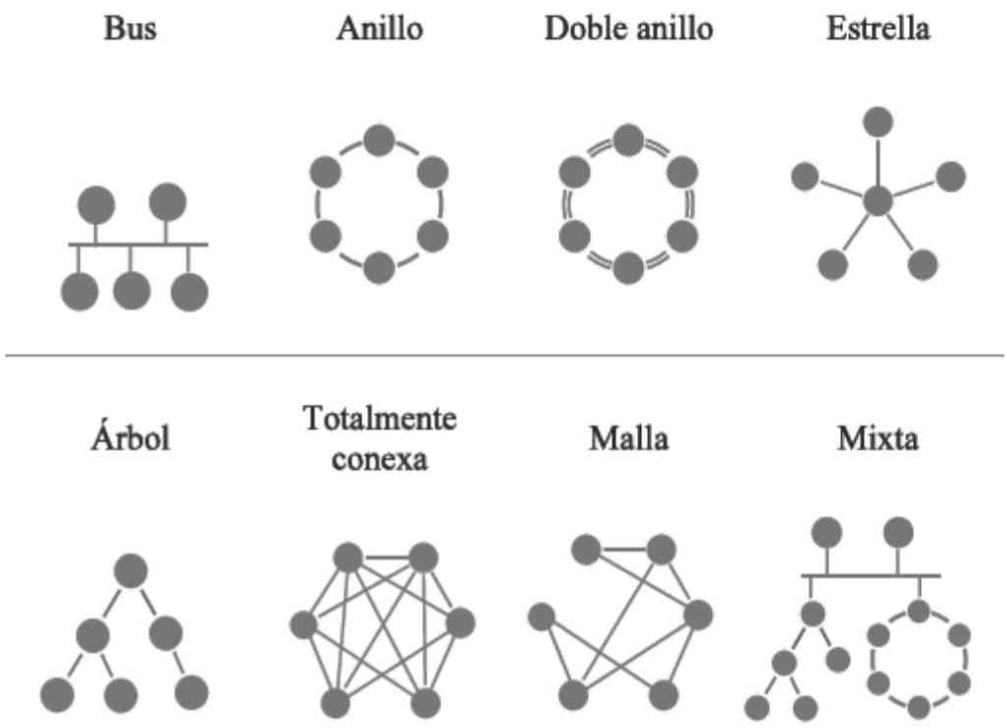


Figura 2.3 Ejemplos de topologías en redes de comunicaciones.

Para el caso concreto del clúster RPi se ha decidido utilizar una topología "estrella", en la cual todos los nodos están conectados mediante un

conmutador. Este tipo de topología es muy sencillo de implementar y, a su vez, requiere pocos recursos. Si algún nodo fallase, el resto podría seguir con sus comunicaciones; sin embargo, en caso de un fallo en el conmutador, la red quedaría inservible.

A pesar de que la RPi viene dotada de una interfaz de red Gigabit Ethernet que ofrece un ancho de banda teórico de 1 Gb/s, esta tasa dista mucho de los 600 Gb/s con los que cuentan algunos de los supercomputadores del TOP500. Además, estas redes de altas prestaciones no se basan en conductores de cobre, sino que utilizan fibra óptica, lo que también contribuye a ofrecer latencias mucho menores, como, por ejemplo, de 0,5 microsegundos.

2.3.5 Periféricos

Al margen de todos los componentes descritos, serán necesarios algunos periféricos adicionales para interactuar con el sistema:

- Monitor. La RPi cuenta con puertos micro HDMI, por lo que el monitor elegido debe poder conectarse mediante un cable HDMI, o bien será necesario utilizar un adaptador.
- Teclado y ratón. Dado que la RPi dispone de puertos USB para estos periféricos, la única observación destacable respecto a cuál elegir es precisamente que se conecten al dispositivo mediante este tipo de entrada.

2.3.6 Coste económico

Los supercomputadores presentan siempre un coste muy elevado en lo que respecta a componentes, el cual fácilmente sobrepasa varios cientos de millones de euros. No obstante, la finalidad de este libro consiste en presentar la supercomputación de un modo asequible, tanto en términos conceptuales y de desarrollo como económicos. Los dispositivos RPi tienen un

precio relativamente bajo (no llega a la centena de euros por unidad) y el conjunto propuesto, descrito en la Tabla 2.5 (dejando a un lado los periféricos), queda muy por debajo de los cientos de millones de euros anteriormente mencionados; de hecho, con alrededor de 300 €–400 €, debe ser posible costear en su totalidad el clúster formado por cuatro nodos RPi descrito en este manual.

Componente	Cantidad
Raspberry Pi	4
Fuente de alimentación	4
Switch	1
Cable Ethernet RJ45	4
Micro SD	4

Tabla 2.5 Componentes necesarios para formar el clúster RPi, obviando periféricos.

En función del presupuesto disponible, el clúster propuesto en este libro podría redimensionarse, es decir, podrían añadirse o quitarse nodos.

2.4 Para profundizar...

1. ¿En qué casos de uso doméstico se podría utilizar Raspberry Pi?

   Piense, por ejemplo, en la creación de un hogar domótico. ¿Para qué se podría usar Raspberry Pi?

2. ¿En qué otros entornos (al margen del doméstico) y para qué podría emplearse Raspberry Pi?

   Por ejemplo, podría aplicarse a la medición de parámetros ambientales o a la automatización de tareas como el riego, entre muchos otros.

3. ¿Con qué otros fines podría aprovecharse tanto la potencia del ordenador Raspberry Pi como su reducido tamaño?

   Por ejemplo, en la emulación de videoconsolas antiguas, cuyo rendimiento estaba muy por debajo del proporcionado por Raspberry Pi, o para la configuración de un sistema *ambilight*, el cual requiere de un dispositivo pequeño pero potente para su gestión. En el siguiente enlace puede encontrarse un ejemplo ilustrativo: https://www.xodustech.com/projects/raspberry-pi-gameboy-pocket

4. Utilizando varios dispositivos RPi se podría implementar un bingo en el que cada nodo correspondiera a una celda del cartón de bingo. De forma distribuida, es decir, cada nodo independientemente, podría comprobar si el número extraído corresponde al de cada nodo. ¿Cuántos

elementos hardware serían necesarios en este caso si se emplea el *front-end* para generar el número en juego y el cartón se compone de cuatro filas y cuatro columnas?

En el siguiente enlace puede observarse un ejemplo de su implementación, en el que el *front-end* es el encargado de generar el número en juego:
https://www.youtube.com/watch?v=DN_cqiSg8jUDt=381s.

5. En el clúster propuesto el almacenamiento está limitado a la capacidad de las tarjetas SD. ¿Cómo podría ampliarse el almacenamiento mediante hardware extra? ¿Qué ocurriría si se dispusiese de discos duros externos?

Esto podría hacerse conectando discos duros externos mediante los puertos USB de la RPi. En ocasiones, puede ser necesario editar el fichero */boot/config.txt* para incluir la opción max_usb_current=1. Puede encontrarse más información respecto a este tema en los foros de la web oficial de Raspberry Pi.

6. En el clúster propuesto se utiliza una topología "estrella" para permitir la comunicación entre los distintos nodos. ¿Podrían conectarse físicamente de manera alternativa los nodos para seguir las distintas topologías presentadas? ¿Serían necesarios recursos hardware extra? En caso afirmativo, ¿cuáles?

Podrían implementarse algunas de las topologías alternativas mostradas en la Figura 2.3. Por ejemplo, para una topología de "anillo" habría que conectar cada RPi directamente a otras dos. Como la RPi

cuenta con dos interfaces de red (una cableada y la otra inalámbrica), se podría conectar cada interfaz a un nodo. No obstante, también se podrían utilizar adaptadores USB-RJ45 para incrementar el número de interfaces de red.

2.5 Material recomendado

Los dispositivos Raspberry Pi son más sensibles que otros con los que habitualmente se trabaja y que componen ordenadores personales, tabletas gráficas, teléfonos inteligentes, etc. Por un lado, excepto si se enfundan, la constante exposición de sus componentes (pines, procesador, puertos, etc.) los vuelve más vulnerables a cambios de temperatura, humedad o golpes, entre otros, por lo que, en ocasiones, dejan de funcionar correctamente. Además, estos microcomputadores también son altamente sensibles a suministros de temperatura irregulares o inapropiados (tanto por defecto como por exceso), así como a conflictos debido a tarjetas de memoria o sistemas operativos incompatibles instalados en ellas. Para facilitar la corrección de las circunstancias que de un modo u otro sufren estos dispositivos y conllevan un mal funcionamiento, es muy recomendable conocer el significado de la información de los LED que llevan incorporados. En este capítulo se han presentado las principales señales luminosas y qué significan, pero esta información puede encontrarse más detallada en la web oficial de Raspberry Pi. Particularmente, la entrada del foro de dicha web [1] proporciona mucha información al respecto.

Otro de los temas a los que se le ha dedicado especial atención a lo largo del capitulo es la memoria principal y el almacenamiento. En el artículo de investigación [2] se pueden encontrar descripciones más detalladas sobre todo lo expuesto y documentación adicional.

[1] Raspberrypi Forum. Detalle sobre LED en la Raspberry Pi. https://www.raspberrypi.org/forums/viewtopic.php?t=58151 , 2013. Accedido por última vez: nov. 2020.

[2] Darko Zivanovic, Milan Pavlovic, Milan Radulovic, Hyunsung Shin,

Jongpil Son, Sally A. Mckee, Paul M. Carpenter, Petar Radojković, and Eduard Ayguadé. Main memory in hpc: Do we need more or could we live with less? *ACM Trans. Archit. Code Optim.* , 14(1), March 2017.

2.6 Resumen

En este capítulo se ha presentado la historia del dispositivo Raspberry Pi, poniendo especial atención a la versión 4 Model B. De este se han descrito la tecnología de sus componentes y la arquitectura de sus comunicaciones. Además, se dispone de una guía de interpretación de los indicadores LED con la que comprender qué está ocurriendo en el dispositivo. En este capítulo se han presentado los componentes más destacables de un clúster en producción y se han comparado con el que se va a desarrollar en esta obra, compuesto por dispositivos RPi.

2.6.1 ¿Qué viene a continuación?

A continuación, en la Parte III, se procede a detallar los pasos necesarios para construir, configurar y evaluar el clúster RPi. Así que, ¡manos a la obra!

1 https://static.raspberrypi.org/files/product-brie-200521+Raspberry+Pi+4+Product+Brief.pdff.pdf

3 https://www.raspberrypi.org/documentation/har-dware/pberrypi/boot_diagnostics.mdcs.md

4 https://www.panasas.com

5 https://www.beegfs.io

6 https://www.lustre.org

7 https://www.ibm.com/products/spectrum-scale

Parte III

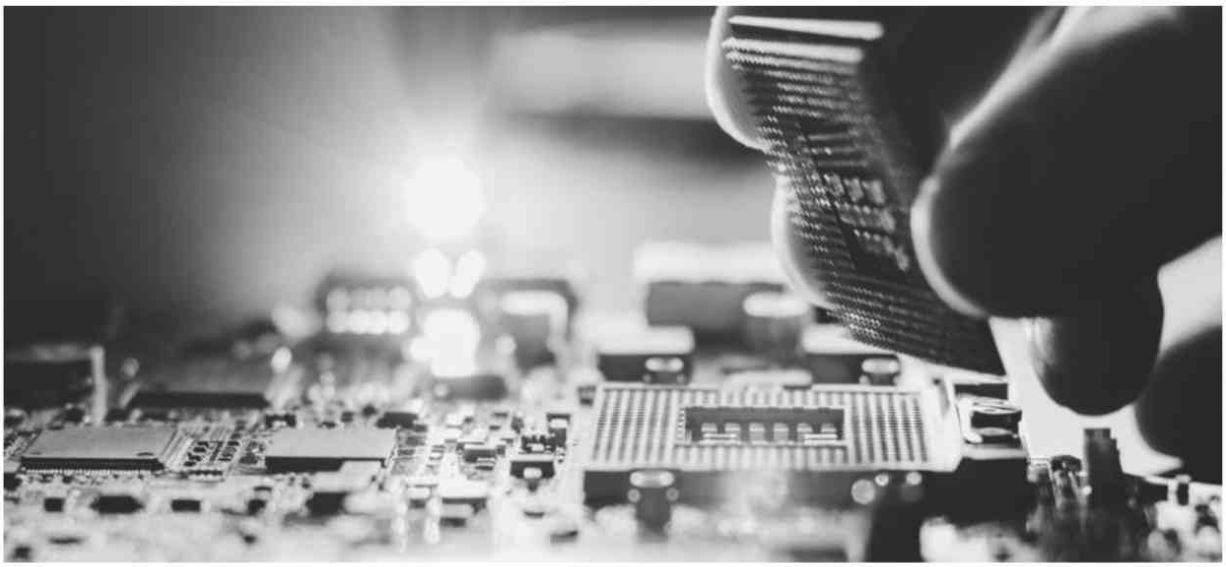
¡Manos a la obra!

Visión general

Esta parte del libro se centra en la adquisición de nuevos conocimientos sobre la HPC a través del montaje y configuración del supercomputador basado en Raspberry Pi. Aunque no se disponga del material necesario para la construcción del clúster, igualmente recomendamos una lectura minuciosa de cada apartado, en los que se explican no solamente nuevos conceptos de HPC, sino que se detallan los pasos a llevar a cabo para una correcta configuración/insta-ción del clúster y las herramientas y aplicaciones presentadas.

La HPC es un área en la que confluyen conocimientos de diversas áreas informáticas, tales como las redes, la arquitectura de computadores o los entornos de usuario, entre otras. Por esa razón, en esta parte del libro, a medida que se lleva a cabo el montaje y configuración del clúster, se van introduciendo conceptos de las distintas áreas, que se explican conforme son necesarios.

El primer capítulo de esta parte (Capítulo 3) presenta el sistema operativo y distintas utilidades y comandos que serán necesarios para poder llevar a cabo los pasos de los siguientes capítulos, desde la instalación del propio sistema operativo hasta la escritura de *scripts* básicos. A continuación, se detalla gráficamente y paso a paso cómo ensamblar todos los componentes que formarán parte del supercomputador RPi. El Capítulo 4 presenta los pasos a seguir para que el clúster montado en el capítulo anterior funcione como un supercomputador; para ello se realiza la configuración de la red interna y del sistema de ficheros compartido. Los Capítulos 5 y 6 muestran algunas herramientas más avanzadas dentro de un supercomputador; particularmente, distintos modelos de programación paralela y el gestor de trabajos. El Capítulo 7 se centra en distintos *benchmarks*, herramientas y aplicaciones en uso en sistemas reales. Finalmente, el Capítulo 8 va un paso más allá, y se centra en aspectos más avanzados: la virtualización y el uso de contenedores.



3. Montaje del clúster

En este capítulo se detalla gráficamente, y paso a paso, el montaje y puesta en marcha del clúster. El capítulo recoge una breve descripción sobre qué es un sistema operativo a modo de introducción y continúa con los pasos detallados para ensamblar los componentes hardware del clúster, junto con la instalación del sistema operativo correspondiente.

3.1 El sistema operativo

El sistema operativo (SO) es el programa informático principal que gestiona los recursos hardware y software disponibles en la máquina anfitriona. Para este propósito, el SO se ejecuta en modo administrador. Este rol de administración, a diferencia de los usuarios estándar del sistema, atribuye una serie de permisos exclusivos para realizar ciertas operaciones cuya mala ejecución puede provocar errores críticos que ocasionen la caída del sistema. A grandes rasgos, un SO básico tiene que proveer una serie de funciones que permitan llevar a cabo las siguientes operaciones:

- Crear, destruir y sincronizar procesos.
- Gestionar el uso de la memoria principal.
- Gestionar el uso del almacenamiento secundario.
- Manejar las interrupciones de los dispositivos de entrada y salida conectados al sistema.
- Gestionar los datos almacenados mediante un sistema de archivos.
- Controlar los permisos de ejecución de las tareas de administración.
- Gestionar las comunicaciones de red.

La Figura 3.1 ilustra la jerarquía de interacción Usuario-Aplicación-SO-Har-dware, donde se puede apreciar que el SO se sitúa justo por encima del hardware para su gestión. El uso de los elementos hardware por parte de las aplicaciones ejecutadas en el sistema operativo es posible gracias a llamadas al sistema o mediante controladores (*driver*, en inglés). Las primeras son funciones nativas del SO (por ejemplo, las que se encargan de acceder a la memoria principal), mientras que los segundos se incorporan a petición del usuario para un dispositivo específico (por ejemplo, una impresora).

Figura 3.1 Componentes del flujo de trabajo habitual entre usuario y hardware.

Actualmente, los SO aprovechan las arquitecturas multitarea (multihilo o multiprocesador) para ejecutar simultáneamente varios procesos concurrentemente, tal como se vio en el Apartado 1.9. Esto permite lanzar procesos en segundo plano que se ejecutan de forma transparente al usuario. En términos informáticos este tipo de procesos se conocen como demonios (*daemon*, en inglés). Los demonios ejecutan servicios del sistema muy variados que no requieren la interacción del usuario; por ejemplo, el demonio cron se encarga de ejecutar tareas planificadas, mientras que el sshd se encarga de gestionar las conexiones SSH (más información del protocolo SSH en el Apartado 4.1.1).

3.2 GNU/Linux

Se llama GNU/Linux al SO que combina el núcleo (*kernel*, en inglés) de Linux con una serie de aplicaciones de código abierto (*open source*, en inglés). Esta familia de SO se utiliza en la inmensa mayoría de sistemas HPC y, en el momento de escribir este libro, todos los supercomputadores del TOP500 se basan en alguna distribución GNU/Linux. Del mismo modo, en el clúster RPi se utilizará una distribución GNU/Linux específica para nuestra RPi (ver Apartado 3.4.3).

En el sistema GNU/Linux existe el rol de superusuario, que es el asignado a las cuentas con permisos de administración. No obstante, lo habitual es disponer de una cuenta de usuario regular y solo ejecutar algunos comandos en modo administrador. Esta característica de los SO GNU/Linux añade una capa extra de seguridad, ya que dificulta realizar cambios que puedan dañar al sistema por error o desconocimiento. Sin embargo, hay ocasiones en las que es de gran utilidad ejecutar un comando en modo administrador; por esta razón, Linux ofrece la utilidad sudo (*SUperuser DO*, del inglés) para tal fin. Tras ejecutar este comando en la terminal, bastará con introducir la contraseña del usuario regular para obtener permiso de superusuario durante 15 minutos, siempre y cuando el usuario tenga permisos para ello. Esto resultará imprescindible para realizar ciertos pasos de la configuración del clúster RPi.

3.2.1 Sistema de ficheros

Un fichero o archivo es un conjunto de datos que contiene información (un documento, una canción, un ejecutable, etc.) almacenada en la memoria secundaria. En los sistemas GNU/Linux, cada fichero tiene asignado un nombre y está almacenado en un directorio o carpeta al que, a su vez,

también se le atribuye un nombre y puede contener tanto ficheros como otros subdirectorios. De este modo, se forma la estructura jerárquica que define el sistema de ficheros. Al directorio que es el origen de dicha estructura y, por tanto, del sistema de ficheros, se le denomina directorio raíz (/). Por su parte, el conjunto de directorios y subdirectorios que hay que recorrer desde la raíz o desde algún otro directorio origen en la jerarquía para llegar a un determinado fichero recibe el nombre de ruta. Si el directorio origen es el directorio raíz, entonces a la ruta se le llama ruta absoluta; en caso contrario, la ruta se denomina ruta relativa. Por ejemplo, el fichero llamado *fichero1.txt* (Figura 3.2) se encuentra en la carpeta *Documentos* en el directorio personal del usuario pi (normalmente, este se encuentra en */home*), entonces la ruta absoluta del fichero es */home/pi/Documentos/fichero1.txt*. La ruta relativa depende del directorio actual (el cual puede obtenerse con el comando pwd), así que si nos encontramos en el directorio */home/pi/I-magenes/Fotos*, la ruta relativa es *../../Documentos/fichero1.txt*. Nótese que . y .. indican, respectivamente, el directorio actual y el directorio anterior en la jerarquía de ficheros.

Existen distintos formatos de sistema de ficheros. Normalmente, el tipo de sistema de ficheros depende del sistema operativo. Por ejemplo, actualmente las distribuciones de Linux utilizan dos sistemas de ficheros de manera simultánea: *ext4* y *swap*. El *swap* es utilizado por el sistema operativo para almacenar archivos temporales que deberían almacenarse en la memoria principal, pero que no caben en un determinado momento. El sistema de ficheros *ext4* es el que soporta toda la jerarquía del sistema de ficheros y, concretamente, esta versión del sistema de ficheros puede almacenar hasta un *exabyte* de datos, es decir, 1018 bytes. Existen otros tipos de ficheros para el intercambio de datos, como, por ejemplo, *Network File System* (NFS), que sirve para intercambiar datos entre distintos equipos Linux, o *Server Message*

Block (SMB), que se utiliza para el intercambio de datos entre equipos Windows y Linux. Cabe destacar que un sistema de ficheros puede tener disponible, a su vez, dentro de su jerarquía, otro sistema de ficheros. A la acción de integrar un sistema de ficheros en un directorio en particular se le denomina montar.

El sistema de ficheros de Linux está formado por distintos directorios y cada uno de ellos tiene un propósito en particular. Cada usuario puede disponer de un directorio para uso personal que se localiza como subdirectorio del directorio *home* bajo el nombre de */home/nombre-deusuario* (por este motivo, a dicho subdirectorio se le suele denominar *home* del usuario). Es posible hacer referencia a este directorio usando el carácter ~ o mediante la variable de entorno HOME (por ejemplo, el comando echo $HOME devolvería /home/nombre-de-usuario). La Figura 3.2 muestra los directorios del sistema de ficheros de Linux, que será el que se encontrará en el sistema operativo que se instalará en las RPi. Cada directorio que pertenece al directorio raíz tiene un propósito en particular[1]:

- **bin** : contiene los archivos binarios que pueden ser ejecutados por los usuarios para garantizar las funciones básicas a nivel de usuario. Algunos de estos binarios son los que permiten realizar los comandos básicos de la consola.
- **boot** : incluye todos los archivos necesarios para iniciar el sistema.
- **dev** : no contiene ficheros estándar como el resto de directorios, sino los asociados a los dispositivos hardware y componentes que son usados por el SO. Ejemplos de estos archivos son los correspondientes a dispositivos de almacenamiento (discos duros, USB, etc.), dispositivos de audio o el generador de números aleatorios (*/dev/random*).
- **etc** : almacena ficheros de configuración de los servicios, la base de datos con los usuarios y grupos de usuarios o la lista de sistemas de ficheros

que deben montarse durante el arranque, entre otros.

■**home** : contiene los directorios personales de los usuarios del sistema.

■**lib** : incluye las bibliotecas esenciales para los binarios de los directorios (*/bin* , */sbin* y */usr/bin*), junto con módulos del núcleo de Linux.

■**lost+found** : almacena ficheros o directorios recuperados tras una revisión o reparación del sistema de ficheros.

■**media** : punto de montaje para dispositivos extraíbles que se montan en el sistema (por ejemplo, un USB).

■**mnt** : punto de montaje temporal de sistemas de ficheros.

■**opt** : punto de instalación de aplicaciones de terceros.

■**sbin** : contiene los archivos binarios que solamente pueden ser ejecutados por el sistema operativo o por usuarios con privilegios de administrador.

■**srv** : suele incluir datos que son usados por el sistema para proporcionar un servicio. Por ejemplo, si se está proporcionando un servidor web, este es el directorio recomendado donde almacenar información.

■**tmp** : contiene archivos temporales usados por las aplicaciones. Normalmente, los ficheros de este directorio son eliminados cuando el nodo se apaga o se reinicia.

■**usr** : contiene las aplicaciones y los archivos relativos al software propio de los usuarios o utilidades del sistema que son accesibles por los usuarios.

■**var** : para ficheros con datos variables, como bases de datos o ficheros de registro (también llamados *logs*).

■**root** : directorio personal del usuario root.

■**proc** : al igual que */dev* , no contiene ficheros estándar, sino archivos especiales que contienen información sobre el sistema (por ejemplo, */proc/cpuinfo* contiene información del procesador) y sobre los procesos

del sistema.

3.2.2 Consola

La consola o terminal es actualmente un emulador software que permite la interacción directa con el sistema operativo a través de una *shell*. Aunque hoy en día se usan como términos sinónimos, en su origen, una consola y una terminal eran cosas distintas. Para entender las diferencias, es necesario remontarse a la época de los *mainframes*, en la que un ordenador centralizaba las operaciones que realizaban todos los usuarios que se conectaban al mismo, por ejemplo, en grandes empresas o campus universitarios. Para poder acceder al *mainframe* o unidad central, los usuarios utilizaban una terminal que consistía en un dispositivo de entrada y salida de texto, es decir, de una pantalla y un teclado.

Por su parte, la consola permitía monitorizar el estado del sistema y controlar el *mainframe*. En realidad, *consola* hacía referencia tanto al puerto al que se conectaba el terminal para poder interactuar directamente con el ordenador central como a la conexión establecida entre ambos. Es decir, la terminal se conectaba físicamente al *mainframe* a través del puerto de consola para poder controlarlo.

Con la aparición de los ordenadores personales, los *mainframes* cayeron en desuso, aunque los términos consola y terminal se mantuvieron vigentes. Sin embargo, acabaron por convertirse en sinónimos, ya que en los ordenadores personales se dispone de emuladores de terminal que permiten interactuar directamente con el sistema operativo como lo haría una consola. En cualquier caso, independientemente del término utilizado para hacer referencia a esta pieza de software, se trata de un entorno que permite la ejecución de una *shell*.

La *shell* es una capa que envuelve al núcleo del sistema operativo y

proporciona una interfaz para interactuar con él a través de una línea de comandos, de modo que cuando recibe un comando introducido por el usuario en la línea de comandos, le pide al sistema operativo que lo ejecute. Existen distintas *shells*, como Ash, Bash, Csh o Zsh, entre otras; todas ellas son capaces de interpretar series de comandos, ya sea escritos directamente en la línea de comandos o bien contenidos en un *script*. Un *script* es un fichero que contiene los comandos a ejecutar por la *shell*. Se ejecutan de la misma forma que si fuesen introducidos uno tras otro en la línea de comandos, pero, además, el *script* facilita su reutilización.

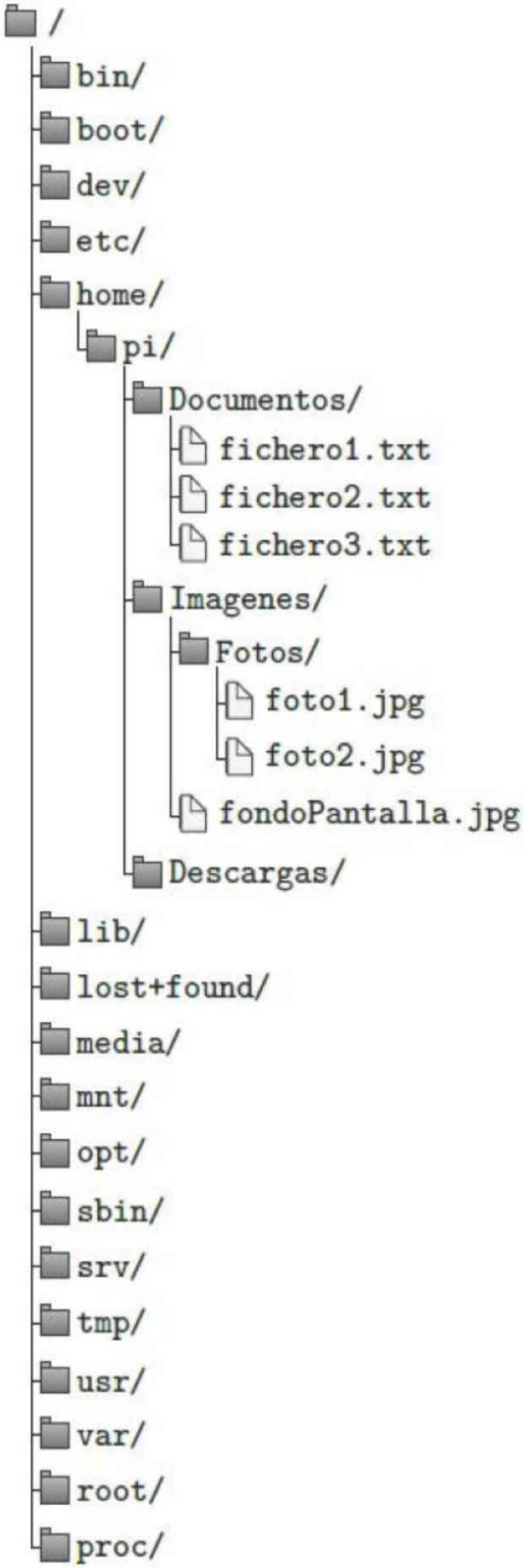

/
bin/
boot/
dev/
etc/
home/
pi/
Documentos/
fichero1.txt
fichero2.txt
fichero3.txt
Imagenes/
Fotos/
foto1.jpg
foto2.jpg
fondoPantalla.jpg
Descargas/
lib/
lost+found/
media/
mnt/
opt/
sbin/
srv/
tmp/
usr/
var/
root/
proc/

Figura 3.2 Ejemplo de la jerarquía que define el sistema de ficheros GNU/Linux.

Dado que el estándar *de facto* en cuanto a la *shell* en entornos GNU/Linux es Bash, a continuación se muestra un ejemplo sencillo de la ejecución desde la terminal o consola de un *script* que ilustra cómo listar el contenido del directorio actual, seguido del contenido de dicho *script*, donde la primera línea del fichero contiene el entorno (#!/bin/bash) que se utilizará para ejecutar los comandos que contenga.

Fichero: contenido_directorio.sh

```
#!/bin / bash
echo Contenido del directorio actual
ls
```

```
pi@raspberry:~ $ ./contenido_directorio.sh
Contenido del directorio actual
Descargas Documentos Escritorio Imágenes Música Vídeos
```

3.3 El sistema operativo para Raspberry Pi

Raspberry Pi OS (también conocido por su anterior nombre Raspbian) es el sistema operativo desarrollado por The Raspberry Pi Foundation, la organización responsable del diseño de los dispositivos Raspberry Pi, la cual también proporciona software que puede ser utilizado en estos.

Típicamente, cada año se lanzan dos versiones de Raspberry Pi OS, una alrededor de febrero-marzo y otra en agosto-septiembre. Desde la web oficial[2] pueden descargarse las distintas versiones del sistema operativo. Concretamente, se ofrecen tres opciones:

- *Raspberry Pi OS (32-bit) with desktop and recommended software* .
- *Raspberry Pi OS (32-bit) with desktop* .
- *Raspberry Pi OS (32-bit) Lite* .

Las dos primeras contienen una interfaz gráfica de usuario (*Graphical User Interface* [GUI], en inglés), mientras que la tercera, no (se interactúa siempre mediante la consola), por lo que se recomienda elegir una de las dos primeras opciones. De entre las dos primeras, basta con elegir la segunda, puesto que la primera incluye software preinstalado para fines educativos o específicos de programación fuera del alcance de esta obra. La mayoría de aplicaciones que se presentan y utilizan a lo largo de este manual se instalan progresivamente sin necesidad de tener ese software disponible. Si se opta por la segunda opción, el proceso de instalación es ligeramente más rápido y el coste en memoria se reduce a menos de la mitad (unos 2,9 GB frente a unos 1,2 GB).

Cabe destacar que la versión utilizada en el desarrollo de este manual es la de enero de 2021, por lo que puede ser necesaria alguna actualización leve de ciertas indicaciones si se utilizan versiones anteriores o posteriores. Por ejemplo, puede ser que el editor de textos que por defecto se incluye en la distribución concreta de Raspberry Pi OS sea uno diferente al que se propone

(por ejemplo, Mousepad).

En este libro se plantea el uso de versiones de 32 bits, aunque también existen versiones de 64 bits disponibles en estado *beta*[3] que podrían probarse. No obstante, si se utilizan versiones de 64 bits, puede ser necesario modificar parte de las instrucciones proporcionadas en este manual para asegurar el correcto funcionamiento.

Crear imagen en la tarjeta de memoria SD

El proceso de creación de una tarjeta SD ejecutable con el sistema operativo descargado es sencillo y se compone de los siguientes pasos:

1. Seleccionar la imagen a instalar (versión descargada del sistema operativo).
2. Indicar el dispositivo donde quiere instalarse (tarjeta SD).
3. Completar la instalación (también denominada proceso de *flash*). Este último paso puede tardar unos minutos y es muy importante no extraer la tarjeta SD en ningún momento de la operación.

El proceso de creación de la imagen en la tarjeta SD es el único en todo el manual que requiere disponer de un ordenador auxiliar desde donde poder descargar la imagen del sistema operativo y crear la tarjeta SD ejecutable. El resto de pasos descritos a lo largo de este libro se realizan directamente en los dispositivos Raspberry Pi que corresponda.

Para completar la serie de pasos descritos, existe el software Raspberry Pi Imager[4]. Alternativamente pueden utilizarse herramientas como Etcher[5], software de código abierto para grabar imágenes ejecutables. La Figura 3.3 muestra el proceso de instalación de ambos softwares en Windows (se hace de forma similar en otros sistemas operativos). En la Figura 3.4 se muestra el aspecto de ambas aplicaciones una vez instaladas.

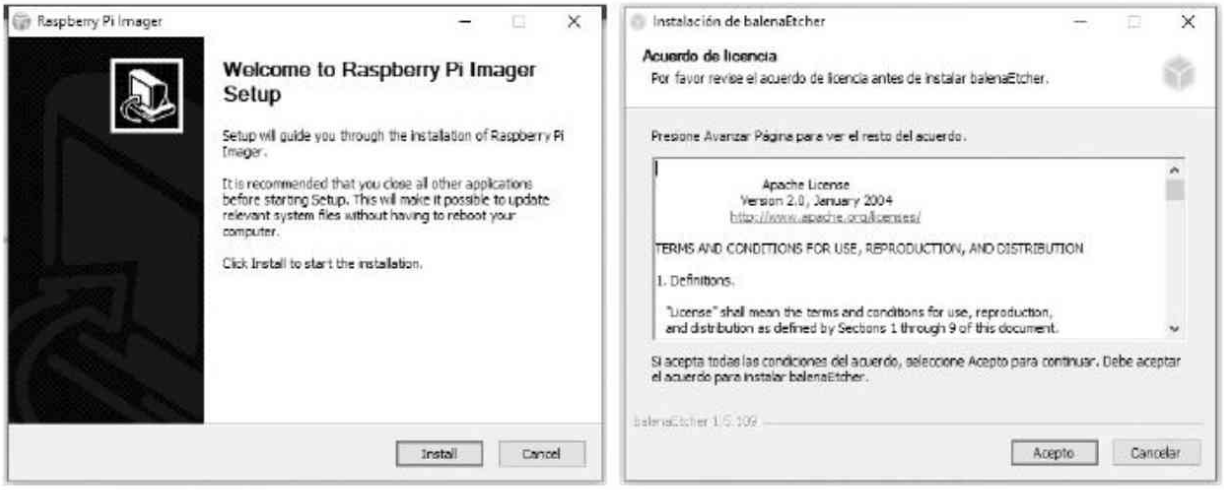


Figura 3.3 Ejemplo de instalación de los softwares Raspberry Pi Imager (izquierda) y Etcher (derecha).

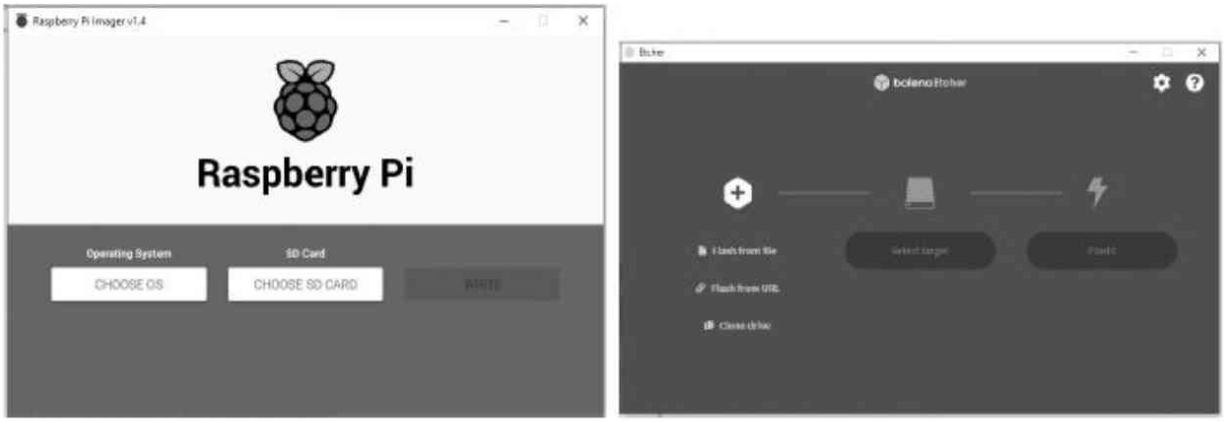


Figura 3.4 Aspecto de las aplicaciones Raspberry Pi Imager (izquierda) y Etcher (derecha) una vez instaladas e iniciadas.

Si se usa Raspberry Pi Imager, no será necesario descargar la imagen del sistema operativo, ya que este software hace eso de manera transparente al usuario, por lo que directamente se procederá a seguir los pasos indicados al inicio de este apartado. En cambio, si se utiliza cualquier software alternativo pensado para grabar cualquier tipo de imagen ejecutable, será necesario descargar la imagen de la versión escogida del sistema operativo desde la web oficial de The Raspberry Pi Foundation, la cual estará comprimida en un

fichero ZIP. Una vez descargado el fichero, se procederá a seguir los pasos descritos para generar la imagen ejecutable del sistema operativo en las tarjetas SD que se utilizarán en cada uno de los nodos RPi.

Las Figuras 3.5-3.8 ilustran el proceso completo de creación de la tarjeta con la imagen del sistema operativo escogido, tanto usando Raspberry Pi Imager (izquierda) como Etcher (derecha). Tras este proceso, al escribir una imagen ejecutable en la tarjeta e introducirla en la RPi, se logra que al encender la RPi, esta inicie la ejecución del SO instalado.

Cabe notar que, en la segunda fila de la Figura 3.5, en la figura de la derecha, se muestra qué versión del sistema operativo se recomienda descargar de entre las disponibles en la web oficial, cuyo fichero comprimido ZIP será el que se buscará en el PC una vez descargado a través del explorador de ficheros que abre Etcher al seleccionar la opción *Flash from file*. La tercera fila muestra qué debe verse en cada aplicación tras completar el Paso 1, si bien es cierto que en ambas aplicaciones podría darse el caso de que, adicionalmente, también la tarjeta se hubiera seleccionado automáticamente al insertarse en el PC.

Respecto a la Figura 3.6, cabe la posibilidad de que este paso no sea necesario, dado que, generalmente, la tarjeta sobre la que generar la imagen se selecciona automáticamente al ser insertada una tarjeta en el PC. Por su parte, en la Figura 3.7 se puede observar que, en ambos casos, tras completarse el proceso de *flash* (imágenes en la segunda fila), se lleva a cabo una verificación (imágenes en la tercera fila).

Por último, en caso de que no se observe lo que se muestra en la Figura 3.8 al completarse el Paso 3, se recomienda iniciar el proceso de nuevo, partiendo del Paso 1.

3.4 Montaje hardware paso a paso

La Figura 3.9 muestra un esquema que ilustra completamente cómo debe quedar el montaje de la parte hardware del clúster. Tal como se detallará en capítulos posteriores, los nodos nodo1, nodo2 y nodo3 serán los llamados nodos de cómputo, y en ellos solamente será necesario conectar los periféricos (teclado, ratón y monitor) al inicio del proceso de configuración, dado que después se accederá a ellos a través del nodo0, al que se denominará *front-end*. Por este motivo, los periféricos y conexiones que solamente deben utilizarse durante un período de tiempo y no siempre que se utilice el clúster se muestran en tonalidades grises en la figura referenciada; sin embargo, los representados en color negro deberán estar operativos siempre que se utilice el clúster.

A continuación se detalla cada uno de los pasos a seguir para realizar el montaje completo, particularizándolo para un dispositivo RPi.

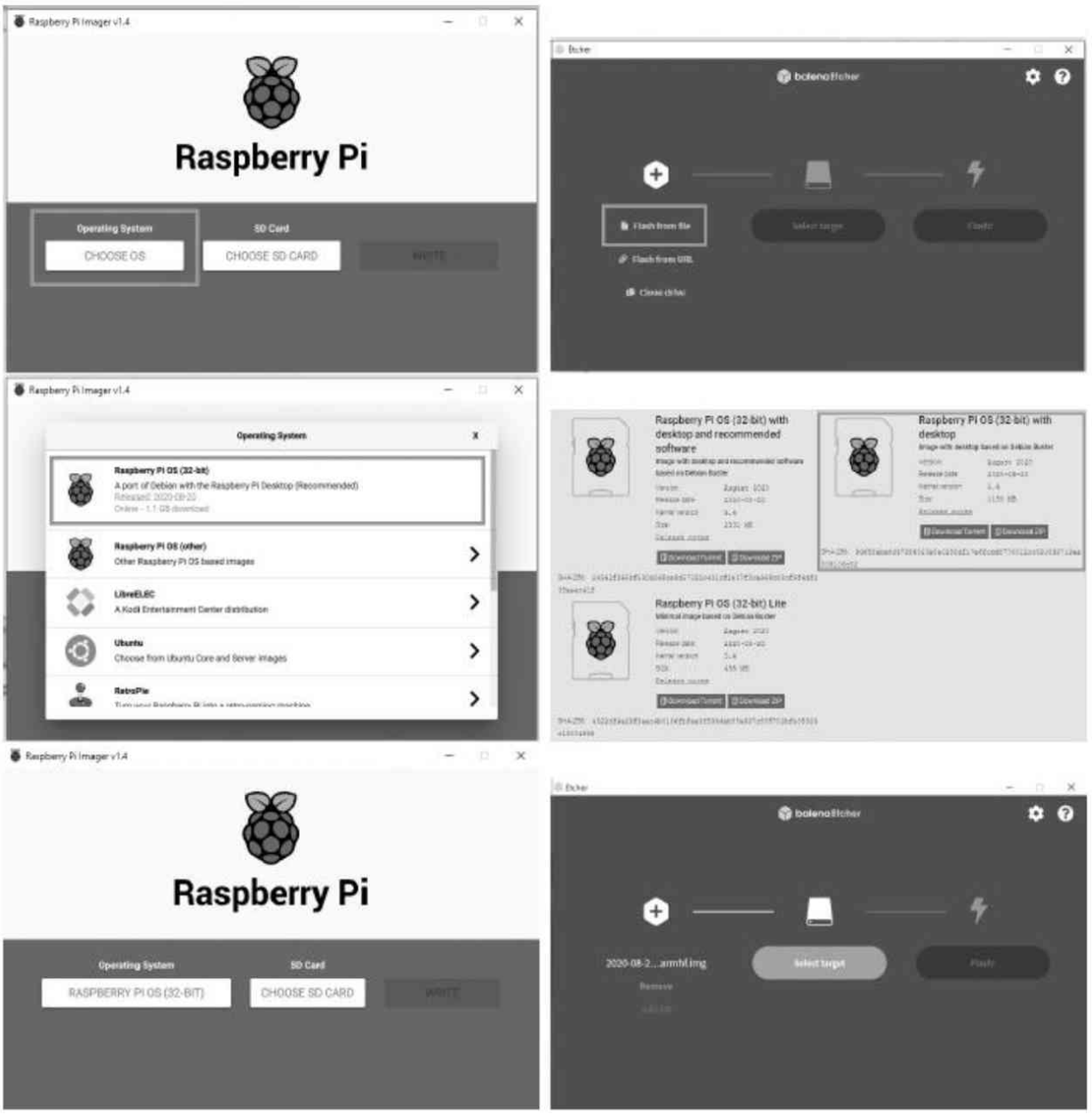


Figura 3.5 Ejemplo de cómo realizar el *Paso 1 - Seleccionar la imagen a instalar (versión descargada del sistema operativo)* utilizando Raspberry Pi Imager (izquierda) y Etcher (derecha).

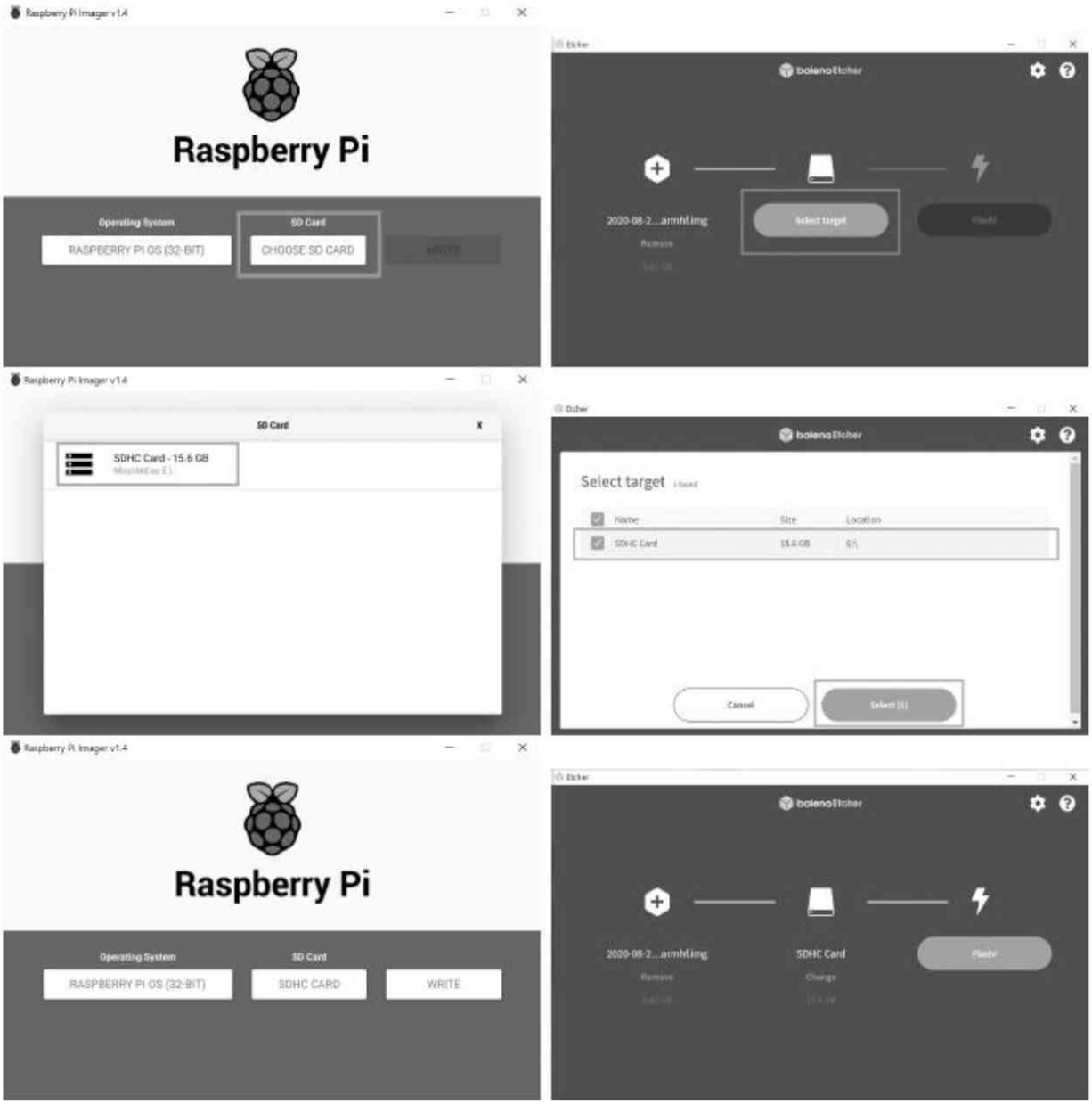


Figura 3.6 Ejemplo de cómo realizar el *Paso 2 - Indicar el dispositivo donde quiere instalarse (tarjeta SD)* utilizando Raspberry Pi Imager (izquierda) y Etcher (derecha).

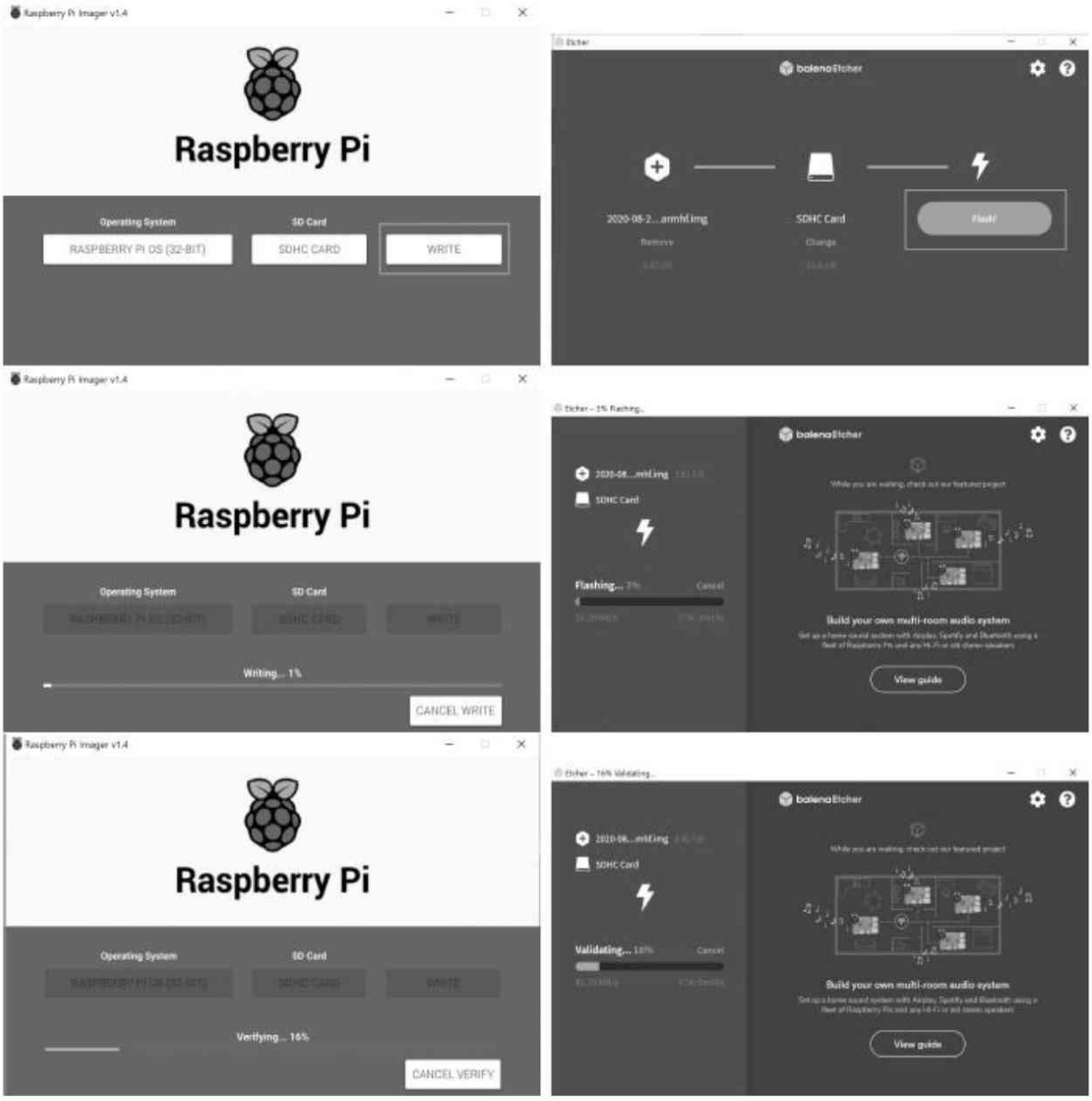


Figura 3.7 Ejemplo de cómo realizar el *Paso 3 - Completar la instalación (también denominado proceso de flash)* utilizando Raspberry Pi Imager (izquierda) y Etcher (derecha).

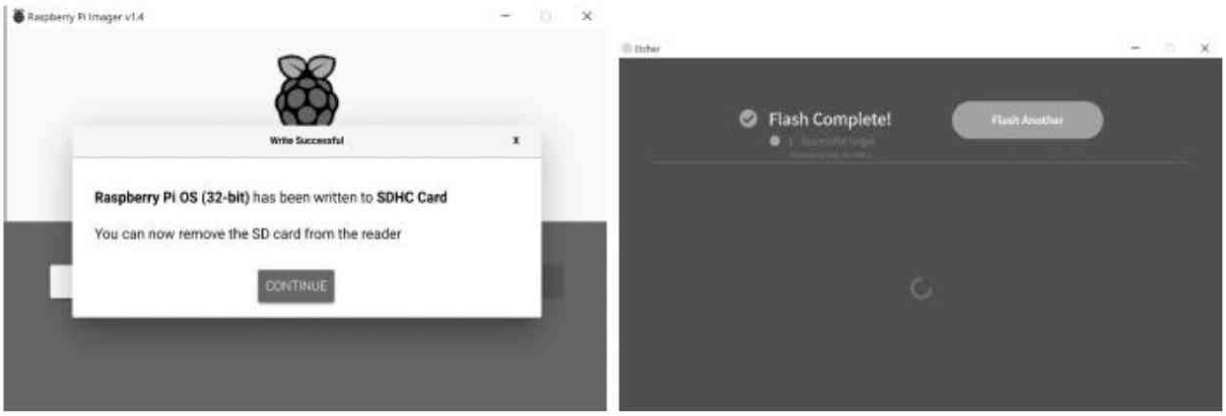


Figura 3.8 Ejemplo de lo que muestran Raspberry Pi Imager (izquierda) y Etcher (derecha) cuando han completado el proceso de *flash* de una tarjeta.

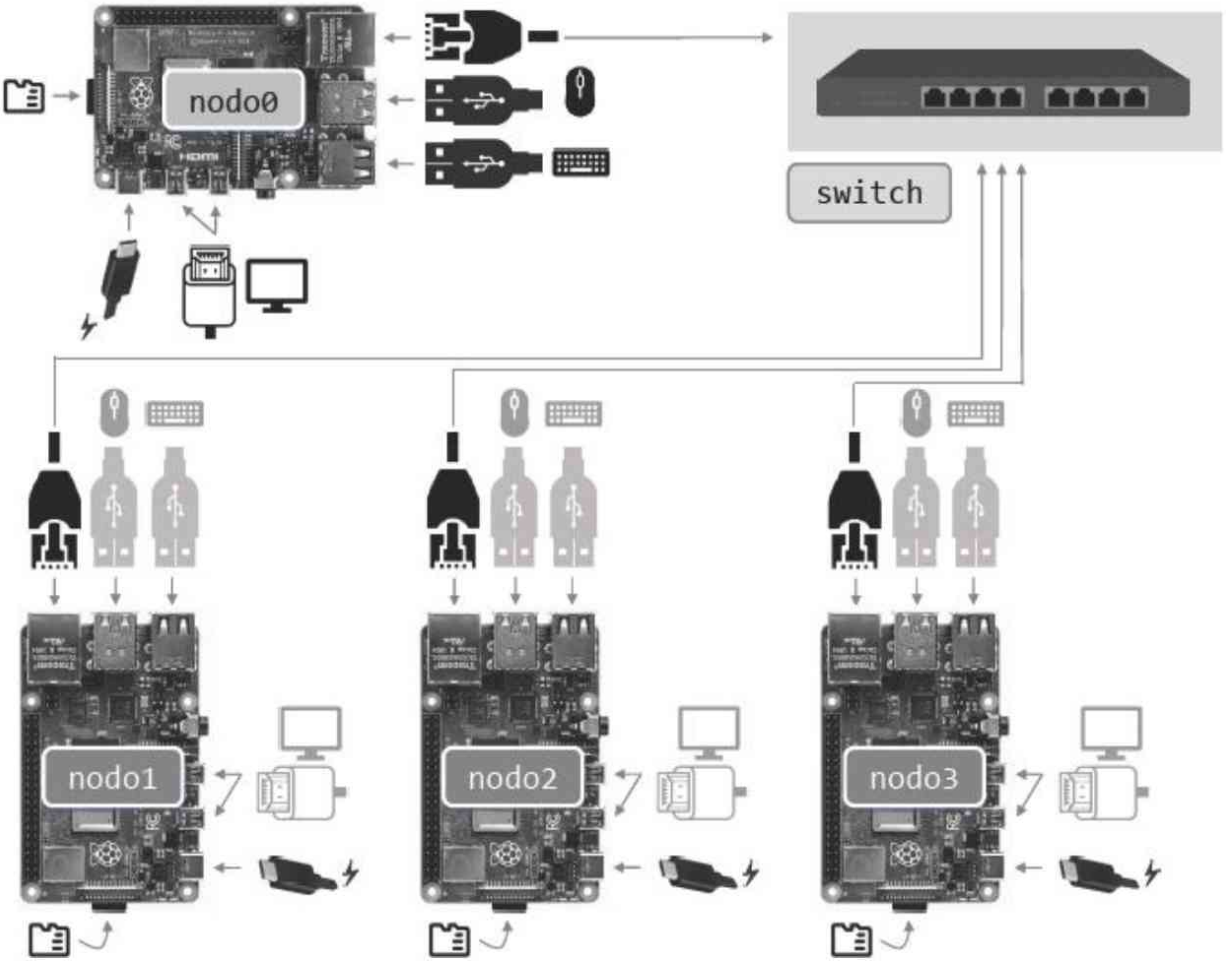


Figura 3.9 Esquema completo del montaje hardware del clúster RPi. Las conexiones y periféricos mostrados en una tonalidad gris serán necesarios

solamente al inicio del proceso de configuración; los representados en color negro deberán estar operativos siempre que se utilice el clúster.

El montaje hardware completo del clúster RPi consta de cuatro pasos principales:

1. Enlazar la RPi a la red de interconexión mediante el puerto Ethernet.
2. Conectar los periféricos.
3. Insertar las tarjetas de memoria Micro SD.
4. Conectar la fuente de alimentación.

3.4.1 Paso 1: Enlazar la RPi a la red de interconexión mediante el puerto Ethernet

Para poder establecer una red local que permita a los nodos comunicarse entre sí, es necesario disponer de un conmutador (*switch*, en inglés) al que conectar cada una de las RPi. Así pues, el cable Ethernet se conectará a la RPi mediante el puerto correspondiente tal como ilustra la Figura 3.10.

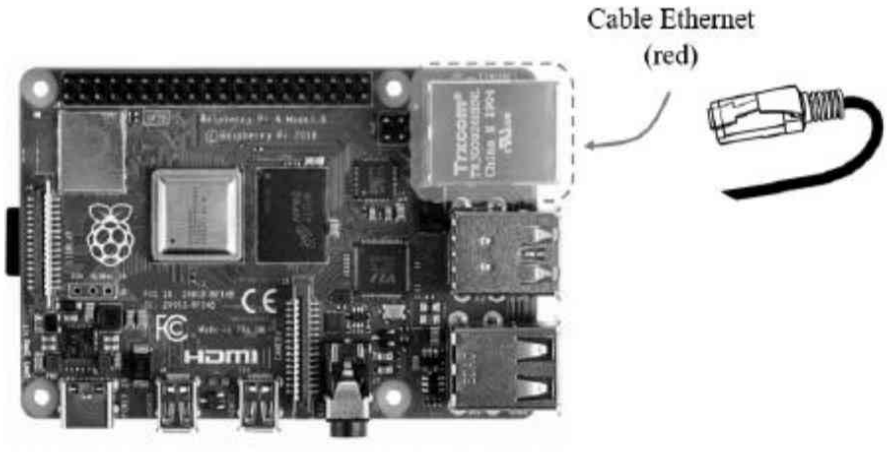


Figura 3.10 Esquema de la inserción del cable Ethernet en el dispositivo RPi.

3.4.2 Paso 2: Conectar los periféricos

Los siguientes componentes a conectar a la RPi son el ratón, el teclado y el monitor. Los dos primeros se conectarán al dispositivo mediante un puerto USB cada uno, y el tercero utilizando el puerto micro-HDMI. La Figura 3.11 ilustra los puertos a utilizar para conectar estos periféricos.

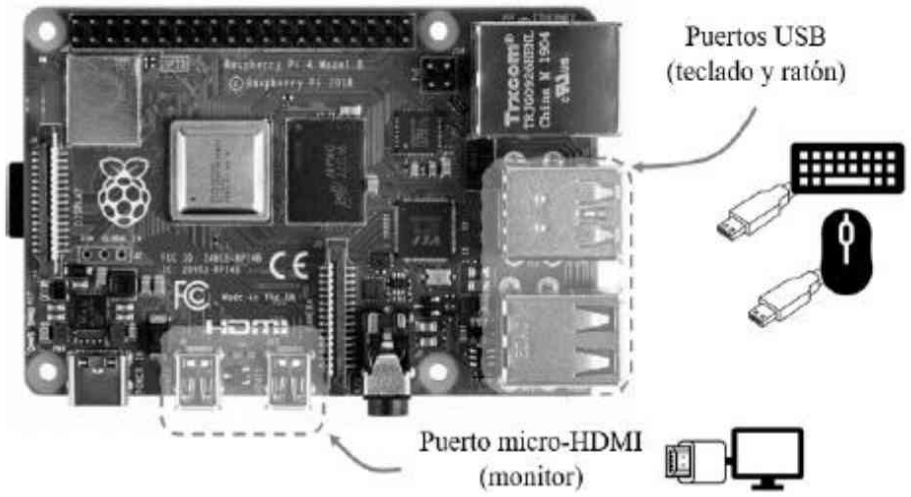


Figura 3.11 Esquema de la inserción de los periféricos (teclado, ratón y monitor) en el dispositivo RPi.

3.4.3 Paso 3: Insertar las tarjetas de memoria SD

Por comodidad y sencillez, se propone la instalación del sistema operativo Raspberry Pi OS antes de la inserción de las tarjetas SD en las ranuras correspondientes. Para ello, será necesario crear una imagen ejecutable del sistema operativo como se ha explicado en el Apartado 3.3.

Una vez completada la instalación del sistema operativo en la tarjeta SD, solamente queda introducir esta en la Raspberry Pi e iniciar el dispositivo, tal como se ilustra en la Figura 3.12. En las versiones modernas de Raspberry Pi OS (exceptuando las Lite), al iniciar el sistema se accede directamente al escritorio y se pueden realizar las configuraciones básicas aprovechando el asistente de configuración inicial (esto se verá en detalle en el Capítulo 4).

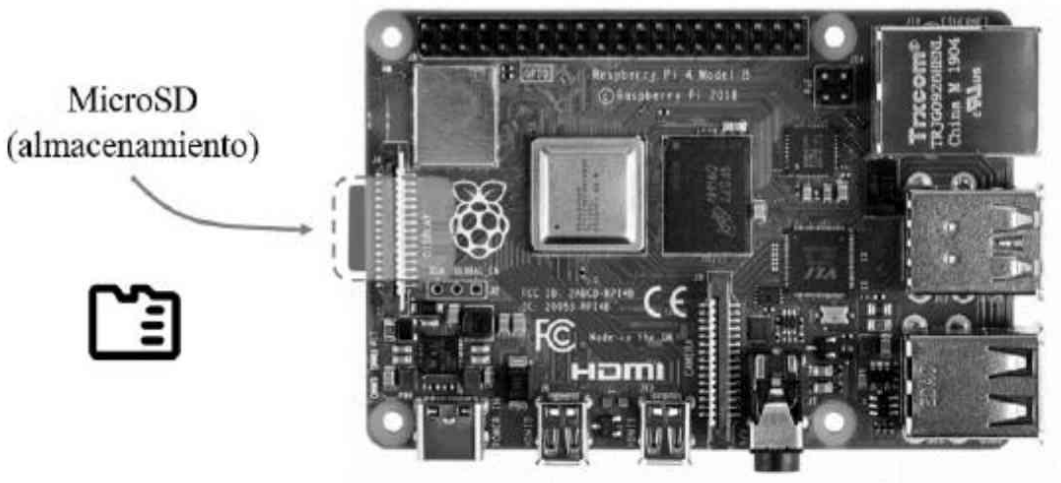


Figura 3.12 Esquema de la inserción de la tarjeta de memoria MicroSD en el dispositivo RPi.

Por defecto, las credenciales son:

- Usuario: pi
- Contraseña: raspberry

3.4.4 Paso 4: Conectar la fuente de alimentación

Lo último que debe conectarse a la RPi es el cable que la enlaza con la fuente de alimentación. Una vez conectado usando el puerto USB C, tal como se ilustra en la Figura 3.13, se iniciará el Sistema operativo instalado en la tarjeta Micro SD que se insertó previamente en el dispositivo y comenzará el proceso de configuración software que se describe en el Capítulo 4.

Es importante prestar atención a los requisitos de alimentación de las Raspberry concretas que se utilicen. Particularmente, la RPi requiere un suministro de 5 voltios y 3 amperios, de lo contrario, puede darse el caso de que la RPi no funcione correctamente, que no arranque, o incluso que deje de funcionar.

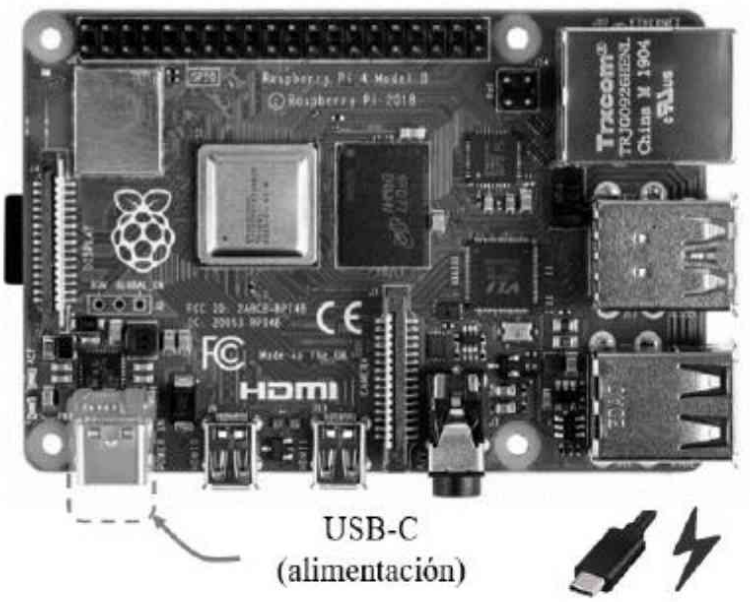


Figura 3.13 Esquema de la inserción de la tarjeta del cable de alimentación Micro USB en el dispositivo RPi.

3.5 Para profundizar...

1. En este capítulo se presenta Raspberry Pi OS como un sistema operativo basado en Linux idoneo para la Raspberry Pi, pero ¿qué sistema operativo se usa comúnmente en los supercomputadores? ¿Qué sistemas operativos conoce? ¿Cuál o cuales de ellos considera que se utilizan en supercomputación?

   Consulte la información recogida en el TOP500 sobre los sistemas operativos utilizados por los supercomputadores recogidos en la lista: https://www.top500.org/statistics/list/

   Seleccione Operating System Family en el desplegable Category. A la vista de los resultados, ¿cree que esta tendencia ha sido siempre así? Consulte la misma información en la lista de junio de 1993 y noviembre de 2001.

2. En este libro se propone la instalación de la versión lanzada en enero de 2021 del sistema operativo Raspberry Pi OS (versión del núcleo de Linux 5.4). ¿Qué versiones de este sistema operativo son las más recientes? ¿Considera que la aparición de nuevas versiones del sistema operativo es frecuente? ¿Qué implicaciones a nivel de administración de sistema puede tener este ritmo de actualización?

3.6 Material recomendado

Uno de los conceptos introducidos en este capítulo es el de *superusuario*, el cual cobra importancia a lo largo de los siguientes capítulos. A modo divulgativo, en la entrada web [1] se explican más detalles sobre este concepto, así que recomendamos revisarla no solamente para conocer más, sino para aclarar bien este concepto antes de proseguir en la lectura de este libro.

Adicionalmente, en este capítulo se ha presentado, de forma breve, la consola Linux. Los comandos más habituales se pueden consultar en el manual [2], donde se puede encontrar una descripción completa y asequible para conocer más sobre estas instrucciones.

[1] Hipertextual - Nasheli Escobar. Qué es el superusuario en Linux y cuál es su importancia, 2015.

[2] William E. Shotts. *The Linux Command Line: A Complete Introduction* . No Starch Press, USA, 2012.

3.7 Resumen

En este capítulo hemos realizado la descripción paso a paso de cómo ensamblar el clúster RPi y sus componentes. Aunque el montaje no requiere ninguna operación a nivel de software, este capítulo propone la instalación del sistema operativo antes de insertar las tarjetas de memoria SD en el sistema para evitar el posible deterioro de las mismas, así como de la ranura correspondiente en la RPi.

En este capítulo se ha aprendido que:

- El sistema operativo es indispensable para el funcionamiento del computador, ya que gestiona los recursos hardware y software mientras atiende las peticiones y necesidades del usuario.
- La consola permite al usuario dar instrucciones al sistema operativo a través de la línea de comandos.
- La familia de sistemas operativos GNU/Linux es ampliamente utilizada en supercomputación por su seguridad, eficiencia y robustez. En este manual se usará una versión de ese sistema operativo adaptado al uso de RPi.
- La secuencia sencilla de pasos para el ensamblado hardware ha resultado en la construcción del clúster RPi que se usará en los siguientes capítulos para seguir avanzando en el campo de la HPC.

3.7.1 ¿Qué viene a continuación?

El resto de capítulos de esta parte de la obra se centran en la configuración software pertinente del clúster RPi para que todos los componentes trabajen conjuntamente como lo harían en un supercomputador.

Concretamente, en el siguiente capítulo se configura la red interna del clúster para permitir la comunicación entre sus distintos nodos y habilitar un

sistema de ficheros compartido.

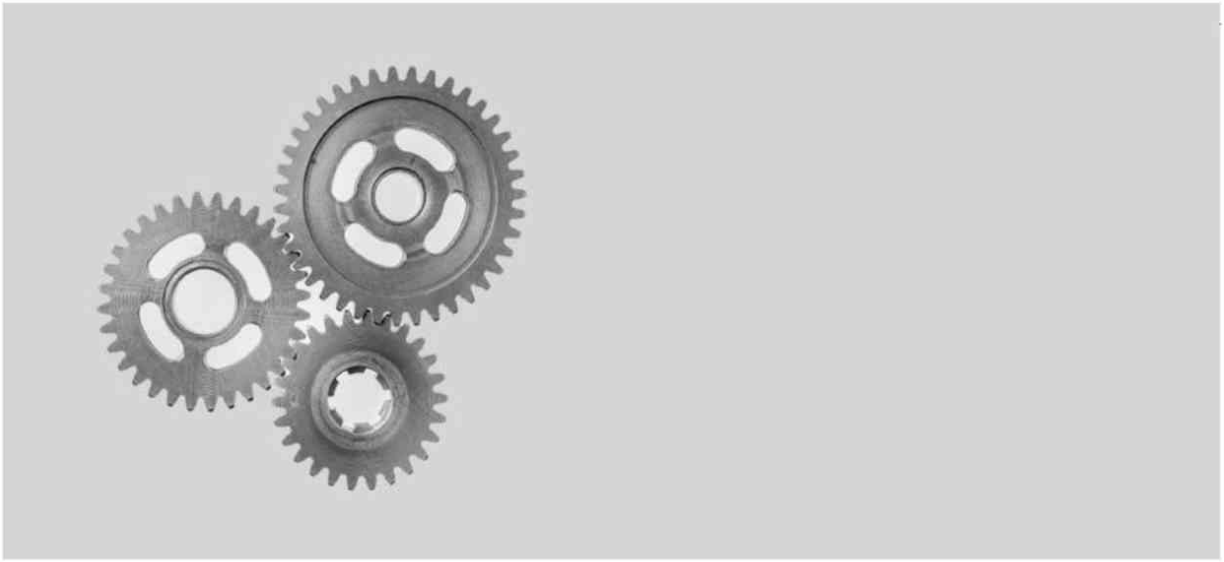
1 https://help.ubuntu.com/kubuntu/desktopguide/es/direc-ries-file-sysms.htmlml

2 https://www.raspberrypi.org/downloads/raspberry-pi-os

3 https://downloads.raspberrypi.org/raspios_arm64/images/

4 https://www.raspberrypi.org/downloads

5 https://www.balena.io/etcher/



4. Configuración del clúster

Por definición, un clúster computacional es un conjunto de ordenadores conectados entre sí para trabajar conjuntamente. Aunque se pueden encontrar clústeres que actúan como servidores web y permiten procesar múltiples peticiones concurrentemente, de forma masiva, actualmente, se utilizan para el procesamiento colaborativo de la información. Por ello, en este capítulo se verá cómo configurar el clúster y qué software se utiliza para que el clúster RPi tenga una funcionalidad similar a la de un supercomputador, aunque, por supuesto, manteniendo las distancias en cuanto al rendimiento.

4.1 Configuración inicial de los nodos

Este apartado parte de la base de que las tarjetas micro SD ya tienen instalado el sistema operativo y el clúster está físicamente montado, tal como se ha descrito en el Capítulo 3. De entre las múltiples aplicaciones preinstaladas en el sistema operativo, durante la configuración del clúster se recomienda el uso de las siguientes:

- El editor de textos Mousepad o Nano, dependiendo de si la interfaz gráfica está habilitada o no.
- La consola del sistema Terminal.

Tenga en cuenta que las aplicaciones preinstaladas pueden variar si la versión del sistema operativo instalado difiere de la propuesta en el Capítulo 3.

Para la configuración del clúster se ha optado por utilizar los siguientes identificadores de nodos, aunque puede utilizar los suyos propios:

- Nodo *front-end* : nodo0 .
- Nodos de cómputo: nodo1, nodo2, nodo3 .

Si no se dispone de un juego de monitor, ratón y teclado para cada nodo RPi, estos periféricos se deberán ir desconectando y conectando a los diferentes nodos a medida que se vayan configurando. Sin embargo, esto no debería ser una preocupación, ya que la configuración inicial solo se tendrá que hacer una vez y tan pronto como la red de interconexión esté lista, el resto de configuraciones en los nodos se podrá hacer remotamente desde el *front-end*, tal como se explica más adelante. Para reducir al máximo la reconexión de los periféricos, se comenzará con la configuración de los nodos de cómputo y, finalmente, se configurará el *front-end*, de modo que en última instancia sea este quien tenga conectado el monitor, el teclado y el ratón.

4.1.1 Nodos de cómputo

En este apartado se describe cómo configurar los nodos de cómputo (nodo1, nodo2, nodo3 en el ejemplo presentado) para que sean accesibles mediante la red del clúster.

En aquellos pasos en los que es necesario modificar un fichero, si se deshabilita la interfaz gráfica de los nodos de cómputo, se deberá útilizar un editor de texto para la línea de comandos. En esta obra se utiliza Nano[1], aunque se puede encontrar también Vi instalado en el sistema operativo.

A modo de resumen, aunque Nano muestra en todo momento las combinaciones de teclas a pulsar para llevar a cabo las acciones posibles en la barra inferior, a continuación se introducen brevemente algunas indicaciones para llevar a cabo operaciones básicas (una vez abierto el fichero):

- Guardar sin salir: pulsar la combinación de teclas [Ctrl] + [O].
- Salir del editor (guardando o no los cambios efectuados): pulsar la combinación de teclas [Ctrl] + [X]. A continuación, el editor pregunta si se quieren guardar los cambios; para confirmar los cambios se pulsa la tecla [S], en caso contrario, [N].

Deshabilitar la interfaz gráfica, aunque no sea necesario, es recomendable para disponer de más recursos para ejecutar las aplicaciones de usuario en los nodos de cómputo.

Configuración básica

Al arrancar por primera vez el sistema operativo de la RPi aparecerá un asistente de configuración básica (veáse Figura 4.1). En caso de no aparecer, se puede acceder a él mediante el menú Preferences → Raspberry Pi Configuration, tal como se explica más adelante.

Figura 4.1 Asistente del sistema operativo para las configuraciones básicas que se muestra al arrancar por primera vez.

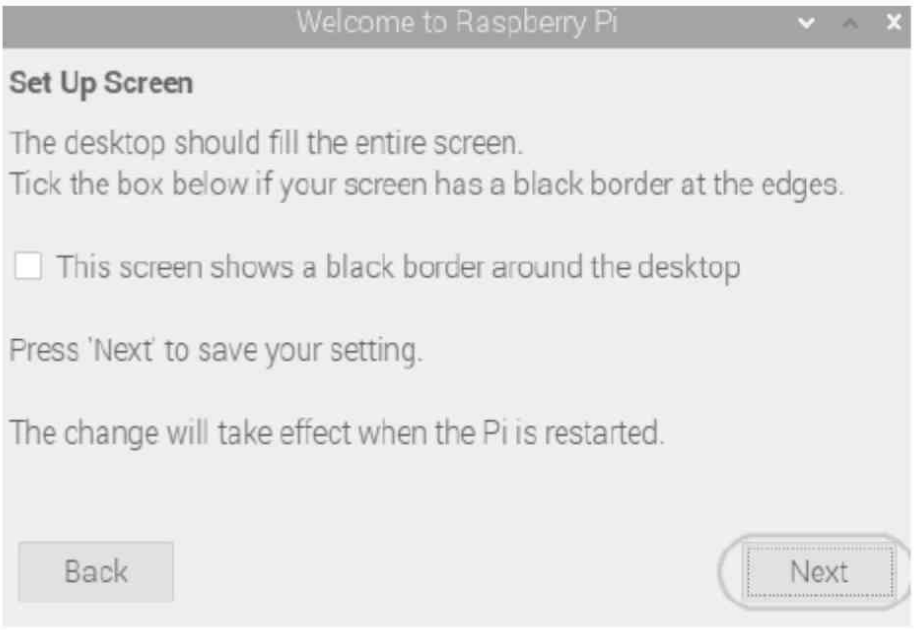
Si se aprovecha el asistente de configuración, en él se fijarán:

☐El país, el idioma y la zona horaria en la que van a operar los dispositivos (por ejemplo, *España* , *Español europeo* , *Madrid*).

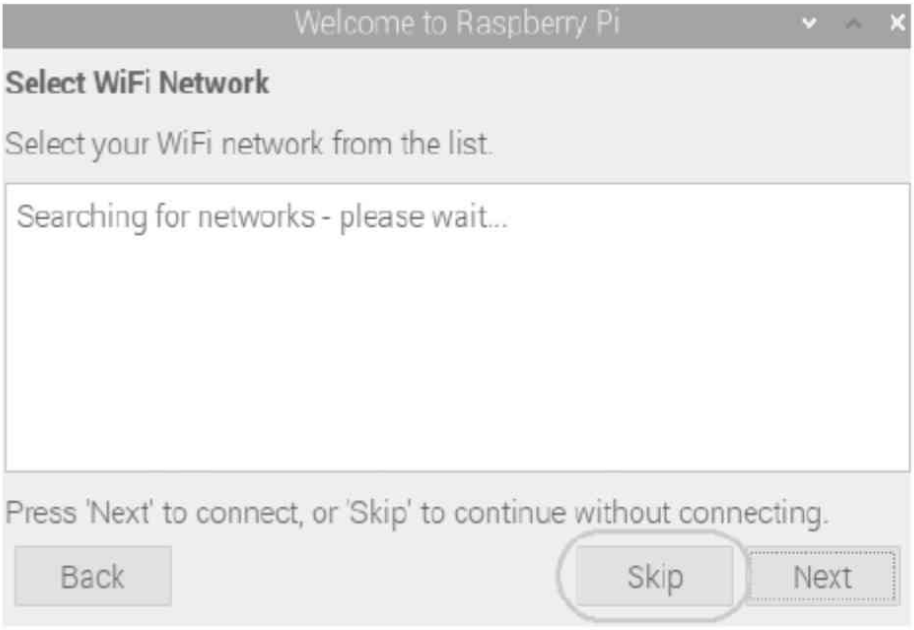
☐Contraseña del sistema. Por defecto, el usuario es pi y la contraseña

raspberry . Aunque en este libro se ha decidido utilizar la contraseña por defecto *raspberry* , puede cambiarse para aumentar el grado de seguridad del sistema. Además, no es necesario que todos los nodos tengan la misma contraseña.

☐Configuración de pantalla. El asistente ofrecerá, a continuación, la posibilidad de eliminar los bordes negros de la pantalla que se muestran en algunos casos (dependiendo de la resolución del monitor). Se avanza en la configuración pulsado en *Next* .



☐Conexión Wi-Fi. Al arrancar el sistema, el asistente sugerirá establecer la conexión con una red Wi-Fi. Este paso debe saltarse y no habilitarse ninguna conexión (opción *Skip*).



☐Actualización del software. Al arrancar el sistema, el asistente sugerirá también actualizar el software. Este paso tampoco debe llevarse a cabo en este punto (opción *Skip*).

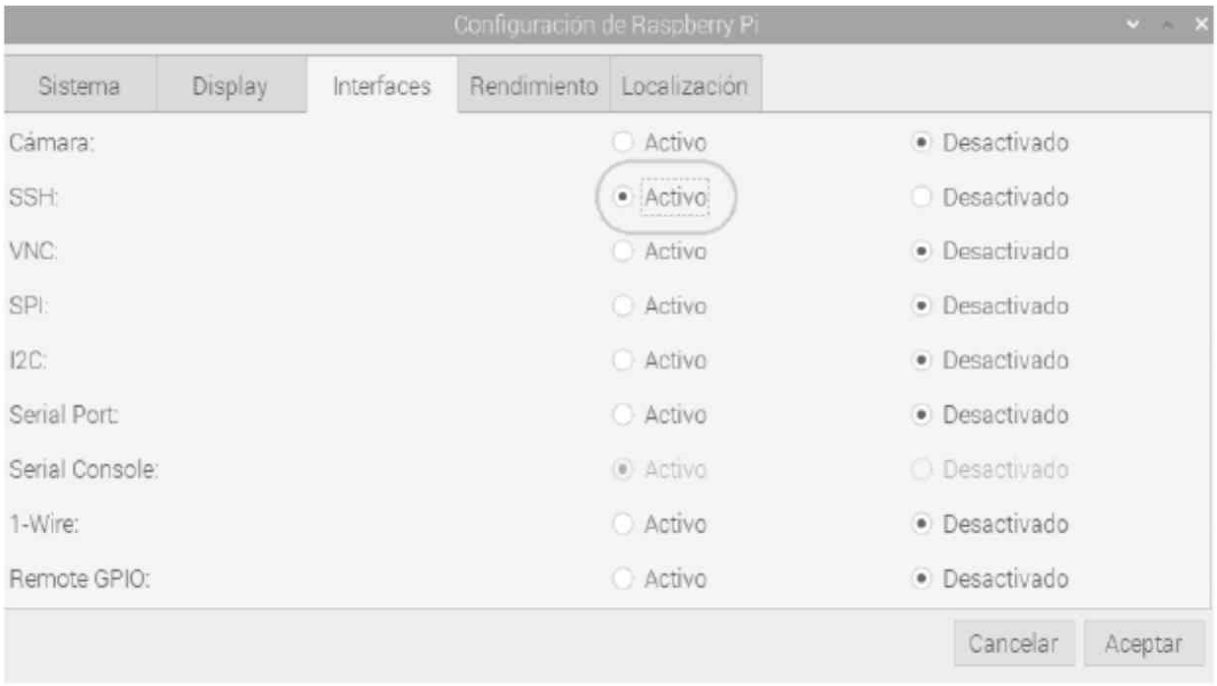
☐ Finalmente se cierra el asistente (opción *Done*).

Para aplicar los cambios se reinicia el dispositivo accediendo al menú Shutdown... y pulsando en la opción *Reboot*.

Tras el reinicio, la configuración continúa accediendo al menú Preferencias → Configuración de Raspberry Pi:

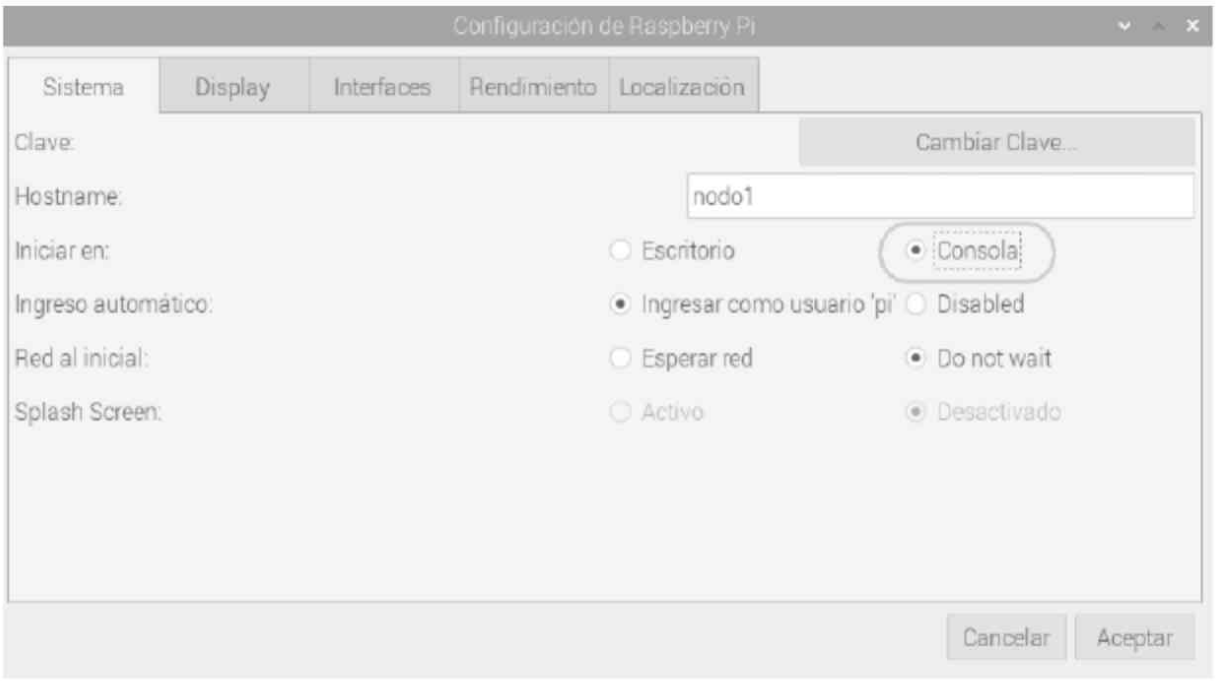
☐ Pestaña *Sistema* : el nombre del nodo (según el nodo que se esté configurando: nodo1, nodo2 o nodo3).

☐ Pestaña *Interfaces* : habilitar la interfaz SSH para poder acceder al nodo remotamente.



Opcionalmente, en este mismo menú Preferencias → Configuración de Raspberry Pi, se recomienda fijar algunas opciones adicionales. Dado que los nodos de cómputo se destinan únicamente al cálculo de operaciones, no es necesario que tengan una interfaz gráfica, ya que no llevarán a cabo un uso intensivo de la GPU ni tendrán acceso a Internet (se deshabilitará el Wi-Fi) ni harán uso de conexiones Bluetooth. Con estos ajustes, se consigue reducir el consumo energético y dedicar más cantidad de memoria RAM a los procesos de usuario:

☐ Pestaña *Sistema* : iniciar en modo consola.



☐ Pestaña *Rendimiento* : limitar la memoria de GPU a 16 MB.

En este punto, los cambios realizados requieren reiniciar el dispositivo.

Opcionalmente, se desconectarán las conexiones Wi-Fi y Bluetooth del nodo modificando el fichero */boot/config.txt* añadiendo las líneas que figuran a continuación. Esto se debe hacer con permisos de superusuario con sudo

nano /boot/config.txt o sudo mousepad /boot/config.txt (esta segunda opción se podrá utilizar si no se ha desactivado la interfaz gráfica):

Fichero: /boot/config.txt

```
dtoverlay=disable–wifi
dtoverlay=disable–bt
```

Estos cambios requieren que se reinicie el nodo. Esto se puede hacer desde la consola con sudo reboot.

Configuración de la red de comunicación

Los nodos del clúster deberán estar interconectados mediante una red funcional. Antes de comenzar con la configuración de la red es conveniente comprender los siguientes conceptos:

- IP (*Internet Protocol*): utiliza una serie de direcciones numéricas asignadas a cada dispositivo conectado a la red para permitir su comunicación. El formato de una dirección IP consiste en cuatro valores decimales, separados por puntos y pertenecientes al rango [0-255].
- SSH (*Secure Shell*): permite acceder y administrar un dispositivo distinto del que se está manejando, estableciéndose para ello una conexión segura entre ambos sistemas.
- DHCP (*Dynamic Host Configuration Protocol*): habilita la configuración automática del direccionamiento IP (IP dinámica), permitiendo así la asignación automática de la dirección IP, la máscara de subred, la dirección de puerta de enlace predeterminada o la dirección del DNS a un dispositivo en una red.
- DNS (*Domain Name Server*): el servidor de nombres de dominio traduce nombres de dominio en direcciones IP numéricas para identificar

servicios o dispositivos en la red. Por ejemplo: la dirección http://www.google.com se traduce como IP *172.217.17.14* . Nótese que esta IP puede cambiar y el DNS se encargará de mantener dicha asociación IP-nombre actualizada. Además, la utilidad del DNS se basa en el hecho de que, para un humano, suele resultar más fácil recordar un nombre de dominio que un número de IP.

Mediante los pasos que siguen se configurará en los nodos de cómputo la interfaz de red que se utilizará para las comunicaciones internas en el clúster:

☐ Para los nodos de cómputo se deshabilitará la configuración DHCP. En lugar de utilizar una dirección IP dinámica, se proporcionará una estática; para ello hay que editar el contenido del fichero */etc/dhcpcd.conf* :

Fichero: /etc/dhcpcd.conf

```
interface eth0
static ip_address =192.168.0.{ X }/24
static routers =192.168.0.1
static domain_name_severs =192.168.0.1
```

La IP 192.168.0.1 se reserva para la puerta de enlace (que también actuará como servidor de nombres), la cual corresponde en este caso al *front-end*. Cada nodo deberá tener su propia IP, por lo que la "{X}" en ip_address deberá ser sustituida en cada caso por el valor que corresponda (en el ejemplo utilizado, un número distinto para los nodos 1, 2 y 3). Por ejemplo, la Tabla 4.1 muestra la configuración utilizada en este manual.

Nodo	Tipo de nodo	Dirección IP asignada
nodo0	*front-end*	192.168.0.1/24
nodo1	Nodo de cómputo	192.168.0.2/24

nodo2	Nodo de cómputo	192.168.0.3/24
nodo3	Nodo de cómputo	192.168.0.4/24

Tabla 4.1 Ejemplo de direcciones IP para cada nodo.

☐ Finalmente, para aplicar esta configuración y hacerla efectiva existen dos opciones: reiniciar la RPi o reiniciar la interfaz de red, tal y como se muestra a continuación (nótese que nodoX representa a nodo1, nodo2 o nodo3).

```
COMENTARIO: Opción 1) Reiniciar RPi
pi@raspberry:~ $ sudo reboot

COMENTARIO: Opción 2) Reiniciar la interfaz de red
pi@raspberry:~ $ sudo ifconfig eth0 down
pi@raspberry:~ $ sudo ifconfig eth0 up
```

Consola 4.1 Distintas opciones para hacer efectivos los cambios en las configuraciones relacionadas con interfaces de red.

Una vez configurada la red por completo (incluyendo lo detallado en el apartado 4.1.2 que sigue), los nodos de cómputo serán accesibles desde el *front-end* (remotamente), por lo que ya no será necesario volver a utilizarlos localmente, es decir, el monitor, el teclado y el ratón solo serán necesarios en el *front-end*.

4.1.2 *Front-end*

El *front-end* o nodo0 será la vía de acceso al clúster. Por tanto, aunque muchas de las configuraciones sean las mismas que en los nodos de cómputo, la

preparación de este nodo será más compleja.

Configuración básica

Del mismo modo que inicialmente se han configurado los nodos de cómputo, se configurará el *front-end*. Desde el asistente de configuración (o bien accediendo al menú Preferences → Raspberry Pi Configuration), se define:

☐ El país, el idioma y la zona horaria en la que van a operar los dispositivos (por ejemplo, *España – Español europeo – Madrid*).

☐ Contraseña del sistema. De nuevo, se sugiere mantener el usuario pi y la contraseña raspberry que vienen por defecto.

☐ Conexión Wi-Fi. Al contrario que en los nodos de cómputo, en el *front-end* sí debe configurarse una conexión con una red Wi-Fi. Será necesario que este nodo disponga de conexión a Internet para poder descargar parte de los paquetes software necesarios para las configuraciones en el clúster.

☐ Actualización del software. Tal como se indicó para los nodos de cómputo, este paso no es necesario en este momento y se realizará posteriormente, por lo que se omite (opción *Skip*).

☐ Finalmente se cierra el asistente mediante *Done* .

Para que los cambios realizados tengan efecto, es necesario reiniciar el nodo.

Adicionalmente, desde el menú Preferencias → Configuración de Raspberry Pi se deben establecer:

☐ Pestaña *Sistema* : el nombre del nodo (nodoo).

☐ Pestaña *Interfaces* : Habilitar la interfaz SSH .

De nuevo, para que los cambios tengan efecto, se reinicia el nodo.

Opcionalmente, dado que en ningún momento se propone el uso de conexión Bluetooth, esta puede desactivarse en el *front-end* también:

☐ Se desconectará la conexión Bluetooth del nodo modificando el fichero

/boot/config.txt añadiendo la línea que figura a continuación (esto se debe hacer con permisos de superusuario):

Fichero: /boot/config.txt

```
dtoverlay=disable–bt
```

En el caso del nodo *front-end* no se recomienda inicialmente ni el inicio en modo consola ni limitar la memoria GPU a 16 MB. No obstante, estos cambios pueden aplicarse (y revertirse) *a posteriori* en cualquier momento.

Tras estos cambios, debe reiniciarse el nodo.

Configuración de las redes de comunicación

A continuación, se explicará cómo configurar la red privada (Ethernet) y cómo se enruta el tráfico del clúster, ya que este nodo actúa como puerta de enlace para los nodos de cómputo. Hay que tener en cuenta que este nodo también necesita estar conectado a una red pública desde la que se tenga acceso a Internet, la cual será accesible mediante la interfaz Wi-Fi del nodo.

Mediante los pasos que siguen se configurarán las comunicaciones en el *front-end*.

☐Configurar la interfaz de red Ethernet con una dirección IP estática para la comunicación en la intranet; se hará editando el contenido del fichero / etc/dhcpcd.conf :

Fichero: /etc/dhcpcd.conf

```
interface eth0
static ip_address =192.168.0.1/24
static routers =192.168.0.1
```

```
static domain_name_severs =192.168.0.1
```

☐Para aplicar esta configuración, se deberá reiniciar la RPi o reiniciar la interfaz de red, tal y como se muestra en la Consola 4.1.

☐Conectarse a una red Wi-Fi (con acceso a Internet), de modo que el clúster tenga acceso al mismo. Asimismo, otros dispositivos que compartan la red Wi-Fi podrán acceder al *front-end* del clúster, y viceversa.

☐Crear el fichero */lib/dhcpcd/dhcpcd-hooks/60-gw* y editarlo con permisos de superusuario (si se emplea el editor de textos Mousepad, por ejemplo, esto puede hacerse ejecutando en el terminal la orden sudo mousepad /lib/dhcpcd/dhcpcd-hooks/60-gw) . Se añadirá el siguiente contenido al fichero:

Fichero: /lib/dhcpcd/dhcpcd-hooks/60-gw

```
route del default gw 192.168.0.1
```

De este modo se indica que por defecto no se dirija el tráfico a la red interna (la que conecta el *front-end* y sus nodos de cómputo), para así poder tener acceso a Internet a través de la interfaz Wi-Fi. Para que los cambios tengan efecto, se reinciará la RPi.

☐Una vez reiniciado el dispositivo, se puede comprobar que desde el *front-end* y mediante SSH se puede acceder a todos los nodos de cómputo. Por ejemplo, ejecutando ssh 192.168.0.2 desde la consola del sistema en el *front-end* , debe poder alcanzarse nodo1 (siempre que se hayan utilizado las direcciones IP presentadas como ejemplo en este manual).

☐Editar el contenido del fichero */etc/hosts* con permisos de superusuario (si se emplea el editor de textos Mousepad, por ejemplo, esto puede hacerse con el comando sudo mousepad /etc/hosts) sobreescribiendo el

contenido con lo siguiente:

Fichero: /etc/hosts

192.168.0.1	nodo0
192.168.0.2	nodo1
192.168.0.3	nodo2
192.168.0.4	nodo3

Es importante añadir una línea por nodo en el clúster. Este fichero es el que utilizará el DNS para identificar cada dirección con el nombre de nodo asignado durante la configuración.

☐ Para aplicar esta configuración, debe reiniciarse de nuevo la RPi. Una vez el dispositivo esté en marcha de nuevo, puede comprobarse que el *front-end* tiene acceso a los nodos de cómputo mediante SSH (ahora también utilizando su nombre y no solamente indicando la IP correspondiente), por ejemplo, ejecutando ssh nodo2 .

☐ El fichero */etc/hosts* debe presentar este mismo contenido en todos los nodos del clúster. Como ya se han configurado las interfaces de red en los nodos de cómputo, el fichero puede ser enviado desde el *front-end* mediante SSH al resto. Para que este envío sea posible, en primer lugar hay que generar las claves SSH del *front-end* utilizando el comando ssh-keygen (disponible en la Consola 4.2). En este manual se han utilizado las opciones por defecto para la generación de estas claves, es decir, en el proceso de creación de claves todas las opciones han sido aceptadas directamente pulsando la tecla de retorno de carro (*intro*) hasta que se ha completado la generación, si bien pueden incluirse contraseñas por

motivos de seguridad.

☐ Enviar la clave pública (comando ssh-copy-id) y el fichero */etc/hosts* que contiene los nombres a los nodos de cómputo desde el *front-end* , tal como se muestra a continuación (de nuevo, {X} debe ser sustituido por el identificador del nodo en cuestión, es decir, 1, 2 o 3):

```
pi@nodoo:~ $ ssh-copy-id nodo{X}
pi@nodoo:~ $ scp /etc/hosts nodo{X}:
pi@nodoo:~ $ ssh nodo{X} sudo mv hosts /etc/hosts
```

Nótese que, tras el segundo comando, el nodo al que se ha enviado el fichero */etc/hosts* lo tiene en su directorio raíz. La tercera instrucción se encarga de mover dicho fichero del directorio raíz al directorio */etc*.

```
pi@nodo0:~ $ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/pi/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/pi/.ssh/id_rsa.
Your public key has been saved in /home/pi/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:sUIgJinD8ka0ElyErJaqqOuUP9ktpSoZefXs9FQ66Kw pi@nodo0
The key's randomart image is:
+---[RSA 2048]----+
|*oBo.            |
|=0.o .           |
|=+o   . .        |
|.+o  o   o .     |
|o.. . + S o      |
|.o..   B +       |
|oo+ o 0 o .      |
|+o.o + = .       |
|=o.ooEo          |
+----[SHA256]-----+
```

Consola 4.2 Generación de las claves SSH utilizando en comando ssh-keygen.

4.2 Configuración del sistema de ficheros compartido

En el Apartado 2.3.3 se describieron algunos de los sistemas de ficheros utilizados habitualmente en entornos HPC. Concretamente, aunque en grandes instalaciones se utilizan sistemas de ficheros distribuidos, la configuración de estos supone una complejidad no asumible para un clúster de las características descritas en este manual. Por esto, se optará por configurar un sistema de ficheros compartido.

Uno de los objetivos a alcanzar es disponer de un sistema de ficheros común a todos los nodos, de manera que todo lo que se almacene allí quede accesible y pueda ser creado, consultado, modificado o eliminado desde cualquier nodo del sistema. Para lograr esto, se empleará el protocolo de sistema de archivos de red (*Network File System* [NFS], en inglés)2.

Configuración a realizar en el *front-end*

En este paso de la configuración, el clúster RPi necesitará por primera vez conexión a Internet para instalar dependencias software. Lo primero que se hará es actualizar los repositorios software ya instalados. Acto seguido se instalarán los paquetes necesarios para poner en marcha el sistema de ficheros compartido. Cabe remarcar que todos los comandos descritos a continuación deben ejecutarse desde el *front-end*.

☐ La actualización de repositorios y la instalación de los paquetes correspondientes al NFS pueden realizarse mediante las instrucciones que figuran a continuación:

```
pi@nodoo:~ $ sudo apt-get update
pi@nodoo:~ $ sudo apt-get install -y nfs-kernel-server
```

El directorio compartido en el clúster RPi se puede configurar mediante los pasos siguientes:

☐Crear un directorio para que sea compartido en la raíz (en nuestro ejemplo se llamará *SHARED*).
☐Extender los permisos de dicho directorio para que todos los nodos del sistema puedan leer, escribir y ejecutar.
☐Mediante el *sticky bit* se añadirá la seguridad necesaria a este directorio para que únicamente su propietario lo pueda borrar. La secuencia de comandos para realizar las tres acciones descritas puede verse en la Consola 4.3.

```
pi@nodo0:~ $ sudo mkdir /SHARED
pi@nodo0:~ $ sudo chmod 777 /SHARED
pi@nodo0:~ $ sudo chmod +t /SHARED
```

Consola 4.3 Creación del directorio */SHARED*; extender sus permisos de lectura, escritura y ejecución, y utilizar el *sticky bit* para que solamente el propietario pueda borrarlo.

☐Editar el contenido del fichero */etc/exports* con permisos de superusuario (si se emplea el editor de textos Mousepad, por ejemplo, esto puede hacerse con el comando sudo mousepad /etc/exports) añadiendo el siguiente contenido:

Fichero: /etc/exports

```
/ SHARED nodo1 (rw, sync, no_subtree_check)
```

```
/ SHARED nodo2 (rw, sync, no_subtree_check)
/ SHARED nodo3 (rw, sync, no_subtree_check)
```

Esto sirve para indicar el directorio de cada nodo donde se montará /SHARED del *front-end*.

Es muy importante no incluir espacios en las opciones escritas entre paréntesis.

☐Finalmente, se debe ejecutar el comando que sigue para exportar el directorio y así permitir que se acceda a él remotamente.

```
pi@nodo0:~ $ sudo exportfs -r
```

Configuración a realizar en los nodos de cómputo

Una vez listo el directorio a compartir, es momento de configurar el acceso a este desde el resto de nodos, de nuevo empleando SSH desde el *front-end* para acceder a ellos. Por ejemplo, para acceder al nodo2 se hará como sigue:

```
pi@nodo0:~ $ ssh nodo2
```

☐Para cada nodo de cómputo se llevarán a cabo los mismos comandos mostrados en la Consola 4.3.

☐A continuación se tiene que editar el contenido del fichero */etc/fstab* con permisos de administrador (si se emplea el editor de textos Mousepad, por ejemplo, esto puede hacerse con el comando sudo mousepad /etc/fstab o sudo nano /etc/fstab si se usa el modo consola), y añadir:

Fichero: /etc/fstab

```
nodo0:/SHARED      /SHARED      nfs      defaults,x-systemd.automount      0      0
```

Este fichero contiene la información para montar sistemas de ficheros. En este caso, se monta mediante NFS el directorio */SHARED* del nodo0 (*front-end*) en el directorio local */SHARED* de cada nodo. Gracias a incluir esta configuración en el fichero */etc/fstab*, el montaje se realiza automáticamente al iniciar el sistema.

☐ Para que todos estos cambios tengan efecto, se ejecutará el comando que sigue en los nodos 1, 2 y 3. Continuando con el ejemplo del nodo2, la consola será la siguiente:

```
pi@nodo2:~ $ sudo mount -a
```

Se recomienda comprobar que todos los nodos comparten efectivamente la carpeta creada para ello y pueden ver/actualizar su contenido. Con este fin, puede crearse un fichero de texto vacío (por ejemplo, con el comando touch /SHARED/prueba) e ir editándolo mediante SSH desde cada uno de los nodos, revisando que el contenido sea el que se espera.

4.3 Para profundizar...

1. Suponiendo que se dispusiera de un nodo adicional y se conectara al clúster ya configurado para que funcionara como nodo de cómputo, ¿qué pasos deberían seguirse para que el nuevo nodo esté operativo y trabajando con el resto de nodos?

Recuerde la información presentada en el Apartado 4.1.1.

2. Y si el nuevo nodo fuese a utilizarse como segundo *front-end* para dar servicio a más usuarios, ¿qué pasos deberían seguirse en este caso?

Recuerde la información presentada en el Apartado 4.1.2.

3. El protocolo DHCP utilizado para configurar la interfaz de red *wlan0* del *front-end* asigna la IP de forma dinámica. Esto quiere decir que en algún momento esta IP podría variar, por ejemplo, si se mueve el clúster a otra red. ¿Cómo se podría conocer la IP pública del clúster sin necesidad de conectarse si se dispusiera de un display LCD?

En el siguiente enlace se muestra una propuesta de solución a este problema:

https://makezine.com/projects/build-compact-4-noderaspberry-pi-clusterter

4.4 Material recomendado

A lo largo de este capítulo se han visto distintos protocolos y servicios propios de las redes de comunicación. Puesto que no es objetivo de este libro explicar en profundidad los detalles de todos ellos, se propone la lectura de materiales como [1], en el que se explican las direcciones IP y el correspondiente protocolo. Respecto a la asignación automática de direcciones IP y a la resolución de nombres, se recomienda la lectura de [2] y [3], en los que se detalla el funcionamiento de DHCP y DNS. Además, para una mejor comprensión del protocolo SSH, se sugiere la lectura de [4].

Por otro lado, en [5] se presenta, en formato vídeo, una introducción al protocolo NFS, así como los pasos a seguir para configurar un sistema de ficheros compartido.

[1] Paessler AG. IT Explained: Dirección IP. https://www.es.paessler.com/it-plained/ip-addressaddress . Accedido por última vez: nov. 2020.

[2] IONOS. El DHCP y la configuración de redes. https://www.ionos.es/digi-talguide/servi-dores/figuracion/que-esel-dhcp-como-funciona/ciona/ . Accedido por última vez: nov. 2020.

[3] IONOS. El servidor DNS y la resolución de nombres en Internet. https://www.ionos.es/digitalguide/servidores/knowhow/que-es-el-servidor-dns-y-como-fun-ciona/ . Accedido por última vez: nov. 2020.

[4] Infranetworking. Servidor SSH. https://blog.infranetworking.com/ser-vidor-ssh/ . Accedido por última vez: nov. 2020.

[5] SuGE3K. Servicio de red NFS. https://www.youtube.com/watch?-v=OB-JKKoKTsyg&-list=-cPK3h0-D7Cs-Bx4eRZgCufM48ntq0 .

Accedido por última vez: nov. 2020.

4.5 Resumen

Siguiendo los pasos descritos en este capítulo, se ha llevado a cabo la configuración básica del clúster RPi, compuesta por:

- Configuración de preferencias básicas (zona horaria, idioma, nombre de los nodos, habilitación de la interfaz SSH, activar/desactivar las conexiones Wi-Fi según proceda, desactivar las conexiones Bluetooth, etc.).
- Configuración de la red de comunicación, la cual abarca configurar las IP de los nodos, las conexiones SSH, el protocolo DHCP y el DNS.
- Configuración del sistema de ficheros compartido.

Antes de continuar con el siguiente capítulo, se recomienda comprobar que todas las configuraciones realizadas en este funcionan correctamente. En particular:

- Comprobar que, mediante SSH y utilizando el nombre del nodo al que se quiera acceder (nodo00, nodo01, nodo02 o nodo03), se puede acceder desde cualquiera de los nodos al resto.
- Comprobar que se puede crear, eliminar o editar un fichero en la carpeta /SHARED y los cambios son visibles desde el resto de los nodos.

4.5.1 ¿Qué viene a continuación?

En el siguiente capítulo se instalarán en el clúster algunas bibliotecas para alcanzar un mejor rendimiento gracias al aprovechamiento de los recursos disponibles en el clúster. Para ello, se utilizarán OpenMP o MPI para paralelizar los códigos a nivel de nodo y a nivel de clúster.

1 https://www.sololinux.es/comandos-de-nano

2 Puede encontrarse información detallada sobre la implementación concreta que se empleará en el libro en http://nfs.sourceforge.net/

5. Modelos de programación paralela

Para ejecutar adecuadamente aplicaciones científicas y así obtener un buen rendimiento, es necesario aprovechar todos los recursos computacionales de los que se dispone de la mejor forma posible. Esto quiere decir que, tanto a nivel de nodo como de conjunto de ellos (clúster), se debe tratar de alcanzar el máximo aprovechamiento de los recursos disponibles; en otras palabras, la optimización del uso de los recursos.

En este capítulo se introduce el software que se utilizará con el objetivo de ejecutar las aplicaciones, optimizarlas y monitorizarlas en el sistema. La primera mitad del capítulo (Apartado 5.1) presenta el compilador GCC, mientras que la segunda mitad (Apartado 5.2) explica el modelo de programación paralela OpenMP, enfocado a aprovechar los recursos disponibles a nivel de nodo. Esta segunda parte también incluye la descripción de MPI, la interfaz para el paso de mensajes que permite comunicar distintos nodos.

5.1 Compilador GCC

En general, un compilador es un programa que se encarga de traducir un código escrito en un lenguaje de alto nivel (como C o Fortran, entre otros) a lenguaje máquina, es decir, a un código directamente interpretable por el microprocesador. La Figura ?? muestra los distintos tipos de códigos para un computador. El código desarrollado por el usuario estará escrito habitualmente en un lenguaje de alto nivel, en otras palabras, un lenguaje de programación más cercano al lenguaje humano. En algunas ocasiones, el usuario utilizará un lenguaje de bajo nivel, más cercano al "idioma" que entiende el computador (este es el lenguaje ensamblador). La tarea del compilador será traducir el código de alto nivel a lenguaje ensamblador. A continuación, el ensamblador generará el código máquina, es decir, una secuencia de ceros y unos ejecutada por el procesador. Dependiendo del lenguaje de alto nivel a compilar, se necesita un compilador u otro para realizar la traducción. En el caso de la compilación de lenguaje C, una de las herramientas más utilizadas es la colección de compiladores GNU (*GNU Compiler Collection* [GCC], en inglés), que es capaz de compilar cualquier programa escrito en C, C++ o Fortran. GCC, además de compilar el código, realiza la tarea de ensamblado.

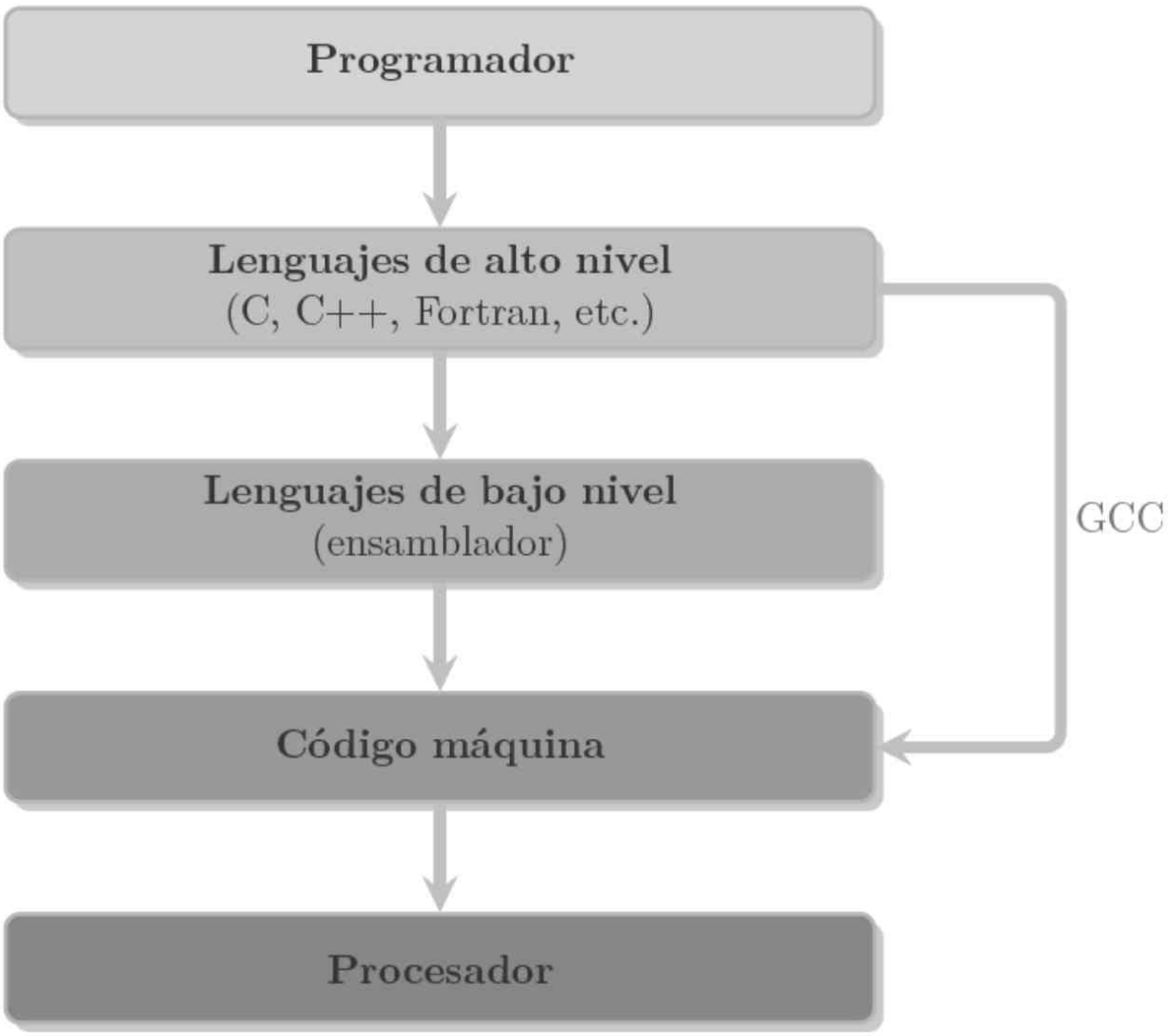


Figura 5.1 Esquema de los distintos tipos de códigos que podemos encontrar en un computador.

Por lo tanto, para poder ejecutar un programa escrito por el usuario será necesario:

- Escribir el programa deseado por el usuario incluyendo todas la bibliotecas necesarias.
- Compilar el programa con éxito (sin errores). Esto implica:
  1. Traducir todas las partes del código escrito por el usuario (programas y subprogramas) a código de bajo nivel.
  2. Enlazar los códigos de bajo nivel generados con las funciones que

se utilizan de las bibliotecas incluidas para generar un módulo ejecutable escrito en código máquina.

- Lanzar el ejecutable generado en la compilación.

Las siguientes secciones muestran los comandos a ejecutar para llevar a cabo los pasos anteriores. Hemos usado el típico programa de ejemplo *hola_mundo*, cuya única finalidad es mostrar por pantalla el mensaje "Hola mundo!".

5.1.1 Ejemplo hola_mundo en serie

A continuación, se muestra el proceso completo de compilación para el programa *hola_mundo.c* en serie, es decir, utilizando un único hilo de ejecución, sin paralelizar el programa.

En el siguiente cuadro se muestra el código que debe escribir el usuario.

Fichero: hola_mundo.c

```
#include <stdio.h>
int main(void)
{
    printf("Hola mundo!\n");
    return 0;
}
```

A continuación, se describe cada línea de código:

- La cláusula #include permite incluir ficheros de cabeceras en los que están definidas determinadas funciones. En este caso es necesario incluir *stdio.h* porque es donde está definida la función printf (), que será utilizada para mostrar por pantalla el mensaje deseado. Es importante incluir este fichero porque, de lo contrario, el compilador nos devolverá un error al no conocer la función que se intenta utilizar.
- Con la línea int main(void) se define la función principal. La ejecución

de un programa escrito en C siempre empieza por la función principal. Además, el tipo de retorno de la función es int . Esto significa que, en este caso, una vez termine la función main se debe devolver un valor entero. Este valor suele ser o si no se ha producido ningún error o distinto de o en caso contrario.

■La función printf () es la primera que se incluye en el cuerpo de la función principal. Como se muestra en el ejemplo, esta recibe como parámetro la cadena de texto que se va a mostrar por pantalla y transmite la orden de que se muestre. El carácter \n sirve para insertar un salto de línea tras el texto cuando se muestre por pantalla.

■Sentencia return . Es lo último que se ejecutará en el programa y se utiliza para devolver el valor de la función. Como se ha apuntado anteriormente, por convención se devuelve o si la ejecución ha sido correcta.

Una vez escrito el programa, será necesario compilarlo y enlazarlo. Para ello, se ejecutará la siguiente instrucción:

```
pi@nodoo:~ $ gcc hola_mundo.c -o hola_mundo
```

Como se muestra en la consola, se utiliza GCC como compilador, al cual se le indica que el código a compilar se encuentra en el fichero de texto *hola_mundo.c*. Con la opción "-o" se combinan en un solo paso la compilación y el enlazado para producir el fichero ejecutable de nombre hola_mundo. En realidad, el uso del compilador es mucho más sofisticado y existen detalles en las distintas etapas de compilación que quedan fuera del alcance de este libro.

Una vez compilado el código (sin errores) será posible su ejecución. En el caso de que se produzcan errores de compilación, será necesario corregirlos siguiendo las pistas proporcionadas por el compilador (mensajes de error) y

volver a compilar el código hasta que se solucionen por completo los errores. Este proceso podría requerir varios ciclos de corrección de errores y compilación.

La ejecución del código generado por el compilador y su salida son:

```
pi@nodoo:~ $ ./hola_mundo
Hola mundo!
```

Tal como se observa en el cuadro anterior, un programa ya compilado (y que se encuentra en el directorio actual) se ejecuta simplemente precediendo con ./ al nombre del ejecutable.

5.1.2 Un paso más allá en la compilación

El ejemplo *hola_mundo* visto hasta el momento ha permitido mostrar la secuencia de pasos a seguir desde la escritura de un programa hasta su ejecución y las instrucciones necesarias para ello. Sin embargo, en un contexto real, la complejidad de cualquier programa o aplicación desarrollada es notablemente mayor. Habitualmente una aplicación real se compondrá de varios ficheros (decenas o cientos) que contienen programas y subprogramas. Además, muy probablemente, será necesario recurrir no solo a las bibliotecas del compilador, sino a bibliotecas software externas que permitan hacer algunas operaciones requeridas por el programa y que ya han sido implementadas y optimizadas con anterioridad en otros paquetes software. Todo esto hace que la compilación del programa requiera distintos pasos (y opciones del compilador) que no se han mostrado, puesto que quedan fuera del alcance de este libro. Por tanto, ¿cómo podrán compilarse las aplicaciones que se proponen más adelante? Esto se hará mediante un fichero *Makefile*.

Makefile es un tipo de fichero que recoge todas las instrucciones necesarias y comandos de compilación con sus correspondientes opciones para construir el programa ejecutable a partir de todos los ficheros fuente escritos por el programador y las bibliotecas software necesarias. De esta forma, cuando se hace pública una determinada aplicación (*release*) que debe ser compilada para su ejecución, se proporciona el *Makefile* que permite llevar a cabo este proceso. Téngase en cuenta que el fichero *Makefile* es un archivo que no tiene ningún efecto por sí solo; para que las instrucciones que contiene den lugar al ejecutable deseado es necesario utilizar la herramienta make. A continuación, se muestra la forma habitual de utilizar esta herramienta.

Asumiendo que el directorio actual contiene el *Makefile* que permite compilar un determinado programa deseado, los dos comandos a utilizar para dicho fin son:

```
pi@nodoo:~ $ make clean
pi@nodoo:~ $ make
```

Habitualmente, la forma de proceder al compilar con make suele ser en dos pasos, aunque el primero puede ser omitido:

1. Eliminar todos los ficheros generados en compilaciones previas por make , tanto intermedios como el ejecutable final. Los ficheros a eliminar vienen indicados en el fichero *Makefile* .
2. Compilar el programa objetivo para generar su ejecutable. Si se ha ejecutado el paso anterior, la ejecución se realiza desde cero; si no se ejecuta el paso previo, solo se recompilarán los ficheros que hayan sufrido cambios. En el segundo supuesto pueden darse algunas inconsistencias con las partes del programa compiladas previamente, aunque el proceso es más rápido.

Finalmente, relacionada con la herramienta make encontramos, entre otras muchas, la opción "-j". Esta permite la compilación paralela para un mejor aprovechamiento de los recursos del sistema (utilizando más de un núcleo). Para ello, basta con indicar el número de hilos de ejecución que se quieren emplear en la compilación, tal como se muestra a continuación. Este número debería estar comprendido entre dos y el máximo número de núcleos disponibles en el sistema.

Nótese que queda fuera del alcance de esta obra explicar el desarrollo de ficheros *Makefile*.

```
pi@nodoo:~ $ make -j 4
```

5.2 Modelos de programación paralela

La programación paralela se basa en aprovechar distintos recursos hardware que colaboran conjuntamente para, así, dotar de mayor potencia al sistema HPC. En este apartado se presentan los modelos de programación paralela OpenMP y MPI, los cuales permiten explotar el paralelismo a nivel de nodo y a nivel de clúster, respectivamente.

5.2.1 Intra-nodo: OpenMP

Para lograr optimizar el uso de los recursos disponibles en un nodo se pueden emplear distintas metodologías, como, por ejemplo, el modelo OpenMP[1] que presentaremos en este manual. OpenMP permite "poner a trabajar" todos los *cores* o núcleos disponibles dentro de un nodo. La Figura 5.2 muestra gráficamente la funcionalidad de OpenMP; en ella, puede observarse que los núcleos de un nodo RPi pueden activarse mediante el uso de este modelo de programación.

Utilizar OpenMP es muy sencillo: simplemente es necesario incluir la cabecera #include <omp.h> y las directivas #pragma en el código. Estas indican al compilador cómo debe ejecutarse (a nivel de núcleo)

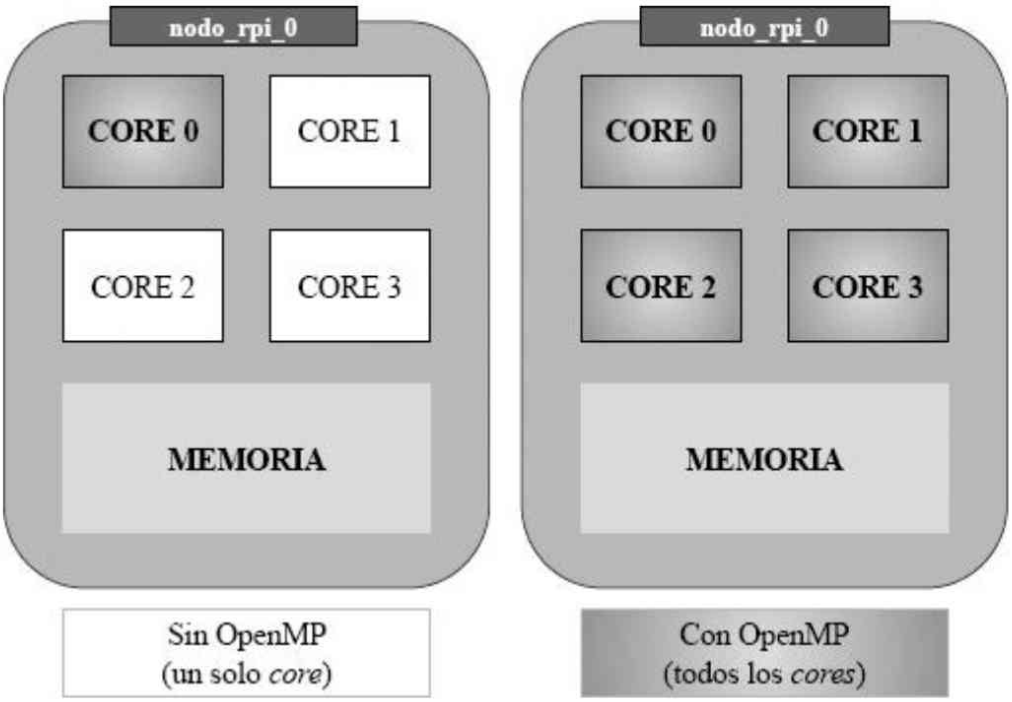


Figura 5.2 Esquema del uso de los *cores* de un nodo RPi sin emplear (izquierda) y empleando (derecha) OpenMP. Puede apreciarse que, gracias al uso de este modelo de programación, se consiguen activar todos los núcleos disponibles, que comparten memoria en el nodo al que pertenecen.

una instrucción o un conjunto de las mismas. Algunos ejemplos de directivas son:

- #pragma omp parallel
- #pragma omp parallel for

La primera permite ejecutar una instrucción (o un conjunto de ellas abarcado entre llaves) en paralelo, activando para ello todos los núcleos disponibles en el sistema; la segunda realiza las iteraciones de un bucle for en paralelo, asignando cada una de ellas (por bloques, una a una, de manera dinámica, etc.) a un núcleo de entre todos los disponibles en el sistema.

Fichero: ejemplo de `hola_mundo.c` incluyendo OpenMP

```
#include <stdio.h>
#include <omp.h>
int main(void)
{
    #pragma omp parallel
    printf("Hola mundo, soy el core %d.\n", ↩
        omp_get_thread_num());
    return 0;
}
```

Una vez incluidas todas las directivas OpenMP necesarias en el código, este debe compilarse usando la opción "-fopenmp". En el caso de no incluir esta opción, el compilador interpretará los pragma como meros comentarios. Por tanto, si se utiliza el compilador GCC, la compilación del código *hola_mundo.c* con OpenMP puede hacerse como sigue:

```
pi@nodoo:~ $ gcc -fopenmp hola_mundo.c -o hola
```

Para la ejecución del programa hola bastará con indicar el número de hilos que se desea utilizar a través de la variable de entorno OMP_NUM_THREADS y ejecutar el programa de este modo:

```
pi@nodoo:~ $ export OMP_NUM_THREADS=4
pi@nodoo:~ $ ./hola
Hola mundo, soy el core 0.
Hola mundo, soy el core 3.
Hola mundo, soy el core 1.
Hola mundo, soy el core 2.
```

La salida del programa ejecutado será similar a la mostrada, aunque es importante notar que el orden en el que los hilos se ejecutan no se controla de ningún modo. Por lo tanto la secuencia en la que se muestran los mensajes puede variar de ejecución a ejecución.

En caso de dudas sobre la diferencia entre los conceptos *proceso* y *núcleo*, se recomienda revisar el Apartado 1.9.

5.2.2 Inter-nodo: MPI

Más allá del uso de OpenMP que permite la colaboración de distintos hilos dentro de un único nodo, es necesario que dentro de un clúster los distintos nodos puedan colaborar entre ellos en caso de necesitar una ejecución distribuida. Coordinar los distintos nodos del sistema exige, por tanto, permitir la comunicación entre ellos. Esto se realiza mediante la interfaz de paso de mensajes (*Message Passing Interface* [MPI], en inglés). El estándar MPI2 define la sintaxis y semántica de las funciones. Existen distintas implementaciones de MPI, las cuales permiten establecer comunicaciones entre los nodos del sistema como muestra la Figura 5.3.

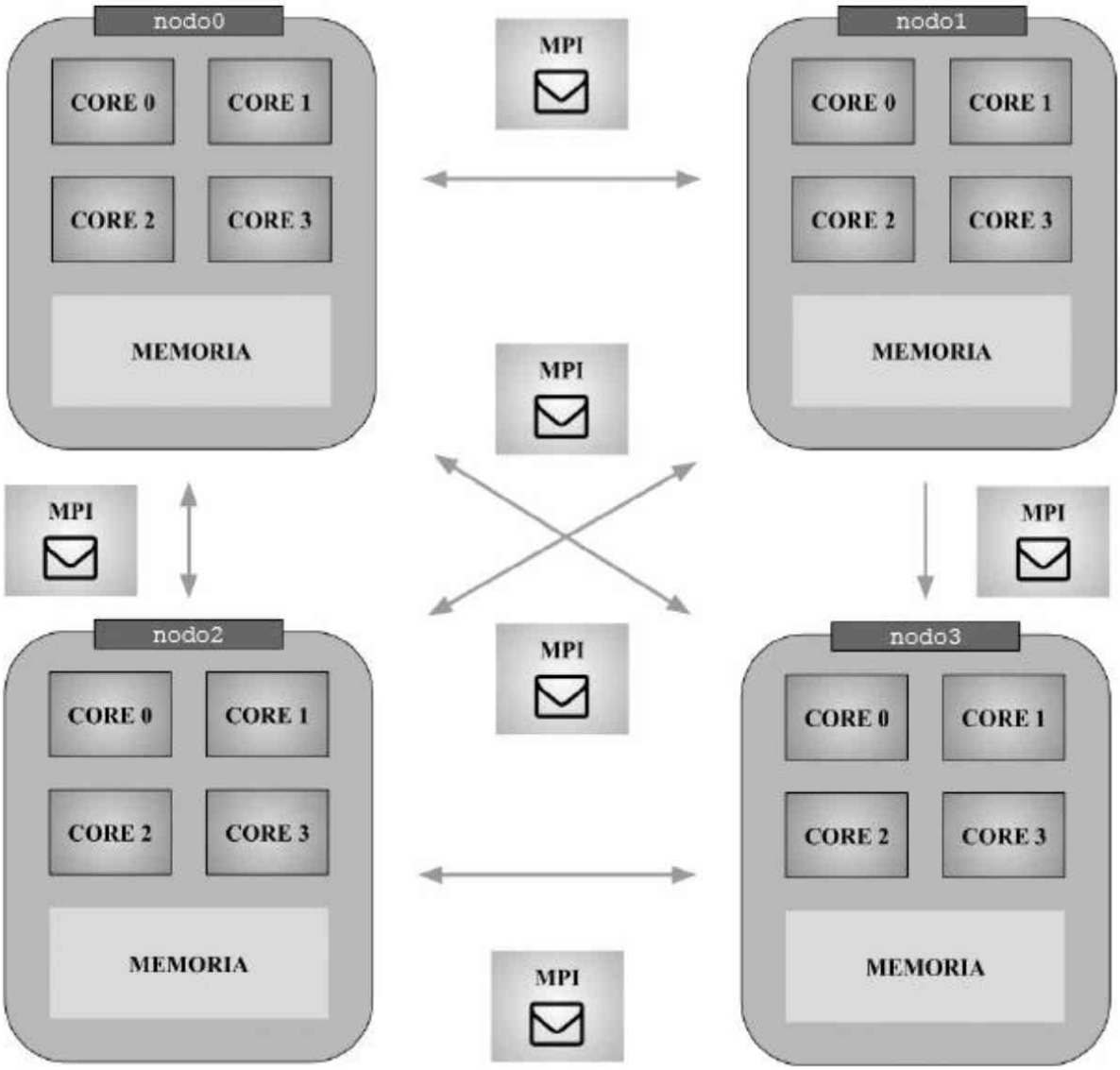


Figura 5.3 Esquema de las comunicaciones que permite el uso de MPI en un clúster de cuatro nodos RPi. Puede apreciarse que gracias al uso de esta interfaz para el paso de mensajes se consigue comunicar núcleos de distintos nodos: cada uno accede a su memoria pero puede transferir información al resto.

Es importante señalar que, mientras en OpenMP se hablaba de hilos, en el caso de MPI se hace referencia a procesos. Es decir, cada entidad de ejecución dispondrá de su propio espacio de memoria, independientemente de que se trate de procesos en el mismo o distintos nodos.

La programación con MPI utiliza funciones específicas que permiten comunicar y sincronizar los procesos. Siguiendo con el ejemplo sencillo de *hola_mundo.c* propuesto en el apartado anterior, en el caso de MPI será necesario incluir la cabecera #include <mpi.h> y algunas funciones que pueden verse en el cuadro que presenta el ejemplo. Además, en MPI se introducen los conceptos de comunicador y rango. Un comunicador es un conjunto de procesos que pueden comunicarse entre sí y el rango de un proceso es el identificador numérico que representa a ese proceso dentro del comunicador al que pertenece.

Fichero: hola_mundo_mpi.c

```
#include <stdio.h>
#include <mpi.h>

int main(int argc, char** argv)
{

    // Inicializar el entorno MPI
    MPI_Init(NULL, NULL);

    // Obtener el número de procesos
    int talla_mundo;
    MPI_Comm_size(MPI_COMM_WORLD, &talla_mundo);

    // Obtener el rango del procesador
    int rango_en_mundo;
    MPI_Comm_rank(MPI_COMM_WORLD, &rango_en_mundo);

    // Obtener el nombre del procesador
    char nombre_procesador[MPI_MAX_PROCESSOR_NAME];
    int talla_nombre;
    MPI_Get_processor_name(nombre_procesador, &↩
        talla_nombre);

    // Mostrar saludo
    printf("Hola mundo desde %s, con rango %d de un total↩
         de %d procesadores.\n", nombre_procesador, ↩
        rango_en_mundo, talla_mundo);

    // Finalizar el entorno MPI
    MPI_Finalize();
}
```

El programa anterior tiene la misma finalidad que el *hola_mundo.c* implementado en OpenMP, con la particularidad de que ahora los procesos (en vez de hilos) pueden estar en el mismo o en distintos nodos, según lo especifique el usuario. En este libro no se va a profundizar en MPI, pero es interesante destacar que, por defecto, los procesos creados se asignan al comunicador MPI_COMM_WORLD.

Instalación de MPI

Puesto que existen distintas implementaciones MPI, en este libro se propone la instalación de dos de las implementaciones más clásicas: OpenMPI[3] y MPICH[4]. Los pasos a seguir, independientemente de la biblioteca MPI instalada, son similares: descargar los códigos fuente de la biblioteca, instalar la biblioteca compilada en el directorio compartido para que sea accesible desde todos los nodos y añadir a la variable de entorno PATH la nueva instalación. A continuación se muestran los detalles y los pasos a seguir para descargar OpenMPI (versión 4.0.2) e instalarlo en el sistema:

```
COMENTARIO: Descargar OpenMPI
pi@nodo0:~ $ wget https://download.open-mpi.org/release/open-mpi/v4.0/
    openmpi-4.0.2.tar.gz

COMENTARIO: Descomprimir el directorio, compilar e instalar OpenMPI
pi@nodo0:~ $ tar -xvf openmpi-4.0.2.tar.gz
pi@nodo0:~ $ rm openmpi-4.0.2.tar.gz
pi@nodo0:~ $ cd openmpi-4.0.2
pi@nodo0:~/openmpi-4.0.2
    $ ./configure --prefix=/SHARED/openmpi-4.0.2-install
    --enable-mpirun-prefix-by-default
pi@nodo0:~/openmpi-4.0.2 $ make -j 4
pi@nodo0:~/openmpi-4.0.2 $ make install
pi@nodo0:~/openmpi-4.0.2 $ cd
```

Finalmente, se añadirá a la variable de entorno PATH la ruta donde se encuentran los comandos de OpenMPI instalados:

```
COMENTARIO: Incluir la ruta de instalación de OpenMPI en el PATH
pi@nodo0:~ $ echo 'export
    PATH=/SHARED/openmpi-4.0.2-install/bin:$PATH' >> .bashrc
pi@nodo0:~ $ source .bashrc
```

Los pasos para descargar MPICH (versión 3.3.2) e instalarlo en el clúster

son:

```
COMENTARIO: Descargar MPICH
pi@nodo0:~ $ wget
     http://www.mpich.org/static/downloads/3.3.2/mpich-3.3.2.tar.gz

COMENTARIO: Descomprimir, compilar e instalar MPICH
pi@nodo0:~ $  tar -xvf mpich-3.3.2.tar.gz
pi@nodo0:~ $  rm mpich-3.3.2.tar.gz
pi@nodo0:~ $  cd mpich-3.3.2
pi@nodo0:~/mpich-3.3.2
     $ ./configure --prefix=/SHARED/mpich-3.3.2-install --disable-f77
     --disable-fc  --disable-fortran
pi@nodo0:~/mpich-3.3.2 $ make -j 4
pi@nodo0:~/mpich-3.3.2 $ make install
pi@nodo0:~/mpich-3.3.2 $ cd
```

De nuevo, se añadirá a la variable de entorno PATH la ruta donde se encuentran los comandos de MPICH instalados:

```
pi@nodoo:~ $ echo
     'export PATH=/SHARED/mpich-3.3.2-install/bin:$PATH' >> .bashrc
pi@nodoo:~ $ source .bashrc
```

Puede comprobarse qué biblioteca MPI se está utilizando para la ejecución paralela mediante la instrucción siguiente:

```
pi@nodoo:~ $ which mpirun
/SHARED/mpich-3.3.2-install/bin/mpirun
```

Los detalles sobre los comandos utilizados en cada caso se verán en el Apartado 5.2.4.

En el caso de tener varias bibliotecas de MPI instaladas en el sistema, como es el caso descrito en este libro, hay que tener en cuenta que la versión utilizada es la correspondiente a la primera aparición, de izquierda a derecha, en la variable PATH.

Se puede mostrar el orden de las rutas añadidas a PATH con echo $PATH y comprobar la versión, en este caso de MPI, que se está utilizando mediante el comando which mpirun. Si se han seguido los pasos de instalación tal y como se han descrito en este capítulo, la biblioteca de MPI utilizada será MPICH, ya que ha sido la última ruta añadida al principio de PATH.

5.2.3 Compilación con MPI

Para compilar un programa MPI, es necesario tener instalada una biblioteca MPI en el sistema. Independientemente de si la biblioteca instalada es MPICH u OpenMPI, la compilación del programa se hará mediante:

```
pi@nodoo:~ $ mpicc hola_mundo_mpi.c -o hola_mundo_mpi
```

5.2.4 Ejecución con MPI

Tras la compilación del programa MPI, la ejecución del mismo dependerá de la biblioteca instalada. Como pauta general, para la ejecución del programa compilado será necesario especificar el número de procesos que se van a utilizar mediante la opción correspondiente. Además, en caso de querer ejecutar procesos en distintos nodos, será necesario indicar el nombre de los mismos en el comando de ejecución, así como disponer del programa compilado en todos los nodos.

A continuación se muestran dos ejemplos de ejecución con cuatro

procesos cada uno; en el primero se ejecutan todos los procesos en el mismo nodo y en el segundo cada proceso se ejecutará en un nodo distinto:

```
COMENTARIO:
    Copiar el fichero hola_mundo_mpi en el directorio compartido
pi@nodo0:~ $ cp hola_mundo_mpi /SHARED
pi@nodo0:~ $ cd /SHARED

pi@nodo0:/SHARED $ mpirun -n 4 ./hola_mundo_mpi
Hola mundo desde nodo0, con rango 3 de un total de 4 procesadores.
Hola mundo desde nodo0, con rango 1 de un total de 4 procesadores.
Hola mundo desde nodo0, con rango 2 de un total de 4 procesadores.
Hola mundo desde nodo0, con rango 0 de un total de 4 procesadores.

pi@nodo0:/SHARED
    $ mpirun -n 4 -host nodo0,nodo1,nodo2,nodo3 ./hola_mundo_mpi
Hola mundo desde nodo1, con rango 1 de un total de 4 procesadores.
Hola mundo desde nodo0, con rango 0 de un total de 4 procesadores.
Hola mundo desde nodo3, con rango 3 de un total de 4 procesadores.
Hola mundo desde nodo2, con rango 2 de un total de 4 procesadores.
```

Hay que recordar que, dado que no se han sincronizado los procesos con ningún mecanismo, los mensajes mostrados con printf() pueden aparecer en cualquier orden y cambiar de una ejecución a otra.

5.3 Para profundizar...

1. Aunque la interfaz de paso de mensajes se presenta como un sistema de comunicación inter-nodo, se puede utilizar para comunicar procesos dentro de un mismo nodo, de modo similar a como se haría con OpenMP. Teniendo en cuenta las diferencias entre procesos e hilos, ¿que método cree que será más rápido para lograr el mismo intercambio de información?

Puede encontrar información en el Apartado 1.9.

2. ¿Sería posible combinar OpenMP y MPI para que dentro de cada proceso MPI se crearan varios hilos?

En el Apartado 7.2.1 se puede encontrar un ejemplo.

5.4 Material recomendado

Los materiales propuestos en este apartado permiten profundizar en el contenido visto en este capítulo. En [1] se muestra una breve pero detallada descripción del compilador GCC y sus opciones, así como de las etapas de compilación. Por su parte, [2] es un libro (en inglés) que recoge todos los detalles del lenguaje de programación C. En cuanto a los modelos de programación vistos, se proponen una serie de tutoriales tanto para MPI [3] como para OpenMP [4], en formato vídeo en el segundo caso.

[1] Víctor A. González Barbone. El Compilador GCC. https://iie.fing.edu.uy/~vago-nbar/gcc-make/c.htm . Accedido por última vez: nov. 2020.

[2] Declan Brady Mike Banahan and Mark Doran. The C Book (en inglés). https://publications.gbdirect.co.uk//c_book/ . Accedido por última vez: nov. 2020.

[3] MPI Tutorial. MPI Tutorials (en inglés). https://mpitutorial.com/tuto-rials/ . Accedido por última vez: nov. 2020.

[4] OpenMP. Presenter: Tim Mattson (Intel Corp.). A "Hands-on" Introduction to OpenMP (en inglés). https://www.youtube.com/watch?-v=nE-Bf8XIBf8XI . Accedido por última vez: nov. 2020.

5.5 Resumen

En este capítulo se ha descrito brevemente la función del compilador y se ha mostrado su funcionamiento al compilar programas secuenciales y paralelos. Además, se han presentado dos modelos de programación paralelos, OpenMP y MPI, los cuales son ampliamente usados en la HPC para aumentar el rendimiento de las aplicaciones susceptibles de ser paralelizadas. Por tanto, se ha aprendido que:

- Para poder ejecutar un programa escrito en C es indispensable compilarlo.
- Existen distintos modelos de programación que permiten aumentar el rendimiento de ciertas aplicaciones gracias a la paralelización del código.
- La paralelización del código se puede hacer a distintos niveles. Por ejemplo, se ha visto que OpenMP permite la paralelización intranodo, mientras que el estándar MPI está enfocado a paralelizar códigos para que sean ejecutados entre distintos nodos.
- Cada modelo de programación tiene sus peculiaridades no solo a nivel del ámbito en el que se ejecuta, sino también de sintaxis, variables a tener en cuenta, forma de compilación y ejecución.

5.5.1 ¿Qué viene a continuación?

En el siguiente capítulo se abordará la utilización de los recursos hardware de un clúster en el escenario más habitual, es decir, en un entorno multiusuario en el que los recursos son asignados temporalmente por un software específico. En concreto, se presentará la herramienta Slurm para la gestión de trabajos y recursos en el clúster.

1 https://www.openmp.org/

2 https://www.mpi-forum.org

3 https://www.open-mpi.org

4 https://www.mpich.org/

6. Gestor de trabajos y recursos

Los centros de supercomputación suelen contar con una gran cantidad de recursos hardware aglutinados en uno o varios clústeres. A su vez, estos recursos pueden ser utilizados por una gran cantidad de usuarios (podrían llegar a contarse por cientos o miles). De hecho, en entornos de producción sería extraño encontrar un supercomputador con únicamente un usuario. Por lo tanto, los usuarios no tienen libre acceso a los recursos, sino que los recursos computacionales se asignan temporalmente. Esto supone que los usuarios tienen que compartir y competir por los recursos. Cuanto mayor es el grado de utilización del sistema, más difícil es el acceso a estos recursos, o lo que es lo mismo, más tiempo tienen que esperar los usuarios para que se les conceda su petición. Para llevar a cabo una buena administración de los recursos es necesario un software de gestión de la carga de trabajo (*Resource Manager System* [RMS], en inglés), al que también se le puede denominar como planificador de trabajos, gestor de recursos o gestor de colas.

6.1 Visión general

Las peticiones de recursos durante un periodo de tiempo se realizan a través de "trabajos" que cada usuario envía al gestor de recursos para que este los administre. Estos "trabajos" no son otra cosa que instancias de las aplicaciones de usuario con una serie de requisitos para ser ejecutados. Cada usuario puede solicitar un número distinto de recursos (en infraestructuras heterogéneas también puede elegirse el tipo de recurso) durante un tiempo determinado para sus diversos trabajos. Por ejemplo, un usuario podría solicitar cuatro nodos durante 10 segundos para ejecutar el comando hostname. En este ejemplo, el usuario enviaría su trabajo a la cola y, una vez se le asignaran los recursos, ejecutaría el comando hostname en cuatro nodos distintos. Particularmente, el comando hostname no necesita mucho tiempo para su ejecución; sin embargo, si el usuario quisiera ejecutar la orden sleep 20 (cuya duración es de 20 segundos) habiendo hecho la misma solicitud de nodos y segundos que en el caso anterior, el gestor abortaría el trabajo y revocaría la asignación de recursos transcurridos 10 segundos, sin llegar a completar el trabajo. El ciclo de vida básico de un trabajo (es decir, los estados por los que transcurre desde que se crea hasta que concluye) se muestra en la Figura 6.1. En ella, se puede apreciar cómo el ciclo empieza en el estado "pendiente", en el que el trabajo permanecerá hasta que el planificador lo "ejecute". Tras la ejecución, el trabajo hará la transición a "completado" si todo ha ido bien. Sin embargo, el trabajo podría terminar en "cancelado" si el usuario o el administrador aborta el trabajo, o bien en "fallido", por ejemplo, en caso de falta de disponibilidad de recursos o un error durante la ejecución.

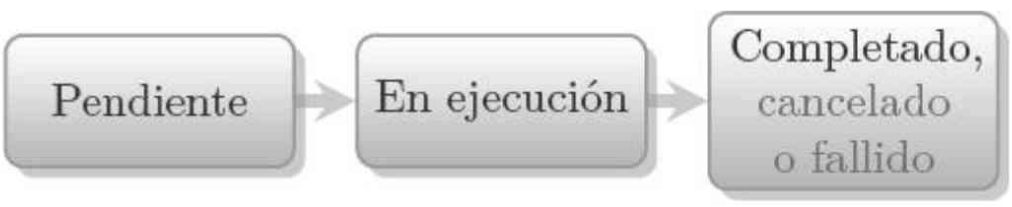


Figura 6.1 Esquema del ciclo de vida de un trabajo que representa las transiciones básicas entre sus estados.

6.1.1 Introducción a Slurm Workload Manager

Concretamente, en este libro se va a utilizar Slurm[1], ya que, además de ser un software de código abierto, es uno de los gestores de carga de trabajos más extendido en el mundo de la supercomputación. Slurm se compone de una serie de procesos no interactivos (demonios) que se ejecutan en cada uno de los nodos del clúster. Concretamente, en los nodos de cómputo se ejecutan los demonios llamados slurmd, mientras que en el nodo controlador (o *front-end*) se ejecuta el demonio slurmctld. Si el *front-end* actúa además como nodo de cómputo, también se ejecuta slurmd. Las Figura 6.2 ilustra qué demonios de Slurm ejecutará cada nodo del clúster diseñado en este libro, en función de su cometido, así como un esquema de la información que maneja cada demonio de Slurm en su nodo.

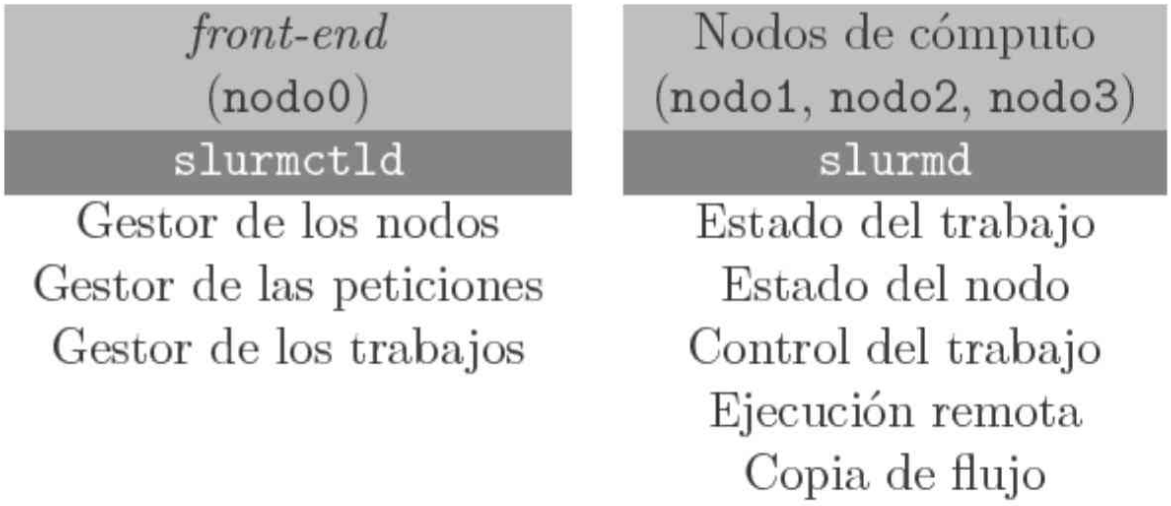


Figura 6.2 Esquema de la información que maneja cada demonio de Slurm en su nodo.

Slurm permite organizar los nodos de cómputo en "particiones", también denominadas "colas", aunque todos los nodos se añaden por defecto a la misma partición o cola. Por su parte, los usuarios interactúan con Slurm a través del nodo controlador (habitualmente el *front-end*) mediante una serie de comandos. A modo ilustrativo, la Tabla 6.1 describe y ejemplifica algunos de los comandos más habituales y que se han utilizado en esta obra. La lista completa de comandos y sus opciones se puede consultar en la documentación oficial de Slurm[2].

6.1.2 Envío de trabajos

Es importante destacar que un trabajo en Slurm (*job*) consiste en uno o varios *steps* que, a su vez, pueden contener una o varias tareas (*task*). Aunque es habitual utilizar el término "trabajo" (*job*) para cualquier programa enviado a la cola, existen diferencias según el tipo de trabajo; en concreto, el *job* se crea mediante el comando sbatch, mientras que un *step* se crea mediante el comando srun. Finalmente, las tareas (*task*) se pueden equiparar a procesos (por ejemplo, rangos MPI) y son definidas con el parámetro "--ntasks" en los comandos de envío.

Por ejemplo, para solicitar ocho procesadores en dos nodos (cuatro en cada nodo) en un trabajo, se podría utilizar el siguiente código, que, además, lanza *steps* con distintas necesidades:

Fichero: /home/pi/job.sbatch

```
#!/bin / bash
#SBATCH --nodes 2
#SBATCH --tasks–per–node 4
#step 0, utiliza 8 procesadores
```

```
srun --ntasks=8 hostname
#step 1, utiliza 4 procesadores
srun --ntasks=4 hostname
```

Se puede enviar el trabajo a la cola o cancelarlo tal como muestran, respectivamente, los comandos de la Consola 6.4. Al enviarlo, Slurm asigna al trabajo enviado un identificador (valor numérico), 117889 en el ejemplo. Por defecto, Slurm utilizará este identificador para que la salida del trabajo se escriba en los ficheros *slurm-117889_0.out* y *slurm-117889_1.out*, cada uno correspondiente a su *step*.

Finalidad	**Comando**	**Ejemplo**
Enviar trabajo a una cola (ejecución bloqueante e interactiva)	srun	Consola 6.7
Enviar trabajo a una cola (ejecución no bloqueante y no interactiva)	sbatch	Consola 6.4
Cancelar trabajos	scancel	Consola 6.4
Mostrar el estado de los trabajos en cola	squeue	Consola 6.1
Mostrar información de los trabajos	sacct	Consola 6.5
Mostrar el estado de los recursos	sinfo	Consola 6.2
Administrar la configuración de Slurm	scontrol	Consola 6.3

Tabla 6.1 Tabla que recoge los comandos de Slurm utilizados en esta obra.

```
pi@nodo0:~ $ squeue
JOBID  PART    NAME    USER    ST  TIME   NODES  LIST(REASON)
23     inter   myjob   rocío   R   3:37   1      nodo3
24     main    lammps  sandra  R   0:32   2      nodo[2,4]
35     main    lammps  sandra  PD  0:00   3      (Resources)
37     inter   myjob   rocío   PD  0:00   4      (Priority)
40     debug   test    sergio  PD  0:00   2      (Priority)
```

Consola 6.1 Ejemplo de la salida del comando squeue, el cual muestra información de los trabajos activos en el sistema. Concretamente, la columna *PART* indica la partición en la que está cada trabajo, *ST* el estado del trabajo (*R*: en ejecución; *PD*: en espera), *LIST(REASON)* los recursos asignados a los trabajos en ejecución o, en su caso, porqué está en espera (*Resources*: no hay recursos disponibles para el trabajo; *Priority*: hay trabajos en espera con más prioridad).

```
pi@nodo0:~ $ sinfo
PART    AVAIL   TIMELIMIT   NODES   STATE   NODELIST
main    up      infinite    2       alloc   nodo[2-4]
main    up      infinite    1       idle    nodo1
inter   up      infinite    1       alloc   nodo3
```

Consola 6.2 Ejemplo de ejecución del comando sinfo, el cual muestra información de las particiones y el estado de los nodos en el sistema.

```
pi@nodo0:~ $
    scontrol update nodename=nodo1 state=down reason="Mantenimiento"
pi@nodo0:~ $ scontrol update nodename=nodo1 state=resume
```

Consola 6.3 Ejemplo de uso del comando scontrol para, respectivamente, deshabilitar el *nodo1* (por mantenimiento) y luego devolverlo a la actividad.

```
pi@nodo0:~ $ sbatch job.sbatch
Submitted batch job 117889
pi@nodo0:~ $ scancel 117889
```

Consola 6.4 Ejecución de los comandos sbatch (para enviar el trabajo *job.sbatch* a la cola) y scancel (para cancelarlo).

Mediante el comando sacct se puede ver la información de todos los trabajos. Particularmente, se puede utilizar el identificador de un trabajo específico para ver su información con la opción "-j". Por ejemplo, se puede consultar la información del trabajo anterior tal como se muestra en la Consola 6.5.

```
pi@nodo0:~ $ sacct -j117889 --format=JobID%15,ntasks
          JobID   NTasks
--------------- --------
         117889
   117889.batch        1
  117889.extern        2
       117889.0        8
       117889.1        4
```

Consola 6.5 Ejecución del comando sacct para ver la información de todos los trabajos.

En ella se puede observar que el trabajo 117889 se compone en realidad de cuatro *steps*:

- ".batch": el propio comando sbatch crea un *step* de una tarea.
- ".extern": *step* para trabajos multi-nodo, con el número de tareas igual al número de nodos reservados.
- Los dos *steps* restantes (.0 y .1), numerados en orden ascendente empezando en "0", corresponden a los srun del script.

6.1.3 Instalación de Slurm

Slurm (versión 20.02.4) depende de varios paquetes que fácilmente se pueden instalar en el *front-end* mediante:

```
pi@nodo0:~ $ sudo apt-get install -y autoconf libtool libgtk2.0-dev
    libglib2.0-dev libssl-dev
```

Además, Slurm también necesita un servicio de autenticación para crear y validar credenciales de usuario. Slurm recomienda MUNGE[3], que está diseñado específicamente para sistemas altamente escalables como lo son los supercomputadores. Así pues, antes de la instalación de Slurm, habrá que instalar MUNGE.

Instalación de MUNGE

MUNGE define un entorno aislado y seguro para que los trabajos de los usuarios puedan utilizar los recursos de los nodos remotos mediante sus propias credenciales.

MUNGE ejecuta un demonio en cada nodo del clúster, por lo que es necesario instalarlo en todos los nodos. Además, tanto Slurm como MUNGE necesitan que los relojes del sistema operativo, los usuarios y sus grupos estén sincronizados en todos los nodos del clúster. Teniendo todo esto en

cuenta, inicialmente se sincronizarán los relojes y se crearán los directorios de la instalación en cada nodo. Se utilizará la opción "-p" de mkdir para crear jerarquías de directorios. El siguiente *script* muestra cómo hacerlo mediante un bucle:

Fichero: /home/pi/prepara_munge_nodos_computo.sh

```
#!/bin/bash

nodelist="nodo1 nodo2 nodo3"
statedir=$HOME/munge-files

for node in $nodelist; do
    ssh $node "sudo date --set=$(date --iso-8601=seconds)"
    ssh $node "mkdir -p $statedir/var"
    ssh $node "mkdir -p $statedir/lib/munge"
    ssh $node "mkdir -p $statedir/log/munge"
    ssh $node "mkdir -p $statedir/run/munge"
done
```

Se puede ejecutar este fichero para que sincronice y cree en el directorio HOME de cada nodo definido en la variable nodelist los directorios necesarios para la instalación de MUNGE. También, del mismo modo se preparará el directorio de instalación en el *front-end*:

```
pi@nodoo:~ $ chmod u+x prepara_munge_nodos_computo.sh
pi@nodoo:~ $ ./prepara_munge_nodos_computo.sh
pi@nodoo:~ $ mkdir -p $HOME/munge-files/var
pi@nodoo:~ $ mkdir -p $HOME/munge-files/lib/munge
pi@nodoo:~ $ mkdir -p $HOME/munge-files/log/munge
pi@nodoo:~ $ mkdir -p $HOME/munge-files/run/munge
```

Por su parte, a continuación se muestra la secuencia de comandos a

seguir para descargar, compilar, instalar y configurar MUNGE (versión 0.5.14):

```
COMENTARIO: Variables
pi@nodo0:~ $ statedir=$HOME/munge-files
pi@nodo0:~ $ installdir=/SHARED/munge-install

COMENTARIO: Obtener el código
pi@nodo0:~ $
    wget https://github.com/dun/munge/archive/munge-0.5.14.tar.gz
pi@nodo0:~ $ tar -xvf munge-0.5.14.tar.gz
pi@nodo0:~ $ rm munge-0.5.14.tar.gz

COMENTARIO: Compilar
pi@nodo0:~ $ cd munge-munge-0.5.14
pi@nodo0:~/munge-munge-0.5.14 $ ./bootstrap
pi@nodo0:~/munge-munge-0.5.14
    $ ./configure --localstatedir=$statedir --prefix=$installdir
pi@nodo0:~/munge-munge-0.5.14 $ make
pi@nodo0:~/munge-munge-0.5.14 $ make install
pi@nodo0:~/munge-munge-0.5.14 $ cd

COMENTARIO: Configurar
pi@nodo0:~ $ chmod 700 $installdir/etc/munge
pi@nodo0:~ $  dd if=/dev/random bs=1 count=1024 >
    $installdir/etc/munge/munge.key
pi@nodo0:~ $  chmod 400 $installdir/etc/munge/munge.key
```

Tras completar los pasos anteriores, MUNGE queda instalado y listo para ser ejecutado. Después de este proceso, los ficheros de configuración se encontrarán en el directorio *$HOME/munge-files* local a cada RPi, mientras que los ejecutables y bibliotecas se instalarán en el directorio compartido */SHARED/munge-install*. Cabe añadir que, en el momento de configurar MUNGE, se crea una clave para las credenciales de MUNGE (*/SHARED/munge-install/etc/munge/munge.key*) y se cambian sus permisos para cumplir con los estándares de seguridad que MUNGE especifica, es decir, permitir solamente que el propietario y nadie más pueda leer la clave.

Normalmente, el control de los demonios en Linux se realiza mediante el

gestor de demonios systemd[4], el cual se encarga de gestionar los servicios en Linux. Para poder iniciar y detener los servicios (en este caso, MUNGE) es necesario generar un fichero en el directorio */usr/lib/systemd/user* que contenga la información necesaria para que systemd funcione correctamente. Una vez se ha generado dicho fichero, al que se le denomina *munge.service*, es posible iniciar, parar o consultar el estado del servicio MUNGE mediante el comando systemctl[5], tal como se describe en la Consola 6.6. Además, mediante el argumento "enable" de systemctl es posible indicar a systemd que se inicie el servicio indicado de manera automática cuando el nodo se encienda. Cada vez que se modifique un fichero del directorio */usr/lib/sys-temd/user/* es necesario recargar la configuración de systemd con el comando systemctl --user daemon-reload.

Dado que es necesario que *munge.service* se encuentre en todos los nodos (*front-end* y nodos de cómputo), el fichero siguiente será generado en el directorio compartido */SHARED/munge-install*.

Fichero: /SHARED/munge-install/munge.service

```
[Unit]
Description=MUNGE authentication service
Documentation=man : munged (8)
After=network . target
After=time–sync . target

[Service]
Type=forking
EnvironmentFile=–/SHARED /munge–install /etc/ default / munge
ExecStart=/SHARED /munge–install / sbin / munged $OPTIONS
PIDFile=/home /pi/munge–files /run/ munge / munged . pid
```

```
Restart=on–abort

[Install]
WantedBy=multi–user . target
```

Una vez generado el fichero anterior, es necesario que se encuentre en el directorio */usr/lib/systemd/user* en todos los nodos donde se iniciará MUNGE. Además, es necesario ejecutar el comando systemctl --user daemon-reload para recargar systemd. Para facilitar la ejecución de comandos en todos los nodos de cómputo, se recomienda la creación del siguiente *script*:

Fichero: /home/pi/ejecutar_en_nodos_computo.sh

```
#!/bin/ bash
nodelist="nodo1 nodo2 nodo3"
for node in $nodelist; do
    ssh $node " $@ "
done
```

Con el fin de facilitar la propagación de cambios en todos los nodos, se ofrece el *script ejecutar_en_nodos_computo.sh*. Al ejecutar este *script*, los comandos dados como argumento serán ejecutados en cada nodo de cómputo. La siguiente consola muestra cómo permitir la ejecución de *ejecutar_en_nodos_computo.sh*. Además, en ella se indica cómo añadir y reconfigurar el demonio *munge.service* en el *front-end* y cómo utilizar *ejecutar_en_nodos_computo.sh* para hacer lo mismo en todos los nodos de cómputo.

```
pi@nodo0:~ $ chmod u+x ejecutar_en_nodos_computo.sh

COMENTARIO: Ubicar y tranferir munge.service
pi@nodo0:~ $
    sudo cp /SHARED/munge-install/munge.service /usr/lib/systemd/user/
pi@nodo0:~ $ ./ejecutar_en_nodos_computo.sh sudo cp
    /SHARED/munge-install/munge.service /usr/lib/systemd/user/

COMENTARIO: Reconfigurar los servicios en todos los nodos
pi@nodo0:~ $ systemctl --user daemon-reload
pi@nodo0:~ $
    ./ejecutar_en_nodos_computo.sh systemctl --user daemon-reload
```

Finalmente, se inician los demonios en todos los nodos (incluido el nodo *front-end*):

```
pi@nodoo:~ $ systemctl --user start munge
pi@nodoo:~ $
    ./ejecutar_en_nodos_computo.sh systemctl --user start munge
```

```
COMENTARIO: Habilitar el servicio munge
pi@nodo0:~ $ systemctl --user enable munge
Created symlink
     /home/pi/.config/systemd/user/multi-user.target.wants/munge.service
      /usr/lib/systemd/user/munge.service.

COMENTARIO: Comprobar el estado del servicio munge
pi@nodo0:~ $ systemctl --user status munge
● munge.service - MUNGE authentication service
   Loaded: loaded (/usr/lib/systemd/user/munge.service; enabled;
     vendor preset: enabled)
   Active: active
     (running) since Mon 2020-11-16 00:34:53 CET; 42min ago
     Docs: man:munged(8)
 Main PID: 28451 (munged)
   CGroup: /user.slice/user-1000.slice/user@1000.service/munge.service
           28451 /SHARED/munge-install/sbin/munged
nov 16 00:34:53 nodo0 systemd[483]: Starting MUNGE authentication
     service...
nov 16 00:34:53 nodo0 systemd[483]: Started MUNGE authentication
     service.

COMENTARIO: Parar el servicio munge
pi@nodo0:~ $ systemctl --user stop munge
```

Consola 6.6 Órdenes para habilitar, parar y comprobar el estado de un servicio con el comando systemctl. En esta consola se utiliza el servicio MUNGE para ejemplificar los usos.

Para comprobar que MUNGE funciona correctamente, se puede ejecutar la siguiente orden para cada nodo de cómputo:

```
pi@nodo0:~ $ /SHARED/munge-install/bin/munge -n | ssh nodo1
    /SHARED/munge-install/bin/unmunge
STATUS:          Success (0)
ENCODE_HOST:     nodo0 (192.168.0.1)
ENCODE_TIME:     2020-10-02 17:10:11 +0200 (1601651411)
DECODE_TIME:     2020-10-02 17:10:10 +0200 (1601651410)
TTL:             300
CIPHER:          aes128 (4)
MAC:             sha256 (5)
ZIP:             none (0)
UID:             pi (1000)
GID:             pi (1000)
LENGTH:          0
```

En la información mostrada por dicho comando, si *STATUS* muestra el valor *success*, significa que la autenticación ha funcionado correctamente. En caso contrario, ha habido algún error. Es posible consultar el *log* del servicio MUNGE en el fichero */home/pi/mungefiles/log/munge/munged.log*.

Uno de los errores más habituales se debe a la desincronización de los relojes entre nodos, lo cual provocaría el siguiente error: Rewound credential (16). Para solucionarlo, deben sincronizarse los relojes como se hace en el fichero *prepara_munge_nodos_computo.sh* y reiniciar los demonios de MUNGE en todos los nodos.

Configuración de Slurm

Una vez MUNGE está correctamente instalado en los nodos del clúster, se procederá a la instalación y configuración de Slurm (versión 20.02.4). Este proceso comienza con su descarga y descompresión, tal como se muestra a continuación:

```
pi@nodoo:~ $
    wget https://download.schedmd.com/slurm/slurm-20.02.4.tar.bz2
```

```
pi@nodoo:~ $ tar -xvf slurm-20.02.4.tar.bz2
pi@nodoo:~ $ rm slurm-20.02.4.tar.bz2
```

Para la instalación en el directorio compartido del clúster, se deberá indicar la ruta de instalación de MUNGE y habilitar la compilación para sistemas de 32 bits con "--enable-deprecated", tal y como se muestra a continuación:

```
pi@nodoo:~ $ mungedir=/SHARED/munge-install
pi@nodoo:~ $ slurmdir=/SHARED/slurm-install
pi@nodoo:~ $ configdir=$slurmdir/confdir
pi@nodoo:~ $ mkdir -p $configdir
pi@nodoo:~ $ mkdir $slurmdir/var
pi@nodoo:~ $ cd slurm-20.02.4
pi@nodoo:~/slurm-20.02.4 $ ./configure --prefix=$slurmdir
    --sysconfdir=$configdir --with-munge=$mungedir --enable-
    deprecated
```

A continuación, se compila Slurm, se instala, y se añaden las rutas de los comandos y de los demonios compilados al PATH añadiendo la orden al fichero *.bashrc* para que cada vez que se inicie la sesión los ejecutables estén accesibles.

```
pi@nodoo:~/slurm-20.02.4 $ make -j 3
pi@nodoo:~/slurm-20.02.4 $ make install
pi@nodoo:~/slurm-20.02.4 $ cd
pi@nodoo:~ $ echo 'export
```

```
    PATH=/SHARED/slurm-install/bin:/
    SHA-RED/slurm-tall/sbin:$-PATH'
    ≫ .bashrc
pi@nodoo:~ $ source .bashrc
```

Una vez instalado Slurm en el directorio compartido, es momento de configurar Slurm para el clúster RPi. El siguiente cuadro muestra un ejemplo de configuración para la instalación descrita en este libro:

Fichero: /SHARED/slurm-install/confdir/slurm.conf

```
ClusterName=Supercomputer
ControlMachine=nodo0
SlurmUser=pi
SlurmdUser=pi
StateSaveLocation=/SHARED/slurm-install/var
SlurmdSpoolDir=/SHARED/slurm-install/var/slurmd.%n
SlurmctldPidFile=/SHARED/slurm-install/var/slurmctld.pid
SlurmdPidFile=/SHARED/slurm-install/var/slurmd.%n.pid
ProctrackType=proctrack/pgid

# LOGGING
SlurmctldDebug=info
SlurmctldLogFile=/SHARED/slurm-install/var/slurmctld.log
SlurmdDebug=info
SlurmdLogFile=/SHARED/slurm-install/var/slurmd.%n.log

# COMPUTE NODES
NodeName=nodo1 CPUs=4 Boards=1 SocketsPerBoard=4 ↩
    CoresPerSocket=1 ThreadsPerCore=1 RealMemory=1867
NodeName=nodo2 CPUs=4 Boards=1 SocketsPerBoard=4 ↩
    CoresPerSocket=1  ThreadsPerCore=1 RealMemory=1867
NodeName=nodo3 CPUs=4 Boards=1 SocketsPerBoard=4 ↩
    CoresPerSocket=1  ThreadsPerCore=1 RealMemory=1867
PartitionName=main Nodes=ALL Default=YES MaxTime=INFINITE↩
     State=UP
```

Dependiendo de las características del nodo, la configuración en Slurm puede variar. Para obtener las características concretas de los nodos que conforman el clúster se puede utilizar el comando slurmd con la opción "-C". Por

ejemplo:

```
pi@nodo0:~ $ slurmd -C
NodeName=nodo0 CPUs=4 Boards=1 SocketsPerBoard=4 CoresPer-
    Socket=1 ThreadsPerCore=1 RealMemory=1867 UpTime=4-01:03:29
```

De igual forma que en el caso de MUNGE, se utilizará systemctl para controlar los demonios slurmctld y slurmd. En el *front-end* y con permisos de superusuario, se creará en el directorio */usr/lib/systemd/user/* el fichero *slurmctld.service*:

Fichero: /usr/lib/systemd/user/slurmctld.service

```
[Unit]
Description=Slurm controller daemon
After=network.target munge.service
ConditionPathExists=/SHARED/slurm-install/confdir/slurm.↩
    conf

[Service]
Type=forking
EnvironmentFile=-/SHARED/slurm-install/confdir/slurmctld
ExecStart=/SHARED/slurm-install/sbin/slurmctld ↩
    $SLURMCTLD_OPTIONS
ExecReload=/bin/kill -HUP $MAINPID
PIDFile=/SHARED/slurm-install/var/slurmctld.pid
LimitNOFILE=65536
TasksMax=infinity

[Install]
WantedBy=multi-user.target
```

Mientras que para el servicio en los nodos de cómputo el fichero será el siguiente *slurmd.service*:

Fichero: /SHARED/slurm-install/slurmd.service

```
[Unit]
Description=Slurm node daemon
After=munge . service network . target remote–fs . target
ConditionPathExists=/SHARED /slurm–install / confdir / slurm .
conf

[Service]
Type=forking
EnvironmentFile=–/SHARED /slurm–install / confdir / slurmd
ExecStart=/SHARED /slurm–install / sbin / slurmd
$SLURMD_OPTIONS
ExecReload=/bin/ kill –HUP $MAINPID
PIDFile=/SHARED /slurm–install /var/ slurmd .% H . pid
KillMode=process
LimitNOFILE =131072
LimitMEMLOCK=infinity
LimitSTACK=infinity
Delegate=yes
TasksMax=infinity

[Install]
WantedBy=multi–user . target
```

Para que el servicio slurmd se encuentre disponible en todos los nodos de cómputo sin tener que crearlo manualmente, este se crea desde el *front-end* en el directorio compartido */SHARED/slurm-install* para luego, utilizando el script *ejecutar_en_nodos_computo.sh*, copiarlo al directorio */usr/lib/sys-temd/user/* de cada nodo de cómputo.

Por tanto, una vez han sido creados los ficheros, se copiará *slurmd.service*

al directorio */usr/lib/systemd/user/*, se recargarán las configuraciones de los servicios en systemd antes de iniciar los demonios en todos los nodos. A modo de resumen, estas operaciones se pueden realizar ejecutando los siguientes comandos:

```
COMENTARIO: Creación de slurmd.service en los nodos de cómputo
pi@nodoo:~ $ ./ejecutar_en_nodos_computo.sh sudo cp
    /SHARED/slurm-install/slurmd.service/usr/lib/systemd/user/

COMENTARIO: Recargar las configuraciones de servicios en systemd
pi@nodoo:~ $ systemctl --user daemon-reload
pi@nodoo:~ $
    ./ejecutar_en_nodos_computo.sh systemctl --user daemon-reload

COMENTARIO: Iniciar los demonios
pi@nodoo:~ $
    ./ejecutar_en_nodos_computo.sh systemctl --user start slurmd
pi@nodoo:~ $ systemctl --user start slurmctld
```

Se pueden comprobar los mensajes generados por los distintos demonios en el directorio */SHARED/slurm-install/var*.

Para comprobar que todos los nodos están activos se puede enviar el primer trabajos tal y como indica la Consola 6.7. Obsérvese que, de nuevo, el orden en que se muestran los mensajes no está controlado.

```
pi@nodoo:~ $ srun -N3 hostname
```

```
nodo1
nodo3
nodo2
```

Consola 6.7 Enviar un trabajo a Slurm que requiera tres nodos y que ejecute el comando hostname.

6.2 Para profundizar...

1. Como sistema gestor de carga de trabajo en este libro se presenta Slurm, sin embargo, existen otros gestores. ¿Conoce otros gestores de carga de trabajo? Entre todos ellos, ¿cuál o cuáles son los más utilizados habitualmente en los supercomputadores del TOP500? Los siguientes recursos web recogen información al respecto: https://arxiv.org/ftp/arxiv/papers/1607/1607.06544.-

   https://www.supercomputingonline.com/latest/topics/applications/engineering/52207-slurm-on-on-the-test-of-the-to-p500-super-putersters

2. Por defecto, Slurm utiliza una política de planificación en la que los trabajos se inician por orden de llegada a la cola (*first come first served* [FIFO], en inglés). Sin embargo, existen otras más complejas, pero más habituales, en sistemas en producción como *backfilling* y sus variaciones. ¿En qué difieren estas políticas?

   Si se considerara interesante, podría habilitarse la política *backfilling* en Slurm añadiendo la siguiente línea en el fichero *slurm.conf* y reiniciando los demonios:

Fichero: /SHARED/slurm-install/confdir/slurm.conf

```
SchedulerType=sched / backfill
```

6.3 Material recomendado

En este capítulo se ha presentado Slurm como gestor de carga de trabajo. La lectura del artículo de investigación [1] es especialmente recomendable para profundizar en el aprendizaje de esta herramienta, ya que presenta una descripción detallada de la misma. Aunque está ampliamente extendido en entornos de supercomputación, existen otros gestores de trabajos que siguen las metodologías genéricas propuestas en los artículos de investigación [2] y [3] para la gestión de recursos y trabajos en entornos de supercomputación, cuya revisión puede ayudar a alcanzar una visión global más amplia de los gestores de trabajos y recursos. Finalmente, en el artículo de investigación [4], se propone una API que permite a gestores de carga externos iniciar y parar los trabajos del sistema y que puede resultar de interés si se desea profundizar en este aspecto.

[1] Andy B Yoo, Morris A Jette, and Mark Grondona. Slurm: Simple linux utility for resource management, 2003.

[2] Dror G. Feitelson. Packing schemes for gang scheduling, 1996.

[3] Dror G Feitelson and Larry Rudolph. Toward convergence in job schedulers for parallel supercomputers. volume 1162/1996, pages 1–26, 1996.

[4] Joseph Skovira, Waiman Chan, Honbo Zhou, and David Lifka. The easy – loadleveler api project. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing* , pages 41–47, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

6.4 Resumen

En este capítulo se ha introducido el concepto de gestión de recursos y su necesidad en entornos de supercomputación. Concretamente, se ha detallado el funcionamiento de Slurm y descrito parte de sus comandos. También se ha visto cómo los usuarios, mediante trabajos, reservan recursos para sus ejecuciones. Posteriormente, se ha descrito paso a paso la instalación de Slurm, para la cual se ha optado por utilizar MUNGE como sistema transversal de autenticación de usuarios en el clúster RPi. El capítulo concluye con una serie de ejemplos que sirven para comprobar la correcta instalación.

6.4.1 ¿Qué viene a continuación?

En el siguiente capítulo se muestran algunas aplicaciones útiles para medir el rendimiento y monitorizar un sistema computacional: cpufrequtils (conjunto de utilidades para modificar la frecuencia a la que trabaja el sistema), stress-ng (herramienta para generar carga en el sistema y medir la capacidad de ciertos componentes del mismo), htop (utilidad para monitorizar el sistema en tiempo real), stream (*benchmark* que permite medir el ancho de banda de memoria y tasa de cálculo) y HPL (LINPACK preparado para su ejecución paralela). Además, se presentan también aplicaciones científicas utilizadas en el ámbito de la HPC hoy en día, dado que presentan un alto coste computacional: *LAMMPS* (simulación de dinámica molecular) y OpenFOAM (simulación de dinámica de fluidos).

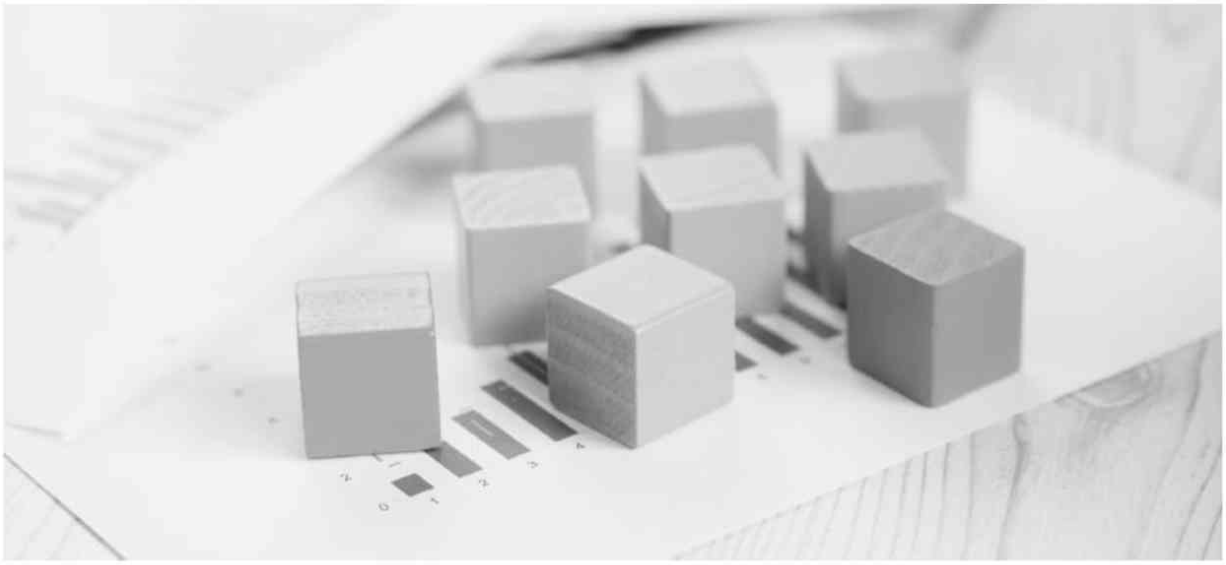
1 https://slurm.schedmd.com

2 https://slurm.schedmd.com/documentation.html

3 https://github.com/dun/munge

4 https://man7.org/linux/man-pages/man1/systemd.1.html

5 https://man7.org/linux/man-pages/man1/systemctl.1.html



7. Aplicaciones

Para analizar el comportamiento de los distintos elementos software y hardware configurados en los capítulos anteriores, en este capítulo se propone el análisis y uso de herramientas de monitorización, *benchmarks*, y aplicaciones científicas ampliamente utilizadas en el campo de la HPC. Este capítulo recoge solo una muestra de ellas, pero a modo introductorio, serán suficientes para analizar el comportamiento del clúster RPi. Concretamente, se tratarán seis herramientas cuya finalidad es evaluar el rendimiento del sistema y monitorizarlo, y dos aplicaciones científicas capaces de poner a prueba las prestaciones del mismo.

7.1 Rendimiento y monitorización del sistema

El rendimiento alcanzado por las aplicaciones no depende solo de cómo estén programadas y optimizadas, también se deben tener en cuenta las características físicas del sistema en el que se ejecutan, así como la utilización por parte de varios usuarios de forma simultánea. Por esta razón, es importante conocer *benchmarks* que nos permitan caracterizar el sistema, así como herramientas que permitan monitorizar en tiempo real el estado del sistema e, incluso, modificar parte de su configuración. En este apartado se presentan cpufrequtils, stress-ng, htop, stream, iperf y LINPACK.

7.1.1 cpufrequtils

cpufrequtils son un conjunto de utilidades que permiten tanto consultar como modificar la frecuencia a la que están trabajando los núcleos y el regulador (más conocido como *governor*, en inglés) que está en uso en el sistema. El regulador es la política que se sigue para, si es pertinente, modificar la frecuencia en función de unas determinadas condiciones. Los reguladores disponibles son los que se muestran en la Tabla 7.1.

Regulador	**Función**
performance	Utiliza la máxima frecuencia
powersave	Utiliza la mínima frecuencia
userspace	Utiliza la frecuencia especificada por el usuario
ondemand	Varía la frecuencia en función de la carga del sistema
conservative	Varía la frecuencia en función de la carga del sistema, pero lo hace de

	forma más gradual que
ondemand schedutil	La selección de la frecuencia depende del planificador del sistema operativo

Tabla 7.1 Reguladores disponibles en el sistema y su función

Las utilidades disponibles a través de cpufrequtils son dos:

- cpufreq-info 1: Devuelve la información sobre frecuencia y reguladores de todas las CPUs del sistema.
- cpufreq-set 2: Permite cambiar el regulador, la frecuencia y los límites mínimo y máximo de la misma. Requiere permisos de administrador.

Aunque también se puede instalar a partir del código fuente en el directorio compartido del clúster para que todos los nodos tengan acceso a la herramienta, a modo ilustrativo, se describe el proceso de instalación de cpufrequtils en el *front-end* haciendo uso del respositorio de aplicaciones del sistema operativo:

```
pi@nodo0:~ $ sudo apt-get install -y cpufrequtils
```

A continuación, se muestra un ejemplo de uso de las utilidades de cpufrequtils mencionadas:

- Comprobación de la configuración del sistema mediante cpufreq-info. Una posible salida de la ejecución de este comando sería (por brevedad, solo se incluye la información de las CPUs 0 y 1):

```
pi@nodo0:~ $ sudo cpufreq-info
cpufrequtils 008: cpufreq-info (C) Dominik Brodowski 2004-2009
Report errors and bugs to cpufreq@vger.kernel.org, please.
analyzing CPU 0:
  driver: cpufreq-dt
  CPUs which run at the same hardware frequency: 0 1 2 3
  CPUs which need to have their frequency coordinated by software: 0 1
    2 3
  maximum transition latency: 0.00 ms.
  hardware limits: 600 MHz - 1.50 GHz
  available frequency steps: 600 MHz, 700 MHz, 800 MHz, 900 MHz, 1000
    MHz, 1.10 GHz, 1.20 GHz, 1.30 GHz, 1.40 GHz, 1.50 GHz
  available cpufreq governors: conservative, ondemand, userspace,
    powersave, performance, schedutil
  current policy: frequency should be within 600 MHz and 1.50 GHz.
                  The governor "ondemand" may decide which speed to use
                  within this range.
  current CPU frequency is 600 MHz.
  cpufreq stats: 600 MHz:49,75%, 700 MHz:11,38%, 800 MHz:3,82%, 900
    MHz:2,02%, 1000 MHz:1,82%, 1.10 GHz:0,02%, 1.20 GHz:0,02%, 1.30
    GHz:0,01%, 1.40 GHz:0,01%, 1.50 GHz:31,15%  (33222)
analyzing CPU 1:
  driver: cpufreq-dt
  CPUs which run at the same hardware frequency: 0 1 2 3
  CPUs which need to have their frequency coordinated by software: 0 1
    2 3
  maximum transition latency: 0.00 ms.
  hardware limits: 600 MHz - 1.50 GHz
  available frequency steps: 600 MHz, 700 MHz, 800 MHz, 900 MHz, 1000
    MHz, 1.10 GHz, 1.20 GHz, 1.30 GHz, 1.40 GHz, 1.50 GHz
  available cpufreq governors: conservative, ondemand, userspace,
    powersave, performance, schedutil
  current policy: frequency should be within 600 MHz and 1.50 GHz.
                  The governor "ondemand" may decide which speed to use
                  within this range.
  current CPU frequency is 600 MHz.
  cpufreq stats: 600 MHz:49,75%, 700 MHz:11,38%, 800 MHz:3,82%, 900
    MHz:2,02%, 1000 MHz:1,82%, 1.10 GHz:0,02%, 1.20 GHz:0,02%, 1.30
    GHz:0,01%, 1.40 GHz:0,01%, 1.50 GHz:31,15%  (33222)
```

Esta salida muestra que todas las CPU utilizan el regulador *ondemand* y que su frecuencia actual es de 600 MHz. Al utilizarse la política *ondemand*, vemos que los núcleos pueden variar su frecuencia entre las disponibles (en este caso 600MHz y 1.5GHz). Además, en la parte

inferior, se muestra para cada núcleo el porcentaje de tiempo que ha estado configurado a las distintas frecuencias disponibles.

- Cambio de regulador y frecuencia de los núcleos. Los cambios efectuados se comprueban con cpufreq-info. A continuación, se muestran los comandos y una posible salida (por brevedad, solo se incluye la información de las CPUs 0 y 1):

```
pi@nodo0:~ $ sudo cpufreq-set -g 'userspace'
pi@nodo0:~ $ sudo cpufreq-set -f 900000
pi@nodo0:~ $ sudo cpufreq-info
cpufrequtils 008: cpufreq-info (C) Dominik Brodowski 2004-2009
Report errors and bugs to cpufreq@vger.kernel.org, please.
analyzing CPU 0:
  driver: cpufreq-dt
  CPUs which run at the same hardware frequency: 0 1 2 3
  CPUs which need to have their frequency coordinated by software: 0 1
    2 3
  maximum transition latency: 0.00 ms.
  hardware limits: 600 MHz - 1.50 GHz
  available frequency steps: 600 MHz, 700 MHz, 800 MHz, 900 MHz, 1000
    MHz, 1.10 GHz, 1.20 GHz, 1.30 GHz, 1.40 GHz, 1.50 GHz
  available cpufreq governors: conservative, ondemand, userspace,
    powersave, performance, schedutil
  current policy: frequency should be within 600 MHz and 1.50 GHz.
                  The governor "userspace" may decide which speed to
    use
                  within this range.
  current CPU frequency is 900 MHz.
  cpufreq stats: 600 MHz:50,44%, 700 MHz:11,23%, 800 MHz:3,75%, 900
    MHz:2,02%, 1000 MHz:1,81%, 1.10 GHz:0,06%, 1.20 GHz:0,15%, 1.30
    GHz:0,07%, 1.40 GHz:0,08%, 1.50 GHz:30,40%  (38404)
analyzing CPU 1:
  driver: cpufreq-dt
  CPUs which run at the same hardware frequency: 0 1 2 3
  CPUs which need to have their frequency coordinated by software: 0 1
    2 3
  maximum transition latency: 0.00 ms.
  hardware limits: 600 MHz - 1.50 GHz
  available frequency steps: 600 MHz, 700 MHz, 800 MHz, 900 MHz, 1000
    MHz, 1.10 GHz, 1.20 GHz, 1.30 GHz, 1.40 GHz, 1.50 GHz
  available cpufreq governors: conservative, ondemand, userspace,
    powersave, performance, schedutil
  current policy: frequency should be within 600 MHz and 1.50 GHz.
                  The governor "userspace" may decide which speed to
    use
                  within this range.
  current CPU frequency is 900 MHz.
  cpufreq stats: 600 MHz:50,44%, 700 MHz:11,23%, 800 MHz:3,75%, 900
    MHz:2,02%, 1000 MHz:1,81%, 1.10 GHz:0,06%, 1.20 GHz:0,15%, 1.30
    GHz:0,07%, 1.40 GHz:0,08%, 1.50 GHz:30,40%  (38404)
```

La salida muestra que para todos los núcleos el regulador pasa a ser ahora *userspace*, y la frecuencia, 900 MHz.

Antes de continuar, restablecemos la configuración anterior del sistema:

```
pi@nodoo:~ $ sudo cpufreq-set -g 'ondemand'
```

7.1.2 stress-ng

stress-ng[3] es una herramienta diseñada para generar carga en el sistema y así medir la capacidad de componentes como la CPU, la memoria, el sistema de entrada/salida o los discos. Todo ello se lleva a cabo bajo unas determinadas condiciones de estrés en el sentido computacional, es decir, forzando un uso elevado de los recursos del sistema. Para su instalación (versión 0.11.24) se pueden seguir los siguientes pasos:

```
COMENTARIO: Descargar y descomprimir los ficheros fuente:
pi@nodoo:~ $
    wget https://github.com/ColinIanKing/stress-ng/archive/master.zip
pi@nodoo:~ $ unzip master.zip -d /SHARED/
pi@nodoo:~ $ rm master.zip
pi@nodoo:~ $ cd /SHARED/stress-ng-master

COMENTARIO: Compilar
pi@nodoo:/SHARED/stress-ng-master $ make
```

Las principales opciones de stress-ng[4] son:

- **-t, --timeout** <segundos>: especifica el tiempo durante el cual se estresa el sistema (en segundos).
- **-c, --cpu** <hilos>: indica el número de hilos que generan carga en los

procesadores.

- **-i, --io** <hilos>: indica el número de hilos que realizan operaciones de entrada/salida para estresar el sistema.
- **-m, --vm** <hilos>: indica el número de hilos que realizan operaciones de reserva y liberación de memoria para estresar la memoria del sistema.
- **-d, --hdd** <hilos>: indica el número de hilos que continuamente escriben, leen y borran ficheros temporales en disco.

Entre otros muchos ejemplos[5], a continuación se muestran los comandos de stress-ng (y unas de sus posibles salidas) que, respectivamente, cargan al 100% cada procesador de la máquina durante cinco minutos y ejecutan dos procesos que reservan el 95% de la memoria principal durante ese mismo intervalo de tiempo.

```
pi@nodo0:/SHARED/stress-ng-master $ ./stress-ng --cpu 0 -t300
stress-ng: info:[29885] dispatching hogs: 4 cpu
stress-ng: info:[29885] successful run completed in 300.09s (5 mins,
    0.09 secs)
pi@nodo0:/SHARED/stress-ng-master $ ./stress-ng --vm 2 --vm-bytes
    95% -t300
stress-ng: info:[29913] dispatching hogs: 2 vm
stress-ng: info:[29913] successful run completed in 300.03s (5 mins, 0.03
    secs)
```

7.1.3 htop

htop[6] es una herramienta, entre muchas otras, que permite la

monitorización del sistema. Permitiendo ver en tiempo real, por ejemplo, el consumo de CPU, de memoria RAM, o qué procesos consumen más. htop se presenta como una mejora de la utilidad top instalada por defecto en muchas distribuciones Linux. Respecto a top, htop interactúa con las acciones del ratón, a la vez que facilita la lectura mediante el uso de colores. Además, a través de las teclas de función del teclado (F1 a F10), se puede acceder a una serie de utilidades (por ejemplo, terminar un proceso), las cuales vienen indicadas en la parte inferior del monitor.

La versión del sistema operativo propuesta en este libro para el clúster RPi, tiene htop instalado por defecto. En caso de que la herramienta no esté instalada en el sistema operativo elegido, los pasos descritos a continuación permiten descargar y configurar htop (versión 3.0.2) para que pueda ser ejecutado en todos los nodos del clúster:

```
COMENTARIO: Descargar y descomprimir los ficheros fuente
pi@nodo0:~ $ wget https://dl.bintray.com/htop/source/htop-3.0.2.tar.gz
pi@nodo0:~ $ tar -xvf htop-3.0.2.tar.gz
pi@nodo0:~ $ rm htop-3.0.2.tar.gz

COMENTARIO: Instalar dependencias y compilar e instalar htop
pi@nodo0:~ $ sudo apt-get install -y libncurses-dev
pi@nodo0:~ $ cd htop-3.0.2/
pi@nodo0:~/htop-3.0.2
     $  ./configure --prefix=/SHARED/htop-install --disable-unicode
pi@nodo0:~/htop-3.0.2 $ make
pi@nodo0:~/htop-3.0.2 $ make install
pi@nodo0:~/htop-3.0.2 $ cd

COMENTARIO: Añadir la ruta de instalación al PATH
pi@nodo0:~ $  export PATH=/SHARED/htop-install/bin:$PATH
```

```
  1  [||                          4.6%]   Tasks: 77, 165 thr; 1 running
  2  [|                           0.7%]   Load average: 1.11 1.53 1.20
  3  [                            0.0%]   Uptime: 00:41:31
  4  [|                           1.3%]
  Mem[||||||||||||||||||1.08G/1.82G]
  Swp[|||||||||||||      47.8M/100.0M]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 1653 pi         20   0  812M  245M 94636 S  4.6 13.1  8:11.21 /usr/lib/chromium-
 2006 pi         20   0  8940  3616  2416 R  2.0  0.2  0:01.45 htop
 1659 pi         20   0  812M  245M 94636 S  2.0 13.1  1:13.34 /usr/lib/chromium-
 1361 pi         20   0  686M  356M  327M S  1.3 19.1  1:27.92 /usr/lib/chromium-
  597 root       20   0  150M 54620 37392 S  0.7  2.9  1:10.83 /usr/lib/xorg/Xorg
 1325 pi         20   0  686M  356M  327M S  0.7 19.1  4:51.82 /usr/lib/chromium-
 1274 pi         20   0  553M  127M 78932 S  0.7  6.8  1:52.88 /usr/lib/chromium-
  647 root       20   0  150M 54620 37392 S  0.7  2.9  0:08.14 /usr/lib/xorg/Xorg
 1656 pi         20   0  812M  245M 94636 S  0.7 13.1  0:18.17 /usr/lib/chromium-
 1859 pi         20   0  553M  127M 78932 S  0.7  6.8  0:05.54 /usr/lib/chromium-
  956 pi         20   0 87620 20736 13464 S  0.0  1.1  0:06.79 lxterminal
 1359 pi         20   0  686M  356M  327M S  0.0 19.1  0:30.22 /usr/lib/chromium-
 1310 pi         20   0  553M  127M 78932 S  0.0  6.8  0:23.94 /usr/lib/chromium-
 1661 pi         20   0  812M  245M 94636 S  0.0 13.1  0:09.93 /usr/lib/chromium-
  726 pi         20   0  414M 18748 12188 S  0.0  1.0  0:04.83 lxpanel --profile
 1336 pi         20   0  280M 45948 35672 S  0.0  2.4  0:17.31 /usr/lib/chromium-
 1332 pi         20   0  280M 45948 35672 S  0.0  2.4  0:21.15 /usr/lib/chromium-
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Figura 7.1 Ejemplo de salida del comando htop.

Una vez finalizada la instalación, bastará con ejecutar htop, cuya salida podría ser similar a la mostrada en la Figura 7.1. En la figura se puede apreciar cómo la parte superior contiene información de la carga de cada uno de los cuatro núcleos de la RPi (existe una leve carga en el primer, segundo y cuarto núcleo). También se observa la ocupación de la memoria (*1,08G/1,82G*) y del espacio de intercambio *swap* (*47,8M/100,0M*). A la derecha se muestran algunas estadísticas, como el número de procesos (*tasks: 77*), de hilos (*165 thr*) o la carga media del sistema (*load average*). En la parte inferior se encuentra detallada la información clasificada de cada proceso. Entre los campos mostrados, cabe destacar el PID (identificador del proceso en el sistema), el propietario (*USER*), el consumo de memoria (*MEM%*) y el de CPU (*CPU%*), así como el tiempo que lleva en ejecución (*TIME+*) y el comando que lo ha generado (*Command*).

7.1.4 stream

stream[7] es un *benchmark* que permite medir el ancho de banda de memoria en MB/s y la tasa de cálculo correspondiente. Sirve para hacer una medición real sobre el computador a través de un código que *estresa* tanto la memoria como la CPU, con la finalidad de medir su rendimiento. Para ello, realiza cuatro operaciones diferentes en las que se hace un número distinto de accesos a memoria.

Los pasos para instalar stream (versión 5.10), configurarlo para que utilice OpenMP y ejecutarlo son:

```
COMENTARIO: Descargar los ficheros fuente
pi@nodoo:~ $ wget https://github.com/jeffhammond/
    STREAM/chive/master.zipr.zip
```

```
pi@nodoo:~ $ unzip master.zip -d /SHARED/
pi@nodoo:~ $ rm master.zip
pi@nodoo:~ $ cd /SHARED/STREAM-master

COMENTARIO: Compilar stream con OpenMP
pi@nodoo:/SHARED/STREAM-master $ gcc -fopenmp stream.c -o
    stream
```

Una vez instalado stream, será necesario determinar el número de hilos que se van a utilizar en la ejecución, usando para ello la variable de entorno OMP_NUM_THREADS. A continuación se muestra un ejemplo de ejecución y la salida asociada a la misma. Se puede observar lo siguiente: el número de bytes que tienen los elementos de los vectores usados en las pruebas, información diversa sobre la memoria utilizada en las pruebas, el número de hilos utilizados (debe coincidir con los indicados en la variable de entorno), la periodicidad de las muestras (*clock granularity*, es decir, cada cuánto se toman los tiempos en la prueba) y, finalmente, el ancho de banda y el tiempo medio, mínimo y máximo para las cuatro operaciones realizadas en el *benchmark*.

```
pi@nodo0:~ $ export OMP_NUM_THREADS=2
pi@nodo0:~ $ /SHARED/STREAM-master/stream
-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 8 bytes per array element.
-------------------------------------------------------------
Array size = 10000000 (elements), Offset = 0 (elements)
Memory per array = 76.3 MiB (= 0.1 GiB).
Total memory required = 228.9 MiB (= 0.2 GiB).
Each kernel will be executed 10 times.
 The *best* time for each kernel (excluding the first iteration)
 will be used to compute the reported bandwidth.
-------------------------------------------------------------
Number of Threads requested = 2
Number of Threads counted = 2
-------------------------------------------------------------
Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 62575 microseconds.
   (= 62575 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:           1863.0     0.087232     0.085881     0.095198
Scale:          1879.2     0.086292     0.085143     0.093836
Add:            2320.4     0.103515     0.103430     0.103737
Triad:          2280.4     0.106328     0.105244     0.112842
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three
     arrays
-------------------------------------------------------------
```

7.1.5 iperf

iperf[8] es una herramienta de medición activa del ancho de banda máximo alcanzable en redes IP. iperf se basa en una arquitectura cliente-servidor, en la que para llevar a cabo la medición, dos nodos de la red tienen

que ejecutar cada uno una de estas partes.

Para la instalación de la versión 3.1.4 en el clúster RPi se seguirán los siguientes pasos:

```
pi@nodo0:~ $ wget https://iperf.fr/download/source/iperf-3.1.3-sour-tar.gzr.gz
pi@nodo0:~ $ tar -xvf iperf-3.1.3-source.tar.gz
pi@nodo0:~ $ rm iperf-3.1.3-source.tar.gz
pi@nodo0:~ $ cd iperf-3.1.3
pi@nodo0:/SHARED/iperf-3.1.3 $ ./configure --prefix=/SHARED/iper-installtall
pi@nodo0:/SHARED/iperf-3.1.3 $ make -j4
pi@nodo0:/SHARED/iperf-3.1.3 $ make install
```

Una vez instalado, se tendrá que iniciar el servidor en un nodo y desde otro nodo establecer la conexión e iniciar la transferencia de datos.

Por ejemplo, la siguiente consola muestra como dejar funcionando el servidor de iperf en el *front-end*:

```
pi@nodo0:~ $ cd /SHARED/iperf3-install/bin
pi@nodo0:/SHARED/iperf-3.1.3 $ ./iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Desde otro terminal, se abrirá una nueva sesión SSH en el nodo1 (por ejemplo) y se lanzará el cliente de iperf durante 5 segundos (opción "-t") para

que inicie una conexión al nodo0 (opción "-c"):

```
pi@nodo1:~ $ cd /SHARED/iperf3-install/bin
pi@nodo1:/SHARED/iperf-3.1.3 $ ./iperf3 -c nodo0 -t5
Connecting to host nodo0, port 5201
[  4] local 192.168.0.2 port 48188 connected to 192.168.0.1 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec   109 MBytes   918 Mbits/sec    0    320 KBytes
[  4]   1.00-2.00   sec   112 MBytes   937 Mbits/sec    0    320 KBytes
[  4]   2.00-3.00   sec   111 MBytes   934 Mbits/sec    0    365 KBytes
[  4]   3.00-4.00   sec   111 MBytes   934 Mbits/sec    0    365 KBytes
[  4]   4.00-5.00   sec   111 MBytes   934 Mbits/sec    0    365 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-5.00   sec   555 MBytes   931 Mbits/sec    0   sender
[  4]   0.00-5.00   sec   554 MBytes   930 Mbits/sec        receiver
```

De vuelta al terminal del *front-end*, se podrán ver las transferencias que se han realizado durante la conexión con el nodo1:

```
Accepted connection from 192.168.0.2, port 48186
[  5] local 192.168.0.1 port 5201 connected to 192.168.0.2 port 48188
[ ID] Interval           Transfer     Bandwidth
[  5]   0.00-1.00   sec   108 MBytes   910 Mbits/sec
[  5]   1.00-2.00   sec   112 MBytes   937 Mbits/sec
[  5]   2.00-3.00   sec   111 MBytes   933 Mbits/sec
[  5]   3.00-4.00   sec   111 MBytes   934 Mbits/sec
[  5]   4.00-5.00   sec   111 MBytes   934 Mbits/sec
[  5]   5.00-5.00   sec   127 KBytes   888 Mbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth
[  5]   0.00-5.00   sec  0.00 Bytes  0.00 bits/sec  sender
[  5]   0.00-5.00   sec   554 MBytes   929 Mbits/sec    receiver
```

Los resultados muestran que se ha conseguido un ancho de banda medio de más de 900 Mbits/s, lo que se aproxima al ancho de banda teórico de la interfaz de red Ethernet de la RPi, que es 1 Gbit/s.

7.1.6 LINPACK

LINPACK[9] es una biblioteca formada por una colección de subrutinas programadas en Fortran que analizan y resuelven ecuaciones lineales y problemas de mínimos cuadrados. Dada la exigencia computacional de algunos de los cálculos que lleva a cabo, como ya se ha comentado en el Capítulo 1, es el *benchmark* empleado en el TOP500 para evaluar y clasificar los sistemas de supercomputación. Bajo las siglas HPL (*High Performance Linpack*, en inglés) se distribuye actualmente la versión para memoria distribuida, la cual incluye un test que permite saber con qué precisión se resuelve un sistema denso de ecuaciones lineales aleatorias y el tiempo empleado para ello.

Los pasos mostrados a continuación ilustran cómo descargar e instalar HPL (versión 2.3) en el directorio compartido del clúster RPi. Cabe destacar que es necesario recompilar MPICH para que pueda trabajar con Fortran, por lo que este será el paso previo a la instalación de HPL. Si se dispone del código fuente descargado en el Capítulo 5, bastará con seguir los pasos que siguen. En caso contrario, será necesario descargar y descomprimir el código fuente de MPICH.

```
COMENTARIO: Descargar e instalar dependencias
pi@nodoo:~ $ sudo apt-get install -y libatlas-base-dev gfortran

COMENTARIO: Compilar MPICH con la biblioteca de Fortran
pi@nodoo:~ $ cd ~/mpich-3.3.2
pi@nodoo:~/mpich-3.3.2 $ ./configure --prefix=/SHARED/
    mpich-3.3.2-fort CC=gcc CXX=g++ FC=gfortran
pi@nodoo:~/mpich-3.3.2 $ make -j 4
pi@nodoo:~/mpich-3.3.2 $ make install
```

```
pi@nodoo:~/mpich-3.3.2 $ cd

COMENTARIO: Añadir MPICH en las variables de entorno
pi@nodoo:~ $ export LD_LIBRARY_PATH=/SHARED/mpich-
    3.3.2-fort/b:$LD_LIBRARY_PATHPATH
pi@nodoo:~ $ export PATH=/SHARED/mpich-3.3.2-fort/bin:$PATH

COMENTARIO: Descargar e instalar HPL
pi@nodoo:~ $ wget https://www.netlib.org/
    bench-mark/hpl/2.3.tar.gzr.gz
pi@nodoo:~ $ tar -xzvf hpl-2.3.tar.gz
pi@nodoo:~ $ rm hpl-2.3.tar.gz

COMENTARIO: Instalar HPL en el directorio compartido /SHARED
pi@nodoo:~ $ mv hpl-2.3 /SHARED/hpl
pi@nodoo:~ $ cd /SHARED/hpl/setup
pi@nodoo:/SHARED/hpl/setup $ sh make_generic
pi@nodoo:/SHARED/hpl/setup $ cd ..
pi@nodoo:/SHARED/hpl $ cp setup/Make.UNKNOWN Make.rpi
```

Una vez creado el fichero */SHARED/hpl/Make.rpi* será necesario configurarlo para que haga referencia a la arquitectura de la RPi. Además, será necesario indicar cuál es la estructura de los directorios HPL, así como las bibliotecas MPI y de álgebra lineal. Para ello, se deberán modificar las secciones del mencionado fichero como se muestra a continuación:

Fichero: /SHARED/hpl/Make.rpi

```
#
# ----------------------------
# - Platform identifier ----------------
# ----------------------------
#
ARCH         = rpi
#
# ----------------------------
# - HPL Directory Structure / HPL library ------
# ----------------------------
#
TOPdir       = /SHARED/hpl
INCdir       = $(TOPdir)/include
```

```
BINdir       = $(TOPdir)/bin/$(ARCH)
LIBdir       = $(TOPdir)/lib/$(ARCH)
#
HPLlib       = $(LIBdir)/libhpl.a
#
# ----------------------------
# - Message Passing library (MPI) -----------
# ----------------------------
# MPinc tells the C compiler where to find the Message ↩
    Passing library header files, MPlib is defined to be ↩
    the name of the library to be used. The variable ↩
    MPdir is only used for defining MPinc and MPlib.
#
MPdir        = /SHARED/mpich-3.3.2-fort
MPinc        = -I $(MPdir)/include
MPlib        = -L$(MPdir)/lib/libmpich.a
#
# ----------------------------
# - Linear Algebra library (BLAS or VSIPL) -----
# ----------------------------
# LAinc tells the C compiler where to find the Linear ↩
    Algebra library header files, LAlib is defined to be ↩
    the name of the library to be used. The variable ↩
    LAdir is only used for defining LAinc and LAlib.
#
LAdir        = /usr/lib/arm-linux-gnueabihf/
LAinc        =
LAlib        = $(LAdir)/libf77blas.a $(LAdir)/libatlas.a
```

Tras la actualización del fichero */SHARED/hpl/Make.rpi* se procederá a la compilación de HPL como se muestra a continuación:

```
pi@nodoo:/SHARED/hpl $ make -j4 arch=rpi
```

Una vez instalado HPL, es necesario configurarlo en función del número de nodos y núcleos que se vayan a utilizar para comprobar el rendimiento. Para ello, es necesario editar el fichero */SHARED/hpl/bin/rpi/HPL.dat*. Este fichero de configuración contiene una larga lista de parámetros que permiten configurar la prueba que se va a lanzar en la máquina. En este caso, se propone hacer los cambios mínimos para que la prueba pueda ejecutarse correctamente en el clúster RPi aprovechando los tres nodos de cómputo al máximo. A continuación, se muestran las líneas que deben modificarse y cómo quedan finalmente para la correcta ejecución de HPL en el clúster RPi con tres nodos de cómputo, con cuatro núcleos cada uno y 2 GB de memoria RAM.

Fichero: /SHARED/hpl/bin/rpi/HPL.dat

1	# of problems sizes (N)
24320	Ns
1	# of NBs
128	NBs
1	# of process grids (P x Q)
3	Ps
4	Qs

16.0	threshold
1	# of panel fact
2	PFACTs (0=left , 1=Crout , 2=Right)
1	# of recursive stopping criterium
4	NBMINs (>= 1)
1	# of recursive panel fact .
1	RFACTs (0=left , 1=Crout , 2=Right)
1	# of broadcast
1	BCASTs (0=1rg ,1=1rM ,2=2rg ,3=2rM ,4=Lng ,5= LnM)
1	DEPTHs (>=0)

Las líneas modificadas en el fichero hacen referencia a los siguientes parámetros:

- N. Número de problemas a resolver.
- Ns. Tamaño del problema a resolver. Se recomienda que el sistema ocupe aproximadamente un 80-90% de la memoria disponible, teniendo en cuenta que se trata de números de ocho bytes y el tamaño de bloque viene definido por NB, es decir:

$$max\_elementos\_\% = \sqrt{\frac{\#nodos \cdot memoria\_bytes}{8}} \cdot \frac{memoria\_\%}{100}$$

$$\#bloques = \frac{max\_elementos\_\%}{NB}$$

$$elementos\_matriz = \#bloques \cdot NB$$

Por ejemplo, en el caso de tres nodos RPi y un tamaño de bloque NB = 128, para el 86% se obtendría:

$$\sqrt{\frac{3\ nodos \cdot (2\ GB \cdot 1.024 \cdot 1.024 \cdot 1.024)\ bytes}{8}} \cdot \frac{86}{100} \approx 24.405\ elementos$$

$$\frac{17.256}{128} \approx 190\ bloques$$

$134 \cdot 128 = 24.320\ elementos$

■# of NBs. Números distintos de tamaño de bloque que se van a probar. En este caso uno.

■NBs. Tamaño del bloque (es decir, de las sub-matrices en las que se particiona la matriz original). Este valor lo elige el usuario y suele ser habitual fijar potencias de dos: 128, 224, 256. . .

■PMAP process mapping. Indica si la matriz a resolver está almacenada por filas o por columnas.

■# of process grids (P x Q). En LINPACK, el número de procesos a utilizar en la resolución del sistema se organiza en una cuadrícula de P filas por Q columnas. Este parámetro define el número de combinaciones P,Q que vamos a probar. En el ejemplo, uno.

■Ps. Número de procesos en filas.

■Qs. Número de procesos en columnas.

■threshold. Valor con el cual se compara el residuo de la operación.

■# of panel fact. Número de factorizaciones de panel a realizar.

■PFACTs (0=left, 1=Crout, 2=Right). Tipo de factorizaciones del panel.

■# of recursive stopping criterium. Número de NBMINs a utilizar.

■NBMINs (>= 1). Número máximo de columnas del panel a factorizar.

■# of recursive panel fact. Número de factorizaciones recursivas del

panel.

- RFACTs (0=left, 1=Crout, 2=Right). Tipo de factorizaciones recursivas del panel.
- # of broadcasts. Número de topologías distintas que se utilizan para hacer la difusión del panel actualmente factorizado.
- BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM). Elección de las topologías utilizadas para la difusión.
- DEPTHs (>=0) Indica la profundidad del *look-ahead* aplicado. En este caso, 1 significa que el siguiente panel se factoriza justo después de haber sido actualizado.

Respecto a los valores de P y Q, de acuerdo con la documentación oficial del *benchmark*, se recomienda que sean lo más parecidos posible (con Q ligeramente mayor que P) y se debe garantizar que el producto $P\,Q$ sea igual al número total de núcleos a utilizar.

Como los parámetros de configuración resultan un tanto difíciles de calcular en ocasiones, existen diferentes herramientas web que facilitan esta tarea, por ejemplo:

- How do I tune my HPL.dat? [10]
- HPL-calculator [11]

En el ejemplo mostrado, se ha utilizado el resultado devuelto por HPL-calculator para una ocupación del 86% de la memoria con un bloque de 128.

Una vez modificado el fichero de configuración, la ejecución del test HPL (ubicado en */SHARED/hpl/bin/rpi*) solo necesita la especificación del número de núcleos que se van a utilizar y el nombre de los nodos que los contienen. En el caso concreto del clúster RPi, con tres nodos de cómputo y cuatro núcleos cada uno, bastará con ejecutar los pasos descritos a continuación:

```
pi@nodo0:~ $ cd /SHARED/hpl/bin/rpi
pi@nodo0:/SHARED/hpl/bin/rpi $ mpirun -n 12 --host nodo1,nodo2,nodo3 ./xhpl
```

Una posible salida del comando anterior podría ser la que se muestra en la Figura 7.2. Al inicio se muestra información sobre HPL y sus autores, así como una descripción de los parámetros de entrada/salida (omitidas en la figura por brevedad). A continuación, se muestra la configuración de los parámetros usados a partir de lo indicado en el fichero *HPL.dat*. Finalmente, se muestra el resultado de los tests que indica el tiempo de ejecución (en segundos) y los GFLOPS. Nótese que el rendimiento alcanzado por el clúster RPi es de 13,684 GFLOPS, mientras que el rendimiento alcanzado por el primer supercomputador de la lista TOP500 de noviembre de 2020 es de 442.010 TFLOPS. Esto implica que el primer supercomputador de la mencionada lista alcanza un rendimiento $442.010 \cdot 103/13{,}684 \approx 32.301.228$ veces superior al del prototipo RPi.

```
The following parameter values will be used:

N      :   24320
NB     :     128
PMAP   : Row-major process mapping
P      :       3
Q      :       4
PFACT  :   Right
NBMIN  :       4
NDIV   :       2
RFACT  :   Crout
BCAST  :  1ringM
DEPTH  :       1
SWAP   : Mix (threshold = 64)
L1     : transposed form
U      : transposed form
EQUIL  : yes
ALIGN  : 8 double precision words

--------------------------------------------------------------------------------

- The matrix A is randomly generated for each test.
- The following scaled residual check will be computed:
||Ax-b||_oo / ( eps * ( || x ||_oo * || A ||_oo + || b ||_oo ) * N )
- The relative machine precision (eps) is taken to be   1.110223e-16
- Computational tests pass if scaled residuals are less than    16.0

================================================================================
T/V                N    NB     P     Q               Time         Gflops
--------------------------------------------------------------------------------
WR11C2R4       24320   128     3     4             700.87     1.3684e+01
HPL_pdgesv() start time Tue Dec 29 12:12:16 2020

HPL_pdgesv() end time   Tue Dec 29 12:23:57 2020

--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)= 9.53448e-04 ...PASSED
================================================================================

Finished      1 tests with the following results:
              1 tests completed and passed residual checks,
              0 tests completed and failed residual checks,
              0 tests skipped because of illegal input values.
--------------------------------------------------------------------------------
End of Tests.
```

Figura 7.2 Posible salida por pantalla tras ejecutar xhpl.

7.2 Aplicaciones científicas

En este apartado se van a descargar, compilar y ejecutar dos aplicaciones científicas utilizadas en centros de supercomputación de todo el mundo: LAMMPS y OpenFOAM. El objetivo es presentar software complejo y de alta relevancia en el contexto de la HPC. Adicionalmente, podrá comprobarse que los distintos procesos de instalación guardan cierta similitud entre las distintas aplicaciones.

7.2.1 LAMMPS

LAMMPS[12] (*Large-scale Atomic/Molecular Massively Parallel Simulator*) es un simulador de dinámica molecular. La complejidad de los cálculos a realizar para completar las simulaciones convierte a LAMMPS en una aplicación que requiere ser ejecutada en un contexto de supercomputación.

A continuación se muestra cómo descargar y compilar LAMMPS (versión *stable_29Oct2020*) en el directorio compartido del clúster; concretamente, se compila la versión distribuida que utiliza MPI empleando tres procesadores:

```
COMENTARIO: Cargar MPICH en las variables de entorno.
pi@nodo0:~ $ export PATH=/SHARED/mpich-3.3.2-install/bin:$PATH

COMENTARIO: Descargar LAMMPS
pi@nodo0:~ $ cd /SHARED
pi@nodo0:/SHARED $ wget https://github.com/lammps/lammps/
    archive/ble_29Oct2020.zipzip
pi@nodo0:/SHARED $ unzip stable_29Oct2020.zip
pi@nodo0:/SHARED $ rm stable_29Oct2020.zip
```

```
COMENTARIO: Compilar la versión MPI de LAMMPS
pi@nodo0:/SHARED $ cd lammps-stable_29Oct2020/src
pi@nodo0:/SHARED/lammps-stable_29Oct2020/src $ make mpi -j4
```

Una vez terminada la compilación, se habrá creado el fichero ejecutable lmp_mpi en el directorio */SHARED/lammps/src/*. Las instrucciones que siguen permiten lanzar el ejecutable generado. Al ejecutar lmp_mpi, SIZE indica el tamaño del dominio tridimensional, en este caso, con $SIZE = 4$ el dominio es $SIZE3 = 43$, e *in.lj* es el conjunto de datos inicial del cual se parte para llevar a cabo la simulación. La Figura 7.3 muestra el resultado de la ejecución de los siguientes comandos:

```
pi@nodo0:/SHARED/lammps-stable_29Oct2020/src $ export SIZE=4
pi@nodo0:/SHARED/lammps-stable_29Oct2020/src $ mpirun -n 2
    -host nodo1,nodo2 ./lmp_mpi -var x $SIZE -var y $SIZE -var z
    $SIZE -in ../bench/in.lj
```

Utilizar más recursos computacionales para resolver el mismo problema (técnica denominada *strong scaling* en el contexto computacional) será tan sencillo como indicarlo a la hora de lanzar la ejecución. Mediante las instrucciones anteriores se han utilizado dos procesos ("-n 2") y dos nodos ("-host nodo1,nodo2"), mientras que, usando las que siguen, se utilizan nueve procesos ("-n 9") y tres nodos ("-host nodo1,nodo2,nodo3"). De este modo, en el primer caso se ejecuta un proceso por nodo, y en el segundo, tres procesos por nodo.

```
pi@nodo0:/SHARED/lammps-stable_29Oct2020/src $ mpirun -n 9
```

```
-host nodo1,nodo2,nodo3 ./lmp_mpi -var x $SIZE -var y $SIZE -var z
$SIZE -in ../bench/in.lj
```

Si se quiere utilizar hilos a la vez que procesos para la ejecución en paralelo de LAMMPS MPI, se deberá compilar la versión OpenMP de LAMMPS[13]. Para conservar los binarios anteriores, se realizará una copia del fichero *Makefile.mpi* (ubicado en el directorio *src/MAKE/*), por ejemplo, con el nombre *Makefile.mpiomp*, tal como sigue:

```
pi@nodo0:/SHARED/lammps-stable_29Oct2020/src $ cp MAKE/
Makefile.mpi MAKE/Makefile.mpiomp
```

En este nuevo fichero habrá que indicar que se va a utilizar OpenMP para el paralelismo intra-nodo, del siguiente modo:

Fichero:
/SHARED/lammps-stable_29Oct2020/src/MAKE/Makefile.mpiomp

```
CCFLAGS= –g –O3 –fopenmp
LINKFLAGS= –g –O3 –fopenmp
```

```
LAMMPS (29 Oct 2020)
Lattice spacing in x,y,z = 1.6795962 1.6795962 1.6795962
Created orthogonal box = (0.0000000 0.0000000 0.0000000) to (134.36770
     134.36770 134.36770)
  1 by 1 by 2 MPI processor grid
Created 2048000 atoms
  create_atoms CPU = 0.417 seconds
Neighbor list info ...
  update every 20 steps, delay 0 steps, check no
  max neighbors/atom: 2000, page size: 100000
  master list distance cutoff = 2.8
  ghost atom cutoff = 2.8
  binsize = 1.4, bins = 96 96 96
  1 neighbor lists, perpetual/occasional/extra = 1 0 0
  (1) pair lj/cut, perpetual
      attributes: half, newton on
      pair build: half/bin/atomonly/newton
      stencil: half/bin/3d/newton
      bin: standard
Setting up Verlet run ...
  Unit style    : lj
  Current step  : 0
  Time step     : 0.005
Per MPI rank memory allocation (min/avg/max) = 314.3 | 314.3 | 314.3 Mb
Step Temp E_pair E_mol TotEng Press
       0         1.44   -6.7733681  0   -4.6133691   -5.0196699
     100   0.75970574   -5.7618658  0   -4.6223078   0.18850941
Loop time of 227.091 on 2 procs for 100 steps with 2048000 atoms
Performance: 190.232 tau/day, 0.440 timesteps/s
98.6% CPU use with 2 MPI tasks x no OpenMP threads
MPI task timing breakdown:
Section |  min time  |  avg time  |  max time  |%varavg| %total
---------------------------------------------------------------
Pair    | 187.68     | 188.12     | 188.55     |   3.2 | 82.84
Neigh   | 22.101     | 22.129     | 22.157     |   0.6 |  9.74
Comm    | 7.1902     | 7.6495     | 8.1088     |  16.6 |  3.37
Output  | 0.011756   | 0.013343   | 0.01493    |   1.4 |  0.01
Modify  | 8.1339     | 8.1936     | 8.2533     |   2.1 |  3.61
Other   |            | 0.9894     |            |       |  0.44
Nlocal:    1.02400e+06 ave 1.02406e+06 max 1.02394e+06 min
Histogram: 1 0 0 0 0 0 0 0 0 1
Nghost:       186832.0 ave      186877 max      186787 min
Histogram: 1 0 0 0 0 0 0 0 0 1
Neighs:    3.84553e+07 ave 3.84984e+07 max 3.84121e+07 min
Histogram: 1 0 0 0 0 0 0 0 0 1
Total # of neighbors = 76910507
Ave neighs/atom = 37.553958
Neighbor list builds = 5
Dangerous builds not checked
Total wall time: 0:03:55
```

Figura 7.3 Ejemplo del resultado de ejecutar LAMMPS.

Finalmente, y una vez en el directorio */lammps/src/*, se procede a la compilación de la aplicación del mismo modo que previamente, pero ahora indicando que se va a utilizar OpenMP. De las instrucciones que siguen, las dos primeras instrucciones sirven para, respectivamente, indicar que se va a utilizar OpenMP y realizar la compilación. El último comando ejecuta dicha versión compilada de LAMMPS. Aparte de utilizar el comando MPI para ejecutar LAMMPS en distintos procesos, con el parámetro "-sf omp" se indica que se utilice la biblioteca OpenMP, para evitar así modificar el fichero de entrada (*in-lj*). Además, con "-pk omp" se especifica el número de hilos OpenMP por proceso MPI.

```
pi@nodo0:/SHARED/lammps-stable_29Oct2020/src $ make yes-user-
    omp
pi@nodo0:/SHARED/lammps-stable_29Oct2020/src $ make mpiomp
    -j4
pi@nodo0:/SHARED/lammps-stable_29Oct2020/src $ mpirun -n 2
    -host nodo1,nodo2 ./lmp_mpiomp -var x $SIZE -var y $SIZE -var z
    $SIZE -in ../bench/in.lj -sf omp -pk omp 2
```

Otro ejemplo del uso de LAMMPS MPI-OpenMP se encuentra a continuación. De igual modo que se ha visto anteriormente, mediante los *flags* "-n" y "-host" es posible indicar el número de procesos y nodos a usar en la ejecución, respectivamente. Así, en la ejecución anterior se usan dos procesos y dos nodos (un proceso por nodo), mientras que en la que sigue se utilizan seis procesos y tres nodos (dos procesos por nodo). A su vez, en la anterior se generan cuatro hilos OpenMP, mientras que en la siguiente se crean dos

hilos.

```
pi@nodo0:/SHARED/lammps-stable_29Oct2020/src $ mpirun -n 6
    -host nodo1,nodo2,nodo3 ./lmp_mpiomp -var x $SIZE -var y $SIZE
    -var z $SIZE -in ../bench/in.lj -sf omp -pk omp 4
```

7.2.2 OpenFOAM

OpenFOAM es un software de código abierto para la simulación de la dinámica de fluidos (*Computational Fluid Dynamics* [CFD], en inglés). OpenFOAM es un referente mundial y se utiliza tanto en entornos empresariales como académicos de distintas áreas de ciencia e ingeniería (ver Figura 7.4). OpenFOAM se compone de multitud de módulos que le permiten resolver complejos problemas hidrodinámicos en reacciones químicas, turbulencia y transferencia de calor, acústica, sólidos o electromagnetismo.

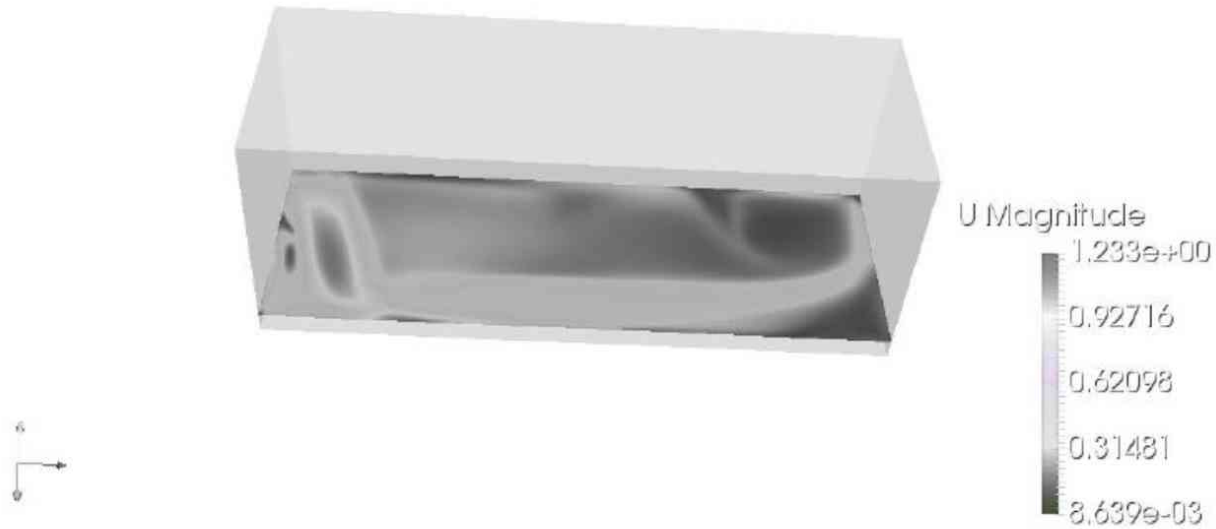


Figura 7.4 Representación de una simulación con OpenFOAM.

Para proceder a la instalación de OpenFOAM (versión 19.12), se instalarán las dependencias y se descargará y descomprimirá el código fuente de OpenFOAM tal como se muestra a continuación:

```
pi@nodoo:~ $ sudo apt-get install -y flex
pi@nodoo:~ $ cd /SHARED/
pi@nodoo:/SHARED $ wget https://sourceforge.net/
    pro-jects/open-plus/plus/files/v1912/FOAM-v1912_200312.-tgz
pi@nodoo:/SHARED $ tar -xvf OpenFOAM-v1912_200312.tar.gz
pi@nodoo:/SHARED $ rm OpenFOAM-v1912_200312.tar.gz
```

En este manual, para la configuración de la instalación de Open-FOAM se utilizará la implementación de MPICH instalada en el Apartado 5.2.2. Así pues, el fichero *.bashrc* debería contener las siguientes variables declaradas para que OpenFOAM pueda cargar las bibliotecas necesarias:

Fichero: /home/pi/.bashrc

```
export MPI_ROOT=/SHARED /mpich –3.3.2 –install
export MPI_ARCH_FLAGS="–DOMPI_SKIP_MPICXX "
export MPI_ARCH_INC="–isystem $MPI_ROOT / include "
export MPI_ARCH_LIBS="–L$MPI_ROOT /lib –lmpi "
export PATH=$MPI_ROOT /bin : $PATH
export LD_LIBRARY_PATH=$MPI_ROOT /lib : $LD_LIBRARY_PATH
```

También hay que indicar a OpenFOAM qué biblioteca MPI tiene que utilizar. Para ello, se realizará la modificación que sigue:

Fichero: /SHARED/OpenFOAM-v1912/etc/bashrc

```
export WM_MPLIB=SYSTEMMPI
```

Adicionalmente, hay que modificar la forma en que el compilador creará las instrucciones de coma flotante (parámetro "-mfloat_abi") para ajustarla a las particularidades de la arquitectura de los dispositivos RPi. Para ello se modificará la opción "-mfloat_abi=softfp", que pasará a ser "-mfloat_abi=hard" en las variables cOPT y c++OPT de los ficheros siguientes. El fichero *cOpt* deberá contener en la variable cOPT el siguiente valor:

Fichero: /SHARED/OpenFOAM-v1912/wmake/rules/linu-M7Gcc/cOptcOpt

```
cOPT = –O3 –floop –optimize –falign –loops –falign–labels -falign –functions –falign –jumps –fprefetch –loop –arrays –mfpu=vfpv3 d16 –mfloat –abi=hard
```

Con este mismo objetivo, habrá que modificar también el siguiente fichero, tal como se muestra a continuación:

Fichero: /SHARED/OpenFOAM-v1912/wmake/rules/linu-M7Gcc/c++Optc++Opt

```
c++OPT = –O3 –floop–optimize –falign–loops –falign–labels –falign–functions –falign–jumps –fprefetch–loop–arrays –mfpu=vfpv3–d16 –mfloat–abi=hard
ROUNDING_MATH = –frounding–math
```

Una vez terminada la configuración, se exportarán de nuevo todas las variables de entorno definidas en los ficheros modificados:

```
pi@nodoo:~ $ source $HOME/.bashrc
pi@nodoo:~ $ source /SHARED/OpenFOAM-v1912/etc/bashrc
No completion added for
/SHARED/OpenFOAM-v1912/platforms/
      linu-xAR-M7GccDPIn-t32Opt/
... incorrect platform, or not yet compiled?
```

El error que se muestra al cargar las variables de entorno de Open-FOAM es esperable, ya que aún no se ha compilado.

El siguente paso a realizar es precisamente la compilación de Open-FOAM. La siguiente consola muestra cómo iniciar la compilación utilizando únicamente un procesador:

```
pi@nodoo:~ $ cd /SHARED/OpenFOAM-v1912
pi@nodoo:/SHARED/OpenFOAM-v1912 $ ./Allwmake -j1
```

La utilización de un solo procesador (opción "-j1" del comando Allwmake) para compilar OpenFOAM, es una recomendación de los autores para el entorno RPi utilizado en esta obra. Durante la preparación de este libro se detectó que utilizar más procesos conllevaba que inevitablemente el nodo dejara de responder, lo cual obligaba a reiniciar el dispositivo, si bien iniciar de nuevo la compilación no implica empezar de cero, sino que se retoma donde se dejó. En conclusión, debido a las limitaciones de memoria principal y anchos de banda para lectura y escritura en disco en la plataforma RPi, la

opción de compilación en serie es la más factible en este contexto, pese a que requiere más de 10 horas para realizarse por completo. Además, es posible que en ciertos dispositivos con una capacidad de memoria principal menor o igual a 1 GB sea necesario deshabilitar en el *front-end* el entorno gráfico y reducir la cantidad de memoria destinada a la GPU, tal y como se sugiere en el Apartado 4.1.1. Con esta operación se consigue aumentar la cantidad de memoria principal disponible, lo que resulta imprescindible para esta compilación.

Una vez compilado, se cargarán las variables de entorno con las rutas correspondientes generadas durante la instalación. También se comprobará que OpenFOAM funciona correctamente mediante uno de los muchos tutoriales que incorpora el paquete de instalación de OpenFOAM. En concreto, se ha seleccionado el "Flujo turbulento constante sobre un escalón orientado hacia atrás"[14], ubicado en */SHARED/OpenFOAM-v1912/tutorials/incompressible/simpleFoam/pitzDaily*. Queda fuera del objetivo de este libro tratar la configuración de las simulaciones, por tanto, se utilizará la establecida por defecto. Antes de lanzar la simulación, se tiene que generar la malla del objeto a simular, la cual corresponderá al dominio (conjunto de datos que forma el problema) del caso. Para ello se utiliza el comando blockMesh. Finalmente, la simulación se inicia con simpleFoam.

El procedimiento completo, junto con las últimas líneas de las salidas que se generan en cada comando, se encuentra en la siguiente consola:

```
pi@nodo0:~ $ source /SHARED/OpenFOAM-v1912/etc/bashrc
pi@nodo0:~ $ cd /SHARED/OpenFOAM-v1912/tutorials/
    incom-pres-sible/pleFoam/pitz-Daily
```

```
pi@nodo0:/SHARED/OpenFOAM-v1912/tutorials/incompressible/simpleFoam/pi
        $ blockMesh
...
---------------
Patches
---------------
    patch 0 (start: 24170 size: 30) name: inlet
    patch 1 (start: 24200 size: 57) name: outlet
    patch 2 (start: 24257 size: 223) name: upperWall
    patch 3 (start: 24480 size: 250) name: lowerWall
    patch 4 (start: 24730 size: 24450) name: frontAndBack
End
pi@nodo0:/SHARED/OpenFOAM-v1912/tutorials/incompressible/simpleFoam/pi
        $ simpleFoam
...
SIMPLE solution converged in 279 iterations
streamLine streamlines write:
    seeded 10 particles
    Tracks:10
    Total samples:10885
    Writing data to "./postProcessing/sets/streamlines/279"
End
```

Esta ejecución generará los directorios *100*, *200* y *279*, los cuales contendrán el estado de la simulación para esos instantes de tiempo.

No obstante, esta simulación se está ejecutando secuencialmente en un único procesador. Tal y como se ha hecho con LAMMPS en el Apartado 7.2.1, uno de los objetivos principales de poner en marcha un clúster es la ejecución paralela de aplicaciones, de modo que puedan aprovecharse todos los

recursos disponibles. Para la ejecución paralela de simulaciones en OpenFOAM, inicialmente es necesario dividir el dominio entre los procesadores participantes en la ejecución. Esta operación es equivalente a "repartir" el caso a simular entre los distintos procesadores involucrados en la ejecución. OpenFOAM ofrece para ello una serie de comandos con los que descomponer y reconstruir el dominio antes y después de la simulación, es decir, para repartir y recolectar los datos, respectivamente. Concretamente, el comando decomposePar es el encargado de dividir el dominio. Este comando lee la configuración del fichero *system/decomposeParDict* ubicado en el propio directorio del caso. Por ejemplo, se podría utilizar el siguiente fichero:

Fichero: ./system/decomposeParDict

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      decomposeParDict;
}
numberOfSubdomains  4;
method  simple;
coeffs {n   (1 2 2);}
```

Este es un fichero típico de configuración de OpenFOAM, donde se puede ver que la primera parte (FoamFile) define el tipo de fichero y el resto de variables se asocian a la configuración de la descomposición del dominio. En este ejemplo, se ha especificado que se divida el dominio en cuatro partes (numberOfSubdomains) utilizando el método simple y distribuyendo los procesos con una topología 1-2-2 (valores asociados a las coordenadas x, y y z, respectivamente).

Antes de empezar la simulación paralela, se borrarán los resultados de la simulación anterior y se descompondrá el dominio. A continuación, se

muestran los comandos para realizar las operaciones descritas, a la vez que las últimas líneas de una posible salida del comando de descomposición.

```
pi@nodo0:/SHARED/OpenFOAM-v1912/tutorials/incompressible/simpleFoam/pi
    $ rm -rf 100 200/ 279/ postProcessing/
pi@nodo0:/SHARED/OpenFOAM-v1912/tutorials/incompressible/simpleFoam/pi
    $ decomposePar
...
Number of processor faces = 329
Max number of cells = 3349 (9.57873% above average 3056.25)
Max number of processor patches = 2 (0% above average 2)
Max number of faces between processors = 169 (2.73556% above average
    164.5)
Time = 0
Processor 0: field transfer
Processor 1: field transfer
Processor 2: field transfer
Processor 3: field transfer
End
```

Para lanzar la simulación, se deberá indicar qué nodos participarán y con cuántos procesadores. La siguiente consola muestra cómo hacerlo empleando cuatro procesos (así se ha descompuesto el dominio anteriormente), con dos procesos en cada uno de los dos nodos de cómputo asignados:

```
pi@nodo0:/SHARED/OpenFOAM-v1912/tutorials/incompressible/simpleFoam/pi
    $ mpirun -n 4 --host nodo1,nodo2 simpleFoam -parallel
```

```
SIMPLE solution converged in 290 iterations
streamLine streamlines write:
    seeded 10 particles
    Tracks:10
    Total samples:10885
    Writing data to "./postProcessing/sets/streamlines/290"
End
Finalising parallel run
```

Finalmente, se procederá a recomponer el dominio con el comando reconstructPar, el cual creará la información para los instantes de tiempo configurados en la simulación. Aunque la simulación ya haya terminado en este punto, este paso resulta esencial para el posterior estudio y análisis de los resultados calculados.

```
pi@nodo0:/SHARED/OpenFOAM-v1912/tutorials/incompressible/simpleFoam/pi
    $ reconstructPar
Time = 290
Reconstructing FV fields
    Reconstructing volScalarFields
        epsilon
        k
        nut
        p
    Reconstructing volVectorFields
        U
    Reconstructing surfaceScalarFields
        phi
```

```
Reconstructing point fields
No point fields
No lagrangian fields
No FA fields
End
```

7.3 Para profundizar...

1. Ya se ha visto que las operaciones para poner en marcha las aplicaciones guardan cierta similitud:

- descargar,
- descomprimir,
- configurar,
- compilar,
- instalar y
- ejecutar.

¿Podría buscar una aplicación e instalarla utilizando su guía de usuario?. Por ejemplo: GROMACS[15], NAMD[16] o E3SM[17].

2. Los ejemplos de ejecución mostrados en este capítulo no utilizan el gestor de trabajos instalado en 6.1.1. ¿Podría crear sus propios *scripts* para el envío de trabajos LAMMPS u OpenFOAM a Slurm?

3. La descomposición del dominio en el caso visto de OpenFOAM es una tarea secuencial. Por tanto, ¿sería razonable pedir recursos a Slurm mediante un trabajo que ejecutara la simulación paralela y a su vez incluyera también la tarea de descomposición y reconstrucción? Por ejemplo, si se utilizara el siguiente *script* para lanzar un trabajo OpenFOAM:

Fichero: /home/pi/launch_openfoam.sbatch

```
#!/bin/ bash
#SBATCH --nodes 2
#SBATCH --tasks–per–node 4
```

```
cd / SHARED / OpenFOAM–v1912 / tutorials / incompressible / simple-
    Foam / pitzDaily
nodelist=$ ( scontrol show hostname $SLURM_JOB_NODELIST | paste –d ,
    –s )
  decomposePar
mpirun –n $SLURM_NTASKS –host $nodelist simpleFoam –parallel
  reconstructPar
  cd
```

¿Se desperdiciarían recursos en algún punto de la ejecución del trabajo?

7.4 Material recomendado

Las herramientas y aplicaciones presentadas en este capítulo son solo una muestra de la gran variedad de software HPC, centrado, o bien en conocer/caracterizar el sistema, o bien en su explotación para la realización de cómputo. El libro [1] y el artículo de investigación [2] presentan también muestras alternativas de ello.

Respecto a las utilidades, *benchmarks* y aplicaciones vistos en este capítulo, se ha presentado una breve descripción de los parámetros y opciones usados con más frecuencia, así como configuraciones que pueden servir de punto de partida para explorar más a fondo el clúster RPi. Para revisar los detalles sobre la instalación de OpenFOAM en el sistema operativo propuesto en este libro se recomienda la lectura de las guías presentes en los enlaces web [3], [4] y [5]. Del mismo modo, pueden consultarse los sitios web [6] y [7] para afianzar lo relativo a la ejecución paralela de dicha aplicación. En lo que a LAMMPS respecta, en el sitio web oficial de la aplicación [8] es donde se presenta detalladamente su documentación. De un modo análogo, puede consultarse el enlace web [9] para la leer el material oficial relativo a la configuración de HPL.

[1] ASC Community. *The Student Supercomputer Challenge Guide* . Springer Singapore, 2018.

[2] Michael A. Heroux and C. Kristopher Garrett. Special issue on sc16 student cluster competition reproducibility initiative. *Parallel Computing* , 70:3 – 4, 2017. SC16 Student Cluster Competition Reproducibility Initiative.

[3] OpenFoamWiki. Installation/Linux/OpenFOAM-dev/Raspbian (en inglés).

https://openfoamwiki.net/index.php/Insta-lla-tion/Linux/FOAM-dev/Ras-pbian . Accedido por última vez: nov. 2020.

[4] MICHIEL'S WEBLOG. OpenFOAM on a Raspberry Pi cluster (en inglés). https://blog.mvbakker.nl/posts/openfoam-on-raspicluster . Accedido por última vez: nov. 2020.

[5] OpenCFD Ltd. OpenFOAM Quick Build Guide (en inglés). https://develop.openfoam.com/Development/openfoam/blob/maintenance-v1912/doc/Buil-d.md . Accedido por última vez: nov. 2020.

[6] OpenFoam. Running applications in parallel (en inglés). https://openfoam.com/documentation/user-guide/running-appli-tions-para-llel.php . Accedido por última vez: nov. 2020.

[7] OpenFoam. OpenFoam User Guide v2006 (en inglés). https://www.openfoam.com/documentation/guides/latest/doc/openfoam-guide-para-l.htmlml . Accedido por última vez: nov. 2020.

[8] Sandia. LAMMPS website (en inglés). https://lammps.sandia.gov/ci-te.ht- . Accedido por última vez: nov. 2020.

[9] Netlib. HPL Tuning (en inglés). https://www.netlib.org/bench-mark/hpl/tunin-mlml . Accedido por última vez: nov. 2020.

7.5 Resumen

En este capítulo se han presentado distintas herramientas y utilidades que permiten tanto caracterizar el sistema como conocer su estado actual. Se han descrito brevemente stress-ng y stream, los cuales permiten conocer el rendimiento que puede proporcionar el hardware subyacente. También se han visto las herramientas htop y cpufrequtils, que permiten conocer la carga computacional del sistema en un momento dado y la configuración de frecuencia de los núcleos, respectivamente. Además, mediante la ejecución del *benchmark* LINPACK, se ha evaluado el rendimiento del clúster RPi.

Por otro lado, la segunda parte del capítulo presenta dos aplicaciones científicas ampliamente utilizadas actualmente en el ámbito de la HPC para realizar simulaciones: LAMMPS (utilizada en dinámica molecular) y OpenFOAM (perteneciente al campo de la dinámica de fluidos).

7.5.1 ¿Qué viene a continuación?

El siguiente capítulo se centra en el software de virtualización y en cómo este puede ser utilizado en el ámbito de la supercomputación. Por un lado, se describirá de manera detallada qué son los contenedores, así como todo su ecosistema: creación, ejecución e interacción con los repositorios que los contienen. Para ello se utilizará la tecnología de contenedores Singularity. Por otro lado, se llevará a cabo la creación de varios contenedores plenamente funcionales.

1 https://linux.die.net/man/1/cpufreq-info

2 https://linux.die.net/man/1/cpufreq-set

3 https://github.com/ColinIanKing/stress-ng

4 https://linuxx.info/stress-ng

5 https://manpages.ubuntu.com/manpages/artful/man1/stress-ng.1.html

6 https://htop.dev.

7 http://www.cs.virginia.edu/stream/ref.html.

8 https://iperf.fr

9 https://www.netlib.org/benchmark/hpl/.

10 https://www.advancedclustering.com/act_kb/tune-hpl-dat-file/

11 http://hpl-calculator.sourceforge.net/

12 https://lammps.sandia.gov/

13 https://lammps.sandia.gov/doc/Build_extras.html#user-omp

14 https://openfoam.com/documentation/tutorial-guide/tutorialse5.php

15 http://manual.gromacs.org/documentation

16 https://www.ks.uiuc.edu/Research/namd

17 https://github.com/E3SM-Project

8. Software de virtualización

En ocasiones, es necesario o conveniente emular un entorno informático (total o parcialmente) dentro de otro físico diferente. Para esto, se recurre a las tecnologías de virtualización. El entorno informático donde se ejecuta el sistema virtualizado es conocido como máquina anfitrión (*host*, en inglés). La virtualización puede realizarse a nivel de entornos totalmente configurados; por ejemplo, en algunas universidades o instituciones, cada usuario dispone de un entorno informático adecuado a sus necesidades, disponible para su utilización incluso desde fuera de la propia institución. Además, también es posible virtualizar las redes, el almacenamiento o solamente algunas aplicaciones.

Hasta los últimos años, el software de virtualización no había sido utilizado en el ámbito de la supercomputación. Esto se debe a que, hasta la aparición de los contenedores, la virtualización únicamente se realizaba mediante la utilización de máquinas virtuales. Cuando se ejecuta una máquina virtual (también llamada huésped) en el recurso anfitrión, en ella se emula tanto el hardware como el software. Por lo tanto, es posible tener un sistema operativo en la máquina virtual completo y totalmente distinto al de la máquina

anfitrión. Esto conlleva que la ejecución de una aplicación dentro de una máquina virtual, en lugar de directamente sobre los nodos de supercomputación, consuma mayor cantidad de recursos, dado que es necesario simular el entorno informático completo. Pese a que las máquinas virtuales ofrecen mayor aislamiento entre los distintos procesos o una mayor personalización por parte del usuario del entorno de ejecución, esto no es suficiente para que las máquinas virtuales se utilicen en el ámbito de la supercomputación. En cambio, la utilización de los entornos virtualizados es la base de la computación en la nube, donde los recursos físicos son "ilimitados" y los usuarios deben poder crear o destruir nodos de procesamiento bajo demanda para, por ejemplo, dar respuesta a nuevas cargas de trabajos o para ahorrar costes eliminando estos nuevos recursos una vez ya no son necesarios. Gracias al auge de las tecnologías de contenedores, la virtualización ha comenzado a extenderse al ámbito de la supercomputación. Por ejemplo, el supercomputador más potente según la lista TOP500 en noviembre de 2020, Fugaku, utiliza la tecnología de contenedores Singularity[1]. Si bien el aumento de la popularidad de las tecnologías de contenedores fue propiciado en 2013 con el lanzamiento de Docker[2], el concepto de contenedor está basado en las jaulas de FreeBSD[3], las cuales fueron introducidas en el sistema operativo FreeBSD en el año 2000.

La utilización de contenedores permite a los usuarios realizar la portabilidad tanto de sus aplicaciones como de un entorno totalmente configurado entre distintas infraestructuras de una manera sencilla. Normalmente, una aplicación no siempre va a ser ejecutada dentro del entorno de desarrollo. En ciertos niveles del desarrollo de una aplicación puede ser interesante realizar pruebas en otros entornos (por ejemplo, en el ordenador personal de otra persona) o en el entorno de producción (donde la versión final de la aplicación será utilizada). A lo largo de este libro, se ha visto la importancia de conocer

el rendimiento de las aplicaciones científicas. Para ello, es común que las aplicaciones tengan que moverse de un entorno de desarrollo a entornos con una gran cantidad de recursos computacionales donde poder ver cómo se comporta la aplicación cuando se realizan ejecuciones a gran escala. Otro aspecto importante que propicia el uso de contenedores es el versionado de las aplicaciones. Las tecnologías de contenedores permiten utilizar distintas versiones de un mismo contenedor.

En este capítulo se va a profundizar en las tecnologías de contenedores. En primer lugar, se describe qué son, cuáles son las diferencias que presentan respecto a las máquinas virtuales y algunas definiciones previas necesarias para trabajar con esta tecnología. Después, se proporcionan las instrucciones necesarias para la instalación de la tecnología de contenedores Singularity[4], así como las nociones básicas para la generación y ejecución de contenedores. Una vez presentados estos conceptos, se procede a la generación de un contenedor que disponga de MPI y que será ejecutado a través de Slurm. Finalmente, se reflexiona sobre el uso de los contenedores en entornos HPC.

8.1 ¿Qué son los contenedores?

Antes de definir qué es un contenedor y en qué se diferencia de una máquina virtual, es necesario profundizar en el concepto de máquina virtual. Como se ha dicho antes, una máquina virtual es una emulación completa de los elementos hardware y software dentro de un sistema físico. La gestión de las máquinas virtuales en los anfitriones es realizada por un software llamado hipervisor. El hipervisor[5] es el encargado de asignar los distintos recursos computacionales del sistema anfitrión a las distintas máquinas virtuales de manera eficiente, así como de gestionar las operaciones realizadas en el hardware emulado para llevarlas a cabo en el hardware físico. Algunos ejemplos de hipervisorees son Microsoft Hyper-V[6], Oracle VirtualBox[7] y VMWare Workstation[8].

Comúnmente, los contenedores son llamados "máquinas virtuales ligeras", aunque no emulan totalmente el entorno como las máquinas virtuales ni son gestionados por un hipervisor. Los contenedores son una pieza de software ejecutable que contiene una aplicación o servicio junto con sus bibliotecas y dependencias. Se trata de un tipo de tecnología de virtualización que se apoya en ciertas características del núcleo de Linux para simular la ejecución de distintos procesos dentro de un sistema físico (la máquina anfitrión) en un sistema de ficheros virtualizado de manera aislada. De esta forma, desde dentro del contenedor, no es posible visualizar ni interactuar con el resto de procesos de la máquina anfitrión que no pertenecen al contenedor.

Los contenedores permiten la simulación de entornos computacionales sin el sobrecoste que produce el hipervisor en el sistema anfitrión (ver Figura 8.1). Desde la perspectiva de los usuarios, los contenedores son controlados por un software llamado *Container Runtime*, el cual es el encargado de interactuar con el núcleo del SO de la máquina anfitrión para completar cada comando ejecutado por el usuario. Debido a que los contenedores no simulan

el hardware ni el núcleo de Linux, el *Container Runtime* solo se encarga de gestionar los contenedores y es el propio núcleo del sistema operativo el encargado de controlar los procesos y los accesos a los recursos computacionales de los contenedores.

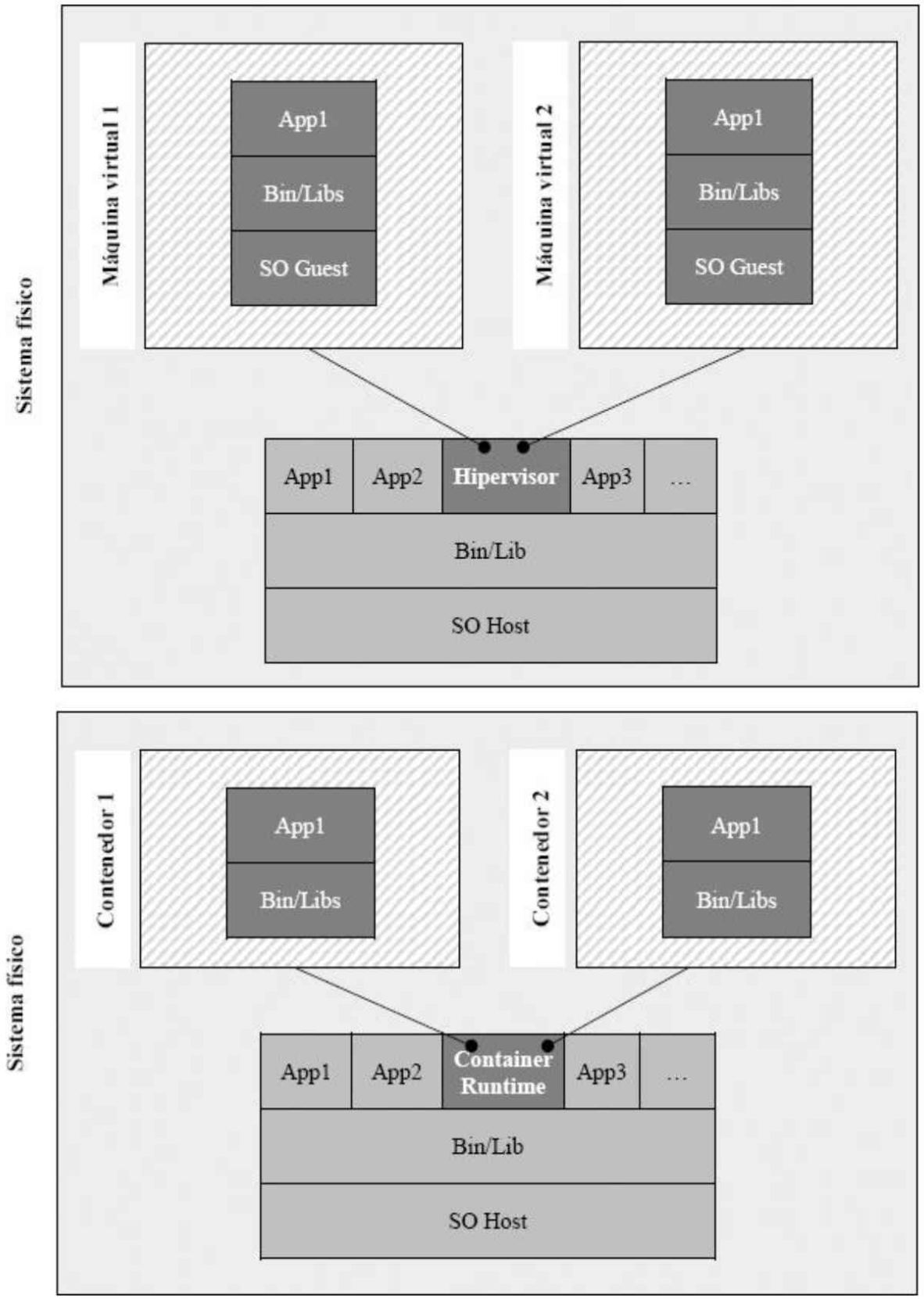


Figura 8.1 Diferencias entre máquinas virtuales (arriba) y contenedores (abajo). Las abreviaturas *App*, *Bin* y *Lib* hacen referencia a, respectivamente,

aplicaciones, binarios y bibliotecas.

Una imagen de contenedor incluye el sistema de ficheros inmutable que contiene todas las aplicaciones, herramientas, dependencias, bibliotecas y demás configuraciones necesarias para permitir el correcto funcionamiento de la aplicación a ejecutar. Las imágenes se almacenan en el repositorio de imágenes o *Container Registry*, el cual puede ser de acceso público o privado y, a su vez, cada imagen puede ser pública o privada. Los repositorios de imágenes más conocidos son Docker Hub[9] y Quay Registry[10]. En el caso de Singularity, existen dos repositorios de imágenes: Sylabs[11] y Singularity Hub[12]. Normalmente, las imágenes se nombran siguiendo una convención que contiene el nombre de la organización o del proyecto, el nombre de la imagen y una versión o etiqueta (*tag*, en inglés):

organización/nombre:*tag*

Por ejemplo, la imagen openfoam/openfoam8-paraview56:latest[13] es una imagen proporcionada por OpenFOAM que contiene OpenFOAM 8 junto con ParaView 5.6, un software para la visualización de datos. En Singularity se ha definido otra manera de nombrar las imágenes dentro de los repositorios, aunque la convención previa sigue siendo válida. Esta nueva manera permite generar colecciones de contenedores:

organización/colección/nombre:*tag*

Por ejemplo, la imagen serlophug/ubuntu16/hola-mundo:ompi-4.0.3[14] es una imagen que contiene el programa *hola_mundo_mpi* y la versión 4.0.3 de OpenMPI proporcionada por el usuario *serlophug* dentro de una colección de

imágenes con Ubuntu 16.04.

La mayoría de las imágenes de los contenedores se construyen a través de una especificación o receta donde se define la imagen base (por ejemplo, un sistema operativo como Ubuntu[15] sin ningún paquete adicional) y un conjunto de instrucciones (por ejemplo, para la instalación de bibliotecas, la definición de variables de entorno o la compilación de la propia aplicación). Tanto la utilización de *tags* como la especificación de la configuración de una imagen a modo de receta han propiciado que los contenedores se posicionen como un método muy utilizado para el envío de aplicaciones entre instituciones o personas.

Cuando un contenedor comienza a ejecutarse, se emula un sistema de ficheros a partir de la imagen junto con la configuración deseada (redes de comunicación, variables de entorno, comandos a ejecutar, montaje de directorios, etc). Las diferentes tecnologías de contenedores ofrecen una amplia variedad de configuraciones, pero, gracias a que la mayoría cumple con los estándares propuestos por la Open Container Initiative[16] (OCI), el aprendizaje de estas tecnologías es sencillo. La OCI es una organización creada en 2015 con el objetivo de definir estándares para todos los componentes que forman el ecosistema de las tecnologías de contenedores: definición de recetas, ejecución y almacenamiento en repositorios. En el Apartado 8.3 se describen algunas de las directrices más comunes utilizadas en los contenedores.

Las tecnologías de contenedores más populares hoy en día son Docker, LxC/LxD[17], Singularity, Podman[18], Charliecloud[19] y Shifter[20]. Probablemente, Docker es la tecnología de contenedores más utilizada actualmente debido a su gran ecosistema de aplicaciones y a que fue la tecnología con la que despegó el uso de los contenedores. Sin embargo, Docker no ha sido adoptado en los entornos HPC debido a que los contenedores son controlados por un demonio que se ejecuta con permisos de administrador. En

cambio, Charliecloud, Shifter, Singularity y Podman fueron diseñados para permitir la ejecución de sus contenedores en el espacio de usuario (sin requerir privilegios de administrador). Singularity es la tecnología de contenedores más utilizada en los entornos HPC debido a que dispone de una gran comunidad de usuarios y un buen ecosistema de herramientas (incluyendo su propio repositorio de imágenes). Por esta razón, se propone utilizarla en este libro.

8.2 Instalación de Singularity

En este apartado, se va a proceder a instalar Singularity (versión 3.5.3) en el directorio compartido del clúster RPi. Para ello, es necesario instalar una serie de paquetes en el *front-end* para la compilación de Singularity, junto con el lenguaje de programación que utiliza el propio Singularity: Go[21] (versión 1.14.12). Aunque Singularity será instalado en el directorio compartido (*/SHARED* en el clúster propuesto en este libro), cada nodo debe disponer de un directorio de Singularity propio o local. El directorio elegido para este fin es /opt/singularity_local, aunque, si el usuario deseara cambiarlo, solamente tendría que modificar la definición de la variable de entorno SINGULARITY_LOCAL_DIR antes de empezar con la instalación.

En primer lugar, se procederá a la instalación de los requisitos de Singularity y Go junto con la definición de unas variables (SHARED_DIR y SINGULARITY_LOCAL_DIR) que definen dónde se realizarán las instalaciones y el directorio local en cada nodo utilizado por Singularity. Esto puede hacerse con los comandos siguientes:

```
pi@nodo0:~ $ sudo apt-get update
pi@nodo0:~ $ sudo apt-get install -y build-essential libssl-dev uuid-dev
    libgpgme11-dev squashfs-tools libseccomp-dev wget pkg-config git
pi@nodo0:~ $ echo 'export SHARED_DIR=/SHARED' >> .bashrc
pi@nodo0:~ $ echo 'export SINGULARITY_LOCAL_DIR=/
    opt/singularity_local' >> .bashrc
```

A continuación, se procederá a la instalación de Go, se añadirá el directorio de instalación a la variable de entorno PATH y se eliminará el fichero comprimido:

```
pi@nodoo:~ $ source .bashrc
pi@nodoo:~ $ export VERSION=1.14.12 OS=linux ARCH=armv6l
pi@nodoo:~ $ wget https://dl.google.com/go/go$-VERSIO-N.$OS-$ARCH.-r.gz
pi@nodoo:~ $ tar -C $SHARED_DIR/ -xzvf go$VERSION.$OS-$ARCH.tar.gz
pi@nodoo:~ $ rm go$VERSION.$OS-$ARCH.tar.gz
pi@nodoo:~ $ echo 'export GOPATH=$SHARED_DIR/go' >> .bashrc
pi@nodoo:~ $ echo 'export PATH=$SHARED_DIR/go:$PATH:$GOPATH/bin' >> .bashrc
pi@nodoo:~ $ source .bashrc
```

Una vez se ha instalado Go en el directorio compartido, es posible la instalación de Singularity. Los siguientes comandos ilustran la descarga y compilación de Singularity. Además, se eliminará el fichero comprimido con el código fuente:

```
pi@nodoo:~ $ export VERSION=3.5.3
pi@nodoo:~ $ wget https://github.com/sylabs/singularity/releases/down-load/v$VER-SION/singu-rity-SION.tar.gzr.gz
pi@nodoo:~ $ tar -xzf singularity-$VERSION.tar.gz
pi@nodoo:~ $ rm singularity-$VERSION.tar.gz
pi@nodoo:~ $ cd singularity
pi@nodoo:~/singularity $ ./mconfig --prefix=$SHARED_DIR/singularity--localstatedir=$SINGULARITY_LOCAL_DIR
pi@nodoo:~/singularity $ make -C ./builddir
pi@nodoo:~/singularity $ sudo make -C ./builddir install
```

El último paso en la instalación de Singularity en el *front-end* es la actualización de la variable de entorno PATH. Dado que es necesario que la creación de una imagen de Singularity se ejecute con permisos de administrador, se modificará la variable para el usuario pi y se realizarán los cambios necesarios para que los usuarios con permiso de administrador (root) puedan ejecutar Singularity. Para ello se modificará el fichero */etc/sudoers* añadiendo el directorio donde se encuentra la instalación de Singularity al final de la variable secure_path con el siguiente texto: ":/SHARED/singularity/bin". La edición de */etc/sudoers* puede realizarse mediante el comando visudo:

```
pi@nodo0:~/singularity $ cd
pi@nodo0:~ $ echo 'export PATH=$SHARED_DIR/
    singularity/bin:$PATH' >> .bashrc
pi@nodo0:~ $ source .bashrc
pi@nodo0:~ $ sudo visudo
```

Tras la instalación en el *front-end*, se han de configurar el resto de nodos (nótese que nodoX del *prompt* hace referencia a los nodos nodo1, nodo2 y nodo3) para que sean capaces de utilizar la instalación de Singularity que se encuentra en el directorio compartido. Para ello basta con actualizar la variable de entorno PATH a partir de las variables de entorno SHARED_DIR y SINGULARITY_LOCAL_DIR. Todo esto se ilustra en los siguientes comandos:

```
pi@nodoX:~ $ echo 'export SHARED_DIR=/SHARED' >> .bashrc
pi@nodoX:~ $ echo 'export SINGULARITY_LOCAL_DIR=/
    opt/singularity_local' >> .bashrc
```

```
pi@nodoX:~ $ echo 'export GOPATH=$SHARED_DIR/go' >> .bashrc
pi@nodoX:~ $ echo 'export PATH=$SHARED_DIR/
    go:$PATH:$GOPATH/bin' >> .bashrc
pi@nodoX:~ $ source .bashrc
pi@nodoX:~ $ sudo mkdir -p $SINGULARITY_LOCAL_DIR/
    singularity/mnt/session/
```

Por último, se recomienda crear un directorio en el espacio compartido, en el cual almacenar todas las imágenes de contenedor generadas.

```
pi@nodoo:~ $ mkdir -p $SHARED_DIR/SingularityImages
```

Después de completar la instalación en todos los nodos, el siguiente paso es comprobar que todo funciona correctamente mediante la ejecución de un contenedor. Para ello, es necesario saber cómo se generan y cómo se ejecutan los contenedores en Singularity.

8.3 Cómo usar los contenedores Singularity

Independientemente de la tecnología de contenedores utilizada, los contenedores se generan a partir de recetas de instalación, las cuales contienen toda la configuración necesaria para la correcta ejecución de una aplicación. Se recomienda que cada contenedor se componga de una única aplicación, dado que se conciben para contener entornos totalmente configurados lo más ligeros posible.

Este apartado está dedicado a familiarizarse con el ecosistema de los contenedores y con la tecnología Singularity. Un contenedor en Singularity se compone de un único fichero con formato Singularity Image File (SIF), el cual contiene el sistema de ficheros configurado con la aplicación. En el Apartado 8.3.1 se describe cómo crear estas recetas de instalación. La ejecución de un contenedor consiste en ejecutar una serie de comandos que pueden haber sido previamente definidos por el desarrollador del contenedor o pueden indicarse en esa ejecución en particular. Además del comando a ejecutar, el usuario puede personalizar una gran cantidad de parámetros de la ejecución del contenedor. Por ejemplo, aunque no es el caso de los contenedores en entornos HPC, es posible redirigir puertos del propio contenedor (por ejemplo, un servidor web) a un puerto en particular del sistema anfitrión o indicar que el contenedor se conecte a una red en particular. En el Apartado 8.3.2 se describen las distintas opciones de ejecución de un contenedor Singularity, así como sus diferencias.

8.3.1 Creación de contenedores - *Build*

Las recetas para la generación de contenedores en Singularity[22] se dividen en dos partes: la cabecera y el resto de secciones. La cabecera se debe definir al principio de la receta y contiene la información sobre la imagen

base de la que parte el contenedor. La cabecera está formada por, al menos, la variable Bootstrap, la cual indica la fuente de la imagen base. Existen varias posibilidades para el valor de esta variable, pero, en este libro, solo se utilizará el valor library, el cual indica que las imágenes base se encuentran en el repositorio de imágenes Singularity. En función del valor de Bootstrap, será necesario indicar otros parámetros[23]. Dado que se usuará Bootstrap: library, es necesario definir el parámetro From, el cual indica el contenedor usado como imagen base. Este puede tomar distintos valores en función de cómo se haya nombrado el contenedor (organización/contenedor:tag o organización/colección/contenedor:tag). Por ejemplo, las dos líneas siguientes indican que se utilizará como imagen base un contenedor con el sistema operativo Ubuntu en la versión 16.04 de la colección *ubuntu16* del usuario *serlophug*.

```
Bootstrap : library
From : serlophug / ubuntu16 / base : latest
```

Las siguientes secciones que pueden encontrarse en una receta de un contenedor Singularity tienen distintos objetivos:

- %labels: definir información que ayude a describir el contenedor, como, por ejemplo, quien generó la receta (nombre, conctacto, etc.) o datos sobre la misma (descripción de la aplicación, versión, etc.).

```
% labels
    Author Sergio López
    Email serlohu@upv.es
    Description Ejemplo de descripci ón del contenedor.
```

■%help: incluir una ayuda que será mostrada cuando el usuario ejecute el comando singularity run-help sobre una imagen de contenedor.

```
% help
    Esto es un texto de ayuda.
```

■%setup: definir los comandos que se ejecutarán en el sistema anfitrión antes de que la construcción del contenedor comience. Cabe destacar que es posible, en esta sección, realizar cambios en el sistema de ficheros del contenedor utilizando la variable de entorno SINGULARITY_ROOTFS , la cual contiene el directorio raíz del sistema de ficheros del contenedor. Esto permite generar carpetas o ficheros dentro del contenedor como en el ejemplo que sigue:

```
% setup
    touch $SINGULARITY_ROOTFS /tmp/ fichero
    touch $SINGULARITY_ROOTFS /mi–directorio
```

■%files: indicar qué ficheros o directorios del sistema anfitrión son copiados al sistema de ficheros del contenedor de la siguiente forma:

```
%files
    fichero_en_anfitrion /tmp/ fichero_en_contenedor
    dir1_en_anfitrion /tmp/ dir1_en_contenedor
```

■%environment: definir las variables de entorno que estarán disponibles cuando se ejecute el contenedor. Estas variables de entorno no estarán disponibles en el momento de la generación del contenedor.

```
%environment
```

```
    export MI_VARIABLE=hola
```

■%post: definir los comandos a ejecutar durante la generación del contenedor. Por ejemplo, estos son los comandos para instalar GCC:

```
%post
    apt–get update
    apt–get install –y gcc
```

■%runscript: definir los comandos que se ejecutarán cuando el contenedor se inicie con el comando singularity run sobre la imagen de contenedor. En el siguiente ejemplo, cuando se inicie el contenedor, se saludará al usuario:

```
%runscript
    echo " Hola mundo desde el contenedor ! "
```

■%test: definir una prueba de validación de la instalación que se ejecutará cuando se termine el proceso de generación del contenedor o cuando el usuario ejecute el comando singularity test sobre la imagen de contenedor.

```
%test
    INSTALL_DIR ="/opt/my–app "
    if [ –d " INSTALL_DIR " ]; then echo " Aplicaci ón instalada en
    $INSTALL_DIR "
    fi
```

Ninguna de las secciones descritas anteriormente debe aparecer

obligatoriamente en una receta de instalación. Normalmente, la cabecera y las secciones post y runscript suelen estar definidas, pero también es recomendable incluir las secciones labels, help y test.

Una vez se han comprendido las distintas partes de una receta de configuración de un contenedor, es posible crear un contenedor con el comando lolcow. En primer lugar, se ha de crear el fichero con la definición de la receta. El contenido de este fichero está descrito en *lolcow.def*. Se recomienda que este fichero se almacene en */SHARED/SingularityImages/lolcow/* con el nombre *lolcow.def* para que las imágenes sean accesibles desde todos los nodos.

Fichero: /SHARED/SingularityImages/lolcow/lolcow.def

```
Bootstrap : library
From : serlophug / ubuntu16 / base : latest

% labels
    author Sergio López–Huguet
    email serlohu@upv.es
    created 30/11/2020

% help
    Ejemplo de contendor que contiene el comando lolcow.

% post
    apt–get –y update
    apt–get –y install fortune cowsay lolcat

% environment
```

```
    export LC_ALL=C
    export PATH=/usr/ games : $PATH

% runscript
    echo " Hola mundo desde el contenedor ! " | cowsay

% test
    export LC_ALL=C
    export PATH=/usr/ games : $PATH
    fortune | cowsay | lolcat
```

Los comandos siguientes muestran cómo crear los directorios recomendados para almacenar los contenedores y sus recetas junto con la generación del contenedor. Además, se utiliza el parámetro "-F" para sobrescribir los labels y, si ya existiera una imagen anterior, generarla de nuevo. También se utiliza el parámetro "--disable-cache" para forzar que se realicen todos los pasos de la creación de la imagen y se descargue la imagen de contenedor base (aunque ya exista en el sistema). Cabe destacar que la creación de contenedores necesita permisos de administrador para realizarse de manera correcta. Por esta razón, la generación del contenedor no se realiza en los entornos HPC, sino en entornos controlados donde se disponga de permisos de administrador.

```
pi@nodo0:~ $ mkdir -p /SHARED/SingularityImages/lolcow
pi@nodo0:~ $ cd /SHARED/SingularityImages/lolcow
pi@nodo0:/SHARED/SingularityImages/lolcow
    $ sudo singularity build -F --disable-cache lolcow.sif lolcow.def
INFO:    Starting build...
INFO:    Running post scriptlet
+ apt-get -y update
Get:1 http://archive.ubuntu.com/ubuntu xenial InRelease [247 kB]
....
Reading package lists... Done
+ apt-get -y install fortune cowsay lolcat
Reading package lists... Done
....
Running hooks in /etc/ca-certificates/update.d...
done.
INFO:    Running test scriptlet
+ export LC_ALL=C
+ export PATH=/usr/games:/usr/local/sbin:/usr/local/bin:
    /usr/sbin:/usr/bin:/sbin:/bin
+ + cowsay+ lolcat + fortune
 ___________________________________
< Your aim is high and to the right. >
 -----------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
INFO:    Adding help info
INFO:    Adding labels
INFO:    Adding environment to container
INFO:    Adding runscript
INFO:    Adding testscript
INFO:    Creating SIF file...
INFO:    Build complete: lolcow.sif
```

Una vez se ha completado la creación del contenedor, este debería estar disponible en */SHARED/SingularityImages/lolcow/lolcow.sif*.

8.3.2 Ejecución de contenedores - *Run*

La ejecución de los contenedores puede configurarse de diversas formas en función de la aplicación a ejecutar y de la tecnología de contenedores

usada. Por defecto, Singularity se configura[24] para lograr que el usuario que está ejecutando el contenedor tenga prácticamente el mismo entorno que en la máquina anfitrión:

- Se incluye el usuario del sistema anfitrión que ejecuta el contenedor dentro del propio contenedor de manera que este tiene los mismos permisos que en el sistema anfitrión.
- Se monta el directorio personal del usuario (*/home/pi* en el clúster RPi) del sistema anfitrión en el contenedor.
- Se montan ciertos directorios del sistema anfitrión dentro del contenedor: */proc* , */dev* , */tmp* y */sys* .
- Se montan ficheros del sistema: */etc/localtime* y */etc/hosts* .
- Por defecto, las redes a las que tiene acceso el contenedor cuando se ejecuta son las mismas que el usuario tiene en el sistema anfitrión.

Singularity dispone de tres modos para la ejecución de contenedores:

- exec : permite la ejecución de un comando dentro del contenedor. Por ejemplo, es posible obtener información del sistema operativo de la imagen del contenedor *lolcow.sif* (generado en el apartado anterior) a partir del fichero */etc/os-release* . Los siguientes comandos permiten ver la diferencia entre el sistema operativo del contenedor y el del sistema anfitrión:

```
pi@nodoo:/SHARED/SingularityImages/lolcow $ singularity exec lolcow.sif cat /etc/os-release
NAME="Ubuntu"
VERSION="16.04.7 LTS (Xenial Xerus)"
ID=ubuntu
ID_LIKE=debian
```

```
PRETTY_NAME="Ubuntu 16.04.7 LTS"
VERSION_ID="16.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
VERSION_CODENAME=xenial
UBUNTU_CODENAME=xenial
pi@nodoo:/SHARED/SingularityImages/lolcow $ cat /etc/os-release
PRETTY_NAME="Raspbian GNU/Linux 10 (buster)"
NAME="Raspbian GNU/Linux"
VERSION_ID="10"
VERSION="10 (buster)"
VERSION_CODENAME=buster
ID=raspbian
ID_LIKE=debian
HOME_URL="http://www.raspbian.org/"
SUPPORT_URL="http://www.raspbian.org/RaspbianForums"
BUG_REPORT_URL="http://www.raspbian.org/RaspbianBugs"
```

■run : indica que deben ejecutarse dentro del contenedor los comandos definidos en la sección runscript de la receta de la imagen. El siguiente comando se utiliza para la ejecución mediante el comando run del contenedor *lolcow.sif* :

```
pi@nodo0:/SHARED/SingularityImages/lolcow
    $ singularity run lolcow.sif
 ---------------------------------
< Hola desde dentro del contenedor! >
 ---------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

■shell : inicia el contenedor y ofrece al usuario una terminal o *shell* dentro del contenedor. A continuación, se muestra un ejemplo de utilización de este comando:

```
COMENTARIO: Ejecución del contenedor mediante el comando shell
pi@nodoo:/SHARED/SingularityImages/lolcow $ singularity shell lolco-
    w.sif
Singularity> whoami
pi
Singularity> echo $HOME
/home/pi
Singularity> exit
exit
```

Los tres modos de ejecución ofrecen al usuario una gran flexibilidad de uso. Este libro se centra en proporcionar una guía práctica con las bases del uso de contenedores Singularity y, por esta razón, no se explicarán todos los posibles parámetros. No obstante, es interesante destacar algunos de los parámetros que comparten los tres comandos descritos anteriormente:

Para más material visita:
https://dogramcode.com/bloglibros/libros-electronica

■**-B, --bind** <src[:dest[:opts]]>: permite el montaje de directorios o ficheros desde el sistema anfitrión (src) al sistema de ficheros del contenedor (dest). Las opciones (opts) disponibles son ro (solo lectura) o rw (modo por defecto que permite la lectura y escritura). Es posible especificar varios puntos de montaje mediante una lista separada por comas. La siguiente consola muestra cómo ejecutar un contenedor que monta los directorios del sistema anfitrión */tmp/dir1* y */tmp/dir2* en el sistema de ficheros del contenedor en */dir1* y */var/dir2* , respectivamente, lo que permite solo la lectura del directorio */dir2* . Nótese que estos directorios no existen, así que antes de la ejecución del contenedor, se deben crear.

```
pi@nodo0:~ $ mkdir -p /tmp/dir1 /tmp/dir2
pi@nodo0:~ $ singularity exec -B /tmp/dir1:/dir1,/tmp/dir2:/var/dir2:ro
    /SHARED/SingularityImages/lolcow/lolcow.sif touch /dir1/hola.txt
pi@nodo0:~ $ ls /tmp/dir1
hola.txt
pi@nodo0:~ $ singularity exec -B /tmp/dir1:/dir1,/tmp/dir2:/var/dir2:ro
    /SHARED/SingularityImages/lolcow/lolcow.sif touch /var/dir2/
    hola.-txt
touch: cannot touch '/var/dir2/hola.txt': Read-only file system
pi@nodo0:~ $ ls /tmp/dir2
```

■**--writable-tmpfs** : el sistema de ficheros de los contenedores en ejecución es, por defecto, de solo lectura. Esto hace que, cada vez que se inicie el contenedor, no pueda realizarse ningún cam-bio. Mediante esta opción se permite la escritura en el sistema de ficheros del contenedor. Sin embargo, cuando se termina la ejecución de ese contenedor, los cambios no quedan almacenados en la imagen del contenedor. En la siguiente

consola se puede ver cómo, una vez se ha salido del entorno del contenedor, el comando no ha quedado almacenado.

```
pi@nodo0:~ $ singularity exec
     /SHARED/SingularityImages/lolcow/lolcow.sif  banner "Hola"
/.singularity.d/actions/exec: 21: exec: banner: not found
pi@nodo0:~ $ sudo singularity shell --writable-tmpfs
     /SHARED/SingularityImages/lolcow/lolcow.sif
Singularity> apt install -y sysvbanner
...
Singularity> banner "Hola"
#     #
#     #   ####   #         ##
#     #  #    #  #        #  #
#######  #    #  #       #    #
#     #  #    #  #       ######
#     #  #    #  #       #    #
#     #   ####   ######  #    #
Singularity> exit
pi@nodo0:~ $ singularity exec
     /SHARED/SingularityImages/lolcow/lolcow.sif banner "Hola"
/.singularity.d/actions/exec: 21: exec: banner: not found
```

■**--nv** : permite el montaje de GPUs en el contenedor. La siguiente consola muestra el uso de este parámetro. Nótese que este comando no puede funcionar en el clúster RPi debido a que las Raspberry Pi no disponen de una tarjeta gráfica adicional que puedan montar.

```
pi@nodoo:~ $ singularity run --nv
/SHARED/SingularityImages/lolcow/lolcow.sif
```

8.3.3 Otros comandos interesantes

Los comandos descritos en los Apartados 8.3.1 y 8.3.2 permiten la creación de imágenes de contenedores y la ejecución de contenedores. En este apartado se va a describir cómo obtener información de una imagen de contenedor previamente creada y cómo interactuar con los repositorios de imágenes, concretamente con el repositorio oficial de Sylabs[25].

El comando inspect[26] proporciona al usuario información sobre el contenedor. En la siguiente consola, se utilizan distintos argumentos de este comando para obtener la información definida en la sección labels (versión por defecto o con el argumento "-l, --labels") o la sección runscript (con el argumento "-r, --runscript") de la receta de la imagen *lolcow.sif*. Además, también es posible obtener, mediante el argumento "-d, --deffile", la receta con la que se generó la imagen del contenedor.

```
pi@nodo0:~ $ singularity inspect /SHARED/SingularityImages/
    lol-cow/lolco-w.sif
author: Sergio López-Huguet
created: 30/11/2020
email: serlohu@upv.es
org.label-schema.build-date: Friday_30_October_2020_14:44:5_UTC
org.label-schema.schema-version: 1.0
org.label-schema.usage: /.singularity.d/runscript.help
org.label-schema.usage.singularity.deffile.bootstrap: library
org.label-schema.usage.singularity.deffile.from: Bootstrap: serlophug/
    ubuntu16/base:latest
org.label-schema.usage.singularity.runscript.help: /.singularity.d/
    runscript.help
org.label-schema.usage.singularity.version: 3.5.0
```

```
pi@nodoo:~ $ singularity inspect --runscript /SHARED/
    SingularityImages/lolcow/lolcow.sif
#!/bin/sh

    echo "Hola desde dentro del contenedor!"| cowsay
pi@nodoo:~ $ singularity inspect --deffile /SHARED/
    SingularityImages/lolcow/lolcow.sif
Bootstrap: library
From: serlophug/ubuntu16/base:latest

%labels
    author Sergio López-Huguet
    email serlohu@upv.es
    created 30/11/2020

%help
    This is an example of a simple container that contains the lolcow
    command.
    Contenedor simple de ejemplo que contiene el comando lolcow.
%post
    apt-get -y update

apt-get -y install fortune cowsay lolcat

%environment
    export LC_ALL=C
    export PATH=/usr/games:$PATH

%runscript
    echo "Hola desde dentro del contenedor" | cowsay
```

```
%test
    export LC_ALL=C
    export PATH=/usr/games:$PATH
    fortune | cowsay | lolcat
```

Los comandos pull[27] y push[28] son el método para, respectivamente, descargar o subir imágenes de contenedores de los repositorios de imágenes. El comando pull permite al usuario descargar imágenes desde los distintos repositorios de Singularity (*library://* y *shub://*), de repositorios de imágenes OCI (*oras://*) y de repositorios de imágenes Docker (*docker://* o repositorios privados). El comando pull permite indicar el directorio donde se almacenará la imagen descargada mediante el parámetro "--dir". Por otro lado, el parámetro "--arch" permite especificar la arquitectura de la imagen a descargar. Por defecto, la arquitectura es *amd64*, aunque, para el caso de la RPi, la arquitectura que debe indicarse es *arm*.

La siguiente consola contiene los comandos necesarios para descargar una imagen de Ubuntu 16.04 junto con una imagen creada por los autores del libro con el contenedor *lolcow*.

```
COMENTARIO: Crear carpeta para almacenar las imagenes Singularity
    de Ubuntu
pi@nodoo:~ $ mkdir /SHARED/SingularityImages/ubuntu
COMENTARIO: Descargar Ubuntu 16.04 especificando el nombre del
    archivo: ubuntu_16_04.sif
pi@nodoo:~ $ singularity pull --arch arm --dir /SHARED/
    SingularityImages/ubuntu ubuntu_16_04.sif library://serlophug/
    ubun-tu16/base:-test
INFO: Downloading library image
```

```
31.12 MiB / 31.12 MiB [==========] 100.00% 4.06 MiB/s 7s
WARNING: unable to verify container: /SHARED/SingularityImages/ubun-ubuntu_16_04.sifsif
WARNING: Skipping container verification
pi@nodoo:~ $ ls /SHARED/SingularityImages/ubun-ubuntu_16_04.sifsif
COMENTARIO: Descargar lolcow sin especificar el nombre del archivo descargado
pi@nodoo:~ $ singularity pull --arch arm --dir /SHARED/SingularityImages/lolcow/ library://serlophug/ubun-tu16/cow:latesttest
INFO: Downloading library image
73.63 MiB / 73.63 MiB [==========] 100.00% 8.11 MiB/s 9s
WARNING: unable to verify container: /SHARED/SingularityImages/lol-cow/lolco-test.sift.sif
WARNING: Skipping container verification
pi@nodoo:~ $ ls /SHARED/SingularityImages/lolcow/
lolcow.def lolcow.sif lolcow_latest.sif
```

La siguiente consola muestra como el comando push se utiliza para subir imágenes que se encuentran en el ordenador a un repositorio de imágenes. Antes de poder subir imágenes es necesario autenticarse en el repositorio de imágenes. Esto se hace en el primer comando, donde se especifica una cadena de texto única (llamada *token*) obtenida desde el repositorio de Sylabs para identificar al usuario que realiza la subida de la imagen (en este caso,

serlophug).

```
COMENTARIO: Identificarse en el repositorio para poder realizar la subida
pi@nodoo:~ $ singularity remote login
INFO: Authenticating with default remote.
Generate an API Key at https://cloud.sylabs.io/auth/tokens, and paste here:
API Key:
INFO: API Key Verified!
pi@nodoo:~ $ singularity push -U /SHARED/SingularityImages/lol-cow/lolco-w.sif library://serlophug/ubuntu16/lolcow:latest
WARNING: Skipping container verifying
73.63 MiB / 73.63 MiB [==========================================] 100.00% 586.12 KiB/s 2m8s
```

8.4 Caso de uso

La adopción de Singularity en los supercomputadores ha sido propiciada por el hecho de que permite la ejecución de los contenedores sin necesitar un demonio con permisos de administrador y de que está diseñado para soportar, de manera nativa, la ejecución de aplicaciones MPI[29] (las dos princpiales implementaciones OpenMPI y MPICH). Singularity se encarga de gestionar la interacción entre el MPI en el sistema anfitrión y el MPI que se encuentra dentro del contenedor. Para que sea posible esta interacción es necesario que ambas versiones sean compatibles. Por ejemplo, en este apartado se va generar una imagen con OpenMPI pero con distinta versión (versión 4.0.3) a la instalada en el Apartado 5.2.2 (versión 4.0.2), se generará otra imagen que contendrá el fichero *hola_mundo_mpi.c* generado en el Apartado 5.2.2 y, finalmente, se ejecutará en todo el clúster RPi utilizando Slurm.

8.4.1 Creación del contenedor con MPI y *hola_mundo_mpi.c*

En primer lugar, se debe generar el contenedor base con OpenMPI. Este contenedor se utilizará para crear otros contenedores que necesiten MPI. De esta forma, se consigue que solamente se deba compilar MPI una única vez. Una vez el contenedor de OpenMPI esté generado, se procederá a generar la imagen con el fichero *hola_mundo_mpi.c*.

Fichero: /SHARED/SingularityImages/ompi/ompi.def

```
Bootstrap : library
From : serlophug / ubuntu16 / base : latest

% labels
    author Sergio López–Huguet
```

```
    email serlohu@upv.es
    created 30/11/2020

% environment
    export OMPI_DIR=/opt/ ompi
    export SINGULARITY_OMPI_DIR=$OMPI_DIR
    export SINGULARITYENV_APPEND_PATH=$OMPI_DIR/bin
    export SINGULAIRTYENV_APPEND_L-
    D_LIBRARY_PATH=$OMPI_DIR/lib

% post
    echo " Installing required packages . . . "
    apt-get update
    apt-get install -y wget git tar bzip2 bash gcc gfortran g++ make file

    echo " Installing Open MPI "
    export OMPI_DIR=/opt/ ompi
    export OMPI_VERSION =4.0.3
    export
    OMPI_URL="https://download.open-mpi.org/release/open-mpi/
    v4.0 / openmpi-$OMPI_VERSION . tar . bz2 "
    mkdir -p /tmp/ ompi
    mkdir -p /opt

    # Download
    cd /tmp/ ompi
    wget -O openmpi-$OMPI_VERSION . tar . bz2 $OMPI_URL tar
    -xjf openmpi-$OMPI_VERSION . tar . bz2
```

```
    # Compile and install
    cd /tmp/ ompi / openmpi–$OMPI_VERSION
    . / configure --prefix=$OMPI_DIR
    make install

    # Set env variables so we can compile our application
    export PATH=$OMPI_DIR /bin : $PATH
    export LD_LIBRARY_PATH=$OMPI_DIR /lib : $LD_LIBRA-
    RY_PATH
    export MANPATH=$OMPI_DIR / share /man : $MANPATH
```

Fichero: /SHARED/SingularityImages/ompi/hola-mundo/hola-mundo-.def

```
Bootstrap : localimage
From : / SHARED / SingularityImages / ompi / ompi . sif

% files
    hola_mundo_mpi . c /opt

% labels
    author Sergio López–Huguet
    email serlohu@upv.es
    created 30/11/2020

% post
    # Set env variables so we can compile our application
    export OMPI_DIR=/opt/ ompi
```

```
        export PATH=$OMPI_DIR /bin : $PATH
        export LD_LIBRARY_PATH=$OMPI_DIR /lib : $LD_LIBRA-
        RY_PATH
        export MANPATH=$OMPI_DIR / share /man : $MANPATH

        # Compiling the application
        cd /opt
        mpicc –o hola_mundo_mpi hola_mundo_mpi . c

    % runscript
        echo " Se va a ejecutar el programa /opt/ hola_mundo_mpi "
        /opt/ hola_mundo_mpi
```

La siguiente consola contiene todos los comandos necesarios para la generación de ambas imágenes:

```
COMENTARIO: Generar las carpetas donde se almacenarán los conte-
    nedores
pi@nodoo:~ $ mkdir -p /SHARED/SingularityImages/
    ompi/hola-mundo
pi@nodoo:~ $ cd /SHARED/SingularityImages/ompi
COMENTARIO: Creación del contenedor base con OpenMPI 4.0.3
pi@nodoo:/SHARED/SingularityImages/ompi $ sudo singularity build
    -F --disable-cache ompi.sif ompi.def
INFO: Starting build...
...
INFO: Build complete: ompi.sif
pi@nodoo:/SHARED/SingularityImages/ompi $ cd hola-mundo
```

```
COMENTARIO: Copiar hola_mundo_mpi.c
pi@nodoo:/SHARED/SingularityImages/ompi/hola-mundo $ cp
    ~/hola_mundo_mpi.c .
COMENTARIO: Creación del contenedor con hola_mundo_mpi.c
pi@nodoo:/SHARED/SingularityImages/ompi/hola-mundo $ sudo
    singularity build -F --disable-cache hola-mundo.sif hola-mundo.def
INFO: Starting build...
...
INFO: Build complete: hola-mundo.sif
```

8.4.2 Ejecución del contenedor en el clúster

Con la imagen del contenedor *hola-mundo.sif* generado en el apartado anterior, es posible probar que dicho contenedor funciona correctamente. Dado que MPICH es la implementación utilizada por defecto, es necesario utilizar la ruta absoluta del binario *mpirun* de nuestra instalación de OpenMPI para que este se comunique correctamente con el OpenMPI instalado en el contenedor. La siguiente consola contiene los comandos para realizar dos ejecuciones del contenedor. El primer comando es la ejecución del contenedor en un nodo (en este caso se está ejecutando en el nodo *front-end*) con el número de procesadores por defecto (uno). El segundo comando se corresponde con la ejecución del contenedor en todos los nodos de cómputo.

```
COMENTARIO: Ejecución del contenedor hola-mundo.sif
pi@nodoo:/SHARED/SingularityImages/openmpi/hola-mundo $
    singularity run hola-mundo.sif
Se va a ejecutar el programa /opt/hola_mundo_mpi
```

```
Hola mundo desde nodo0, con rango 0 de un total de 1 procesadores

pi@nodo0:/SHARED/SingularityImages/openmpi/hola-mundo $
    /SHARED/openmpi-4.0.2-install/bin/mpirun -n 3 --host nodo1,no-
    do2,nodo2 /SHARED/singularity/bin/singularity run hola-
    mundo.sif
Se va a ejecutar el programa /opt/hola_mundo_mpi
Hola mundo desde nodo1, con rango 0 de un total de 3 procesadores
Se va a ejecutar el programa /opt/hola_mundo_mpi
Hola mundo desde nodo2, con rango 1 de un total de 3 procesadores
Se va a ejecutar el programa /opt/hola_mundo_mpi
Hola mundo desde nodo3, con rango 2 de un total de 3 procesadores
```

Como se ha comentado a lo largo del libro, la ejecución de los trabajos se realiza mediante un gestor de trabajos (en el Apartado 6.1.3 se ha instalado Slurm). Para ello, es necesario generar el siguiente fichero:

Fichero: /SHARED/SingularityImages/ompi/hola-mundo/hola-mundo-.sba-

```
#!/bin/ bash
#SBATCH --nodes 3
#SBATCH --job–name hola–mundo–singularity

MPIRUN=/SHARED / openmpi /bin/ mpirun
SINGULARITY=/SHARED / singularity /bin/ singularity
IMAGE=/SHARED / SingularityImages / ompi /hola–mundo /hola-
mundo . sif
$MPIRUN –n 3 --host nodo1 , nodo2 , nodo3 $SINGULARITY run
```

```
$IMAGE
```

Ahora ya es posible crear el trabajo en Slurm. Los siguientes comandos se corresponden con el envío del trabajo, la comprobación de que el trabajo esta ejecutándose y la salida de ejecución del trabajo:

```
pi@nodo0:/SHARED/SingularityImages/openmpi/hola-mundo
    $ sbatch hola-mundo.sbatch
Submitted batch job 23
pi@nodo0:/SHARED/SingularityImages/openmpi/hola-mundo $ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES
    NODELIST(REASON)
                23      main hola-mun       pi  R       0:05      3
    nodo[1-3]
pi@nodo0:/SHARED/SingularityImages/openmpi/hola-mundo
    $ cat slurm-23.out
Se va a ejecutar el programa /opt/hola_mundo_mpi
Se va a ejecutar el programa /opt/hola_mundo_mpi
Hola mundo desde nodo1, con rango 0 de un total de 3 procesadores
Hola mundo desde nodo2, con rango 1 de un total de 3 procesadores
Se va a ejecutar el programa /opt/hola_mundo_mpi
Hola mundo desde nodo3, con rango 2 de un total de 3 procesadores
```

8.5 ¿Por qué usar contenedores?

La popularización de los contenedores se debe a que facilitan la portabilidad de entornos totalmente configurados. La ejecución de estos entornos se hace de manera aislada respecto al resto de procesos de la máquina anfitrión o de otros contenedores con un coste menor al de ejecutar estos entornos en máquinas virtuales.

Si bien es cierto que al coste temporal de la creación de las aplicaciones hay que añadir el coste de la creación de la receta para construir la imagen del contenedor, este proceso se realiza, por regla general, solamente una vez. Además, dado que los desarrolladores son quienes más conocen cómo deben configurarse sus aplicaciones, la utilización de recetas proporcionadas por los desarrolladores puede ayudar a que las aplicaciones estén configuradas de forma óptima. Por otro lado, dado que la configuración está descrita en un fichero estructurado, esto facilita futuras modificaciones de la configuración, así como la posibilidad de disponer de recetas para la instalación de distintas versiones de las aplicaciones o sistemas operativos.

Por otra parte, cuando los usuarios no disponen de permisos de administrador pero sí pueden ejecutar contenedores, estos permiten la ejecución de aplicaciones o bibliotecas cuyas versiones sean incompatibles con las instaladas en el sistema o, incluso, que no estén disponibles en el sistema. Esto puede facilitar enormemente el trabajo de los desarrolladores de aplicaciones en entornos HPC cuando hay problemas de este tipo.

Respecto al rendimiento de los contenedores frente a ejecutar las aplicaciones directamente sobre el sistema anfitrión, existe un pequeño sobrecoste debido a que son necesarias distintas capas para obtener el aislamiento. En el caso de aplicaciones muy costosas temporalmente, este coste se amortiza debido a la pequeña proporción que representa la creación del contenedor respecto al tiempo total de ejecución.

8.6 Para profundizar...

1. Durante el capítulo se ha trabajado con la tecnología de contenedores Singularity. Como se ha comentado en el Apartado 8.1, existen diversas tecnologías actualmente. Respecto a estas tecnologías, ¿cuáles serían las más adecuadas para ser usadas en un entorno HPC?

   Para responder, es necesario considerar cómo afecta quien está ejecutando el contenedor, si hay algún programa malicioso en la imagen y cómo podría este provocar daños en el sistema, etc.

2. Normalmente, los contenedores son controlados por un orquestador (*Container Orchestrator* , en inglés). ¿Cuáles son los orquestadores más utilizados actualmente? ¿Cuáles de ellos soportan Singularity?

   Debido a que la tecnología Docker propició el auge de los contenedores, la mayoría de los orquestadores fueron diseñados para soportar Docker. Algunos de estos orquestadores también soportan otras tecnologías de contenedores de serie, y otros pueden personalizarse lo suficiente como para admitir otras tecnologías pero, por ejemplo, el orquestador Docker Swarm solo soporta Docker.

3. En el Apartado 8.4.1 se ha generado un contenedor con OpenMPI. ¿Qué ocurriría si, en lugar de ejecutarse mediante mpirun de OpenMPI, se ejecutara utilizando mpirun de MPICH?

   Mediante la comprobación de esta sugerencia, se podrán experimentar de primera mano las incompatibilidades existentes entre ambos.

4. La creación de las imágenes de contenedores necesita de permisos de administrador, ya que parte de las subtareas que componen este proceso no pueden completarse sin dichos permisos (por ejemplo, la instalación de paquetes). ¿Singularity permite de alguna otra forma la generación de imágenes sin permisos de administrador o la creación de imágenes de manera remota? ¿Qué sería necesario?

Puede encontrarse más información en: https://sylabs.io/guides/3.5/user-guide/fake-root.ht-https://sylabs.io/guides/3.5/user-guide/cloud_library.html#remote-builder.

8.7 Material recomendado

Los materiales propuestos en este capítulo tienen como objetivo ayudar a profundizar en el campo de las tecnologías de los contenedores, así como ampliar los conocimientos relativos a Singularity. La referencia web [1] (en inglés) contiene información sobre qué son los contenedores en Linux, concretamente sobre Linux Containers (LxC) y Docker. Por otro lado, en el enlace web [2] se puede encontrar un tutorial (en inglés) sobre la utilización de contenedores Docker. Este tutorial puede hacerse sin que sea necesario llevar a cabo ninguna instalación, ya que Docker proporciona un entorno de prácticas donde realizar el tutorial sin ningún tipo de coste. Finalmente, la web [3] contiene enlaces a distintos tutoriales de Singularity (en inglés) de varios niveles de dificultad.

[1] Inc. Red Hat. El concepto de los contenedores de Linux. https://www.redhat.com/es/topics/containers . Accedido por última vez: nov. 2020.

[2] Sylabs™ . Play with Docker.Hands-on Docker Tutorials for Developers. https://www.docker.com/play-with-docker . Accedido por última vez: nov 2020.

[3] Pawsey Supercomputing centre. Container Workshops. https://pawseysc.github.io/containers.html . Accedido por última vez: nov 2020.

8.8 Resumen

En este capítulo se han presentado diferentes tecnologías de virtualización y se ha profundizado en el estudio de una de ellas: los contenedores. La mayor parte del capítulo se ha centrado en entender qué son los contenedores, cómo pueden usarse y cuáles son sus ventajas e inconvenientes. Por otro lado, se ha realizado una introducción a la tecnología de contenedores Singularity en la que, partiendo de los conceptos más simples, se han llegado a generar imágenes complejas que afianzan los conocimientos adquiridos durante el transcurso de toda la lectura del libro.

En resumen, los principales aspectos tratados en este capítulo han sido:

- Definición de los conceptos de tecnología de virtualización, máquinas virtuales y contenedores. Se han presentado también las diferencias entre ambos modos de virtualización.
- Identificación de las ventajas y desventajas del uso de contenedores.
- Reflexión sobre los inconvenientes y fortalezas de las distintas tecnologías de contenedores, que las hacen más apropiadas para unos entornos u otros.
- Adquisición de un conocimiento general sobre el ecosistema de contenedores: recetas (o ficheros de definición de la configuración), imágenes, contenedores y repositorios de imágenes. Además, se ha profundizado en Singularity, sus particularidades y cómo puede ser utilizado en el ámbito de la HPC.
- Creación de recetas de configuración de imágenes de contenedores.
- Ejecución de contenedores complejos a través del método usual en el ámbito de la supercomputación, es decir, a través de un gestor de colas.

8.8.1 ¿Qué viene a continuación?

Este capítulo cierra la parte III del libro, centrada en el montaje, configuración y uso del clúster RPi. A continuación, se presenta una serie de actividades (con su correspondiente solución) para practicar con parte del software presentado a lo largo de todo el libro.

1 https://sylabs.io/2020/08/worlds-fastest-super-com-puter-ssingula-rity-pro

2 https://www.docker.com/

3 https://www.freebsd.org/doc/es_ES.ISO8859-books/handbook/jails.htmlml

4 https://sylabs.io/singularity/

5 https://www.redhat.com/es/topics/virtualization/what-is-ahypervisor

6 https://docs.microsoft.com/es-es/virtualization/hyper-v-on-windows/

7 https://www.virtualbox.org/

8 https://www.vmware.com/es/products/workstation-player.html

9 https://hub.docker.com/

10 https://quay.io/

11 https://cloud.sylabs.io/library

12 https://singularity-hub.org/

13 https://hub.docker.com/r/openfoam/openfoam8-paraview56

14 https://cloud.sylabs.io/library/serlophug/ubuntu16/hola-mundo

15 https://ubuntu.com/

16 https://opencontainers.org/

17 https://linuxcontainers.org/

18 https://podman.io/

19 https://hpc.github.io/charliecloud/

20 https://www.nersc.gov/research-and-de-velopment/user-defi-ima-ges/

21 https://golang.org/

22 https://sylabs.io/guides/3.5/user-guide/definition_files.html

23 Apéndice sobre los distintos parámetros en función de la variableBootstrap: https://sylabs.io/guides/3.5/user-guide/appen-dix.htmbuildmodu-les

24 Esta configuración se puede modificar editando el fichero */SHARED/singularity/etc/singularity/singularity.conf*

25 https://cloud.sylabs.io/library

26 https://sylabs.io/guides/3.5/user-guide/environment_and_metadata.html#the-inspect-c

27 https://sylabs.io/guides/3.5/user-guide/cli/singularity_pull.html

28 https://sylabs.io/guides/3.5/user-guide/cli/singularity_push.html

29 https://sylabs.io/guides/3.5/user-guide/mpi.html

Parte IV

Actividades propuestas

9. Actividades propuestas

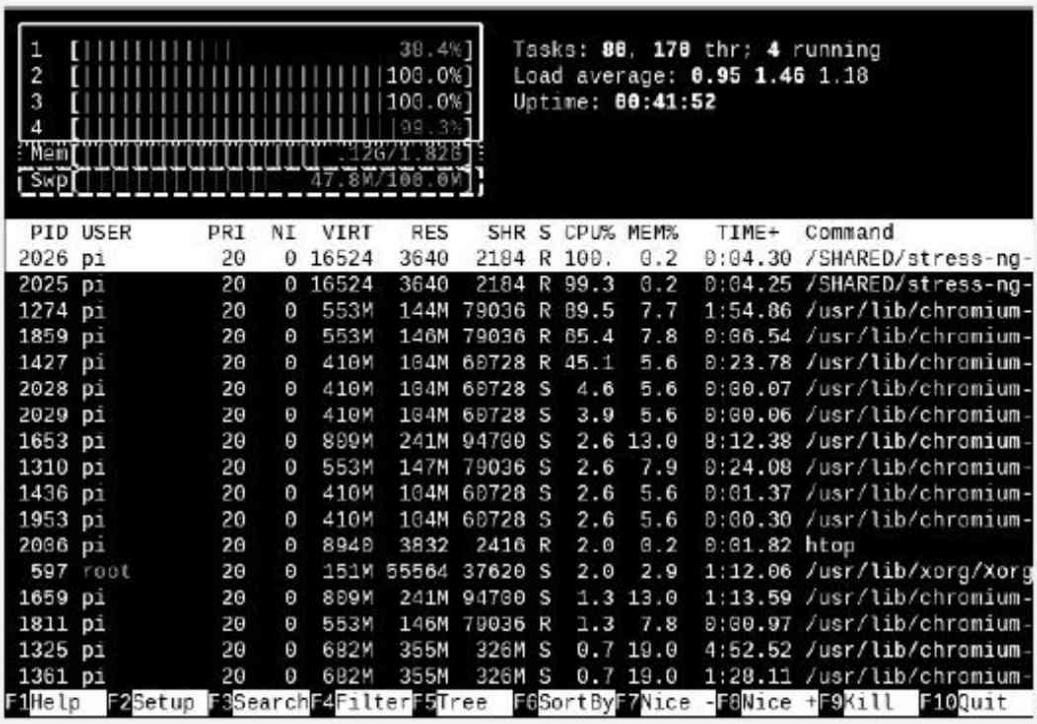
Ejercicio 1

Utilizando htop, responda a las siguientes preguntas:

1. ¿Cuántas CPU hay en el sistema? ¿Qué carga tiene cada una de ellas?
2. ¿Cuánta memoria RAM tiene el sistema? ¿Cuánta queda libre?
3. ¿Se está haciendo uso del espacio *swap* ?

Solución 1

Se presenta la solución a las preguntas propuestas en una ejecución concreta sobre uno de los nodos del clúster RPi. La imagen siguiente muestra la salida de ejecución de htop.



De acuerdo a la información de la figura:

1. Se observa que hay cuatro CPU, cuya carga es 38,4%, 100,0%, 100,0% y 99,3% en el momento de ejecutar la aplicación (nótese que la carga varía con el tiempo según el trabajo que se esté llevando a cabo en el dispositivo).
2. La memoria RAM del sistema es de 1,82 GB y quedan disponibles 1,82 −0,12 = 1,7 GB.
3. Se muestra que, efectivamente, se está haciendo uso del espacio *swap* ; en concreto se están utilizando 47,8 MB de 100 MB.

Ejercicio 2

Ejecute stream con uno, dos y cuatro hilos. Observe:

1. Para un mismo número de hilos, ¿el ancho de banda obtenido es similar para todas las operaciones?

2. Al variar el número de hilos, ¿para qué cantidad de hilos se observa mayor ancho de banda?

Solución 2

Se presenta la solución a las preguntas propuestas en una ejecución concreta sobre uno de los nodos del clúster RPi.

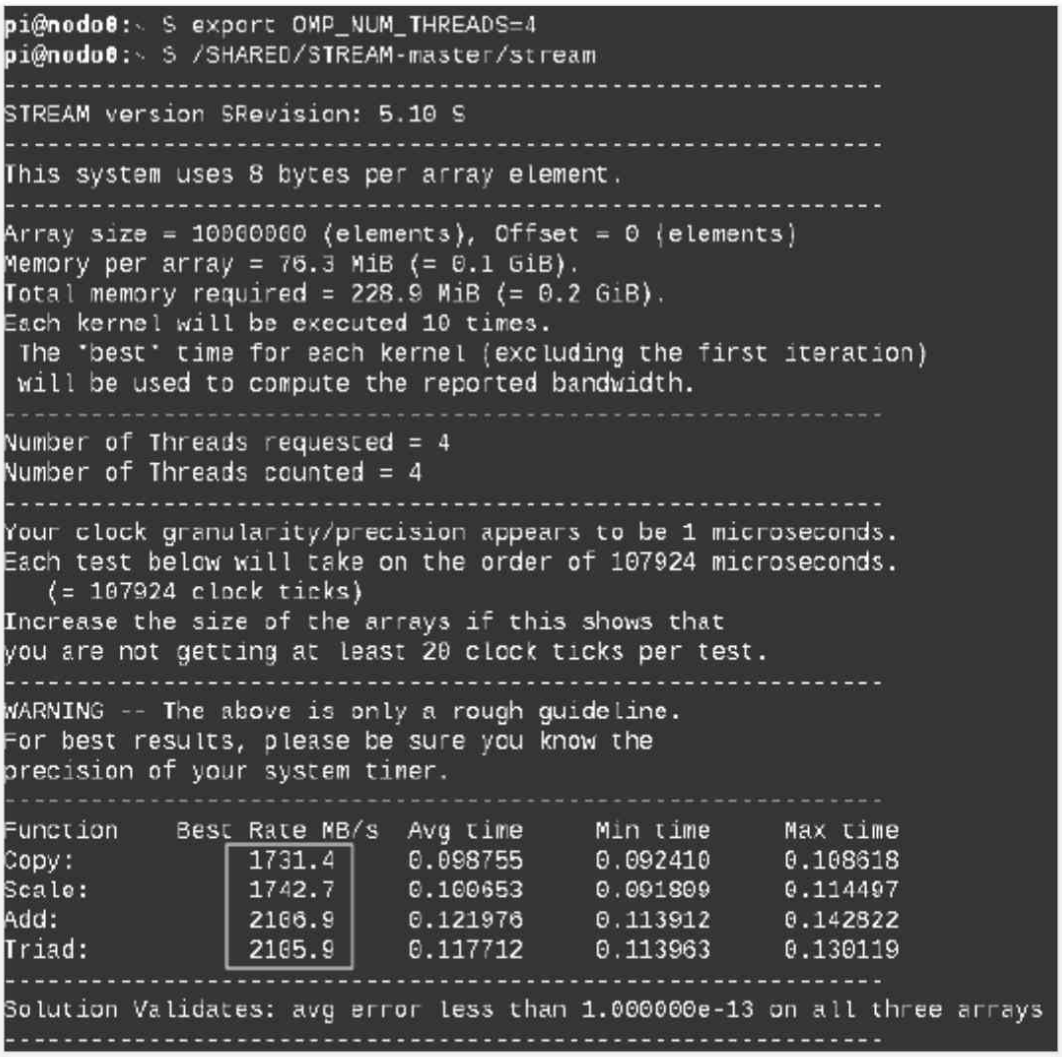
- **Un hilo:**

```
pi@nodo8:~ $ export OMP_NUM_THREADS=1
pi@nodo8:~ $ /SHARED/STREAM-master/stream
-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 8 bytes per array element.
-------------------------------------------------------------
Array size = 10000000 (elements), Offset = 0 (elements)
Memory per array = 76.3 MiB (= 0.1 GiB).
Total memory required = 228.9 MiB (= 0.2 GiB).
Each kernel will be executed 10 times.
 The "best" time for each kernel (excluding the first iteration)
 will be used to compute the reported bandwidth.
-------------------------------------------------------------
Number of Threads requested = 1
Number of Threads counted = 1
-------------------------------------------------------------
Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 149857 microseconds.
   (= 149857 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:           1000.3     0.166392     0.159958     0.170203
Scale:           955.6     0.170452     0.167436     0.174591
Add:            1153.4     0.214215     0.208075     0.220465
Triad:          1171.4     0.211700     0.204877     0.215598
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
-------------------------------------------------------------
```

- **Dos hilos:**

```
i@nodo0:~ $ export OMP_NUM_THREADS=2
i@nodo0:~ $ /SHARED/STREAM-master/stream
-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 8 bytes per array element.
-------------------------------------------------------------
Array size = 10000000 (elements), Offset = 0 (elements)
Memory per array = 76.3 MiB (= 0.1 GiB).
Total memory required = 228.9 MiB (= 0.2 GiB).
Each kernel will be executed 10 times.
 The "best" time for each kernel (excluding the first iteration)
 will be used to compute the reported bandwidth.
-------------------------------------------------------------
Number of Threads requested = 2
Number of Threads counted = 2
-------------------------------------------------------------
Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 82376 microseconds.
   (= 82376 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:           1695.6     0.094646     0.094362     0.095139
Scale:          1730.4     0.094007     0.092463     0.100315
Add:            2123.2     0.114139     0.113038     0.117313
Triad:          2096.0     0.115063     0.114504     0.116237
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
-------------------------------------------------------------
```

■Cuatro hilos:



De acuerdo a la información de la figuras, se observa una cierta variación en el tiempo para las distintas operaciones debido a que cada una hace un número distinto de accesos a memoria. Además, también se puede observar que el mayor ancho de banda se obtiene al utilizar cuatro hilos.

Ejercicio 3

Analice el impacto de la frecuencia de los núcleos en la temperatura del sistema. Utilizando cpufrequtils y vcgencmd para medir la temperatura y stress-ng:

1. Establezca la frecuencia de todos los núcleos a 600 MHz.

2. Compruebe la temperatura actual del chip.

3. Complete los espacios señalados con llaves {} con los parámetros necesarios que garanticen que se ejecute stress-ng con cuatro hilos durante 30 segundos y que a continuación se mida la temperatura de nuevo.

```
stress-ng --cpu {__} -t {__} ;
/opt/vc/bin/vcgencmd {____}
```

Repita los mismos pasos, pero ahora establezca la frecuencia a 1,40 GHz. Conteste a las siguientes preguntas:

- ¿La temperatura inicial es mayor, menor o igual que con 600 MHz?
- ¿La temperatura final es mayor, menor o igual que con 600 MHz?

Solución 3

1. Establezca la frecuencia de todos los cores a 600 MHz:

```
pi@nodoo:~ $ sudo cpufreq-set -g userspace
pi@nodoo:~ $ sudo cpufreq-set -f 600000
```

2. Compruebe la temperatura del sistema:

```
pi@nodoo:~ $ /opt/vc/bin/vcgencmd measure_temp temp=45.7'C
```

3. Genere carga al sistema mediante el comando stress-ng con 4 hilos

durante 30 segundos y, acto seguido, mida la temperatura del sistema:

```
pi@nodoo:~ $ /SHARED/stress-ng-master/stress-ng --cpu 4 -t 30
stress-ng: info:[25974] dispatching hogs: 4 cpu
stress-ng: info:[25974] successful run completed in 30.08s
pi@nodoo:~ $ /opt/vc/bin/vcgencmd measure_temp temp=47.5'C
```

Repita los mismos pasos, pero ahora establezca la frecuencia a 1,50 GHz:

```
pi@nodoo:~ $ sudo cpufreq-set -f 1500000
pi@nodoo:~ $ /opt/vc/bin/vcgencmd measure_temp temp=46.2'C
pi@nodoo:~ $ /SHARED/stress-ng-master/stress-ng --cpu 4 -t 30
stress-ng: info:[26402] dispatching hogs: 4 cpu
stress-ng: info:[26402] successful run completed in 30.97s
pi@nodoo:~ $ /opt/vc/bin/vcgencmd measure_temp temp=54.5'C
```

Por tanto, se puede concluir que la ejecución de stress-ng eleva la temperatura del chip en ambas frecuencias. Esta variación es mayor cuando se utilizan frecuencias más altas.

Ejercicio 4

Lleve a cabo un análisis de escalabilidad del clúster Raspberry Pi utilizando HPL. Asuma como factor para el particionado por bloques NB=128. Tenga en cuenta que necisitará ejecutar tantos tests como nodos de cómputo tiene el clúster.

1. Determine el número total de procesos necesarios en cada test y la

distribución de los mismos (parámetros P y Q).

2. Determine el tamaño de la matriz a utilizar teniendo en cuenta la memoria RAM disponible. Recuerde que un valor adecuado suele estar comprendido entre el 80% y 90% de la memoria total.

3. Modifique el fichero *HPL.dat* con los valores calculados y ejecute los tests necesarios.

4. ¿Escala correctamente el sistema?

Solución 4

Esta propuesta de solución asume que el estudio de escalabilidad se realiza en un clúster Raspberry Pi con tres nodos de cómputo. Cada nodo dispone de cuatro núcleos y una memoria RAM de 2 GB.

Para realizar el estudio de escalabilidad será necesario realizar tres tests: el primero (T1) se ejecutará en un único nodo, el segundo (T2), en dos y el tercero (T3), en los tres nodos de cómputo.

1. Determine el número total de procesos necesarios en cada test y la distribución de los mismos (parámetros P y Q). Puesto que se necesita ejecutar tres tests y que la distribución de los procesos debe hacer que los valores de P y Q sean lo más parecidos posible (con Q ligeramente superior a P):

   - T1: 1 nodo –4 núcleos = 4 procesos. $P = 2$, $Q = 2$.
   - T2: 2 nodos –4 núcleos = 8 procesos. $P = 2$, $Q = 4$.
   - T1: 3 nodos –4 núcleos = 12 procesos. $P = 3$, $Q = 4$.

2. Determine el tamaño de la matriz a utilizar teniendo en cuenta la memoria RAM disponible. Recuerde que un valor adecuado suele estar comprendido entre el 80% y 90% de la memoria total. Para determinar este valor, se propone el uso de la herramienta

http://hpl-calculator.sourceforge . net seleccionando el valor asociado al 86% de la memoria para cada test:

- T1: 14080.
- T2: 19840.
- T3: 24320.

Este resultado se puede calcular teniendo en cuenta que se trata de nodos con 2 GB de RAM, que la matriz es cuadrada, que sus elementos ocupan 8 bytes y que el factor para el particionado por bloques es NB=128. El cálculo sería el siguiente (indicando el porcentaje en tanto por uno):

$$max\_elementos\_\% = \sqrt{\frac{\#nodos \cdot memoria\_en\_bytes}{8}} \cdot \%$$

#bloques = *max_elementos_%*/128

tamaño_matriz = *#bloques* 128

En el caso de 1 nodo, para el 86%:

$$\sqrt{\frac{1nodo \cdot 2GB \cdot 1024 \cdot 1024 \cdot 1024}{8}} \cdot 0,86 \approx 14090$$

14090/128≈110

110·128=14080

3. Modifique el fichero *HPL.dat* con los valores calculados y ejecute las pruebas necesarias.

- Para T1, en el fichero *HPL.dat* deben modificarse las líneas:

Fichero: /SHARED/hpl/bin/rpi/HPL.dat

```
14080           Ns
2               Ps
2               Qs
```

- Para T2, en el fichero *HPL.dat* deben modificarse las líneas:

Fichero: /SHARED/hpl/bin/rpi/HPL.dat

```
19840           Ns
2               Ps
4               Qs
```

- Para T3, en el fichero *HPL.dat* deben modificarse las líneas:

Fichero: /SHARED/hpl/bin/rpi/HPL.dat

```
24320           Ns
3               Ps
4               Qs
```

Las pruebas se realizan con los siguientes comandos:

```
pi@nodo0:~ $
    export PATH=/SHARED/mpich-3.3.2-fort/bin:$PATH
pi@nodo0:~ $
    export LD_LIBRARY_PATH=/SHARED/mpich-3.3.2-fort/lib:
                $LD_LIBRARY_PATH
pi@nodo0:~ $
    mpiexec -n 4 --host nodo1 /SHARED/hpl/bin/rpi/xhpl
COMENTARIO: Una salida resumida de dicha ejecución es:
T/V             N    NB     P     Q    Time          Gflops
--------------------------------------------------------------------
WR11C2R4    9856   128     2     2  113.61     5.6197e+00
```

```
pi@nodo0:~ $ mpiexec -n 8 --host nodo1,nodo2
    /SHARED/hpl/bin/rpi/xhpl
COMENTARIO: Una salida resumida de dicha ejecución es:
T/V             N    NB     P     Q    Time     Gflops
--------------------------------------------------------------------
WR11C2R4    19840   128     2     4  498.69 1.0441e+01
```

```
pi@nodo0:~ $ mpiexec -n 12 --host nodo1,nodo2,nodo3
    /SHARED/hpl/bin/rpi/xhpl
COMENTARIO: Una salida resumida de dicha ejecución es:
T/V             N    NB     P     Q     Time       Gflops
--------------------------------------------------------------------
WR11C2R4    24320   128     3     4   700.87   1.3684e+01
```

4. ¿Escala correctamente el sistema?

   Algunos ejemplos de los GFLOPS obtenidos al ejecutar los tests son los siguientes:

Número de nodos	GFLOPS
1	5,6197
2	10,044
3	13,684

A partir de los valores anteriores se puede calcular la proporción de mejora al añadir recursos al cómputo. Es decir, teniendo en cuenta el rendimiento con un único nodo, se puede calcular si la mejora es la esperada.

- Al utilizar dos nodos (mejora teórica=2): 10,044/5,6197 –1,79.
- Al utilizar tres nodos (mejora teórica=3): 13,684/5,6197 –2,44

Los resultados muestran que la escalabilidad es buena; tenga en cuenta que debido a retardos introducidos por el hardware y a los detalles de implementación software, es imposible que la escalabilidad sea perfecta.

Ejercicio 5

Lleve a cabo un análisis de escalabilidad de la aplicación LAMMPS utilizando MPICH como biblioteca MPI y lanzando los tests necesarios a través del gestor de colas Slurm.

1. Determine cuál es la configuración óptima de nodos y procesos para el dominio *(x,y,z) = (4,4,4)* y el caso *in.lj* ubicado en *./ bench* dentro del directorio de instalación de LAMMPS. Por ejemplo, si la instalación se encuentra en */SHARED/-lammps* , se podría ejecutar LAMMPS en dos nodos y con dos procesos por nodo del siguiente modo:

mpirun –n 4 --host nodo2 , nodo3 / SHARED / lammps /src/ -lmp_mpi –var x 4 –var y 4 –var z 4 –in / SHARED / -lammps /

bench /in . lj

2. ¿Podría llevarse a cabo el mismo análisis de escalabilidad pero para el tamaño de dominio *(x,y,z) = (8,8,8)* ? ¿Cuál sería la configuración que ofrece mayor rendimiento?

3. ¿Cuál sería la configuración óptima para un dominio con dimensiones *(x,y,z) = (2,2,2)* ?

Solución 5

Se asume que el estudio se va a realizar en un clúster con tres nodos de cómputo. Esta solución aprovecha el planificador de cargas de trabajo Slurm para gestionar los distintos trabajos que debemos ejecutar.

1. Dominio *(x,y,z) = (4,4,4)*

   Primero se creará el fichero para lanzar el trabajo */SHARED/lammps.sbatch* con el siguiente contenido:

Fichero: lammps.sbatch

```
#!/bin/bash

export PATH=/SHARED/mpich-3.3.2-install/bin:$PATH

nodelist=$(scontrol show hostname $SLURM_JOB_NODELIST
    ↩ | paste -d, -s)

mpirun -n $SLURM_NTASKS -host $nodelist /SHARED/lammps
    ↩ /src/lmp_mpi -var x 4 -var y 4 -var z 4 -in /
    ↩ SHARED/lammps/bench/in.lj
```

A continuación, desde el directorio compartido (*/SHARED*) se lanzarán los trabajos con las configuraciones deseadas. Nótese que el fichero */SHARED/lammps.sbatch* deberá tener permisos de ejecución. En este caso se han ejecutado los siguientes comandos:

- Tres nodos, cuatro procesos por nodo:

  sbatch -N3 -n12 lammps.sbatch

  O también:

  sbatch -n12 lammps.sbatch

- Tres nodos, dos procesos por nodo:

  sbatch -N3 -n6 lammps.sbatch

- Tres nodos, un proceso por nodo:

  sbatch -N3 -n3 lammps.sbatch

- Dos nodos, cuatro procesos por nodo:

  sbatch -N2 -n8 lammps.sbatch

  O también:

  sbatch -n8 lammps.sbatch

- Dos nodos, dos procesos por nodo:

  sbatch -N2 -n4 lammps.sbatch

- Dos nodos, un proceso por nodo:

  sbatch -N2 -n2 lammps.sbatch

- Un nodo, cuatro procesos por nodo:

  sbatch -n4 lammps.sbatch

- Un nodo, dos procesos por nodo:

  sbatch -n2 lammps.sbatch

- Un nodo, un proceso por nodo:

  sbatch -n1 lammps.sbatch

Como posible resultado se obtienen estos tiempos:

	1 ppn	2 ppn	3 ppn
1 nodo	0:07:24	0:03:59	0:02:25
2 nodos	0:03:55	0:02:03	0:01:20
3 nodos	0:02:42	0:01:28	0:00:58

De ellos se deduce que el máximo rendimiento se obtiene utilizando tres nodos y cuatro procesos por nodo.

2. Dominio *(x,y,z) = (8,8,8)*

Para este dominio no se pueden ejecutar los mismos trabajos que en el caso anterior, ya que al ser un dominio más grande se necesita la memoria de al menos tres nodos. Por lo que el estudio quedaría del siguiente modo:

	1 ppn	2 ppn	3 ppn
1 nodo	-	-	-
2 nodos	-	-	-
3 nodos	0:25:14:	0:13:29	0:07:12

De nuevo, tres nodos y cuatro procesos por nodo es la configuración que ofrece mayor rendimiento.

3. Dominio *(x,y,z) = (2,2,2)*

En el caso de dominios más pequeños, se puede ver en los tiempos que el hecho de añadir más nodos no supone una notable mejora.

	1 ppn	2 ppn	3 ppn
1 nodo	0:00:54	0:00:29	0:00:18
2 nodos	0:00:29	0:00:16	0:00:11
3 nodos	0:00:20	0:00:11	0:00:08

Debido a que la cantidad de cómputo es menor en dominios pequeños, el coste de transferir datos entre nodos se amortiza menos. Por tanto, aunque el máximo rendimiento se obtiene de nuevo con tres nodos, opciones de ejecución que requieran menos

nodos pueden ser alternativas atractivas. Por ejemplo, imagínese que el coste económico de ejecutar un programa estuviera tasado en 0,01 € por segundo y procesador. En ese caso, utilizar 12 procesadores durante ocho segundos supondría un coste de 0,96 € , mientras que si se utilizaran 8 procesadores durante 11 segundos, el coste sería 0,88 € .

Ejercicio 6

Partiendo de la configuración de Slurm presentada en el Apartado 6.1.3, configure Slurm de modo que el *front-end* actúe también como nodo de cómputo. Además, cree dos particiones: una debe poder compartir nodos entre los trabajos, y la otra ,asignar nodos de forma exclusiva a los trabajos. Tenga en cuenta que el hecho de poder compartir nodos entre trabajos hace que el recurso mínimo sea la CPU en vez de el nodo.
Por ejemplo, al ejecutar el comando sinfo para que muestre en la salida un formato concreto de la información, se podría obtener lo siguiente:

```
pi@nodo0:~ $ sinfo --format="%.9P %.5a %.5D %.6t %13h %N"
PARTITION AVAIL NODES  STATE OVERSUBSCRIBE NODELIST
   shared    up     1   idle NO            nodo[0-1]
    main*    up     1   idle EXCLUSIVE     nodo[2-3]
```

Solución 6

Para modificar la configuración de Slurm, habrá que detener los

demonios, tanto el controlador como los de cómputo.

Lo primero que se va a hacer es añadir el *front-end* del clúster RPi como nodo de cómputo. Para ello habrá que añadirlo al fichero *slurm.conf*. En este fichero se definen las características del nodo que se pueden obtener mediante la siguiente ejecución:

```
pi@nodoo:~ $ slurmd -C
NodeName=nodoo CPUs=4 Boards=1 SocketsPerBoard=4
    CoresPerSocket=1 ThreadsPerCore=1 RealMemory=1867
    UpTime=4-01:03:29
```

Por tanto, en la configuración se añadirá el nodoo con su configuración y, a su vez, se modificará la partición para incluir lo que sigue. Tenga en cuenta que el texto del siguiente fichero debe escribirse en una única línea.

Fichero: slurm.conf

```
NodeName=nodoo CPUs=4 Boards=1 SocketsPerBoard=4
CoresPerSocket=1 ThreadsPerCore=1 RealMemory =1867
```

Una vez añadido el nodo, debido a que la partición "main" incluye "Nodes=ALL", se iniciarán de nuevo los demonios. Nótese que se pueden utilizar los comandos de gestión de servicios presentes en la Consola 6.6 para iniciar los demonios Slurm explicados en el Apartado 6.1.3. Además, para este caso, habrá que iniciar el demonio slurmd también en el *front-end*, por ejemplo, así:

```
pi@nodoo:~ $ sudo cp/SHARED/slurm-install/slurmd.service
```

```
    /usr/lib/systemd/user/
pi@nodoo:~ $ systemctl --user daemon-reload
pi@nodoo:~ $ systemctl --user start slurmd
```

Como resultado, el comando sinfo debería mostrar algo parecido a esto:

```
pi@nodo0:~ $ sinfo
PARTITION AVAIL NODES  STATE OVERSUBSCRIBE NODELIST
    main*    up     1   idle EXCLUSIVE     nodo[0-3]
```

En el Capítulo 6 se presenta una configuración de Slurm en la que por defecto el recurso mínimo de asignación es el nodo. Lo que se pretende ahora es utilizar como recurso mínimo la CPU. Para ello, se modificará la política de selección de recursos de Slurm para que cambie la unidad mínima de recurso a la CPU. La política a utilizar se llama *consumable resources* y se habilita añadiendo estas líneas al fichero *slurm.conf* :

Fichero: slurm.conf

```
SelectType=select / cons_res
SelectTypeParameters=CR_CPU
```

Ahora es turno de definir y configurar las particiones. Recuerde que se van a configurar dos particiones. Por ejemplo, en esta solución se ha decidido crear:

- Partición “shared”: permitirá compartir los nodos entre los trabajos. Esto significa que los trabajos de esta partición podrán pedir CPU para su ejecución en lugar de nodos completos. Esta partición la compondrán el nodoo y el nodo1 .
- Partición “main”: asignará nodos completos a los trabajos y la

compondrán el nodo2 y el nodo3 .

Esta configuración se puede implementar modificando el fichero *slurm.conf* como se muestra a continuación. De nuevo, la configuración para cada partición debe indicarse en una única línea.

Fichero: slurm.conf

```
PartitionName=shared Nodes=nodo0 MaxTime=INFINITE State=UP
PartitionName=main Nodes=nodo1 OverSubscribe=EXCLUSIVE
    ↩ Default=YES
    MaxTime=INFINITE State=UP
```

Acto seguido, tan solo quedará reiniciar los demonios de nuevo. Para comprobar el correcto funcionamiento de la configuración, aparte de ejecutar sinfo y ver que las particiones coinciden con lo esperado, se pueden lanzar varios trabajos a la partición “shared” del siguiente modo:

```
pi@nodo0:~ $ srun -n2 -p shared sleep 100 &
pi@nodo0:~ $ srun -n2 -p shared sleep 100 &
pi@nodo0:~ $ squeue
 JOBID PARTITION     NAME     ST    TIME  NODES NODELIST(REASON)
    29    shared    sleep     R    0:12      1 nodo0
    30    shared    sleep     R    0:03      1 nodo0
```

El comando squeue muestra que hay dos trabajos ejecutándose concurrentemente en el nodo0.

Ejercicio 7

El objetivo final de este ejercicio es ejecutar un contenedor que contenga la aplicación LAMMPS junto MPICH mediante la creación de un *job* en

Slurm. Para ello, se ha desglosado este objetivo en varios apartados:

1. Creación de un contenedor con la biblioteca MPICH. Puede realizar la misma instalación que en el Apartado 5.2.2 o apoyarse en la que está disponible en la documentación de Singularity *a* . La imagen base debe ser Ubuntu 18.04 o posterior. Para ello puede utlizarse el Bootstrap: docker y la imagen *ubuntu:18.04* .
2. Es recomendable comprobar que la instalación de MPICH se ha realizado correctamente. Para ello, se propone la creación de una imagen que tenga como imagen base la imagen creada en el apartado anterior y que contenga, compile y ejecute el programa *hola_mundo_mpi.c* .
3. Creación de un contenedor que, a partir de la imagen creada en el primer apartado *b* , contenga una instalación de LAMMPS (se recomienda apoyarse en el Apartado 7.2.1 para realizar la instalación). Si la imagen de contenedor resultante se llama *lammps.sif* y la instalación de LAMMPS se encuentra en */opt/lammps-stable_29Oct2020* , podría probar la instalación con el siguiente comando:

```
pi@nodo0:/SHARED/SingularityImages/lammps
    $ singularity exec lammps.sif
    /opt/lammps-stable_29Oct2020/src/lmp_mpi -var x 3
    -var y 3 -var z 3 -in
    /opt/lammps-stable_29Oct2020/bench/in.lj
```

4. Ejecución de la imagen generada en el Apartado 3 *c* como un *job* de Slurm.

a https://sylabs.io/guides/3.5/user-guide/mpi.html

b Si no se ha conseguido completar el Apartado 1, la imagen *serlophug/ubuntu18/mpich:3.3* alojada en el repositorio de imágenes de contenedor de Sylabs podría ser usada como imagen base.

c Si no se ha conseguido completar el Apartado 3, puede usarse la imagen *serlophug/ubuntu18/lammps:latest* del repositorio de imágenes para este apartado.

Solución 7

Siguiendo con la estrategia utilizada a lo largo del Capítulo 8, las imágenes de contenedor y sus recetas de creación se almacenarán en distintas carpetas del directorio compartido */SHARED/SingularityImages*.

1. La creación de la imagen del contenedor de MPICH puede realizarse a partir de la receta *mpich.def* . En esta receta se realiza una instalación de MPICH versión 3.3 en el directorio */opt/mpich* . Cabe remarcar que esta receta es una variación de la que está disponible en la documentación de Singularity.

   Fichero: /SHARED/SingularityImages/mpich/mpich.def

```
Bootstrap: docker
From: ubuntu:18.04

%environment
    export MPICH_DIR=/opt/mpich
    export SINGULARITY_MPICH_DIR=$MPICH_DIR
    export SINGULARITYENV_APPEND_PATH=$MPICH_DIR/bin
    export SINGULAIRTYENV_APPEND_LD_LIBRARY_PATH=
        ↩ $MPICH_DIR/lib

%post
    echo "Installing required packages..."
    apt-get update
    apt-get install -y wget git bash gcc gfortran g++
        ↩ make

    # Information about the version of MPICH to use
    export MPICH_VERSION=3.3
    export MPICH_URL="http://www.mpich.org/static/
        ↩ downloads/$MPICH_VERSION/mpich-
        ↩ $MPICH_VERSION.tar.gz"
    export MPICH_DIR=/opt/mpich

    echo "Installing MPICH..."
    mkdir -p /tmp/mpich
    mkdir -p /opt

    # Download
    cd /tmp/mpich
    wget -O mpich-$MPICH_VERSION.tar.gz $MPICH_URL
    tar xzf mpich-$MPICH_VERSION.tar.gz

    # Compile and install
    cd /tmp/mpich/mpich-$MPICH_VERSION
    ./configure --prefix=$MPICH_DIR
    make install
```

Una vez el fichero se ha generado, es necesario ejecutar la orden siguiente para crear la imagen de contenedor:

```
pi@nodo0:/SHARED/SingularityImages/mpich
    $ sudo singularity build -F mpich.sif mpich.def
INFO:    Starting build...
...
INFO:    Creating SIF file...
INFO:    Build complete: mpich.sif
```

Esta imagen de contenedor también está disponible para ser descargada en el repositorio de imágenes de Sylabs con el nombre *serlophug/ubuntu18/mpich:3.3*.

2. A continuación, se va generar la imagen de contenedor con *hola_mundo_mpi.c* para comprobar que funciona correctamente la instalación de MPICH. Para poder generar esta imagen a partir de la receta *hola-mundo.def* (definida a continuación) es necesario que el fichero *hola_mundo_mpi.c* se encuentre en el directorio de la receta. Por esta razón, hay que copiar este fichero *hola_mundo_mpi.c* dentro de él. Si el fichero se encuentra en */home/pi* , entonces esto puede hacerse con el siguiente comando:

```
pi@nodoo:/SHARED/SingularityImages/mpich$ cp
    /home/pi/hola_mundo_mpi.c .
```

Fichero:/SHARED/SingularityImages/mpich/holamundo.def

```
Bootstrap: localimage
From: /SHARED/SingularityImages/mpich/mpich.sif

%files
    hola_mundo_mpi.c /opt

%post
    # Set env variables so we can compile our
        ↩ application
    export MPICH_DIR=/opt/mpich
    export PATH=$MPICH_DIR/bin:$PATH
    export LD_LIBRARY_PATH=$MPICH_DIR/lib:
        ↩ $LD_LIBRARY_PATH
    export MANPATH=$MPICH_DIR/share/man:$MANPATH

    # Compiling the application
    cd /opt && mpicc -o hola_mundo_mpi hola_mundo_mpi.
        ↩ c

%runscript
    /opt/hola_mundo_mpi
```

De la misma forma que en el Apartado 1, ahora hay que generar la imagen del contenedor. Los siguientes comandos corresponden a la creación de la imagen y su posterior ejecución en todos los nodos de cómputo:

```
pi@nodo0:/SHARED/SingularityImages/mpich/hola-mundo
     $sudo singularity build -F hola-mundo.sif
     hola-mundo.def
INFO:    Starting build...
...
INFO:    Creating SIF file...
INFO:    Build complete: hola-mundo.sif

pi@nodo0:/SHARED/SingularityImages/mpich/hola-mundo
     $  mpirun -n 3 --host nodo1,nodo2,nodo2
     /SHARED/singularity/bin/singularity run
     hola-mundo.sif
Hola mundo desde nodo1, con rango 0 de un total de 3
     procesadores
Hola mundo desde nodo2, con rango 1 de un total de 3
     procesadores
Hola mundo desde nodo3, con rango 2 de un total de 3
     procesadores
```

3. Si se han completado correctamente los dos apartados anteriores, entonces es el momento de crear la imagen de contenedor que contendrá LAMMPS.

Fichero:/SHARED/SingularityImages/lammps.def

```
Bootstrap: library
From: serlophug/ubuntu18/mpich:3.3

%labels
    author Sergio López-Huguet
    email serlohu@upv.es
    created 30/11/2020


%post
    export MPICH_DIR=/opt/mpich
    export LAMMPS_DIR=/opt
    export PATH=$MPICH_DIR/bin:$PATH
    export LD_LIBRARY_PATH=$MPICH_DIR/lib:
        ↩ $LD_LIBRARY_PATH
    export MANPATH=$MPICH_DIR/share/man:$MANPATH
    export LAMMPS_PACKAGE_NAME=stable_29Oct2020

    apt-get  update
    apt-get -y install wget bash gcc gfortran g++ make
        ↩  file unzip

    cd ${LAMMPS_DIR}
    wget https://github.com/lammps/lammps/archive/${
        ↩ LAMMPS_PACKAGE_NAME}.zip
    unzip ${LAMMPS_PACKAGE_NAME}.zip
    rm ${LAMMPS_PACKAGE_NAME}.zip

    cd ${LAMMPS_DIR}/lammps-${LAMMPS_PACKAGE_NAME}/src
    make mpi

%environment
    export MPICH_DIR=/opt/mpich
    export SINGULARITY_MPICH_DIR=$MPICH_DIR
    export SINGULARITYENV_APPEND_PATH=$MPICH_DIR/bin
    export SINGULAIRTYENV_APPEND_LD_LIBRARY_PATH=
        ↩ $MPICH_DIR/lib

%test
    export MPICH_DIR=/opt/mpich
    export SINGULARITY_MPICH_DIR=$MPICH_DIR
    export PATH=$MPICH_DIR/bin
    export LD_LIBRARY_PATH=$MPICH_DIR/lib:
        ↩ LD_LIBRARY_PATH

    LAMMPS_DIR=/opt
    LAMMPS_PACKAGE_NAME=stable_29Oct2020
    SIZE=3

    ${LAMMPS_DIR}/lammps-${LAMMPS_PACKAGE_NAME}/src/
        ↩ lmp_mpi -var x ${SIZE} -var y ${SIZE} -var
        ↩ z ${SIZE} -in ${LAMMPS_DIR}/lammps-${
        ↩ LAMMPS_PACKAGE_NAME}/bench/in.lj
```

Nótese que se ha definido la sección test en la receta para evitar el uso del

comando proporcionado en el enunciado. De hecho, ambos comandos son idénticos.

Si no se dispone de una imagen de contenedor con MPICH propia, puede utilizarse *serlophug/ubuntu18/mpich:3.3* como imagen base sustituyendo la cabecera de la receta:

```
Bootstrap : library
From : serlophug / ubuntu18 / mpich : 3 . 3
```

Una vez se ha creado la receta, se puede proceder a crear la imagen de contenedor:

```
pi@nodo0:/SHARED/SingularityImages/lammps
    $ sudo singularity build -F lammps.sif lammps.def
INFO:    Starting build...
...
INFO:    Creating SIF file...
INFO:    Build complete: lammps.sif
```

Completada la creación de la imagen de contenedor, la primera prueba consiste en ejecutar la aplicación en un solo nodo. Esto puede hacerse con el primer comando. El segundo comando corresponde a la ejecución del contenedor en todos los nodos de cómputo. La salida de cada comando debe ser muy similar a la ilustrada en la Figura 7.3 (esta ejecución de LAMMPS se realizó sin usar contenedores y con dos nodos).

```
pi@nodo0:/SHARED/SingularityImages/lammps
     $ singularity test lammps.sif
LAMMPS (29 Oct 2020)
Lattice spacing in x,y,z = 1.6795962 1.6795962 1.6795962
Created orthogonal box = (0.0000000 0.0000000 0.0000000)
     to (100.77577 100.77577 100.77577)
  1 by 1 by 1 MPI processor grid
...
pi@nodo0:/SHARED/SingularityImages/lammps
     $ mpirun -n 3 --host nodo1,nodo2,nodo3
     /SHARED/singularity/bin/singularity test lammps.sif
LAMMPS (29 Oct 2020)
Lattice spacing in x,y,z = 1.6795962 1.6795962 1.6795962
Created orthogonal box = (0.0000000 0.0000000 0.0000000)
     to (100.77577 100.77577 100.77577)
  1 by 1 by 3 MPI processor grid
...
```

4. Para lanzar la ejecución como un *job* de Slurm es necesario crear un documento con la especificación del *job* . Partiendo del fichero */SHARED/Singu-larityI-mages/ompi/hola-mundo/mundo.sbatchtch* , un ejemplo podría ser:

Fichero:/SHARED/SingularityImages/lammps/lammps.sh #!/bin/ bash

```
#!/bin/bash
#SBATCH --nodes 3
#SBATCH --job-name lammps

SINGULARITY=/SHARED/singularity/bin/singularity
IMAGE=/SHARED/SingularityImages/lammps/lammps.sif

mpirun -n 3 --host nodo1,nodo2,nodo3 $SINGULARITY test
    ↩ $IMAGE
```

Una vez se ha generado la especificación del *job*, los siguientes comandos se encargan de enviar el trabajo a Slurm, de comprobar la cola para ver

que está en ejecución y, una vez el trabajo termine, de mostrar las primeras diez líneas del fichero de salida.

```
pi@nodo0:/SHARED/SingularityImages/lammps
     $ sbatch lammps.sh
Submitted batch job 30
pi@nodo0:/SHARED/SingularityImages/lammps $ squeue
             JOBID PARTITION     NAME     USER ST
     TIME  NODES NODELIST(REASON)
                30      main   lammps       pi  R
     0:05      3 nodo[1-3]
pi@nodo0:/SHARED/SingularityImages/lammps
     $ head slurm-30.out
LAMMPS (29 Oct 2020)
Lattice spacing in x,y,z = 1.6795962 1.6795962 1.6795962
Created orthogonal box = (0.0000000 0.0000000 0.0000000)
     to (100.77577 100.77577 100.77577)
  1 by 1 by 3 MPI processor grid
Created 864000 atoms
  create_atoms CPU = 0.192 seconds
Neighbor list info ...
  update every 20 steps, delay 0 steps, check no
  max neighbors/atom: 2000, page size: 100000
  master list distance cutoff = 2.8
```

Parte V

Anexos

A. Temporización aproximada

En este anexo se detallan los tiempos aproximados de los procesos de compilación, instalación, ejecución, etc. integrados en cada uno de los apartados que componen los distintos capítulos del libro. Nótese que se trata de tiempos aproximados y que no incluyen el tiempo necesario para leer el contenido presentado en este manual ni comprenderlo o completarlo, sino exclusivamente el invertido en trabajar sobre el propio clúster y esperar a que cada etapa se complete. Asimismo, el tiempo necesario para completar las actividades tampoco se incluye.

Pasos previos	
Flash de la tarjeta SD con el SO	20 minutos (cada SD)
Montaje hardware	10 minutos

Configuración inicial	
Nodo cómputo - configuración básica	5 minutos
Nodo cómputo - configuración red	5 minutos

Front-end - configuración básica	5 minutos
Front-end - configuración red	10 minutos
Sistema de ficheros compartido	10 minutos

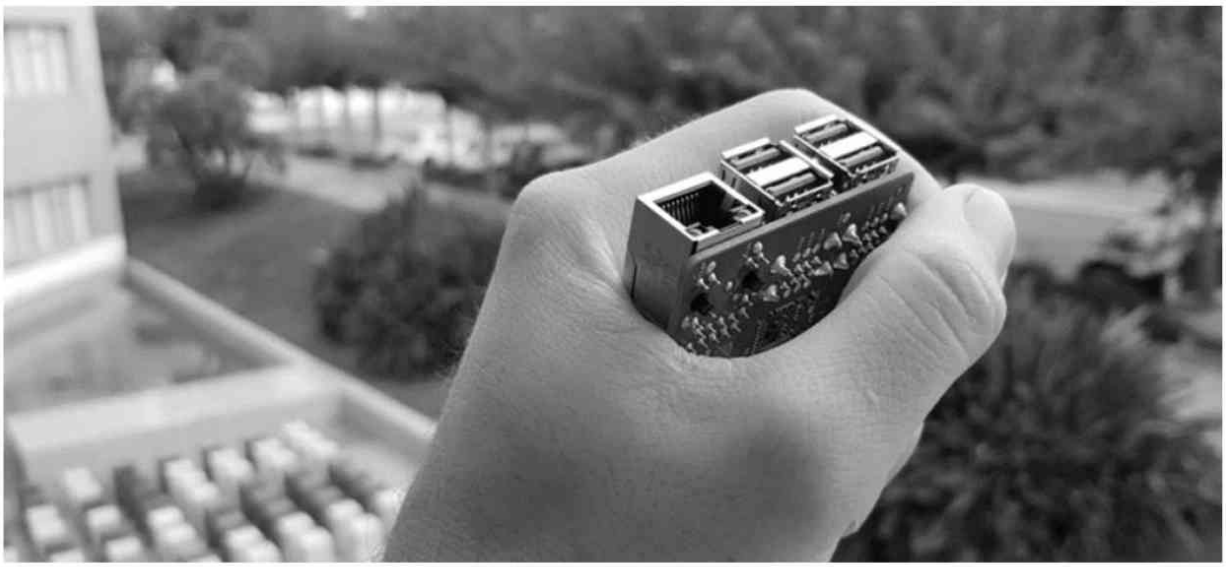
Modelos de programación paralela	
Ejemplo *hola_mundo* en serie	2 minutos
Intra-nodo: OpenMP	5 minutos
Instalación de MPI - OpenMPI	23 minutos
Instalación de MPI - MPICH	15 minutos
Ejemplo	5 minutos

Gestor de trabajos y recursos	
Instalación de los paquetes necesarios	2 minutos
Instalación de MUNGE	5 minutos
Instalación, configuración y prueba de Slurm	11 minutos

Aplicaciones de rendimiento y monitorización del sistema	
cpufreq-utils	1 minuto
stress-ng	15 minutos
stream	1 minuto
iperf	1 minuto
LINPACK - instalación	18 minutos
LINPACK - prueba con 12 procesos	13 minutos

Aplicaciones científicas	
LAMMPS - instalación	12 minutos
LAMMPS - pruebas sin OpenMP	5 minutos
LAMMPS - compilación con OpenMP	10 minutos
LAMMPS - pruebas con OpenMP	3 minutos
OpenFOAM - instalación	14 horas
OpenFOAM - pruebas	2 minutos

Software de virtualización	
Singularity - instalación	10 minutos
Singularity - creación del contenedor *lolcow*	10 minutos
Singularity - ejecución de contenedores	10 minutos
Singularity - caso de uso	1 hora 10 minutos



B. Raspberry Pi Model 3 B+

En este anexo se describen algunos aspectos de la Raspberry Pi 3 Model B+, en adelante referida como RPi3. Concretamente, se presentan algunas de las diferencias más destacables respecto a la Raspberry Pi 4 Model B. Las especificaciones técnicas de la RPi3[1] proporcionadas por el fabricante se resumen en la Tabla B.1. Los componentes presentados en esta tabla se disponen en la placa de la RPi3 como se muestra en la Figura B.1.

B.1 Comunicaciones internas entre componentes

Internamente la RPi3 establece las conexiones entre componentes tal y como muestra la Figura B.2. Aunque la RPi3 cuenta con conectividad Gigabit Ethernet (ofrece un ancho de banda teórico de 1000 Mbps), este puerto comparte el controlador LAN95142 con los cuatro puertos USB 2.0 (con un ancho de banda teórico de 480 Mbps). A su vez, el controlador se une al procesador mediante un bus que ofrece un ancho de banda máximo de 300 Mbps. Por tanto, la conectividad a la red de interconexión cableada estará limitada a 300 Mbps teóricos, lo que supone un cuello de botella a la hora de comunicarse con otros dispositivos.

Procesador	Broadcom BCM2847B0, Cortex-A53 64-bit SoC 1,4GHz
Memoria RAM	1GB LPDDR2 SDRAM
Wi-Fi	1,4GHz y 5GHz IEEE 802.11.b/g/n/ac Wireless LAN
Bluetooth	4.2, BLE
Puertos USB	4 x USB 2.0
Puerto para cámara	MIPI SCI Camera Port
Puerto para monitor	Full Size HDMI
Puerto para vídeo/audio	4 Pole Stereo Output and Composite Video Port
Puerto para alimentación	5V/2,5A DC via MicroUSB
Puerto Ethernet	Gigabit Ethernet
Otros	Puerto MIPI DSI, 40 Pines GPIO

Tabla B.1 Especificaciones técnicas de la Raspberry Pi 3 Model B+.

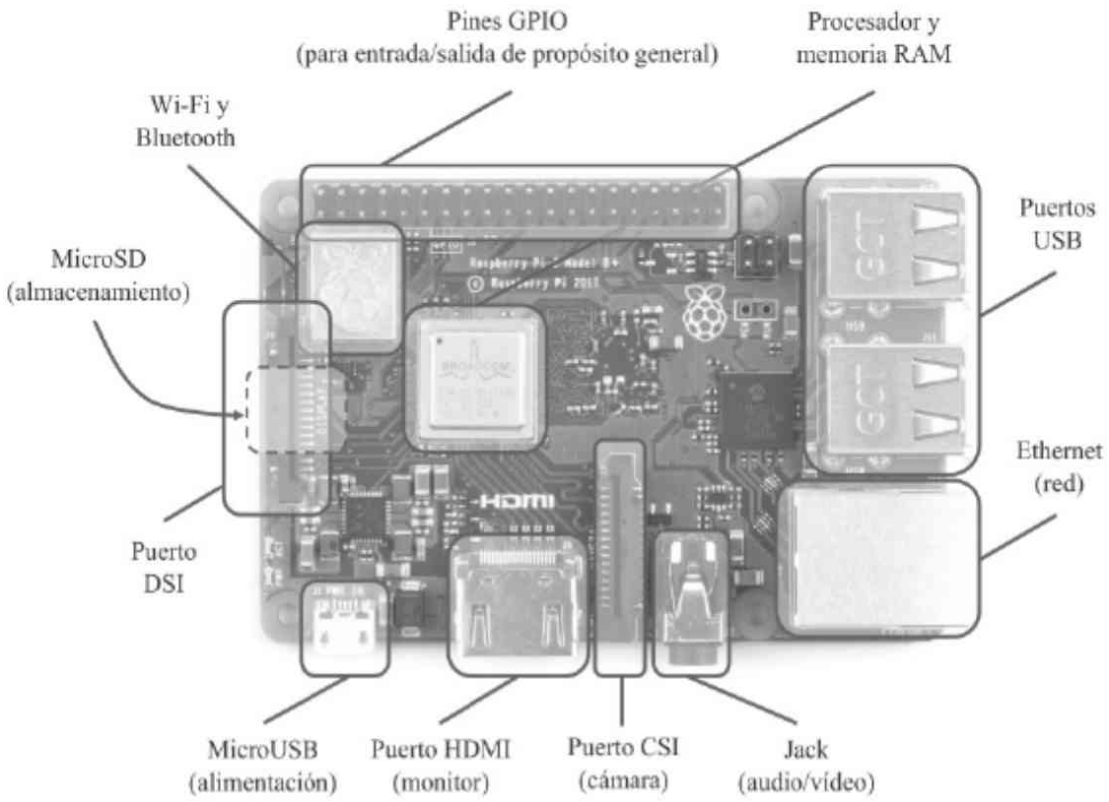


Figura B.1 Esquema de los componentes que integran una Raspberry Pi 3 Model B+.

Algo parecido pasa con el controlador Wi-Fi BCM434383. Aunque el protocolo de red que soporta es el 802.11n con un ancho de banda teórico de 600 Mbps, la interfaz que lo conecta al procesador puede transferir datos a un ratio máximo de 200 Mbps.

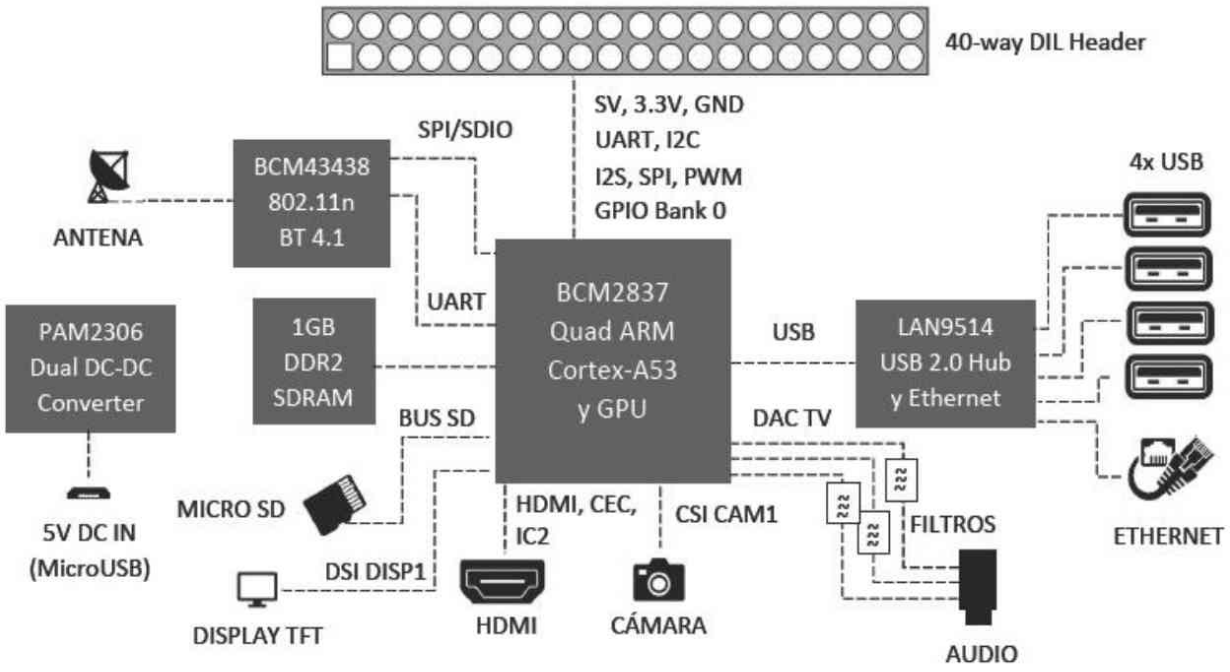


Figura B.2 Esquema de las comunicaciones entre componentes de una Raspberry Pi 3 Model B+[4].

B.2 Interpretación de los indicadores LED

Al igual que la RPi, la RPi3 incorpora dos LED, situados en una esquina de la placa, entre el puerto DSI y el Micro USB, tal como puede observarse en la Figura B.3. Concretamente, estos dos LED están etiquetados como "ACT" y "PWR".

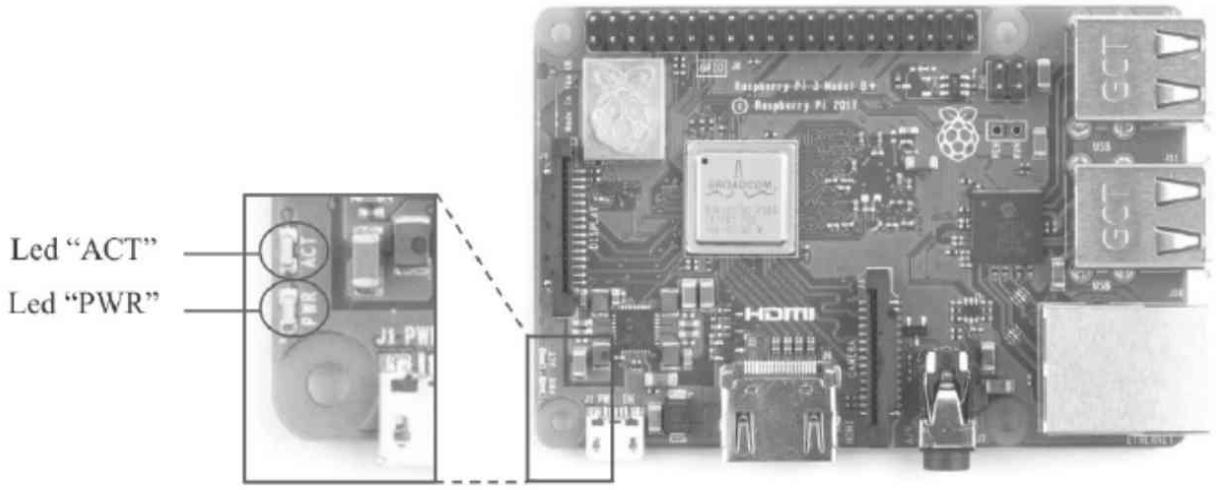


Figura B.3 Detalle de los LED que incorpora la RPi 3B+.

B.2.1 LED ACT

El LED "ACT" es el que emite una luz de color verde y está directamente relacionado con la capacidad de la RPi para leer del dispositivo conectado en el zócalo Micro SD.

Al arrancar la RPi3, si la lectura procede sin incidencias, podrá observarse un parpadeo irregular durante algunos segundos a partir del momento en el que se inicia. En caso contrario, se estará produciendo algún error. La Tabla B.2 muestra el significado de los distintos patrones que se pueden observar frente a distintos problemas.

Parpadeos largos	Parpadeos cortos	Significado
0	3	Fallo genérico al arrancar
0	4	No se ha encontrado (o está corrupto) el fichero *start*.elf* al arrancar
0	7	No se ha encontrado la imagen del núcleo del SO
0	8	No se reconoce la SDRAM
0	9	La SDRAM disponible es insuficiente
0	10	El dispositivo se encuentra en estado *HALT* (parada)
2	1	La partición del sistema de ficheros no es de tipo FAT
2	2	No se ha podido leer desde la partición del sistema de ficheros
2	3	La partición extendida del sistema de ficheros no es de tipo FAT
4	4	No se soporta el tipo de placa
4	5	Se ha producido un error fatal de *firmware*
4	6 ó 7	Fallo de potencia

Tabla B.2 Significado asociado al indicador LED "ACT" según el número y

tipo de parpadeos emitidos.

B.2.2 LED PWR

El comportamiento del indicador LED "PWR" de las RPi3, el cual emite una luz roja, está asociado a la cantidad de energía que la placa está recibiendo. La RPi3 se alimenta con 5 voltios (V) y 2,5 amperios (A).

Si en algún caso (mientras está arrancada la RPi) este indicador LED no está encendido de manera constante, sino que parpadea o incluso se apaga, entonces es necesario revisar de inmediato la fuente de alimentación, ya que muy posiblemente sea insuficiente.

Alternativamente, el hecho de utilizar un software obsoleto o no soportado por la RPi3 también puede provocar que este indicador LED refleje problemas. En tal caso, se observarían también parpadeos en lugar de una luz roja constante.

B.3 Deshabilitar conexiones inalámbricas

Para desconectar las conexiones Wi-Fi y Bluetooth, siguiendo la configuración descrita en el Apartado 4.1.1, para la RPi3 se modificará el fichero */boot/config.txt* añadiendo:

Fichero: /boot/config.txt

```
dtoverlay=pi3-disable-wifi
dtoverlay=pi3-disable-bt
```

Esto se debe hacer con permisos de superusuario. Además, para que estos cambios tengan efecto se debe reiniciar el nodo.

B.4 Temporización aproximada

La Tabla B.3 detalla los tiempos aproximados de los procesos de compilación, instalación, ejecución, etc. integrados en cada uno de los apartados que componen los distintos capítulos del libro para la RPi3.

Si se comparan estas mediciones con las mostradas en el Anexo A, se aprecia un aumento del tiempo debido a la diferencia de prestaciones entre ambos modelos de Raspberry.

Modelos de programación paralela	
Ejemplo *hola_mundo* en serie	2 minutos
Intra-nodo: OpenMP	5 minutos
Instalación de MPI - OpenMPI	40 minutos
Instalación de MPI - MPICH	25 minutos
Ejemplo	5 minutos

Gestor de trabajos y recursos	
Instalación de paquetes necesarios	3 minutos
Instalación de MUNGE	15 minutos
Instalación, configuración y prueba de Slurm	25 minutos

Aplicaciones de rendimiento y monitorización del sistema	
cpufreq-utils	2 minutos
stress-ng	25 minutos
stream	3 minutos
LINPACK - instalación	55 minutos

LINPACK - prueba con 12 procesos	20 minutos

Aplicaciones científicas	
LAMMPS - instalación	25 minutos
LAMMPS - pruebas sin OpenMP	10 minutos
LAMMPS - compilación con OpenMP	20 minutos
LAMMPS - pruebas con OpenMP	5 minutos
OpenFOAM - instalación	25 horas
OpenFOAM - pruebas	5 minutos

Software de virtualización	
Singularity - instalación	15 minutos
Singularity - creación del contenedor *lolcow*	5 minutos
Singularity - ejecución de contenedores	10 minutos
Singularity - caso de uso complejo	1 hora 30 minutos

Tabla B.3 Tiempos aproximados para la ejecución de las tareas listadas.

1 https://static.raspberrypi.org/files/product-brie-Raspberry-Pi-Model-Bplus-Product-Brief.pdff.pdf

2 http://ww1.microchip.com/downloads/en/DeviceDoc/00002306A.pdf

3 https://www.cypress.com/file/298076

4 https://www.element14.com/community/community/raspberry-pi/blog/2017/01/raspberry-rry-pi-3-block-gramgram

Listado de acrónimos

API Application Program Interface.

ARM Acorn RISC Machine.

ASCII American Standard Code for Information Interchange.

CFD Computational Fluid Dynamics.

CISC Complex Instruction Set Computer.

CPU Central Processing Unit.

CSI Camera Serial Interface.

DDR Double Data Rate.

DHCP Dynamic Host Configuration Protocol.

DNS Domain Name System.

DSI Display Serial Interface.

FLOP Floating Point Operation.

FLOPS Floating Point Operations per Second.

GCC GNU Compiler Collection.
GNU GNU's Not Unix.
GPU Graphics Processing Unit.
GUI Graphical User Interface.

HDMI High-definition Multimedia Interface.
HPC High Performance Computing.
HPL High-Performance LINPACK Benchmark.

IP Internet Protocol.

KVM Kernel-based Virtual Machine.

LPDDR Low-Power Double Data Rate.
LXC/LXD Linux Containers.

Mbps Megabits per second.
MPI Message Passing Interface.

NFS Network File System.

OCI Open Container Initiative.

PID Process IDentifier.

RAM Random Access Memory.
RISC Reduced Instruction Set Computer.
RMS Resource Manager System.

SD Secure Digital.

SDHC Secure Digital High Capacity.

SDRAM Synchronous Dynamic Random-Access Memory.

SIF Singularity Image File.

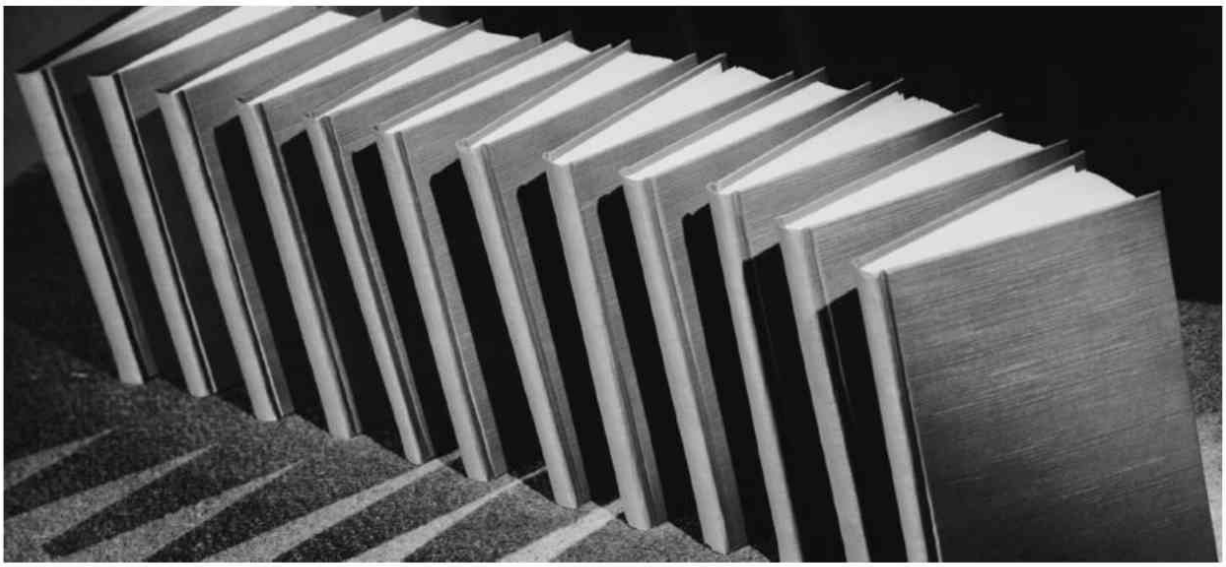
SMB Server Message Block.

SMT Simultaneous Multithreading.

SO Sistema Operativo.

SSH Secure Shell.

USB Universal Serial Bus.



Glosario

Algoritmo Conjunto de reglas que definen el proceso a seguir para resolver un problema.

Ancho de banda Medida de la cantidad máxima de información que se puede transmitir desde un punto a otro en una red, durante un cierto periodo de tiempo dado. Comunmente conocida como *velocidad*..

Arquitectura de un computador Especificación que detalla cómo un conjunto de estándares tecnológicos de software y hardware interactúan para formar un sistema computacional.

Benchmark Aplicación cuya función es medir el nivel de calidad de un sistema, comparándolo con otros.

Biblioteca Software que recoge distintas funcionalidades que pueden ser utilizadas por otro software.

Big Data Vastos conjuntos de datos que pueden ser tratados computacionalmente para detectar patrones, tendencias, asociaciones, etc. comúnmente relacionados con el comportamiento humano y sus interacciones.

Binario Archivo que puede ser ejecutado para iniciar uno o varios programas.

Bucle Estructura de control que permite repetir una o más instrucciones un

determinado número de veces.

Bus En arquitectura de computadores, canal de transferencia de datos entre distintos componentes del computador.

Ciclo de reloj de procesador Intervalo constante entre dos pulsos eléctricos emitidos por el reloj del procesador.

Clúster Conjunto de computadores independientes (nodos) que pueden trabajar colectivamente.

Coma flotante Aritmética utilizada en computación para representar números reales de manera aproximada, dado que el número de decimales representables es limitado.

Concurrencia Simultaneidad de ejecución de un programa por parte de varios procesos.

Contenedor En computación, puede referirse a:

1. Tipo de virtualización que proporciona al usuario un entorno informático aislado sin la emulación de los recursos físicos ni del núcleo del sistema operativo.
2. Una pieza de software ejecutable que contiene una aplicación o servicio junto con sus bibliotecas y dependencias.
3. Instancia de una imagen de contenedor con una serie de recursos asignados con el objetivo de ejecutar uno o más procesos.

Controlador Aplicación que permite al sistema operativo interactuar con un periférico.

Código abierto Modelo de desarrollo de software basado en la colaboración de dominio público mediante el acceso al código fuente del software.

Código máquina Secuencia de unos y ceros interpretable por el procesador.

Demonio Proceso ejecutado en segundo plano que puede ser iniciado como servicio.

Dinámica de fluidos En física e ingeniería, disciplina que describe el flujo de líquidos y gases.

Dominio computacional Simplificación del dominio físico en términos de representación geométrica y condiciones en los límites.

Dominio físico En física y matemáticas, conjunto de valores que puede tomar la variable independiente de una función para los que existe un valor asociado a la variable dependiente.

Domótica Conjunto de sistemas que automatizan una vivienda o edificio y permiten mejorar la seguridad, el bienestar o el consumo energético del mismo, entre otros.

Ecuación lineal Igualdad formada por términos independientes o acompañados de una variable cuyo exponente es siempre uno (motivo por el cual también recibe el nombre de "ecuación de primer grado").

Eficiencia (de un algoritmo) Medida que evalúa cuán bien se aprovechan los recursos utilizados.

Ejecutable Véase *Binario*.

Enlazar Parte del proceso de compilación en el que se combinan todos los códigos máquina involucrados, el código desarrollado por el usuario y también el de las bibliotecas incluidas.

Escalabilidad (de un algoritmo) Propiedad que indica cuán bien responde un algoritmo al cambiar su configuración o capacidad, a partir de la fracción entre dos medidas de tiempo, cada una correspondiente a una configuración o capacidad diferente.

FLOP Operación en coma flotante (véase *Coma flotante*), es decir, aquellas operaciones matemáticas que el procesador es capaz de realizar teniendo en cuenta las limitaciones físicas en cuanto a la precisión (número de decimales) que se puede albergar.

FLOPS Unidad de medida para el rendimiento de un computador que contabiliza el número de FLOP que este puede realizar por segundo.

Frecuencia de reloj En un procesador, medida que indica cuántos ciclos de reloj transcurren en un tiempo determinado.

Front-end Nodo de un clúster con el que interactúan los usuarios del mismo, responsable de coordinar al resto de nodos (particularmente, a los de cómputo) a través de su red privada.

Hardware Conjunto de elementos físicos que componen un sistema.

Hebra Entidad de ejecución independiente que habita en un proceso.

Hilo Véase *Hebra*.

Hipervisor Software encargado de gestionar las máquinas virtuales, así como de asignarles y proporcionarles los distintos recursos físicos en el entorno informático en el que se ejecutan.

Imagen Conjunto de datos inmutables que pueden contener aplicaciones, herramientas, dependencias, bibliotecas y demás configuraciones necesarias para el correcto funcionamiento de aquello que instancia (sistema operativo, contenedor, máquina virtual, etc.) almacenado en un sistema de ficheros.

Interfaz gráfica Software intermediario entre el usuario y la máquina, encargado de mostrar de forma visual todas las acciones posibles en una plataforma, la información disponible, etc.

Interrupción Señal enviada al procesador para indicarle que debe detener el curso de ejecución actual para ejecutar el código específico que trata la situación generada. Una vez ejecutado el código que trata la interrupción, se recupera el flujo de ejecución que se había suspendido.

Job (Slurm) Petición enviada al gestor para la ejecución de un conjunto de instrucciones haciendo uso de una cantidad específica de recursos

cionales.

Latencia Medida de la velocidad a la que se pueden enviar y recibir de vuelta (cliente-servidor) los datos a través de una red determinada.

Lenguaje de alto nivel Lenguaje de programación fácilmente entendible por las personas, es decir, muy cercano al lenguaje humano.

Lenguaje de bajo nivel Lenguaje de programación entendible por las personas debidamente formadas, más cercano al lenguaje del procesador.

Lenguaje máquina Véase *Código máquina*.

Llamada al sistema Mecanismo para solicitar un servicio al sistema operativo desde una aplicación.

Mainframe Predecesora de los actuales servidores, es la computadora central utilizada en las organizaciones para el procesamiento de datos masivos.

Malla En software de simulación, se conoce como malla al conjunto de vértices, caras y aristas que forman los poliedros que dan volumen a un cuerpo geométrico.

Máquina virtual En computación, puede referirse a:

1. Tipo de tecnología de virtualización que proporciona al usuario un entorno informático prácticamente completo e indistinguible de un sistema físico gracias a la emulación tanto del software como de los recursos físicos o hardware.
2. Instancia de una imagen de máquina virtual que contiene todo el software con una serie de recursos hardware asignados.

Máscara de subred Conjunto de cifras (representadas en el sistema binario) que indican qué parte de una dirección IP hace referencia a la subred a la que se encuentra conectado el equipo.

Mínimos cuadrados Técnica de análisis numérico cuyo objetivo es encontrar la función continua que mejor se ajuste, conforme al error cuadrático, a un

conjunto de datos dado y una familia determinada de funciones.

Núcleo En computación, puede referirse a:

1. Núcleo del procesador: conjunto completo de registros y unidades de ejecución dentro de un procesador.
2. Núcleo del sistema operativo: software que realiza la comunicación efectiva entre el hardware y el software. Constituye la parte central del sistema operativo y se ejecuta en modo privilegiado.

Orquestador Herramienta que permite la gestión de la ejecución de los distintos contenedores en una infraestructura informática.

Periférico Componente hardware que no pertenece originalmente al computador al que se enlaza, pero conectado al mismo permite incluir nuevas funcionalidades (por ejemplo, un monitor).

Procesador Conjunto de una o más unidades de procesamiento que comparten la memoria principal y los periféricos.

Procesador lógico Véase *SMT*.

Procesamiento distribuido Ejecución de un programa de forma con-junta por parte de varios procesos que están localizados en sistemas independientes (al menos, en términos de memoria).

Procesamiento paralelo Ejecución de un programa de forma conjunta por parte de varios procesos de modo concurrente.

Proceso Entidad independiente de ejecución gestionada por el sistema operativo. Se compone del conjunto de instrucciones de un programa a ejecutar, su estado de ejecución en cada momento, memoria en la que se almacenan los datos a tratar y toda la información que permite al sistema operativo decidir qué proceso se ejecuta en cada momento.

Programa Aplicación software que implementa un algoritmo para ser

tado en un computador.

Receta En computación, documento de texto que contiene una especificación de toda la configuración necesaria para la creación de imágenes de contenedores. Incluye el conjunto de instrucciones para la instalación y configuración de aplicaciones, la información sobre la imagen base utilizada y la información sobre el contenedor que se creará.

Regulardor Política del sistema operativo que determina la frecuencia del procesador en cada momento.

Reloj del sistema operativo Reloj software que mantiene la fecha y hora en el sistema operativo.

Reloj interno de la CPU Componente del microprocesador que emite una serie de pulsos eléctricos a intervalos constantes llamados ciclos, el inicio de los cuales determina el instante en el que puede ejecutarse un paso de una instrucción.

Repositorio Espacio que almacena información digital.

Repositorio de imágenes Espacio donde se almacenan las imágenes de contenedores.

Script Archivo que contiene instrucciones en un cierto lenguaje de programación. Este puede ser ejecutable.

Sistema de ficheros Conjunto de estructuras de datos y algoritmos para almacenar, localizar y recuperar información de un dispositivo de almacenamiento persistente perteneciente a un computador.

Sistema operativo Programa informático principal que gestiona los recursos hardware y software disponibles en un computador.

SMT Conjunto de recursos hardware que componen un único núcleo, también conocido como procesador lógico.

Software Conjunto de programas, instrucciones y reglas informáticas para

ejecutar tareas en un computador.

Step (Slurm) "Subtrabajo" que se crea dentro de un trabajo para ejecutar un código en los recursos asignados.

Strong scaling Relación entre tiempo de ejecución y número de procesos para un tamaño de problema total fijo.

Supercomputador Véase *Clúster*.

Superusuario En sistemas Unix, cuenta de usuario que posee todos los permisos para administrar el sistema operativo.

Task (Slurm) Cada una de las tareas de un trabajo que se ejecutan en una CPU.

Topología En redes de computadores, disposición física o lógica de los componentes.

Variable de entorno Nombre al que se asocia una cadena de caracteres, que podrá ser utilizada por el sistema operativo y sus aplicaciones para determinar parte o toda su configuración. Su valor puede ser cambiado de forma dinámica, por lo que puede afectar al comportamiento de los procesos en ejecución del sistema.

Vector En ciencia de computadores, un vector es un conjunto de datos ordenados, donde cada uno de los elementos se identifica con un índice que equivale a la posición que ocupa.

Virtualización Tecnología que permite la emulación de un entorno informático (de manera completa o parcial) dentro de otro entorno informático físico de manera aislada.

Weak scaling Relación entre tiempo de ejecución y número de procesos para un tamaño de problema fijo por proceso.

Zócalo En electrónica, soporte que un componente electrónico ofrece para conectar otro componente sobre él.

Zócalo En electrónica, soporte que un componente electrónico ofrece para conectar otro componente sobre él.