UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Análisis comparativo de arquitecturas de MareNostrum y desarrollo de un entorno de monitorización

MEMORIA DEL TRABAJO FINAL DE GRADO

Felix Fernando Ramos Velázquez

Director
Eduard Ayguadé Parra
Arquitectura de Computadores

Codirector
David Vicente Dorca
Operaciones - BSC

Grado en Ingeniería Informática
Especialidad Ingeniería de Computadores
FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)
UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) – BarcelonaTech

Fecha de defensa: 29 de enero de 2019

Resumen

Los supercomputadores necesitan ser reemplazados de forma periódica, siendo el tiempo de vida habitual para una máquina de alrededor de 3 años. Esto implica la necesidad de probar el rendimiento de las nuevas máquinas durante la fase de instalación, para asegurar que la nueva máquina cumpla con los criterios incluidos en la adquisición. Además, también es importante obtener factores de conversión entre máquinas, ya que son necesarios para la migración de proyectos. Los proyectos, que se inician en la máquina antigua, tienen horas concedidas, que deben traducirse a la nueva. Por estas razones, es un requisito para cualquier centro de HPC trabajar con un conjunto de pruebas de referencia con aplicaciones reales y sintéticas y utilizar algunas medidas que pueden usarse para comparar el rendimiento de las máquinas y encontrar las ventajas y desventajas de cada arquitectura.

El objetivo de este proyecto es encontrar los puntos de referencia y las métricas correctas para comparar arquitecturas como las de los clústeres MareNostrum 3 y MareNostrum 4. Un análisis profundo de estas dos arquitecturas permitirá comprender qué mejoras ofrece la nueva máquina y cuáles son las mejores aplicaciones para ejecutar en cada entorno.

Abstract

Supercomputers need to be replaced in a periodically, with the usual life time for a machine being around 3 years. This implies the need to test the performance of new machines during the installation phase, to assure the new machine fulfils the criteria included in the procurement. In addition, it is also important to get conversion factors between machines, as they are needed for the migration of projects. Projects , which are started in the old machine, are granted hours, which need to be translated to the new one . For these reasons, it is a requirement for any HPC center to work with a benchmark suite with real and synthetic applications and use some metrics that can be used to compare the performance of the machines and find the advantages and disadvantages of each architecture.

The goal of this project is to find the right benchmarks and metrics to compare architectures like those in MareNostrum 3 and MareNostrum 4 clusters. A deep analysis of these two architectures will make it possible to understand what improvements the new machine provides and what are the best applications to run in each environment.

Índice general

1. Introducción y contextualización **3**

1.1. Contexto 3

1.2. Motivación 4

1.3. Actores implicados 5

1.4. Estado del arte 8

2. Alcance del proyecto **11**

2.1. Objetivos 11

2.2. Alcance 12

2.2.1. Fase I - Análisis comparativo de las dos arquitecturas 12

2.2.2. Fase II - Desarrollo de un entorno de monitorización 13

2.3. Requerimientos 13

2.4. Obstáculos y riesgos 13

2.5. Metodología y rigor 14

2.5.1. Método de trabajo 14

2.5.2. Herramientas de seguimiento y método de validación 14

3. Planificación temporal **17**

3.1. Duración del proyecto 17

3.2. Fases del proyecto y descripción de tareas 17
3.2.1. Fase preliminar – Curso de GEP 17
3.2.2. Fase I – Análisis comparativo de arquitecturas 18
3.2.3. Fase II – Desarrollo de entorno de monitorización 18
3.2.4. Documentación (Memoria) del TFG 19
3.3. Desglose de trabajo 19
3.4. Necesidades de recursos 19
3.4.1. Recursos humanos 19
3.4.2. Recursos hardware 21
3.4.3. Recursos software 21
3.5. Estimación de trabajo y secuenciación de tareas 21
3.5.1. Clasificación de tareas por nivel de riesgo 22
3.5.2. Distribución del esfuerzo 22
3.5.3. Relaciones de dependencias entre tareas 22
3.6. Calendario 23
3.6.1. Desviaciones y plan de acción 23

4. Gestión económica (presupuesto) 27

4.1. Fuentes de costes 27
4.1.1. Costes laborales 27
4.1.2. Equipamiento 27
4.1.3. Gasto general 28
4.2. Estimación de costes 28
4.2.1. Costes directos 28
4.2.2. Costes indirectos 29

4.2.3. Presupuesto 31

4.3. Control de gestión 31

5. Sostenibilidad y compromiso social 33

5.1. Dimensión económica 33

5.2. Dimensión social 34

5.3. Dimensión ambiental 35

5.4. Matriz de sostenibilidad 35

6. Descripción de arquitecturas 39

6.1. Procesadores: Intel Xeon E5-2670 frente a Intel Xeon Platinum 8160 . . . 39

6.1.1. Evolución de las arquitecturas/microarquitecturas: Sandy Bridge frente a Skylake 40

6.1.2. Resumen de procesadores 42

6.1.3. Núcleo del procesador (processor core) 44

6.1.4. Jerarquía de caché 46

6.1.5. Lógica uncore (Agente del Sistema) 48

6.2. Nodos de cómputo 50

6.2.1. Resumen de nodos de cómputo 50

6.2.2. Configuración de la memoria 51

6.2.3. Topología del sistema a nivel de nodo 52

6.3. Redes de interconexión 55

6.4. Modelos Roofline 56

6.4.1. Modelo a nivel de procesador/memoria 57

6.4.2. Modelo a nivel de nodo completo 61

7. Diseño de benchmarking 63

7.1. Necesidades y componentes de evaluación 63
7.2. Descripción de benchmarks 64
7.2.1. Benchmarks sintéticos (microbenchmarks) 65
7.2.2. Aplicaciones científicas reales 66
7.3. Metodología general 67
7.4. Metodología general de benchmarking 68
7.4.1. Elección de versiones de software (compiladores y librerías) . . . 68
7.4.2. Número de ejecuciones y recolección de resultados 69

8. Análisis de microbenchmarking 71

8.1. Evaluación de rendimiento de la jerarquía de memoria 71
8.1.1. Latencia 72
8.1.2. Ancho de banda 81
8.2. Evaluación de rendimiento de las unidades de cálculo 85
8.2.1. Rendimiento en un core 85
8.2.2. Rendimiento en un nodo 87
8.2.3. Rendimiento en un "rack" 89
8.3. Evaluación de rendimiento de la red de interconexión e implementación MPI 90
8.3.1. Rendimiento en comunicaciones punto a punto internodo 90
8.3.2. Rendimiento en comunicaciones colectivas internodo 94

9. Análisis de aplicaciones y factores de conversión 97

9.1. Rendimiento a nivel de core (pruebas preliminares) 97
9.2. Escalabilidad de aplicaciones 98
9.3. Factores de conversión 99

10. Diseño y especificación del monitoring **103**

10.1. Necesidades reales para la puesta en marcha del monitoring actualmente 103

10.1.1. BSC User Portal 103

10.2. Especificación del monitoring 103

10.3. Selección de benchmarks y métricas para monitoring 104

10.3.1. Test A (diario) 104

10.3.2. Test B (semanal) 105

10.3.3. Test C (postmantenimiento) 105

Conclusiones y valoración final **107**

Índice de figuras

1.1. MareNostrum 4 4

1.2. Og3: monitorización en tiempo real de MareNostrum 4 9

3.1. Entregables o subtareas de la fase de implementación del monitoring . . 18

3.2. Estructura de desglose de trabajo del TFG 20

3.3. Calendario en forma de diagrama de Gantt 25

6.1. Roadmap de microarquitecturas Intel desde Sandy Bridge hasta Skylake . 40

6.2. Arquitecturas básicas de configuraciones de tipo 2S 42

6.3. Cuadro resumen de procesadores 43

6.4. Microarquitectura Sandy Bridge 44

6.5. Microarquitectura SkyLake 45

6.6. Frecuencias máximas de procesadores en modo Turbo 46

6.7. Rediseño de los niveles L2 y L3 de la jerarquía de caché de SkyLake-SP . 47

6.8. Subsistema uncore de la familia de procesadores Intel Xeon E5-2600 . . . 49

6.9. Subsistema uncore de la familia de procesadores Intel Skylake (hasta 28 cores) 49

6.10. Topología de memoria de MareNostrum 3 (niveles socket y nodo) 51

6.11. Topología de memoria de MareNostrum 4 (nivel socket) 52

6.12. Diagramas de bloques de nodos de cómputo 53

6.13. Topología del sistema de MareNostrum 3 54

6.14. Topología del sistema de MareNostrum 4 54

6.15. Red de interconexión de MareNostrum3 55

6.16. Red de interconexión de MareNostrum4 56

6.17. Modelo Roofline básico: interrelación de elementos que condicionan el rendimiento 57

6.18. Sistemas formados por procesador y memoria DRAM con sus valores máximos teóricos 60

6.19. Modelos Roofline para procesador y memoria DRAM + NUMA 61

6.20. Modelos Roofline para nodo completo 62

7.1. Pruebas de rendimiento de librerías Intel MPI 69

8.1. Latencia de memoria en acceso secuencial de un solo core 74

8.2. Estadísticas de lat_mem_rd en acceso secuencial recopiladas con perf stat 74

8.3. Observación del uso de memoria según la política de NUMA 77

8.4. Efectos de *stride* y *política de NUMA* en la latencia de memoria 78

8.5. Latencia de memoria en acceso paralelo de todos los cores de un socket . 79

8.6. Estadísticas de lat_mem_rd en acceso paralelo recopiladas con perf stat . 80

8.7. Comprobación de la afinidad de threads mediante KMP_AFFINITY (Intel) 82

8.8. Rendimiento en ancho de banda de un nodo 83

8.9. Rendimiento en ancho de banda de socket y nodo 84

8.10. Rendimiento en GFLOPS de un core 87

8.11. Estadísticas de LINPACK y HPCG al correr en un core (perf stat) 87

8.12. Rendimiento en GFLOPS de un nodo 89

8.13. Rendimiento en GFLOPS de 12 nodos 90

8.14. Patrones de comunicación de benchmarks de operaciones punto a punto 91

8.15. Rendimiento en comunicaciones punto a punto internodo 93

8.16. Patrones de comunicación de benchmarks de operaciones colectivas . . . 94

8.17. Rendimiento en comunicaciones punto a punto internodo 95

9.1. Rendimiento de aplicaciones en un core 98

9.2. Scaling de la frecuencia de un core al ejecutar aplicaciones 98

9.3. Rendimiento y escalabilidad de aplicaciones multinodo 99

Índice de tablas

3.1. Asignación de recursos hardware 21

3.2. Asignación de recursos hardware 21

3.3. Calendario en forma de tabla 24

4.1. Costes por mano de obra directa 28

4.2. Costes por mano de obra de gestión 29

4.3. Costes por amortización de equipamiento 30

4.4. Coste por horas de cómputo de máquinas HPC 30

4.5. Costes por gastos generales de oficina 31

4.6. Presupuesto 31

5.1. Cálculo de consumo máximo por rack de MN3 y MN4 34

5.2. Matriz de sostenibilidad 35

6.1. Resumen de características de memorias cachés 48

6.2. Cuadro resumen de nodos de cómputo 50

6.3. Cálculo de picos teóricos en GFLOPS por socket 59

7.1. Pila de software utilizado (compilador y librerías) 69

8.1. Conversión a ciclos de reloj de latencias de caché L1 y L2 en acceso secuencial 75

8.2. Conversión a ciclos de reloj de latencias de caché L1 y L2 en acceso paralelo 80

9.1. Configuración de factores de conversión 100

9.2. Factores de conversión de aplicaciones 101

10.1. Configuración de Test A 104

10.2. Configuración de Test A 105

10.3. Configuración de Test A 105

Gestión del proyecto (GEP)

Capítulo 1

Introducción y contextualización

Los superordenadores son utilizados hoy en día por prácticamente todas las disciplinas científicas: la astrofísica, la física de materiales o la biomedicina, pasando también por la ingeniería y por la industria. Son útiles para la investigación básica y aplicada, debido a su capacidad de realizar grandes cálculos, ejecutar grandes y complejas simulaciones, y analizar grandes volúmenes de datos [1].

Para conseguir una correspondencia óptima entre las grandes cargas de trabajo y las arquitecturas HPC (High Performance Computing) se requiere tanto optimizar las aplicaciones HPC para una arquitectura determinada como caracterizar el rendimiento de las aplicaciones transversalmente entre diferentes arquitecturas [2, p. 1].

1.1. Contexto

El superordenador MareNostrum 4, propiedad del Barcelona Supercomputing Center – Centro Nacional de Supercomputación (BSC–CNS), está íntegramente destinado a la generación de conocimiento científico. La parte más grande y potente de este superordenador, su clúster de propósito general, es uno de los más rápidos de Europa y actualmente ocupa el puesto número 25 en el ranking Top500 [3].

MareNostrum 4 dedica actualmente a la producción científica 11,1 Petaflops (11.100 billones de operaciones de coma flotante por segundo), lo que supone una potencia diez veces superior a la de su antecesor MareNostrum 3 —instalado entre 2012 y 2013—, estando a disposición de los científicos de toda Europa [1].

La RES (Red Española de Supercomputación) y PRACE (Partnership for Advanced Computing in Europe) dan acceso al 96 % de las horas de cálculo de la máquina —16 %

Figura 1.1: MareNostrum 4

y 80 % respectivamente— a través de procesos de selección gestionados por comités científicos. El 4 % restante queda a disposición de los investigadores del BSC–CNS [1]. Cabe destacar también que en el caso de los proyectos de la RES, las horas de cálculo se clasifican en 3 tipos —internamente hay 3 niveles de prioridad a la hora de correr las aplicaciones—.

Con este Trabajo Final de Grado (TFG) se pretende realizar, en una primera fase, un análisis comparativo de las diferentes arquitecturas que ha tenido MareNostrum desde 2012, es decir concretamente de las arquitecturas de los clústeres de MareNostrum3 (MN3) y MareNostrum4 (MN4), y en una segunda fase, crear un entorno para la monitorización de aplicaciones reales multinodo utilizando un sistema de extracción de métricas.

Este proyecto se realizará, con estrechísima relación, en el ámbito del Departamento de Operaciones del BCS–CNS —departamento que se encarga de la gestión, seguridad y administración del MareNostrum— y que es mi lugar de trabajo actual conjuntamente con el codirector del proyecto David Vicente —que es también el jefe de mi equipo—, donde desempeñamos las respectivas funciones de User Support Manager y Junior HPC User Support.

1.2. Motivación

La arquitectura de computadores va cambiando continuamente. Cada vez hay nuevas arquitecturas y es necesario saber cuánto más rápido van a ir las aplicaciones, saber qué mejora y qué empeora, en definitiva, ser capaces de caracterizar esas arquitecturas para darnos cuenta de cómo la evolución hardware está afectando al software.

La fase primera del proyecto está enfocada a este primer planteamiento, que consiste en analizar/comparar las diferencias más importantes entre ambas arquitecturas a distintos niveles. Esto es a nivel de cores, a nivel de nodos y a nivel de racks, teniendo en cuenta que en MareNostrum 4 tenemos más nodos por rack, más cores por nodo y una red de interconexión que es mejor que la anterior, sin olvidarnos de la parte relacionada con el consumo.

La idea es, por una parte, sacar factores de conversión de forma que podamos hacer conclusiones del tipo "MareNostrum 3 es 3 veces más lenta ejecutando según qué tipo de aplicaciones", ó "1 hora de MareNostrum 3 equivale a 0,25 horas de MareNostrum 4". ¿Por qué esto es importante? Principalmente porque hay muchos usuarios —la comunidad científica a través de RES y PRACE— que pueden adquirir horas y hay que poder poner "precio" a esas horas dependiendo de un factor de conversión.

Pongamos por caso un proyecto de la RES, que tenga asignadas un número determinado de horas en MN4 y, por alguna razón en un momento dado —por ejemplo que MN4 esté lleno— se necesita pasar esas horas a MN3 o a una máquina similar a MN3. Entonces hay que ser capaz de adaptar esas horas con un factor de conversión para que el proyecto pueda correr con normalidad en esa otra máquina. Para comprender mejor esta idea, sería algo así como poder decir al usuario: "en este momento no puedo darte 100000 horas en MN4, te daré 300000 en MN3 que son equivalentes".

Por otra parte, los resultados conseguidos con la primera fase deben establecer puntos de referencia y servir como base para desarrollar la segunda fase del proyecto, el *monitoring*. Se quiere desarrollar un entorno de monitorización del estado y comportamiento de la máquina diariamente, donde se pueda plasmar gráficamente aquellos datos, métricas u otro tipo de información para saber cómo afectan ciertos parámetros globales de la máquina a la ejecución de nuestras aplicaciones.

El objetivo es saber si la máquina está funcionando como se espera, si tiene algún problema y la causa de éste, o apreciar cómo va degenerando con el tiempo, aspecto este último que puede ser importantísmo por ejemplo para ser capaces de anticiparse y poner remedio a ciertas anomalías habituales antes de que éstas ocurran, todo esto desde una visión global, como un superordenador, no como nodos individuales. A día de hoy estas funcionalidades no existen o son limitadas, queriéndose dar un paso más allá.

1.3. Actores implicados

Una de las lecturas que podríamos hacer llegados a este punto es el amplio espectro que abarca este proyecto en lo que se refiere a la gran cantidad de personas involucradas, con perfiles muy diversos. Como es lógico, algunas de estas personas tendrán la máxima relevancia e influencia al estar directamente implicadas en los procesos de definición,

planificación, desarrollo y/o evaluación de resultados del proyecto. Pero también habrá otros participantes, algunos a priori más secundarios, que pueden contribuir de manera más o menos directa o bien ser potenciales beneficiarios, o ambas cosas a la vez tal como se explica a continuación.

Barcelona Supercomputing Center – Centro Nacional de Supercomputación

Claramente es a quien va dirigido este trabajo, quien hará uso directo de los resultados del estudio a realizar así como de la herramienta de monitorización, y será beneficiario de los logros conseguidos en forma de mejoras y optimización en el uso y gestión de los recursos —MareNostrum y otras máquinas HPC—.

Director

Es una de las figuras clave, con influencia determinante durante todo el proyecto. Sus responsabilidades van desde ayudar al estudiante —el desarrollador, yo en este caso— a elegir el tema del TFG, validar que el proyecto sea coherente con el desarrollo de las competencias técnicas y transversales asociadas al mismo y que los objetivos sean asumibles, hacer el seguimiento y mostrar al estudiante cuáles deben ser los resultados esperados en cada hito, así como ofrecerle la ayuda en la resolución de problemas y en la búsqueda de información que pueda necesitar.

En el contexto de este proyecto se da la situación de que el director (Eduard Ayguadé) ejerce una doble influencia, en primer lugar, como profesor de la FIB donde imparte y es responsable de asignaturas directamente ligadas a mi especialidad, y en segundo lugar, como investigador y Director Asociado del Departamento de Ciencias de la Computación del BSC–CNS. Desde ese punto de vista yo sería uno de los grandes beneficiarios, al poder contar con su vasta experiencia en ambas facetas y por todo el conocimiento que puede aportarme como colofón a mis estudios.

Codirector

Su papel también es crucial por unos cuantos motivos, el primero porque comparte junto con el director la mayoría de sus responsabilidades. Aparte, como ya se ha mencionado, (David Vicente) es el jefe del equipo que da soporte a los usuarios del MareNostrum y en el cual desempeño mis funciones actuales. Desde esa posición es quien ha concebido el planteamiento inicial del proyecto y me ha ofrecido la posibilidad de realizarlo poniendo a mi disposición todos los recursos y el soporte necesarios.

Su planteamiento responde a una doble motivación, la de cubrir una necesidad que contribuya al mejor funcionamiento de nuestro departamento y que puede reportar

beneficios a diferentes escalas dentro de la institución, pero a la vez que me sirva como un entrenamiento muy completo que me garantice una sólida formación, lo cual se traduciría en un mejor desempeño de mi trabajo y por consiguiente también en beneficios en los dos sentidos.

Desarrollador

En este proyecto soy yo el único desarrollador, asumiendo en ese rol las tareas siguientes:

1. Realizar el estudio previo para recopilar información sobre herramientas actuales disponibles —tanto software como hardware— y averiguar si existen proyectos similares o relacionados que sean aprovechables total o parcialmente

2. Realizar las pruebas de rendimiento pertinentes —ejecución de benchmarks y profiling— para la obtención de los diferentes parámetros y métricas deseadas siguiendo las pautas fijadas por el director y el codirector

3. Desarrollar los scripts o cualquier código necesario para la implementación del monitoring

4. Elaborar las conclusiones en base a los resultados obtenidos junto con toda la documentación requerida —memoria del proyecto, guía de usuario, etc.—

Al margen de otros beneficios ya comentados, este TFG representa el último paso para conseguir mi titulación y la oportunidad de poder hacerlo en el ámbito de un centro líder en su rama, con muchísimo renombre, reconocimiento y rigor científico, en una disciplina totalmente alineada con mi especialidad.

BSC HPC User Support Team

Los miembros del grupo de Soporte de Usuario —mis actuales compañeros de trabajo— pueden implicarse activa o pasivamente, ya sea concediéndome la ayuda o el apoyo que yo pueda necesitar a través de los diferentes hitos que se vayan alcanzando desde su experiencia contrastada en el uso del MareNostrum, como también convirtiéndose en potenciales usuarios/beneficiarios finales. Es decir, que las mejoras y beneficios que puedan derivar de este proyecto serían utilizadas y aprovechadas de forma transversal por todo el equipo.

Usuarios de MareNostrum y de otras máquinas HPC

Tener un mejor conocimiento de MareNostrum, de cómo se comporta ante distintas aplicaciones y cómo evoluciona su estado con el tiempo de funcionamiento, o ser capaces de establecer factores de conversión respecto a otras máquinas, contribuiría a optimizar la gestión y administración de los actuales recursos y a proporcionar un mejor soporte a los usuarios. O sea que de forma implícita serían también grandes beneficiarios. En sentido inverso, el feedback que necesariamente ofrecen en su día a día al correr sus aplicaciones en MareNostrum puede ser otra fuente de información útil para este propósito.

1.4. Estado del arte

A día de hoy no se tiene constancia de ningún análisis comparativo entre MareNostrum 3 y MareNostrum 4, mirando a ambas desde un punto de vista de clúster, como un superordenador —son máquinas únicas en su topología y en el caso de MareNostrum 4 de novedad muy reciente—. En cuanto al monitoring, las prestaciones actuales no satisfacen todas las necesidades tal como se desea, algunas de las cuales se quieren cubrir con este proyecto.

En este apartado se presenta de forma resumida un conjunto amplio de herramientas de las que disponemos actualmente y que, después de un primer estudio del escenario actual, creemos son un buen punto desde donde comenzar. Claro que a medida que el proyecto avance no se descarta que puedan incorporarse otras herramientas.

Herramientas propias del BSC–CNS

- **SPMA (Aplicación de Supervisión de Rendimiento de Soporte):** desarrollada durante 2015 por el grupo de Soporte de Usuario, consiste en la ejecución diaria de un conjunto de benchmarks en MareNostrum 3 para mostrar el comportamiento actual de la máquina —un subconjunto enfocado a aplicaciones HPC tales como WRF, GROMACS, CPMD, NAMD, AMBER y VASP, y otro a GPFS que da algunas métricas sobre el ancho de banda y el uso actual de los sistemas de ficheros— [4].

- **Og3:** herramienta usada actualmente para monitorizar MareNostrum 4 y otras máquinas HPC, que muestra información en tiempo real de métricas en su mayoría relacionadas con la carga de trabajo y el estado actual de los nodos.

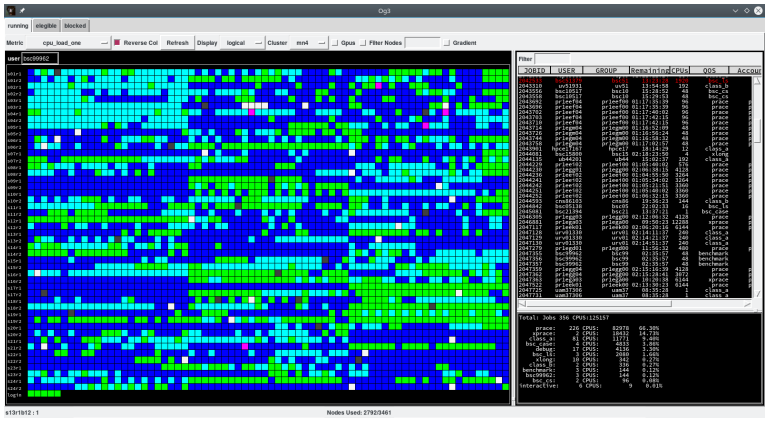


Figura 1.2: Og3: monitorización en tiempo real de MareNostrum 4

Otras herramientas de interés

- **Grafana:** plataforma de análisis que es propicia para consultar, visualizar, alertar y comprender métricas sin importar dónde estén almacenadas. Actualmente se usa con MareNostrum 4 y otras máquinas para mostrar métricas tales como carga de cpu, uso de memoria o consumo, a nivel de nodos individuales [5].
- **Kibana:** plugin de visualización de datos para Elasticsearch, de código abierto, con el que se pueden crear diagramas, gráficos o mapas sobre grandes volúmenes de datos, dando la libertad de seleccionar la forma en la cual visualizarlos [6].
- **Nagios:** poderoso sistema de monitoreo con el que es posible identificar y resolver problemas de infraestructura de TI antes de que afecten a procesos críticos, al ser capaz de monitorear aplicaciones, servicios, sistemas operativos, protocolos de red, métricas del sistema y componentes de infraestructura en una única herramienta [7].

Capítulo 2

Alcance del proyecto

2.1. Objetivos

Con el proyecto se aspira a conseguir dos principales objetivos de los que derivan otros objetivos específicos:

1. **Realizar un análisis comparativo entre las arquitecturas de MN3 y MN4**

    a) Analizar mediante el uso de herramientas —*accounting*, *profiling*, etc.— el comportamiento de las aplicaciones en cada arquitectura

    b) Sacar diferencias y mejoras entre las dos arquitecturas

    c) Identificar los parámetros que permitan a los usuarios la optimización de códigos en función de la arquitectura

    d) Obtener factores de conversión de rendimiento

2. **Desarrollar un entorno de monitorización de MN4 al ejecutar un conjunto de aplicaciones**

    a) Determinar un conjunto de benchmarks y métricas para el diseño de un benchmarking que permita la monitorización global del sistema

    b) Extraer métricas periódicamente que sirvan para monitorizar el estado global de la máquina y detectar patrones de comportamiento

    c) Analizar los datos obtenidos para la detección anticipada y resolución de incidencias que pueden afectar al sistema

2.2. Alcance

El proyecto por su propia naturaleza se divide en dos partes o fases bien diferenciadas, aunque con bastante interdependencia entre ellas.

2.2.1. Fase I - Análisis comparativo de las dos arquitecturas

A efectos de marketing suele interesar definir una velocidad máxima para una CPU. En la práctica, esta velocidad máxima es inalcanzable y muy pocos códigos consiguen acercarse. El benchmark LINPACK es una de las medidas que nos puede indicar lo cerca que se puede llegar a ella, aunque no todas las cargas de trabajo son como LINPACK [8, 9]. Entonces la meta es hacer múltiples ejecuciones de LINPACK junto con otros benchmarks sintéticos así como aplicaciones reales y comparar el rendimiento fijándonos en los diferentes niveles de granularidad deseados.

Core a core

A nivel de arquitectura del procesador el mayor interés recaerá en la vectorización —AVX vs. AVX512—, siendo un componente importante para lograr un alto rendimiento en los superordenadores actuales. Básicamente consiste en poder operar en paralelo sobre un conjunto de datos ejecutando una sola instrucción dentro de un único núcleo de CPU, mejorando notablemente el rendimiento [10].

Node a node

Los aspectos a analizar en este nivel serán principalmente los relacionados con el rendimiento de la jerarquía de memoria (cache) y la escalabilidad, basándonos en el hecho de que existe una diferencia importante en cuanto al número de cores por nodo de una máquina respecto a la otra.

Rack a rack

El rendimiento del sistema está cada vez más limitado por la comunicación en lugar de la computación. Por esta razón la red se ha convertido en un factor clave para determinar el rendimiento del sistema. Además las redes conectan tanto a nodos de procesamiento como a dispositivos de E/S distribuidos. De modo que en esta parte entra en juego todo lo visto anteriormente, añadiendo el componente de la red de interconexión: InfiniBand (IB) vs. Omni–Path (OPA).

2.2.2. Fase II - Desarrollo de un entorno de monitorización

En esta fase la finalidad es implementar un entorno de monitorización que ejecute diariamente un conjunto de entre 5-6 aplicaciones predefinidas —aplicaciones reales y benchmarks sintéticos— que nos permitan llevar un control del estado y comportamiento global de la máquina.

2.3. Requerimientos

Durante prácticamente toda la duración del proyecto será necesario disponer de:

- Hardware requerido (véase 3.4.2 Recursos hardware)
- Cuentas de usuario en MN3 y MN4
- Conexión de acceso remoto a MN3 y MN4 mediante el protocolo SSH
- Horas de cómputo suficientes en MN3 y MN4
- Software requerido instalado (véase 3.4.3 Recursos software)
- Horas de dedicación

2.4. Obstáculos y riesgos

Número limitado de racks de MN3 (Nord 3)

Como ya es sabido MareNostrum4 sustiyó hace pocos meses a MareNostrum3, de manera que MN3 como tal ya no existe en toda su magnitud, pero sí se conserva uno de sus racks que continúa en explotación bajo el nombre de Nord 3. Este hecho no debe suponer un gran obstáculo para el estudio que se ha previsto inicialmente, aunque sí es una limitación a tener en cuenta en caso de que surgiese la necesidad de realizar pruebas usando más de un rack a medida que avance el proyecto.

Caída de MN4, MN3 (Nord 3) u otros sistemas

En caso de ocurrir fallos técnicos de tal magnitud que provocasen la caída de alguna de las máquinas que son objeto de estudio o de los sistemas asociados —red de interconexión,

sistema eléctrico, internet, etc.—, podría suponer un serio percance o incluso convertirse en un obstáculo insalvable que implicaría la paralización temporal del proyecto.

Obtención de resultados no concluyentes

Es un riesgo quizá menor, pero podría ocurrir que una vez realizadas todas las pruebas previstas los resultados no fuesen del todo concluyentes para determinar por ejemplo algún factor de conversión. En este caso la solución podría ser investigar más en profundidad y realizar nuevas pruebas.

2.5. Metodología y rigor

2.5.1. Método de trabajo

Durante la primera fase se seguirá una metodología de desarrollo en cascada, de manera que las diferentes tareas se realicen gradualmente y de forma secuencial, lo cual no impedirá que en determinados casos pueda interesar el solapamiento de algunas tareas no dependientes entre sí, siempre que lo permita la distribución de recursos. En la segunda fase será más conveniente seguir una metodología de desarrollo iterativa e incremental.

Con vistas a obtener un mejor resultado también se pueden combinar las metodologías anteriores con la metodología Scrum, que consiste en realizar entregas parciales y regulares del proyecto final priorizadas por el beneficio que aportan. Puede interesar especialmente para resolver situaciones cuando las entregas no cumplen todos los requisitos o se alargan demasiado, los costes se disparan o la calidad no es aceptable. También sería indicada cuando fuese necesario identificar y solucionar ineficiencias sistemáticamente [11].

2.5.2. Herramientas de seguimiento y método de validación

Trello

Trello es un software de administración de proyectos con interfaz web, siendo en este caso la herramienta elegida para tal fin. Funciona como un tablero con tarjetas virtuales para organizar tareas, agregar listas, adjuntar archivos, etiquetar eventos, agregar comentarios y compartir tableros. Permite ver todo el proyecto con tan solo

echar un vistazo al tablero, y todo se actualiza en tiempo real. No hay nada que configurar y todo el mundo lo obtiene al instante [12].

GitLab

Gitlab es un servicio web de control de versiones y desarrollo de software colaborativo que también puede ser necesario en distintas etapas del proyecto que requieran la implementación de código [13].

Reuniones de seguimiento

Las herramientas antes comentadas son un complemento perfecto a las reuniones de seguimiento con el codirector —con una frecuencia semanal— y con el director, cuya frecuencia podría ser cada 2-3 semanas en función de la evolución de las tareas y de los resultados conseguidos. Estas reuniones deben ser claves para definir la estrategia y la hoja de ruta a seguir en cada una de las etapas, así como para evaluar resultados y sacar las conclusiones correspondientes.

Capítulo 3

Planificación temporal

3.1. Duración del proyecto

El presente Treball de Final de Grau (TFG) se enmarca en la normativa del Grau en Enginyeria Informàtica (GEI) de la Facultat d'Informàtica (FIB) y tiene una carga lectiva de 18 créditos ECTS, lo que supone una carga de trabajo por parte del estudiante de entre 25 y 30 horas por crédito.

El período previsto para su realización está comprendido entre el 19 de septiembre de 2017 —fecha de inicio de la asignatura GEP— y el 8 de enero de 2018 —fecha fijada una semana antes de la fecha límite de presentación de la memoria final—, otorgando un plazo máximo de 16(+1) semanas aproximadamente, es decir unos 4 meses.

3.2. Fases del proyecto y descripción de tareas

3.2.1. Fase preliminar – Curso de GEP

La fase preliminar o anteproyecto corresponde a la asignatura semipresencial Gestión de Proyectos (GEP) cuyos dos principales objetivos son, presentar al estudiante las metodologías y herramientas que le permitan desarrollar con éxito su TFG, y proporcionarle una amplia base de contenidos esenciales para el ejercicio de su actividad profesional.

Consta de 6 entregables en los que el estudiante ha de ser capaz de definir claramente el problema, estado del arte, objetivos y alcance del proyecto, realizar una planificación

inicial junto con un presupuesto e informe de sostenibilidad asociados, y documentar el problema que se quiere resolver desde un punto de vista técnico, en este caso dentro del contexto específico de la especialidad Ingeniería de Computadores.

3.2.2. Fase I – Análisis comparativo de arquitecturas

Dentro del grupo de tareas a realizar durante la primera fase está, en primer lugar, el estudio tanto de las dos arquitecturas (MN3 y MN4) como de un conjunto de benchmarks —sintéticos y reales— por determinar, seguido de la instrumentación y ejecución de los benchmarks que finalmente sean escogidos para la obtención de las distintas métricas deseadas. El resto consiste en analizar e interpretar los resultados de las ejecuciones y por último elaborar el informe de comparativa.

3.2.3. Fase II – Desarrollo de entorno de monitorización

Durante la segunda y última fase se han de llevar a cabo las tareas necesarias para la puesta en marcha del monitoring, comenzando por seleccionar el conjunto de benchmarks que se ejecutarán diariamente, definir cuáles métricas se quieren recolectar, y decidir entre las varias opciones disponibles cuál será el sistema gestor de base de datos (SGBD) más conveniente para almacenar dichas métricas así como la herramienta más idónea para su posterior visualización.

Sin duda la tarea de mayor complejidad en esta fase final será la implementación del entorno de monitorización, que se divide en 4 subtareas todas ellas muy bien definidas (véase Figura 3-1) y que representan los siguientes entregables: 1) implementación del lanzador de jobs diarios, 2) instalación y configuración de la BD, 3) implementación del recolector de métricas y 4) integración del visualizador.

Como último paso es obligado realizar un grupo de pruebas (testing) y comprobar el correcto funcionamiento de todo el sistema de monitorización.

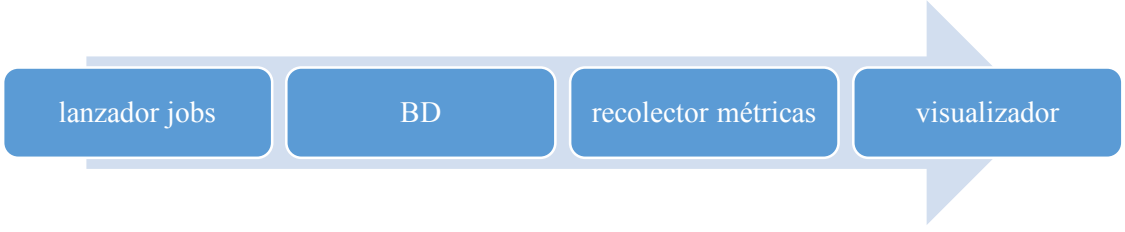


Figura 3.1: Entregables o subtareas de la fase de implementación del monitoring

3.2.4. Documentación (Memoria) del TFG

El penúltimo hito del TFG consiste en la presentación de una memoria en la que el estudiante ha de plasmar y documentar todo el trabajo realizado. La redacción de esta memoria se irá haciendo a lo largo de casi todo el proyecto y en paralelo a las diferentes fases del mismo, de forma que pueda ir realimentándose con cada uno de los procesos y entregables que se vayan completando.

3.3. Desglose de trabajo

La estructura de desglose de trabajo es una lista de tareas lógica, jerárquica y organizada que nos da una visión global de todo el trabajo que va a realizarse en el proyecto. Es una herramienta esencial para desarrollar un calendario realista, determinar los recursos necesarios y estimar los costes de forma fiable [14].

La Figura 3.2 representa en forma de esquema la estructura de desglose de trabajo del TFG. El nivel superior de esta estructura corresponde a los entregables principales o fases del proyecto, mientras que el nivel inferior es el paquete de trabajo o nivel de actividad, y se utilizará para desarrollar el calendario y más adelante para la estimación de los costes.

3.4. Necesidades de recursos

"Por recurso entendemos el trabajo de las personas o cosas necesarias para realizar alguna tarea" [15] como son las funciones de personas, herramientas o instalaciones.

3.4.1. Recursos humanos

- **Desarrollador:** este proyecto consta de una sola persona a título de desarrollador —estudiante de fase final de ingeniería informática—, de manera que a lo largo del mismo deberá asumir múltiples roles —analista, programador, etc.— con el fin de llevar a cabo todas las tareas planificadas.
- **Director y codirector:** serán dos personas clave durante todo el proyecto para el seguimiento y supervisión de las tareas.
- **Personal de Sistemas/Soporte:** puede ser necesaria su intervención en algunas tareas, ya sea solucionando problemas técnicos imprevisibles o bien dando su apoyo ante las posibles dudas o dificultades que puedan advenir.

1	**Fase preliminar – Curso de GEP**
2	Definición del alcance y contextualización
3	Planificación temporal
4	Gestión económica y sostenibilidad
5	Presentación preliminar
6	Módulo específico de Ingeniería de Computadores (EC)
7	Presentación oral y entrega final
8	**Fase I – Análisis comparativo de arquitecturas**
9	Estudio de las arquitecturas
10	Estudio y selección de benchmarks
11	Instrumentación y ejecución de benchmarks
12	Análisis e interpretación de resultados
13	Informe de comparativa
14	**Fase II – Desarrollo de entorno de monitorización**
15	Selección de benchmarks y definición de métricas
16	Estudio y selección de BD para almacenamiento de métricas
17	Estudio y selección de herramienta de visualización de métricas
18	**Implementación**
19	Lanzador de jobs diarios
20	Instalación y configuración de BD
21	Recolector de métricas
22	Visualizador
23	Testing final
24	Documentación (memoria del TFG)

Figura 3.2: Estructura de desglose de trabajo del TFG

3.4.2. Recursos hardware

Tabla 3.1: Asignación de recursos hardware

Recurso	Tareas
Ordenador portátil Apple MacBook Air	1, 9, 10, 13, 24
Ordenador portátil Dell Latitude E7440	11, 12, 14
Rack Superordenador MareNostrum3 (Nord3)	9–12
Rack Superordenador MareNostrum4	9–12, 18, 23
Máquina Virtual en Servidor (por determinar)	18, 23
Cámara de video Apple iPhone 7 Plus	5

3.4.3. Recursos software

Tabla 3.2: Asignación de recursos hardware

Recurso	Tareas
Software de productividad	
Microsoft Word 2016 para Mac	2–4, 6, 7, 13, 24
Microsoft Excel 2016 para Mac	3, 4
Microsoft PowerPoint 2016 para Mac	7
Gantter	3
GitLab	11, 18
Dropbox, GoogleDrive, Trello	1–24
Herramientas de *benchmarking* (a estudiar)	10, 11, 14
Herramientas de *profiling*	11, 12, 14
Herramientas de programación	11, 18
Herramienta de visualización (*Dashboard*) (a estudiar)	22, 23
Sistema gestor de base de datos (SGDB) (a estudiar)	20–23

3.5. Estimación de trabajo y secuenciación de tareas

Con el fin de precisar un calendario realista se emplearán simultáneamente las técnicas de estimación completa —estimación para los paquetes de trabajo—, estimación de distribución del esfuerzo —utiliza porcentajes para cada fase del proyecto—, y estimación heurística —juicio de expertos—, en combinación con el método de factores de riesgos,

este último enfocado básicamente en la complejidad y limitaciones de recursos de las tareas y procesos [14].

3.5.1. Clasificación de tareas por nivel de riesgo

Avanzando con la idea anterior, las estimaciones de trabajo deben hacerse teniendo en consideración los riesgos conocidos del proyecto [14]. Desde esa perspectiva se han identificado como riesgos potenciales: 1) la inexperiencia por parte del desarrollador con la tecnología, 2) una curva de aprendizaje más larga o ardua de lo esperado para las nuevas herramientas de desarrollo, y 3) un tiempo extra de entrenamiento y/o trabajo redundante como resultado de un mal uso inicial de metodologías de trabajo desconocidas [16].

En consecuencia se han clasificado las tareas en 3 niveles de riesgo —alto, medio y bajo—, definiéndose las tareas 11, 12 y 18 como las tareas con un riesgo mayor y que pueden recibir un mayor impacto (véase Tabla 3.3), si bien es cierto que estos factores se pueden atenuar al poder contar con el apoyo del personal de Soporte, con más experiencia.

3.5.2. Distribución del esfuerzo

En la Tabla 3.3 se cuantifica en horas la cantidad de esfuerzo estimado por cada tarea, reflejándose de forma evidente el impacto que pueden tener los niveles de riesgo antes detectados. De manera que las tareas 11 y 12 de la Fase I, y 18 de la Fase II, acaparan cerca del 60 % de todo el esfuerzo necesario en sus respectivas fases.

3.5.3. Relaciones de dependencias entre tareas

Llegados a este punto es importante tener en cuenta dos cosas, la primera, captar las relaciones lógicas existentes entre tareas para determinar en qué orden deben ejecutarse, y la segunda, pensar en cuáles tareas pueden ejecutarse simultáneamente haciendo una distribución de recursos realista —nivelación de recursos— [14]. La secuenciación de tareas resultante al aplicar estos dos criterios queda plasmada tanto en la Tabla 3.3 como en la Figura 3.3.

3.6. Calendario

Finalmente el calendario es la herramienta donde convergen casi todos los esfuerzos de planificación del proyecto [14]. Para la programación del calendario se han utilizado los métodos de "estructura de desglose de trabajo modificada" (Tabla 3.3) y "diagrama de Gantt" (Figura 3.3)

3.6.1. Desviaciones y plan de acción

El control del rendimiento de un proyecto pasa por completo por la prevención, detección y acción [14]. Este TFG —como todo proyecto— tiene elementos de incertidumbre que pueden generar riesgos, conocidos o no conocidos, siendo los primeros aquellos que han sido identificados y analizados, para los que se puede planificar una respuesta. De ahí la necesidad de llevar a cabo la planeación de la administración de estos riesgos, su identificación, análisis, respuesta, monitoreo y control.

En síntesis, el plan de acción se sustentará en los dos pilares siguientes:

- **Calendario recortado y flexible:** se han seguido técnicas y métodos de rigor para intentar hacer una planificación lo más realista posible. Además, el planteamiento inicial de ajustar el calendario dejando una semana de holgura hasta la fecha límite, el contemplar sólo días laborables, junto con la flexibilidad que permiten algunas dependencias entre tareas al estar determinadas únicamente por la nivelación de recursos, da un buen margen de maniobra en caso de ser necesaria una resecuenciación de tareas o, redistribuir o aumentar el esfuerzo.
- **Técnicas de control:** la utilización de paquetes de trabajo reducidos, las reuniones de seguimiento sistemáticas, la definición de puntos de control (hitos), y el establecimiento de criterios claros de finalización de los entregables, garantizan poder llevar a cabo medidas correctivas si se detectase cualquier variación.

Tabla 3.3: Calendario en forma de tabla

	Nombre	Duración	Esfuerzo	Riesgo	Inicio	Fin	Predecesoras
1	**⊟Fase preliminar – Curso de GEP**	**25d?**	**75h**		**19/09/2017**	**23/10/2017**	
2	Definición del alcance y contextualización	6d?	24.5h	Bajo	19/09/2017	26/09/2017	
3	Planificación temporal	4d?	8.25h	Bajo	27/09/2017	02/10/2017	2
4	Gestión económica y sostenibilidad	5d?	9.25h	Bajo	03/10/2017	09/10/2017	3
5	Presentación preliminar	5d?	6.25h	Bajo	10/10/2017	16/10/2017	4
6	Módulo específico de Ingeniería de Computadores (EC)	5d?	8.5h	Bajo	17/10/2017	23/10/2017	5
7	Presentación oral y entrega final	10d?	18.25h	Bajo	10/10/2017	23/10/2017	4
8	**⊟Fase I – Análisis comparativo de arquitecturas**	**24d?**	**169h**		**24/10/2017**	**24/11/2017**	
9	Estudio de las arquitecturas	4d?	28.5h	Bajo	24/10/2017	27/10/2017	6,7
10	Estudio y selección de benchmarks	3d?	21h	Bajo	30/10/2017	01/11/2017	9
11	Instrumentación y ejecución de benchmarks	6d?	42.5h	Medio	02/11/2017	09/11/2017	10
12	Análisis e interpretación de resultados	9d?	64h	Medio	10/11/2017	22/11/2017	11
13	Informe de comparativa	2d?	13h	Bajo	23/11/2017	24/11/2017	12
14	**⊟Fase II – Desarrollo de entorno de monitorización**	**29d?**	**206h**		**27/11/2017**	**04/01/2018**	
15	Selección de benchmarks y definición de métricas	3d?	21h	Bajo	27/11/2017	29/11/2017	13
16	Estudio y selección de BD para almacenamiento de métricas	4d?	28.5h	Bajo	30/11/2017	05/12/2017	15
17	Estudio y selección de herramienta de visualización de métricas	4d?	28.5h	Bajo	06/12/2017	11/12/2017	15
18	Implementación	17d?	121h	Alto	12/12/2017	03/01/2018	16,17
23	Testing final	1d?	7h	Bajo	04/01/2018	04/01/2018	18
24	**⊟Documentación (memoria del TFG)**	**74d?**	**90h**		**27/09/2017**	**08/01/2018**	
25	Documentación [Parte 1]	9d?	35h	Bajo	27/09/2017	09/10/2017	
26	Documentación [Parte 2]	55d?	55h	Bajo	24/10/2017	08/01/2018	6,7
27	Presentación de la memoria	0d	0h		15/01/2018	15/01/2018	
28	Lectura del TFG	0d	0h		22/01/2018	22/01/2018	

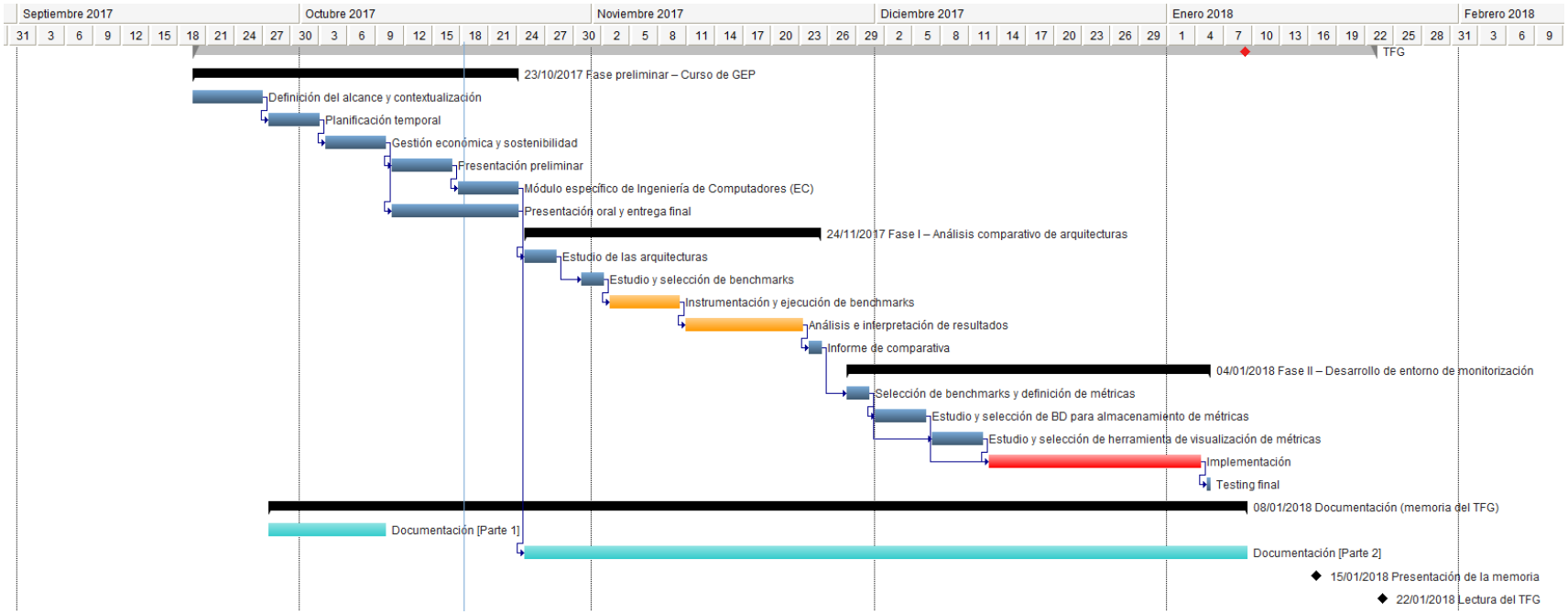


Figura 3.3: Calendario en forma de diagrama de Gantt

Capítulo 4

Gestión económica (presupuesto)

4.1. Fuentes de costes

Las bases sobre las cuales desarrollar un presupuesto sólido son la estructura de desglose de trabajo, las estimaciones del trabajo y el calendario del proyecto, siendo la identificación de los costes el primer paso para construir el presupuesto [14].

4.1.1. Costes laborales

Los costes laborales deben reflejar un elemento por cada persona o rol, basándose en las tasas de los recursos humanos asignados y en las estimaciones de trabajo [14]. Con este fin se han asignado los roles de "Director de proyecto" y "Jefe de proyecto" al director y codirector respectivamente, además del rol de "Desarrollador" que supone el principal coste en lo que a recursos humanos se refiere.

4.1.2. Equipamiento

Para cada herramienta hardware y software se debe estimar un coste en función de las horas reales de utilización respecto al período de amortización [17]. Los elementos a tener en cuenta en este apartado serán los dos ordenadores portátiles, la cámara de vídeo (smartphone) y únicamente el software de Microsoft —en principio el resto del software es gratuito—. Los supercomputadores al tener un tratamiento distinto y especial se incluirán en la partida de gasto general.

4.1.3. Gasto general

En esta partida se considerará en primer lugar el coste de las máquinas HPC y las instalaciones relacionadas aplicando una tarifa fija por horas de cómputo. Esta tarifa englobará todos los objetivos de coste asociados como son amortización, consumo de electricidad, sistema de refrigeración, mantenimiento, etc. También entrarán aquí los gastos generales de oficina, a lo que habría que añadir el coste del servidor requerido para el desarrollo de la segunda fase del proyecto, pero en un principio se optará por algún servicio gratuito como Amazon Web Services (AWS) [18].

4.2. Estimación de costes

4.2.1. Costes directos

Aunque no siempre es fácil distinguir entre "directo" e "indirecto", digamos que lo directo suele ser medible, mientras que lo indirecto en muchas ocasiones no lo es [19].

Mano de obra directa

Los costes directos del presente TFG se atribuyen totalmente a las horas de trabajo del desarrollador único del proyecto (véase Tabla 4.1). Para calcular el precio por hora real se ha tomado como referencia una jornada laboral efectiva de 1800 horas en cómputo anual, según se establece por convenio [20]. En el trabajo estimado están incluidas también las horas de esfuerzo dedicadas a las reuniones de seguimiento con el director y codirector —la planificación de estas reuniones se explica con detalle en el apartado siguiente—.

Tabla 4.1: Costes por mano de obra directa

Rol	Remuneración €	Horas reales	Precio/hora €/h	Horas estimadas	Coste estimado €
Analista programador	27K [25]	1,8K	15,00	376,5	5.513
Desarrollador	21K [25]	1,8K	11,70	163,5	1.913
Total				540	**7.426**

4.2.2. Costes indirectos

Entendemos por costes indirectos aquellos costes cuya asignación no se controla de forma individualizada a un objetivo de coste, sino que se requiere precisar criterios de reparto subjetivos [21].

Mano de obra de gestión

En este apartado se contempla el trabajo estimado por parte del director y codirector del proyecto para la realización de las reuniones de seguimiento y control (véase Tabla 4.2). En total se han planificado 5 reuniones con el Director de proyecto de 1 hora de duración y con frecuencia cada 2 semanas, una reunión con el Jefe de proyecto al comienzo de cada tarea durante las fases primera y segunda del proyecto, lo que supone 13 reuniones en este caso de 1/2 hora de duración, y una última reunión de supervisión con ambos líderes que se realizará al término de la segunda y última fase con duración de 1 hora.

Tabla 4.2: Costes por mano de obra de gestión

Rol	Remuneración €	Horas reales	Precio/hora €/h	Horas estimadas	Coste estimado €
Director de proyecto	65K [25]	1,8K	36,10	6	217
Jefe de proyecto	50K [25]	1,8K	27,80	7,5	208
Total					**425**

Amortizaciones

En la Tabla 4.3 se pormenoriza el coste de amortización de los recursos materiales antes identificados —excepto las máquinas HPC que se tratarán aparte—. Las horas estimadas de amortización concuerdan con la estimación de trabajo y la asignación de recursos de la parte previa de planificación.

Tabla 4.3: Costes por amortización de equipamiento

Concepto	Precio €	Horas reales	Precio/hora €/h	Horas estimadas	Coste estimado €
Amortización Apple MacBook Air 13”	1.622	5×1,8K	0,18	227,5	41
Amortización Dell Latitude E7440	1.600	4×1,8K	0,22	312,5	69
Amortización cámara de video iPhone 7 Plus	927	3×1,8K	0,17	6,25	1
Amortización licencia Microsoft Office 2016	149	3×1,8K	0,03	171,75	5
Total					**116**

Instalaciones

Dado que el proyecto implica la utilizacion de máquinas e instalaciones con un elevado coste económico tanto a nivel de inversión como energético es importante precisar estos costes lo más ajustado posible, no sólo de cara al desarrollo del presupuesto sino también con vistas a la repercusión que pueda tener en el informe de sostenibilidad. Como se ha comentado en el apartado de identificación de costes, una buena manera de conseguirlo es aplicar una tarifa de coste general o global a las horas de utilización de las máquinas HPC —llamadas horas de cómputo— tal como se muestra en la Tabla 4.4.

Los precios por hora de cómputo indicados son información contrastada de uso interno del BSC, mientras que la estimación de horas se hace en base a una previsión inicial del número de ejecuciones de código —benchmarks y aplicaciones—, número de cores por ejecución y duración estimada por ejecución medida en horas por core, que es la forma habitual y más conveniente para el cómputo de horas de trabajo en este tipo de máquinas.

Esa es una de las razones por las que las horas de cómputo no constan como estimación de esfuerzo en la planificación temporal. Otra razón es que las horas de cómputo en forma de ejecuciones de programas en MN3 y MN4 serán independientes a la jornada de trabajo del desarrollador —por ej. será factible y bastante probable programar/lanzar ejecuciones durante la noche o en paralelo mientras se realizan otras tareas—.

Tabla 4.4: Coste por horas de cómputo de máquinas HPC

Máquina HPC	Precio horas/*core* €	Horas/*core* estimadas	Coste estimado €
Rack MareNostrum3	0,02	15K	300
Rack MareNostrum4	0,01	10K	100
Total			**400**

Por último, se ha estimado un coste fijo mensual en concepto de gastos generales de oficina (véase Tabla 4.5), ya que una parte considerable de la cantidad de trabajo a realizar a lo largo de los 4 meses que durará el proyecto se hará en las instalaciones del BSC, reuniones de seguimiento incluidas.

Tabla 4.5: Costes por gastos generales de oficina

Concepto	Precio/mes €/mes	Meses	Coste estimado €
Agua, luz, teléfono, internet, etc.	40	4	160
Total			**160**

4.2.3. Presupuesto

Tabla 4.6: Presupuesto

Concepto	Importe	Observaciones
Mano de obra directa	7.426	
Total costes directos (CD)	**7.426**	
Mano de obra de gestión	425	
Amortizaciones	116	
Instalaciones	560	400 + 160
Total costes indirectos (CI)	**1101**	
Contingencia	**853**	0,1 × (CD + CI)
+35 horas analista programador (riesgo = 50%)	262	0,5 × 35 × 15,00
+45 horas desarrollador (riesgo = 30%)	158	0,3 × 45 × 11,70
+1 reunión de 1h con director (riesgo = 20%)	10	0,2 × (36,10 + 11,70)
+2 reuniones de 0,5h con codirector (riesgo = 30%)	12	0,3 × (27,80 + 11,70)
+10K horas de cómputo MN3 (riesgo = 5%)	10	0,05 × 10K × 0,02
+10K horas de cómputo MN4 (riesgo = 10%)	10	0,1 × 10K × 0,01
Total imprevistos	**462**	
TOTAL	**9.842**	

4.3. Control de gestión

El presupuesto nos puede servir para validar el calendario y viceversa. [...] Hacer una evaluación del rendimiento del proyecto basada en uno solo de estos indicadores puede dar a veces una visión irreal además de dificultar la detección de variaciones en fases tempranas. [...] De ahí que la mejor técnica de control para anticiparse a las variaciones de rendimiento del proyecto sea la gestión del valor acumulado o análisis de variación, ya que su mayor utilidad es la de medir y hacer el seguimiento en conjunto del rendimiento de los costes y del calendario [14].

Se trata de ir comparando sistemáticamente, por ej. al término de cada tarea del calendario, el valor del trabajo que realmente se hecho con el coste real invertido en ese momento, que no necesariamente han de coincidir y en tal caso indica una variación en el plazo de finalización previsto para esa tarea. [...] Cuando se detecta una variación significativa hay que modificar las estimaciones iniciales adaptando las tareas pendientes a la nueva situación que es ya irreversible [22].

La herramienta que se utilizará periódicamente para poder comprobar la situación real respecto al avance del proyecto será el informe de progreso, que incluirá información imprescindible de cada tarea como las fechas reales de comienzo y fin, el trabajo real ejecutado o coste consumido respecto al original previsto, y el trabajo o coste restante [22].

Capítulo 5

Sostenibilidad y compromiso social

El gran objetivo del desarrollo sostenible consiste en "satisfacer las necesidades del presente sin poner en peligro la capacidad de las generaciones futuras para satisfacer sus propias necesidades" [23].

5.1. Dimensión económica

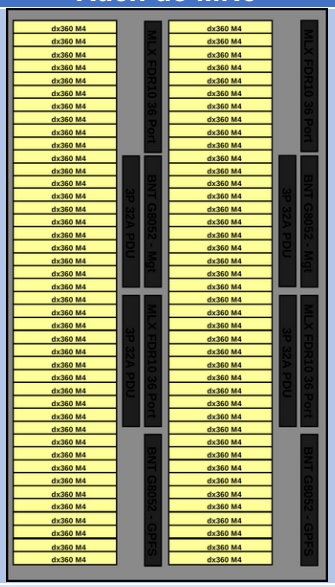
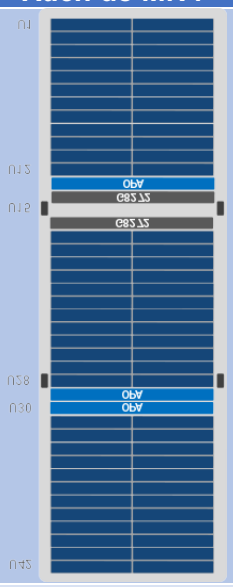
Económicamente hablando el proyecto es totalmente viable al contar con un presupuesto bastante ajustado dadas las características del mismo, en el que se ha tenido cuidado de no pasar por alto ningún objetivo de coste significativo. Los costes de mano de obra directa son acordes al mercado laboral actual, mientras que los costes por mano de obra de gestión se han ajustado reduciendo el número de reuniones de seguimiento en la medida de lo posible.

El resto de costes es también muy reducido, máxime teniendo en cuenta que se utilizan recursos con gastos globales muy elevados. Se puede comprobar en la Tabla 5.1 cómo a partir del consumo máximo por rack se ha hecho una aproximación del consumo por core, dando como resultado unos valores prácticamente idénticos a los precios por hora de la Tabla 4.4.

Si multiplicamos los valores obtenidos por el total de horas de cómputo —horas por core— planificadas para cada máquina, y por el precio de referencia de 0,12973 kWh [24], obtenemos unos costes aproximados de 39 € para MN3 y 13 € para MN4, en concepto de consumo de electricidad, es decir unos costes realmente bajos, casi despreciables.

En la matriz de sostenibilidad (véase Tabla 5.2) se penaliza moderadamente la posibilidad de riesgo ante una eventual necesidad de aumento del número de horas de dedicación

Tabla 5.1: Cálculo de consumo máximo por rack de MN3 y MN4

	Rack de MN3	*Rack* de MN4
		
Peak power consumption	28,04 kWh [29]	33,7 kWh [30]
Nodes per rack	84	72
Cores per node	16	48
Cores per rack	1.344	3.456
≈ kWh per core	≈ 0,02	≈ 0,01

por cuenta del analista programador y/o del desarrollador.

5.2. Dimensión social

Este proyecto responde a una doble necesidad del departamento de Operaciones del BSC, primero para resolver un problema que no está cubierto actualmente y que puede repercutir en un mejor servicio a todos los usuarios del BSC y resto de la comunidad científica, pero también supone un medio valioso de formación para mí, lo que implica elevar mis habilidades en el ámbito de la computación de altas prestaciones y como ingeniero informático en general, permitiéndome mejorar muchísimo en el desempeño de mis tareas profesionales.

Por otra parte se puede considerar que, grosso modo, se beneficiarán todos los actores implicados como consecuencia de las posibles mejoras y logros conseguidos.

5.3. Dimensión ambiental

Los sistemas tecnológicos causan a menudo un alto impacto ecológico, al igual que un gran impacto social que en ocasiones es consecuencia del primero [22]. Si se mira a MN3 y MN4 sólo desde el punto de vista del consumo energético es evidente que representan un coste ambiental altísimo. Sin embargo, el poder juntar tanta potencia de cálculo en un único espacio físico usando una única red de interconexión, sistema de refrigeración, etc., sí que supone una gran ventaja tanto a nivel ambiental como económico.

Con este proyecto se puede contribuir a hacer un poco más sostenible el uso de una máquina de incalculable valor para la investigación científica en muchas áreas, una de ellas precisamente las ciencias de la tierra donde se hacen predicciones y se desarrollan modelos climáticos, por ejemplo, todo ello causando un impacto ambiental mínimo al usarse la máquina de forma concurrente con su explotación diaria y en una proporción apenas perceptible.

5.4. Matriz de sostenibilidad

Tabla 5.2: Matriz de sostenibilidad

	PPP	**Vida útil**	**Riesgos**
Ambiental	Consumo del diseño	Huella ecológica	Riesgos ambientales
	9	19	-1
Económico	Factura	Plan de viabilidad	Riesgos económicos
	9	19	-4
Social	Impacto personal	Impacto social	Riesgos sociales
	10	20	0
Rango	28	58	-5
Sostenibilidad	**81**		

Análisis comparativo de arquitecturas y monitoring

Capítulo 6

Descripción de arquitecturas

En este capítulo se describen las dos arquitecturas de referencia que se han utilizado en este proyecto para realizar el análisis comparativo de rendimiento, poniendo el foco en las características y diferencias más relevantes. Se incluye para ello la descripción del procesador que forma un nodo del clúster, los componentes de dicho nodo y su interconexión para formar el clúster completo. Los dos sistemas analizados son:

- **MareNostrum 3**: supercomputador instalado en el año 2012 compuesto de 48896 cores Intel Sandy Bridge-EP (8 cores por procesador) en 3056 nodos de cómputo iDataPlex de IBM, ofreciendo una capacidad de cómputo de 1,1 Petaflops, red de interconexión Infiniband y sistema operativo Linux [25]. Actualmente, parte de Marenostrum 3 está todavía en producción con el nombre Nord 3, con 66 nodos distribuidos en 2 racks.
- **MareNostrum 4**: supercomputador instalado en el año 2017 compuesto de 165888 cores Intel Skylake-SP (24 cores por procesador) en 3456 nodos de cómputo SD530 de Lenovo, ofreciendo una capacidad de cómputo de 11,1 Petaflops. Red de interconexión Intel Omni-Path y sistema operativo Linux [26].

6.1. Procesadores: Intel Xeon E5-2670 frente a Intel Xeon Platinum 8160

Comencemos por contextualizar ambos procesadores dentro del roadmap "Tick-Tock" de microarquitecturas de Intel y encarar así un primer análisis comparativo a nivel de chips. Tick-Tock [27–29] no es más que una estrategia seguida por Intel para presentar sus nuevos procesadores, presentando primero una nueva arquitectura (Tock) y luego

un proceso de fabricación más avanzado que se basaba en la arquitectura existente (Tick). Normalmente, los procesadores basados en nueva arquitectura (Tock) son los que muestran un gran aumento del rendimiento en general.

En este contexto, se puede comprobar (véase Figura 6.1) que tanto el procesador Intel Xeon E5-2670 de MareNostrum 3, basado en la microarquitectura Sandy Bridge, como el procesador Intel Xeon Platinum 8160 de MareNostrum 4, basado en la microarquitectura Skylake, son procesadores "Tock" y entre ellos ha habido cuatro saltos generacionales.

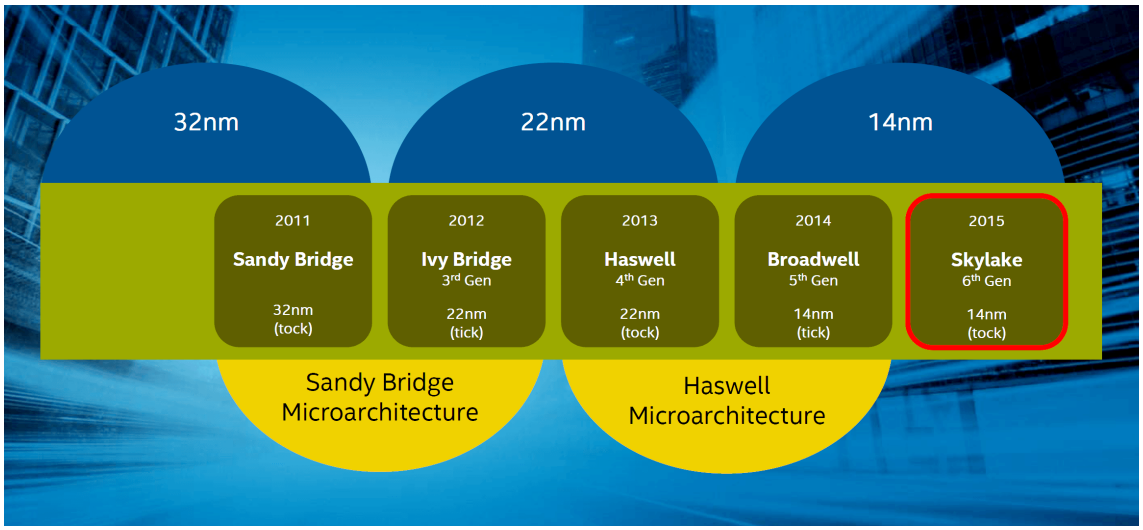


Figura 6.1: Roadmap de microarquitecturas Intel desde Sandy Bridge hasta Skylake

6.1.1. Evolución de las arquitecturas/microarquitecturas: Sandy Bridge frente a Skylake

Entre las nuevas prestaciones y extensiones que aporta la arquitectura Sandy Bridge [30, 31] con respecto a sus antecesoras destacan:

a) Conexión on-chip en "anillo" de núcleos con caché L3, agente del sistema (System Agent), controlador de memoria, agente QPI (QPI Agent) y controladores de E/S para incrementar la escalabilidad.

b) Aumento de la memoria caché L3 por núcleo, de 2 MB a 2.5 MB.

c) Cuatro canales de memoria DDR3 (en lugar de tres). La velocidad de la memoria aumenta de 666 MHz (Westmere) a 800 MHz.

d) Nueva unidad Intel Advanced Vector eXtensions (AVX) con registros vectoriales de 256 bits (en lugar de 128 bits), duplicando así el rendimiento en coma flotante.

e) Nueva versión 2.0 de Turbo Boost con un aumento de frecuencia del procesador de hasta 600 MHz (en lugar de hasta 400 MHz).

f) Integración de controladores PCI-Express 3.0 en el subsistema de E/S del chip. Las líneas PCIe aumentan de 36 a 40.

g) Dos enlaces QPI (en lugar de un enlace anteriormente) para la comunicación entre procesadores formando un sistema NUMA. La velocidad del enlace QPI aumenta de 6.4 a 8 GT/s.

h) Dos "loads" más un "store" por ciclo (anteriormente un "load" más un "store"), doblando así el ancho de banda de 'load'.

Según las fuentes citadas, la mejora potencial del rendimiento de la arquitectura Sandy Bridge (con respecto a los procesadores Nehalem y Westmere) se atribuye principalmente a las prestaciones $a)$ y $c)$, con todo y que, de entre todas las nuevas características arquitectónicas, la principal es sin duda $d)$, Advanced Vector eXtensions (AVX), que permite realizar nuevas operaciones SIMD de 256 bits.

Por su parte, entre los cambios de mayor relevancia en la arquitectura de la familia de procesadores escalables Xeon Skylake-SP [32–34] se incluyen:

a) Núcleos de procesador dispuestos en una interconexión de "malla" (reemplazando la interconexión de anillo doble más antigua).

b) Hasta 28 núcleos de procesador por socket.

c) Nuevas instrucciones AVX-512 de doble rendimiento (hasta 16 FLOPS de doble precisión por ciclo y unidad FMA[1] AVX-512). Hasta dos unidades FMA AVX-512 por núcleo de procesador.

d) Controlador de memoria de 6 canales en cada procesador (en lugar de 4 canales en plataformas anteriores). Velocidad de memoria DDR4 de hasta 1333 MHz.

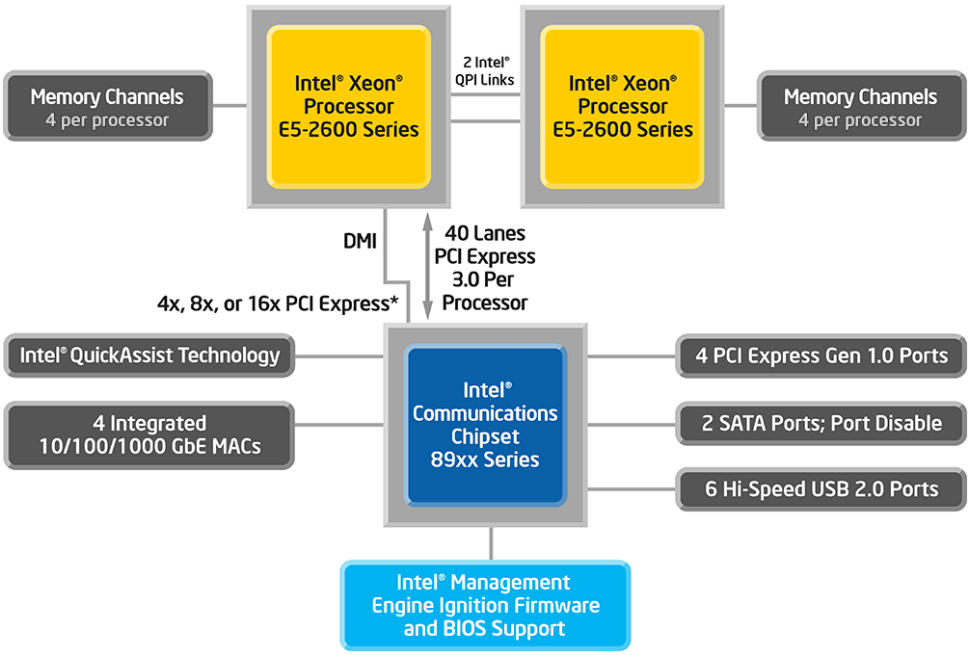
e) Enlaces más rápidos entre sockets de procesador con hasta tres enlaces UPI de 10.4GT/s (reemplazando la anterior interconexión QPI).

f) Mayor conectividad de E/S con 48 carriles (lanes) de PCI-Express 3.0 por CPU (en comparación con 40 carriles anteriormente).

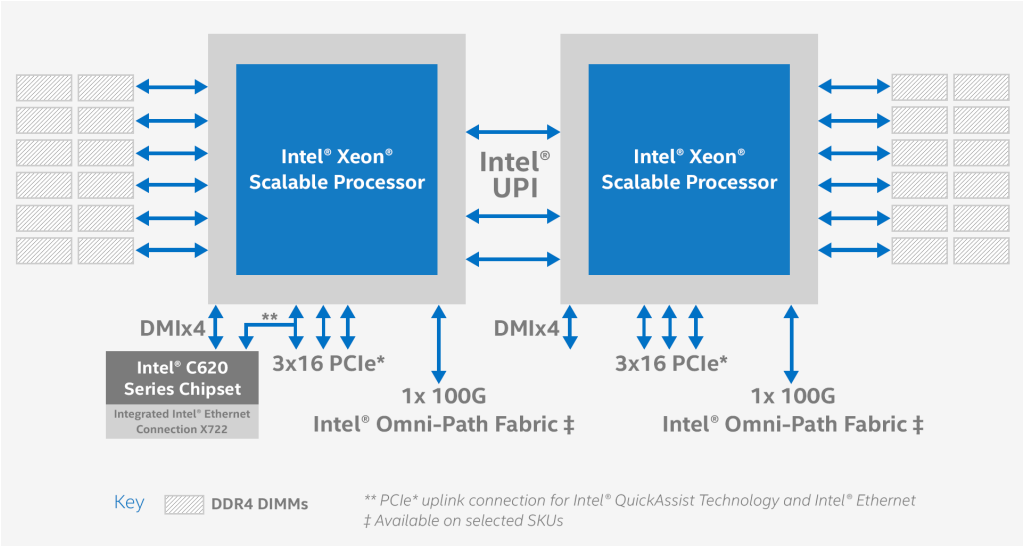
g) Fabric Omni-Path opcional de 100 Gbps integrada en el procesador (disponible sólo en ciertos SKU) que permite la interconexión de nodos para formar un clúster.

h) Turbo Boost optimizado que permite frecuencias más altas incluso cuando muchos núcleos de procesador están en uso.

[1] Multiplicación y suma encadenada



(a) Sistema NUMA formado por 2 procesadores Sandy Bridge



(b) Sistema NUMA formado por 2 procesadores SkyLake

Figura 6.2: Arquitecturas básicas de configuraciones de tipo 2S

6.1.2. Resumen de procesadores

En la Figura 6.3 se expone un cuadro resumen con las características más importantes a nivel de arquitectura de los dos procesadores en cuestión. En la parte superior, el gráfico "Número de núcleos/hilos" muestra la cantidad de cores o núcleos físicos (área

más oscura) junto con la cantidad de threads o hilos de ejecución adicionales que ofrece la tecnología Hyper-Threading[2] (área más clara). En la tabla de al lado, "Frecuencia de operación", se representa la frecuencia base o nominal mediante el color oscuro, mientras que el área más clara corresponde a la frecuencia máxima adicional proporcionada por la función Turbo (este punto se trata en detalle al final de la sección, en el apartado 6.1.3).

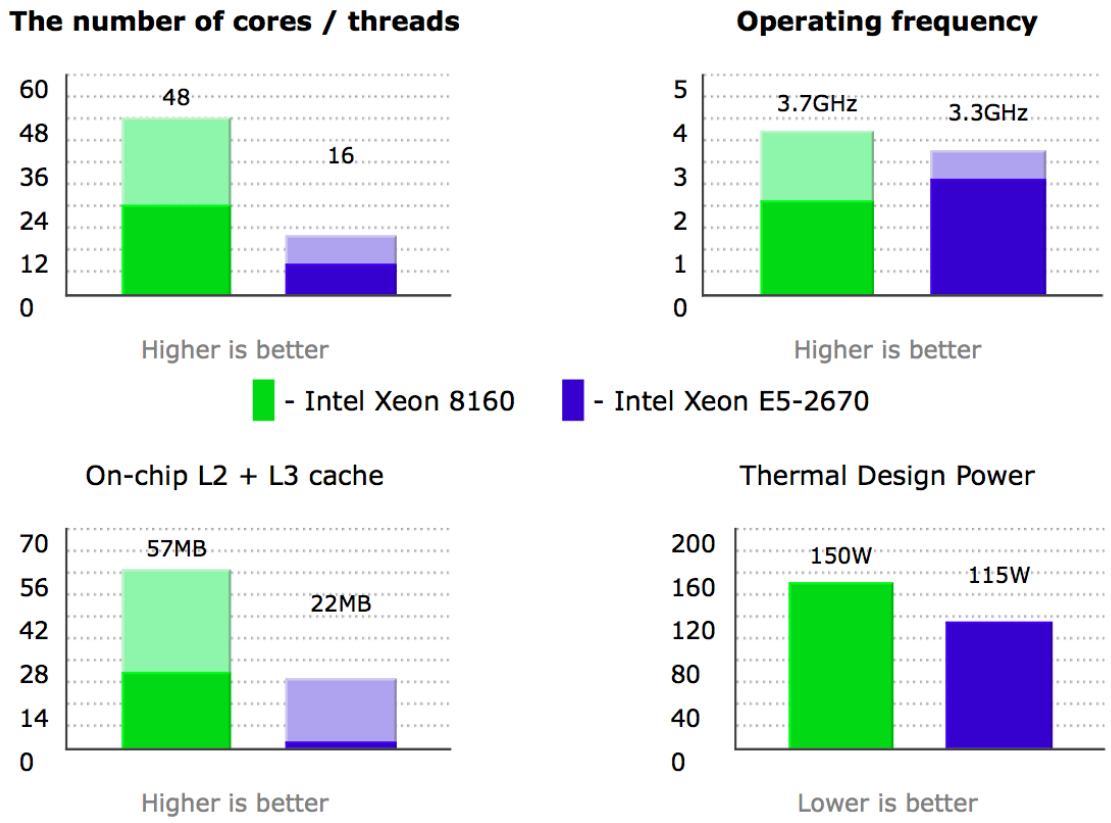


Figura 6.3: Cuadro resumen de procesadores

Continuando con la parte inferior de la Figura 6.3, el gráfico "Caché en el socket" muestra los tamaños de las caches L2 y L3, representados por las áreas más oscura y más clara, respectivamente. Por último, se incluye un gráfico donde simplemente se comparan los valores relativos al TDP (Thermal Design Power).

[2]Esta prestación que ofrece la tecnología del procesador resulta irrelevante en este trabajo al no estar activada esa funcionalidad en ninguno de los dos sistemas.

6.1.3. Núcleo del procesador (processor core)

Unidades SIMD

Una de las innovaciones más llamativas en la evolución de la microarquitectura Sandy Bridge es a la que [35] se refiere como una manera "cool" de poder ejecutar una suma y una multiplicación por ciclo, sin unidades FMA –remarquemos que aún no habían sido incorporadas a la unidad de cálculo del procesador–. Esta importante innovación no es más que reutilizar los caminos de datos existentes para un uso dual, de manera que Intel AVX utiliza dos pilas de ejecución de 128 bits a través de los puertos 0, 1 y 2 (véase Figura 6.4) para realizar una multiplicación de 128+128 bits y una suma de igual anchura de datos.

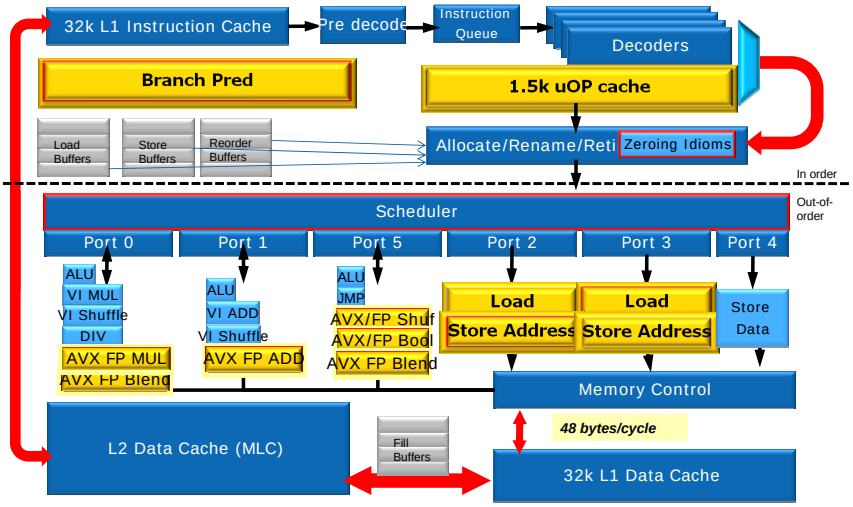


Figura 6.4: Microarquitectura Sandy Bridge

Por lo que respecta a Skylake, cada núcleo del procesador Xeon 8160 incorpora una unidad Fused Multiply Add (FMA) extra de 256 bits, para uso exclusivo en operaciones de cálculo vectoriales AVX-512.

AVX y AVX2 habían sido los conjuntos de instrucciones de Intel más avanzados hasta la llegada de Intel Advanced Vector eXtensions 512 (o simplemente AVX-512) [36,37], cuya novedad más destacada es el ancho de registro vectorial de 512 bits, esto es el doble de tamaño de los anteriores registros AVX/AVX2 de 256 bits, duplicando también el número de registros y el ancho de las unidades FMA con respecto a sus antecesores, todo ello para disminuir aún más la latencia. Esto es gracias a una técnica similar a la usada por la "pionera" Sandy Bridge", concretamente usando dos FMAs de 128 bits en paralelo (vía Port 0 y 1) junto con la FMA "extra"de 256 bits (Port 5). Esta detalle puede verse con claridad en la Figura 6.5 si nos fijamos en el recuadro dibujado con líneas discontinuas azules.

Intel AVX-512 ofrece pues hasta 32 operaciones de coma flotante de doble precisión y 64 de simple precisión por ciclo de reloj en vectores de 512 bits, así como ocho enteros de 64 bits y dieciséis de 32 bits, con hasta 2 unidades combinadas de multiplicación y suma de 512 bits (FMA). Además, AVX-512 añade otras capacidades inexistentes en AVX/AVX2, al incluir optimizaciones adicionales para acelerar todavía más las tareas para las cargas de trabajo más actuales.

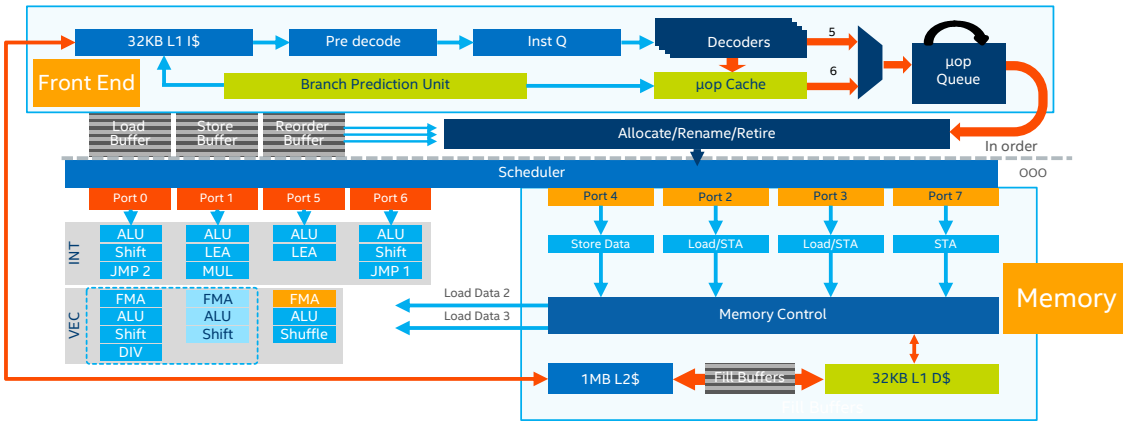


Figura 6.5: Microarquitectura SkyLake

Frecuencias de reloj

En el procesador Xeon E5-2670 (Sandy Bridge), el escalado de frecuencia varía en función del número de cores utilizados. En cambio, en el procesador Xeon 8160 (Skylake-SP) el escalado de frecuencia no sólo depende del número de cores sino también del conjunto de instrucciones utilizados, por lo que tenemos que diferenciar hasta 5 tipos de frecuencias de reloj:

a) Base Clock

b) Max Turbo Frequency

c) Non-AVX All-Core-Turbo

d) AVX2 All-Core-Turbo

e) AVX-512 All-Core-Turbo

En la Figura 6.6 se comparan de forma gráfica los rangos de escalado de frecuencia en modo Turbo, siendo un tema de máxima relevancia para la realización de este trabajo.

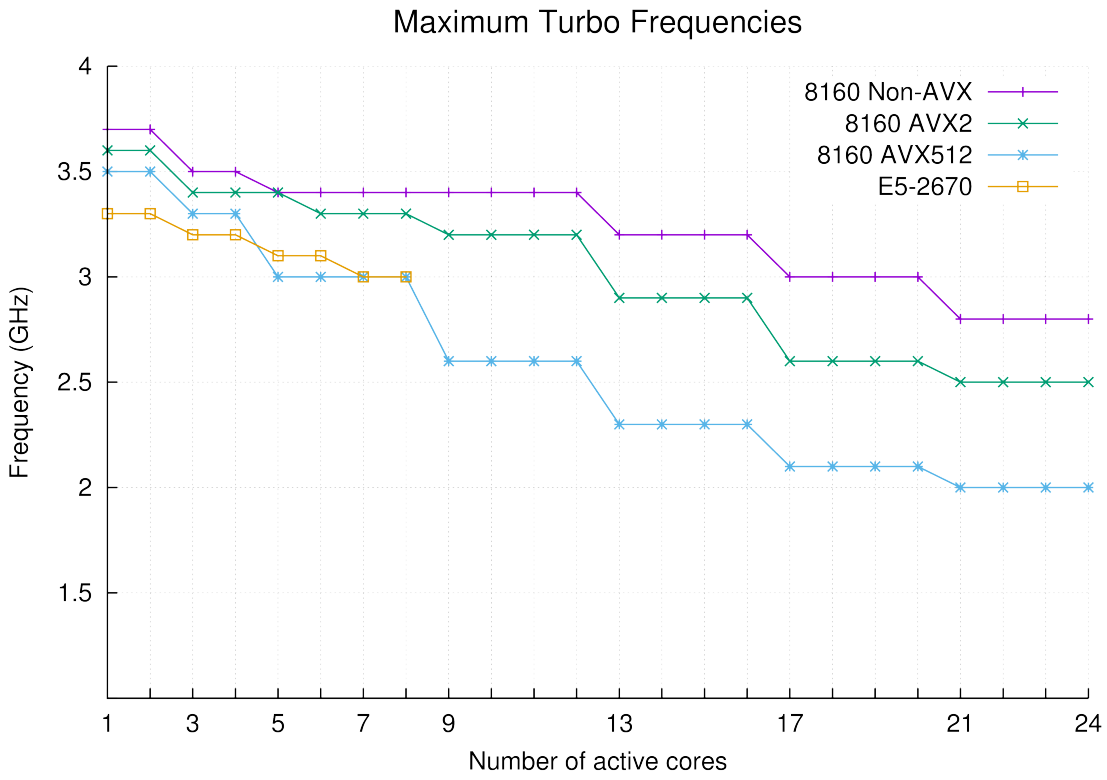


Figura 6.6: Frecuencias máximas de procesadores en modo Turbo

6.1.4. Jerarquía de caché

El procesador Intel Xeon E5-2670 basado en la microarquitectura Sandy Bridge tiene 3 niveles de memoria caché, con cachés L1 y L2 dedicadas por cada núcleo. La caché L1D por núcleo es una caché asociativa de 8 vías con un tamaño de 32 KB. Utiliza una política de escritura Write-Back y tiene una latencia constante de 4 ciclos para las uops de enteros (para FP y SIMD es de 1 ó 2 ciclos más). Al tener múltiples bancos de 8 bytes (caché multibanco) permite accesos simultáneos, siendo capaz de sostener dos loads de 128 bits y un store de 128 bits en cada ciclo, aumentando así en un 50 % el ancho de banda con respecto a su antecesor.

La caché L2 unificada por núcleo es una caché asociativa de 8 vías con un tamaño de 256 KB. La latencia es de 12 ciclos y tiene una relación no exclusiva/no inclusiva con las cachés L1. La política de escritura es Write-Back.

La caché L3 es compartida por los núcleos (además de por la GPU integrada y otros controladores de medios) y se accede a ella a través del anillo. Cada núcleo tiene una porción de la caché L3 y cada porción puede proporcionar la mitad de una línea de caché (32B) al anillo de datos por ciclo. La caché L3 es **inclusiva** con una política de escritura

Write-Back, siendo escalable y específica para la implementación de cada procesador. En el procesador E5-2670, cada porción de 2,5 MB es una caché asociativa de 16 vías con una latencia de 26-31 ciclos, dependiendo del número de saltos en el anillo.

En el procesador Intel Xeon 8160 basado en la microarquitectura SkyLake, la gran diferencia con respecto al Xeon E5-2670 radica en las cachés L2 y L3, que están organizadas de manera diferente como se ilustra en la Figura 6.7. Uno de los cambios notables es que se ha cambiado a una estructura de caché L3 no inclusiva, lo que quiere decir que las líneas de caché L2 pueden no estar en la caché L3 compartida.

Otra característica evolutiva destacable es que la caché L3 es bastante más pequeña, con 1.375 MB por núcleo, pero que se compensa con una caché L2 más grande. En general, se dice que esto hace que las cachés puedan alimentar mejor a los núcleos, cuestión particularmente importante para las cargas de trabajo multihilo. Como inconveniente, la latencia de la caché L2 se ha incrementado en dos ciclos, aunque este aumento de la latencia debería quedar enmascarado por el tamaño de caché L2 más grande.

En la Tabla 6.1 están resumidas las características de las cachés de ambos procesadores.

Figura 6.7: Rediseño de los niveles L2 y L3 de la jerarquía de caché de SkyLake-SP

Tabla 6.1: Resumen de características de memorias cachés

	Xeon E5-2670	Xeon 8160
L1 instruction cache		
size	32 K	32 K
assoc	8-way	8-way
L1 data cache		
size	32 K	32 K
assoc	8-way	8-way
latency	4 cycle	4 cycle
L2 unified cache		
size	256 K	1024 K
assoc	8-way	16-way
latency	12 cycle	14 cycle
L3 unified cache		
size	20 M, shared	33 M, shared
	2,5 M/core	1,375 M/core
	(inclusive)	(non-inclusive)
assoc	20-way	11-way
latency	26-31 cycle	50-70 cycle
network	Ring	Mesh

6.1.5. Lógica uncore (Agente del Sistema)

En cuanto a la lógica denominada *uncore*, las características más importantes a resaltar de ambos sistemas son las siguientes:

- **MareNostrum 3**: 20 MB de cache de último nivel (L3 LLC), 2 puertos QPI 8 GT/s para la interconexión de los dos procesadores del nodo y 40 líneas PCIe 3.0. La lógica de uncore también incluye 1 controlador de memoria con 4 canales a memoria.
- **MareNostrum 4**: 33 MB de LLC, 2 puertos UPI 10.4 GT/s para la interconexión de los dos procesadores del nodo y 48 líneas PCIe 3.0. La lógica de uncore también incluye 2 controladores de memoria, cada controlador de memoria con 3 canales a memoria para un total de 6 canales a memoria.

Las Figuras 6.8 y 6.9 detallan la arquitectura de ambos procesadores, incluyendo los cores y su interconexión, en anillo para Sandy Bridge y en malla 2D para Skylake, así como la lógica uncore comentada en los párrafos anteriores.

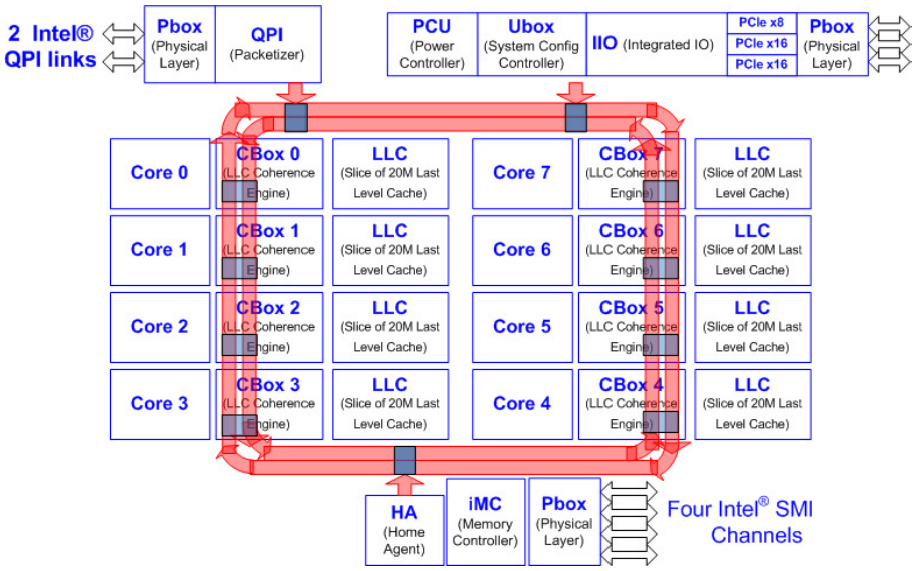


Figura 6.8: Subsistema uncore de la familia de procesadores Intel Xeon E5-2600

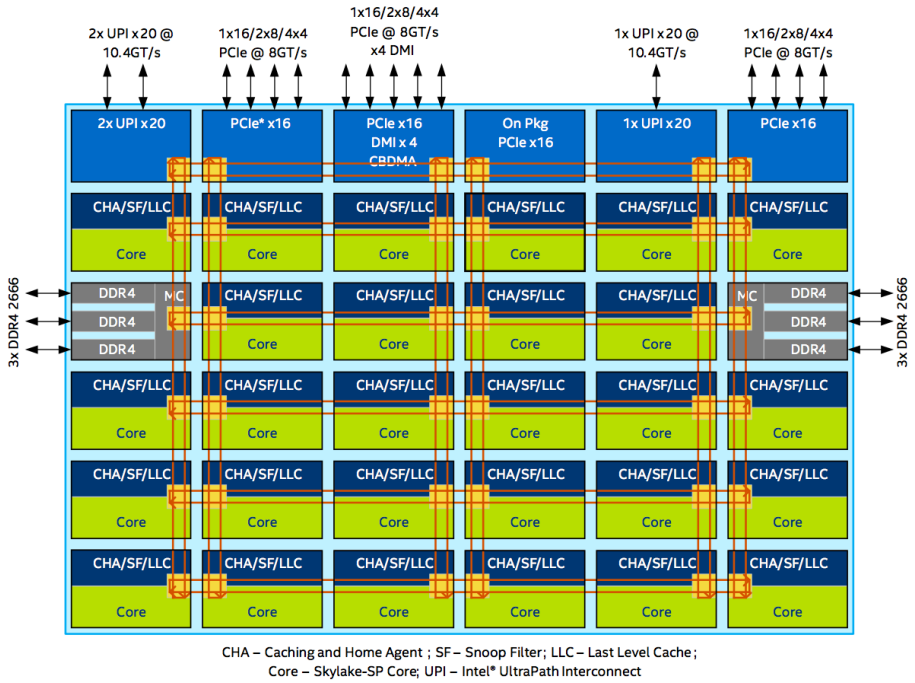


Figura 6.9: Subsistema uncore de la familia de procesadores Intel Skylake (hasta 28 cores)

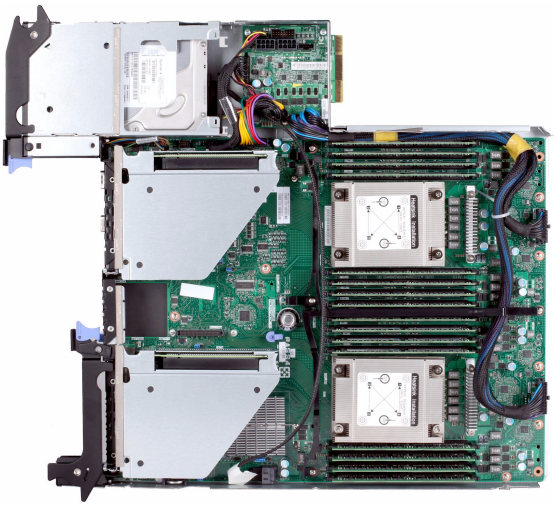
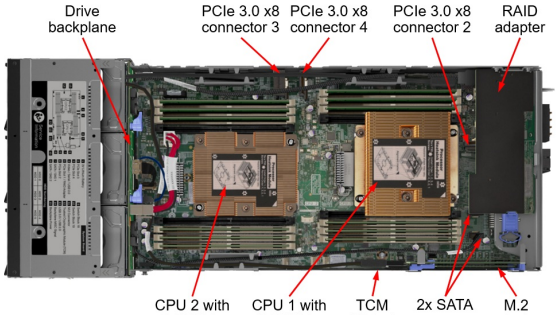
6.2. Nodos de cómputo

Como ya se ha visto, entre las dos últimas generaciones de MareNostrum ha habido un cambio sustancial en cuanto al número de cores disponibles en el procesador y el número de canales a memoria, así como las frecuencias de operación de los cores y los DIMM de memoria.

6.2.1. Resumen de nodos de cómputo

En la Tabla 6.2 se resumen las características de ambos nodos, detallando información del procesador, memoria, disco e interfaz de red.

Tabla 6.2: Cuadro resumen de nodos de cómputo

MareNostrum 3	**MareNostrum 4**
	
IBM System x iDataPlex dx360 M4 server	Lenovo ThinkSystem SD530 server
2x Intel Xeon E5-2670 8C 2.6GHz 20MB L3 1600MHz 115W	2x Intel Xeon Platinum 8160 24C 2.1GHz 33MB L3 2666MHz 150W
8x 16GB DDR3-1600 DIMMs 1.5V 1600MHz (8GB/core)	12x 8GB(32GB) TruDDR4 DIMMs 1.2V 2666MHz (2GB/core)
IBM 500GB 7.2K 6Gbps NL SATA 3.5	Intel 240GB s3520 SSD
Mellanox ConnectX-3 Dual Port QDR/FDR10 Mezzanine Card	Single Intel Omni-Path 100 Gigabit port
2x Gigabit Ethernet network card	2x 10Gigabit Ethernet network port

6.2.2. Configuración de la memoria

Con vistas a maximizar el ancho de banda de la memoria y el rendimiento global, los nodos de MareNostrum 3 y 4 tienen implementada una configuración de memoria equilibrada (*balanced memory*). Este tipo de configuración permite una intercalación óptima en todos los canales de memoria conectados y debe ajustarse a unas pautas básicas que conforme a [38] son las siguientes:

1. Todos los canales de memoria ocupados deben tener la misma capacidad de memoria total y el mismo número total de rangos.
2. Todos los controladores de memoria en un socket de procesador deben tener la misma configuración de DIMMs de memoria.
3. Todos los sockets del procesador en el mismo servidor físico deben tener la misma configuración de DIMMs de memoria.

En la misma línea, en [39, p. 18] además se recomienda poblar todos los canales de memoria, dado que el rendimiento de la memoria escala casi linealmente con la cantidad de canales ocupados. Juntando las pautas, para el servidor iDataPlex dx360 M4 (MareNostrum 3) se propone como regla general de rendimiento instalar 8 unidades DIMM del mismo tamaño y tipo, una por canal de memoria.

En consonancia con todo lo anterior, cada servidor iDataPlex dx360 M4 del clúster actual de MareNostrum 3 (Nord 3) cuenta con 8 DIMMs DDR3-1600 de 16 GB, ofreciendo así una capacidad máxima de 128 GB/node (8 GB/core). En la Figura 6.10 se muestra la topología de memoria del servidor iDataPlex dx360 M4 que conforma un nodo de MareNostrum 3.

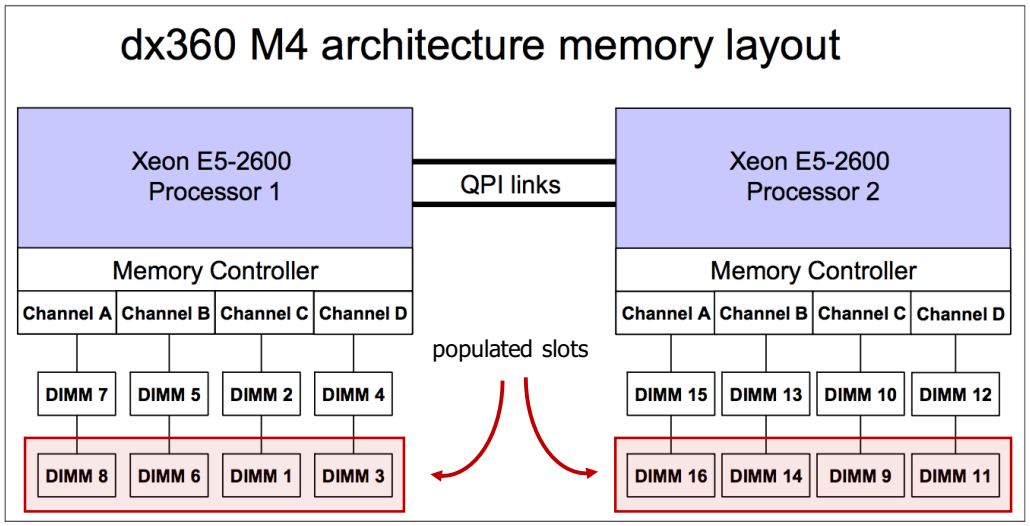


Figura 6.10: Topología de memoria de MareNostrum 3 (niveles socket y nodo)

Por la parte contraria, entre las opciones de configuración de memoria que admite el servidor SD530 de MareNostrum4, tenemos:

- Configuración de memoria con 12 módulos DIMM (balanced), optimizada para el **máximo rendimiento**.
- Configuración de memoria con 16 módulos DIMM (unbalanced), optimizada para la máxima capacidad de memoria con un rendimiento reducido[3].

La configuración implementada en el servidor SD530 para conformar un nodo de MareNostrum 4 es obviamente la primera, la que proporciona el máximo rendimiento. Bajo esta configuración, 3240 nodos (del total de 3456 nodos) alojan 12 DIMMs DDR4-2667 de 8 GB, distribuidos en 6 DIMMs por socket de la manera que se muestra en la Figura 6.11, uno por canal de memoria, ofreciendo una capacidad máxima de 96 GB/core (2 GB/core). Los 216 nodos restantes son los llamados "fat" o "highmem", con una topología exactamente igual salvo que cada DIMM DDR4-2667 tiene una capacidad de 32 GB, para una capacidad máxima de 384 GB/node (8 GB/core).

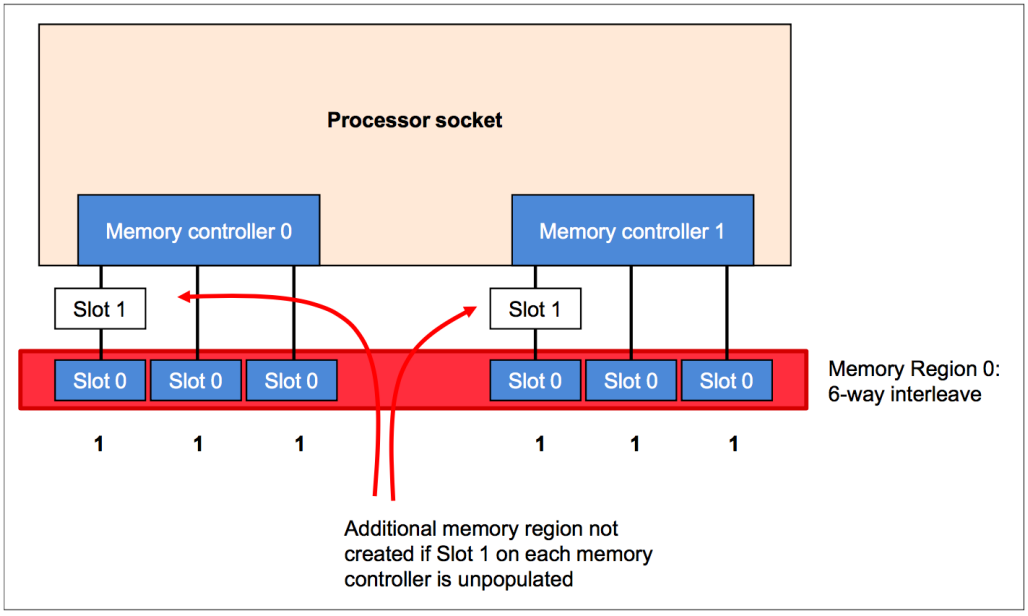


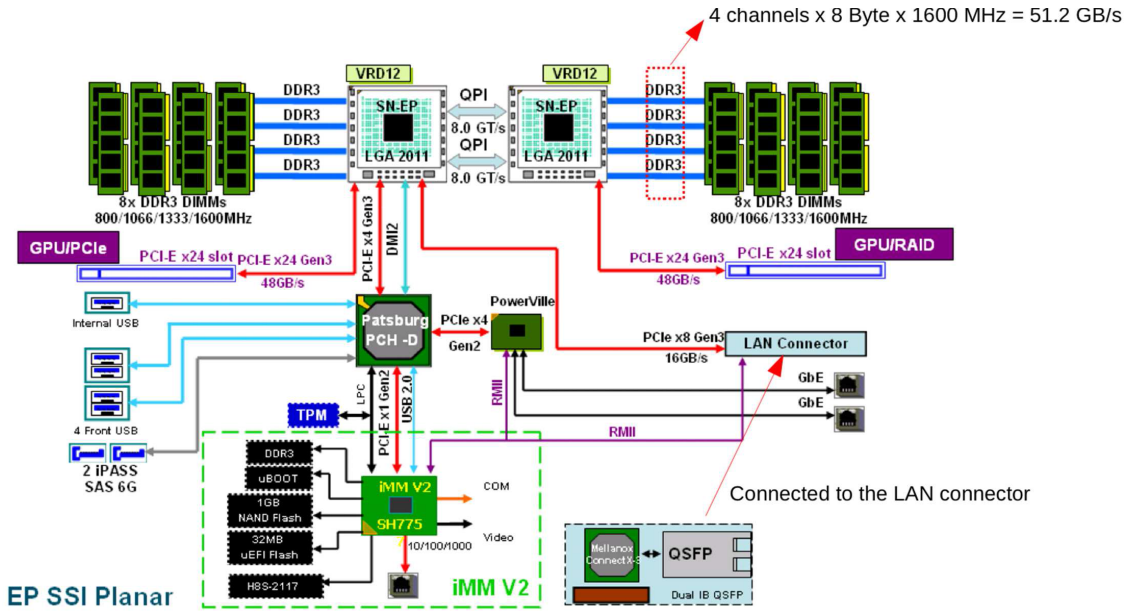
Figura 6.11: Topología de memoria de MareNostrum 4 (nivel socket)

6.2.3. Topología del sistema a nivel de nodo

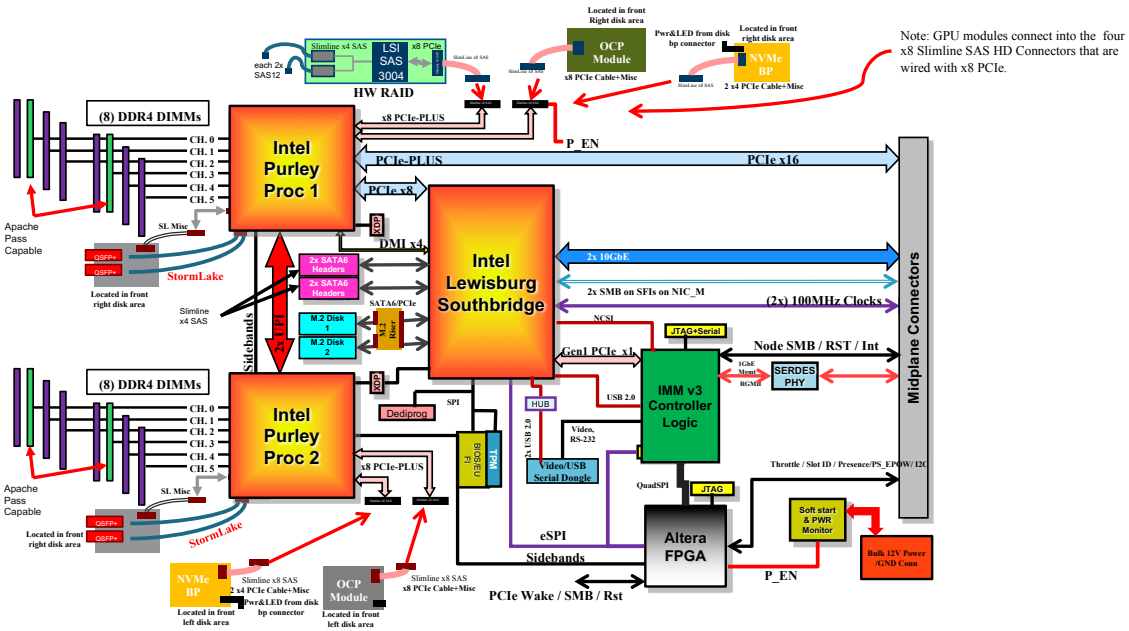
En los dos sistemas analizados, la interconexión de los dos procesadores a través de los enlaces QPI/UPI forman un sistema de memoria compartida coherente de la manera que

[3]Según la fuente citada, con esta configuración se ve reducido el ancho de banda pico hasta un 60 % a la vez que la latencia se incrementa un 6 %.

se ilustra en la Figura 6.12, donde pueden apreciarse también otras diferencias a nivel de arquitectura de los nodos que conforman los clústeres.



(a) MareNostrum 3



(b) MareNostrum 4

Figura 6.12: Diagramas de bloques de nodos de cómputo

Las Figuras 6.13 y 6.14 muestran la topología de ambos sistemas tal como es mostrada por la herramienta **lstopo** de Linux. En estas figuras también pueden observarse las capacidades de las memorias caché de niveles L1, L2 y L3, las capacidades de las memorias

principales (por socket y total por nodo), así como los adaptadores de red conectados a los puertos PCIe.

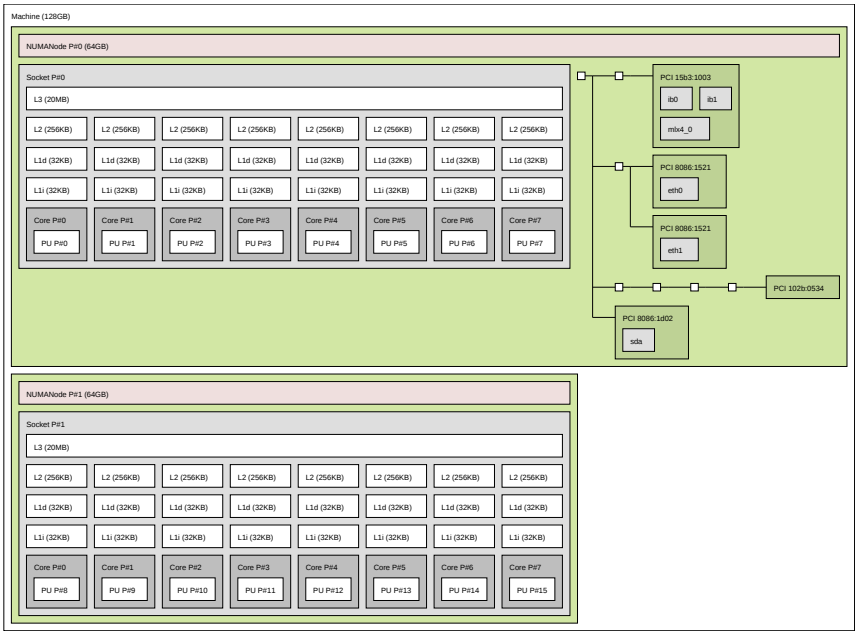


Figura 6.13: Topología del sistema de MareNostrum 3

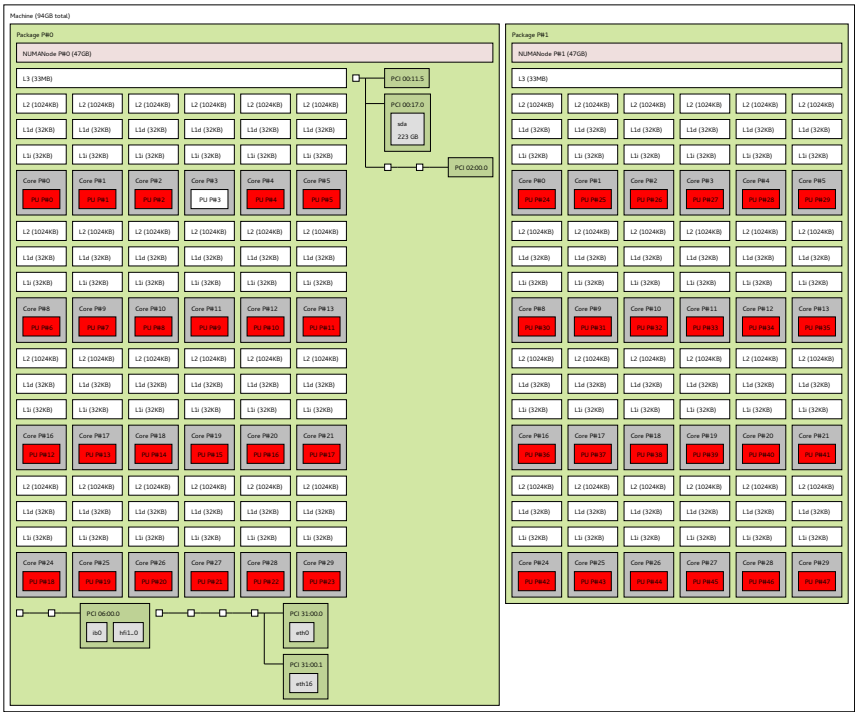


Figura 6.14: Topología del sistema de MareNostrum 4

6.3. Redes de interconexión

MareNostrum3, y por consiguiente su versión actual en producción Nord3, basa la interconexión de los nodos de cómputo descritos en el apartado anterior en una red de interconexión InfiniBand FDR10 (40 Gbps por enlace) *Full Fat Tree*. En la Figura 6.15 se muestra la interconexión de los 3056 nodos, utilizando conmutadores de 36 puertos en el primer nivel y conmutadores de 522 puertos en el segundo nivel.

Por su parte, MareNostrum4 basa la interconexión de los nodos de cómputo antes descritos en una red de interconexión Intel Omni-Path (100 Gbps) Full Fat Tree. En la Figura 6.16 se muestra la interconexión de los 3456 nodos, utilizando conmutadores de 48 puertos en el primer nivel y conmutadores de 768 puertos en el segundo nivel. Actualmente la red está sobredimensionada, pudiendo albergar más nodos en el futuro.

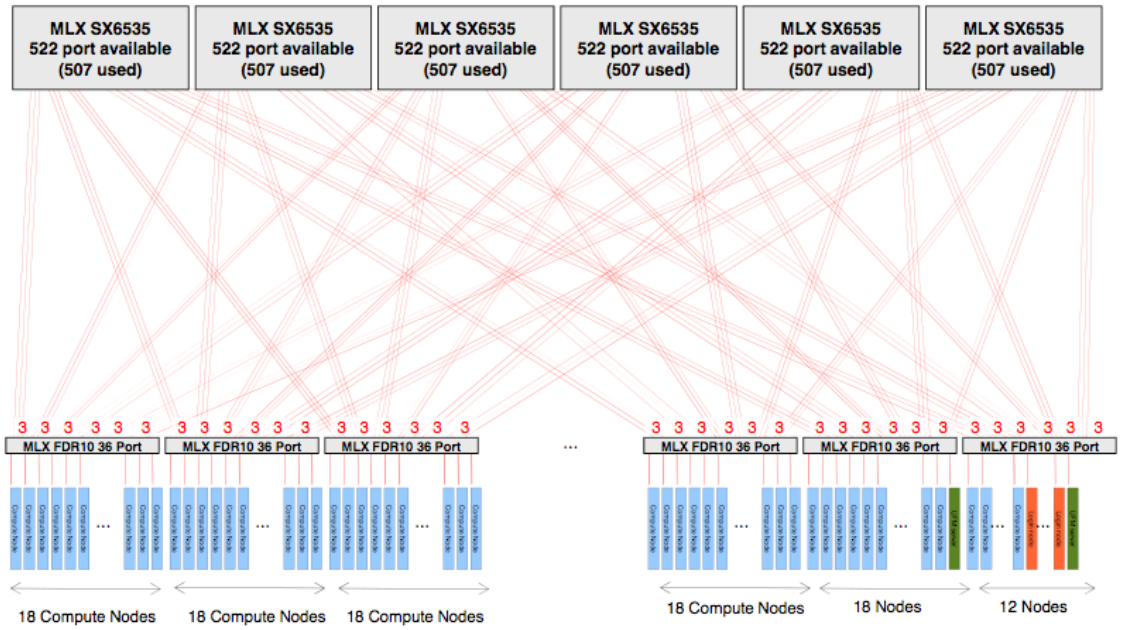


Figura 6.15: Red de interconexión de MareNostrum3

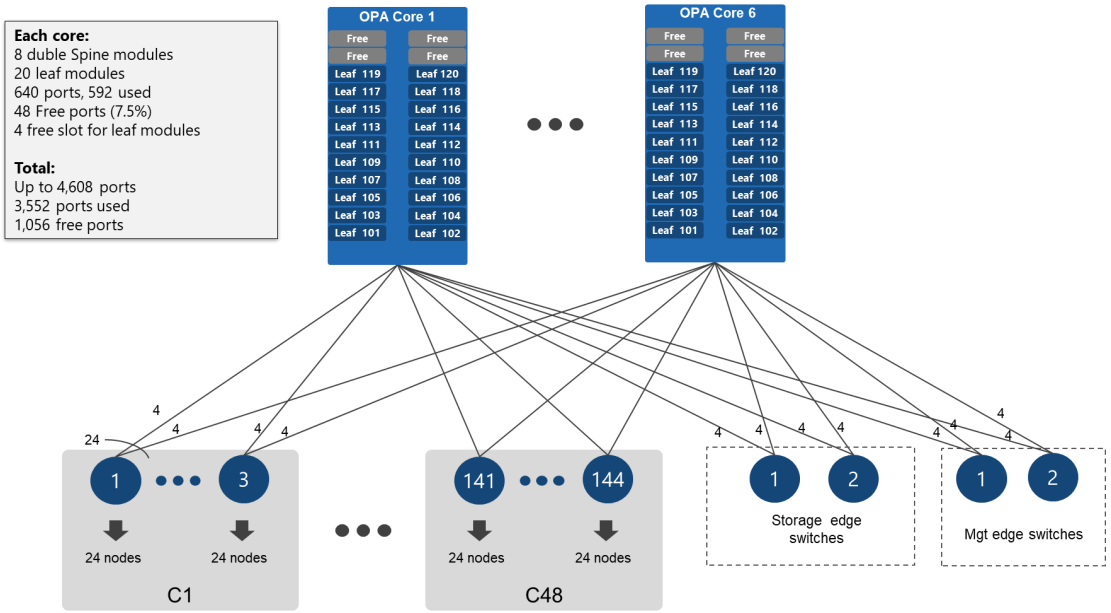
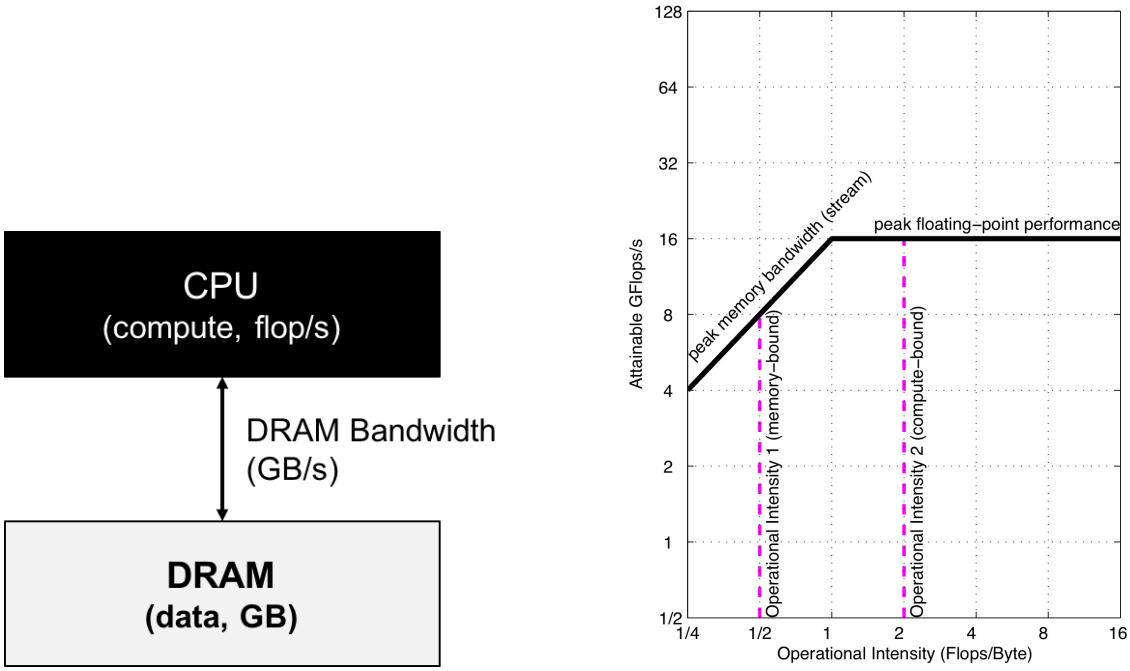


Figura 6.16: Red de interconexión de MareNostrum4

6.4. Modelos Roofline

El modelo *Roofline* [40–45] ofrece un modelo y una figura intuitivos para comprender el rendimiento del *kernel* en varias arquitecturas, siendo un modelo exclusivo de cada arquitectura que nos brinda un buen equilibrio entre simplicidad y representación descriptiva basado en dos conceptos importantes: la *intensidad operacional (OI)* y el *número de FLOPS*, usando el término 'intensidad operacional' para referirnos al número de operaciones de coma flotante por cada byte de tráfico de DRAM.

En este trabajo se utiliza la versión más simple del modelo Roofline, el cual permite visualizar de forma sencilla el balance entre la capacidad de cómputo —por ejemplo en GFlops de aritmética en doble precisión— de un elemento de proceso (núcleo, procesador, nodo, ...) y la capacidad de transferir datos a memoria —ancho de banda o memory bandwidth en GB/s— de ese elemento de proceso (véase Figura 6.17a). Los principales objetivos del modelo son los de proporcionarnos expectativas teóricas de rendimiento mediante una ayuda gráfica, además de mostrarnos limitaciones para un kernel o aplicación por parte de los elementos de proceso *(compute bound)* o por parte de la memoria *(memory bound)*.



(a) (b)

Figura 6.17: Modelo Roofline básico: interrelación de elementos que condicionan el rendimiento

Resumiendo, en un modelo Roofline básico (véase Figura 6.17b) el rendimiento está limitado tanto por la tasa máxima de FLOPS como por el producto del ancho de banda de transmisión por la relación flop:byte (OI). O dicho de otro modo, no siempre se puede alcanzar el máximo rendimiento en FLOPS, ya que la localidad finita (reutilización) y el ancho de banda de memoria pueden limitar dicho rendimiento.

En definitiva, la fórmula que sustenta el modelo Roofline es:

$$GFLOPS = min \begin{cases} Peak\ Floating\ Point\ Performance \\ Peak\ Memory\ Bandwidth\ \times\ Operational\ Intensity \end{cases} \tag{6.1}$$

6.4.1. Modelo a nivel de procesador/memoria

Lo primero que necesitamos para construir los modelos Roofline es calcular los máximos teóricos del rendimiento computacional (FLOPS) y del ancho de banda (GB/s) de cada elemento en los respectivos nodos, esto es considerando los accesos a los DIMMs locales del procesador y a los DIMMs remotos del otro procesador (NUMA) a través de los enlaces coherentes QPI o UPI.

Cálculo de los máximos teóricos del rendimiento computacional

Para calcular el rendimiento máximo en GFLOPS o *Peak Floating Point Performance* de un procesador (socket), utilizaremos la fórmula:

$$\frac{FLOPS}{socket} = \overbrace{\underbrace{\frac{flop}{operation}}_{\text{instructions}} \times \underbrace{\frac{operation}{instruction}}_{\text{vector}} \times \underbrace{\frac{instruction}{cycle}}_{\text{super-scalar}}}^{\text{micro-architecture}} \times \overbrace{\underbrace{\frac{cycle}{second}}_{\text{frequency}} \times \underbrace{\frac{core}{socket}}_{\text{multi-core}}}^{\text{machine architecture}} \quad (6.2)$$

La ecuación 6.2, que tomamos de [46], nos muestra de forma explícita cuáles factores forman parte o dependen de la microarquitectura (cómo está diseñada la CPU), y cuáles son parte de la arquitectura de la máquina (cuántos dispositivos, qué dispositivos, qué velocidad). A continuación se explica cada término de la ecuación con más detalle:

- $flop/operation$: cuántas "operaciones" (en el sentido matemático de la palabra) de punto flotante están integradas en la semántica de la instrucción (por ej., fused multiply-add $d = a \times b + c$ se representa por la relación $2/1$).
- $operation/instruction$: cuántas operaciones se ejecutan de forma simultánea o en grupo por una instrucción. Este es el modo llamado "vector" o "SIMD" (por ej., una instrucción que usa un vector de 4 elementos como operandos para una relación de $4/1$).
- $instruction/cycle$: cuántas instrucciones se pueden ejecutar simultáneamente en un ciclo (por ej., si una CPU puede ejecutar dos fused multiply-add a la vez, la proporción será $2/1$).
- $cycle/second$: cuántos ciclos en un segundo. Las CPU modernas tienen el modo "turbo" mediante el cual algunos de los núcleos pueden tener una velocidad mayor que la nominal. Por el contrario, el modo de ahorro de energía reduce la frecuencia. Las CPU Intel recientes tienen una frecuencia base diferente cuando se explotan ciertas instrucciones, en particular las instrucciones AVX.
- $core/socket$: cuántos núcleos en un nodo. El número de núcleos no tiene en cuenta el *Hyper-Threading*, ya que si bien éste agrega recursos adicionales a los procesadores vistos como "núcleos" por el sistema operativo, no agrega unidades de ejecución.

Los resultados de los cálculos mediante 6.2 se muestran en la tabla siguiente, con dos variantes de picos teóricos en GFLOPS para cada modelo de CPU, la primera aplicando

Tabla 6.3: Cálculo de picos teóricos en GFLOPS por socket

	MareNostrum 3		**MareNostrum 4**	
	AVX (DP)		AVX-512 + 2x FMA (DP)	
	Base freq.	Turbo freq.	Base freq.	Turbo freq.
$flop/operation$	1	1	2	2
$\times\ operation/instruction$	4	4	8	8
$\times\ instruction/cycle$	2	2	2	2
$\times\ cycle/second$	2,6 G	3 G	1,4 G	2 G
$\times\ core/socket$	8	8	24	24
$= FLOPS/socket$	166,4 G	192 G	1075,2 G	1535 G

la frecuencia base del procesador, y la segunda considerando el incremento de frecuencia (overlocking) proporcionado por el control Turbo Boost al utilizarse el máximo de cores.

Es importante recalcar sin embargo que "la única frecuencia 'garantizada' para todas las cargas de trabajo es la frecuencia nominal (base) del procesador, aunque en la práctica parte de la capacidad de Turbo Boost sigue siendo posible incluso con códigos AVX altamente optimizados" [39, p. 15].

Cálculo de los máximos teóricos del rendimiento de ancho de banda

En cuanto al ancho de banda máximo *(Peak Memory Bandwidth)*, tanto para acceso local (DRAM) como remoto (NUMA), la fórmula será:

$$PMB = N \times T \times B \tag{6.3}$$

donde N es el número de canales de memoria del procesador (acceso local) o el número de enlaces QPI/UPI (acceso remoto), T el número de transacciones por segundo —*GT/s, giga-transfers per sec.*[4]—, y B el número de bytes por transacción.

MareNostrum 3 DDR3 socket memory:

$PMB_1 = 4\ DDR3\ channel \times 1,6\ GT/s \times 8\ Byte/T = 51,2\ GB/s$

[4]Equivale a la frecuencia multiplicada por 2

MareNostrum 3 QPI inter-socket:

$$PMB_2 = 2\,QPI\ link \times 8\,GT/s \times 2\,Byte/T = 32\,GB/s$$

MareNostrum 4 DDR4 socket memory:

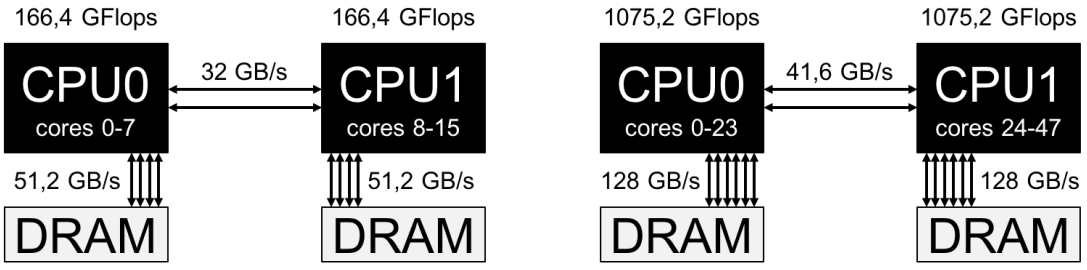
$$PMB_3 = 6\,DDR4\ channel \times 2,667\,GT/s \times 8\,Byte/T = 128\,GB/s$$

MareNostrum 4 UPI inter-socket:

$$PMB_4 = 2\,UPI\ link \times 10,4\,GT/s \times 2\,Byte/T = 41,6\,GB/s$$

Modelos Roofline resultantes

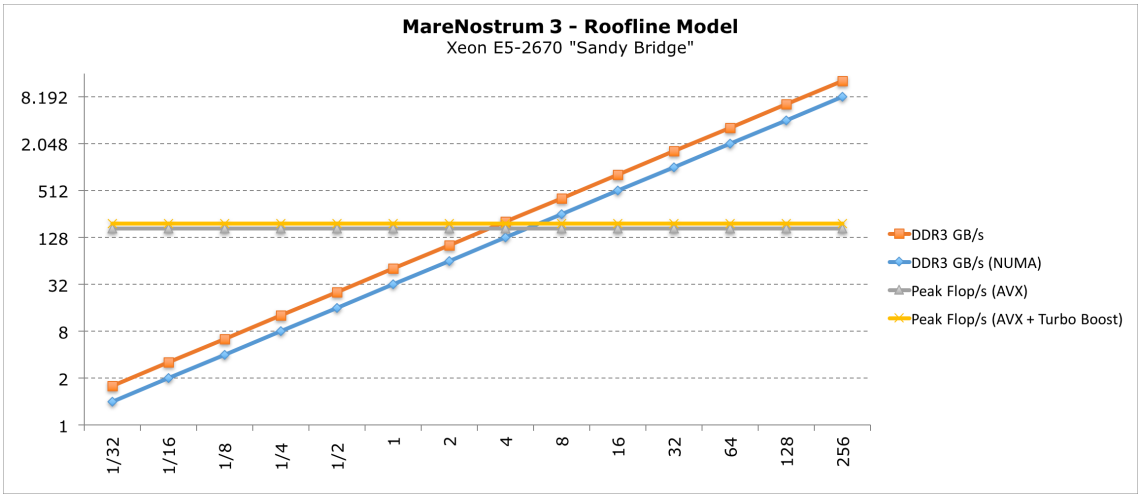
Los resultados nos dejan dos escenarios como los que se muestran en la Figura 6.18, dando lugar a los respectivos modelos correspondientes a cada arquitectura, quedando estos modelos finalmente representados en la Figura 6.19. Observar como el punto de corte del modelo se ha desplazado hacia la derecha en la transición de Marenostrum 3 a 4 (OI por un factor entre 2 y 4), indicando que la relación entre GFlops y GB/s ha aumentado, o en otras palabras, que el sistema procesador/memoria ofrece menos ancho de banda por capacidad de cómputo.



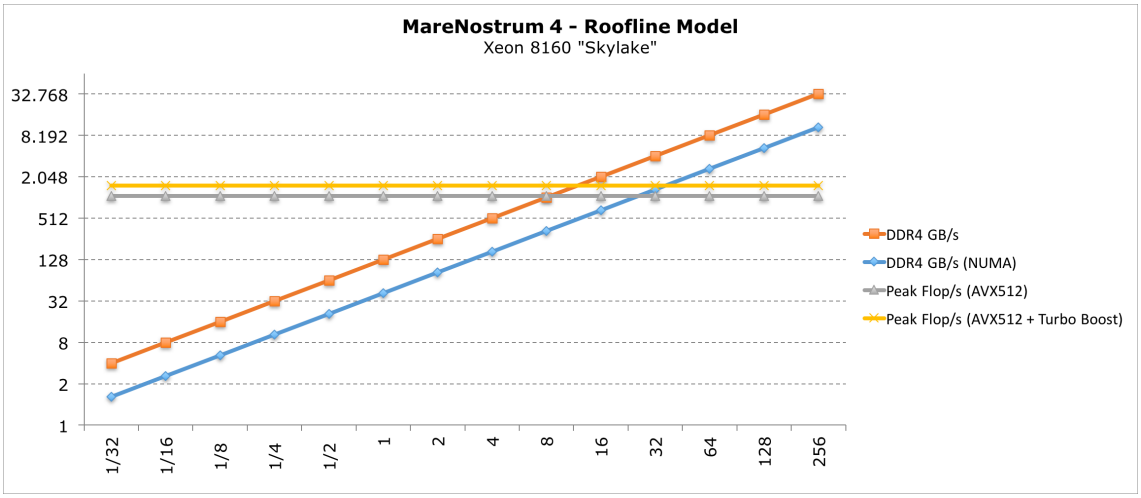
(a) MareNostrum 3

(b) MareNostrum 4

Figura 6.18: Sistemas formados por procesador y memoria DRAM con sus valores máximos teóricos



(a) MareNostrum 3

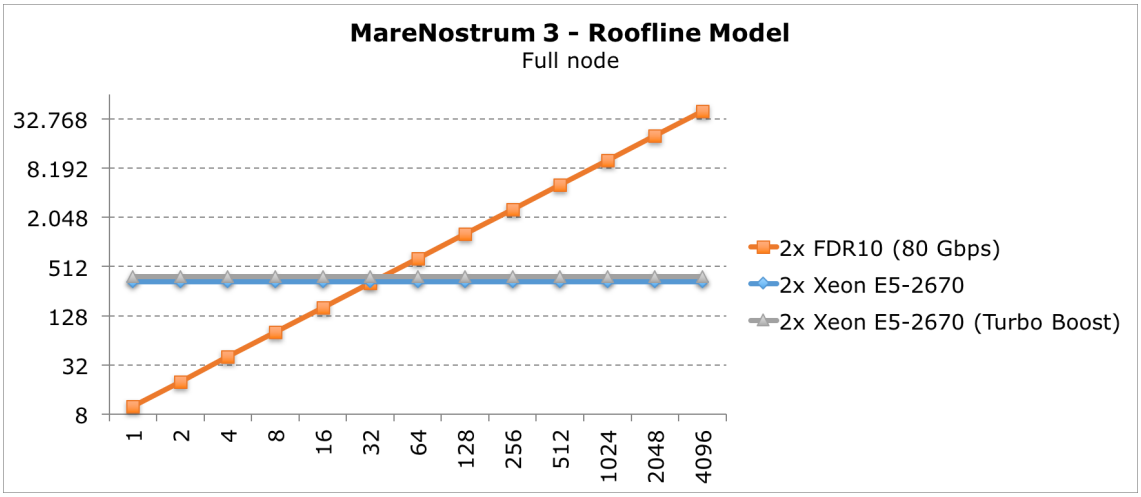


(b) MareNostrum 4

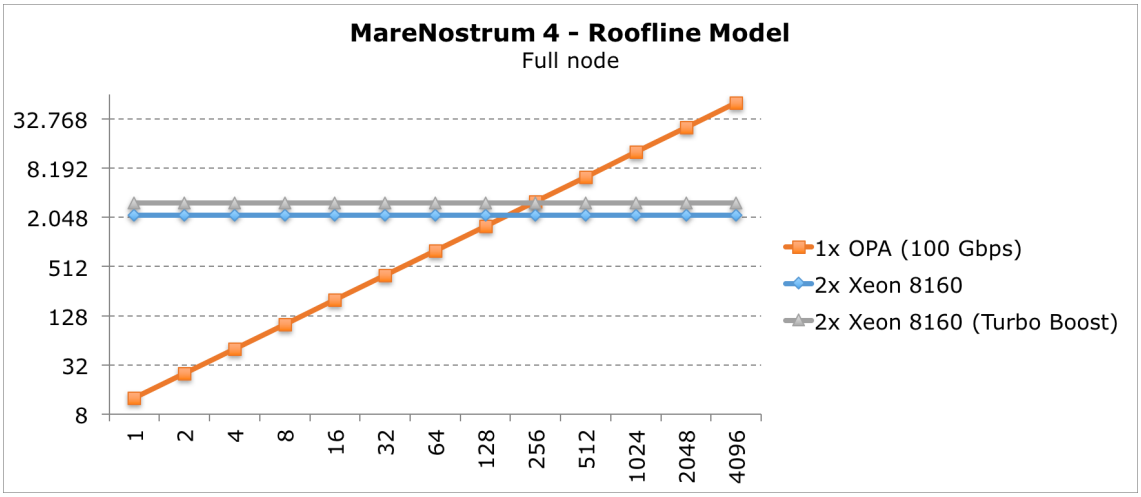
Figura 6.19: Modelos Roofline para procesador y memoria DRAM + NUMA

6.4.2. Modelo a nivel de nodo completo

El modelo de roofline model puede ampliarse a un nodo completo, permitiendo en este caso analizar de forma rápida e intuitiva la relación entre la capacidad de computo del nodo y la capacidad de transferir datos a través de las interfaces de red disponibles para el intercambio de datos entre nodos (red de interconexión). En los dos casos se ha considerado el uso de la interfaz InfiniBand (IB)/Omni-Path (OPA) únicamente para acceder a la red de interconexión.



(a) MareNostrum 3, con adaptador de red IB FDR10



(b) MareNostrum 4, con adaptador de red OPA

Figura 6.20: Modelos Roofline para nodo completo

La Figura 6.20 muestra la representación gráfica para cada modelo a nivel de nodo de las arquitecturas analizadas. Para Marenostrum 3 se ha considerado que la interfaz IB FDR10 ofrece un ancho de banda de 80 Gbps (2 puertos por nodo) mientras que la interfaz OPA ofrece un ancho de banda de 100 Gbps (1 puerto por nodo). De nuevo observar como en la transición de Marenostrum 3 a 4 el sistema se ha desbalanceado en cuanto a capacidad de cómputo y transferencia, moviéndose el punto de corte de una OI por un factor de 4.

Capítulo 7

Diseño de benchmarking

7.1. Necesidades y componentes de evaluación

En todo centro de supercomputación resulta imprescindible evaluar el rendimiento con vistas a su optimización. Si bien es posible consultar las características y especificaciones técnicas que proporciona el fabricante de un supercomputador, el comportamiento real de cada máquina puede variar dependiendo de factores tales como la carga de trabajo a la que se someta. Otro factor clave consiste en la correcta parametrización de las diversas aplicaciones con el fin de aprovechar las características de una arquitectura dada. Estos parámetros sólo se podrán ajustar mediante el análisis en profundidad del comportamiento de los distintos subsistemas [47, p. 8].

Con las mediciones de los componentes del sistema no sólo obtenemos información sobre el comportamiento de las características individuales de la máquina. También podemos calcular, a partir del rendimiento relativo de las mediciones, la contribución de una característica a la solución global de un problema computacional. El rendimiento de los componentes respecto a un límite máximo nos muestra cuan eficientemente se explota una prestación de la máquina en comparación con los límites teóricos [48, pp. 6-7].

Por otra parte, si hablamos de tipos de cargas de trabajo, coincideremos con [49, p. 4] en incluir a la mayoría de las aplicaciones en una de las categorías siguientes:

a) **Serial**: las aplicaciones serie son aquellas que usan un solo núcleo y rara vez usan instrucciones vectoriales, siendo en este escenario la frecuencia de Turbo Boost el aspecto más importante a tener en cuenta desde el punto de vista del procesador.

b) **Scalar, multicore**: estas aplicaciones utilizan múltiples hilos o procesos para aprovechar todos los núcleos de una CPU. No usan instrucciones vectoriales y la frecuencia de Turbo Boost está limitada por el número de núcleos activos.

c) **Vector, multicore**: son aplicaciones plenamente optimizadas para procesadores paralelos, con el fin de escalar a través de todos los núcleos y además hacer uso del paralelismo de datos *(DLP, data-level parallelism)* dentro de un núcleo, también llamado "procesamiento vectorial". Cuando estas aplicaciones se ven limitadas por el rendimiento aritmético, operan cerca de las frecuencias de Turbo Boost específicas para este tipo de cargas de trabajo.

d) **Bandwidth-limited**: las frecuencias de reloj en este tipo de cargas de trabajo llegan a ser inútiles dado que, aún siendo aplicaciones altamente paralelas y con estructuras de datos 'amigables' con la vectorización, no alcanzan el máximo rendimiento aritmético, estando éste limitado por el ancho de banda de la memoria. Generalmente esto sucede cuando el código no está suficientemente optimizado para la utilización de la memoria caché o porque la intensidad operacional del algoritmo no es lo suficientemente alta (véase sección 6.4 Roofline models).

En general, los benchmarks sintéticos como LINPACK o STREAM están desarrollados en base a códigos específicos que han sido seleccionados *ad-hoc*, es decir, están pensados para un determinado fin o un escenario en concreto, de manera que en el mejor de los casos sólo reflejan el rendimiento (pensemos en rendimiento global y no de una característica individual o subsistema) de un conjunto muy limitado de aplicaciones. Por eso no nos deben servir como punto de referencia general contra la cual se pueda juzgar el progreso en el rendimiento real de una aplicación [50, p. 1].

En consecuencia, es ampliamente reconocida la necesidad de evaluación del rendimiento y benchmarking con aplicaciones realistas, por parte de comunidades que exigen mejores y más amplios criterios, desde clientes de sistemas HPC hasta fabricantes de computadoras, agencias de financiación de la investigación o equipos científicos [48, p. 2].

7.2. Descripción de benchmarks

En esta sección se enumeran (ordenados alfabéticamente) los benchmarks seleccionados y se ofrece una breve descripción de los mismos. Éstos se han dividido en dos grupos o categorías: 1) benchmarks sintéticos *(micro-benchmarking)* y 2) aplicaciones científicas reales, criterio que se ha establecido como parte de la metodología a seguir y también para favorecer una mejor organización de este documento.

7.2.1. Benchmarks sintéticos (microbenchmarks)

HPCG (High Performance Conjugate Gradients)

HPCG [51, 52] está pensado como un complemento de LINPACK. La diferencia crucial entre HPL y HPCG radica en que el kernel computacional en HPL es una multiplicación de matriz, mientras que en HPCG es una multiplicación vectorial de matriz. Al haber una mayor tasa de Byte/Flop de la multiplicación vectorial matricial, esto hace que esté más limitado por la memoria en el hardware actual, lo que le convierte en más representativo para las aplicaciones del mundo real.

Intel MPI Benchmarks

MPI Benchmarks [53] realiza un conjunto de mediciones de rendimiento MPI para operaciones de comunicación global y punto a punto para una variedad de tamaños de mensaje, caracterizando completamente el rendimiento de un sistema de clúster, lo que incluye el rendimiento de nodos, la latencia de la red y la productividad, además de la eficiencia de la implementación de MPI utilizada.

lat_mem_rd (memory read latency benchmark)

Este benchmark forma parte de LMbench - Tools for Performance Analysis [54], una suite de becnhmarks simples y portables para comparar el rendimiento de diferentes (sub)sistemas UNIX. Con lat_mem_Rd se puede medir la latencia de lectura de memoria de toda la jerarquía de memoria, lo cual incluye la latencia y tamaño de memoria de las caches integrada y externa, la latencia de memoria principal y la latencia de fallos de TLB.

LINPACK

LINPACK (o HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers) es el benchmark ampliamente más utilizado y de referencia para la lista TOP500, para medir la capacidad global de coma flotante de los sistemas HPC. El informe de resultados del benchmark describe el rendimiento al resolver un problema general de matriz densa $Ax = b$ en aritmética de coma flotante de 64 bits, con tres niveles de tamaño de problema y de optimización que son: problema de 100 por 100 con optimización de bucle interno, problema de 1000 por 1000 con optimización de 3 bucles (todo el programa) y un problema paralelo escalable [55]

STREAM

STREAM [56–58] es un programa sintético simple de referencia que mide el ancho de banda de memoria sostenible (MB/s) y la tasa de cálculo correspondiente para kernels vectoriales simples. Escrito en Fortran 77 estándar con su correspondiente versión en C, el programa mide el rendimiento de cuatro operaciones típicas de los *building blocks* de operaciones sobre vectores de gran tamaño, que son:

name	kernel	bytes/iter	FLOPS/iter
COPY:	a(i) = b(i)	16	0
SCALE:	a(i) = q*b(i)	16	1
SUM:	a(i) = b(i) + c(i)	24	1
TRIAD:	a(i) = b(i) + q*c(i)	24	2

definiéndose los tamaños de vectores de modo que cada array sea más grande que la memoria caché de la máquina. El código está estructurado para que no sea posible la reutilización de datos, ya sea en registros o en caché.

7.2.2. Aplicaciones científicas reales

ALYA

Alya - High Performance Computational Mechanics [59, 60] es un código de simulación para mecánicas computacionales de alto rendimiento, para resolver problemas complejos provenientes principalmente del ámbito de la ingeniería, entre los que destacan flujo incompresible/compresible, mecánica de sólidos no lineales, química, transporte de partículas, transferencia de calor, modelos de turbulencia, propagación eléctrica, etc.

CPMD

CPMD [61] es un programa que contiene una implementación del método Car-Parrinello, diseñado para cálculos *ab initio* de dinámica molecular con la Teoría del Funcional de la Densidad (DFT).Su uso está restringido a los usuarios académicos

GROMACS

GROMACS (GROningen MAchine for Chemical Simulations) [62] es un paquete versátil para realizar dinámicas moleculares, o sea simular las ecuaciones newtonianas de

movimiento para sistemas con cientos o millones de partículas.

VASP

El paquete de simulación Vienna Ab initio (VASP) [63, 64] es un programa para el modelado de materiales a escala atómica, como cálculos de estructura electrónica y mecánica cuántica para simulaciones de dinámica molecular, a nivel ab initio.

WRF

The Weather Research and Forecasting (WRF) Model [65, 66] es un sistema de cálculo numérico para simulación atmosférica (NWP) diseñado para satisfacer las necesidades tanto de investigación como de predicción atmosféricas.

7.3. Metodología general

A lo largo de este trabajo se ha consultado gran variedad de documentación dedicada en buena medida al análisis de resultados de multitud de benchmarks, Sin embargo se ha constatado que, en general, existe escasa cobertura en cuanto a la metodología de evaluación del rendimiento, tal como se manifiesta en [67]. En esta misma fuente se enumeran tres factores principales que afectan el rendimiento de un sistema informático y que han servido como base de la metodología aplicada en este análisis comparativo de rendimiento:

1. Diseño del sistema
2. Implementación del sistema
3. Carga de trabajo a la que está sujeto el sistema

Partimos del hecho de que la evaluación comparativa de los procesadores modernos es algo bastante complejo, puesto que se tienen que controlar, como mínimo y según [31, p. 1], características como:

- frecuencia del procesador
- overclocking mediante el modo Turbo
- número de cores físicos en uso

- uso de cores lógicos mediante SMT (Simultaneous Multi-Threading)
- tamaños de caché disponibles
- configuración de memoria instalada
- configuración de potencia (si se va a medir el rendimiento por vatio)
- vectorización
- software: operating system, compiler, libraries, ...

En esta sección se explican distintos criterios tenidos en cuenta para la realización del benchmarking, así como la metodología o metodologías seguidas en líneas generales.

7.4. Metodología general de benchmarking

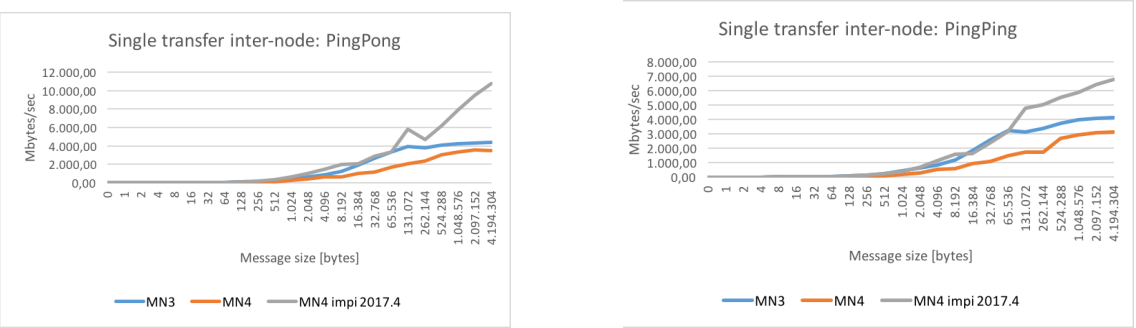
7.4.1. Elección de versiones de software (compiladores y librerías)

Una de las decisiones importantes durante los comienzos del benchmarking consistió en la elección de las versiones de software a utilizar. Concretamente nos referimos a los compiladores que hemos hecho servir para generar los binarios, pero también de las librerías usadas para el linkado estático/dinámico de los programas. En un principio, la intención fue la de usar el mismo software en ambas máquinas, ayudándonos para tal fin mediante el sistema de módulos de Linux [68], intentando así conseguir dos entornos lo más parecidos posible tanto en tiempo de compilación como de ejecución y poder obsevar el comportamiento de las máquinas en entornos "gemelos".

Enseguida nos dimos cuenta de que no era un planteamiento acertado. Lo primero, no tenía demasiada lógica utilizar versiones de software que no fuesen las más rodadas y optimizadas para cada máquina hasta la fecha, las de uso más común entre la comunidad de usuarios del supercomputador. Por otra parte, tampoco le vimos ningún sentido a hacer un estudio comparativo sobre la evolución de la nueva arquitectura respecto a la anterior y no usar las actualizaciones del software pertinentes, en su mayoría dirigidas a corregir o mejorar las prestaciones en función de las nuevas características resultantes de dicha evolución —nuevo conjunto de intrucciones AVX512, nueva red de interconexión, etc.—.

En esas topamos con un claro ejemplo que fue definitivo para reconsiderar esta cuestión. Al ejecutar los benchmarks pingpong y pingping de la suite Intel MPI Benchmark, con la versión de la librería Intel MPI que habíamos elegido para uso común en las dos máquinas

(2017.1), no se entendían los resultados tan negativos que se daban en MareNostrum 4, representados por la línea naranja en la Figura 7.1. Después de indagar descubrimos que una de las aportaciones relevantes de la versión 2017.4 de esa librería radicaba en el "ajuste de rendimiento para procesadores basados en microarquitectura Intel con el nombre en código Skylake y para Intel Omni-Path Architecture" [69].



(a) (b)

Figura 7.1: Pruebas de rendimiento de librerías Intel MPI

Al repetir las ejecuciones usando la actualización 2017.4, se consiguieron los resultados representados por la línea gris en la misma Figura 7-1, resultados diametralmente opuestos a los anteriores.

Finalmente, se eligieron las versiones de las librerías de Intel que se detallan en la Tabla 7.1, siendo las utilizadas en casi la totalidad de las pruebas de este trabajo y las que se cargan por defecto vía módulos en los entornos correspondientes.

Tabla 7.1: Pila de software utilizado (compilador y librerías)

	MN3	**MN4**
Intel C++ Compiler	2017.1	2017.4
Intel MPI Library	2017.1	2017.4
Intel Math Kernel Library (MKL)	2017.1	2017.4

7.4.2. Número de ejecuciones y recolección de resultados

Con respecto al número de ejecuciones, se ha jugado con varios aspectos. El primero de ellos ha sido la propia naturaleza del benchmark, puesto que el número de repeticiones que puede ejecutar un determinado benchmark respecto a otro para construir el correspondiente informe de resultados puede variar notablemente. De manera que en los casos de ejecuciones únicas (por ej. las aplicaciones reales) hemos ejecutado los jobs

entre 18-36 veces, utilizando casi siempre job arrays de tamaños 3, 6 y 9, esto para cada variante o configuración del job.

Por el contrario, para los benchmarks ya diseñados para realizar múltiples ejecuciones o que permiten parametrizar la cantidad de repeticiones o muestras deseadas, nos ha parecido suficiente entre 6-9 ejecuciones por variante/configuración.

Otra cuestión tenida en cuenta ha sido la homogeneidad (o variabilidad) de los resultados, es decir que en algunos casos se han repetido jobs más veces de las previstas de cara a tener mayor certeza acerca de las métricas obtenidas o de identificar mejor los patrones de comportamiento. Por último, aunque en bastante menor medida, nos hemos encontrado con jobs de nuestro interés que, o bien sus ejecuciones han sido considerablemente extensas, o han estado muy limitadas por la disponibilidad de recursos, de manera que en esos casos (excepcionales) nos hemos conformado con un par o tres de ejecuciones, eso sí, siempre contextualizando los resultados en base a este hándicap.

En cuanto a la recolección de métricas desde los ficheros generados por los benchmarks, se han utilizado varios indicadores estadísticos estándares (GEOMEAN, AVERAGE, MIN, MAX y STDEV) procurando una mejor interpretación y validación de los resultados. A la hora de plasmar los resultados en nuestras gráficas, el indicador por el que hemos optado es la media goemétrica (GEOMEAN), a menos que algunos resultados nos lo haya impedido (ceros), utilizando entonces la media aritmética (AVERAGE).

Al mismo tiempo, para los benchmarks que generan informes de resultados complejos o densos (por ej. Intel MPI Benchmarks) ha sido imprescindible la programación de scripts para agilizar y precisar la extracción de las métricas que necesitamos. En ese sentido, estos scripts pueden reportar un doble beneficio ya que pueden ser reaprovechables en la implementación del monitoring.

Capítulo 8

Análisis de microbenchmarking

8.1. Evaluación de rendimiento de la jerarquía de memoria

En esta sección se medirá **latencia** y **ancho de banda** sobre varios niveles de la jerarquía de memoria, motivados por el hecho de que el rendimiento de acceso a memoria de los programas se puede optimizar para ambos términos. Aunque la mayoría de los programas parece que tengan preferencia por una latencia más baja, hay algunos que prefieren mayor ancho de banda [70, p. 2]. Más aún, son aspectos que están intrínsecamente interrelacionados y entender esta relación puede ser clave para comprender el impacto del ancho de banda en el rendimiento, como se explica en [71].

Para evaluar el rendimiento del subsistema de memoria en términos de latencia y ancho de banda, es necesario poner el foco en dos cuestiones:

- *NUMA policy*
- *CPU affinity (Thread affinity, Processor affinity, CPU pinning)*

La *política de NUMA*[60-64] determina cómo colocar asignaciones de memoria en nodos específicos para que los programas accedan a él lo más rápido posible. La manera básica de hacerlo es asignando la memoria para un proceso en su nodo local y manteniendo allí su ejecución (afinidad de nodo), lo que proporciona la mejor latencia de acceso a memoria y minimiza el tráfico a través de la interconexión global.

Si hablamos de la *afinidad de CPU* [72, 73] nos estamos refiriendo a la capacidad del planificador de Linux para vincular uno o más procesos a uno o más procesadores con

el fin de aprovechar la localidad de la memoria, lo que reporta como primer beneficio la optimización del rendimiento de la caché al reducirse la tasa de fallos.

8.1.1. Latencia

Con la utilidad **lat_mem_rd** de LMBench es posible obtener mediciones de latencia de acceso a datos (se omite la caché de instrucciones) bastante precisas. El benchmark ejecuta dos bucles anidados: el bucle externo es el *stride*, el bucle interno es el tamaño de matriz. Para cada tamaño de matriz, se crea un anillo de punteros que apuntan una "zancada" (stride) hacia adelante. La matriz se recorre mediante:

```
p = (char **) * p;
```

Acceso secuencial de un solo core

Para empezar se ha ejecutado el benchmark lat_mem_rd lanzando 1 proceso sobre 1 core, ocupando ese proceso todo un nodo de cómputo en exclusiva y repitiendo varias veces el mismo experimento sobre 4 cores lógicos distintos, elegidos al azar.

Con la utilidad **numactl** [74] de Linux podemos especificar desde la línea de comandos el core en el que debe ejecutarse el proceso (afinidad de CPU) y el nodo NUMA en el que debe asignarse la memoria (política de NUMA). Entonces hay que conocer el inventario de nodos NUMA de los dos sistemas objeto de estudio, esto lo hacemos ejecutando el comando

```
numactl –H
```

En MareNostrum 3 el comando genera la salida

```
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7
node 0 size: 65508 MB
node 0 free: 61500 MB
node 1 cpus: 8 9 10 11 12 13 14 15
node 1 size: 65536 MB
node 1 free: 62648 MB
node distances:
node   0   1
0: 10  11
1: 11  10
```

En MareNostrum 4 la salida es

```
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
19 20 21 22 23
node 0 size: 47958 MB
node 0 free: 14085 MB
node 1 cpus: 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47
node 1 size: 48352 MB
node 1 free: 11720 MB
node distances:
node   0   1
0: 10  21
1: 21  10
```

Se comprueba que la configuración en los dos sistemas es del tipo SNC-2. Aparte del número de nodos NUMA, otra información que nos interesa es la relacionada con los procesadores lógicos individuales asociados con cada nodo NUMA, para su uso en la afinidad de procesos. Las distancias entre nodos no debemos tenerlas demasiado en cuenta, ya que como se verá más adelante no son valores del todo fiables. La justificación puede estar en que algunos proveedores (especialmente para sistemas de 2 nodos) tienen como norma simplemente informar "10" y "20" sin tener en cuenta las diferencias reales de latencia de la memoria [70, p. 8].

Así por ejemplo, para lanzar lat_mem_rd sobre el core 7 con asignación de memoria en el nodo NUMA 0, ejecutamos la línea de comando:

```
numactl --membind=0 --physcpubind=7 lat_mem_rd -N 50 1536 64
```

Mediante los parámetros "`-N 50`", "`1536`" y "`64`" definimos, respectivamente, el número de repeticiones —obtención de resultados estadísticamente fiables—, el tamaño de matriz en MBytes —qué tan lejos en la memoria se medirá—, y el tamaño de stride en Bytes. Los resultados de estas primeras mediciones se muestran en la Figura 8.1.

Antes de analizar los resultados, dado que este benchmark utiliza offsets de tamaño fijo, se advierte que puede ser vulnerable a las políticas de prefetching de caché "inteligentes". En ese sentido, si el stride no es lo suficientemente grande, los procesadores modernos son capaces de obtener previamente los datos y proporcionar latencias artificiales para la región de memoria del sistema.

A través de otra gran herramienta de Linux, **perf stat** [75], se han recopilado estadísticas de contadores de rendimiento en tiempo de ejecución que son provechosas para este análisis (Figura 8.2).

Una indicación a priori de qué tan bien está funcionando la caché nos la puede dar la proporción *fallos de caché por instrucción*:

$$CMPI_{MN4} = \frac{11{,}1 \times 10^9\ cache\text{-}miss}{83{,}4 \times 10^9\ instruction} = \mathbf{0{,}13}\ cache\text{-}miss/instruction$$

$$CMPI_{MN3} = \frac{2{,}9 \times 10^9\ cache\text{-}miss}{114{,}5 \times 10^9\ instruction} = \mathbf{0{,}03}\ cache\text{-}miss/instruction$$

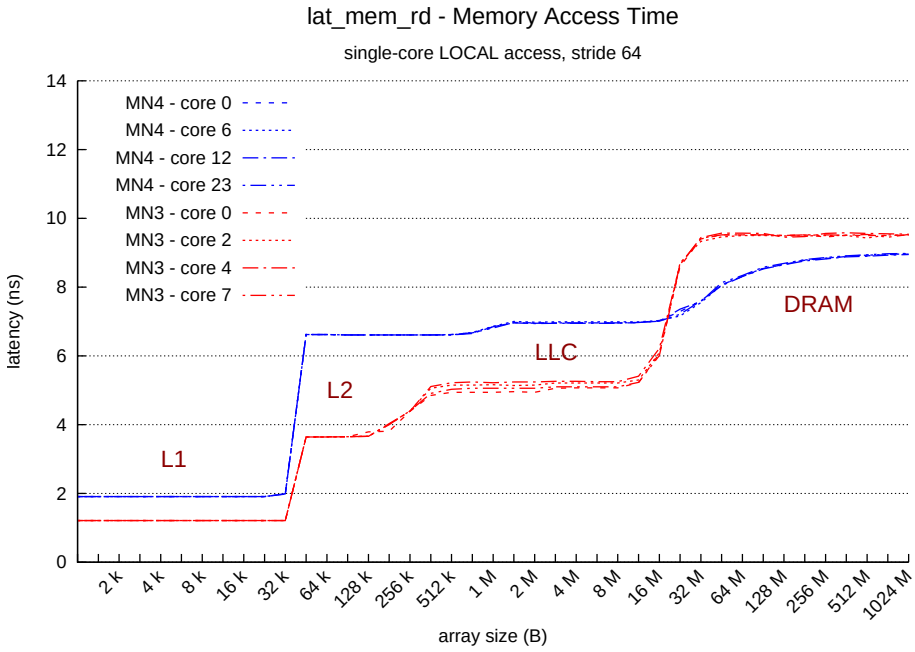


Figura 8.1: Latencia de memoria en acceso secuencial de un solo core

```
  272999.987160   task-clock (msec)   #    1.000   CPUs utilized
571,946,432,677   cycles              #    2.095   GHz
 83,427,738,607   instructions        #     0.15   insns per cycle
 21,028,350,140   cache-references    #   77.027   M/sec
 11,051,551,083   cache-misses        #   52.555   % of all cache refs

  273,028456700 seconds time elapsed
```

(a) MareNostrum 4

```
  277134.217129   task-clock          #    0.997   CPUs utilized
915,935,292,935   cycles              #    3.305   GHz
114,476,516,202   instructions        #     0.12   insns per cycle
  9,123,925,779   cache-references    #   32.922   M/sec
  2,938,639,015   cache-misses        #   32.208   % of all cache refs

  277.945492801 seconds time elapsed
```

(b) MareNostrum 3

Figura 8.2: Estadísticas de lat_mem_rd en acceso secuencial recopiladas con perf stat

Las primeras valoraciones de este primer test a partir de las estadísticas y cálculos previos podrían ser:

1. La tasa de fallos de caché por instrucción en MarenNostrum 4 es 4,33 veces peor.
2. MarenNostrum 4 realiza el mismo trabajo con un 27 % menos de instrucciones.
3. MarenNostrum 3 experimenta casi un 40 % menos de fallos de caché.

Avanzando un poco más, la latencia de los niveles de caché L1 y L2 va ligada al reloj del core. Es decir, la frecuencia juega un papel fundamental en la rapidez con la que se accede a L1 y L2 en tiempo real. Dado que el número de ciclos de reloj (core clocks) permanece igual independientemente de la frecuencia, y que los valores de frecuencia medidos en cada procesador son dispares, conviene hacer la conversión de nanosegundos a ciclos de reloj. Esto es tan fácil como multiplicar la latencia que nos da el benchmark lat_mem_rd por la frecuencia del procesador que hemos medido con la herramienta perf stat.

En la Tabla 8.1 hacemos esa conversión a partir de las latencias medidas con los tamaños de array 2 KB y 64 KB, que son representativos de los dos niveles de caché que ahora interesan. Notemos que los números resultantes coinciden plenamente con los valores obtenidos desde la bibliografía (mostrados en la Tabla 6.1).

Tabla 8.1: Conversión a ciclos de reloj de latencias de caché L1 y L2 en acceso secuencial

(a) Latencia de caché L1

	$nsec$	GHz	$cycle$
MareNostrum 4	1.910	2.095	**4.001**
MareNostrum 3	1.213	3.305	**4.009**

(b) Latencia de caché L2

	$nsec$	GHz	$cycle$
MareNostrum 4	6.616	2.095	**13.861**
MareNostrum 3	3.652	3.305	**12.070**

Además de la frecuencia y las políticas de prefetching de CPU, consideremos otros factores que pueden afectar los resultados observados en los siguientes niveles de la jerarquía de memoria:

- velocidad del bus frontal *(Front Side Bus)*
- número de cores
- velocidad, número de canales y rangos de la memoria DRAM

Centrándonos ahora en los tramos correspondientes al último nivel de caché (LLC) de la Figura 8.1, lo primero a destacar es que el incremento de la latencia en MareNostrum 4 al

saltar desde el nivel L2 es mínimo en comparación con el mismo salto en MareNostrum 3, lo que cobra bastante sentido si pensamos en la relación existente en cada caso entre las capacidades de la caché L2 y la porción de caché L3 por core, esto es 1024 KB y 1,375 MB/core en MareNostrum 4, 256 KB y 2,5 MB/core en MareNostrum 3 (véase Tabla 6.1).

La otra cosa que llama la atención es que las trazas de MareNostrum 3 en el tramo de LLC no son tan homogéneas como el resto. Si aumentamos de tamaño (zoom in) la gráfica para examinar con detalle esa región, no es difícil comprobar que la latencia más baja corresponde al core 0 y la más alta al core 4. Sin duda aquí debe jugar un papel decisivo la velocidad y la topología de la red de interconexión del chip, inclinándose la balanza hacia MareNostrum 4 en ese aspecto.

Efectos de stride y política de NUMA en la latencia

Un análisis más profundo consiste en evaluar la manera en que puede verse afectada la latencia en ambas máquinas al variar el stride y el modo en que se asigna la memoria del proceso, o sea la política de NUMA. Para ello se ha repetido el experimento anterior iterando sobre un array de strides y aplicando las siguientes configuraciones mediante numactl:

```
numactl --membind=0 --physcpubind=0  # local access

numactl --membind=1 --physcpubind=0  # remote access

numactl --interleave=all --physcpubind=0  # interleave access
```

La manera de comprobar que realmente se producen los 3 escenarios deseados —por ejemplo durante la ejecución de un job en MareNostrum 4 (SLRUM)— es siguiendo estos pasos:

1. Ejecutar `squeue` para averiguar el host en el que está corriendo el job
2. Saltar al host vía `ssh`
3. Ejecutar `top` en el host para obtener el PID del proceso
4. Ejecutar `numastat -p`, pasando como parámetro el PID del proceso

En la Figura 8.3 se muestra la salida generada al usar este procedimiento para cada una de las variantes.

En la Figura 8.4 se comparan las latencias medidas para cada combinación y máquina, fijando la misma escala XY de todos los gráficos con tal de facilitar una comparación más

intuitiva. En general, para todas las combinaciones de parámetros aplicadas se observan unos patrones de comportamiento muy similares en ambas máquinas. Sin embargo, el hecho de que las curvas que dibujan las transiciones de LLC a DRAM en MareNostrum 4 sean más suaves y menos empinadas que en MareNostrum 3 puede deberse a varios factores ya comentados, en particular a una política de prefetching más evolucionada. También se aprecian algunas situaciones "curiosas" en que no siempre un stride más grande conlleva a una mayor latencia, como por ejemplo lo que ocurre con los valores 192 y 256, que se comportan de forma inversa en una y otra máquina.

```
Per-node process memory usage (in MBs) for PID 366361
(lat_mem_rd)
                       Node 0          Node 1           Total
                  -------------   -------------   -------------
Huge                      0.00            0.00            0.00
Heap                      0.07            0.00            0.07
Stack                     0.02            0.00            0.02
Private                1026.24            0.62         1026.86
-------------     -------------   -------------   -------------
Total                  1026.33            0.62         1026.95
```

(a) local

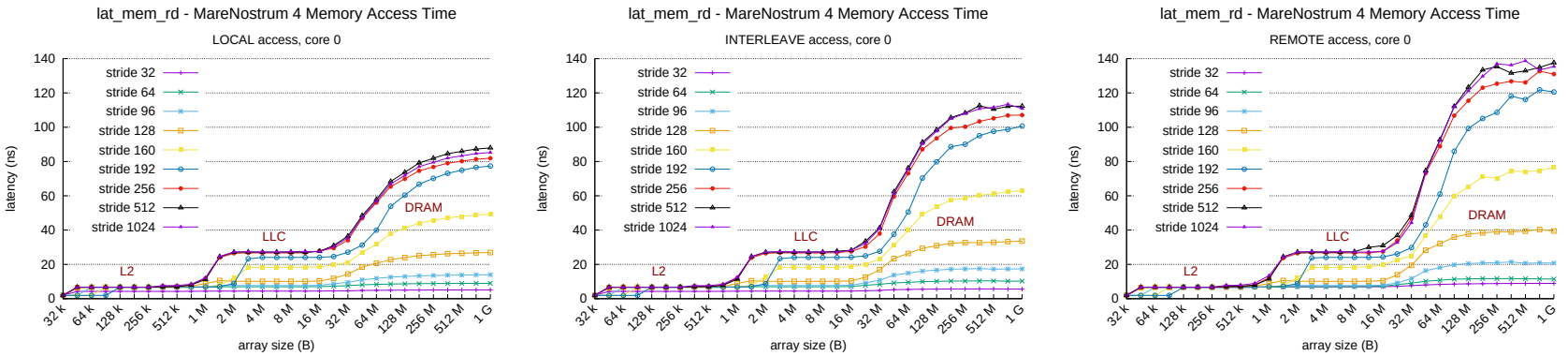
```
Per-node process memory usage (in MBs) for PID 366597
(lat_mem_rd)
                       Node 0          Node 1           Total
                  -------------   -------------   -------------
Huge                      0.00            0.00            0.00
Heap                      0.00            0.07            0.07
Stack                     0.00            0.02            0.02
Private                   0.16         1026.71         1026.87
-------------     -------------   -------------   -------------
Total                     0.16         1026.80         1026.96
```

(b) remote

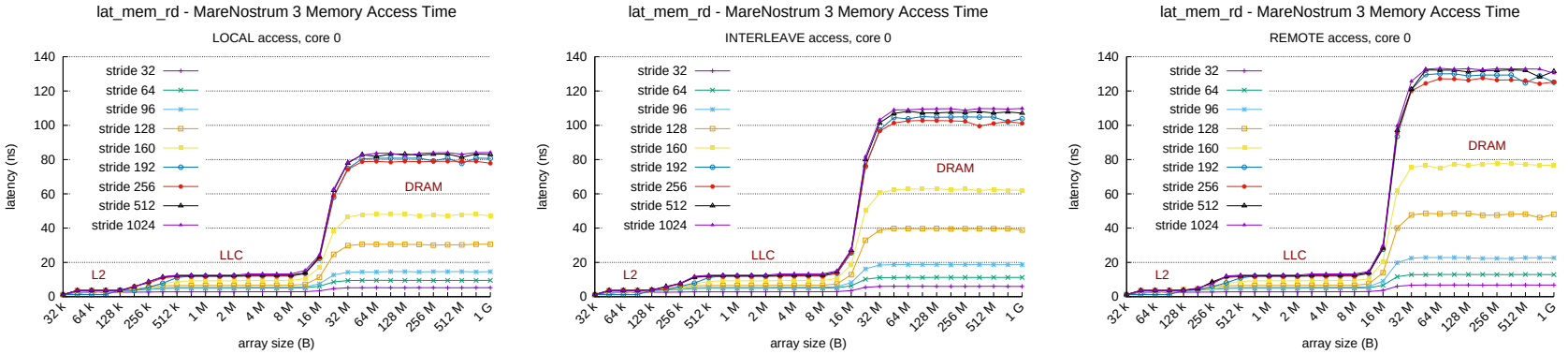
```
Per-node process memory usage (in MBs) for PID 367222
(lat_mem_rd)
                       Node 0          Node 1           Total
                  -------------   -------------   -------------
Huge                      0.00            0.00            0.00
Heap                      0.03            0.04            0.07
Stack                     0.01            0.01            0.02
Private                 514.18          512.70         1026.88
-------------     -------------   -------------   -------------
Total                   514.22          512.74         1026.96
```

(c) interleave

Figura 8.3: Observación del uso de memoria según la política de NUMA



(a) MareNostrum 4



(b) MareNostrum 3

Figura 8.4: Efectos de *stride* y *política de NUMA* en la latencia de memoria

Acceso en paralelo de todos los cores de un socket

Con este experimento se quiere evaluar cómo se ve afectada la latencia cuando todos los cores de un socket acceden concurrentemente a la memoria local. Este escenario se consigue una vez más con ayuda de numactl, añadiendo además el parámetro "`-P 24`" para indicar a lat_mem_rd el número de procesos que se quiere lanzar en paralelo:

```
numactl --membind=0 --cpunodebind=0 lat_mem_rd -P 24 -N 50 1536
64
```

Los resultados se muestran en la Figura 8.5, donde se comparan con las mediciones realizadas en el primer test sobre un solo core. Las estadísticas de perf correspondientes a esta gráfica son las mostradas en la Figura 8.6.

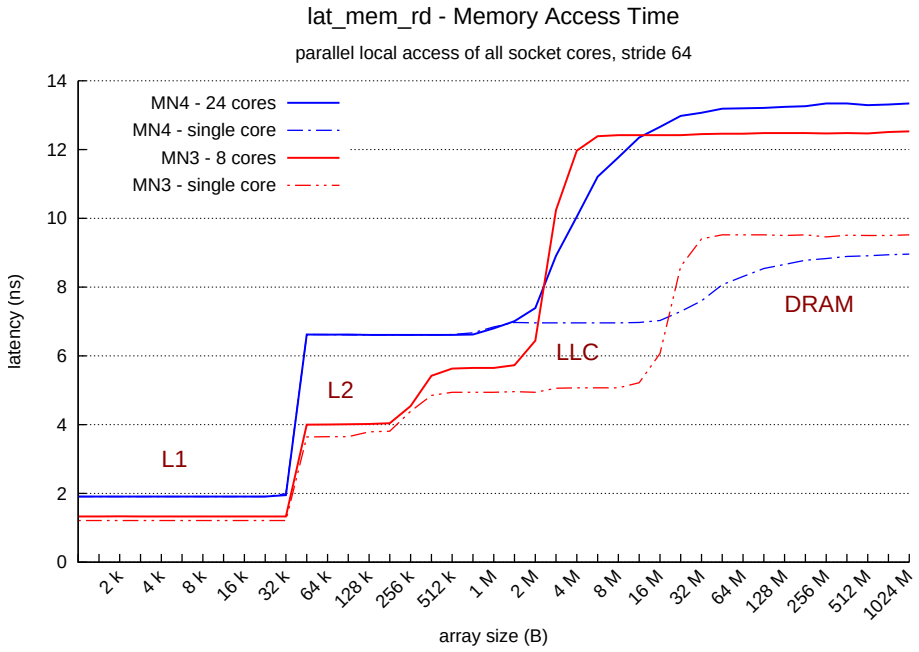


Figura 8.5: Latencia de memoria en acceso paralelo de todos los cores de un socket

Repitiendo los cálculos del apartado anterior para analizar la proporción fallos de caché por instrucción, ahora tenemos:

$$CMPI_{MN4} = \frac{2{,}90 \times 10^{12}\ cache\text{-}miss}{13{,}54 \times 10^{12}\ instruction} = \mathbf{0{,}21}\ cache\text{-}miss/instruction$$

$$CMPI_{MN3} = \frac{0{,}14 \times 10^{12}\ cache\text{-}miss}{6{,}63 \times 10^{12}\ instruction} = \mathbf{0{,}02}\ cache\text{-}miss/instruction$$

```
    70352293.970835  task-clock (msec)   #   22.019  CPUs utilized
147,392,202,432,089  cycles              #    2.095  GHz
 13,539,080,888,498  instructions        #     0.09  insns per cycle
  3,541,875,351,056  cache-references    #   50.345  M/sec
  2,895,566,844,608  cache-misses        #   81.752   % of all cache refs

     3195.110630611 seconds time elapsed
```

(a) MareNostrum 4

```
    24280525.020719  task-clock         #    7.381  CPUs utilized
 73,081,357,672,146  cycles             #    3.010  GHz
  6,626,767,379,982  instructions       #     0.09  insns per cycle
    513,090,347,665  cache-references   #   21.132  M/sec
    145,268,184,258  cache-misses       #   28.312   % of all cache refs

     3289.486485445 seconds time elapsed
```

(b) MareNostrum 3

Figura 8.6: Estadísticas de lat_mem_rd en acceso paralelo recopiladas con perf stat

En este nuevo escenario, la proporción fallos de caché por instrucción de MareNostrum 4 aumenta un 62 % respecto al ratio anterior, mientras que la de MareNostrum 3 se reduce un 33 %, sin perder de vista que ahora no compiten en igualdad de condiciones en cuanto al número de procesos (cores) que corren en paralelo.

En la nueva gráfica (Figura 8.6) se aprecia un ligero incremento de la latencia de MareNostrum 3 en los accesos a L1 y L2, no ocurriendo lo mismo en MareNostrum 4. Parece estar claro que esta diferencia de comportamiento se debe a que MareNostrum 3 reduce la frecuencia al usar todos los cores del socket. En cambio en MareNostrum 4 se mantiene exactamente el mismo valor medido anteriormente, lo que significa que hasta el momento no se ha observado el escalado de frecuencia esperado, acorde con el número de cores utilizados.

Si comparamos estos resultados rehaciendo la conversión de las latencias medidas en nanosegundos a ciclos de reloj (Tabla 8.2), vemos que el número de ciclos en los accesos a L1 y L2 permanece prácticamente intacto en ambas máquinas, dejando claro que el incremento de la latencia experimentado por MareNostrum 3 está vinculado a la disminución de la frecuencia antes comentada.

Tabla 8.2: Conversión a ciclos de reloj de latencias de caché L1 y L2 en acceso paralelo

(a) Latencia de caché L1

	$nsec$	GHz	$cycle$
MareNostrum 4	1.910	2.095	**4.001**
MareNostrum 3	1.334	3.010	**4.015**

(b) Latencia de caché L2

	$nsec$	GHz	$cycle$
MareNostrum 4	6.617	2.095	**13.863**
MareNostrum 3	4.005	3.010	**12.055**

Pasando al último nivel de caché, no parece casual que se replique el mismo comportamiento en forma de un incremento considerable de la latencia en el segmento comprendido entre los tamaños de array 1,5-3 MB. Como ahora los cores no disponen de toda la memoria caché de último nivel en exclusiva sino que es compartida, obviamente comienza a haber congestión desde el momento en que el tamaño de array sobrepasa la capacidad de la porción de caché LLC local.

8.1.2. Ancho de banda

La política de NUMA por defecto consigue un buen equilibrio en muchos entornos informáticos generales, ya que proporciona un buen soporte genérico para las topologías de memoria NUMA sin que tenga que intervenir manualmente el usuario. Sin embargo el ancho de banda máximo sólo puede alcanzarse si se emplean ciertas optimizaciones, entre las que destaca precisamente la asignación y uso de NUMA [43, 76].

Cabe que añadir que a pesar de que *Sub-NUMA clustering* es el modo de operación recomendado para las aplicaciones del tipo *NUMA-aware*, es decir, aplicaciones que fijan los hilos de procesamiento y su memoria a los nodos NUMA respectivos, el desarrollador debe hacer que la aplicación sea compatible con NUMA para aprovechar esta arquitectura [77].

Todas estas observaciones se relacionan también con la cuestión de la "afinidad de threads", o afinidad de CPU como ya hemos venido llamándole, un tema que se vuelve importante en los nodos multisocket [78].

Efectos de la afinidad de CPU en el ancho banda de memoria

El objetivo de esta última parte de la sección es mirar exactamente qué influencia puede tener la afinidad de threads en el rendimiento en ancho de banda de las dos arquitecturas. No perdamos de vista que al final lo que se quiere averiguar es el máximo ancho de banda que se puede conseguir y de qué manera podemos conseguirlo.

Para las mediciones de ancho de banda de la memoria usaremos el clásico **STREAM**, utilizando 2 binarios distintos, uno de ellos generado con el compilador `icc` (véase versiones de software, apartado), el otro mediante la versión 7.2.0 del compilador `gcc`.

Los binarios se han generado ejecutando los comandos

```
icc -O -fopenmp -mcmodel medium -shared-intel
-DSTREAM_ARRAY_SIZE=96000000 -DNTIMES=500 stream.c -o stream_96M

gcc -O -fopenmp -DSTREAM_ARRAY_SIZE=96000000 -DNTIMES=500
```

```
stream.c -o stream_96M_gcc
```

Mediante las opciones `-DSTREAM_ARRAY_SIZE=96000000` y `-DNTIMES=500` se puede sobrescribir el código y ajustar en tiempo de compilación el tamaño del array así como el número de repeticiones que queremos ejecutar cada kernel. El objetivo es, por una parte, hacer que el conjunto de datos sea mucho más grande que el tamaño total de la cache L3 de los procesadores, y por otra, mantener el flujo de STREAM durante más tiempo de forma que nos brinde suficiente margen de cara a la monitorización en tiempo de ejecución.

Las opciones elegidas para experimentar con la afinidad de threads son, `close` y `spread` de OpenMP, junto con `compact` y `scatter`, que son específicas de Intel. A pesar de que el binario generado con el compilador de Intel debería reconocer las variables de OpenMP, se pudo comprobar que no funcionaba correctamente, de ahí viene que se hayan generado dos ejecutables.

Básicamente estas cuatro variantes se diferencian en que `close` y `compact` intentan siempre llenar todo un socket (nodo NUMA) antes de pasar a usar el otro, mientras que `spread` y `scatter` tienen el comportamiento opuesto, aunque cada una de estas dos últimas hace la distribución equitativa de threads entre los sockets de forma muy diferente en función de los identificadores lógicos de los threads/cores.

Para comprobar que las distintas variantes de afinidad de procesos se comportan de la forma esperada, se ha modificado ligeramente un pequeño código existente del estilo "Hello World" en el que cada proceso escribe su información relevante (identificador lógico de thread OpenMP, core y nodo NUMA donde está corriendo).

En el caso particular del compilador de Intel, éste ya cuenta con sus propias variables de entorno para tal propósito, así que nos hemos aprovechado de la opción `verbose` lo cual nos ha facilitado bastante esta tarea, mostrándose como ejemplo la captura de pantalla de la Figura 8.7.

Figura 8.7: Comprobación de la afinidad de threads mediante KMP_AFFINITY (Intel)

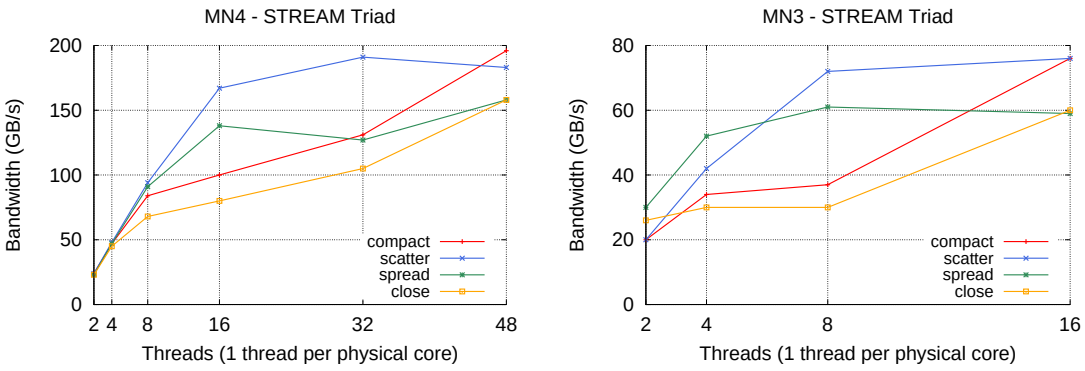
La Figura 8.8 muestra los patrones resultantes al ejecutar `STREAM Triad` en un nodo de cada máquina, variando el número de threads por nodo y usando las cuatro

configuraciones antes descritas. Los resultados no pueden ser más contundentes sobre el efecto que puede causar la afinidad de threads/CPU junto con otros factores como son el compilador, las librerías OpenMP o el número de procesos/cores utilizados.

En primer lugar hay que subrayar que los patrones manifestados al ejecutar cada una de las tres operaciones restantes —Copy, Scale y Add, que ya no mostramos— son casi calcados, lo que corrobora que al ejecutar STREAM en hardware actual, cada una de estas pruebas logra un ancho de banda similar debido a que la principal restricción recae en la memoria y no en la ejecución de coma flotante [79, p. 5]. Esta peculiaridad es fácilmente apreciable en la Figura 7-8 (a) - (d), que se explicará un poco más adelante.

Del igual modo, si comparamos patrón a patrón entre máquinas —que es en verdad de lo que más va este trabajo—, observamos también gran similitud entre miembros de cada "pareja".

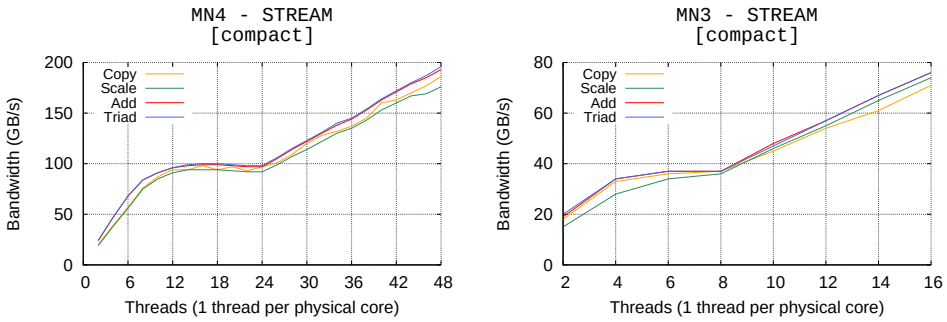
Y pasando a las cifras puras y duras, el dato concluyente es que MareNostrum 4 lógicamente gana la comparación de forma aplastante, cosa que no es nada extraña teniendo en cuenta todas las mejoras que habido en la evolución de la arquitectura y que impactan directamente en las prestaciones de ancho de banda, todas ellas descritas con bastante detalle en el capítulo correspondiente.



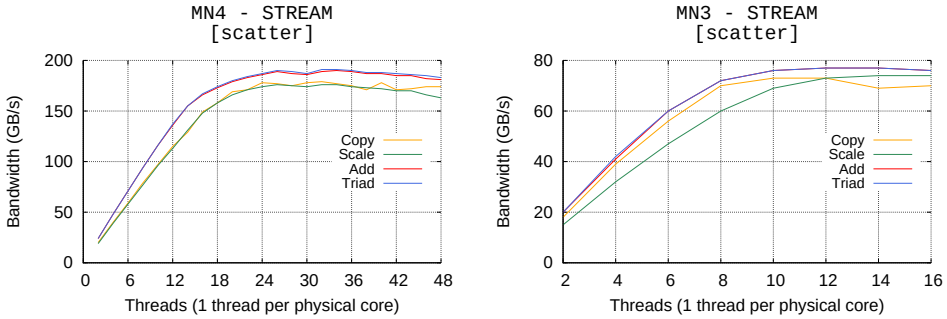
(a) MareNostrum 4 (b) MareNostrum 3

Figura 8.8: Rendimiento en ancho de banda de un nodo

Si en el experimento anterior se probaron cuatro configuraciones de afinidad de threads usando un sólo código (Triad), en el siguiente se quiere dar un paso más. Probaremos los cuatros kernels de STREAM con las configuraciones que consiguieron mayor ancho de banda en el test previo: `compact` y `scatter`.



(a) compact thread affinity



(b) scatter thread affinity

Figura 8.9: Rendimiento en ancho de banda de socket y nodo

Los resultados conseguidos se pueden ver en la Figura 8.9. Si nos fijamos en las dos gráficas de las ejecuciones usando la afinidad `compact`, se ve claramente la penalización de rendimiento en el intervalo de valores que va desde 2 hasta justamente el número máximo de núcleos por socket (nodo NUMA) en cada máquina, 8 y 24 respectivamente. Por el contrario, en las ejecuciones con `scatter` el comportamiento es bien distinto habiendo alcanzado en dichos "puntos frontera" prácticamente el total del ancho de banda máximo en ambos casos.

Esto acredita que si bien el uso de memoria local en un nodo NUMA garantiza la mejor latencia, se puede obtener más ancho de banda usando en paralelo los controladores de memoria de múltiples nodos NUMA, de manera análoga en que RAID puede mejorar el rendimiento de I/O del disco expandiendo las operaciones I/O en múltiples discos [70, p. 2].

Concretamente en estos dos escenarios (léase MareNostrum 3 y 4) y en base a esa premisa, se ha conseguido doblar el ancho de banda utilizando el mismo número de cores (8 y 24 cores respectivamente), simplemente optimizando la afinidad de procesos.

8.2. Evaluación de rendimiento de las unidades de cálculo

En este apartado se mide el rendimiento en GFLOPS mediante LINPACK y HPCG, dos benchmarks de máxima referencia en el área de la supercomputación. Los binarios tanto de LINPACK como de HPCG son los únicos que no han sido compilados expresamente para la realización de este proyecto, utilizando en vez de ello binarios de uso interno del BSC proporcionados por Intel.

Estos ejecutables están "tuneados" —altamente optimizados— para las máquinas del centro con el objetivo de obtener los mejores resultados, entiéndase lo más cerca posible al peak teórico, muy enfocado por ejemplo a las mediciones destinadas al ranking TOP500.

8.2.1. Rendimiento en un core

El único fichero necesario para correr LINPACK es `HPL.dat`. Para configurarlo nos hemos apoyado en [80], definiendo los parámetros más relevantes de la forma siguiente:

	MN3	**MN4**
Ns	40960	40960
NBs	256	384
Ps	1	1
Qs	1	1

En cuanto a HPCG, se puede configurar directamente desde la línea de comandos. Para esta medición en concreto se han usado los parámetros `–n160 –t2100` que son el tamaño de matriz y el tiempo máximo que queremos mantener la ejecución del benchmark*(wall time)*, respectivamente. Esto quiere decir que a diferencia de LINPACK, a HPCG hay que "matarlo", con lo cual no procede hacer comparaciones de tiempos de ejecución.

Observando en la Figura 8.10 los resultados obtenidos con LINPACK, no dejan ninguna duda de la superioridad del Xeon 8160 (MN4), que consigue una mejora de 2,5x. Por el contrario, con HPCG los resultados son bastante más equilibrados y esta mejora de MN4 sobre MN3 se queda en un 32 %, que tampoco es una cifra despreciable.

A partir de las métricas recolectadas con perf (véase Figura 8.11), se puede intentar dar una explicación a estos resultados. En primer lugar, se observa otra vez en la gráfica "CPU

Clock Frequencies" que el core de MN4 no escala la frecuencia turbo a su máximo teórico en ninguna de las dos pruebas. Aquí aprovechamos para avisar que este comportamiento se mantendrá ya patente durante el resto de las pruebas de este proyecto.

Bastante más sentido tiene la gráfica "Cache-references" donde se observan proporciones muy similares, es decir, en ambos cores HPCG supera aproximadamente en 5 veces el número de accesos a caché con respecto a LINPACK. Esto cuadra también con que HPCG es una aplicación de tipo memory bound mientras que LINPACK es compute bound, pudiendo ser la principal causa de los resultados vistos.

No obstante si nos fijamos en la tasa de fallos de caché (gráfica "Cache-misses"), también es de esperar que, a mayor número de accesos, MN4 pierda por goleada, manteniendo un comportamiento similar al que ya se vio durante la evaluación de la jerarquía de memoria. Este aumento de fallos de caché en HPCG tiene un impacto directo en la reducción significativa del IPC que se ve reflejado en la gráfica correspondiente.

En cuanto a las estadísticas de instrucciones de salto —número de instrucciones y eficiencia de predicción—, también se observa gran disparidad en las métricas de HPCG. Por una parte MN3 ejecuta un número de instrucciones de salto muy superior, por otra MN4 falla en la predicción de saltos con una diferencia en porcentaje bastante cercano a la diferencia en número de instrucciones. Con estos datos no se ve demasiado claro sacar alguna conclusión fiable, salvo que este comportamiento en la predicción de saltos en HPCG perjudica también al IPC de MN4 mientras que beneficia al de MN3, contribuyendo seguramente al mayor equilibrio de GFLOPS resultantes.

En el sentido opuesto, MN4 se beneficia enormemente cuando no está limitado por memoria sino por la potencia de cálculo (LINPACK), y ahí sí pueden verse los beneficios de la vectorización.

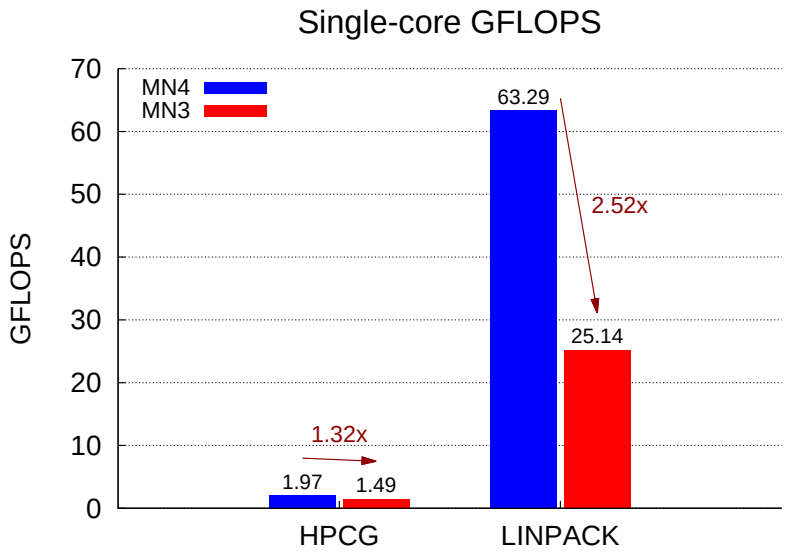


Figura 8.10: Rendimiento en GFLOPS de un core

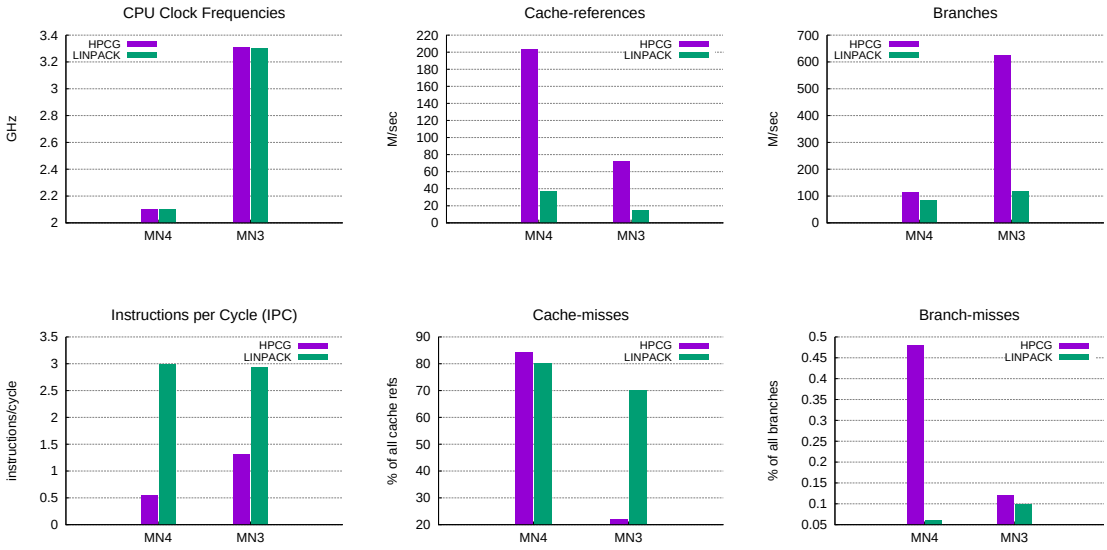


Figura 8.11: Estadísticas de LINPACK y HPCG al correr en un core (perf stat)

8.2.2. Rendimiento en un nodo

Para la evaluación de rendimiento en un nodo se ha utilizado igual metodología que en el apartado anterior. En este caso los parámetros establecidos mediante `HPL.dat` para correr LINPACK en un nodo son los siguientes:

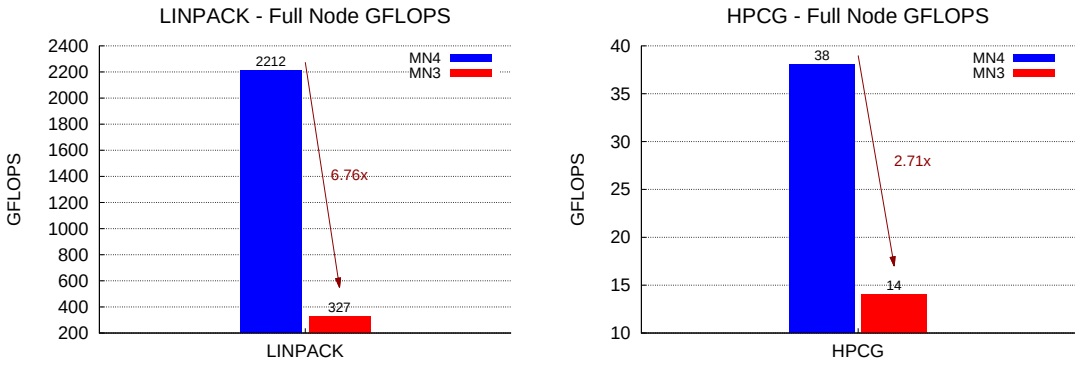
	MN3	MN4
Ns	99840	99840
NBs	256	384
Ps	1	1
Qs	1	1

La configuración híbrida OpenMP+MPI que nos ofrece los mejores resultados al correr LINPACK sobre un nodo en ambas máquinas ha sido la de un único proceso MPI y tantos threads como el número de cores disponibles en cada nodo, es decir 16 para MN3 y 48 para MN4. En el caso de HPCG, la mejor configuración ha sido la de 2 procesos MPI x 8 threads para MN3, 2 procesos MPI x 24 threads para MN4.

En esta comparativa ya entran en juego otros "actores" como el ancho de banda y la latencia de los accesos compartidos a memoria, o la intercomunicación entre procesos, sin obviar el papel más determinante que asume la vectorización al haber ahora una diferencia importante tanto en número de cores como en número de operaciones en paralelo mediante instrucciones SIMD.

En la Figura 8.12a vemos que todo junto hace aumentar el speedup de MN4 en LINPACK hasta casi un 6,8x. Una cosa que interesa mirar también es cuánto aporta cada core en el rendimiento global del nodo —se divide el resultado por el número de cores—, en lo que hay que destacar dos cosas. La primera, que en ese punto también gana MN4 con una aportación por core de más del doble con respecto a lo que aporta cada core en MN3. La segunda nos da una medida de la pérdida de rendimiento con respecto a la ejecución en un solo core, concretamente de un 27 %.

En cuanto a HPCG, en la Figura 8.12b se puede apreciar que aunque la mejora de MN4 respecto a MN3 sigue estando lejos de la conseguida con LINPACK, hay mayor diferencia que antes, casi seguro debido a que la parte de la ganancia que aporta la vectorización comienza a cobrar más peso.



(a) (b)

Figura 8.12: Rendimiento en GFLOPS de un nodo

8.2.3. Rendimiento en un "rack"

Para concluir esta parte se pone en escena al último "actor", no por ello menos determinante: la red de interconexión. Ante la imposibilidad de ejecutar en todo un rack de MareNostrum 3 (Nord3) —debido a la falta de capacidad de la configuración actual de la máquina—, hemos reajustado esta prueba lanzando los benchmarks sobre 12 nodos, escogiendo a conciencia este número entre otros motivos para que sea sencillo extrapolar los resultados al número total de nodos en cada rack, 72 para MN4, 84 para MN3.

Como en verdad queremos conseguir el máximo posible de GFLOPS, se ha impuesto una restricción a nivel de switches para correr el benchmark sobre 12 nodos de un mismo nivel —un único salto para todas las comunicacies entre 2 nodos cualesquiera—, intentando de esta manera minimizar el impacto de la red. El fichero de entrada `HPL.dat` se ha configurado con los parámetros siguientes:

	MN3	**MN4**
Ns	347136	347136
NBs	256	384
Ps	3	3
Qs	4	4

Centrándonos en LINPACK —llegados a este punto sabemos que HPCG seguirá estando limitado por memoria—, los resultados de la Figura 8.13 nos revelan esta vez un speedup en MN4 de 6.6x, que es ligeramente inferior al speedup alcanzado al correr en un nodo. Sin embargo, lo más relevante es que en este escenario la aportación de cada core a la capacidad de cómputo global se mantiene casi intacta con respecto a la ejecución en

un nodo, con una muy baja penalización de alrededor de un 5 % por parte de la red de interconexión y las comunicaciones internodo en general. Por lo tanto estas son muy pero que muy buenas noticias.

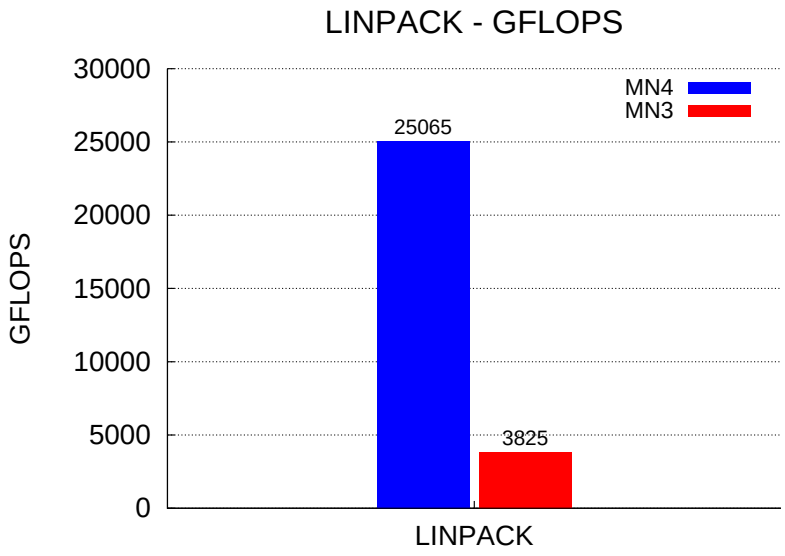


Figura 8.13: Rendimiento en GFLOPS de 12 nodos

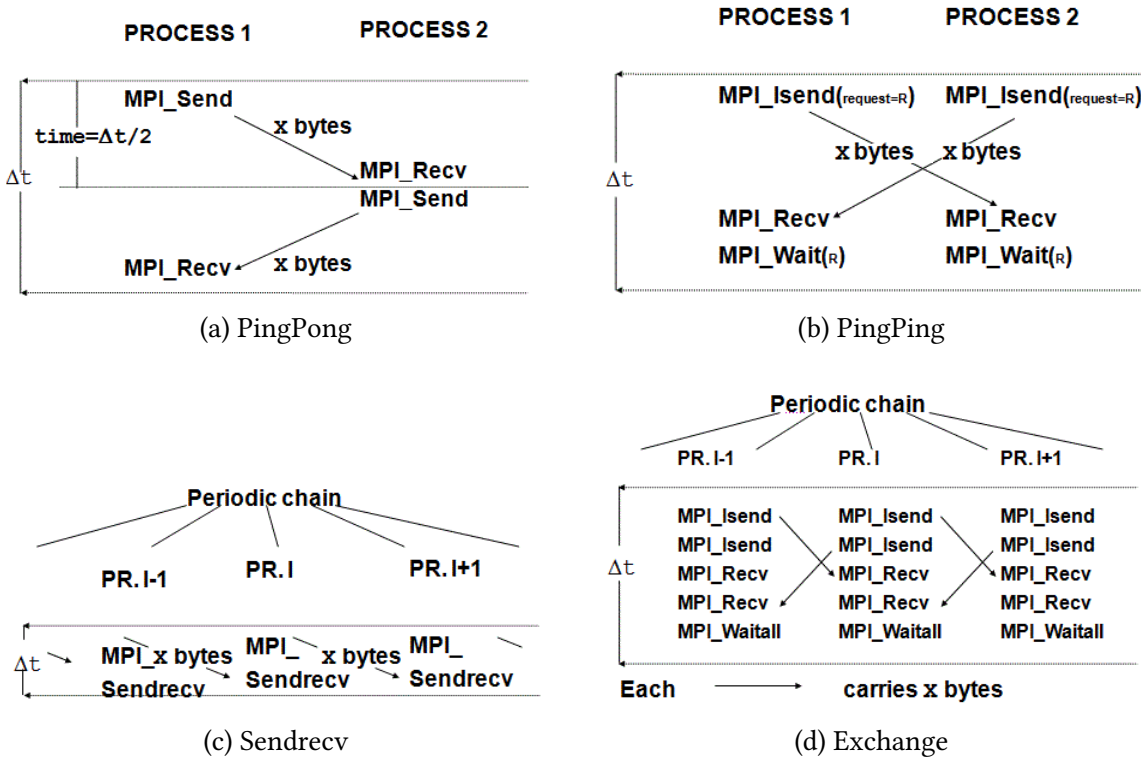
8.3. Evaluación de rendimiento de la red de interconexión e implementación MPI

Los benchmarks usados en todas las pruebas de esta parte del análisis son los de la suite Intel MPI Benchmarks.

8.3.1. Rendimiento en comunicaciones punto a punto internodo

En este apartado se escogen cuatro microbenchmarks de la suite Intel MPI Benchmark a fin de evaluar el rendimiento de las comunicaciones entre nodos al ejecutar operaciones del tipo punto a punto. Las operaciones punto a punto de MPI involucran el paso de mensajes entre dos y sólo dos tareas MPI diferentes. Una de las tareas realiza una operación de envío, la otra tarea realiza la operación de recepción correspondiente [81]. Los cuatro benchmarks seleccionados junto con los patrones de comunicación asociados se representan gráficamente en la Figura 8.14.

Los benchmarks se han lanzado sobre 6 nodos, asignando 2 procesos por nodo, 1 proceso por socket, cada proceso fijado al core 0 en su socket correspondiente. La idea es



(a) PingPong (b) PingPing

(c) Sendrecv (d) Exchange

Figura 8.14: Patrones de comunicación de benchmarks de operaciones punto a punto

conseguir que la mitad de las comunicaciones se produzcan entre cores ubicados en el socket 0, que es donde está conectada la interfaz de red, y la otra mitad entre cores ubicados en el socket que no está conectado directamente a la red.

Intel MPI Benchmark proporciona muchas opciones para gestionar su flujo de control. Para conseguir el escenario deseado hemos utilizado además un sencillo script con el cual fijar los procesos a los cores deseados, en combinación con la opción de mapeo de procesos que ofrece el benchmark. Otro de los parámetros tenidos en cuenta es el de invalidación de caché, para evitar la reutilización de la caché y obtener así un escenario lo más desfavorable posible. Como última observación, los benchmarks se han lanzado formando varios grupos de la manera que se ilustra a continuación:

```
# Benchmarking Multi-PingPing
# ( 6 groups of 2 processes each running simultaneous )
# Group 0: 0 2
#
# Group 1: 1 3
#
# Group 2: 4 6
#
# Group 3: 5 7
#
```

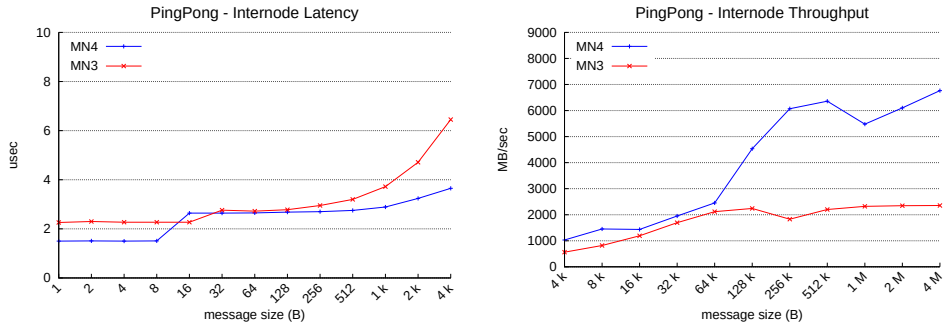
```
# Group 4: 8 10
#
# Group 5: 9 11

# Benchmarking Multi-Sendrecv
# ( 3 groups of 4 processes each running simultaneous )
# Group 0: 0 2 4 6
#
# Group 1: 5 7 9 11
#
# Group 2: 8 10 1 3
```

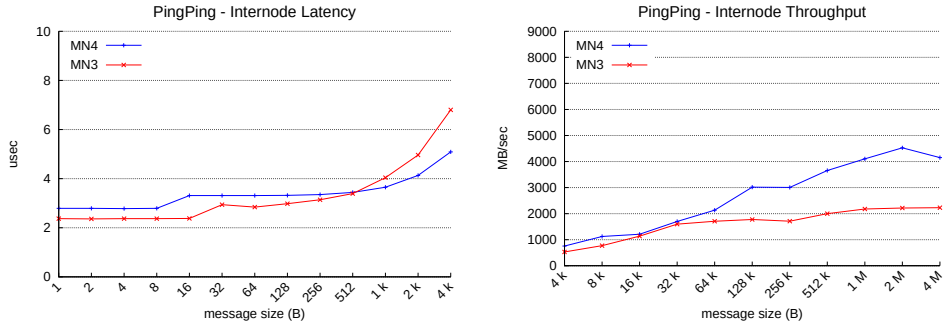
Con arreglo al criterio seguido en [47], creemos que es un total acierto dividir las gráficas en dos partes, debido a que la latencia resulta vital para tamaños de mensaje pequeños, pero para tamaños de mensaje grandes es más importante el ancho de banda. Los resultados son los mostrados en la Figura 8.15.

En líneas generales y no entrando en detalles, en cada par de gráficas asociadas se observa un patrón bastante uniforme, especialmente en las gráficas donde se representa el rendimiento en términos de ancho de banda. En las tres primeras gráficas de latencias, las métricas son bastante parejas aunque en todas ellas la tendencia de MN3 es ir a peor con respecto a MN4 a medida que se va acercando al punto de corte. En la parte contrapuesta, las gráficas de ancho de banda muestran con toda claridad como en MN4 el rendimiento va en aumento más o menos progresivamente.

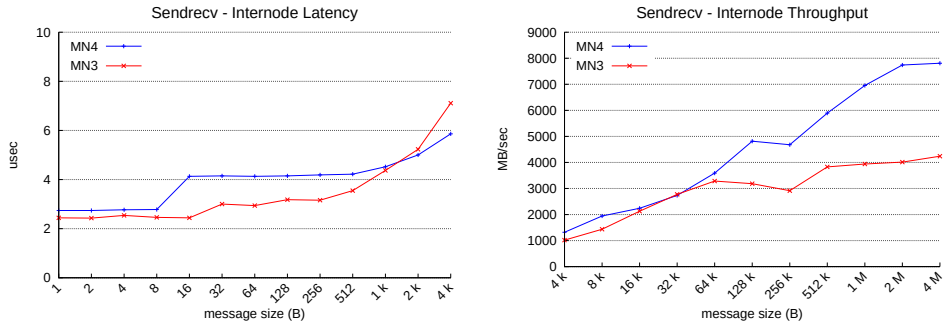
También son buenas noticias pues de momento se confirma que la red de MareNostrum 4 es bastante mejor que la de Marenostrum 3.



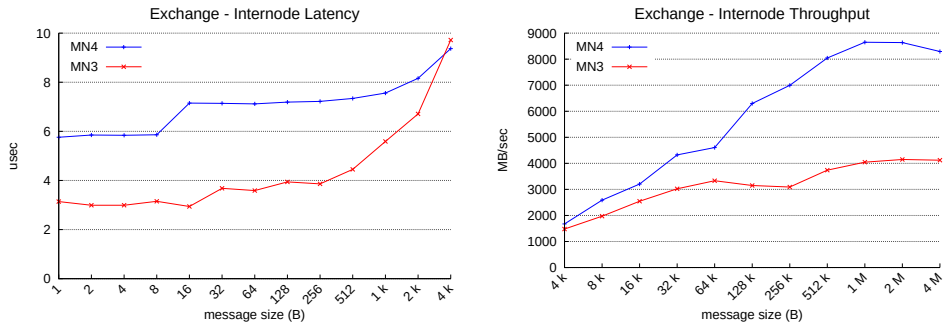
(a) PingPong



(b) PingPing



(c) Sendrecv



(d) Exchange

Figura 8.15: Rendimiento en comunicaciones punto a punto internodo

8.3.2. Rendimiento en comunicaciones colectivas internodo

Una peculiaridad de las operaciones colectivas es que todos los participantes invocan la misma función. Hay que mencionar también que aquí no sólo es relevante la capacidad de paso de mensajes del sistema, sino también la calidad de la implementación. En la Figura se ilustran los patrones de comunicación de las operaciones colectivas más comunes en MPI.

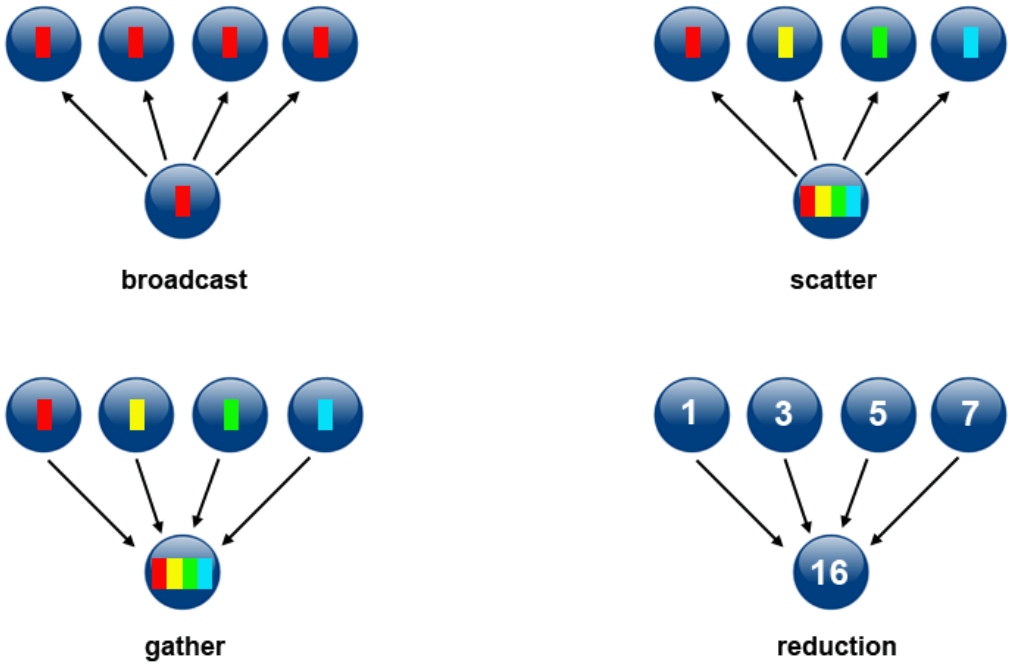
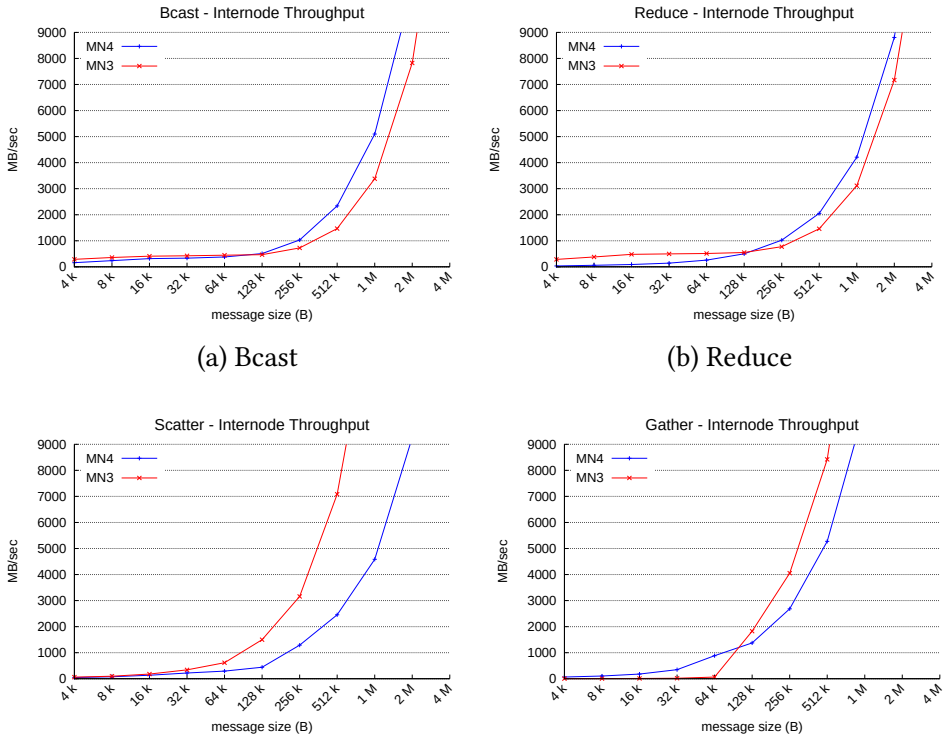


Figura 8.16: Patrones de comunicación de benchmarks de operaciones colectivas

Los microbenchmarks seleccionados para esta prueba son los siguientes: Bcast, Scatter, Gather y Reduce.

En este segundo experimento se lanzan 96 procesos MPI, esto es 2 nodos completos de MN4 frente a 6 nodos completos de MN3. Los resultados son los mostrados en la Figura 8.17.



(a) Bcast

(b) Reduce

(c) Scatter

(d) Gather

Figura 8.17: Rendimiento en comunicaciones punto a punto internodo

Capítulo 9

Análisis de aplicaciones y factores de conversión

La meta principal de este capítulo es ejecutar varias aplicaciones multinodo usando distintas configuraciones en cuanto al tamaño en número de cores, para evaluar la escalabilidad y detectar aquellos puntos que nos pueden servir para calcular factores de conversión entre ambas máquinas.

9.1. Rendimiento a nivel de core (pruebas preliminares)

La idea de realizar estas mediciones en "modo calentamiento" sobre un solo core se debe, entre otras cosas, a establecer una primera comparación (9.1) reduciendo al mínimo el nivel de paralelismo (ILP + DLP), es decir, sacar de escena a todos los actores posibles.

Aparte de eso, uno de los descubrimientos y resultados más importantes de este trabajo ha sido descubrir el problema que existe actualmente con el scaling de la frecuencia turbo de MN4, siendo justamente estas pruebas las que terminaron por ratificar esa incongruencia y mal funcionamiento, tal y como puede verse en la Figura 9.2.

Además, nos ha servido como iniciación en el aprendizaje y como banco de pruebas para irnos familiarizando con las características de cada aplicación: cómo responde una aplicación ante una determinada optimización a través del compilador, cómo muestra los resultados de ejecución (métricas), en qué ficheros los muestra, es decir, una serie de aspectos que en algunos casos pueden irse complicando a medida que aumenta el tamaño de los jobs.

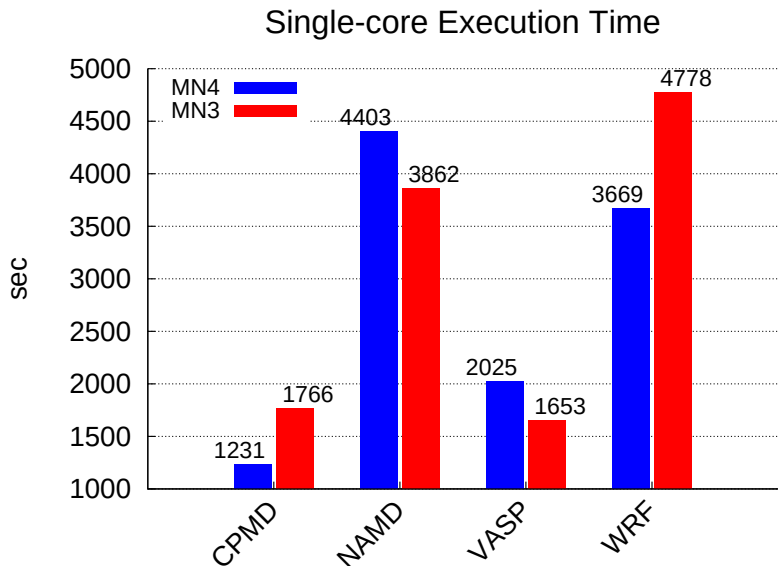


Figura 9.1: Rendimiento de aplicaciones en un core

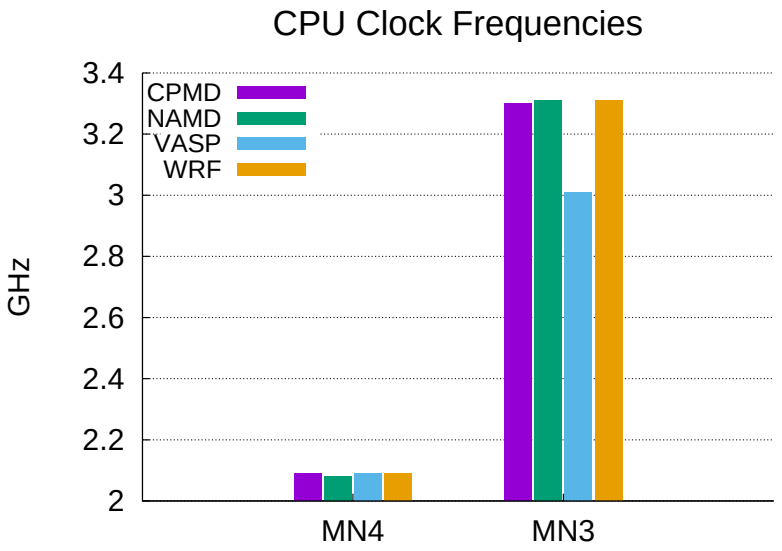
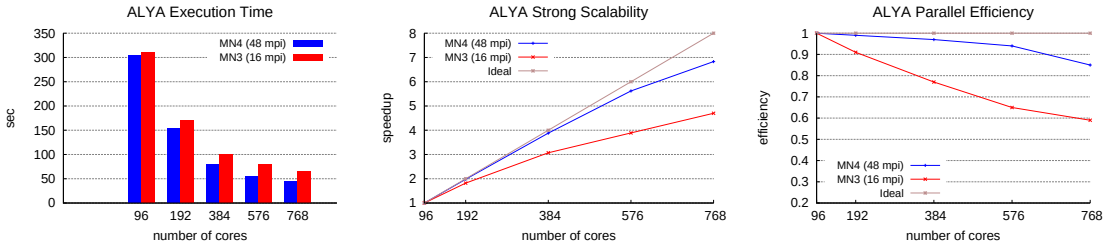


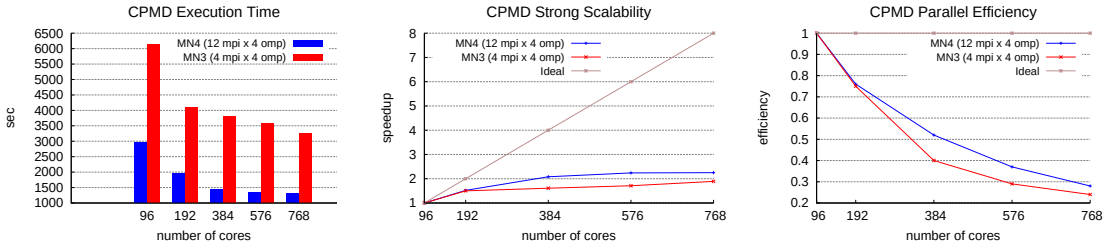
Figura 9.2: Scaling de la frecuencia de un core al ejecutar aplicaciones

9.2. Escalabilidad de aplicaciones

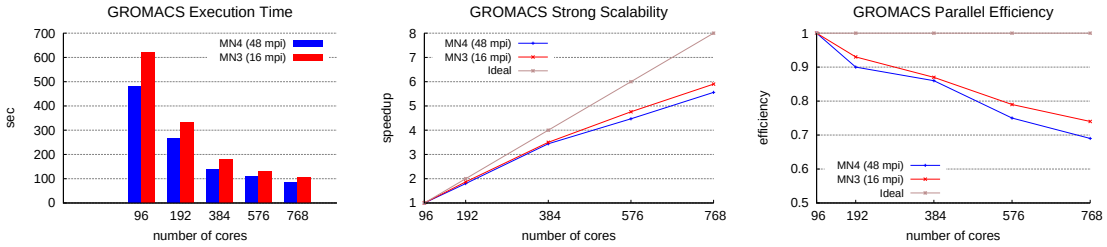
En esta sección tan solo mostramos los resultados de forma gráfica de todos los timings conseguidos a partir de las pruebas y optimizaciones realizadas durante muchas semanas, con vistas a conseguir los resultados más óptimos y poder usarlos en la siguiente sección, que es cálculo de nuestros factores de conversión.



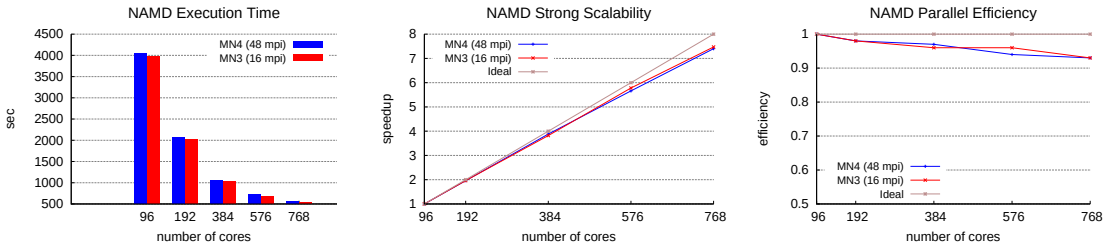
(a) ALYA



(b) CPMD



(c) GROMACS



(d) NAMD

Figura 9.3: Rendimiento y escalabilidad de aplicaciones multinodo

9.3. Factores de conversión

Existe un consenso general sobre la conveniencia de evaluar el rendimiento general utilizando mediciones de tiempo de reloj *(wallclock time)*, abriendo un debate a menudo

controvertido sobre cómo combinar las mediciones de múltiples benchmarks en un solo número. Esto es, mientras algunos apuestan por informar cada código por separado, otros por el contrario informan la media geométrica de los resultados de rendimiento del programa individual [82, p. 4] .

Sobre esta idea se han establecido puntos comunes en los que poder comparar el rendimiento de las aplicaciones escogidas. Esta metodología común intenta tener en cuenta aspectos individuales de cada aplicación, entre ellos la escalabilidad, cuestión que ya de por sí es un indicador del impacto de la mayoría de componentes que afectan al rendimiento.

Así, buscaremos los factores de conversión utilizando las métricas obtenidas en el apartado anterior, cuyas configuraciones son las siguientes:

Tabla 9.1: Configuración de factores de conversión

	Número de cores				
	96	**192**	**384**	**576**	**768**
Número de nodos MN3	6	12	24	36	48
Número de nodos MN4	2	4	8	12	16

Así mismo se ha establecido un criterio adicional que está ligado a la eficiencia del paralelismo conseguido en esos puntos, dándole más robustez a esta metodología. Concretamente, seleccionaremos las métricas en sólo aquellos puntos donde la eficiencia es mayor o igual a 0.6, para lo cual nos vienen muy bien las gráficas de la última columna a la derecha en la Figura 9.3.

A partir de los resultados de cada punto, el factor de cada aplicación lo calculamos con la media geométrica de todos ellos, tal como se muestra en la Tabla 9.2. De esta forma se consigue una métrica que tiene en cuenta un rango relativamente amplio y que además es justamente el tramo en el que se detecta el patrón correspondiente al grado de escalabilidad de la aplicación. Mediante el uso de la media geométrica (en lugar de la media aritmética) se intenta minimizar el efecto ante eventuales valores “intrusos” o extremos.

Finalmente, calculamos el factor de conversión global mediante la media aritmética, el cual nos da **1.294x**.

Tabla 9.2: Factores de conversión de aplicaciones

ALYA	**MN4**	**MN3**	**1.255**
96	304.8	309.3	1.015
192	154.0	170.2	1.105
384	78.5	100.7	1.283
576	54.3	79.6	1.466
768	44.6	65.8	1.475
CPMD	**MN4**	**MN3**	**2.082**
96	2963.9	6120.4	2.065
192	1948.9	4091.8	2.099
GROMACS	**MN4**	**MN3**	**1.266**
96	478.5	619.6	1.295
192	265.7	332.1	1.250
384	139.0	177.1	1.274
576	107.0	130.0	1.295
768	86.1	104.9	1.218
NAMD	**MN4**	**MN3**	**0.978**
96	4043.4	3968.3	0.981
192	2067.0	2025.5	0.980
384	1042.0	1037.9	0.996
576	714.3	686.4	0.961
768	546.2	531.1	0.972
VASP	**MN4**	**MN3**	**1.094**
96	76.5	85.7	1.120
192	58.9	62.9	1.068
WRF	**MN4**	**MN3**	**1.088**
96	1428.9	1625.20	1.137
192	764.9	796.3	1.041

Capítulo 10

Diseño y especificación del monitoring

10.1. Necesidades reales para la puesta en marcha del monitoring actualmente

10.1.1. BSC User Portal

Durante la realización de este proyecto se desarrolló en paralelo por miembros del equipo de Support un entorno para monitorizar los jobs por parte de los propios usuarios. Este entorno, que ya se encuentra en producción, ha acabado absorbiendo a nivel de implementación todos los requerimientos de la parte del monitoring de este proyecto, dejando solamente abierta la parte del diseño a nivel de selección de los benchmarks, métricas a recolectar y formato de fichero pactado para poder integrar esta funcionalidad en esa plataforma que está ya operativa.

10.2. Especificación del monitoring

El monitoring debe ofrecer alternativas y ser lo suficientemente flexible para adaptarse tanto a las necesidades de evaluación del rendimiento como a la disponibilidad de los recursos de la máquina **MareNostrum 4** en función de la demanda que pueda haber en cada momento. Desde esa premisa, el monitoring constará en un principio con tres tipos de pruebas bien distintas que se explicarán a continuación.

El primer test, que llamaremos **Test A**, debe consistir en uno o varios jobs sencillos con

los que poder obtener diariamente un par o tres de métricas básicas, en un breve intervalo de tiempo y ocupando muy pocos recursos.

El propósito de un segundo test, llamémosle **Test B**, sería el de hacer una evaluación más a fondo, por ejemplo semanalmante, para detectar situaciones que puedan pasarse por alto con el Test A. Para ello se necesita ejecutar un mayor número de pruebas y esto implica la utilización de más recursos en términos de hardware y horas de cómputo. En ese sentido, la idea de correr este test con una frecuencia semanal —preferiblemente en horario nocturno— responde también al requerimiento de interferir en la menor medida posible en las colas de jobs de los usuarios.

Por último, cuando se lleva a cabo un mantenimiento general que implica la parada del clúster, hay un momento "crítico" que es cuando se comprueba si la máquina funciona correctamente antes de ponerla de nuevo en explotación. Este sería el momento ideal para lanzar un tercer test o **Test C** que debe incluir una serie de pruebas a gran escala, y donde cobra suma importancia evaluar todos o la mayoría de los componentes del clúster en el menor tiempo posible, como por ejemplo la capacidad de cómputo (GLOPS), el funcionamiento óptimo de las redes, los sistemas de ficheros y los nodos de acceso (logins), de forma muy resumida.

El Test C cuenta con la gran ventaja de no tener limitaciones de recursos hardware —teóricamente podría disponer del 100 % de los recursos del clúster—, permitiendo lanzar jobs de gran tamaño en cuanto a número de cores. Sin embargo también tiene el hándicap de que debe llevarse a cabo en un intervalo de tiempo reducido —una vez finalizado el mantenimiento la máquina debe volver a producción en un corto plazo—.

10.3. Selección de benchmarks y métricas para monitoring

10.3.1. Test A (diario)

Tabla 10.1: Configuración de Test A

benchmark	nodes/job	total jobs	wall time	total nodes
HPCG	4	2	15-20 min	8
LINPACK	4	2	15-20 min	8
WRF	4	2	15-20 min	8

10.3.2. Test B (semanal)

Tabla 10.2: Configuración de Test A

benchmark	nodes/job	total jobs	wall time	total nodes
HPCG	12	4	40-60 min	48
LINPACK	12	4	40-60 min	48
ALYA	12	4	40-60 min	48
NAMD	2	4	40-60 min	8
Inteil MPI-IO	8	2	100-120 min	16

10.3.3. Test C (postmantenimiento)

Tabla 10.3: Configuración de Test A

benchmark	nodes/job	total jobs	wall time	total nodes
HPCG	1	full machine	5 min	full machine
LINPACK	1	full machine	5 min	full machine
HPCG	1000	1	5 min	1000
LINPACK	1000	1	5 min	1000
ALYA	100	12	10 min	1200
NAMD	100	12	10 min	1200
Intel MPI-IO	10	40	10 min	400

Conclusiones y valoración final

Gracias a descubrimientos que son el resultado de este trabajo se está investigando ya cómo mejorar el escalado de frecuencia en MareNostrum 4, por qué está fijo en 2.095 GHz, si esto se debe a un mal funcionamiento del control de Turbo Boost, o si es debido a que SLURM lo está "capando" de alguna manera…, lo cual es un tema de investigación que podría llegar a ser sumamente provechoso en términos de mejora del rendimiento global del clúster.

Otro beneficio ha sido la creación de factores de conversión siguiendo una metodología que es interesante de cara a distribuir las horas de uso de la máquina en un futuro, como por ejemplo las horas de cómputo de la RES.

Obviamente, este proyecto cuenta con un margen de recorrido más amplio y deja puertas abiertas de cara a analizar con más profundidad el comportamiento de las aplicaciones, para poder comprender mejor los resultados y las causas de algunas de las incongruencias detectadas. En el tema que más se ha insistido es en el rendimiento de la memoria, porque pienso que es la parte de la arquitectura que ha sufrido mayores cambios evolutivos y sin embargo continua siendo un talón de aquiles en la búsqueda de mejora del rendimiento.

A título personal, he aprendido muchísimo acerca de las arquitecturas de MareNostrum 3 y 4, lo que es una cosa muy buena teniendo en cuenta que son máquinas que van a continuar en explotación y por tanto se seguirá ofreciendo soporte a sus usuarios. De modo que en el desempeño de mis funciones como HPC User Support podré sacar bastante provecho de todo lo aprendido, cumpliendo sobradamente con uno de los objetivos iniciales del proyecto.

Por poner una nota negativa, es también un hecho que durante la etapa de planificación quizá no se valoró en su justa medida el alcance y todo un mundo de posibilidades que se abren al emprender este tipo de estudio, sin dejar a un lado la cantidad de horas que se necesitan para alcanzar un nivel óptimo de conocimientos y destreza en el manejo de tantas herramientas de análisis, ante un supercomputador de tal magnitud y toda su tecnología asociada.

En este documento no queda plasmada una parte considerable de todo el trabajo que ha

habido detrás, como largas y complejas compilaciones (WRF), el estudio y aprendizaje de un conjunto variopinto de benchmarks, aplicaciones, de la configuración de sus inputs, parámetros, de interpretar resultados, de las equivalencias entre dos mundos distintos como LSF y SLURM, cada uno con sus peculiaridades, y muchísimas cosas más…

Como punto final y siendo muy modesto, considero que el proyecto puede servir como una buena base sobre la que se puede continuar trabajando e investigando, y pone de manifiesto que MareNostrum 4 no sólo representa una impactante cifra en TFLOPS, sino que tiene limitaciones que nos impiden aprovechar buena parte de toda la capacidad de cómputo que a día de hoy sólo ofrece sobre el papel.

Bibliografía

[1] "MareNostrum 4 entra en producción," jun 2017. [Online]. Available: https://www.bsc.es/sites/default/files/public/about/news/20170629_marenostrum_4_entra_en_produccion_esp.pdf

[2] D. Skinner, "Performance monitoring of parallel scientific applications," Lawrence Berkeley National Laboratory (LBNL), Berkeley, CA, Tech. Rep., may 2005. [Online]. Available: http://www.osti.gov/servlets/purl/881368-dOvpFA/

[3] "TOP500 List - November 2018," TOP500.org, 2018. [Online]. Available: https://www.top500.org/list/2018/11/?page=1

[4] "BSC–CNS Annual Report 2015. Operations Department," Barcelona Supercomputing Center, 2015. [Online]. Available: https://www.bsc.es/annual-report/2015/INDEX_PHP/OPERATIONS_DEPARTMENT.HTM

[5] "Grafana," Grafana Labs. [Online]. Available: https://grafana.com

[6] "Kibana," Elastic. [Online]. Available: https://www.elastic.co/products/kibana

[7] "Nagios," Nagios. [Online]. Available: https://www.nagios.org

[8] V. Eijkhout, E. Chow, and R. van de Geijn, "Introduction to High Performance Scientific Computing," Texas Advanced Computing Center, 2014. [Online]. Available: http://pages.tacc.utexas.edu/~eijkhout/Articles/EijkhoutIntroToHPC.pdf

[9] C. Minkenberg, "Interconnection Network Architectures for High-Performance Computing," 2013. [Online]. Available: https://www.systems.ethz.ch/sites/default/files/file/Spring2013_Courses/AdvCompNetw_Spring2013/13-hpc.pdf

[10] "Vectorization." [Online]. Available: http://www.nersc.gov/users/computational-systems/edison/programming/vectorization/

[11] "Qué es SCRUM." [Online]. Available: https://proyectosagiles.org/que-es-scrum/

[12] "Trello." [Online]. Available: https://trello.com

[13] "GitLab." [Online]. Available: https://about.gitlab.com

[14] G. M. Horine, *Gestion de proyectos. Edicion revisada y actualizada 2010.* Anaya Multimedia, 2009.

[15] J. O. Montesa Andrés, "Asignación de recursos en los proyectos informáticos." [Online]. Available: www.upv.es/~jmontesa/eog/6-eog00.doc

[16] C. E. Vargas, "Administración del Riesgo en Proyectos Informáticos," Club de Investigación Tecnológica, Costa Rica, 2007. [Online]. Available: http://www.clubinvestigacioncr.com/docs/informe39.pdf

[17] "Tabla de coeficientes de amortización lineal," Agencia Tributaria. [Online]. Available: http://www.agenciatributaria.es/AEAT.internet/Inicio/_Segmentos_/Empresas_y_profesionales/Empresas/Impuesto_sobre_Sociedades/Periodos_impositivos_a_partir_de_1_1_2015/Base_imponible/Amortizacion/Tabla_de_coeficientes_de_amortizacion_lineal_.shtml

[18] "Capa gratuita de AWS," Amazon Web Services, 2017. [Online]. Available: https://aws.amazon.com/es/free/#legal

[19] D. Paez Sanchez, "Gastos generales y Costes indirectos," mar 2013. [Online]. Available: http://www.eoi.es/blogs/embaon-alumnos/2013/03/18/gastos-generales-y-costes-indirectos/

[20] "BOE-A-2017-542." [Online]. Available: https://www.boe.es/boe/dias/2017/01/18/pdfs/BOE-A-2017-542.pdf

[21] J. I. González Gómez, "Los Costes. Concepto y Clasificación," 2005. [Online]. Available: http://ocw.uniovi.es/pluginfile.php/3088/mod_resource/content/1/Clasificacion_de_costes.pdf

[22] M. de Cos Castillo, *Teoría General del Proyecto. Volumen I: Dirección de Proyectos.* Editorial Síntesis, 2007.

[23] "Guía para la elaboración de Memorias de Sostenibilidad," Global Reporting Initiative, 2011. [Online]. Available: https://www.globalreporting.org/resourcelibrary/Spanish-G3.1-Complete.pdf

[24] "Término de facturación de energía activa del PVPC," Red Eléctrica de España, oct 2017. [Online]. Available: https://www.esios.ree.es/es/pvpc?date=09-10-2017

[25] "MareNostrum3 Technical Information," Barcelona Supercomputing Center, 2017. [Online]. Available: https://www.bsc.es/marenostrum/marenostrum/mn3

[26] "MareNostrum4 Technical Information," Barcelona Supercomputing Center, 2017. [Online]. Available: https://www.bsc.es/marenostrum/marenostrum/technical-information

[27] "Intel Tick-Tock Model," Intel Corporation. [Online]. Available: https://www.intel.com/content/www/us/en/silicon-innovations/intel-tick-tock-model-general.html

[28] K. Crijns, "Intel five generation IPC test: Broadwell, Haswell, Ivy Bridge, Sandy Bridge and Nehalem," Hardware.Info, aug 2015. [Online]. Available: https://us.hardware.info/reviews/6215/intel-five-generation-ipc-test-broadwell-haswell-ivy-bridge-sandy-bridge-and-nehalem

[29] S. Walton, "Intel Core i7-6700K Skylake CPU Review," TechSpot, aug 2015. [Online]. Available: https://www.techspot.com/review/1041-intel-core-i7-6700k-skylake/

[30] S. Saini, J. Chang, and H. Jin, "Performance Evaluation of the Intel Sandy Bridge Based NASA Pleiades Using Scientific and Engineering Applications," NASA Advanced Supercomputing Division; NASA Ames Research Center, Tech. Rep., 2015. [Online]. Available: https://www.nas.nasa.gov/assets/pdf/papers/NAS_Technical_Report_NAS-2015-05.pdf

[31] S. Jarp, A. Lazzaro, J. Leduc, and A. Nowak, "Evaluation of the Intel Sandy Bridge-EP server processor," CERN openlab, Tech. Rep., mar 2012. [Online]. Available: https://cds.cern.ch/record/1434748/files/CERN_openlab-Evaluation_of_the_Intel_Sandy_Bridge-EP_serverprocessor.pdf

[32] D. Kinghorn, "Intel Scalable Processors Xeon Skylake-SP (Purley) Buyers Guide," Puget Systems, dec 2017. [Online]. Available: https://www.pugetsystems.com/labs/hpc/Intel-Scalable-Processors-Xeon-Skylake-SP-Purley-Buyers-Guide-1077/

[33] "Detailed Specifications of the "Skylake-SP" Intel Xeon Processor Scalable Family CPUs," Microway Knowledge Center, 2018. [Online]. Available: https://www.microway.com/knowledge-center-articles/detailed-specifications-of-the-skylake-sp-intel-xeon-processor-scalable-family-cpus/

[34] T. Prickett Morgan, "Drilling Down Into The Xeon Skylake Architecture," The Next Platform, aug 2017. [Online]. Available: https://www.nextplatform.com/2017/08/04/drilling-xeon-skylake-architecture/

[35] B. Valentine, "Introducing Sandy Bridge." [Online]. Available: https://www.cesga.es/pt/paginas/descargaDocumento/id/135

[36] A. Eltablawy and A. Vladimirov, "Capabilities of Intel® AVX-512 in Intel® Xeon® Scalable Processors (Skylake)," Colfax International, sep 2017. [Online]. Available: https://colfaxresearch.com/skl-avx512/

[37] "Intel® Advanced Vector Extensions 512," Intel Corporation. [Online]. Available: https://www.intel.es/content/www/es/es/architecture-and-technology/avx-512-overview.html

[38] D. Colglazier, J. Jakubowski, and T. T. Brown, "Maximizing System x and ThinkServer Performance with a Balanced Memory Configuration," Lenovo Press, Tech. Rep., oct 2017. [Online]. Available: https://lenovopress.com/lp0501.pdf

[39] "Performance Guide for HPC Applications on iDataPlex dx360 M4 systems," IBM Systems and Technology Group, jun 2012. [Online]. Available: https://www.ibm.com/developerworks/community/wikis/form/anonymous/api/wiki/366c0058-50f7-4c6d-9505-9d8af81d40f1/page/837794ea-273b-4afb-bb97-40df65a100a8/attachment/41c9ccac-37fe-49cc-b553-7cda68f37580/media/performanceguideforhpcapplicationsonidata

[40] S. Williams, "Introduction to the Roofline Model," feb 2018. [Online]. Available: https://crd.lbl.gov/assets/Uploads/ECP18-Roofline-1-intro.pdf

[41] ——, "Performance Modeling and Analysis," p. 2018, jan 2018. [Online]. Available: https://crd.lbl.gov/assets/Uploads/CS267-Roofline-SWWilliams.pdf

[42] S. Williams, A. Waterman, and D. Patterson, "Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures," apr 2009. [Online]. Available: https://people.eecs.berkeley.edu/~kubitron/cs252/handouts/papers/RooflineVyNoYellow.pdf

[43] S. Williams and D. Patterson, "The Roofline Model: A pedagogical tool for program analysis and optimization," Electrical Engineering & Computer Sciences (EECS), Berkeley Par Lab, Tech. Rep., aug 2008. [Online]. Available: https://crd.lbl.gov/assets/pubs_presos/parlab08-roofline-talk.pdf

[44] B. Van Straalen, T. Ligocki, L. Lo, W. Spear, M. Cordery, S. Williams, L. Oliker, and N. Wright, "Empirical Roofline Toolkit (ERT)." [Online]. Available: https://blogs.fau.de/hager/files/2015/07/ISC15-PM-VanStraalen.pdf

[45] O. G. Lorenzo, T. F. Pena, J. C. Cabaleiro, J. C. Pichel, and F. F. Rivera, "Using an extended Roofline Model to understand data and thread affinities on NUMA systems." [Online]. Available: https://dialnet.unirioja.es/descarga/articulo/4744516.pdf

[46] R. Dolbeau, "Theoretical Peak FLOPS per instruction set on modern Intel CPUs," Bull – Center for Excellence in Parallel Programming, 2016. [Online]. Available: http://www.dolbeau.name/dolbeau/publications/peak.pdf

[47] D. Álvarez Mallón, A. Gómez Tato, and G. López Taboada, "Análisis Técnico de Rendimiento del Finis Terrae," Centro de Supercomputación de Galicia CESGA, Departamento de Aplicaciones y Proyectos, Tech. Rep., 2009. [Online]. Available: https://www.cesga.es/es/biblioteca/downloadAsset/id/416

[48] B. Armstrong, H. Bae, R. Eigenmann, F. Saied, M. Sayeed, and Y. Zheng, "HPC Benchmarking and Performance Evaluation With Realistic Applications," Purdue

University, Tech. Rep., 2006. [Online]. Available: https://pdfs.semanticscholar.org/beb6/196a7d6dfc5a447ac0967051b4677290106b.pdf

[49] A. Vladimirov, "A Survey and Benchmarks of Intel® Xeon® Gold and Platinum Processors," Colfax International, Tech. Rep., nov 2017. [Online]. Available: https://colfaxresearch.com/download/7107/

[50] E. Strohmaier and H. Shan, "Architecture Independent Performance Characterization and Benchmarking for Scientific Applications," in *Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*. Future Technology Group, Lawrence Berkeley National Laboratory, 2004. [Online]. Available: https://crd.lbl.gov/assets/pubs_presos/mascots.pdf

[51] "HPCG." [Online]. Available: http://www.hpcg-benchmark.org

[52] V. Marjanovic, J. Gracia, and C. W. Glass, "Performance modeling of the HPCG benchmark," High Performance Computing Center Stuttgart (HLRS), University of Stuttgart, Tech. Rep. [Online]. Available: https://pdfs.semanticscholar.org/6cd3/eaac09ccb608e30ce1cc85fb1c2971b958f5.pdf

[53] "Intel MPI Benchmarks User Guide," Intel Developer Zone, Tech. Rep., 2017. [Online]. Available: https://software.intel.com/en-us/imb-user-guide

[54] L. McVoy and C. Staelin, "LMbench - Tools for Performance Analysis." [Online]. Available: http://lmbench.sourceforge.net

[55] J. J. Dongarra, P. Luszczek, and A. Petitet, "The LINPACK Benchmark: Past, Present, and Future," jul 2002. [Online]. Available: http://www.netlib.org/utk/people/JackDongarra/PAPERS/hplpaper.pdf

[56] J. D. McCalpin, "STREAM: Sustainable Memory Bandwidth in High Performance Computers," Department of Computer Science School of Engineering and Applied Science University of Virginia. [Online]. Available: https://www.cs.virginia.edu/stream/

[57] ——, "Sustainable Memory Bandwidth in Current High Performance Computers," Advanced Systems Division Silicon Graphics, Inc., oct 1995. [Online]. Available: http://www.cs.virginia.edu/~mccalpin/papers/bandwidth/bandwidth.html

[58] ——, "Memory Bandwidth and Machine Balance in Current High Performance Computers," University of Delaware, sep 1995. [Online]. Available: https://www.researchgate.net/publication/213876927_Memory_Bandwidth_and_Machine_Balance_in_Current_High_Performance_Computers

[59] "Alya - High Performance Computational Mechanics," Barcelona Supercomputing Center, 2019. [Online]. Available: https://www.bsc.es/research-development/research-areas/engineering-simulations/alya-high-performance-computational

[60] M. Vázquez, G. Houzeaux, S. Koric, A. Artigues, J. Aguado-Sierra, R. Arís, D. Mira, H. Calmet, F. Cucchietti, H. Owen, A. Taha, and J. M. Cela, "Alya: Towards Exascale for Engineering Simulation Codes," Barcelona Supercomputing Center, Tech. Rep., 2014. [Online]. Available: https://www.bsc.es/sites/default/files/public/bscw2/content/research-line/documentation/1404.4881v1.pdf

[61] "CPMD." [Online]. Available: http://www.cpmd.org

[62] "Gromacs." [Online]. Available: http://www.gromacs.org

[63] "The VASP site." [Online]. Available: https://www.vasp.at

[64] G. Kresse, M. Marsman, and J. Furthmüller, "VASP the GUIDE," Computational Materials Physics, Faculty of Physics, Universität Wien, Tech. Rep., apr 2016. [Online]. Available: http://cms.mpi.univie.ac.at/vasp/vasp.pdf

[65] "Weather Research and Forecasting (WRF) Model." [Online]. Available: https://www.mmm.ucar.edu/weather-research-and-forecasting-model

[66] "Modelo WRF." [Online]. Available: https://es.wikipedia.org/wiki/Modelo{_}WRF

[67] D. G. Feitelson, *Workload Modeling for Computer Systems Performance Evaluation*. Cambridge University Press, 2014. [Online]. Available: http://www.cs.huji.ac.il/~feit/wlmod/wlmod.pdf

[68] "Environment Modules." [Online]. Available: http://modules.sourceforge.net

[69] R. P., "Intel MPI Library Release Notes for Linux* OS," Intel Developer Zone, 2017. [Online]. Available: https://software.intel.com/en-us/articles/intel-mpi-library-release-notes-linux

[70] A. Kleen, "An NUMA API for Linux," SUSE Labs, aug 2004. [Online]. Available: http://halobates.de/numaapi3.pdf

[71] K. Asifuzzaman, M. Radulovic, and P. Radojkovic, "Report on the HPC application bottlenecks," ExaNoDe, Tech. Rep., 2017. [Online]. Available: http://exanode.eu/wp-content/uploads/2017/04/D2.5.pdf

[72] Y. H. He, "Process and Thread Affinity with MPI/OpenMP on KNL," 2016. [Online]. Available: https://www.nersc.gov/assets/Uploads/IXPUG-2016-HelenHe.pdf

[73] Robert Love, "CPU Affinity," 2003. [Online]. Available: https://www.linuxjournal.com/article/6799

[74] "numactl(8) - Linux manual page," Linux man-pages project, 2018. [Online]. Available: http://man7.org/linux/man-pages/man8/numactl.8.html

[75] "perf-stat(1) - Linux manual page," Linux man-pages project, 2018. [Online]. Available: http://man7.org/linux/man-pages/man1/perf-stat.1.html

[76] J. Beckett, "NUMA Best Practices for Dell PowerEdge 12th Generation Servers Tuning the Linux OS for optimal performance with NUMA systems. Tuning the Linux OS for optimal performance with NUMA systems," DELL TechCenter, Tech. Rep. [Online]. Available: http://en.community.dell.com/techcenter/extras/m/white_papers/20266946/download

[77] A. Vladimirov and R. Asai, "Clustering Modes in Knights Landing Processors," Colfax International, may 2016. [Online]. Available: https://colfaxresearch.com/knl-numa/

[78] V. Eijkhout, "Parallel Programming in MPI and OpenMP," Texas Advanced Computing Center, 2016. [Online]. Available: http://pages.tacc.utexas.edu/~eijkhout/pcse/html/

[79] L. Bergstrom, "Measuring NUMA effects with the STREAM benchmark," University of Chicago, Tech. Rep., 2012. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.398.7394&rep=rep1&type=pdf

[80] "How do I tune my HPL.dat file?" Advanced Clustering Technologies. [Online]. Available: http://www.advancedclustering.com/act_kb/tune-hpl-dat-file/

[81] B. Barney, "Message Passing Interface (MPI)," Lawrence Livermore National Laboratory, 2018. [Online]. Available: https://computing.llnl.gov/tutorials/mpi/

[82] B. Armstrong, H. Bae, R. Eigenmann, F. Saied, M. Sayeed, and Y. Zheng, "HPC Benchmarking and Performance Evaluation With Realistic Applications." [Online]. Available: https://pdfs.semanticscholar.org/beb6/196a7d6dfc5a447ac0967051b4677290106b.pdf