E-Infrastructures
H2020-INFRAEDI-2018-2020

INFRAEDI-01-2018: Pan-European High Performance Computing infrastructure and services (PRACE)

PRACE-6IP

PRACE Sixth Implementation Phase Project

Grant Agreement Number: INFRAEDI-823767

D7.4

Evaluation of Benchmark Performance
Final

Version: 1.1
Author(s): Walter Lioen (SURF), Miguel Avillez (UEVORA), Cevdet Aykanat (Bilkent), Stefan Becuwe (UAntwerpen), Maxwell Cai (SURF), Dimitris Dellis (GRNET), Andrew Emerson (CINECA), Jacob Finkenrath (CyI), Cédric Jourdain (CINES), Holly Judge (EPCC), Ozan Karsavuran (Bilkent), Kurt Lust (UAntwerpen), Cristian Morales (BSC), Charles Moulinec (STFC), Andrew Sunderland (STFC)
Date: 30.11.2021

Project and Deliverable Information Sheet

PRACE Project	**Project Ref. №: INFRAEDI-823767**	
	Project Title: PRACE Sixth Implementation Phase Project	
	Project Web Site: https://www.prace-ri.eu/about/ip-projects/	
	Deliverable ID: < **D7.4**>	
	Deliverable Nature: <DOC_TYPE: Report>	
	Dissemination Level: PU*	**Contractual Date of Delivery:** 30 / November / 2021
		Actual Date of Delivery: 30 / November / 2021
	EC Project Officer: Leonardo Flores Añover	

* - The dissemination level are indicated as follows: **PU** – Public, **CO** – Confidential, only for members of the consortium (including the Commission Services) **CL** – Classified, as referred to in Commission Decision 2005/444/EC.

Document Control Sheet

Document	**Title: Evaluation of Benchmark Performance**	
	ID: D7.4	
	Version: <1.1>	**Status:** ***Final***
	Available at: https://www.prace-ri.eu/about/ip-projects/	
	Software Tool: Microsoft Word 2016	
	File(s): D7.4-final-1.1.docx	
Authorship	**Written by:**	Walter Lioen (SURF), Miguel Avillez (UEVORA), Cevdet Aykanat (Bilkent), Stefan Becuwe (UAntwerpen), Maxwell Cai (SURF), Dimitris Dellis (GRNET), Andrew Emerson (CINECA), Jacob Finkenrath (CyI), Cédric Jourdain (CINES), Holly Judge (EPCC), Ozan Karsavuran (Bilkent), Kurt Lust (UAntwerpen), Cristian Morales (BSC), Charles Moulinec (STFC), Andrew Sunderland (STFC)
	Contributors:	
	Reviewed by:	David Vicente, BSC Dirk Brömmel, JUELICH
	Approved by:	MB/TB

Document Status Sheet

Version	Date	Status	Comments
0.1	09/August/2021	Draft	Skeleton
0.2	13/October/2021	Draft	First integrated version
0.3	17/October/2021	Draft	
0.4	24/October/2021	Draft	
0.5	31/October/2021	Draft	
0.6	3/November/2021	Draft	For PRACE internal review
0.7	10/November/2021	Draft	
0.8	17/November/2021	Draft	PRACE internal review check
0.9	21/November/2021	Draft	
1.0	23/November/2021	Final version	For approval by MB/TB
1.1	30/November/2021	Final version	Final adjustments

Document Keywords

Keywords:	PRACE, HPC, Research Infrastructure, Applications, Benchmarking, Energy Efficiency, Systems, Energy to Solution, Time to Solution, Performance

Table of Contents

List of Figures

List of Tables

References and Applicable Documents

[1] https://www.prace-ri.eu

[2] NEMO website: https://www.nemo-ocean.eu

[3] Inputs-Outputs (using XIOS): https://forge.ipsl.jussieu.fr/nemo/wiki/Users/ModelInterfacing/InputsOutputs

[4] Standard model Output (IOM): https://www.nemo-ocean.eu/doc/node75.html#SECTION001421200000000000000

[5] https://geodynamics.org/cig/software/specfem3d_globe/

[6] D. Peter, D. Komatitsch, Y. Luo, R. Martin, N. Le Goff, E. Casarotti, P. Le Loher, F. Magnoni, Q. Liu, C. Blitz, T. Nissen-Meyer, P. Basini, and J. Tromp. Forward and adjoint simulations of seismic wave propagation on fully unstructured hexahedral meshes. Geophys. J. Int., 186(2):721–739, 2011. doi: 10.1111/j.1365-246X.2011.05044.x

[7] D. Komatitsch. Fluid-solid coupling on a cluster of GPU graphics cards for seismic wave propagation. C. R. Acad. Sci., Ser. IIb Mec., 339:125–135, 2011. doi: 10.1016/j.crme.2010.11.007

[8] Piz Daint, Tier-0 system at CSCS, Switzerland: https://www.cscs.ch/computers/piz-daint/

[9] SuperMUC-NG, Tier-0 system at LRZ, Germany: https://doku.lrz.de/display/PUBLIC/SuperMUC-NG

[10] GPAW website: https://wiki.fysik.dtu.dk/gpaw/

[11] A G Sunderland, C J Noble, V M Burke and P G Burke, A Parallel R-matrix Program PRMAT for Electron-Atom and Electron-Ion Scattering Calculations, Comput. Phys. Commun. (CPC) 145 (2002), 311–340

[12] MILC code suite: http://www.physics.utah.edu/~detar/milc/

[13] Gray, Alan, and Kevin Stratford. *A lightweight approach to performance portability with targetDP*. The International Journal of High Performance Computing Applications (2016): 1094342016682071, Also available at https://arxiv.org/abs/1609.01479

[14] B. Joo, D. D. Kalamkar, K. Vaidyanathan, M. Smelyanskiy, K. Pamnany, V. W. Lee, P. Dubey, and W. Watson III. *Lattice QCD on Intel Xeon Phi*. International Supercomputing Conference (ISC'13), 2013

[15] R. Babbich, M. Clark, and B. Joo. *Parallelizing the QUDA Library for Multi-GPU Calculations in Lattice Quantum Chromodynamics*. SC 10 (Supercomputing 2010)

[16] Constantia Alexandrou et al., *Adaptive Aggregation-based Domain Decomposition Multigrid for Twisted Mass Fermions,* Phys.Rev.D 94 (2016) 11, 114509, 1610.02370 [hep-lat]

[17] Lattice QCD multigrid library: https://github.com/sbacchio/DDalphaAMG

[18] A library for lattice QCD on GPUs: https://lattice.github.io/quda/

[19] A Python API for lattice QCD applications: https://lyncs-api.github.io/

[20] GADGET-4 website: https://wwwmpa.mpa-garching.mpg.de/gadget4/

[21] Springel, V., Pakmor, R., Zier, O., and Reinecke, M., "Simulating cosmic structure formation with the GADGET-4 code", Monthly Notices of the Royal Astronomical Society 506, 2871-2949 (2021)

[22] DEISA Benchmark Suite. Note: the DEISA Benchmarking Suite website is no longer online, but a copy can be found via the Internet Archive: https://web.archive.org/web/20120110132601/http://www.deisa.eu/science/benchmarking

[23] PRACE Accelerator Benchmark Suite. Original GitLab location: https://misterfruits.gitlab.io/ueabs/ms33.html. This repository is obsoleted by [24].

[24] UEABS, the Unified European Application Benchmark Suite. PRACE GitLab repository: https://repository.prace-ri.eu/git/UEABS/ueabs/tree/master

[25] Alan D. Simpson, Mark Bull, and Jon Hill. Identification and Categorisation of Applications and Initial Benchmarks Suite. PRACE-PP Deliverable D6.1, June 27, 2008. https://prace-ri.eu/wp-content/uploads/PP-D6.1.pdf

[26] Peter Michielse, Jon Hill, Guillaume Houzeaux, Olli-Pekka Lehto, and Walter Lioen. Report on available Performance Analysis and Benchmark Tools, Representative Benchmark. PRACE-PP Deliverable D6.3.1. November 24, 2008. https://prace-ri.eu/wp-content/uploads/PP-D6.3.1.pdf

[27] Peter Michielse, Lukas Arnold, Olli-Pekka Lehto, and Walter Lioen. Final Benchmark Suite. PRACE-PP Deliverable D6.3.2. June 18, 2010. https://prace-ri.eu/wp-content/uploads/PP-D6.3.2.pdf

[28] Mark Bull, Stefanie Janetzko, Jose Carlos Sancho, and Jeroen Engelberts. Benchmarking and Performance Modelling on Tier-0 Systems. PRACE-1IP Deliverable D7.4.2. March 26, 2012. https://prace-ri.eu/wp-content/uploads/1IP-D7.4.2.pdf

[29] Mark Bull. Unified European Applications Benchmark Suite. PRACE-2IP Deliverable D7.4. July 26, 2013. https://prace-ri.eu/wp-content/uploads/2IP-D7.4.pdf

[30] Mark Bull. UEABS Benchmarking Results. PRACE-3IP Deliverable D7.3.2. February 20, 2014. https://prace-ri.eu/wp-content/uploads/3IP-D7.3.2.pdf

[31] G. Hautreux, D. Dellis, C. Moulinec, A. Sunderland, A. Gray, A. Proeme, V. Codreanu, A. Emerson, B. Eguzkitza, J. Strassburg, and M. Louhivuori. Description of the initial accelerator benchmark suite. PRACE White Paper WP212. https://prace-ri.eu/wp-content/uploads/WP212.pdf

[32] Victor Cameo Ponz. Application performance on accelerators. PRACE-4IP Deliverable D7.5. March 24, 2017. https://prace-ri.eu/wp-content/uploads/4IP-D7.5.pdf

[33] Victor Cameo Ponz. Performance and energy metrics on PCP systems. PRACE-4IP Deliverable D7.7. January 8, 2018. https://prace-ri.eu/wp-content/uploads/4IP-D7.7.pdf

[34] Walter Lioen, Miguel Avillez, Valeriu Codreanu, Dimitris Dellis, Sagar Dolas, Andrew Emerson, Jacob Finkenrath, Cédric Jourdain, Martti Louhivuori, Cristian Morales,

Charles Moulinec, Arno Proeme, and Andrew Sunderland, authors. Giannis Koutsou and Srijit Paul, contributors. Evaluation of Accelerated and Non-accelerated Benchmarks. PRACE-5IP Deliverable D7.5. PRACE, April 18, 2019. https://prace-ri.eu/wp-content/uploads/5IP-D7.5.pdf

[35] Stephen Booth. Technical lessons learnt from the implementation of the joint PCP for PRACE-3IP. PRACE-3IP Deliverable D8.3.4, January 10, 2018. https://prace-ri.eu/wp-content/uploads/3IP-D8.3.4.pdf

[36] CORAL Benchmarks: https://asc.llnl.gov/coral-benchmarks

[37] CORAL-2 Benchmarks: https://asc.llnl.gov/coral-2-benchmarks

[38] Code_Saturne: https://www.code-saturne.org/cms/web/

[39] AmgX: https://github.com/NVIDIA/AMGX

[40] Benjamin Lindner, Loukas Petridis, Roland Schulz, and Jeremy C. Smith. Solvent-Driven Preferential Association of Lignin with Regions of Crystalline Cellulose in Molecular Dynamics Simulation. Biomacromolecules 2013, 14, 10, 3390–3398. http://pubs.acs.org/doi/abs/10.1021/bm400442n

[41] Intel-Optimised Math Library for Numerical Computing https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html

[42] Tomorov, Dongara, Baboulin. *Towards dense linear algebra for hybrid GPU accelerated manycore systems*, Parallel Computing 36 (2010) 232–240

[43] MAGMA for Matrix Algebra on GPU and Multicore architectures https://icl.cs.utk.edu/magma/

[44] LAPACK: Linear Algebra PACKage www.netlib.org/lapack

[45] AMD BLAS Library https://developer.amd.com/amd-aocl/blas-library/

[46] ICL Research Profile https://www.icl.utk.edu/research/magma

[47] Cray XC Advanced Power Management Updates: https://cug.org/proceedings/cug2018_proceedings/includes/files/pap174s2-file1.pdf

List of Acronyms and Abbreviations

aisbl	Association International Sans But Lucratif (legal form of the PRACE-RI)
AMD	Advanced Micro Devices
AmgX	Algebraic Multigrid Solver library (NVIDIA)
API	Application Programming Interface
ARM	previously Advanced RISC Machine, originally Acorn RISC Machine
ASE	Atomic Simulation Environment: A Python library for working with atoms
AVX	Advanced Vector Extensions
BCO	Benchmark Code Owner
BLAS	Basic Linear Algebra Subprograms
BSD	Berkeley Software Distribution
BXI	Bull eXascale Interconnect
CAPMC	Cray Advanced Platform Monitoring and Control
CC	C Compiler
CFD	Computational Fluid Dynamics
CG	Conjugate Gradient
CIG	Computational Infrastructure for Geodynamics
CORAL	Collaboration Oak Ridge, Argonne, Livermore
CP2K	Car-Parinello 2k

CPU	Central Processing Unit
cuBLAS	CUDA BLAS (NVIDIA)
CUDA	Compute Unified Device Architecture (NVIDIA)
cuDNN	CUDA DNN (NVIDIA)
cuFFT	CUDA FFT (NVIDIA)
cuFFTW	CUDA FFTW (NVIDIA)
DAVIDE	Development for an Added Value Infrastructure Designed in Europe
DBCSR	Distributed Block Compressed Sparse Row
DDR4	Double Data Rate 4
DEEP	Dynamical Exascale Entry Platform
DEISA	Distributed European Infrastructure for Supercomputing Applications EU project by leading national HPC centres
DFT	Density-Functional Theory
DFTB	Density-Functional based Tight Binding
DNN	Deep Neural Network
DoA	Description of Action (formerly known as DoW)
DP	Double Precision
DRAM	Dynamic RAM
EC	European Commission
EDF	Électricité de France R&D
EDR	Enhanced Data Rate
ELPA	Eigenvalue Solvers for Petaflop Applications
ESSL	Engineering Scientific Subroutine Library (IBM)
FCC	Face-Centred Cubic
FFT	Fast Fourier Transform
FFTW	Fastest Fourier Transform in the West
flop	floating-point operation
FMM	Fast-Multipole Method
FP32	32-bit Floating-Point
FP64	64-bit Floating-Point
FWI	Full Waveform Imaging
GADGET	GAlaxies with Dark matter and Gas intEracT
GAPW	Gaussian Augmented Plane Wave method
GB	Giga (= 2^{30} ~ 10^9) Bytes (= 8 bits), also GByte
Gb/s	Giga (= 10^9) bits per second, also Gbit/s
GB/s	Giga (= 10^9) Bytes (= 8 bits) per second, also GByte/s
GCC	GNU Compiler Collection
GDR	GPUDirect (NVIDIA)
Gflop/s	Giga (= 10^9) floating-point operations (usually in 64-bit, i.e. DP) per second, also GF/s
GHz	Giga (= 10^9) Hertz, frequency =10^9 periods or clock cycles per second
GNU	GNU's Not UNIX
GPGPU	General-Purpose GPU
GPL	GNU Public License
GPU	Graphic Processing Unit
GPW	Gaussian Plane Wave method
GROMACS	GROningen MAchine for Chemical Simulations
GSL	GNU Scientific Library
HBM	High Bandwidth Memory
HDF5	Hierarchical Data Format 5
HDR	High Data Rate

HFI	Host Fabric Interface
HIP	Heterogeneous-Computing Interface for Portability
HPC	High Performance Computing; Computing at a high performance level at any given time; often used synonym with Supercomputing
HPE	Hewlett Packard Enterprise
HPL	High Performance LINPACK
IB	InfiniBand
IBM	International Business Machines
IC	Initial Condition
ICL	Innovative Computing Laboratory (University of Tennessee)
ICT	Information Communication Technology
IO	Input/Output
IPMB	Intelligent Platform Management Bus/Bridge
ISC	International Supercomputing Conference; European equivalent to the US based SCxx conference. Held annually in Germany.
JU	Joint Undertaking
JUWELS	Jülich Wizard for European Leadership Science
KB	Kilo (= 2^{10} ~10^3) Bytes (= 8 bits), also Kbyte
KNC	Knights Corner (Intel)
KNL	Knights Landing (Intel)
LAPACK	Linear Algebra Package
LGPL	GNU Lesser General Public License
LINPACK	Software library for Linear Algebra
MB	Management Board (highest decision making body of the project)
MB	Mega (= 2^{20} ~ 10^6) Bytes (= 8 bits), also MByte
MB/s	Mega (= 10^6) Bytes (= 8 bits) per second, also MByte/s
MCDRAM	Multi-Channel DRAM
MD	Molecular Dynamics
Mflop/s	Mega (= 10^6) floating-point operations (usually in 64-bit, i.e. DP) per second, also MF/s
MIC	Many-Integrated Core (Intel)
MILC	MIMD Lattice Computation
MIMD	Multiple Instruction Multiple Data
MKL	Math Kernel Library (Intel)
MOOC	Massively open online Course
MoU	Memorandum of Understanding.
MPI	Message Passing Interface
MPICH	MPI over CHameleon
MPT	Message Passing Toolkit (HPE)
NAMD	Nanoscale Molecular Dynamics
NCCL	NVIDIA Collective Communications Library
NEMO	Nucleus for European Modelling of the Ocean
NetCDF	Network Common Data Form
NUMA	Non-Uniform Memory Access
OMP	OpenMP
OMPI	Open MPI
oneAPI	an open standard for a unified API
OPA	Omni-Path (Intel)
OpenACC	Open Accelerators
OpenCL	Open Computing Language
OpenMP	Open Multi-Processing

OpenMPI	Open MPI
OS	Operating System
PA	Preparatory Access (to PRACE resources)
PABS	PRACE Application Benchmark Suite
PAPI	Performance Application Programming Interface
PB	Peta (= 2^{50} ~ 10^{15}) Bytes (= 8 bits), also PByte
PC	Personal Computer
PCH	Platform Controller Hub
PCI	Peripheral Component Interconnect
PCIe	PCI express
PCP	Pre-Commercial Procurement
Pflop/s	Peta (=10^{15}) floating-point operations (usually in 64-bit, i.e. DP) per second, also PF/s
PGI	Portland Group, Inc (acquired by NVIDIA)
PLE	Parallel Locator Exchange (coupling library)
PRACE	Partnership for Advanced Computing in Europe; Project Acronym
PRACE 2	The upcoming next phase of the PRACE Research Infrastructure following the initial five year period.
PRMAT	Parallel R-matrix Program
PWscf	Plane-Wave Self-Consistent Field
PyCUDA	Python CUDA
QCD	Quantum Chromodynamics
QE	Quantum ESPRESSO
QM/MM	Quantum Mechanics/Molecular Mechanics
QUDA	A library for QCD on GPUs
RAM	Random-Access Memory
RI	Research Infrastructure
RISC	Reduced Instruction Set Computer
RUR	Resource Utilisation Reporting
SC	Supercomputing Conference (in the US)
ScaLAPACK	Scalable LAPACK
SCF	Self-Consistent Field method
SDK	Software Development Kit
SDV	Software Development Vehicle (DEEP-ER prototype)
SEM	Spectral-Element Method
SFC	Space-Filling Curve
SHOC	Scalable HeterOgeneous Computing
SIMD	Single Instruction Multiple Data
SKL	Skylake (Intel)
SKU	Stock-Keeping Unit
Slurm	Slurm Workload Manager, formerly known as Simple Linux Utility for Resource Management
SM	Streaming Multiprocessor
SPH	Smoothed-Particle Hydrodynamics
SSD	Solid-State Disk
STMV	Satellite Tobacco Mosaic Virus
SuSE	Software und System-Entwicklung
SVE	Scalable Vector Extension (ARM)
TACC	Texas Advanced Computing Center
TB	Technical Board (group of Work Package leaders)
TB	Tera (= 2^{40} ~ 10^{12}) Bytes (= 8 bits), also TByte

Tflop/s	Tera (= 10^{12}) floating-point operations (usually in 64-bit, i.e. DP) per second, also TF/s
Tier-0	Denotes the apex of a conceptual pyramid of HPC systems. In this context the Supercomputing Research Infrastructure would host the Tier-0 systems; national or topical HPC centres would constitute Tier-1
TPU	Tensor Processing Unit
TreePM	Tree Particle Mesh
UCX	Unified Communication – X framework library (Mellanox)
UEABS	Unified European Applications Benchmark Suite
US	United States
WLM	WorkLoad Manager
XIOS	XML-IO Server

List of Project Partner Acronyms

BADW-LRZ	Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften, Germany (3rd Party to GCS)
BILKENT	Bilkent University, Turkey (3rd Party to UHEM)
BSC	Barcelona Supercomputing Center - Centro Nacional de Supercomputacion, Spain
CaSToRC	The Computation-based Science and Technology Research Center (CaSToRC), The Cyprus Institute, Cyprus
CCSAS	Computing Centre of the Slovak Academy of Sciences, Slovakia
CEA	Commissariat à l'Energie Atomique et aux Energies Alternatives, France (3rd Party to GENCI)
CENAERO	Centre de Recherche en Aéronautique ASBL, Belgium (3rd Party to UANTWERPEN)
CESGA	Fundacion Publica Gallega Centro Tecnológico de Supercomputación de Galicia, Spain, (3rd Party to BSC)
CINECA	CINECA Consorzio Interuniversitario, Italy
CINES	Centre Informatique National de l'Enseignement Supérieur, France (3 rd Party to GENCI)
CNRS	Centre National de la Recherche Scientifique, France (3 rd Party to GENCI)
CSC	CSC Scientific Computing Ltd., Finland
CSIC	Spanish Council for Scientific Research (3rd Party to BSC)
CYFRONET	Academic Computing Centre CYFRONET AGH, Poland (3rd Party to PNSC)
DTU	Technical University of Denmark (3rd Party of UCPH)
EPCC	EPCC at The University of Edinburgh, UK
EUDAT	EUDAT OY
ETH Zurich (CSCS)	Eidgenössische Technische Hochschule Zürich – CSCS, Switzerland
GCS	Gauss Centre for Supercomputing e.V., Germany
GÉANT	GÉANT Vereniging
GENCI	Grand Equipement National de Calcul Intensif, France
GRNET	National Infrastructures for Research and Technology, Greece
ICREA	Catalan Institution for Research and Advanced Studies (3rd Party to BSC)
INRIA	Institut National de Recherche en Informatique et Automatique, France (3rd Party to GENCI)

IST-ID	Instituto Superior Técnico for Research and Development, Portugal (3rd Party to UC-LCA)
IT4I	Vysoka Skola Banska - Technicka Univerzita Ostrava, Czech Republic
IUCC	Machba - Inter University Computation Centre, Israel
JUELICH	Forschungszentrum Jülich GmbH, Germany
KIFÜ (NIIFI)	Governmental Information Technology Development Agency, Hungary
KTH	Royal Institute of Technology, Sweden (3rd Party to SNIC-UU)
KULEUVEN	Katholieke Universiteit Leuven, Belgium (3rd Party to UANTWERPEN)
LiU	Linkoping University, Sweden (3rd Party to SNIC-UU)
MPCDF	Max Planck Gesellschaft zur Förderung der Wissenschaften e.V., Germany (3rd Party to GCS)
NCSA	NATIONAL CENTRE FOR SUPERCOMPUTING APPLICATIONS, Bulgaria
NTNU	The Norwegian University of Science and Technology, Norway (3rd Party to SIGMA2)
NUI-Galway	National University of Ireland Galway, Ireland
PRACE	Partnership for Advanced Computing in Europe aisbl, Belgium
PSNC	Poznan Supercomputing and Networking Center, Poland
SDU	University of Southern Denmark (3rd Party to UCPH)
SIGMA2	UNINETT Sigma2 AS, Norway
SNIC-UU	Uppsala Universitet, Sweden
STFC	Science and Technology Facilities Council, UK (3rd Party to UEDIN)
SURF	SURF is the collaborative organisation for ICT in Dutch education and research
TASK	Politechnika Gdańska (3rd Party to PNSC)
TU Wien	Technische Universität Wien, Austria
UANTWERPEN	Universiteit Antwerpen, Belgium
UC-LCA	Universidade de Coimbra, Labotatório de Computação Avançada, Portugal
UCPH	Københavns Universitet, Denmark
UEDIN	The University of Edinburgh
UEVORA	University of Évora, Portugal (3rd Party to UC-LCA)
UHEM	Istanbul Technical University, Ayazaga Campus, Turkey
UIBK	Universität Innsbruck, Austria (3rd Party to TU Wien)
UiO	University of Oslo, Norway (3rd Party to SIGMA2)
UL	UNIVERZA V LJUBLJANI, Slovenia
ULIEGE	Université de Liège; Belgium (3rd Party to UANTWERPEN)
U Luxembourg	University of Luxembourg
UM	Universidade do Minho, Portugal, (3rd Party to UC-LCA)
UmU	Umea University, Sweden (3rd Party to SNIC-UU)
UnivEvora	Universidade de Évora, Portugal (3rd Party to UC-LCA)
UnivPorto	Universidade do Porto, Portugal (3rd Party to UC-LCA)
UPC	Universitat Politècnica de Catalunya, Spain (3rd Party to BSC)
USTUTT-HLRS	Universitaet Stuttgart – HLRS, Germany (3rd Party to GCS)
WCSS	Politechnika Wroclawska, Poland (3rd Party to PNSC)

Executive Summary

This deliverable presents the results of running the Unified European Application Benchmark Suite (UEABS). This work has been undertaken by Task 7.4 “Supporting European HPC Researchers” in the PRACE Sixth Implementation Phase (PRACE-6IP) project and is an extension of the work carried out in previous PRACE Implementation Phase projects.

The UEABS is a set of currently 13 application codes taken from the pre-existing DEISA Benchmark Suite, the PRACE Application Benchmark suite, and the PRACE Accelerator Benchmark Suite. The objective is providing a single benchmark suite of scalable, currently relevant and publicly available application codes and datasets, of a size which can realistically be run on large systems, and maintained in the future.

We present benchmark results and performance analyses on the current PRACE Tier-0 systems and for some applications on the recently available EuroHPC pre-exascale system HPC Vega. Furthermore, we compare the energy efficiency from an application point of view of systems where energy measurements at job level are possible. Finally, we conclude with a high-level comparison of the benchmark systems: starting with the ubiquitous LINPACK performance; followed by both application performance (time to solution, or speed) as well as energy efficiency (energy to solution). For this we combine all benchmark results and derive a comparison of the overall performance of the systems, and a comparison of the energy efficiency for the systems where we obtained energy measurements.

The results demonstrate that for some benchmarks there are significant differences in the performance obtained on the different architectures, and no one architecture gives the best performance on all the benchmarks.

The energy efficiency of the systems where energy measurements are possible strongly depends on the application benchmark / dataset / problem size / node count. Where usable, the GPU-based systems are the most energy efficient.

As expected, the optimal system/architecture strongly depends on the application benchmark / dataset / problem size / node count. The conclusion might be that LINPACK performance still is a reasonable indicator for application performance, but most people – including the LINPACK originators themselves – will disagree.

1 Introduction

The Unified European Application Benchmark Suite (UEABS) [24] is a set of currently 13 application codes taken from the pre-existing DEISA Benchmark Suite [22], the PRACE Application Benchmark suite (PABS) [27], and the PRACE Accelerator Benchmark Suite [32]. The objective is providing a single benchmark suite of scalable, currently relevant and publicly available application codes and datasets, of a size which can realistically be run on large systems, and maintained in the future.

1.1 UEABS History and Previous Work

The PRACE benchmarking activity was started during the PRACE-PP project [25][26][27] and the benchmark activities continued in PRACE-1IP [28]. The UEABS itself was only publicly released (Version 1.0) by the PRACE-2IP project [29]. Benchmarking activities continued in PRACE-3IP resulting in a new release (Version 1.1) and a benchmark report [30].

In PRACE-4IP the UEABS was updated twice (Version 1.2 and 1.3) and a separate activity on the PRACE Accelerator Benchmark Suite was started. The Accelerator Benchmark Suite [31] was based on a subset of the UEABS Version 1.2, where some applications were removed because of a lack of accelerator potential; and one application and a synthetic benchmark have been added. The Accelerator Benchmark Suite was published as GitLab repository [23], and a benchmark report targeting GPUs and Xeon Phi has been produced [32]. In the PRACE-4IP extension, a benchmark report targeting the PCP prototypes was produced [33].

In PRACE-5IP we re-integrated the accelerator versions and moved the UEABS to the PRACE git repository. The original benchmark scope has been extended by including two PRACE-3IP PCP [35] prototype systems: DAVIDE and Frioul; the Mont-Blanc 3 prototype system Dibona; and the DEEP-ER prototype system SDV. The UEABS was updated twice (Version 2.0 and 2.1). Finally, an extensive benchmark report was published [34].

1.2 Work Described in this Report

In the PRACE-6IP DoA we committed the following: “This task will also update and maintain the Unified European Applications Benchmark Suite (UEABS), including versions for standard CPUs and for GPUs. We will evaluate the results on PRACE systems using both the standard benchmarks and the accelerated benchmarks, compare where both are available, and will strive to identify reasons for, and patterns in, the performance. Task 7.4 will also investigate the maturity of energy measurement tools and, where possible, use these to analyse the energy usage of the benchmarks. UEABS can be used in future procurements and can help guide European researchers selecting systems that are most appropriate for their computational requirements.” We improved the presentation of the UEABS at the PRACE git repository, following the setup of the CORAL Benchmarks [36][37]. We updated applications and datasets, and ported the applications to new systems. Apart from the updates, we replaced the synthetic SHOC benchmark suite by TensorFlow, a well-known and frequently used software library for machine learning and artificial intelligence. For the benchmarking itself, we focused on the PRACE Tier-0 systems, but managed to extend the original scope a bit by including the recently available HPC Vega system (a EuroHPC pre-exascale system) for two of the UEABS applications.

In December 2021 we will release UEABS Version 2.2 an updated version that reflects the applications and datasets as described and used in this report.

1.3 Outline

Section 2 describes the application benchmarks, the test problems and datasets. Section 3 provides descriptions of the benchmark systems. Section 4 presents the benchmark results per application. Finally, in Section 5 – based on the benchmark results – a comparison is presented on the relative performance of the benchmark systems.

1.4 Intended Audience

The UEABS can be used as one of the benchmarks in future procurements and it can help European researchers chose systems that are appropriate for their computational requirements.

2 Application Benchmarks

Currently, the UEABS is a set of 13 application codes. In the sections below, we describe the benchmark applications, the benchmark problems and the datasets.

2.1 Alya

2.1.1 Code Description

The Alya System is a computational mechanics code capable of solving different types of physics, each one with its own model characteristics, in a coupled way. Among the problems it solves are: convection-diffusion reactions, incompressible flows, compressible flows, turbulence, bi-phasic flows and free surface, excitable media, acoustics, thermal flow, quantum mechanics (DFT) and solid mechanics (large strain).

From scratch, Alya was specially designed for massively parallel supercomputers, and the parallelisation embraces four levels of the computer hierarchy. A substructuring technique with MPI as the message passing library is used for distributed memory supercomputers. At the node level, both loop and task parallelisms are considered using OpenMP as an alternative to MPI. Dynamic load balance techniques have been introduced as well to better exploit computational resources at the node level. At the CPU level, some kernels are also designed to enable vectorisation. Finally, accelerators like GPUs are also exploited through OpenACC pragmas or with CUDA to further enhance the performance of the code on heterogeneous computers.

2.1.2 Test Cases

- Test Case A: A 132 million element mesh representing the flow around a sphere. 25-step simulation.
- Test Case B: A 1056 million element mesh representing the flow around a sphere. 100-step simulation.

2.2 Code_Saturne

2.2.1 Code Description

Code_Saturne [38] is an open-source multi-purpose CFD software, primarily developed by EDF R&D and maintained by them. The main discretisation relies on the finite volume method and a collocated arrangement of unknowns to solve the Navier-Stokes equations, for incompressible or compressible flows, laminar or turbulent flows and non-Newtonian and Newtonian fluids. Another discretisation, based on the Compatible Discrete Operators strategy is currently under development in the code. A highly parallel coupling library (Parallel Locator Exchange – PLE) is also available in the distribution to account for other physics, such as conjugate heat transfer and structure mechanics. For the incompressible solver, the pressure is solved using an integrated Algebraic Multi-Grid algorithm and the scalars are computed by conjugate gradient-like methods or Gauss-Seidel/Jacobi.

The original version of the code is written in C for pre-postprocessing, IO handling, parallelisation handling, linear solvers and gradient computation, and Fortran95 for most of the physics implementation. More and more modules of the codes are now translated into C. MPI is used on distributed memory machines and OpenMP pragmas have been added to the most costly parts of the code. The version used in this work (also freely available) relies also on external libraries, such as AmgX [39] for the pressure, and CUDA to take advantage of potential GPU acceleration.

The equations are solved iteratively using time-marching algorithms, and most of the time spent during a time step is usually due to the computation of the velocity-pressure coupling, for simple physics. For this reason, the test cases chosen for the benchmark suite have been designed to assess the velocity-pressure coupling computation, and rely on the same configuration, with Test Case A being the baseline test case and the 3 others being obtained by mesh multiplication (or global refinement), the time step being divided by 3, after each refinement, to ensure stability of the simulations.

2.2.2 Test Cases

Four test cases are dealt with the mesh size and the time step being changed. Depending on the architecture run on and the type of physics investigated, it is expected that 10,000 to 40,000 cells per MPI task are required to keep good performance.

- Test Case A: A 13 million tetrahedral cell mesh to simulate a laminar flow in a 3-D lid-driven cavity.
- Test Case B: A 111 million tetrahedral cell mesh to simulate a laminar flow in a 3-D lid-driven cavity. It is obtained by triggering mesh multiplication once, with the mesh being generated on-the-fly, or written on the disk, and read again for other tests. The time-step is divided by 3 compared to Test Case A.
- Test Case C: A 888 million tetrahedral cell mesh to simulate a laminar flow in a 3-D lid-driven cavity. It is obtained by triggering mesh multiplication twice, with the mesh being generated on-the-fly, or written on the disk, and read again for other tests. The time-step is divided by 9 compared to Test Case A.
- Test Case D: A 7 billion tetrahedral cell mesh to simulate a laminar flow in a 3-D lid-driven cavity. It is obtained by triggering mesh multiplication three times, with the mesh

being generated on-the-fly, or written on the disk, and read again for other tests. The time-step is divided by 27 compared to Test Case A.

2.3 CP2K

2.3.1 Code Description

CP2K is a freely available quantum chemistry and solid-state physics software package for performing atomistic simulations. It can be used to perform atomistic simulations of biological, chemical, liquid, crystal, molecular and solid-state systems. At the core of the density-functional theory calculations in CP2K is the QuickStep algorithm which is based on the Gaussian and Plane-Waves method (GPW). This makes use of a dual basis of atom centred Gaussian orbital and plane waves. CP2K has a wide variety of features such as molecular dynamics, QM/MM, vibrational analysis, minimisation, Monte Carlo, core level spectroscopy. Supported theory levels include DFTB, LDA, GGA, MP2, RPA, semi-empirical methods (AM1, PM3, PM6, RM1, MNDO, ...), and classical force fields.

CP2K is written in Fortran and is fully MPI parallelised. It also contains threaded OpenMP regions and can be run in hybrid MPI+OpenMP mode. This has the advantage of allowing for greater memory usage across processes. OpenMP has been increasingly added to the code and is now present in all key areas. CP2K can also make use of GPU accelerators through the addition of offloading via CUDA. At present GPU offloading has been enabled in CP2K's DBCSR library and its collocate and integrate grid operations. There is also support for using the CUDA libraries cuFFTW and cuBLAS.

CP2K makes use of LAPACK, ScaLAPACK, and BLAS for its numerical operations and these libraries are required to install CP2K. In addition, the FFTW library is also highly recommended for good performance of FFTs. Other libraries are also recommended for improving the performance of key areas of the code such as ELPA for diagonalisation and libxsmm and Cosma for matrix operations. Some features and options in CP2K are only available when it is built with particular libraries. Libint is needed for calculations of the Hartree-Fock exchange, and Libxc provides additional exchange-correlation functionals. Additional libraries supported include Sirius, Plumed, Libvori and spglib.

2.3.2 Test Cases

Test Case A – H2O-512

Test Case A is an ab initio molecular dynamics simulation of 512 liquid water molecules. The Born-Oppenheimer approach is used along with Quickstep density-functional theory (DFT). The local density approximation is used for the exchange-correlation functional and the TZV2P basis set is used. The plane wave energy cut off is set to 280 Ry. The molecular dynamics simulation runs for 10 steps.

Test Case B – LiH-HFX

Test Case B is a single point energy calculation of a 216 atom LiH crystal. The DFT calculation is done with Quickstep GAPW (augmented GPW) and the hybrid Hartree Fock exchange (HFX) is used for the exchange-correlation energy. The plane wave energy cutoff is 300 Ry. This calculation is memory intensive and requires a memory parameter (MAX_MEMORY) to

be set at run time to set the amount of memory to be assigned to the HFX module. The choice for this value depends on the amount of memory available per process, which depends on the memory per node and the number of MPI processes used at run time. Because of the high memory requirements this calculation may benefit from using multiple threads with a combination MPI+OpenMP, which increases the available memory per MPI process. For a benchmark run a converged SCF wavefunction must be supplied in order to assist with the calculation of the initial electron density.

Test Case C – H2O-DFT-LS

Test Case C is a single energy point calculation of 2048 water molecules (6144 atoms). It uses linear scaling DFT which allows for much better scaling for simple systems, with scaling up to 1 million atoms. The LDA functional is used with a DZVP molecular optimised (MOLOPT) basis set and a 300 Ry plane wave energy cutoff. The main component of this calculation involves sparse matrix-matrix multiplications which are done through the DBCSR library. This library includes operations which are offloaded to GPU.

2.4 GADGET

2.4.1 Code Description

GADGET-4 (GAlaxies with Dark matter and Gas intEracT) is an N-body/smoothed particle hydrodynamics code that is used primarily in cosmological simulations coupling gas and dark matter, and galaxies evolution (including collisions among them) taking into account the galaxies' disks, bulge, halo, and dark matter all distributed in particles. The code can also be used in classical gas dynamics problems in one, two and three dimensions.

The code includes major improvements on several algorithms over previous versions, GADGET-2 and GADGET-3. In particular the improvements occurred in the force calculation accuracy, in time-stepping, in adaptivity to a large dynamic range in time-scales, in computational efficiency, and a more-sophisticated domain decomposition algorithm. It offers several variants of Poisson solvers, among them a classic one-side tree-based multipole expansion or a fast-multipole method (FMM), both up to triakontadipole (5^{th}) order, and optional combinations of them with an FFT-based particle-mesh algorithm for the long-range gravitational field. GADGET-4 includes complex functionality for IC generation and postprocessing, such as on the fly group finders and light cone outputs.

A novelty in the code is the parallel scalability through a special MPI/shared-memory parallelisation and communication strategy based on MPI 3. When more than one MPI task is used, the code will use a hybrid communication scheme in which data stored by different MPI tasks on the same compute node are accessed directly via shared-memory. The code automatically detects groups of MPI ranks running on the same node. If more than one node is in use, at least one MPI task on each node is set aside for asynchronously serving incoming communication requests from other nodes (if only a single shared-memory node is used, this is not done). This means that multi-node jobs must have a minimum of two MPI ranks on each node. On multicore single nodes, MPI is still needed as it forms the base of the inter-core communications.

This new version of GADGET, developed by Volker Springel (the main developer) and collaborators, is mostly written in C++ (C++11 standard) and runs on Linux/Unix platforms,

including MacOS and other BSD based facilities using GSL, FFTW3, and HDF5 libraries. It can be found at [20] and is presented in a paper published in the Monthly Notices of the Royal Astronomical Society journal in 2021 [21].

2.4.2 Test Cases

2.4.2.1 Test Case A: Cosmological Dark Matter-only Simulation

This test case involves the three-dimensional simulation of the structure formation in the universe in a small box of linear length (in each direction) of 50 Mpc/h (pc denotes a parsec = 3.086×10^{16} m; Mpc = 10^6 pc; h denotes the Hubble constant) using 512^3 dark matter particles. The initial conditions are created on the fly after start-up of the simulation at redshift $Z = 63$. The simulation evolves until redshift $Z = 50$. In order to minimise memory consumption 32-bit arithmetic is used.

2.4.2.2 Test Case B: Blob Test

The blob test described in [21] and references therein consists in the simulation of a spherical cloud (blob) that is placed in a wind tunnel in pressure equilibrium with the surrounding medium. The cloud has a temperature and a density 10 times lower and higher, respectively, than the surrounding medium. This test allows for the development of hydrodynamical instabilities at the cloud surface, e.g. Kelvin-Helmholtz and Rayleigh-Taylor, leading to the cloud breakup with time. The cloud is setup with 1 million SPH particles. A more sizeable test is done with 10 million particles.

2.5 GPAW

2.5.1 Code description

GPAW [10] is a density-functional theory (DFT) program for ab initio electronic structure calculations using the projector augmented wave method. It uses a uniform real-space grid representation of the electronic wave functions that allows for excellent computational scalability and systematic converge properties.

GPAW is written mostly in Python but includes also computational kernels written in C as well as leveraging external libraries such as NumPy, BLAS, LAPACK, and ScaLAPACK. Parallelisation is based on message-passing using MPI without support for multithreading. GPAW is a CPU-based code. There is, however, a GPU version under development (again, as the first effort started in 2012 but died) by a group at CSC with no official releases so far. That code is based on CUDA (PyCUDA, cuBLAS, cuFFT, and custom CUDA kernels) but currently being ported to AMD GPUs using HIP to be ready for the upcoming LUMI pre-exascale system. GPAW is freely available under the GPL license.

Given that the GPU version is currently under heavy development and very immature, the benchmarking in PRACE-6IP is restricted to the CPU version.

2.5.2 Test Cases

The test cases were re-developed for GPAW 20.1.0 and 20.10.0. Test Cases M and L do not work well with older versions of GPAW. During the course of the PRACE-6IP project, version

21.1.0 and 21.6.0 also became available. 21.1.0 was also tested but is not fully compatible with the L test case. 21.6.0 became available too late in the project to test.

2.5.2.1 Test Case S: Carbon Nanotube

A ground state calculation for a carbon nanotube in vacuum. By default, uses a 6-6-10 nanotube with 240 atoms (freely adjustable) and serial LAPACK with an option to use ScaLAPACK. Expected to scale up to 100 MPI tasks.

2.5.2.2 Test Case M: Copper Filament

A ground state calculation for a copper filament in vacuum. By default it uses a 2×2×3 FCC lattice with 71 atoms (freely adjustable) and ScaLAPACK for parallelisation. Expected to scale up to 1000 MPI tasks.

2.5.2.3 Test Case L: Silicon Cluster

A ground state calculation for a silicon cluster in vacuum. By default, the cluster has a radius of 15 Å (freely adjustable) and consists of 702 atoms, and ScaLAPACK is used for parallelisation. Expected to scale up to 2500 MPI tasks after which scaling becomes poor on most clusters.

2.6 GROMACS

2.6.1 Code Description

GROMACS is a versatile package to perform molecular dynamics, i.e. simulate the Newtonian equations of motion for systems with hundreds to millions of particles. It is primarily designed for biochemical molecules like proteins, lipids and nucleic acids that have a lot of complicated bonded interactions, but since GROMACS is extremely fast at calculating the non-bonded interactions (that usually dominate simulations) many groups are also using it for research on non-biological systems, e.g. polymers. GROMACS supports all the usual algorithms you expect from a modern molecular dynamics implementation, but there are also quite a few features that make it stand out from the competition.

GROMACS provides extremely high performance compared to all other programs. A lot of algorithmic optimisations have been introduced in the code. In recent versions of GROMACS, on almost all common computing platforms, the innermost loops are written in C using intrinsic functions that the compiler transforms to SIMD machine instructions, to utilise the available instruction-level parallelism. These kernels are available in either single or double precision, and support all the different kinds of SIMD instructions found in x86-family (and other) processors. It is capable of hybrid parallelisation, i.e. both MPI and OpenMP and supports offloading to accelerators using CUDA.

GROMACS is free software, available under the GNU Lesser General Public License (LGPL), version 2.1 or later.

2.6.2 Test Cases

2.6.2.1 Test Case A: GluCl Ion Channel

The ion channel system is the membrane protein GluCl, which is a pentameric chloride channel embedded in a lipid bilayer. The GluCl ion channel was embedded in a DOPC membrane and solvated in TIP3P water. This system contains 142k atoms, and is a quite challenging parallelisation case due to the small size. However, it is likely one of the most wanted target sizes for biomolecular simulations due to the importance of these proteins for pharmaceutical applications. It is particularly challenging due to a highly inhomogeneous and anisotropic environment in the membrane, which poses hard challenges for load balancing with domain decomposition. This test case was used as the "Small" test case in previous PRACE-2IP–5IP projects. Benchmark is a 50000 MD steps run. It is reported to scale efficiently up to 300–1000 cores on recent x86 based systems.

2.6.2.2 Test Case B: Lignocellulose

A model of cellulose and lignocellulosic biomass in an aqueous solution [40]. This system of 3.3 million atoms is inhomogeneous. Reaction-field electrostatics are used instead of PME and therefore scales well. Benchmark is a 50000 MD steps run. This test case was used as the "Large" test case in previous PRACE-2IP–5IP projects. It is reported in previous PRACE projects to scale efficiently on 10000+ recent x86 cores.

2.6.2.3 Test Case C: STMV.28M

This is a 3×3×3 replication of the original NAMD STMV dataset from the official NAMD site, created during PRACE-6IP project. Benchmark is a 10000 MD steps run. The system contains roughly 28 million atoms and is expected to scale efficiently up to few tens of thousands x86 cores.

2.7 NAMD

2.7.1 Code Description

NAMD is a widely used molecular dynamics application designed to simulate bio-molecular systems on a wide variety of compute platforms. NAMD is developed by the "Theoretical and Computational Biophysics Group" at the University of Illinois at Urbana Champaign. In the design of NAMD particular emphasis has been placed on scalability when utilising a large number of processors. The application can read a wide variety of different file formats, for example force fields, protein structures, which are commonly used in bio-molecular science. A NAMD license can be applied for on the developer's website free of charge. Once the license has been obtained, binaries for a number of platforms and the source can be downloaded from the website. Deployment areas of NAMD include pharmaceutical research by academic and industrial users. NAMD is particularly suitable when the interaction between a number of proteins or between proteins and other chemical substances is of interest. Typical examples are vaccine research and transport processes through cell membrane proteins. NAMD is written in C++ and parallelised using Charm++ parallel objects, which are implemented on top of MPI, supporting both pure MPI and hybrid parallelisation. Offloading to accelerators is implemented for both GPU and MIC (Intel Xeon Phi).

2.7.2 Test Cases

The datasets are based on the original "Satellite Tobacco Mosaic Virus (STMV)" dataset from the official NAMD site. The memory optimised build of the package and datasets are used in benchmarking. Data are converted to the appropriate binary format used by the memory optimised build.

- Test Case A: STMV.8M

  This is a 2×2×2 replication of the original STMV dataset from the official NAMD site. The system contains roughly 8 million atoms. Benchmark is a 10000 MD steps run. This dataset scales efficiently up to 1000 x86 cores.

- Test Case B: STMV.28M

  This is a 3×3×3 replication of the original STMV dataset from the official NAMD site, created during PRACE-2IP project. Benchmark is a 50000 MD steps run. The system contains roughly 28 million atoms and is expected to scale efficiently up to few tens of thousands x86 cores.

- Test Case C: STMV.210M

  This is a 5×6×7 replication of the original STMV dataset from the official NAMD site. The system contains roughly 210 million atoms and is expected to scale efficiently up to more than hundred thousand recent x86 cores. Due to its size, benchmark is a 1200 MD steps run.

2.8 NEMO

2.8.1 Code Description

NEMO (Nucleus for European Modelling of the Ocean) [2] is a mathematical modelling framework for research activities and prediction services in ocean and climate sciences developed by a European consortium. It is intended to be a tool for studying the ocean and its interaction with the other components of the earth climate system over a large number of space and time scales. It comprises of the core engines namely OPA (ocean dynamics and thermodynamics), SI3 (sea ice dynamics and thermodynamics), TOP (oceanic tracers) and PISCES (biogeochemical process).

Prognostic variables in NEMO are the three-dimensional velocity field, a linear or non-linear sea surface height, the temperature and the salinity. In the horizontal direction, the model uses a curvilinear orthogonal grid and in the vertical direction, a full or partial step z-coordinate, or s-coordinate, or a mixture of the two. The distribution of variables is a three-dimensional Arakawa C-type grid for most of the cases.

The model is implemented in Fortran 90, with pre-processing (C-pre-processor). It is optimised for vector computers and parallelised by domain decomposition with MPI. It supports modern C/C++ and Fortran compilers. All input and output is done with third party software called XIOS with a dependency on NetCDF (Network Common Data Form) and HDF5. It is highly

scalable and a perfect application for measuring supercomputing performances in terms of compute capacity, memory subsystem, I/O and interconnect performance.

2.8.2 Test Cases

The GYRE configuration has been built to model the seasonal cycle of the double gyre box model. It consists of an idealised domain over which a seasonal forcing is applied. This allows for studying a large number of interactions and their combined contribution to large scale circulation.

The domain geometry is rectangular bounded by vertical walls and flat bottom. The configuration is meant to represent the idealised North Atlantic or North Pacific basin. The circulation is forced by analytical profiles of wind and buoyancy fluxes. The wind stress is zonal and its curl changes sign at 22 and 36. It forces a subpolar gyre in the north, a subtropical gyre in the wider part of the domain and a small recirculation gyre in the southern corner. The net heat flux takes the form of a restoring toward a zonal apparent air temperature profile.

A portion of the net heat flux which comes from the solar radiation is allowed to penetrate within the water column. The fresh water flux is also prescribed and varies zonally. It is determined such that, at each time step, the basin-integrated flux is zero.

The basin is initialised at rest with vertical profiles of temperature and salinity uniformity applied to the whole domain. The GYRE configuration is set through the `namelist_cfg` file. The horizontal resolution is determined by setting `nn_GYRE` as follows:

Jpiglo = 30 × nn_GYRE+ 2

Jpjglo = 20 × nn_GYRE + 2

In this configuration, we use a default value of 30 ocean levels, depicted by `jpkglo=31`. The GYRE configuration is an ideal case for benchmark tests as it is very simple to increase the resolution and perform both weak and strong scalability experiment using the same input files. We use two configurations as follows:

2.8.2.1 Test Case A:

- nn_GYRE = 48 suitable up to 1,000 cores
- Number of Time steps: 101
- Time step size: 20 mins
- Number of seconds per time step: 1200

We performed benchmark tests on 1024 cores using Test Case A.

2.8.2.2 Test Case B:

- nn_GYRE = 192 suitable up to 20,000 cores.
- Number of time step: 101
- Time step size(real): 20 mins
- Number of seconds per time step: 1200

We performed benchmark tests on 10,240 cores using Test Case B.

Both these test cases can give us quite good understanding of node performance and interconnect behaviour. The `ln_bench` attribute should be set to true for benchmarking. We

switch off the generation of mesh files by setting the flag `ln_meshmask` to `false` in the `namelist_ref` file. Recall that NEMO utilises XIOS for IO operations. NEMO supports both attached and detached modes of the XIOS. In the attached mode all cores are responsible for both computation and IO, whereas in the detached mode each core is only responsible for either computation or IO. This option is controlled by the `using_server` attribute defined in `iodef.xml` file. If it is set to `false`, NEMO runs on attached mode, whereas if it is set to `true`, NEMO runs on detached mode.

Since NEMO supports both weak and strong scalability, Test Case A and Test Case B both can be scaled down to run on smaller number of processors while keeping the memory per processor constant achieving similar results for step time.

2.9 PFARM

2.9.1 Code Description

PFARM is part of a suite of programs based on the 'R-matrix' ab initio approach to the variational solution of the many-electron Schrödinger equation for electron-atom and electron-ion scattering [11]. The package has been used to calculate electron collision data for astrophysical applications (such as: the interstellar medium, planetary atmospheres) with, for example, various ions of Fe and Ni and neutral O, plus other applications such as plasma modelling and fusion reactor impurities. The code has recently been adapted to form a compatible interface with the UKRmol suite of codes for electron (positron) molecule collisions thus enabling large-scale parallel outer-region calculations for molecular systems as well as atomic systems.

In the R-matrix approach, configuration space is partitioned into internal, external and asymptotic regions and the calculation is adapted accordingly for each region (Figure 1). Inner region calculations use a separate program. To enable efficient computation, the external region calculation takes place in two distinct stages, named EXDIG and EXAS, with intermediate files linking the two.

EXDIG is dominated by the assembly of sector Hamiltonian matrices and their subsequent eigensolutions, with full sets of both eigenvalues and eigenvectors required. The sector Hamiltonian matrices are dense, real, and symmetric. For electron-atom or electron-ion calculations (e.g. Test Case 1a – 1d), a very fine energy mesh is required at the lower end of the energy range in order to resolve clustered Rydberg resonances converging to all thresholds. This necessitates a large number of Legendre basis functions in the sector Hamiltonian leading to relatively large matrix sizes with closely-coupled eigenvalues. However, this level of accuracy is computationally wasteful for scattering energies at the mid-to-higher end of the energy range. To resolve this problem, the external region is configured twice within EXDIG, firstly for the *FINE* mesh (fewer, larger matrices) and then a *COARSE* mesh (more, smaller matrices). Therefore, two series of sector calculations take place within the same run. Matrix sizes are constant with each mesh.

EXAS propagates scattering energies across the external region configuration space and uses a combined functional/domain decomposition approach where good load-balancing is essential to maintain efficient parallel performance. Each of the main stages in the calculation is written in Fortran 2003, is parallelised using MPI and is designed to take advantage of highly optimised, numerical library routines. Hybrid MPI/OpenMP parallelisation has also been introduced into

the code via shared memory enabled numerical library kernels. Given the high computation, high memory load and high storage load, EXDIG is chosen here as the PFARM benchmark application code.

The MPI/OpenMP version of EXDIG employs a high-level MPI parallelisation, which assigns the complete calculation of each sector (or sub-region) to an MPI task – a 'sector MPI task'. The sector Hamiltonian matrix assembly and eigensolution is undertaken by each individual sector MPI task. Highly optimised platform-specific numerical libraries employing parallel threads, such as Intel MKL [41] and MAGMA [42] [43] are used to optimise the eigensolutions of the sector Hamiltonian matrices. Given the required full set of closely-coupled eigenpairs the eigensolver routine DSYEVD is favoured, which employs a divide-and-conquer algorithm. In this model, the maximum number of MPI tasks is equivalent to the number of sectors defined. With 1 MPI task per node, the number of OpenMP threads is usually set to the number of physical cores in a node. With multiple MPI tasks per node, the total available physical cores for OpenMP threading is divided equally between the MPI tasks

Accelerator-based implementations have been implemented for EXDIG. The GPU-enabled version of EXDIG uses the MAGMA numerical library routine MAGMA_DSYEVD to employ multiple GPUs per node for the eigensolution. The Xeon Phi-enabled version of EXDIG uses a machine-optimised version of Intel MKL, akin to the CPU version.

A fully distributed-data version using MPI with ScaLAPACK/ELPA routines is also available (though not benchmarked here). This version is suitable for very large cases, where memory within a node is insufficient for a single sector Hamiltonian matrix.

Given that the overall runtime is dominated by calls to dense linear algebra routines, PFARM performance usually attains a relatively high fraction of the peak performance of the architecture.

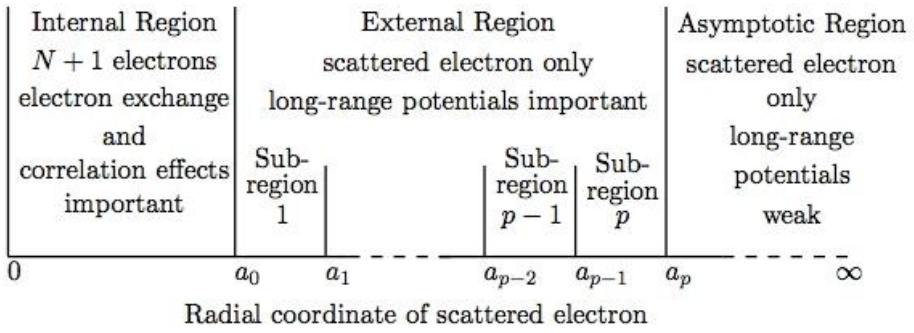

Figure 1: Partitioning of Configuration Space in PFARM

2.9.2 Test Cases

External region R-matrix propagations take place over the outer partition of configuration space, including the region where long-range potentials remain important. The length of this region is determined from the user input and the program decides upon the best strategy for dividing this space into multiple sub-regions (or sectors). Generally, a choice of larger sector lengths requires the application of larger numbers of Legendre basis functions (and therefore larger Hamiltonian matrices) to maintain accuracy across the sector and vice-versa. The test

cases chosen for the benchmarking exercise represent a representative range of configurations for atomic problems.

2.9.2.1 Test Case 1 (Atomic)

This dataset is an electron-atom scattering case with 1181 channels calculating electron scattering with FeIII. A very fine energy mesh is required at the lower end of the energy range in order to resolve multiple Rydberg resonances. The computation is divided into fine mesh and coarse mesh calculations, with larger Hamiltonian matrices associated with each sector of the fine region and smaller Hamiltonian matrices associated with each sector of the coarse region.

Test Case 1 is replicated into four different configurations to test benchmark performance across the range of benchmarking platforms.

	Test Case 1a	**Test Case 1b**	**Test Case 1c**	**Test Case 1d**
Sector Hamiltonian Dimension (Fine Region)	25982	25982	25982	18896
Sector Hamiltonian Dimension (Coarse Region)	11810	11810	11810	9448
Number of Sectors (Fine Region)	16	256	64	1024
Number of Sectors (Coarse Region)	16	256	64	1024

Table 1: PFARM (EXDIG) benchmarking datasets

Test Cases 1a and 1b are used for runs with 1 MPI task per computational node (usually using all the available cores for threads). Test Cases 1c and 1d are used for benchmark runs involving both single MPI tasks per node and multiple MPI tasks per computational node with multiple threads associated to each MPI task. For the benchmarking runs the multiple MPI task count per node is usually set to 4. This is set to: a) fit multiple sector Hamiltonian matrices on a node within node memory limits, and b) make useful GPU node comparisons where current node architectures usually have 4 GPU devices. Test Cases 1a and 1c are suitable for runs involving lower number of nodes, whilst 1b and 1d are larger, more demanding calculations that are suitable for runs involving higher node counts.

2.10 QCD

The QCD benchmark consists of a set of different kernels and comes with three different parts. Namely a legacy part, which consists of 5 different QCD kernels, taken from software packages of major European QCD collaborations and representing the most computation intensive kernels at the early stage of the UEABS benchmark suite. Within the PRACE-4IP project, the benchmark suite was extended to include kernels capable of using accelerators. Here we report on performance results obtained from “Kernel E” of the non-accelerated QCD UEABS kernels, which we will denote here as “Part 1”, as well as the accelerated kernels added during PRACE-4IP, which we will denote as “Part 2”. Kernel E is extracted from the MILC code suite (cf. [12]). The performance-portable targetDP model has been used to allow the benchmark to utilise NVIDIA GPUs, Intel Xeon Phi manycore CPUs, and traditional multi-core CPUs. The use of MPI (in conjunction with targetDP) allows multiple nodes to be used in parallel (cf. [13]).

Part 2 includes kernels from the library QPhiX[14], optimised for Intel architectures such as Skylake and KNL Xeon Phi cards, and QUDA[15], for NVIDIA GPUs.

2.10.1 Details on Benchmark Kernels:

For the used cases, namely Part 1 and Part 2, the benchmark kernels repeatedly apply the Wilson Dirac operator on an iteratively updated vector. For all cases, these repeated operator applications are carried out within a conjugate gradient (CG) method implemented in double precision, i.e. an iterative Krylov subspace solver, which apart from the operator application includes BLAS-like linear algebra operations and global reductions. The Wilson Dirac operator represents a discrete, 4-dimensional covariant derivative, defined on a regular 4-dimensional Cartesian grid. In a parallel implementation, the lattice volume is decomposed into 4-dimensional sub-domains, using one MPI process per sub-domain. As in any parallel implementation of such stencil operations, the application of the operator on grid-points of the sub-domain boundary requires information from the nearest neighbouring processes. This nearest-neighbour communication, along with a global reduction for the residual required in iterative solvers, is the most frequent communication required in any lattice QCD application, which is of the order of once every millisecond.

We perform strong scaling tests of the benchmark kernels using small to moderate problem sizes, namely V=8×64×64×64 grid points for Part 1 and V=96×32×32×32 and V=128×64×64×64 grid points for Part 2. The former two fit on typical small HPC systems, while the later problem size is representative of current state-of-the-art lattice simulations and can be scaled up to *O*(1000) of nodes.

2.11 Quantum ESPRESSO

2.11.1 Code Description

Quantum ESPRESSO is an integrated suite of open-source computer codes for electronic-structure calculations and materials modelling at the nanoscale. It is based on density-functional theory, plane waves, and pseudopotentials. The distribution consists of a “historical” core set of components, and a set of plug-ins that perform more advanced tasks, plus several third-party packages designed to be inter-operable with the core components. For the benchmarking task we chose the PWscf (Plane-Wave Self-Consistent Field) package since this is the most used and exists in both standard and accelerated versions. The program is written in Fortran and parallelised with MPI and OpenMP; the accelerated version instead requires CUDA Fortran which is available from the NVIDIA HPC SDK toolkit. The application is highly portable and in general, any combination of compilers and MPI implementations can be used to install the package, although the use of a well-optimised linear algebra library is beneficial for performance reasons. The main hardware requirement involves the accelerated version which currently can only run on NVIDIA GPUs. Because of its popularity, many computer systems in Europe already provide Quantum ESPRESSO as a pre-compiled module and this was used when found on the system, otherwise the code was downloaded from the repository and compiled separately. For the purposes of clarity, we will refer to the PWscf program simply as “Quantum ESPRESSO”.

2.11.2 Test Cases

For the benchmark datasets, we use input files from the Quantum ESPRESSO repository, in particular those referred to as AUSURF, GRIR443 and CNT. These files consist of input files representing the materials to model, together with a parameter to control various aspects of the calculation. These datasets have been renamed as small, medium and large in the UEABS repository to reflect the resources required to perform the benchmark. Since the small dataset scales only to a few tens of cores, only the medium and large inputs have been used in this study. It should be recalled that to get the best performance with PWscf it is important to specify the *k*-points displayed by the input structure on the command line with the *-npool* option. For example, for the medium (GRIR443) dataset we have 4 *k*-points so PWscf should be run as:

mpirun pw.x -npool 4 -input pw.in

Since MPI tasks are assigned to each *k*-point, the total number of tasks needs to be divisible by the number of *k*-points. For the large (CNT) dataset instead we have only 1 *k*-point and therefore this option is not required. Given the high memory requirements of the application, we frequently use hybrid MPI/OpenMP – in most of our runs we have fixed the number of OpenMP threads to be 8. Finally, we note that for the benchmarks described here we used Quantum ESPRESSO versions 6.5–6.8 (according to availability). The minor releases of the code represent bug fixes and slightly different functionalities, which should not influence the performance.

2.12 SPECFEM3D

2.12.1 Code Description

In collaboration with Princeton and the University of Pau (France), CIG offers the software package SPECFEM3D_GLOBE (version 7.0) [5] which simulates global and regional (continental-scale) seismic wave propagation. More precisely, it simulates three-dimensional global and regional seismic wave propagation and performs full waveform imaging (FWI) or adjoint tomography based upon the spectral-element method (SEM). The SEM is a continuous Galerkin technique [5], which can easily be made discontinuous; it is then close to a particular case of the discontinuous Galerkin technique, with optimised efficiency because of its tensorised basis functions. In particular, it can accurately handle very distorted mesh elements (Oliveira and Seriani 2011). Effects due to lateral variations in compressional-wave speed, shear-wave speed, density, a 3D crustal model, ellipticity, topography and bathymetry, the oceans, rotation, and self-gravitation are included. The package can accommodate full 21-parameter anisotropy as well as lateral variations in attenuation. Adjoint capabilities and finite-frequency kernel simulations are also included.

This package is one of the very few and still popular open-source codes for the global computational seismology community. Although the last official release was in 2015; the SPECFEM3D_GLOBE code is still maintained and still widely used by the community as it stands out as one of the most advanced codes for 3D global simulations (NVIDIA uses it for one of its benchmarks). It is also very well suited to parallel implementation on very large supercomputers as well as on clusters with GPUs as accelerators. As a result, SPECFEM3D_GLOBE is a reference application for supercomputer benchmarking thanks to its good scaling capabilities.

SPECFEM3D_GLOBE is mainly written in Fortran but a subset has been ported to C, in order to experiment with CUDA, StarSs and OpenCL. This subset contains the main computation loop of the main application. The full application consists of 100k lines of Fortran, while the subset contains 20k lines of C. It uses no obsolete or obsolescent features of Fortran. The package uses parallel programming based upon the Message Passing Interface (MPI), which use non-blocking MPI for much better performance for medium or large runs [6] and which includes several performance improvements in mesher and solver. The package includes support for GPU acceleration [7] and also supports OpenCL.

2.12.2 Test Cases

The test cases simulate the earthquake of June 1994 in Northern Bolivia at a global scale with the global (3D) shear-wave speed model named S362ANI.

The different test cases correspond to different meshes of the earth. The size of the mesh is determined by a combination of following variables: NCHUNKS, the number of chunks in the cubed sphere (6 for global simulations), NPROC_XI, the number of processors or slices along one chunk of the cubed sphere and NEX_XI, the number of spectral elements along one side of a chunk in the cubed sphere. These three variables give us the number of degrees of freedom of the mesh and determine the amount of memory needed per core. The SPECFEM3D_GLOBE mesher and solver must be recompiled each time we change the mesh size because the solver uses a static loop size and the compilers know the size of all loops only at the time of compilation and can therefore optimise them efficiently.

To benchmark and measure the performance of each system we used three test cases.

2.12.2.1 Validation Test Case

A small Validation Test Case called “small_benchmark_run_to_test_more_complex_Earth” which is a native SPECFEM3D_GLOBE benchmark to validate the compilation and behaviour of the code. Indeed; the solver calculates seismograms for 129 stations, these histograms allow to scientifically validate the compilation and the results of the simulation thanks to a Python script which allows to compare the results of the simulated histograms with the results of reference histograms.

This benchmark is designed to run on a system with at least 24 x86 cores. The simulation runs with 24 MPI tasks using hybrid parallelisation (MPI+OpenMP or MPI+OpenMP+CUDA depending on the system tested) and has the following mesh characteristics: NCHUNKS=6, NPROC_XI=2 and NEX_XI=80.

2.12.2.2 Test Case A

Test Case A is designed to run on a system that has up to about 1,000 x86 cores, or equivalent. The simulation runs with 96 MPI tasks using hybrid parallelisation and has the following mesh characteristics: NCHUNKS=6, NPROC_XI=4 and NEX_XI=384.

2.12.2.3 Test Case B

Test Case B is designed to run on systems up to about 10,000 x86 cores, or equivalent. The simulation runs with 1,536 MPI tasks using hybrid parallelisation and has the following mesh characteristics: NCHUNKS=6, NPROC_XI=16 and NEX_XI=384.

2.13 TensorFlow

2.13.1 Code Description

TensorFlow is a popular open-source library for symbolic math and linear algebra, with particular optimisation for neural-networks-based machine learning workflow. Maintained by Google, it is widely used for research and production in both the academia and the industry.

TensorFlow supports a wide variety of hardware platforms (CPUs, GPUs, TPUs) and can be scaled up to utilise multiple computing devices on one or more compute nodes. The main objective of this benchmark is to profile the scaling behaviour of TensorFlow on different hardware, and thereby provide a reference baseline of its performance for different sizes of applications.

TensorFlow is a Python and C/C++ library rather than a standalone application, and therefore there are many possible implementations depending on the actual applications. We choose DeepGalaxy, an astrophysics-oriented scalable galaxy image classification/searching deep learning network with TensorFlow backend. Within a node or a CPU socket, parallelism is done with OpenMP. If GPUs are available, most convolutional operations and gradient calculations are done on the GPUs (with cuDNN being the backend). If GPUs are not available, TensorFlow makes use of oneAPI (formally MKL-DNN) to speed-up the calculation on Intel CPUs, and the AVX2 instructions for AMD CPUs. DeepGalaxy is scaled up in the data parallel manner, that is, the neural network is cloned to make multiple copies, and each copy receives different training data as input. These training data result in different neural network activation maps, but they are periodically synchronised to make sure that the weights are updated collectively and consistently. When DeepGalaxy is trained on multiple nodes, collective communication protocols such as MPI and NCCL are used to communicate the gradients obtained from each worker. The AllReduce() primitive in MPI/NCCL is particularly relevant. These communications are handled by an open-source framework Horovod, which essentially acts as a wrapper for TensorFlow and PyTorch to handle gradient communications.

DeepGalaxy is written in Python and is open-source and freely available. The datasets required for training the neural network are also publicly available. DeepGalaxy and its underlying libraries (TensorFlow, Horovod, MPI, NCCL, cuDNN, oneAPI) are highly optimised, and therefore this benchmark suite is particularly useful to test the scaling efficiency of HPC systems: the closer to a linear scaling behaviour, the better scaling efficiency for a HPC system.

2.13.2 Test Cases

The benchmarks can be done on a wide range of systems, from PCs to supercomputers. Three test cases are designed to systems with different hardware configurations.

2.13.2.1 Test Case A (small)

This test case is designed to test the training performance of a small-to-medium size dataset (~2 GB compressed, ~100 GB uncompressed) on a medium-size deep neural network (DNN). The DNN is relatively small (about 17 million parameters) and therefore can be trained on a single modern GPU.

2.13.2.2 Test Case B (medium)

This test case is designed to test the training performance of a small-to-medium size dataset (~2 GB compressed, ~100 GB uncompressed) on a moderately large DNN. The large DNN contains 64 million parameters. In comparison, the popular ResNet50 architecture for image classification contains 23 million parameters. This test case requires top-class GPUs with 24 GB of GPU memory or more. Even with such a GPU, the local batch size should usually be limited to 4, otherwise the GPU memory will be exhausted.

2.13.2.3 Test Case C (large)

This test case is designed to push a HPC system to the limit. The dataset is large (16 GB compressed, 2 TB uncompressed), running on a large DNN consisting 64 million parameters. With such a combination, the memory footprint is roughly 160 GB even with a batch size of 1, making it nearly impossible to fit into any GPU memory. As such, this test case is currently run on the CPUs. It is possible to run this case on GPUs using unified memory, although in this case the overhead of transferring data between the host memory and the GPU becomes a bottleneck.

3 Benchmark Systems

3.1 PRACE Tier-0 Systems

3.1.1 Hawk

Hawk is a Tier-0 system hosted by HLRS in Germany. Hawk is an HPE Apollo machine and has 5,632 compute nodes (720,896 cores) which are based on AMD EPYC processors. The system has a peak performance of 26 Pflop/s. Each node consists of:

- 2 × 64 core AMD EPYC 7742 processors which operate at 2.25 GHz.
- 256 GB of memory, or 2 GB per core.
- A hierarchical architecture where cores are grouped into 4 core complexes which share an L3 cache of 16 MB.

The interconnect used is InfiniBand HDR200, which has a bandwidth of 200 Gbit/s and an approximate latency of 1.3 microseconds per link. The interconnect topology is a 9-dimensional hypercube. Due to the use of topology aware scheduling larger jobs can only request 64, 128, 256, 512, 1024, 2048 or 4096 nodes.

3.1.2 JUWELS

The supercomputer **JU**elich **W**izard for **E**uropean **L**eadership **S**cience, known as JUWELS, consists of two main modules, the Cluster Module based on Intel Xeon Skylake chips and the Booster Module based on NVIDIA GPGPU A100. The system is hosted by the Jülich Supercomputing Centre and is currently the fastest system in Europe with 73 Petaflop per second theoretical peak performance by the Booster Module. Both Modules are connected to the storage cluster JUST via 350/250 GB/s network from Booster/Cluster respectively. The older Cluster module consists of 2271 standard compute nodes each with

- 2 × Intel Xeon Platinum 8168 CPU, 2 × 24 cores, 2.7 GHz
- 96 (12× 8) GB DDR4, 2666 MHz
- InfiniBand EDR (Connect-X4)

Additionally, a smaller large memory partition with 240 nodes equipped with 196 GB DDR4, and a GPU partition with 56 nodes equipped with 4 NVIDIA V100 is available. The network is a Mellanox InfiniBand EDR fat-tree network with 2:1 pruning at leaf level and top-level HDR switches. Moreover, it provides a 40 Tb/s connection to the Booster Module for modular supercomputing.

The newer Booster Module consists of 936 compute nodes each with

- 2 × AMD EPYC Rome 7402 CPU, 2 × 24 cores, 2.8 GHz
- 512 GB DDR4, 3200 MHz
- 4 × NVIDIA A100 GPU, 4 × 40 GB HBM2e
- 4 × InfiniBand HDR (Connect-X6, 200 Gbit/s each) with DragonFly+ topology with 20 cells

The 4 NVIDIA A100 GPUs on each node are connected via NVLink3 to each other while the CPU, GPU, and network adapter are connected via 2 PCIe Gen 4 switches with 16 PCIe lanes which are going to each device.

3.1.3 Joliot-Curie

Joliot-Curie is a Tier-0 machine hosted by CEA's Very Large Computing Centre (TGCC) in France. It is made of several partitions, the main ones being Skylake, KNL, and Rome. The system's peak performance is 22 Pflop/s.

The two first partitions are based on BULL Sequana X1000 and are split as follows:

- The SKL Irene (Skylake) partition consists of:
    i. 1,656 dual-processor Intel Skylake 8168 fine nodes at 2.7 GHz with 24 cores per processor, for a total of 79,488 computing cores and a power of 6.86 Pflop/s,
    ii. 192 GB of DDR4 memory/node (or 4 GB per core),
    iii. InfiniBand EDR interconnect network.
- The KNL Irene (Knights Landing) partition consists of:
    i. 828 Intel KNL 7250 manycore nodes at 1.4 GHz with 68 cores per processor, for a total of 56,304 cores and a power of 2 Pflop/s,
    ii. 96 GB of DDR4 memory + 16 GB of MCDRAM memory/node,
    iii. Bull eXascale Interconnect network (BXI).

The last main partition is built on Bull Sequana XH2000 as:

- The Rome Irene partition consists of:
    i. 2292 dual-processor AMD Rome (EPYC) compute nodes at 2.6 GHz with 64 cores per processor, for a total of 293,376 computing cores and a power of 11.75 Pflop/s,
    ii. 256 GB DDR4 memory/node (or 2 GB per core),
    iii. InfiniBand HDR100 interconnect network.

The SKL partition is ranked 101th in June 2021 TOP500 list and the Rome one 59th in the same list. Scratch and work Lustre file systems are available, with 4.6 and 8 PB, respectively.

3.1.4 MARCONI100

The MARCONI100 Tier-0 system is hosted by CINECA and consists of 980 compute nodes based on IBM Power9 processors and NVIDIA V100 GPUs. Each node consists of:

- 2 × 16 core IBM Power9 processors running at 3.1 GHz and with 4-way hyperthreading to give 128 virtual cores
- 4 × NVIDIA V100 GPUs each with 16 GB of memory and connected with NVLink 2.0.
- 256 GB of main memory.

Each node can provide a performance of 32 Tflop/s, which gives a combined peak performance of close to 32 Pflop/s for the whole system. In addition, the nodes are connected by a Mellanox IB EDR DragonFly network and can access 8 PB of disk space.

In the latest TOP500 ranking (June 2021), the MARCONI100 is in 14th position which makes it the 3rd most powerful supercomputer in Europe. Finally, we note that since the system is based on a relatively small number of very powerful nodes, the best application performance will be obtained from applications which demonstrate good GPU acceleration rather than high parallel scalability.

3.1.5 MareNostrum4

MareNostrum4 is the Tier-0 system hosted by BSC, Spain. It is based on Intel Xeon Platinum processors from the Skylake generation. It is a Lenovo system composed of SD530 Compute Racks, an Intel Omni-Path high performance network interconnect and running SuSE Linux Enterprise Server as operating system. Its current LINPACK R_{max} performance is 6.2 Pflop/s.

This general-purpose block consists of 48 racks housing 3456 nodes with a grand total of 165,888 processor cores and 390 TB of main memory. Compute nodes are equipped with:

- 2 sockets Intel Xeon Platinum 8160 (Skylake) CPU with 24 cores each @ 2.10 GHz for a total of 48 cores per node; L1d 32 kB; L1i cache 32 kB; L2 cache 1024 kB; L3 cache 33792 kB
- 96 GB of main memory 1.9 GB/core (216 nodes high memory, 10368 cores with 7.9 GB/core)
- 100 Gbit/s Intel Omni-Path HFI Silicon 100 Series PCIe adapter (in a full fat tree topology)
- 10 Gbit Ethernet
- 200 GB local SSD available as temporary storage during jobs

3.1.6 SuperMUC-NG

SuperMUC-NG [9] is the Tier-0 system hosted by LRZ, Germany. SuperMUC-NG contains thin and fat compute nodes which differ in memory size. Properties of those nodes are as follows:

- 6,336 Thin compute nodes each with 96 GB memory
- 144 Fat compute node each with 768 GB memory

- Both thin and fat compute nodes are equipped with 2 × 24 core Intel Xeon Platinum 8174 (Skylake) processor running at 3.1 GHz.

Therefore, in total the system contains 311,040 compute cores with a main memory of 719 TB and has a peak performance of 26.9 Pflop/s.

The internal interconnect is an Omni-Path network with 100 Gbit/s. The compute nodes are bundled into 8 domains (islands). Within one island, the Omni-Path network topology is a 'fat tree' for highly efficient communication. The Omni-Path connection between the islands is pruned (pruning factor 1:4).

In addition to the compute nodes there are 64 nodes in the Compute Cloud of SuperMUC-NG (half of them equipped with two GPUs each), and one huge memory node with 6 TB and 192 cores.

3.1.7 Piz Daint

Piz Daint [8] is the Tier-0 system hosted by CSCS, Switzerland. Piz Daint is a Cray XC40/XC50 system:

- 5704 XC50 nodes with one Intel Xeon E5-2690 v3 (Haswell) @ 2.60 GHz (12 cores, 64 GB RAM) and one NVIDIA Tesla P100 (16 GB)
- 1813 XC40 nodes with two Intel Xeon E5-2695 v4 (Broadwell) @ 2.10 GHz (2 × 18 cores, 64/128 GB RAM).

The system has an Aries interconnect using a Dragonfly topology.

Cray XC40/ XC50 has advanced power monitoring and control features enabled on the compute blades. This helps system administrators and researchers involved in advanced power monitoring, power aware computing, and energy efficient computing. All blades developed for Cray XC platform supports out of band collection of energy statistics by default at 1 Hz.

Node level, cabinet level and system level energy data are exposed via Cray advanced platform monitoring and control (CAPMC) to the system workload manager (WLM). The additional or optional way of collecting energy statistics is through pm counters located on "**/sys/cray/PM_COUNTERS**" path. Cray supports resource utilisation reporting (RUR) and PAPI (Performance application performance interface) [47].

Node level power capping on Cray XC50 blade supporting Intel Xeon scalable processors utilises Intel node manager firmware running on the platform controller hub (PCH). Cray firmware communicates with the Intel firmware over an Intelligent Platform Management Bus (IPMB). The implemented power capping utilises the Intel Running Average Power limit.

Additional references for Cray's energy monitoring and documentation can be found in [47].

3.2 EuroHPC System

3.2.1 HPC Vega

Vega is a Tier-0 system hosted at the IZUM, Slovenia. Vega has three partitions in total: thin and fat differing in the memory size and an NVIDIA GPU partition. All are based on dual AMD EPYC Rome CPUs with the following characteristics:

- Standard compute nodes partition: 768 dual AMD EPYC Rome CPUs, with a total of 98,304 cores, 256 GB RAM DDR4-3200 per node, corresponding to 2 GB RAM/core.
- Large memory compute nodes partition: 192 dual AMD EPYC Rome CPUs, in a total of 24576 cores, and 1 TB RAM DDR4-3200 per node, corresponding to 8 GB RAM/core.
- GPU partition: 60 dual AMD EPYC Rome CPUs (total of 7680 cores) with 512 GB RAM DDR4-3200 per node, corresponding to 4 GB RAM/CPU core, and quad NVIDIA Ampere A100 PCIe GPUs (40 GB, 3456 FP64 CUDA cores, 432 Tensor cores, Peak FP64 9.7 Tflop/s, FP64 Tensor Core 19.5 Tflop/s)

In total the system contains 1020 compute nodes with dual AMD CPUs with 130,560 cores and 414 TB RAM. Sustained performance on all CPUs is 3.8 Pflop/s. 240 GPU accelerators with a total of 829,440 FP64 CUDA cores and 103,680 Tensor cores perform 3.1 Pflop/s. Overall Vega has a sustained performance of 6.9 Pflop/s and a peak performance of 10.1 Pflop/s.

The internal interconnect consists of 68 × 40-port Mellanox HDR switches with a Dragonfly+ topology, with all 960 compute nodes, 60 GPU, and 8 login nodes connected through Mellanox ConnectX-6 (single or dual port).

In addition to the compute nodes and the GPU partitions, Vega has a Lustre-based high-performance storage tier and a Ceph-based large-capacity storage tier. Furthermore, Vega has a virtualisation partition composed of 30 dual AMD EPYC 7502 virtualisation nodes, each with 512 GB of RAM DDR4-3200 and two interconnects – 100 GbE DP and InfiniBand HDR100.

3.3 Energy Measurement Capability/Availability

Energy accounting at job level was not available on all systems. Piz Daint has integrated accounting as it is described in Section 3.1.7 and its references. The MareNostrum4 accounting system supports the job consumed energy but in the end this energy is not logged for all jobs. SuperMUC-NG supports job energy accounting but after an update this became unavailable. Thus, the only system with full energy to solution for all jobs ran is Piz Daint, while for MareNostrum4 and SuperMUC-NG, these measurements are available for jobs that ran while the energy accounting was active.

4 Benchmark Results per Application

4.1 Alya

The Alya benchmarks have been performed on systems with different architectures, Skylake (JUWELS, Irene, SuperMUC-NG, MareNostrum4), AMD (Irene, Hawk), and GPU-NVIDIA (Piz Daint, MARCONI100). Both Test Case A and Test Case B have been tested on all the mentioned systems. The version “open-alya” of Alya was used in all the cases.

The elapsed time of only the time-integration phase has been considered, since it is the dominant part in the production runs of Alya. Likewise, the node workload for each system was selected according to the similar configurations used in scientific simulations.

The energy measurements were done on MareNostrum4 and Irene-Skylake. During the allocation period, the energy was not recorded on SuperMUC-NG. The energy measurements

were obtained using the sacct command with the field variable ConsumedEnergy. The energy measurements are from the whole simulation, but the performance is calculated using the time-integration phase as commented before.

Pure MPI runs were performed for all test cases on all machines. Except for Test Case B on Skylake systems, all the test were performed on fully occupied nodes. For Test Case B on the Skylake systems, we observed better performance and better scalability using 46 processes per node instead of 48.

4.1.1 Performance on Skylake Systems (Test Case A)

Table 2: Alya, Test Case A, MareNostrum4

– Table 5 present the results for the Skylake systems for Test Case A from 192 cores to 1536 cores. On the four Skylake systems Alya was compiled with Intel Compilers.

We observe better than ideal performance on most of the test cases due to the memory exhaustion of the smallest runs. We observe that the best performance of Alya on Test Case A is running it on JUWELS Cluster, but close to SuperMUC-NG and Irene-Skylake. Additionally, we observe that the slower system is MareNostrum4, due to the lower CPU frequency (2.1 GHz) compared with the other systems (2.7 GHz). The parallel efficiency is very similar on the four systems, but slightly better on the JUWELS system.

Nodes	**Time (s)**	**Speed-up**	**Efficiency**
4	148.72	1.00	100%
8	69.94	2.13	106%
16	36.01	4.13	103%
32	20.19	7.37	92%

Table 2: Alya, Test Case A, MareNostrum4

Nodes	**Time (s)**	**Speed-up**	**Efficiency**
4	125.29	1.00	100%
8	57.06	2.20	110%
16	27.93	4.49	112%
32	15.76	7.95	99%

Table 3: Alya, Test Case A, JUWELS

Nodes	**Time (s)**	**Speed-up**	**Efficiency**
4	130.16	1.00	100%
8	59.51	2.19	109%
16	28.99	4.49	112%
32	14.93	8.72	109%

Table 4: Alya, Test Case A, SuperMUC-NG

Nodes	Time (s)	Speed-up	Efficiency	Energy (kJ)
4	137.95	1.00	100%	279.23
8	64.88	2.13	106%	271.23
16	31.81	4.34	108%	265.23
32	17.67	7.81	98%	280.12

Table 5: Alya, Test Case A, Irene-SKL

4.1.2 Performance on Skylake Systems (Test Case B)

Table 6 – Table 9 present the results for the Skylake systems for Test Case B from 1536 to 12288 cores. As for Test Case A, Alya is compiled with Intel Compilers. As we commented before, these runs were with 46 MPI tasks per node instead of 48 MPI tasks per node. We observe better performance on JUWELS and SuperMUC-NG. These systems are the ones with hyperthreading enabled. Alya's performance is highly affected by context switches, and when using hyperthreading, the processor handles the context switches quicker. Also, this is why the performance is better when we leave two cores free per node for the system processes, therefore avoiding context switches. Additionally, the scalability of Alya on the Skylake systems are very similar except for Irene-Skylake, where it is a slightly slower and we observe a lower scalability.

For both test cases we observe a similar power consumption on MareNostrum4 and Irene-Skylake. For Test Case B, the energy consumption is higher on Irene, but it is because the simulation is slower than on MareNostrum4 and it takes more time.

Nodes	Time (s)	Speed-up	Efficiency	Energy (kJ)
32	877.01	1.00	100%	11894.84
64	440.55	1.99	100%	13102.04
128	227.02	3.86	97%	12526.71
256	144.15	6.08	76%	13446.46

Table 6: Alya, Test Case B, MareNostrum4

Nodes	Time (s)	Speed-up	Efficiency
32	710.39	1.00	100%
64	349.93	2.03	102%
128	184.78	3.84	96%
256	116.93	6.08	76%

Table 7: Alya, Test Case B, JUWELS

Nodes	Time (s)	Speed-up	Efficiency
32	740.12	1.00	100%
64	395.83	1.87	93%
128	201.53	3.67	92%
256	120.24	6.16	77%

Table 8: Alya, Test Case B, SuperMUC-NG

Nodes	Time (s)	Speed-up	Efficiency	Energy (kJ)
32	897.84	1.00	100%	12034.11
64	432.52	2.08	104%	12523.14
128	271.15	3.31	83%	16345.45
256	175.87	5.11	64%	17113.82

Table 9: Alya, Test Case B, Irene-SKL

4.1.3 Performance on AMD Systems (Test Case A)

Table 10 and Table 11 present the results for the AMD systems for Test Case A from 512 to 4128 cores. The code was compiled with Intel Compilers enabling the AVX2 instruction set with the compiler flags. We used a pure MPI configurations on both systems with 128 MPI tasks per node.

The performance and scalability are very similar on both systems, as they have very similar CPU and network. If we compare the AMD results with the Skylake results, we observe that the core-to-core performance are similar, but if we compare the node-to-node performance, the AMD systems are faster, as they have 128 cores per node.

Nodes	Time (s)	Speed-up	Efficiency
4	68.92	1.00	100%
8	31.11	2.22	111%
16	15.65	4.40	110%
32	8.12	8.49	106%

Table 10: Alya, Test Case A Irene-Rome

Nodes	Time (s)	Speed-up	Efficiency
4	67.92	1.00	100%
8	29.30	2.32	116%
16	14.34	4.74	118%
32	7.44	9.13	114%

Table 11: Alya, Test Case A, Hawk

4.1.4 Performance on AMD Systems (Test Case B)

Table 12 and Table 13 present the results of the AMD systems for Test Case B from 2048 to 16384 cores. The code was compiled and run like for Test Case A.

As we have seen for Test Case A, the performance is very similar for both systems. If we compare the parallel efficiency with the Skylake system results, we observe that on the AMD systems the efficiency is higher with the largest runs because it is with less nodes than the largest on the Skylake systems.

Nodes	Time (s)	Speed-up	Efficiency
16	842.94	1.00	100%
32	434.53	1.94	97%
64	236.77	3.56	89%
128	121.12	6.96	87%

Table 12: Alya, Test Case B Irene-Rome

Nodes	Time (s)	Speed-up	Efficiency
16	757.98	1.00	100%
32	419.34	1.81	90%
64	191.82	3.95	99%
128	120.40	6.30	79%

Table 13: Alya, Test Case B, Hawk

4.1.5 Performance on GPU Systems (Test Case A and Test Case B)

Table 14 and Table 15 present the results for the GPU systems for Test Case A and Table 16 and Table 17 present the results for the GPU systems for Test Case B. Alya was compiled on both systems using PGI and CUDA compilers. All the runs were with 1 MPI task per physical core and using all the GPUs available on each node, 1 on Piz Daint and 4 on MARCONI100. Despite the decrease in parallel performance, the GPU on average still runs 2.5 times faster than the pure CPU implementation on Skylake systems. Although, the parallel efficiency of Test Case A is slightly worse on the GPU systems compared to the other kind of systems, where we usually observe ideal scaling. On the other hand, the scalability of Test Case B on the GPU systems is similar to the other systems.

Nodes	Time (s)	Speed-up	Efficiency
4	335.12	1.00	100%
8	166.67	2.01	101%
16	87.11	3.85	96%
32	44.54	7.52	94%

Table 14: Alya, Test Case A, Piz Daint

Nodes	Time (s)	Speed-up	Efficiency
1	320.52	1.00	100%
2	172.27	1.86	93%
3	119.03	2.69	90%
4	92.78	3.45	86%

Table 15: Alya, Test Case A, MARCONI100

Nodes	Time (s)	Speed-up	Efficiency
32	1059.21	1.00	100%
64	538.72	1.81	90%
128	280.15	3.94	98%
256	143.62	6.25	78%

Table 16: Alya, Test Case B, Piz Daint

Nodes	Time (s)	Speed-up	Efficiency
16	723.41	1.00	100%
32	393.04	1.84	92%
64	213.08	3.40	85%
128	117.26	6.17	77%

Table 17: Alya, Test Case B, MARCONI100

4.2 Code_Saturne

The tests have been conducted on 6 machines: SuperMUC-NG, MareNostrum4, Hawk, JUWELS, Joliot-Curie - Skylake and Joliot-Curie - Rome. Test Case A has been run on all the machines using the default SFC Morton partitioner. Test Case B has been used on SuperMUC-NG to investigate the influence of the partitioner on the time to solution to derive the best strategy to run Test Cases C and D, and beyond. It was also possible to run an extremely large simulation (8 times the size of Test Case D) on SuperMUC-NG and Hawk and performance results are included for this case. Finally, with energy measurements taken on MareNostrum4 and Joliot-Curie - Skylake and Rome, runs were performed for Test Cases A and B, without and with postprocessing to check the influence of dumping files on the disk.

4.2.1 Performance Results

Runs were performed on fully occupied nodes of each machine, using Code_Saturne version 7.0 (official release at the time of the project). The mesh size depends on the test case considered, and the time-step is adapted to fulfil the code's stability requirements. One hundred time-steps are run for all the tests but the very large ones, where only 5 time-steps are run because of the cost of these jobs. The timings are computed as the averaged time per time-step (in seconds), over 97 of these 100 time-steps in order to only account for solver time and not initialisation nor IO.

Four of the machines, i.e. SuperMUC-NG, MareNostrum4, JUWELS, and Joliot-Curie - Skylake are made of nodes consisting of 48 cores each, whereas two of them, Hawk and Joliot-Curie - Rome have nodes of 128 cores each. It has been decided to first compare the timings of the first four machines and to identify the fastest of the four ones, then to compare the timings for the two last machines, and find the fastest of them, before conducting a node-to-node comparison between the two fastest machines of each group. Node-to-node comparison was preferred to core-to-core comparison, because this is how HPC centres nowadays usually allocated compute time.

4.2.1.1 Performance for Test Case A

The default partitioner, e.g. SFC Morton is used for this case. For all the machines, the tests are carried out using 1 to 16 nodes, all fully populated, using MPI only.

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
1	11.31	100	1	1
2	5.44	103	2.08	2
4	2.58	110	4.38	4
8	1.25	113	9.02	8
16	0.63	113	18.03	16

Table 18: Code_Saturne, Test Case A - SuperMUC-NG

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
1	11.87	100	1	1
2	5.83	102	2.04	2
4	2.91	102	4.08	4
8	1.46	102	8.15	8
16	0.89	83	13.31	16

Table 19: Code_Saturne, Test Case A - MareNostrum4

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
1	11.06	100	1	1
2	5.39	103	2.05	2
4	2.64	105	4.19	4
8	1.32	105	8.38	8
16	0.71	97	15.58	16

Table 20: Code_Saturne, Test Case A – JUWELS

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
1	11.56	100	1	1
2	5.64	102	2.05	2
4	2.82	102	4.10	4
8	1.44	100	8.01	8
16	0.88	82	13.11	16

Table 21: Code_Saturne, Test Case A - Joliot-Curie – Skylake

The four aforementioned tables present the results for nodes consisting of a maximum of 48 physical cores. On all the machines, the time-to-solution decreases when the number of MPI

tasks increases. The best time-to-solution and performance observed is for SuperMUC-NG, where a super-linear behaviour occurs, also going from 1 to 16 nodes.

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
1	5.37	100	1	1
2	2.46	109	2.18	2
4	1.52	88	3.53	4
8	0.60	111	8.92	8
16	0.49	69	10.96	16

Table 22: Code_Saturne, Test Case A - Joliot-Curie - Rome

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
1	4.93	100	1	1
2	2.23	111	2.21	2
4	0.96	129	5.15	4
8	0.52	119	9.55	8
16	0.44	71	11.30	16

Table 23: Code_Saturne, Test Case A - Hawk

The two aforementioned tables show that Hawk is the faster machine of the two, even if for both, the performance is very much reduced for 16 nodes, because the load per MPI task is very small (about 12,700 cells only per task), and therefore communication plays an important part.

Note that on Joliot-Curie - Rome, the performance observed on 4 nodes is not consistent with the one on 2 and 8 nodes, showing very poor efficiency. More tests have been conducted for the same 4-node case with the exact same settings, which showed that computing the pressure takes about 40% more time than expected in the presented case, most certainly due to bad communications between MPI tasks, just for this case.

A node-to-node (respectively core-to-core) time-to-solution comparison between Hawk and SuperMUC-NG shows a speed-up of about 2.29 (respectively 0.86) for 1 node (respectively core), which drops to 1.44 (respectively 0.54) for 16 nodes (respectively cores), both in favour of Hawk in case of the node-to-node comparison, showing that for Code_Saturne and Test Case A, the AMD nodes are faster.

4.2.1.2 Preparing for Larger Runs - Test Case B on SuperMUC-NG

A thorough analysis is carried out using from 8 to 128 nodes, to decide which partitioning tool translates into the best performance for Code_Saturne itself, between SFC Morton, SFC Hilbert, METIS, SCOTCH and PT-SCOTCH, for Test Case B. Code_Saturne gives the option to run a single simulation, where several partitions are created once for good by a given partitioner and stored into files, named as *domain_number_**, where "*" is the number of sub-domains. This gives the option to use the serial partitioners, METIS and SCOTCH, on fat nodes to take advantage of their large RAM. For all the tests carried out in this sub-section, the *domain_number_** files, corresponding to a given partitioner are generated beforehand, and then read at the start of each performance test. The five tables below show efficiency and speed-up for the various partitioners, and apart for PT-SCOTCH, increasing the number of nodes leads to a significant decrease in compute time per time-step and decent parallel performance. Overall, the best performance for this configuration (tetrahedral mesh), based on the solver

timings only, comes when the partitioning is serial. METIS provides the best timings, and also performance, with an efficiency of over 93% going from 8 to 128 nodes, whereas SFC Morton (respectively SFC Hilbert) are about 74% (respectively 75%).

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
8	13.19	100	1	1
16	6.46	102	2.04	2
32	3.41	97	3.87	4
64	1.89	87	6.96	8
128	1.09	75	12.04	16

Table 24: Code_Saturne, Test Case B - SFC Morton

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
8	12.98	100	1	1
16	6.46	100	2.01	2
32	3.35	97	3.87	4
64	1.78	91	7.31	8
128	1.09	74	11.88	16

Table 25: Code_Saturne, Test Case B - SFC Hilbert

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
8	12.10	100	1	1
16	5.88	103	2.06	2
32	2.99	101	4.05	4
64	1.51	100	8.01	8
128	0.81	94	15.01	16

Table 26: Code_Saturne, Test Case B – METIS

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
8	12.58	100	1	1
16	6.45	97	1.95	2
32	3.46	91	3.64	4
64	1.57	100	8.01	8
128	0.87	91	14.52	16

Table 27: Code_Saturne, Test Case B – SCOTCH

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
8	14.48	100	1	1
16	7.84	92	1.85	2
32	7.59	48	1.91	4
64	6.01	30	2.41	8
128	7.09	13	2.04	16

Table 28: Code_Saturne, Test Case B - PT-SCOTCH

The fact that METIS is serial is an issue to use it directly for Test Case C and D, and beyond, because these tests require a lot of RAM to create the sub-domain partitions. However, given the gain observed by using METIS over SFC Morton/Hilbert for Test Case B, it was decided

that for Test Case C and D, and beyond, the partitions would not be computed on the actual mesh using SFC Morton/Hilbert, but that the meshes would be generated by Mesh Multiplication (MM) from Test Case B's mesh, using the partitions generated by METIS for Test Case B's mesh. All the results presented below are such that the mesh for Test Case B is read, as well as its sub-domain partition for the given number of MPI tasks, and then several levels of Mesh Multiplication are applied for the larger cases (1 level for Test Case C, 2 levels for Test Case D and 3 levels for the largest case).

4.2.1.3 Performance for Test Case B

Very good performance is observed on all the machines up to 64 nodes (efficiency over 80%). However, for 128 nodes, only the runs on SuperMUC-NG and Hawk keep very good performance, with an efficiency of about 90%. For this number of nodes, running a node-to-node comparison between SuperMUC-NG and Hawk shows that the latter is twice as fast as the former.

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
8	12.10	100	1	1
16	5.88	103	2.06	2
32	2.99	101	4.05	4
64	1.51	100	8.01	8
128	0.81	94	15.01	16

Table 29: Code_Saturne, Test Case B – METIS - SuperMUC-NG

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
8	15.75	100	1	1
16	7.93	99	1.99	2
32	4.45	89	3.54	4
64	2.40	82	6.56	8
128	1.90	52	8.28	16

Table 30: Code_Saturne, Test Case B – METIS - MareNostrum4

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
8	12.03	100	1	1
16	5.96	101	2.02	2
32	3.20	94	3.76	4
64	1.70	89	7.08	8
128	1.08	70	11.13	16

Table 31: Code_Saturne, Test Case B – METIS – JUWELS

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
8	12.96	100	1	1
16	6.49	100	2.00	2
32	3.30	98	3.93	4
64	1.99	82	6.52	8
128	1.39	58	9.36	16

Table 32: Code_Saturne, Test Case B – METIS - Joliot-Curie – Skylake

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
8	6.32	100	1	1
16	2.87	110	2.20	2
32	1.38	115	4.59	4
64	0.72	109	8.74	8
128	0.60	66	10.55	16

Table 33: Code_Saturne, Test Case B – METIS - Joliot-Curie – Rome

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
8	5.75	100	1	1
16	2.57	112	2.24	2
32	1.19	121	4.84	4
64	0.68	106	8.51	8
128	0.40	90	14.35	16

Table 34: Code_Saturne, Test Case B – METIS - Hawk

4.2.1.4 Performance for Test Cases C and D

Test Case C (888M)

There are no results on MareNostrum4 because of shortage of resources. Simulations over 256 nodes of Joliot-Curie - Rome could not be completed because of issues with UCX. Again, the best results are obtained on SuperMUC-NG and Hawk, and for 1024 nodes, running on Hawk is 2.6 times faster than on SuperMUC-NG, if a node-to-node comparison is performed. Note that a core-to-core comparison would show a speed-up of 1 between both machines.

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
32	28.51	100	1	1
64	14.67	97.16	1.94	2
128	7.72	92.29	3.69	4
256	3.82	93.22	7.46	8
512	2.04	87.34	13.97	16
1024	1.17	76.02	24.33	32

Table 35: Code_Saturne, Test Case C – METIS + MM - SuperMUC-NG

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
32	28.77	100	1	1
64	15.00	96	1.92	2
128	8.50	85	3.38	4
256	5.18	69	5.56	8
512	3.40	53	8.47	16
1024	N/A	N/A	N/A	32

Table 36: Code_Saturne, Test Case C – METIS + MM – JUWELS

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
32	31.33	100	1	1
64	16.33	96	1.92	2
128	8.75	90	3.58	4
256	5.79	68	5.41	8
512	4.42	44	7.08	16
1024	N/A	N/A	N/A	32

Table 37: Code_Saturne, Test Case C – METIS + MM - Joliot-Curie – Skylake

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
32	16.10	100	1	1
64	7.99	101	2.01	2
128	4.17	97	3.87	4
256	N/A	N/A	N/A	8
512	N/A	N/A	N/A	16
1024	N/A	N/A	N/A	32

Table 38: Code_Saturne, Test Case C – METIS + MM - Joliot-Curie – Rome

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
32	14.83	100	1	1
64	7.33	101	2.02	2
128	3.50	106	4.24	4
256	1.74	107	8.53	8
512	0.90	103	16.52	16
1024	0.67	69	22.17	32

Table 39: Code_Saturne, Test Case C – METIS + MM - Hawk

Test Case D (7B)

The same trend is observed for this case as for Test Case C, Hawk being the machine where the code is running the fastest, in case of a node-to-node comparison. And for 2048 nodes, it is just over twice as fast as on SuperMUC-NG. However, a core-to-core comparison would show a speed-up of 0.77.

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
256	36.51	100	1	1
512	18.99	96.10	1.92	2
1024	9.85	92.67	3.71	4
2048	5.56	82.09	6.57	8
2500	4.84	77.27	7.55	9.77

Table 40: Code_Saturne, Test Case D – METIS + MM - SuperMUC-NG

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
256	38.51	100	1	1
512	20.01	96	1.92	2
1024	14.18	68	2.72	4

Table 41: Code_Saturne, Test Case D – METIS + MM – JUWELS

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
256	42.17	100	1	1
512	25.02	84	1.69	2

Table 42: Code_Saturne, Test Case D – METIS + MM - Joliot-Curie – Skylake

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
256	19.21	100	1	1
512	9.08	106	2.12	2
1024	4.58	105	4.19	4
2048	2.74	88	7.00	8

Table 43: Code_Saturne, Test Case D – METIS + MM - Hawk

4.2.1.5 Very Large Simulation on SuperMUC-NG and Hawk (56B)

The largest case is made of a mesh of about 56 billion cells (56B), by using the same strategy, e.g. Mesh Multiplication (3 levels) from Test Case B and corresponding partitions. On both machines, very good performance is achieved, with over 92% parallel efficiency on 4,096 nodes (524,288 cores) of Hawk.

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
2048	68.59	100	1	1
2500	56.87	99	1.21	1.22

Table 44: Code_Saturne, Very large case (56B) – METIS + MM - SuperMUC-NG

nodes	Time (s)	Efficiency	Speed-up (SU)	Ideal SU
1024	68.96	100	1	1
2048	34.77	99	1.98	2
4096	18.68	92	3.69	4

Table 45: Code_Saturne, Very large case (56B) – METIS + MM - Hawk

4.2.2 Energy Consumption – Comparison for Several Machines

Energy consumption is obtained on 3 machines: MareNostrum4, Joliot-Curie Skylake and Rome. It is given from the workload accounting logs by using the ‘sacct’ command or equivalent on these 3 machines. As expected, increasing the number of nodes also increases energy consumption. Table 46 shows it for Test Cases A, B and C. The time to solution (see sub-sections 4.2.1.1and 4.2.1.2) for Test Case A (respectively B) on MareNostrum4 is more than 10% (respectively 20%) expensive than on Joliot-Curie Skylake, but it requires about 40% (respectively about 100%) more energy. The time to solution for all the tests on Joliot-Curie Skylake is at least twice as big as their counterparts on Joliot-Curie Rome, but energy consumption is much less than twice as big, especially when the number of nodes increases. Unfortunately, there are no results for 256, 512, 1024 nodes on Joliot-Curie Rome for Test Case C to confirm or infirm the trend observed for Test Case B for the largest number of nodes (64 and 128), where Joliot-Curie Rome's simulations use more energy than Joliot-Curie Skylake ones, even if they are faster.

	nodes	MareNostrum4	Joliot-Curie Skylake	Joliot-Curie Rome
Test Case A	1	1,070	629	369
	2	1,080	607	356
	4	1,100	627	338
	8	1,120	657	436
	16	1,220	822	690
Test Case B	8	10,180	5,540	3,270
	16	10,220	5,510	3,170
	32	6,300	5,800	3,570
	64	12,090	7,230	7,600
	128	17,890	10,810	17,760
Test Case C	8	N/A	54,130	32,320
	16	N/A	55,470	35,150
	32	N/A	61,340	44,880

Table 46: Code_Saturne, Energy consumption in kJ - Comparison between machines

4.2.3 Energy Consumption – Comparison Without and With Output on the Disk

For Test Case A it happens that all the times to solution for the solver are smaller in case of outputting on the disk (W P cases) as shown in Table 47. This explains that the energy consumption for 1 to 4 nodes is smaller when outputting on the disk, as it does not seem to introduce any overhead. For 16 nodes however, there is a clear increase in energy consumption when writing on the disk (W P), for all the machines.

For Test Case B, the trend in energy consumption on 8 and 16 nodes is not easy to identify, but increasing the number of nodes to 32, 64 and 128 clearly shows the influence of writing on the disk, as for instance shown for the 128-node case: 33% more energy on MareNostrum4, 12% more on Joliot-Curie Skylake and 9% more on Joliot-Curie Rome.

	nodes	MareNostrum4		Joliot-Curie Skylake		Joliot-Curie Rome	
		N P	W P	N P	W P	N P	W P
Test Case A	1	1,070	994	629	600	370	353
	2	1,080	924	607	607	356	351
	4	1,100	946	627	635	338	504
	8	1,120	1,030	657	673	456	414
	16	1,220	1,300	821	846	690	797
Test Case B	8	10,180	12,340	5,540	5,650	3,270	3,380
	16	10,220	9,850	5,510	5,640	3,170	3,210
	32	6,300	7,540	5,800	6,150	3,570	3,800
	64	12,090	15,580	7,230	7,430	7,600	7,910
	128	17,890	23,940	10,810	12,120	17,760	19,280

Table 47: Code_Saturne, Energy consumption in kJ - Comparison without (N P) and with (W P) output on (postprocessing)

4.2.4 Conclusions

All the performance tests carried out on the six machines show that Code_Saturne scales very well, also up to over 0.5M MPI tasks on Hawk, for the 56B case, with a mesh of over 56 billion

cells, if good care is taken at the partitioning stage. Here, using METIS and Mesh Multiplication to generate the meshes from Test Case C on, proves to translate into better timings (and performance) than using directly SFC Morton/Hilbert for the largest meshes.

Among the 3 machines, where energy consumption was obtained, Joliot-Curie Rome is the one that shows best values for a small number of nodes, with respect to the mesh size, but Joliot-Curie Skylake is less energy demanding (even if slower than Joliot-Curie Rome), when the number of nodes is increased.

4.3 CP2K

4.3.1 Installation of CP2K

Version 8.1 of CP2K was used in all test cases. The GCC compilers are the recommended compilers for CP2K, with the most up to date versions being supported. Therefore, the GCC compilers were used for all machine builds. The MPI library used depends on the availability on each system; the MPI library must be compatible with the GCC compilers, and when possible, the system recommended MPI library is used. BLAS, LAPACK, and ScaLAPACK are required by default for MPI builds, for all systems these can be provided from a central system install either through MKL, OpenBLAS, or LibSci. FFTW is also required for good performance of FFTs, and again this is available centrally on most systems, however it is installed if there is not a GCC compatible version. Version 3.3.8.8 is used. Other optional libraries can be installed in order to improve the performance, and in this case ELPA (v2020-05) and libxsmm (v16.1) have been used. These offer improved performance for diagonalisation and matrix multiplication respectively. Libsxmm is used only on CPUs however ELPA is suitable for GPU and CPU builds, with GPU offloading of diagonalisation routines. Libint, which offers support for calculations of the Hartree-Fock exchange is installed as it is required for Test Case B.

The GPU architectures all use NVIDIA GPUs, and therefore CUDA was used to compile the accelerated code. The CP2K compile flags -D_ACC and -D__DBCSR_ACC enable accelerator support for matrix multiplications within CP2K's DBCSR library and the -D__PW_CUDA flag gives CUDA support for plane wave calculations.

Prior to starting the benchmarking, the builds were tested by running the CP2K test suite and doing a quick performance check to compare the performance with the centrally installed CP2K. In all cases the performance was similar or better than the central install.

4.3.2 Running Benchmarks

Runs were performed on fully occupied nodes of each machine. Hybrid MPI+OpenMP was used for each test case, with thread values set as to sensibly occupy the NUMA regions, ensuring no threads span multiple regions. For each test case multiple runs were performed where the number of OpenMP threads was varied. This allows us to find the optimum number of threads which gives the best performance in each case. Below we present results for the thread values which gave the best performance and show which configuration (number of threads per MPI task) this result was for.

Test Case B has an adjustable parameter to set the memory per process used in the HFX module. This value was set on each system to use the maximum amount of memory available.

For Piz Daint we compare the performance with and without usage of the GPUs. The runs were both performed on the XC50 GPU enabled compute nodes, however the CPU only build was compiled without any offloading to the GPU enabled.

For Test Case C out of memory errors are reported on some systems when running on low node counts. Where possible the run was repeated on high memory nodes, in other cases there is no result reported.

4.3.3 Benchmark Results

The results below show the best configuration of threads and processes which gave the shortest time to solution. For the energy consumption results the energy shown is for the corresponding run time result, however this may not be the lowest energy reported. Each run is performed three times and the average of the run time and the energy consumed is taken (apart from on MARCONI100 where some runs were not repeated due to budget constraints). The speed-up and parallel efficiency are reported with reference to the run time on a single node.

4.3.3.1 Performance on Hawk

Nodes	Best time (s)	Speed-up	Parallel efficiency	Configuration (threads per MPI task)
1	689.4	1	100	1
2	377.9	1.82	91.21	2
4	235.1	2.93	73.31	2
8	138.7	4.97	62.15	4
16	101.37	6.80	42.50	8
32	74.40	9.27	28.96	4
64	60.03	11.48	17.94	8

Table 48: CP2K, Test Case A, Hawk

Nodes	Best time (s)	Speed-up	Parallel efficiency	Configuration (threads per MPI task)
1	534.27	1	100	2
2	274.49	1.95	97.32	4
4	139.57	3.83	95.70	4
8	72.09	7.41	92.64	4
16	37.85	14.11	88.21	8
32	21.96	24.33	76.04	4
64	14.00	38.16	59.62	8
128	9.80	54.50	42.58	8
256	10.09	52.93	20.68	16

Table 49: CP2K, Test Case B, Hawk

Nodes	Best time (s)	Speed-up	Parallel efficiency	Configuration (threads per MPI task)
1	588.36	1	100	8
2	329.68	1.78	89.23	8
4	183.75	3.20	80.05	8
8	98.27	5.99	74.84	4
16	54.11	10.87	67.95	8
32	34.35	17.12	53.52	8
64	25.51	23.06	36.03	16
128	19.77	29.77	23.26	8
256	19.54	30.11	11.76	8

Table 50: CP2K, Test Case C, Hawk

4.3.3.2 Performance on Irene-Rome (Joliot-Curie)

Nodes	Best time	Speed-up	Parallel efficiency	Energy consumed (kJ)	Configuration (threads per MPI task)
1	663.97	1	100	145	1
2	380.05	1.75	87.35	421	2
4	230.88	2.87	71.90	503	2
8	169.72	3.91	48.90	639	4
16	134.97	4.92	30.75	1353	2
32	105.99	6.26	19.58	1684	4

Table 51: CP2K, Test Case A, Irene-Rome

Nodes	Best time	Speed-up	Parallel efficiency	Energy consumed (kJ)	Configuration (threads per MPI task)
1	499.04	1	100	310	2
2	255.38	1.95	97.71	330	2
4	130.55	3.82	95.56	353	2
8	68.37	7.30	91.24	369	2
16	37.60	13.27	82.96	370	4
32	23.87	20.90	65.32	470	4
64	16.42	30.39	47.49	653	16
96	14.37	34.71	36.16	859	16

Table 52: CP2K, Test Case B, Irene-Rome

Nodes	Best time	Speed-up	Parallel efficiency	Energy consumed (kJ)	Configuration (threads per MPI task)
1	593.32	1	100	359	8
2	334.80	1.77	88.61	402	4
4	191.56	3.10	77.43	426	4
8	109.12	5.44	67.97	505	4
16	69.50	8.54	53.36	575	8
32	49.24	12.05	37.65	747	16
64	37.83	15.68	24.51	1185	8
96	31.71	18.71	19.49	1562	8

Table 53: CP2K, Test Case C, Irene-Rome

4.3.3.3 Performance on JUWELS

Nodes	Best time (s)	Speed-up	Parallel efficiency	Configuration (threads per MPI task)
1	1347.24	1.00	100.00	2
2	833.10	1.62	80.86	1
4	419.01	3.22	80.38	2
8	258.91	5.20	65.04	1
16	176.75	7.62	47.64	2
32	166.10	8.11	25.35	4
64	126.92	10.62	16.59	12

Table 54: CP2K, Test Case A, JUWELS

Nodes	Best time (s)	Speed-up	Parallel efficiency	Configuration (threads per MPI task)
1	1179.99	1.00	100.00	4
2	596.18	1.98	98.96	2
4	304.84	3.87	96.77	4
8	157.20	7.51	93.83	4
16	80.44	14.67	91.69	2
32	42.32	27.88	87.13	12
64	23.82	49.54	77.41	24
128	15.48	76.21	59.54	24
256	11.17	105.66	41.28	24
512	15.11	78.08	15.25	24

Table 55: CP2K, Test Case B, JUWELS

Nodes	Best time (s)	Speed-up	Parallel efficiency	Configuration (threads per MPI task)
1	941.58	1.00	100.00	4
2	525.80	1.79	89.54	4
4	291.88	3.23	80.65	4
8	157.83	5.97	74.57	4
16	101.34	9.29	58.07	4
32	56.93	16.54	51.68	24
64	42.89	21.95	34.30	12
128	27.74	33.95	26.52	24
256	21.34	44.13	17.24	24

Table 56: CP2K, Test Case C, JUWELS

4.3.3.4 Performance on MareNostrum4

Nodes	Best time (s)	Speed-up	Parallel efficiency	Energy consumed (kJ)	Configuration (threads per MPI task)
1	1511.23	1.00	100.00	1322	2
2	830.96	1.82	90.93	1426	1
4	478.20	3.16	79.01	1357	2
8	306.75	4.93	61.58	1911	2
16	196.86	7.68	47.98	2107	2
32	154.59	9.78	30.55	2733	4
64	138.77	10.89	17.02	4520	4

Table 57: CP2K, Test Case A, MareNostrum4

Nodes	Best time (s)	Speed-up	Parallel efficiency	Energy consumed (kJ)	Configuration (threads per MPI task)
1	1856.62	1.00	100.00	961	4
2	934.07	1.99	99.38	1019	4
4	476.75	3.89	97.36	1063	12
8	246.50	7.53	94.15	1239	2
16	125.39	14.81	92.54	1234	4
32	66.19	28.05	87.66	1154	12
64	36.06	51.48	80.44	1226	12
128	23.38	79.41	62.04	1560	12
192	17.74	104.68	54.52	1354	24

Table 58: CP2K, Test Case B, MareNostrum4

Nodes	Best time (s)	Speed-up	Parallel efficiency	Energy consumed (kJ)	Configuration (threads per MPI task)
1	1133.11	1.00	100.00	1188	4
2	619.03	1.83	91.52	1013	4
4	335.35	3.38	84.47	1134	2
8	180.57	6.28	78.44	1265	4
16	106.85	10.60	66.28	1293	4
32	61.54	18.41	57.54	1523	4
64	43.35	26.14	40.84	1682	12
128	32.63	34.72	27.13	2301	12
192	31.24	36.27	18.89	3208	12

Table 59: CP2K, Test Case C, MareNostrum4

4.3.3.5 Performance on MARCONI100

Nodes	Best time (s)	Speed-up	Parallel efficiency	Configuration (threads per MPI task)
2	3959.804	1.00	100.00	16
4	2098.29	1.89	94.36	8
8	1198.94	3.30	82.57	8
16	785.5965	5.04	63.01	16
32	525.29	7.54	47.11	8

Table 60: CP2K, Test Case A, MARCONI100

Nodes	Best time (s)	Speed-up	Parallel efficiency	Configuration (threads per MPI task)
2	1492.34	1.00	100.00	8
4	639.03	2.34	116.77	16
8	326.56	4.57	114.25	16
16	215.67	6.92	86.50	8
32	105.59	14.13	88.33	16
64	67.76	22.02	68.82	16
128	41.82	35.68	55.75	16
256	33.93	43.98	34.36	16

Table 61: CP2K, Test Case B, MARCONI100

Nodes	Best time (s)	Speed-up	Parallel efficiency	Configuration (threads per MPI task)
2	253.13	1.00	100.00	8
4	153.19	1.65	82.62	16
8	98.92	2.56	63.97	16
16	66.93	3.78	47.28	8
32	43.08	5.88	36.72	16
64	31.15	8.13	25.39	16
128	21.67	11.68	18.25	8
192	19.04	13.30	13.85	4

Table 62: CP2K, Test Case C, MARCONI100

4.3.3.6 Performance on Piz Daint GPU

Nodes	Best time (s)	Speed-up	Parallel efficiency	Energy consumed (kJ)	Configuration (threads per MPI task)
1	3261.21	1.00	100.00	699	2
2	1716.09	1.90	95.02	798	1
4	952.85	3.42	85.56	808	3
8	588.73	5.54	69.24	942	3
16	338.87	9.62	60.15	1079	3
32	255.01	12.79	39.96	1544	3
64	135.68	24.04	37.56	1733	3
128	114.26	28.54	22.30	2645	6

Table 63: CP2K, Test Case A, Piz Daint GPU

Nodes	Best time (s)	Speed-up	Parallel efficiency	Energy consumed (kJ)	Configuration (threads per MPI task)
1	5300.28	1	100	1063	3
2	2673.49	1.98	99.13	1058	3
4	1375.68	3.85	96.32	1084	6
8	712.25	7.44	93.02	1120	3
16	382.18	13.87	86.70	1235	1
32	199.81	26.53	82.90	1246	6
64	104.89	50.53	78.96	1594	6
128	59.22	89.50	69.92	1552	6
256	37.46	141.48	55.27	1971	12
512	34.86	152.05	29.70	3758	3

Table 64: CP2K, Test Case B, Piz Daint GPU

Nodes	Best time (s)	Speed-up	Parallel efficiency	Energy consumed (kJ)	Configuration (threads per MPI task)
8	240.71	1.00	100.00	458	2
16	145.02	1.66	82.99	538	2
32	95.12	2.53	63.27	632	3
64	60.70	3.97	49.57	891	2
128	42.67	5.64	35.26	1037	12
256	31.89	7.55	23.59	1772	6
512	24.24	9.93	15.52	4300	3

Table 65: CP2K, Test Case C, Piz Daint GPU

4.3.3.7 Performance on Piz Daint CPU

Nodes	Best time (s)	Speed-up	Parallel efficiency	Energy consumed (kJ)	Configuration (threads per MPI task)
1	5121.66	1.00	100.00	1044	1
2	2552.14	2.01	100.34	1041	1
4	1430.27	3.58	89.52	1199	1
8	783.48	6.54	81.71	1280	1
16	447.41	11.45	71.55	1512	1
32	246.92	20.74	64.82	1514	3
64	142.97	35.82	55.97	1790	3
128	123.25	41.55	32.46	2831	3

Table 66: CP2K, Test Case A, Piz Daint CPU

Nodes	Best time (s)	Speed-up	Parallel efficiency	Energy consumed (kJ)	Configuration (threads per MPI task)
1	5260.95	1.00	100.00	1023	1
2	2654.45	1.98	99.10	1017	1
4	1362.65	3.86	96.52	1064	1
8	693.89	7.58	94.77	1079	1
16	353.36	14.89	93.05	1051	6
32	180.29	29.18	91.19	1191	1
64	94.86	55.46	86.65	1161	6
128	50.15	104.89	81.95	1279	3
256	27.31	192.67	75.26	2238	1
512	22.94	229.36	44.80	3314	3
1024	25.12	209.45	20.45	7512	2

Table 67: CP2K, Test Case B, Piz Daint CPU

Nodes	Best time (s)	Speed-up	Parallel efficiency	Energy consumed (kJ)	Configuration (threads per MPI task)
8	718.95	1.00	100.00	1187	1
16	387.39	1.86	92.79	1299	2
32	201.14	3.57	89.36	1368	1
64	109.23	6.58	82.28	1463	2
128	63.46	11.33	70.80	1531	12
256	38.79	18.53	57.92	2664	2
512	23.79	30.22	47.23	3547	2

Table 68: CP2K, Test Case C, Piz Daint CPU

4.3.3.8 Performance on SuperMUC-NG

Nodes	Best time (s)	Speed-up	Parallel efficiency	Energy consumed (kJ)	Configuration (threads per MPI task)
1	1287.93	1.00	100.00	493	2
2	681.04	1.89	94.56	561	1
4	419.60	3.07	76.74	606	2
8	249.53	5.16	64.52	778	1
16	165.82	7.77	48.55	883	2
32	136.47	9.44	29.49	1241	4
64	120.02	10.73	16.77	2005	4

Table 69: CP2K, Test Case A, SuperMUC-NG

Nodes	Best time (s)	Speed-up	Parallel efficiency	Energy consumed (kJ)	Configuration (threads per MPI task)
1	1773.00	1.00	100.00	503	1
2	904.64	1.96	97.99	527	2
4	456.54	3.88	97.09	537	1
8	236.96	7.48	93.53	569	1
16	122.74	14.45	90.29	565	4
32	65.33	27.14	84.82	605	4
64	35.89	49.41	77.20	642	24
128	23.08	76.84	60.03	791	12
256	13.52	131.13	51.22	No value	24
512	11.06	160.32	31.31	No value	24

Table 70: CP2K, Test Case B, SuperMUC-NG

Nodes	Best time (s)	Speed-up	Parallel efficiency	Energy consumed (kJ)	Configuration (threads per MPI task)
1	963.94	1.00	100.00	545[1]	4
2	549.52	1.75	87.71	441	4
4	303.43	3.18	79.42	468	4
8	165.04	5.84	73.01	513	4
16	97.40	9.90	61.86	589	4
32	58.31	16.53	51.66	693	4
64	41.62	23.16	36.19	883	12
128	31.28	30.82	24.08	1306	12

Table 71: CP2K, Test Case C, SuperMUC-NG

4.3.4 Performance Comparison

The results presented in this section are for the run time of the best performing configuration of threads and processes at that particular node count on each system. Hence, we present a comparison of the most optimal runs on each system for each test case. Figure 2 shows a performance comparison of the systems for Test Case A.

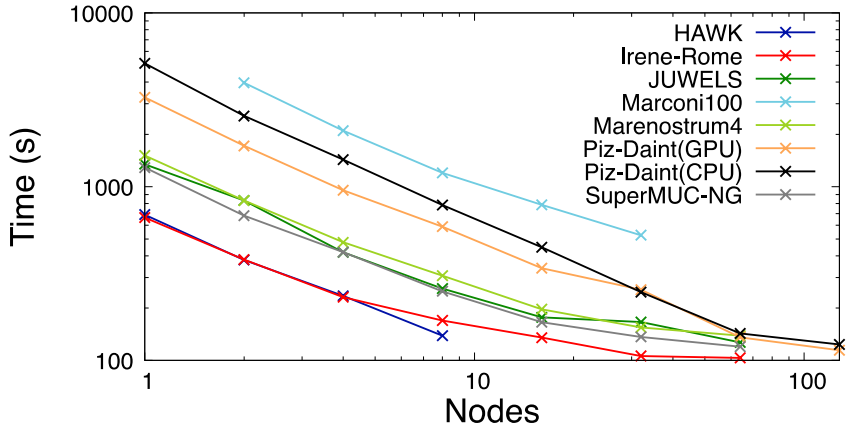

Figure 2: CP2K run times for Test Case A.

The figure illustrates the effect on the run time due to the differing processor type and cores per node across the different systems. Irene-Rome and Hawk which have two 64 core AMD Rome EPYC processors per node have similar results at low node counts and are the best performing systems for this test case on a node-for-node comparison. Hawk slightly outperforms Irene-Rome at higher node counts due to better performing MPI calls at this scale (as shown in the CP2K log files) owing to its higher interconnect bandwidth. JUWELS, MareNostrum4 and SuperMUC-NG, which each have two 24 core Intel Xeon Skylake processors per node have similar performance with SuperMUC-NG outperforming slightly are higher node counts. For Piz Daint, where we have reported the performance on its 12 core Intel processors with an NVIDIA GPU but run with and without offloading to the GPU itself, the results show that offloading to the GPUs is mostly optimal, however at high node counts the CPU and GPU results are similar. MARCONI100, which has 4 V100 GPUs per node, shows worse performance than the other systems across all node counts. From looking at the CP2K logs this

[1] This result was performed on the high memory nodes due to the memory requirements.

appears to be because this test case is dominated by the diagonalisation routines which were not offloaded to GPU. On MARCONI100 it is suggested to run with 4 MPI processes per node (1 per GPU), and therefore without good GPU utilisation performance will be reduced.

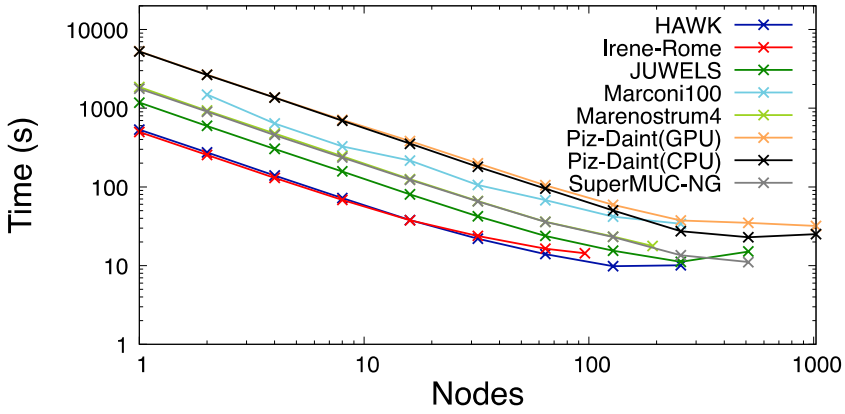

Figure 3: CP2K run times for Test Case B.

Figure 3 shows the performance for Test Case B. Most of the systems show good scaling up to 128 nodes with a parallel efficiently around 70%. As with Test Case A, there is similarity between the performance of Hawk and Irene-Rome (which have the same processor architecture), with Irene-Rome doing better at low node counts and Hawk slightly outperforming Irene-Rome at higher node counts. Of the Intel Xeon Skylake systems (JUWELS, SuperMUC-NG and MareNostrum4) JUWELS is the best performing with MareNostrum4 and SuperMUC-NG having similar performance. For Piz Daint the CPU and GPU performance is similar. From the CP2K log files it can be seen that this test case is dominated by the computation of the Hartree Fock exchange which is currently handled by the Libint library and not offloaded to GPU. MARCONI100 shows better performance than Piz Daint for this test case. This may be due to the large amount of memory available per process when running on MARCONI100. There is 242 GB per node, which is divided amongst the 4 processes used per node to give around 60 GB per process. The performance of the HFX calculation in this test case is affected by the memory assigned to it through the MAX_MEMORY input parameter which is set to 55 GB in this case.

Figure 4 shows the performance for Test Case C on the different Tier-0 systems. The key operation in this test case in matrix-matrix multiplication, which is handled in the DBCSR library and can be offloaded to GPU. As a result of this the performance of GPU-based systems is much improved compared to the other test cases, MARCONI100 is among the best performing systems across all node counts, and the GPU build of Piz Daint clearly outperforms the CPU version. For the Cpu-based systems there is a similar trend to the other test cases, with there being a gap between the 128 core AMD EPYC systems and the smaller 48 core Intel Xeon systems.

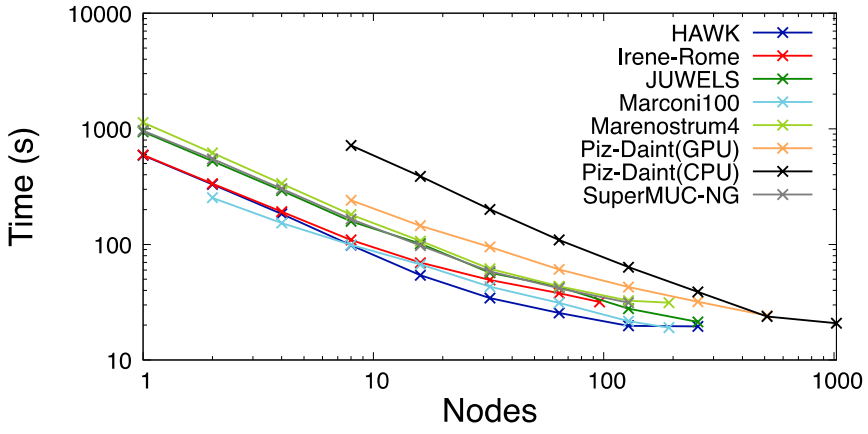

Figure 4: CP2K run times for Test Case C.

In general, the systems with larger nodes performed best across the three test cases when taking a node-for-node comparison. Hawk performed slightly better than Irene-Rome at higher node counts, perhaps owning to its larger interconnect bandwidth. The three Intel Xeon Skylake systems (JUWELS, SuperMUC-NG and MareNostrum4) had similar performance for the test cases, with JUWELs exceeding the performance of the other two systems in Test Case B and C. For Piz Daint offloading to GPU was more performant than CPU for Test Case A and C, however this effect diminishes at higher node counts due to the increased contribution of MPI communications. MARCONI100 was the worst performing for Test Case A but among the best performing for Test Case C. The main computation effort in Test Case C is handled by the DBCSR library, which is offloaded to GPU, and therefore the GPU-based systems perform well in this test case. Overall, the performance of the GPU-based systems is dependent on the test case.

4.3.5 Threading Options

As previously mentioned for each system we run each test case with various choices for the number of threads and then select the best performing configuration at each node count. These are summarised in Table 72 for Test Case A and Table 73 for Test Case C.

Nodes	Hawk	Irene-Rome	SuperMUC-NG	MareNostrum4	JUWELS	Piz Daint GPU	Piz Daint CPU	MARCONI100
1	1	1	2	2	2	2	1	-
2	2	2	1	1	1	1	1	16
4	2	2	2	2	2	3	1	8
8	4	4	1	2	1	3	1	8
16	8	2	2	2	2	3	1	16
32	4	4	4	4	4	3	3	16
64	8	8	4	4	12	3	3	-

Table 72: CP2K – the optimum number of threads for Test Case A.

Nodes	Hawk	Irene-Rome	SuperMUC-NG	MareNostrum4	JUWELS	Piz Daint GPU	Piz Daint CPU	MARCONI100
1	8	8	4	4	4	-	-	-
2	8	4	4	4	4	-	-	16
4	8	4	4	2	4	-	-	16
8	4	4	4	4	4	2	1	16
16	8	8	4	4	4	2	2	16
32	8	16	4	4	24	3	1	16
64	16	8	12	12	12	2	2	16
128	8	8	12	12	24	12	12	8

Table 73: CP2K – the optimum number of threads for Test Case C.

Most systems show the trend of the optimum number of threads increasing when run on more nodes. This can be explained by the improved performance of MPI calls when running on more threads. Multi-threading usually allows for less inter-node messages. The tables also show that the systems with similar processors typically have the similar values for the optimum number of threads. This can be seen when comparing Hawk and Irene-Rome, and SuperMUC-NG, MareNostrum4 and JUWELS. The GPU-based systems tend to perform better with more threads per process. For MARCONI100 it is recommended to use 4 processes per node and therefore more threads can utilise more of the cores. It is worth noting that when using 16 threads there are 2 threads running on a core.

4.3.6 Energy Consumption Comparison

In this section we report the energy consumed by running each of the CP2K test cases on the Tier-0 systems. In each case the total energy for running the job on the system is given from the workload accounting logs by using the 'sacct' command or similar. This energy includes contributions from both the node energy and the switch energy. Unfortunately, some of the systems do not report this energy, hence we present results only for Piz Daint, Irene-Rome, SuperMUC-NG and MareNostrum4. It is worth noting that for Irene-Rome in some runs the energy was not reported or gave an excessively high value, so these runs were repeated.

The total energy consumed for running Test Case A is shown in Figure 5. In all cases the energy consumption increases when running on more nodes. When comparing the Piz Daint CPU and GPU builds, we can see that the energies are similar, however the GPU requires less energy overall. This is likely since the run time when using the GPU is shorter, and in this case the energy consumption is proportional to the performance for both builds. SuperMUC-NG and MareNostrum4 had similar run times for this test case, however the energy consumption for MareNostrum4 is over 2 times larger. Irene-Rome has the lowest energy used at smaller node counts, but this increases the most when going to larger node counts. This was among the best performing systems for this test case.

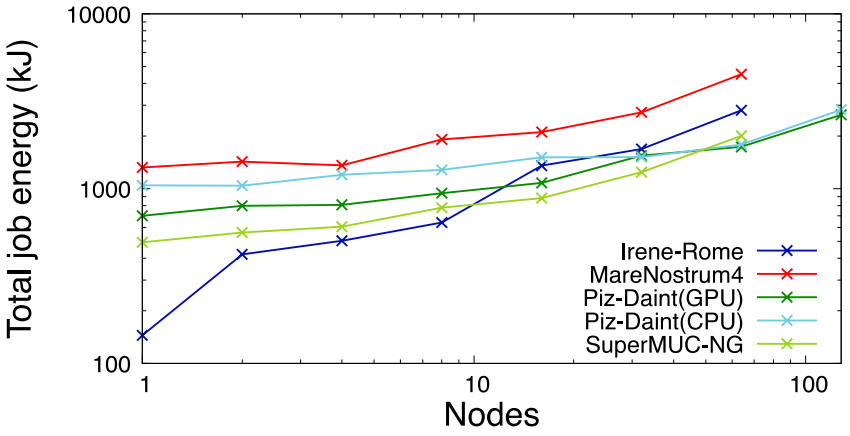

Figure 5: CP2K energy consumption for Test Case A.

Figure 6 shows the energy consumed for Test Case B. For this test case for most systems the energy used increases slowly up to around 64 nodes and then more rapidly above this. For Piz Daint the energy for the GPU and CPU build is similar, reflecting the almost matching performance for these builds with this test case. MareNostrum4 again has a higher energy consumption than SuperMUC-NG despite having the same processor and showing similar performance for this test case. Irene-Rome uses less energy than the other machines, due to its better performance.

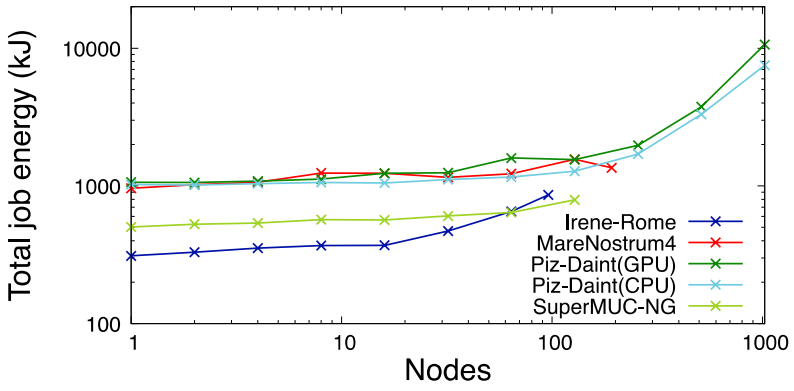

Figure 6: CP2K energy consumption for Test Case B.

Figure 7 shows the energy consumed for Test Case C. Here the energy used by Irene-Rome, SuperMUC-NG and Piz Daint GPU are similar with these consuming less energy than MareNostrum4 and Piz Daint CPU in most cases. The energy results for Piz Daint again reflect the performance with the energy used per second being similar for both builds. This shows that this test case benefits from being run on the GPU.

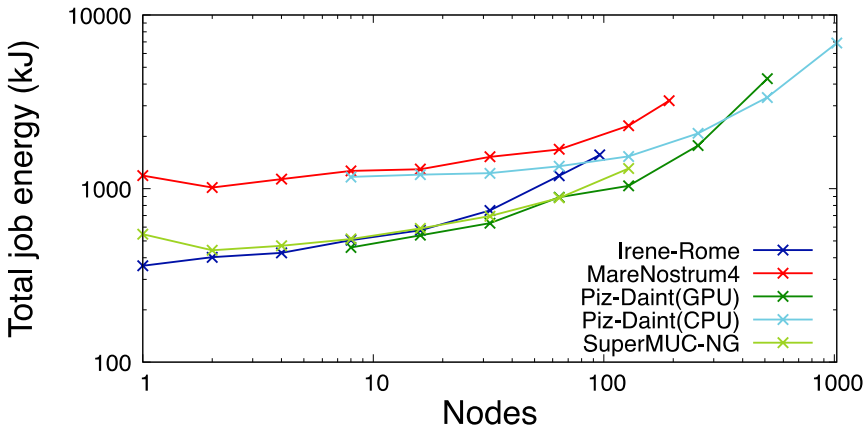

Figure 7: CP2K energy consumption for Test Case C.

Overall MareNostrum4 consumed the most energy in all three test cases. Irene-Rome consumed least energy in most cases and was one of the better performing machines when taking a node-for-node comparison. The energy consumed by the GPU and CPU builds on Piz Daint is proportional to the run time of the builds, with the better performing GPU build using less energy.

4.3.7 Energy Usage Considerations

The energy consumption results presented are shown for the best performing configuration of threads and processes. However, considering the global need to reduce energy usage, and that some HPC centres are now beginning to charge based on energy used rather than on core hours used, it may be worth finding the configuration with the lowest overall energy consumption instead. Therefore, in this section we will look at the impact of choosing the configuration which uses the least energy rather than the best performing.

As an example, Table 74 shows the run times and energy consumption for the best performing and lowest energy consuming configurations for Test Case C on Piz Daint GPU.

Nodes	Best performance			Lowest energy			Energy ratio	Run time ratio
	Energy (kJ)	Run time (s)	Config (threads per MPI task)	Energy (kJ)	Run time (s)	Config (threads per MPI task)		
8	457.57	240.71	2	433.20	241.72	3	0.95	1.00
16	538.22	145.02	2	506.19	147.42	3	0.94	1.02
32	632.15	95.12	3	596.65	97.64	6	0.94	1.03
64	891.17	60.70	2	729.49	61.86	12	0.82	1.02
128	1036.90	42.67	12	1036.90	42.67	12	1.00	1.00
256	1772.32	31.89	6	1772.32	31.89	6	1.00	1.00
512	4300.24	24.24	3	2970.91	26.77	12	0.69	1.10

Table 74: CP2K – run times and energy consumption for Test Case C on Piz Daint GPU.

The table shows that the lowest energy configuration typically has more threads per process than the best performing configuration. In most cases there is only a slight reduction in energy used when choosing the lowest energy configuration (and in some cases the lowest energy and

best performing configurations are the same). However, some results show it is possible to reduce the energy used by 20–30% with only a small negative effect on the performance. On 512 nodes running on 12 threads per process rather than 3 threads per process reduces the energy by 30%, for only a 10% increase in the run time. In some circumstances, it may be preferable to choose this configuration.

4.3.8 Conclusions

In summary, we investigated the performance and energy consumption of three CP2K test cases on the PRACE Tier-0 systems. When taking a node-for-node comparison of the performance it was seen that Hawk and Irene-Rome (which have two 64 core AMD EPYC processors per node) performed better than the other systems, as they have more cores per node. However, these are less performant when taking a core-for-core comparison. The three Intel Xeon Skylake systems had similar performance in most cases, however, JUWELS had better performance in Test Case B and C. The performance of the GPU-based systems was shown to be dependent on the test case. For Test Case C GPU offloading was shown to be advantageous but this was not the case for Test Case B.

The energy consumption was mostly reflective of the run time, with the best performing runs using less energy. However, MareNostrum4 consumed more energy than SuperMUC-NG despite having similar performance. We also briefly investigated how changing the number of threads might affect the energy consumption, and whether it could be advantageous to choose the lowest energy configuration at a slight loss to performance. It was shown that this may be possible in some cases, but this would require more detailed investigation.

4.4 GADGET

4.4.1 System Software Environment

The GADGET-4 benchmarks have been performed on systems with Skylake architecture using the Intel Platinum CPU (Irene-SKL, JUWELS, and MareNostrum4).

GADGET-4 requires a C++ compiler (C++11 standard), Message Passing Interface (MPI) version 3.0 or higher, the GNU scientific library (GSL), the Fastest Fourier Transform in the West (FFTW3), the Hierarchical Data Format version 5 (HDF5), and the hardware locality library (hwloc). In addition, the code requires the Vector Class Library by Agner Fog for explicit vectorisation via the AVX instruction set is enabled.

The FFTW3 library is not explicitly required (it makes no difference whether it is available or not as GADGET-4 implements its own communication routines when MPI is used). FFTW is only needed for simulations that use the TreePM algorithm, or if power spectra are estimated, or cosmological ICs are created. The hwloc library is useful for allowing the code to probe the processor topology it is running on and enable a pinning to individual cores. This library is optional as many MPI libraries nowadays enable pinning by default. The code also makes use of GNU make and Python as part of its build process.

The software versions used in each machine are displayed in Table 75.

Machine	C++ compiler	MPI flavour	FFTW3	GSL	HDF5	Hwloc
Irene-SKL	Intel 2020.2	Intel MPI 2020.0	3.3.8	2.6	1.10.1	2.2.0
JUWELS	Intel 2020.2	Intel MPI 2021.2	3.3.8	2.6	1.10.6	2.4.1
MareNostrum4	Intel 2019.5	Intel MPI 2018.4	3.3.8	2.7	1.10.5	2.0.0

Table 75: Software versions used in GADGET-4 benchmarks

4.4.2 Code Compilation and Extra MPI Tasks for Incoming Communications

After setting up the software environment through modules, GADGET-4 is compiled from its top-level directory using GNU make and the configuration file (Config.h) that contains the compile-time options. As the location and versions of the C++ compiler and libraries needed to run the code vary among different machines, the Makefile is divided into 4 files: (i) the Makefile which should not be changed in any significant way, (ii) a Makefile.systype file in which the system type is declared, (iii) a Makefile.comp file with the compilation flags, and (iv) a Makefile.path file with the lib and include paths of FFTW3, GSL, HDF5, hwloc, and vectorclass. Note that the latter library is bundled with the GADGET-4 source. Thus, the user only needs to adapt these files to the machine being used and the code compilation is straight forward. The code does not need to be recompiled for a different number of cores, or for a different problem size. However, when using multi nodes, there is the need to include an extra MPI task to handle for asynchronously serving incoming communication requests from other nodes.

4.4.3 Setup of the Runs

In order to study the scalability of the software two approaches were considered:

a) A core-based performance analysis where 1 MPI task per core, 16 cores per socket, that is 16 MPI tasks per socket, and 1 extra core per compute node to handle communications when multiple compute nodes were used. For the runs on a single node (that is with the number of cores varying between 1 and 32) no extra core was considered.
b) A node-based performance analysis where 1 MPI task per core, and all cores in the socket, that is 24 MPI tasks per socket, including an extra core for MPI communications when multiple nodes are used. For runs on a single node there is no need to use an extra core for communications.

In both setups the compute nodes were used with exclusivity. These approaches allow us to identify which setup provides the better performance for the GADGET-4 code.

4.4.4 Performance Results

Test Case A (Cosmological dark matter-only simulation). The timings, speed-up, and parallel efficiency of this test measured in JUWELS and MareNostrum4 are displayed in Table 76 and Table 77, while the energy consumption is shown in Table 78. The energy measurements were obtained using the sacct command with the field variable ConsumedEnergy.

# Nodes	# Cores	# Cores for Comm	JUWELS		
			Timings [s]	Speed-up	Parallel Efficiency [%]
1	8	0	5062	1.00	100.0
1	16	0	2592	1.95	97.6
1	32	0	1425	3.55	88.8
2	64	2	842	6.01	75.1
4	128	4	399	12.69	79.3
8	256	8	234	21.63	67.6
16	512	16	150	33.75	52.7
32	1024	32	118	42.90	33.5
64	2048	64	120	42.18	16.5
128	4096	128	221	22.91	4.5

Table 76: Timings, speed-up, and parallel efficiency of Test Case A of GADGET-4 on JUWELS

# Nodes	# Cores	# Cores for Comm	MareNostrum4		
			Timings [s]	Speed-up	Parallel Efficiency [%]
1	8	0	7762	1.00	100.0
1	16	0	4037	1.92	96.1
1	32	0	2179	3.56	89.1
2	64	2	1252	6.20	77.5
4	128	4	577	13.45	84.1
8	256	8	328	23.67	74.0
16	512	16	211	36.79	57.5
32	1024	32	164	47.33	37.0
64	2048	64	156	49.76	19.4
128	4096	128	160	48.51	9.5

Table 77: Timings, speed-up, and parallel efficiency of Test Case A of GADGET-4 on MareNostrum4

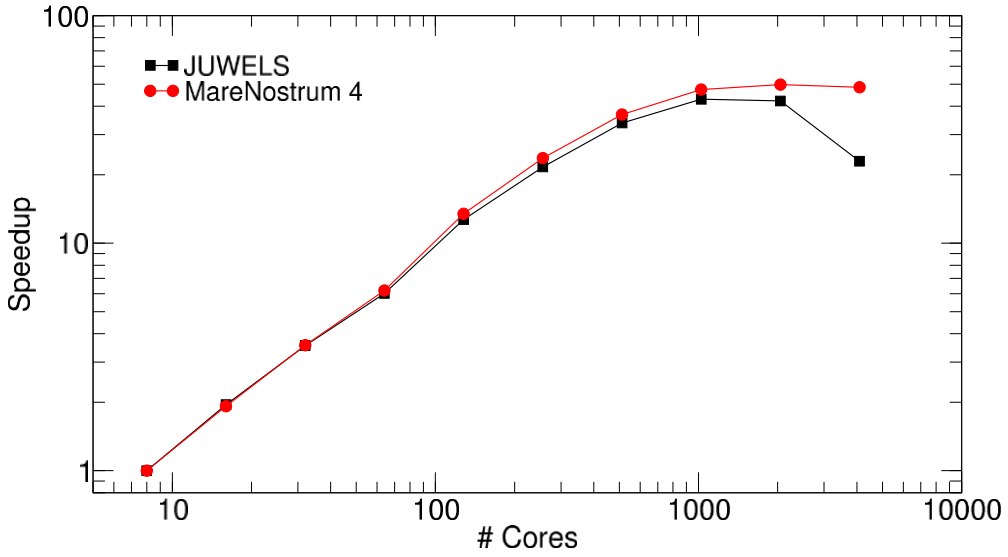

Figure 8: Speed-up comparisons for Test Case A with GADGET-4 on JUWELS and MareNostrum4.

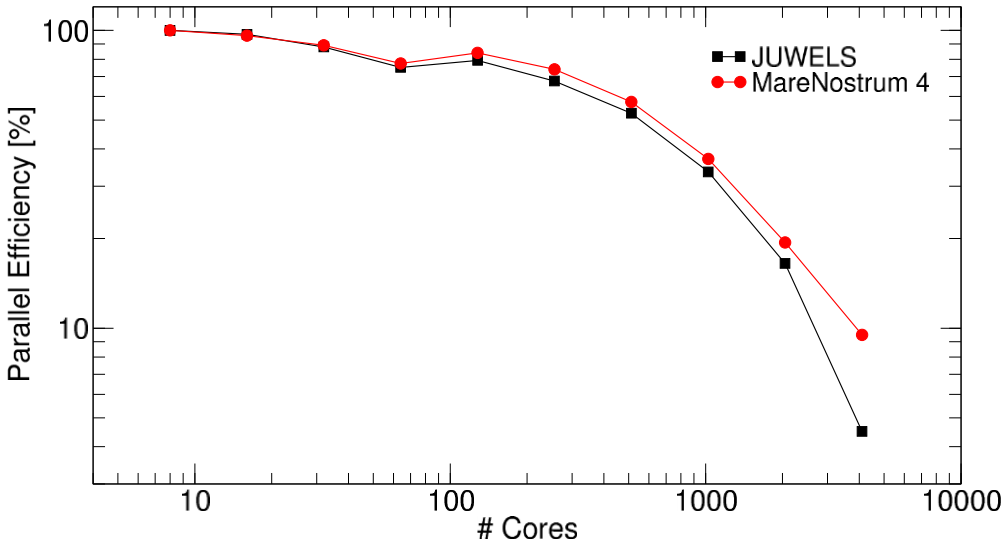

Figure 9: Parallel efficiency for Test Case A with GADGET-4 on JUWELS and MareNostrum4.

# Nodes	# Cores	# Cores for Comm	MareNostrum4 Energy [kJ]
1	8	0	2810.0
1	16	0	1600.0
1	32	0	526.3
2	64	2	643.3
4	128	4	588.6
8	256	8	684.7
16	512	16	854.2
32	1024	32	1260.0
64	2048	64	2230.0
128	4096	128	4880.0

Table 78: Energy measurements for GADGET-4 Test Case A on MareNostrum4.

Test Case B (The Blob test). The core-based and node-based performance approaches were used in this test and are named "Test Case B-c" and "Test Case B-n", respectively. The timings, speed-up, and parallel efficiency of this test measured in Irene-SKL, JUWELS, and MareNostrum4 are displayed in Table 79 and Table 80.

Test Case B – c (Core-based approach)

Benchmark	#Nodes	#Cores	#Cores for comm.	Total #cores	Irene-SKL		
					Timings [s]	Speed-up	Parallel Efficiency [%]
Test Case B - c	1	2	0	2	9064	1.00	100.0
Test Case B - c	1	4	0	4	4962	1.83	91.3
Test Case B - c	1	8	0	8	2452	3.70	92.4
Test Case B - c	1	16	0	16	1298	6.98	87.3
Test Case B - c	1	32	0	32	749	12.10	75.6
Test Case B - c	2	64	2	66	439	20.65	64.5
Test Case B - c	4	128	4	132	299	30.31	47.4
Test Case B - c	8	256	8	264	273	33.20	25.9
Test Case B - c	16	512	16	528	267	33.95	13.3
Test Case B - c	32	1024	32	1056	422	21.48	4.2
Test Case B - c	64	2048	64	2112	1031	8.79	0.9

Table 79: Timings, speed-up and parallel efficiency of Test Case B-c (core-based approach) obtained with GADGET-4 on Irene-SKL.

Benchmark	#Nodes	#Cores	#Cores for comm.	Total #cores	JUWELS		
					Timings [s]	Speed-up	Parallel Efficiency [%]
Test Case B - c	1	2	0	2	8184	1.00	100.0
Test Case B - c	1	4	0	4	4540	1.80	90.1
Test Case B - c	1	8	0	8	2381	3.44	85.9
Test Case B - c	1	16	0	16	1255	6.52	81.5
Test Case B - c	1	32	0	32	672	12.18	76.1
Test Case B - c	2	64	2	66	421	19.44	60.7
Test Case B - c	4	128	4	132	279	29.33	45.8
Test Case B - c	8	256	8	264	234	34.97	27.3
Test Case B - c	16	512	16	528	210	38.97	15.2
Test Case B - c	32	1024	32	1056	249	32.87	6.4
Test Case B - c	64	2048	64	2112	576	14.21	1.4

Table 80: Timings, speed-up and parallel efficiency of Test Case B-c (core-based approach) obtained with GADGET-4 on JUWELS.

Benchmark	#Nodes	#Cores	#Cores for comm.	Total #cores	MareNostrum4		
					Timings [s]	Speed-up	Parallel Efficiency [%]
Test Case B - c	1	2	0	2	10826	1.00	100.0
Test Case B - c	1	4	0	4	6974	1.55	77.6
Test Case B - c	1	8	0	8	3507	3.09	77.2
Test Case B - c	1	16	0	16	1859	5.82	72.8
Test Case B - c	1	32	0	32	956	11.32	70.8
Test Case B - c	2	64	2	66	565	19.16	59.9
Test Case B - c	4	128	4	132	396	27.34	42.7
Test Case B - c	8	256	8	264	344	31.47	24.6
Test Case B - c	16	512	16	528	325	33.31	13.0
Test Case B - c	32	1024	32	1056	399	27.13	5.3
Test Case B - c	64	2048	64	2112	1360	7.96	0.8

Table 81: Timings, speed-up and parallel efficiency of Test Case B-c (core-based approach) obtained with GADGET-4 on MareNostrum4.

Test Case B – n (Node-based approach)

Benchmark	#Nodes	#Cores	#Cores for comm.	Total #cores	Irene-SKL		
					Timings [s]	Speed-up	Parallel Efficiency [%]
Test Case B-n	1	2	0	2	9064	1.00	100.0
Test Case B-n	1	4	0	4	4962	1.83	91.3
Test Case B-n	1	8	0	8	2452	3.70	92.4
Test Case B-n	1	16	0	16	1298	6.98	87.3
Test Case B-n	1	32	0	32	749	12.10	75.6
Test Case B-n	2	64	2	66	446	20.32	63.5
Test Case B-n	3	128	3	131	382	23.73	37.1
Test Case B-n	6	256	6	262	303	29.91	23.4
Test Case B-n	11	512	11	523	389	23.30	9.1
Test Case B-n	22	1024	22	1046	565	16.04	3.1
Test Case B-n	44	2048	44	2092	1457	6.22	0.6

Table 82: Timings, speed-up and parallel efficiency of Test Case B-n (node-based approach) obtained with GADGET-4 on Irene-SKL.

Benchmark	#Nodes	#Cores	#Cores for comm.	Total #cores	JUWELS		
					Timings [s]	Speed-up	Parallel Efficiency [%]
Test Case B-n	1	2	0	2	8184	1.00	100.0
Test Case B-n	1	4	0	4	4540	1.80	90.1
Test Case B-n	1	8	0	8	2381	3.44	85.9
Test Case B-n	1	16	0	16	1255	6.52	81.5
Test Case B-n	1	32	0	32	672	12.18	76.1
Test Case B-n	2	64	2	66	421	19.44	60.7
Test Case B-n	3	128	3	131	315	25.98	40.6
Test Case B-n	6	256	6	262	283	28.92	22.6
Test Case B-n	11	512	11	523	290	28.22	11.0
Test Case B-n	22	1024	22	1046	331	24.73	4.8
Test Case B-n	44	2048	44	2092	789	10.37	1.0

Table 83: Timings, speed-up and parallel efficiency of Test Case B-n (node-based approach) obtained with GADGET-4 on JUWELS.

Note that only the results obtained with Irene-SKL and JUWELS are displayed here as their comparison is enough to conclude on the relative importance of the two approaches. It should also be mentioned that the timings, speed-up, and parallel efficiency only differ from those obtained in the core-based approach when more than 64 cores, that is more than 2 compute nodes, are used as it should be expected. Hence, what should be looked at is the variation on performance for more than 2 compute nodes. Figure 10 – Figure 12 display the run time, speed-up, and parallel efficiency vs. number of cores for the two approaches used in Test Case B.

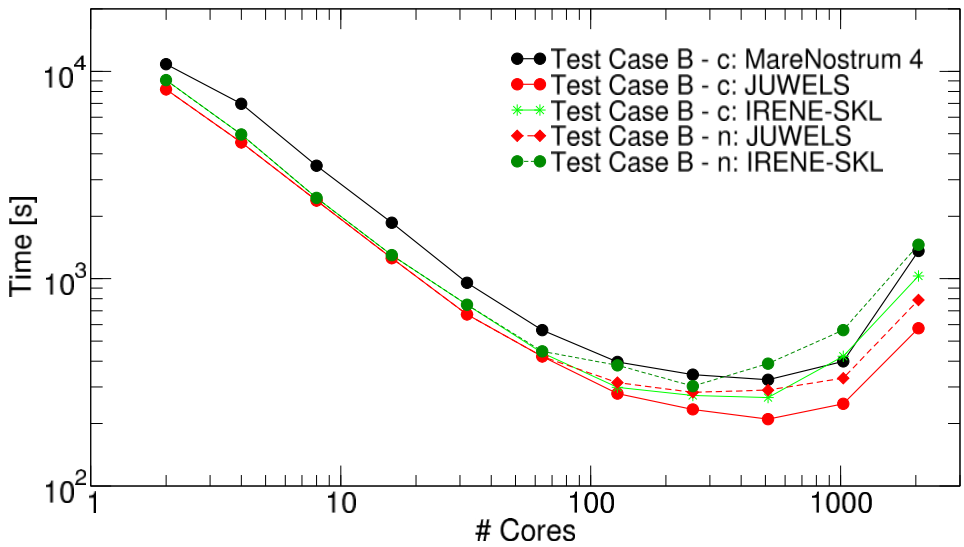

Figure 10: Simulation time for Test Case B - c (core-based approach; solid lines) and Test Case B-n (node-based approach; dashed lines) obtained with GADGET-4 on Irene-SKL, JUWELS, and MareNostrum4.

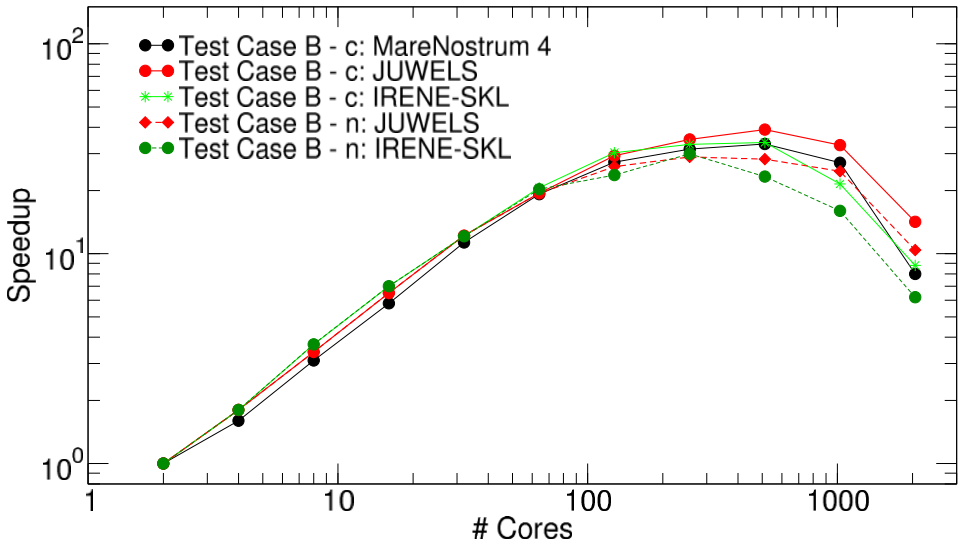

Figure 11: Speed-up for Test Case B - c (core-based approach; solid lines) and Test Case B-n (node-based approach; dashed lines) obtained with GADGET-4 on Irene-SKL, JUWELS, and MareNostrum4.

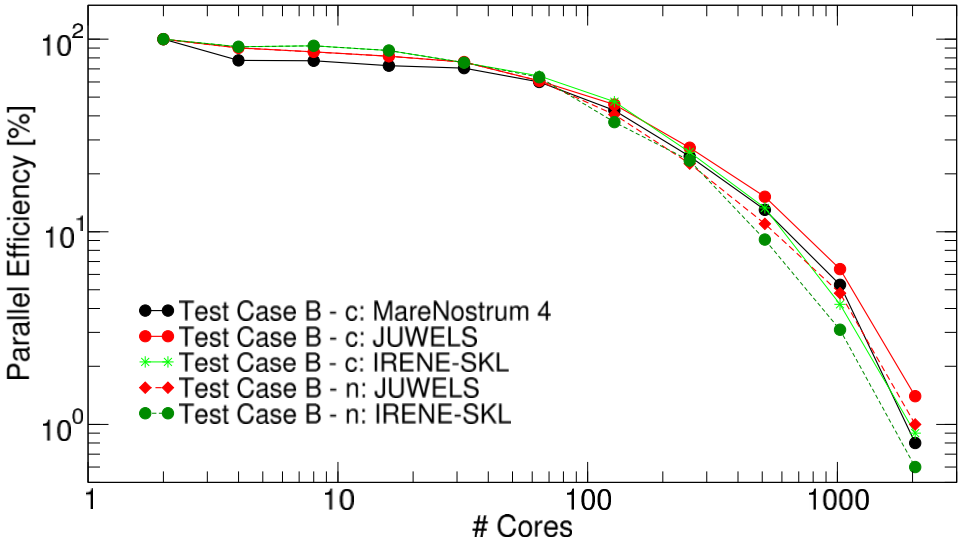

Figure 12: Parallel efficiency for Test Case B - c (core-based approach; solid lines) and Test Case B-n (node-based approach; dashed lines) obtained with GADGET-4 on Irene-SKL, JUWELS, and MareNostrum4.

4.4.5 Discussion

Test Case A (Cosmological dark matter-only simulation). The timings on JUWELS are faster than on MareNostrum4, which can be explained by the different clock speeds of the SKL Platinum CPUs used in the two machines (2.7 GHz for JUWELS vs. 2.1 GHz for MareNostrum4). The speed-ups of GADGET-4 on the two machines are similar up to 128 cores showing a departure for a larger number of cores. The parallel efficiency of the code in both machines has a similar evolution with JUWELS having the lower efficiency that becomes noticeable above 128 cores varying from 80%–84% (JUWELS-MareNostrum4) to less than 20% and 10% for 2048 and 4096 cores, respectively, for both machines. This is consistent with the breaking of the scalability of the code above 1024 cores. This efficiency is related mostly

to the code functions dealing with the tree-structure for the force field and the imbalance and communications losses.

Energy consumption typically decreases with the increase in the number of cores in a single compute node, but shows a slow increase with the number of compute nodes added to the calculations.

Test Case B (Blob test). The results of the test runs are displayed in Table 79 – Table 83 and in Figure 10 – Figure 12. MareNostrum4 holds the worst performance in this test as a result of the CPU clock speed being lower than that used in JUWELS and Irene-SKL for the Platinum CPUs. Both JUWELS and Irene-SKL show similar timings up to 64 cores, at this point the scaling is completely lost and there is no point in adding more resources. This breakup in performance results from the domain decomposition algorithm, while the actual SPH calculations still performs well at this point, although imbalance and communication losses are quite noticeable (see discussion [21]).

The differences between the two approaches are quite noticeable when the number of compute nodes becomes larger than 2. The core-based approach is more efficient than the node-based approach as can be seen in Figure 10 for the test runs with JUWELS and Irene-SKL. However, the benefits of using the core-based approach come at a price – per compute node there are 15 cores not being used (this number comes from taking 16 cores per socket for the calculations plus 1 core to handle the communications giving a total of 33 cores per compute node).

This loss of scalability of GADGET-4 for this test is easily explained by its size – it uses 1 million SPH particles. A more sizeable problem, with 10 million SPH particles, shows a better scalability with the three Tier-0 machines up to 640 cores.

4.5 GPAW

4.5.1 Performance Results

Two setups of GPAW were tested for this release of the UEABS:

- Python 3.8.7, NumPy 1.18.5, SciPy 1.5.4, ASE 3.19.3 and GPAW 20.1.0
- Python 3.9.4, NumPy 1.19.5, SciPy 1.5.4, ASE 3.20.1 and GPAW 20.10.0

The exception is SuperMUC-NG, where we had to fall back to NumPy 1.18.5 for GPAW 20.10.0 also as SciPy 1.5.4 would not compile with NumPy 1.19.5.

Note that even though GPAW does not pose an upper limit on the NumPy and SciPy versions, older versions of GPAW produce lots of deprecated feature warnings with newer versions so the versions of NumPy and SciPy were also chosen according to this.

More recent patch levels of Python 3.8 and 3.9 made changes to installation process of Python packages that also break the installation procedure that was used.

On all systems the Intel compilers and Intel MKL were used, also on AMD EPYC systems, as the centres did not discourage it and used it themselves in many software installations. On Irene, Open MPI was used instead of Intel MPI, and on Hawk most tests were done with the vendor-proprietary HPE MPT library and Open MPI.

Note also that GPAW in many cases favours the use of regular configurations with the same number of processes on each node rather than filling some nodes completely and leaving one or more cores unused on other nodes. Hence, the number of cores chosen for comparing the benchmark results is a compromise: We took some regular numbers that were tested on all clusters, and then sets in functions of the number of cores per node for both the Skylake-based clusters and the AMD Rome based clusters. Not all cases could be run on all core configurations due to the limited resources available and in some cases also due to problems occurring on some clusters in some configurations.

The reported times are as reported by GPAW. This removes a lot of the setup overhead and ensures that it does not become a file system benchmark as the start-up of large Python jobs can be expensive due to the number of files involved.

4.5.1.1 Test Case S

The small test case was run on the Skylake-based clusters JUWELS, SuperMUC-NG and MareNostrum4, and on the AMD Rome-based clusters Irene (Rome section) and Hawk. On Hawk, both the vendor-provided MPT and Open MPI were used.

However, on Irene GPAW failed to start if the requested number of cores was not a multiple of the number of cores per node, leading to only one result for this test case. As we have more results on Irene for the other benchmarks, we do give the results here too for completeness.

GPAW 20.1.0 / Test Case S						
Cores	**JUWELS**	**SuperMUC-NG**	**MareNostrum4**	**Irene-Rome**	**Hawk (MPT)**	**Hawk (OMPI)**
1	1607 s	2207 s	2387 s		1619 s	1638 s
10	181 s	242 s	260 s		295 s	173 s
24	94.2 s	1178 s	133 s		168 s	84.6 s
25	97.7 s	120 s	138 s		162 s	88.2 s
32	80.4 s	96.8 s	108 s		124 s	68.2 s
48	72.2 s	78.5 s	91.3 s		88.4 s	59.9 s
50	52.2 s	64.2 s	74.3 s		81.9 s	57.8 s
64	48.3 s	60.2 s	68.0 s		72.7 s	53.4 s
96	41.0 s	47.0 s	54.4 s		51.4 s	46.3 s
100	37.3 s	44.8 s	51.0 s		50.5 s	51.1 s
128	34.5 s	39.0 s	44.7 s	44.2 s	41.3 s	45.2 s
144	32.6 s	36.8 s	43.1 s		40.7 s	41.4 s

Table 84: Benchmark run time, GPAW 20.1.0, Test Case S

GPAW 20.10.0 / Test Case S						
Cores	**JUWELS**	**SuperMUC-NG**	**MareNostrum4**	**Irene-Rome**	**Hawk (MPT)**	**Hawk (OMPI)**
1	1636 s	2258 s	2378 s		1613 s	1630 s
10	187 s	250 s	263 s		296 s	174 s
24	96.6 s	122 s	132 s			84.9 s
25	102 s	124 s	137 s		162 s	88.6 s
32	82.6 s	99.1 s	107 s		124 s	68.5 s
48	74.4 s	80.6 s	101 s			59.1 s
50	53.4 s	66.4 s	73.2 s		81.7 s	56.7 s
64	49.5 s	61.8 s	69.5 s		72.6 s	53.3 s
96	42.5 s	48.2 s	55.5 s		51.1 s	46.7 s
100	37.9 s	46.3 s	51.5 s		50.7 s	50.8 s
128	35.3 s	39.6 s	47.2 s	44.1 s	41.2 s	45.1 s
144	33.4 s	37.9 s	43.1 s		40.4 s	42.2 s

Table 85: Benchmark run time, GPAW 20.10.0, Test Case S

For all systems except for Irene, we also computed the efficiency by comparing the compute time and number of cores used with the time for a run on a single core.

GPAW 20.1.0 / Test Case S					
Cores	**JUWELS**	**SuperMUC-NG**	**MareNostrum4**	**Hawk (MPT)**	**Hawk (OMPI)**
1	100%	100%	100%	100%	100%
10	89%	91%	92%	55%	95%
24	71%	78%	75%	40%	81%
25	66%	74%	69%	40%	74%
32	63%	71%	69%	41%	75%
48	46%	59%	55%	38%	57%
50	62%	69%	64%	40%	57%
64	52%	57%	55%	35%	48%
96	41%	49%	46%	33%	37%
100	43%	49%	47%	32%	32%
128	36%	44%	42%	31%	28%
144	34%	42%	39%	28%	28%

Table 86: Efficiency with respect to a single core run, GPAW 20.1.0, Test Case S

GPAW 20.10.0 / Test Case S					
Cores	**JUWELS**	**SuperMUC-NG**	**MareNostrum4**	**Hawk (MPT)**	**Hawk (OMPI)**
1	100%	100%	100%	100%	100%
10	88%	90%	90%	55%	94%
24	71%	77%	75%		80%
25	65%	73%	70%	40%	74%
32	62%	71%	69%	41%	74%
48	46%	58%	49%		57%
50	61%	68%	65%	40%	58%
64	52%	57%	54%	35%	48%
96	40%	49%	45%	33%	36%
100	43%	49%	46%	32%	32%
128	36%	45%	39%	31%	28%
144	34%	41%	38%	28%	27%

Table 87: Efficiency with respect to a single core run, GPAW 20.10.0, Test Case S

A few things are worth noting.

It is not surprising that MareNostrum4 is slower than JUWELS as its CPUs have a lower clock speed. The nominal clock speed of the CPU used in SuperMUC-NG is higher than on JUWELS which is not reflected in the results. It appears that due to energy management issues it is run at a lower clock speed. Even when running within a node, the efficiency decreases faster on JUWELS though. Possible causes may be a different internode communication strategy (most MPI implementations support multiple options, with some requiring a kernel extension), but it may also be due to the dynamic way in which Intel processors change their clock speed depending on the load of a socket and the type of instructions used which may be different on SuperMUC-NG and JUWELS. It is possible that the single core result on JUWELS is a bit increased by a small boost in clock speed, or that that particular high-frequency SKU sees a larger reduction in clock speed when AVX512 instructions are used and all cores are used, lowering the result on a full node, and this behaviour would also be reflected in a lower efficiency when more cores are used.

What is also surprising is the poor scaling on Hawk. This is even more remarkable as all runs up to 128 cores stay within a single node. With MPT, the scaling is already surprisingly bad using only a limited number of cores, while with Open MPI, the scaling is OK at low core counts but becomes nearly as bad as with MPT when using half or more of the cores of a node. This is because with MPT, processes are by default allocated sequentially from core 0 while with Open MPI processes were spread over the full node. E.g. in the 10 MPI ranks case with MPT the run time is close to 300 s while with Open MPI the run time is around 170 s. With MPT all processes were running in the first NUMA domain and competing for memory bandwidth in that domain while with Open MPI they were spread over all NUMA domains. Therefore, the run with MPT is already more memory bandwidth constrained. When nearly all cores are used, the benchmark times (and efficiency as the single core times are about the same) become comparable again as in both cases cores on both sockets are used. The fact that using 10 cores we see such a big influence of the distributions of processes over the cores of the node indicates that certainly on AMD Rome processors this benchmark is already memory bandwidth bound rather than compute bound which is somewhat surprising for a DFT code that supposedly

makes good use of BLAS operations and FFT in optimised libraries, though another explanation could be that the BLAS and FFT libraries do not sufficiently recognise the sizes of the L2 and L3 cache on the AMD Rome CPU and therefore cause a heavier load on the memory system (we used MKL for BLAS but FFTW for the FFT operations). When the processes are more spread out as is the case with Open MPI, going from 96 cores (3 cores/CCU) to 128 cores (4 cores/CCU) does not really give any benefit at all so this may be an example of a code where for a given number of nodes it may make sense to not use all cores on 64-cores-per-socket AMD Rome CPUs.

Also somewhat surprising is the performance of the Rome partition of Irene. According to the documentation, the nominal clock speed is 2.6 GHz while for Hawk the nominal clock speed is 2.25 GHz, yet the full node result on Irene is worse than on Hawk. The compilers we used were of the same generation. On Irene, we failed to produce results using partial nodes due to problems with the process starter, making the comparison somewhat limited.

There is also no noticeable difference between GPAW 20.1.0 and 20.10.0, despite the latter using a newer version of Python that is claimed to have improved memory management and except on SuperMUC-NG also a newer version of NumPy. Both setups were using the same compilers and same optimised mathematics libraries though.

As a node-based on the AMD EPYC Rome CPU is likely not much more expensive than a node-based on Intel Skylake CPUs, it may make sense to also look at the run times on a node base. Compute centres running AMD hardware often admit that the core performance of AMD is lower than for recent Intel processors but that this is more than compensated by the fact that you get a lot more cores for the same amount of money. For GPAW 20.1.0, we get

GPAW 20.1.0 / Test Case S						
Nodes	**JUWELS**	**SuperMUC-NG**	**MareNostrum4**	**Irene-Rome**	**Hawk (MPT)**	**Hawk (OMPI)**
1	72.2 s	78.5 s	91.3 s	44.2 s	41.3 s	45.2 s
2	41.0 s	47.0 s	54.4 s			
3	32.6 s	36.8 s	43.1 s			

Table 88: Benchmark run time per node, GPAW 20.1.0, Test Case S

Two nodes of JUWELS or SuperMUC-NG or three nodes of MareNostrum4 (which all have 48 cores per node) are needed to get a similar run time as on one 128-core node of Hawk or Irene.

4.5.1.2 Test Case M

The medium test case was run on the same machines and in the same configurations as the small test case: the Skylake-based clusters JUWELS, SuperMUC-NG and MareNostrum4, and the AMD Rome-based clusters Irene (Rome section) and Hawk. On Hawk, both the vendor-provided MPT and Open MPI were used. As for Test Case S, runs that were not using all cores on all allocated nodes failed to start with error messages from the resource manager on Irene.

GPAW 20.1.0 / Test Case M						
Cores	**JUWELS**	**SuperMUC-NG**	**MareNostrum4**	**Irene-Rome**	**Hawk (MPT)**	**Hawk (OMPI)**
48	1967 s	1967 s	2091 s		2613 s	1507 s
50	1541 s	1899 s	2030 s		2819 s	1830 s
100	924 s	1015 s	1102 s		1510 s	1293 s
128	709 s	730 s	839 s	1082 s	1122 s	1073 s
240	528 s	602 s	680 s		781 s	782 s
250	526 s	619 s	676 s		781 s	804 s
256	429 s	495 s	543 s	620 s	671 s	636 s
480	260 s	303 s	354 s		403 s	395 s
500	286 s	320 s	354 s		416 s	412 s
512	231 s	259 s	292 s	306 s	329 s	333 s
720	203 s	229 s	289 s		303 s	293, s
768	173 s	200 s	280 s	229 s	290 s	248 s
960	162 s	186 s	237 s		252 s	220 s
1000	169 s	196 s	232 s		249 s	225 s
1008	153 s	182 s	219 s		208 s	
1024	144 s	166 s	178 s	183 s	201 s	192 s

Table 89: Benchmark run times, GPAW 20.1.0, Test Case M

GPAW 20.10.0 / Test Case M						
Cores	**JUWELS**	**SuperMUC-NG**	**MareNostrum4**	**Irene-Rome**	**Hawk (MPT)**	**Hawk (OMPI)**
48	1983 s	1983 s	2093 s		2568 s	1522 s
50	1536 s	1846 s	2017 s		2814 s	1843 s
100	918 s	1010 s	1090 s		1520 s	1301 s
128	714 s	732 s	833 s	1062 s	1118 s	1063 s
240	536 s	584 s	682 s		782 s	753 s
250	523 s	603 s	682 s		786 s	791 s
256	431 s	496 s	553 s	620 s	669 s	637 s
480	260 s	290 s	361 s		405 s	394 s
500	280 s	322 s	357 s		418 s	428 s
512	229 s	257 s	295 s	308 s	326 s	335 s
720	204 s	235 s	281 s		300 s	292 s
768	174 s	199 s	273 s	230 s	286 s	248 s
960	161 s	190 s	237 s		257 s	218 s
1000	171 s	190 s	232 s		250 s	224 s
1008	154 s	176 s	203 s		207 s	
1024	142 s	166 s	177 s	179 s	200 s	190 s

Table 90: Benchmark run times, GPAW 20.10.0, Test Case M

As was the case for Test Case S, SuperMUC-NG is not doing better than the other Skylake-based cluster despite its processor SKU with higher nominal clock speed and scaling is worse than on JUWELS. However, Irene is now slightly faster than Hawk which is more in line with the expectations based on the nominal clock speed of their processors.

To have a better look at the scaling behaviour we first compute the efficiency with respect to a 48-core run (one full node on the Skylake-based clusters) for those clusters for which we have results for a 48-core run.

GPAW 20.1.0 / Test Case M					
Cores	**JUWELS**	**SuperMUC-NG**	**MareNostrum4**	**Hawk (MPT)**	**Hawk (OMPI)**
48	100%	100%	100%	100%	100%
50	123%	99%	99%	89%	79%
100	102%	93%	91%	83%	56%
128	104%	101%	94%	87%	53%
240	75%	65%	62%	67%	39%
250	72%	61%	59%	64%	36%
256	86%	75%	72%	73%	44%
480	76%	65%	59%	65%	38
500	66%	59%	57%	60%	35%
512	80%	71%	67%	75%	43%
720	65%	57%	48%	57%	34%
768	71%	62%	47%	56%	38%
960	61%	53%	44%	52%	34%
1000	56%	48%	43%	50%	32%
1008	61%	52%	46%	60%	
1024	64%	56%	55%	61%	37%

Table 91: Efficiency with respect to a 48-core run, GPAW 20.1.0, Test Case M

GPAW 20.10.0 / Test Case M					
Cores	**JUWELS**	**SuperMUC-NG**	**MareNostrum4**	**Hawk (MPT)**	**Hawk (OMPI)**
48	100%	100%	100%	100%	100%
50	124%	103%	99%	88%	79%
100	104%	94%	92%	81%	56%
128	104%	102%	94%	86%	54%
240	74%	68%	61%	66%	40%
250	73%	63%	59%	63%	37%
256	86%	75%	71%	72%	45%
480	76%	69%	58%	63%	39%
500	68%	59%	56%	59%	34%
512	81%	72%	67%	734%	43%
720	65%	56%	48%	57%	35%
768	71%	62%	48%	56%	38%
960	61%	52%	44%	50%	35%
1000	56%	50%	43%	49%	33%
1008	62%	54%	49%	59%	
1024	65%	56%	55%	60%	38%

Table 92: Efficiency with respect to a 48-core run, GPAW 20.10.0, Test Case M

Here JUWELS scales slightly better than the other two Skylake-base clusters SuperMUC-NG and MareNostrum4, and MareNostrum4 often shows the worst efficiency. The outlier on JUWELS for the 50-core case is due to the way the MPI processes are distributed on JUWELS. Each 48-core node receives 25 MPI processes and they are properly distributed over the cores by default. We did not experiment with process pinning to see if similar results could be obtained on the Skylake systems.

The results for both MPI implementations on Hawk should be compared with care. The Open MPI runs only scale worse because the 48-core run time is much shorter than with MPT, again due to the process distribution across the 128-core node. The MPT process starter packs all 48 processes on the first socket, while the Open MPI process starter lets them migrate over the whole node. In fact, comparing actual run times instead shows that Open MPI performs better than MPT on this benchmark, even when all cores are used as is the case for the 128, 256, 512, 768 and 1024 core runs. It is also hard to compare Hawk to the three Skylake-based system as there is no common reference point. As the 48-core run on Hawk does not even fill an entire socket, the memory bandwidth available to each core may be better than on a fully loaded node. This produces a better-than-expected run time for the reference case, reducing the efficiency for the cases using more cores. The MPT results are probably most relevant as a reference point and they would put Hawk in the middle of the ballpark.

To compare the two AMD EPYC systems, we also computed the efficiency with respect to a single node run on those two systems:

GPAW 20.1.0 / Test Case M			
Cores	**Irene-Rome**	**Hawk (MPT)**	**Hawk (OMPI)**
128	100%	100%	100%
256	87%	84%	84%
512	88%	85%	81%
768	79%	65%	72%
1024	74%	70%	70%

Table 93: Efficiency with respect to a single node run on AMD EPYC, GPAW 20.1.0, Test Case M

GPAW 20.10.0 / Test Case M			
Cores	**Irene-Rome**	**Hawk (MPT)**	**Hawk (OMPI)**
128	100%	100%	100%
256	86%	84%	84%
512	86%	86%	79%
768	77%	65%	71%
1024	74%	70%	70%

Table 94: Efficiency with respect to a single node run on AMD EPYC, GPAW 20.10.0, Test Case M

Here we see that there is no noticeable difference in the scaling behaviour of the AMD Rome partition of Irene and Hawk. The results for MPT and Open MPI on Hawk are also more or less the same but remember that the actual run times with Open MPI are always shorter than with MPT. The fact that this is also the case for a single full node may indicate that the intra-node communication is faster in Open MPI than in MPT.

To compare the benchmark time in function of the number of nodes, we again restrict ourselves to GPAW 20.1.0:

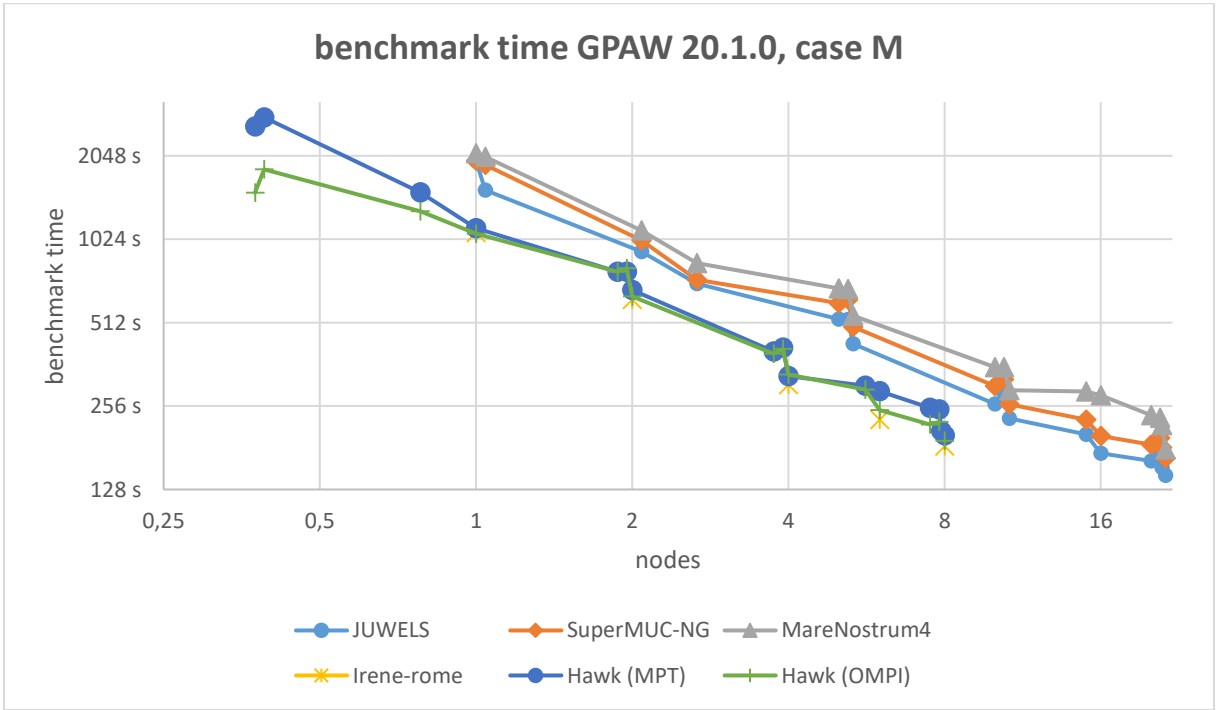

Figure 13: Benchmark time for GPAW 20.1.0, Test Case M, as function of the number of nodes

As expected, in terms of the number of nodes, the AMD-based clusters are superior again.

4.5.1.3 Test Case L

For Test Case L we took 10 nodes on a Skylake cluster as the minimal configuration. On Irene again runs not employing all cores on all assigned nodes failed with error message from the process starter / resource manager. In the interest of time, we restricted ourselves on Hawk to the vendor-provided MPT implementation (also triggered by some failures with Open MPI which we could not diagnose in time).

The following tables show the run times as reported by GPAW:

GPAW 20.1.0					
Cores	**JUWELS**	**SuperMUC-NG**	**MareNostrum4**	**Irene-Rome**	**Hawk (MPT)**
480	1031 s	1151 s	1446 s		1627 s
500	993 s	1073 s	1242 s		1568 s
512	992 s	1090 s	1245 s	1444 s	1494 s
960	586 s	645 s	828 s		942 s
1000	605 s	614 s	981 s		901 s
1024	560 s	614 s	690 s	749 s	805 s
2000	362 s	391 s	759 s		559 s
2048	345 s	367 s	503 s	453 s	491 s
2400	309 s	374 s	476 s		507 s
4096	232 s	271 s	419 s	353 s	394 s
4800	221 s	249 s	419 s		387 s
5000	233 s	248 s	379 s		403 s
6144	205 s	233 s	317 s	324 s	
8192	193 s	206 s	437 s	289 s	350 s
9600		191 s	296 s		412 s
9884				248 s	
10000	227 s	197 s	310 s		500 s
10032		196 s	353 s		410, s
10240		203 s	354 s	300 s	429 s

Table 95: Benchmark run times, GPAW 20.1.0, Test Case L

GPAW 20.10.0					
Cores	**JUWELS**	**SuperMUC-NG**	**MareNostrum4**	**Irene-Rome**	**Hawk (MPT)**
480	1031 s	1159 s	1458 s		1626 s
500	1009 s	1098 s	1242 s		1577 s
512	1006 s	1105 s	1222 s	1457 s	1493 s
960	594 s	655 s	824 s		939 s
1000	609 s	626 s	984 s		903 s
1024	566 s	618 s	690 s	7647 s	801 s
2000	366 s	375 s	768 s		558 s
2048	349 s	370 s	512 s	467 s	492 s
2400	315 s	339 s	499 s		508 s
4096	238 s	255 s	433 s	358 s	392 s
4800	226 s	236 s	391 s		397 s
5000	236 s	232 s	375 s		404 s
6144	206 s	231 s	320 s	327 s	387 s
8192	191 s	185 s	431 s	291 s	347 s
9600		172 s	322 s		411 s
9884				246 s	
10000		183 s	310 s		498 s
10032		184 s	336 s		409 s
10240		191 s	351 s	301 s	419 s

Table 96: Benchmark run times, GAPW 20.10.0, Test Case L

The 9884-core run failed on all three 48-core node systems and on Hawk. Note also that there is some irregularity in the results. This is because of the algorithms used in this benchmark. Some core configurations work considerably better than others.

However, another problem also becomes immediately clear from this table. Test Case L is no longer suited for modern systems or at least does not live up to the promises of being suitable for up to 10,000 cores which was the original intent of this case. Even from the timings we can already see that using more than roughly 2500 cores makes no sense.

Another remarkable issue is that Hawk stops scaling completely at a lower core and node number than Irene-Rome and neither of the two AMD-based clusters can reach the minimum run times obtained on JUWELS and SuperMUC-NG (but neither can MareNostrum4). However, on all five clusters this is already outside the range of number of cores or nodes that is reasonable from an economical point of view.

As for Test Case M, two sets of tables are computed to compare the efficiency of the systems: One for all systems on which a 480-core run worked, and one for the two AMD systems, using 4 full nodes as the reference configuration.

GPAW 20.1.0				
Cores	**JUWELS**	**SuperMUC-NG**	**MareNostrum4**	**Hawk (MPT)**
480	100%	100%	100%	100%
500	100%	103%	112%	100%
512	97%	99%	109%	102%
960	88%	89%	87%	86%
1000	82%	90%	71%	87%
1024	86%	88%	98%	95%
2000	68%	71%	46%	70%
2048	70%	74%	67%	78%
2400	67%	62%	61%	64%
4096	52%	50%	41%	48%
4800	47%	46%	35%	42%
5000	43%	45%	37%	39%
6144	39%	39%	36%	33%
8192	31%	33%	19%	27%
9600		30%	24%	20%
10000	22%	28%	22%	16%
10032		28%	20%	19%
10240		27%	19%	18%

Table 97: Efficiency with respect to a 480-core run, GPAW 20.1.0, Test Case L

GPAW 20.10.0				
Cores	**JUWELS**	**SuperMUC-NG**	**MareNostrum4**	**Hawk (MPT)**
480	100%	100%	100,	100%
500	98%	101%	113%	99%
512	96%	98%	112%	102%
960	87%	89%	89%	70%
1000	81%	89%	71%	64%
1024	85%	88%	99%	95%
2000	68%	74%	46%	49%
2048	69%	73%	67%	77%
2400	65%	68%	59%	64%
4096	51%	53%	39%	49%
4800	46%	49%	37%	41%
5000	42%	48%	37%	39%
6144	39%	39%	36%	33%
8192	32%	37%	20%	28%
9600		34%	23%	20%
10000		30%	23%	16%
10032		30%	21%	19%
10240		28%	20%	18%

Table 98: Efficiency with respect to a 480-core run, GPAW 20.10.1, Test Case L

For this test case, there is little difference in the scaling behaviour between SuperMUC-NG and JUWELS, but MareNostrum4 performs worse. In the 1024–2048 core range, which may realistically be the maximum for this benchmark if cost efficiency matters, Hawk does very well compared to the other systems.

We again compared both AMD systems using 4 full nodes as the reference.

GPAW 20.1.0 / Test Case L		
Cores	**Irene-Rome**	**Hawk (MPT)**
512	100%	100%
1024	96%	93%
2048	80%	76%
4096	51%	47%
6144	37%	33%
8192	31%	27%
9884	30%	
10240	24%	17%

Table 99: Efficiency with respect to a 512-core run, GPAW 20.1.0, Test Case L

GPAW 20.10.0 / Test Case L		
Cores	**Irene-Rome**	**Hawk (MPT)**
512	100%	100%
1024	95%	93%
2048	78%	76%
4096	51%	48%
6144	37%	32%
8192	31%	27%
9884	31%	
10240	24%	18%

Table 100: Efficiency with respect to a 512-core run, GPAW 20.10.0, Test Case L

On Test Case L, Irene scales slightly better than Hawk.

Finally, we compare the benchmarks times for GPAW 20.1.0 in function of the number of nodes rather than the number of cores:

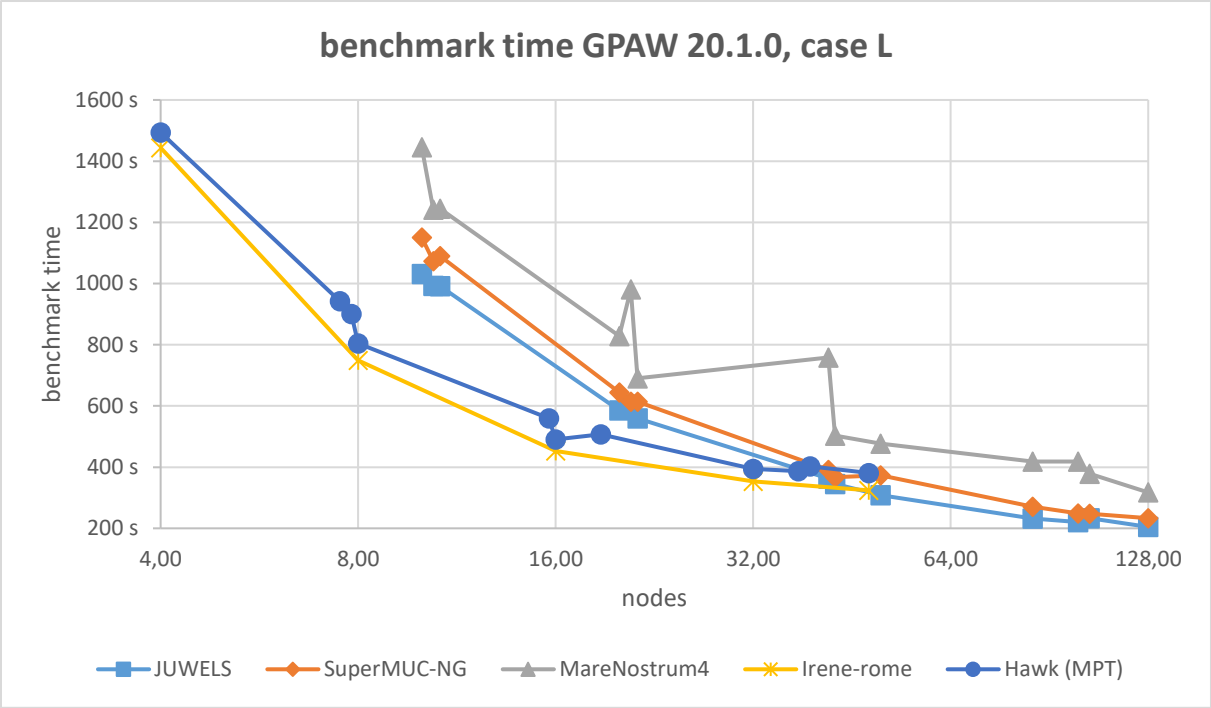

Figure 14: Benchmark time for GPAW 20.1.0, Test Case L, in function of the number of nodes.

Up to around 32 nodes the AMD-based clusters are superior, but after that JUWELS and SuperMUC-NG take over.

4.5.2 Energy Performance

The energy performance results for the GPAW benchmark are rather limited. Tests were run in a period that energy measurements were not available on SuperMUC-NG. On MareNostrum4, results were easily obtained but turned out to be somewhat unreliable, maybe due to the relatively short run time of the benchmark. Several tests had to be rerun several times to get a credible result as some results were as much as 50% off of the expectations, and one can expect that the error margin on the reported results is significant. We did not obtain energy results for Irene on time as there is no direct access to the information.

Based on the energy consumption and run time as measured by Slurm, we also report the power. The power is computed based on the time reported by Slurm and not the benchmark time used before (which does not include some initialisations) as that is the time that corresponds to the interval over which the energy consumption was measured. As the results for both versions of GPAW tested are very similar, we restrict the results to GPAW 20.10.0.

The results for Test Case S are:

GPAW 20.10.0 / Test Case S			
Cores	**Energy**	**Power**	**Node hours**
1	361 kJ	0.15 kW	0.663
10	60 kJ	0.22 kW	0.077
24	42 kJ	0.29 kW	0.040
25	41 kJ	0.28 kW	0.041
32	36 kJ	0.32 kW	0.032
48	42 kJ	0.37 kW	0.032
50	48 kJ	0.54 kW	0.049
64	44 kJ	0.59 kW	0.041
96	40 kJ	0.66 kW	0.034
100	50 kJ	0.86 kW	0.048
128	46 kJ	0.89 kW	0.043
144	53 kJ	0.92 kW	0.048

Table 101: Power and cost data for MareNostrum4, GPAW 20.10.0, Test Case S

The total energy consumption for the single core run is very high, though as expected the power draw is the lowest of all. This is because the measurement is for the full node even though only one core is used. As confirmed by the Slurm logs, the other cores do clock back but there is still a considerable power consumption in those idle cores. It is not unexpected that the minimal power consumption occurs at a rather small configuration, as with more cores there is a significant communication overhead that also costs energy (even though the energy spent by the switches in the communication network is not measured). In this case the optimum is actually at the 32-core run which is two thirds of a node, though the increase beyond that point remains reasonable with another local minimum at 96 cores which is two full nodes.

The results for Test Case M are:

GPAW 20.10.0 / Test Case M			
Cores	**Energy**	**Power**	**Node hours**
48	1100 kJ	0.52 kW	0.58
50	1231 kJ	0.61 kW	1.13
100	1141 kJ	1.04 kW	0.91
128	970 kJ	1.14 kW	0.71
240	1255 kJ	1.81 kW	0.97
250	1588 kJ	2.29 kW	1.16
256	1165 kJ	2.06 kW	0.94
480	1275 kJ	3.49 kW	1.01
500	1249 kJ	3.38 kW	1.13
512	966 kJ	3.08 kW	0.96
720	1508 kJ	5.01 kW	1.25
768	1526 kJ	5.37 kW	1.26
960	1608 kJ	6.11 kW	1.46
1000	1648 kJ	6.65 kW	1.45
1008	1241 kJ	5.72 kW	1.27
1024	1244 kJ	6.65 kW	1.14

Table 102: Power and cost data for MareNostrum4, GPAW 20.10.0, Test Case M

The optimum in terms of power consumption now appears to be at 128 or 512 cores, though especially the latter result is somewhat suspicious. However, it is not surprising that in terms of node hours consumed a single node configuration is best.

The results for Test Case L are:

GPAW 20.10.0 / Test Case L			
Cores	**Energy**	**Power**	**Node hours**
480	5396 kJ	3.67 kW	4.08
500	4163 kJ	3.33 kW	3.82
512	4150 kJ	3.36 kW	3.77
960	5355 kJ	6.40 kW	4.65
1000	6531 kJ	6.52 kW	5.84
1024	5092 kJ	7.21 kW	4.31
2000	7319 kJ	9.34 kW	9.15
2048	7762 kJ	14.70 kW	6,31
2400	7667 kJ	14.92 kW	7.14
4096	11456 kJ	25.34 kW	10.80
4800	11804 kJ	29.36 kW	11.17
5000	10816 kJ	27.45 kW	11.49
6144	13461 kJ	40.30 kW	11.88
8192	21752 kJ	48.02 kW	21.52
9600	20301 kJ	59.53 kW	18.94
10000	19833 kJ	60.84 kW	18.93
10032	21604 kJ	60.86 kW	20.61
10240	22624 kJ	61.15 kW	21.99

Table 103: Power and cost data for MareNostrum4, GPAW 20.10.0, Test Case L

Here the optimum did not occur in the smallest configuration which was 10 nodes completely filled or 480 cores, but at a 512-core configuration, employing 11 nodes. There is another local minimum at 1024 cores but at a 22% higher power consumption than in the 512 core computation. The optimum in terms of node hours is also reached at 512 MPI processes.

4.5.3 Discussion

Although GPAW can scale to fairly large clusters, GPAW may not be an ideal benchmark for procurements. In favour of GPAW is that it uses a FFT library (with FFTW interface) and BLAS, LAPACK, and ScaLAPACK, so it also tests several libraries that are often delivered with vendor-specific optimisations by cluster vendors. However, as it is Python-based and also uses NumPy and SciPy, the installation of the software is complicated. To avoid library conflicts, one should ensure that NumPy/SciPy and GPAW use the same FFT, BLAS and LAPACK library. The build process of optimised Python software is far from straightforward. Python has its own processes for building Python packages from C/C++ sources, but it does not support proper building of MPI-based packages nor is it straightforward (or is there a universal way) to specify additional libraries that should be used. Moreover, the development process of NumPy and SciPy seems to favour new features over stabilising the code and adapting to modern language standards. The amount of compiler warnings when using recent compilers is overwhelming (to the extent the actual problems are easily overlooked) and there is no easy-to-find documentation about the test sets to diagnose the source of failing tests. On parallel file systems, installing GPAW puts a very high stress on the file system. Even a fairly minimal

installation, with a Python installed from scratch partly disabling several of the standard library packages that are not needed for the benchmarks, produces an installation of over 20,000 mostly very small files. On some of the PRACE Tier-0 machines a single compile took over 4 hours, so getting a quick turnaround time in case of correcting compile or performance problems is very hard to impossible if it involves re-installing from Python itself or NumPy up. Several pure Python packages take minutes to install on a parallel file system while they install in under a second on a local workstation SSD.

Some combinations of Python, NumPy and SciPy would not install on all machines tested even though the dependency files in the Python packages showed this as a valid combination and even though often compiler versions only differed at the patch level. Hence installation instructions for the benchmark may not only see changes between clusters to accommodate for a different set of optimised BLAS libraries, but may even see changes in the versions of Python packages used.

GPAW itself also contains two annoying problems that do not seem to get fixed and do not show up in every installation. One (causing runtime crashes) may be due to a different interpretation of the OpenMP standard. A clause is used to indicate that it is safe to vectorise a particular loop, but the data is not always correctly aligned. Even though the 64-bit x86 instruction set does support unaligned memory access, this is ambiguous. The instructions for aligned memory access are faster. The Intel compiler assumes that "safe for vectorisation" also implies that all memory accesses are correctly aligned and uses the AVX instructions for correctly aligned data, causing runtime errors in some cases. Other compilers may not make that assumption. The solution is to either turn off OpenMP SIMD support in the compiler or to develop a patch that removes the OpenMP pragmas from the offending loop. A second problem is within the installation procedure which uses a mix of compilation through setup tools for parts of the code that do not involve MPI and direct calls to the compiler for other parts. However, it does turn out that at least one of the files that does contain the MPI header file gets compiled without calling the MPI compiler wrapper. On systems where the regular compiler is configured to use CPATH to search for extra include files and where the MPI headers are also added to CPATH, compilation proceeds without problems but on other systems compilation fails and GPAW must be forced to use the MPI compiler wrappers for everything.

Another problem is that GPAW regularly breaks compatibility with input files for older versions, so any benchmark may be short-lived. Test Cases M and L cases needed significant redevelopment when upgrading from the GPAW version used in PRACE-5IP to GPAW 20.1.0 which restricted the time available to actually look for the best run configuration. Test Case L is no longer compatible with GPAW 21.1.0, which changed some models considerably and made the old behaviour optional. Both the final solution and convergence behaviour change significantly.

4.6 GROMACS

4.6.1 System Software Environment

GROMACS requires a C/C++ compiler, MPI, the FFTW library, and for GPU support a CUDA SDK installation. The FFTW 3.3.8 library was compiled from source on all systems using the underlying compilers. GROMACS version 2020.3 was used on all systems. This was the latest during first runs. The software stack used on the machines is summarised in Table 104.

Machine	C/C++ compiler	MPI flavour	CUDA
Hawk	GCC 9	HP-MPI	
Irene	Intel 19	OpenMPI 4.0.2	
JUWELS	Intel 20	ParaStation MPI	
MARCONI100	GCC 8	Spectrum MPI	10.2
MareNostrum4	Intel 19	Intel MPI	
Piz Daint	Cray CC (GCC 8.3)	Cray MPICH	10.1
SuperMUC-NG	Intel 19	Intel MPI	

Table 104: Software environment used in GROMACS Benchmarks

4.6.2 Performance Results

Since GROMACS uses hybrid MPI/OpenMP parallelisation, before running the full benchmarks on each machine, a number of small runs was performed in order to find the combination of tasks per node / threads per task that yields the best performance. Performance is reported by GROMACS in its logfile. Performance results are presented grouped by system type in Table 105 – Table 107. Test Case A is small for multi-GPU systems like MARCONI100. Additional performance results are presented for MARCONI100 using 1 and 2 GPUs and the corresponding number of cores/threads. It should be noted that for Test Case C the GPU memory requirements can be fulfilled by a minimum of 8 nodes. Energy to solution accounting was not available on all systems. On some of the systems, only partial energy accounting is available.

			Hawk	Irene
Nodes	**Tasks / Node**	**Threads Task**	**Performance [ns / day]**	
Test Case A				
1	128	1	107.332	120.407
2	128	1	171.061	177.344
4	128	1	209.473	230.406
8	128	1	298.657	253.150
16	128	1	268.477	
Test Case B				
1	128	1	5.148	6.282
2	128	1	10.717	12.405
4	128	1	20.134	23.659
8	128	1	35.139	42.817
16	128	1	64.596	72.798
32	128	1	101.897	91.091
64	128	1	162.760	147.331
128	128	1	272.654	177.306
256	128	1	414.019	238.696
Test Case C				
1	128	1	0.390	0.468
2	128	1	0.818	0.918
4	128	1	1.589	1.799
8	128	1	2.654	3.487
16	128	1	6.074	7.236
32	128	1	11.466	12.005
64	128	1	20.008	22.259
128	128	1	35.723	31.269
256	128	1	42.324	

Table 105: GROMACS performance on AMD EPYC based Systems

	SuperMUC-NG			MareNostrum4			JUWELS	
Nodes	**Tasks/Node - Threads / Task**	**Performance [ns / day]**	**Energy to solution [kJ]**	**Tasks / Node - Threads / Task**	**Performance [ns /day]**	**Energy to solution [kJ]**	**Tasks / Node - Threads / Task**	**Performance [ns / day]**
Test Case A								
1	48 - 2	59.652	76.2	48 - 1	38.381		48 - 2	47.539
2	48 - 2	94.901	112.2	48 - 1	85.908		48 - 2	84.157
4	48 - 2	145.120	136.8	48 - 1	132.370		48 - 2	118.165
8	48 - 2	235.283	200.0	48 - 1	183.030		48 - 2	167.114
16	48 - 2	305.863	335.8	48 - 1	240.189		48 - 2	194.980
Test Case B								
1	48 - 2	2.956	1194	48 - 1	2.462		48 - 2	3.235
2	48 - 2	5.695	1162	48 - 1	4.590		48 - 2	6.561
4	48 - 2	11.598	1184	48 - 1	8.125		48 - 2	12.621
8	48 - 2	21.352	1267	48 - 1	15.859		48 - 2	23.838
16	48 - 2	40.511	1420	48 - 1	27.764		48 - 2	41.741
32	48 - 2	60.335	1843	48 - 1	44.268		48 - 2	61.207
64	48 - 2	88.026	2657	48 - 1	67.036		48 - 2	86.013
Test Case C								
1	48 - 2	0.239	3587	48 - 1	0.191	3601		
2	48 - 2	0.480	3495	48 – 1	0.376	3671	48 - 2	0.470
4	48 - 2	0.956	3554	48 - 1	0.758	3792	48 - 2	0.889
8	48 - 2	1.960	3464	48 - 1	1.417	3844	48 - 2	1.797
16	48 - 2	3.190	4210	48 - 1	2.898	3765	48 - 2	3.286
32	48 - 2	6.461	4098	48 - 1	5.057	4494	48 - 2	6.008
64	48 - 2	10.648	5117	48 - 1	7.672	5971	48 - 2	9.715
128	48 - 2	13.817	8286	48 - 1	11.162	7594	48 - 2	10.970
256	48 - 2	25.382	10644	48 - 1				

Table 106: GROMACS performance on Skylake based Systems

	Piz Daint			MARCONI100	
Nodes	**Tasks / Node - Threads / Task**	**Performance [ns / day]**	**Energy to solution [kJ]**	**Tasks / Node - Threads / Task**	**Performance [ns /day]**
Test Case A					
¼				8 - 4/1 GPU	15.64
½				16 - 4/2 GPUs	23.26
1	4 - 6	43.729	71.5	16 - 8	56.367
2	4 - 6	74.680	84.7	16 - 8	59.461
4	4 - 6	110.815	108.9		
8	4 - 6	142.546	167.9		
16	4 - 6	165.268	267		
Test Case B					
1	4 - 6	3.104	773	32 - 4	10.939
2	4 - 6	6.098	772	32 - 4	15.003
4	12 - 2	10.132	991	32 - 4	21.071
8	12 - 2	20.056	985	32 - 4	21.235
16	12 - 2	35.530	1110	32 - 4	23.614
32	4 - 6	62.909	1290		
64	4 - 6	103.251	1380		
128	12 - 2	129.250	2340		
Test Case C					
1				32 - 4	0.506
2				32 - 4	0.986
4				32 - 4	1.700
8	12 - 2	1.231	3210	32 - 4	2.895
16	12 - 2	2.286	3660	32 - 4	4.361
32	12 - 2	3.833	4140	32 - 4	6.158
64	12 - 2	5.245	4650		
128	12 - 2	8.653	6220		
256	12 - 2	13.844	8920		

Table 107: GROMACS performance on GPU-based Systems

4.6.3 GROMACS Performance Comparison.

Benchmark measurements are presented in tables Table 105 – Table 107 grouped by system type and shown in Figure 15.

The two AMD EPYC based systems have similar performance at low node count given the difference in frequency. Irene is slightly faster with low node count. As node count increases, the execution speed is affected by the interconnect characteristics and parameters. At these ranges, the Hawk interconnect seems to exhibit better performance.

The three Skylake systems also exhibit similar behaviour given the differences in CPU frequency, hyperthreading settings and interconnect types.

The two GPU-based systems are quite different. Piz Daint has one GPU and interconnect per node while MARCONI100 has four GPUs and one interconnect per node. Test Case A is small enough to reach the full performance of GPUs for both systems. With Test Case B, it seems that one MARCONI100 node with its 4 GPUs has marginally higher performance than four Piz

Daint nodes. This is consistent with the similar in performance GPUs. As the node count increases the efficiency decreases, with MARCONI100 decreasing faster as function of number of nodes.

Comparing the three types of systems, with their relatively small differences in performance we see that AMD EPYC nodes are roughly two times faster than Skylake nodes. This is expected since AMD EPYC based nodes have roughly 2.5× number of cores. AMD EPYC systems exhibit higher performance per node than Piz Daint. MARCONI100 is faster when using up to 4 nodes and large datasets.

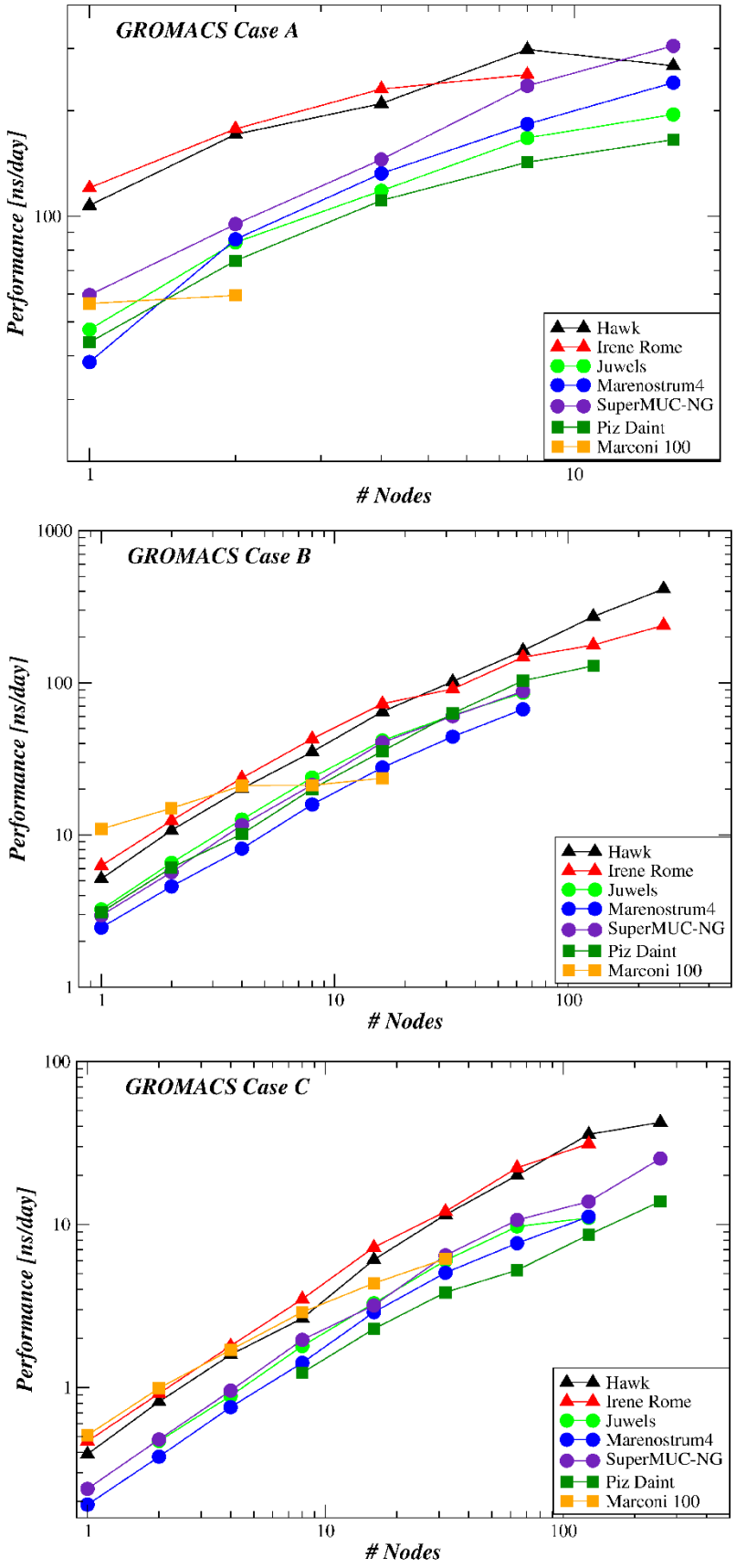

Figure 15: GROMACS performance comparison as function of number of nodes.

Job energy accounting was available for all jobs only on Piz Daint and partially on SuperMUC-NG and MareNostrum4. Energy to solution is similar between two types of machines with Piz Daint being more efficient. As node count increases the energy to solution increases. This is the result mainly of reduced parallel efficiency. The parallel efficiency with respect the lower possible node count is reported in Table 108 for all runs on all systems.

Nodes	Parallel Efficiency [%]						
	Hawk	Irene	JUWELS	MareNostrum4	SuperMUC	Piz Daint	MARCONI100
	Test Case A						
1	100.0	100.0	100.0	100.0	100.0	100.0	100.0
2	79.7	73.6	88.5	111.9	79.5	85.4	52.7
4	48.8	47.8	62.1	86.2	60.8	63.4	
8	34.8	26.3	43.9	59.6	49.3	40.7	
16	15.6		25.6	39.1	32.0	23.6	
Test Case B							
1	100.0	100.0	100.0	100.0	100.0	100.0	100.0
2	104.1	98.7	101.4	93.2	96.3	98.2	68.6
4	97.8	94.2	97.5	82.5	98.1	81.6	48.1
8	85.3	85.2	92.1	80.5	90.3	80.8	24.2
16	78.4	72.4	80.6	70.5	85.6	71.5	13.5
32	61.9	45.3	59.1	56.2	63.8	63.3	
64	49.4	36.6	41.4	42.5	46.5	52.0	
128	41.4	22.1					
256	31.4	14.8					
Test Case C							
1	100.0	100.0	100.0	100.0	100.0		100.0
2	104.9	98.1	94.6	98.4	100.4		97.1
4	101.9	96.1	95.6	99.2	100.0		83.3
8	85.1	93.1	87.4	92.7	102.5	100.0	71.1
16	97.3	96.6	79.9	94.8	83.4	92.8	53.4
32	91.9	80.2	64.6	82.7	84.5	77.8	37.7
64	80.2	74.3	36.5	62.7	69.6	53.3	
128	71.6	52.2		45.6	45.2	43.9	
256	42.4				41.5	35.1	

Table 108: GROMACS parallel efficiency for all Test Cases and Systems.

4.7 NAMD

4.7.1 System Software Environment

NAMD requires a C/C++ compiler, MPI except for GPU enabled builds, the FFTW library and for GPU support a CUDA SDK installation. The FFTW 3.3.8 library was compiled from source an all systems using the underlying compilers. In addition, to be able to compile, an extra flag

was added in the corresponding NAMD arch file: `arch/Linux-POWER.cuda10, --compiler-options=-mno-float128`. The software stack used on the machines is summarised in Table 109.

Machine	C/C++ compiler	MPI flavour	CUDA
Hawk	GCC 9	HP-MPI	
Irene	Intel 19	OpenMPI 4.0.2	
JUWELS	Intel 20	ParaStation MPI	
MARCONI100	IBM XL 16	ibverbs was used instead of MPI + hydra process management.	10.1
MareNostrum4	Intel 19	Intel MPI	
Piz Daint	Cray CC (GCC 8.3)	Cray GNI	10.2
SuperMUC-NG	Intel 19	Intel MPI	

Table 109: Software environment used in NAMD Benchmarks

4.7.2 Performance Results

NAMD uses hybrid MPI/ Threads parallelisation. For each task one has to reserve one core for the communicator between processes. Thus, the cores available for computation decrease by one for each task. Before running the full benchmarks on each machine, a number of test runs was performed in order to find the combination of tasks per node / threads per task that yields the best performance. NAMD reports various timings in its logfile. Typically, the WallClock reported at the end of logfile is what one needs. NAMD reads 2 large datafiles at startup, does the distribution among processes and then starts calculations. It was noted that this startup time has large deviations between machines and even when repeating runs on the same machine. In some cases, the startup time varied from 0.6 seconds up to 90 seconds. This can be omitted for runs where the calculation is hours or days as it happens in real production runs but introduces an inconsistency for runs where the real calculations need less than 1 minute. At the end of the run, NAMD also writes two large output datafiles, that also have variable times. For these reasons, the startup and final write files times have to subtracted from the total wall time. Startup time is reported in the logfile as `Info: Finished startup at` and writing final output files as `The last position output (seq=-2) takes` and `The last velocity output (seq=-2) takes`. The reported wall time has these startup and finishing times subtracted. Performance results are presented grouped by CPU type in Table 110 – Table 112. Energy to solution accounting was not available on all systems. On some of the systems, only partial energy accounting is available.

	Hawk		Irene	
Nodes	**Tasks / Node – Threads / Task**	**Wall Time [s]**	**Tasks / Node – Threads / Task**	**Wall Time [s]**
Test Case A				
1	8 - 16	4997.8		
2	16 - 8	2675.6		
4	4 - 32	1310.7	8 - 16	1429.8
8	4 - 32	670.4	8 - 16	779.6
16	16 - 8	346.6	16 - 8	419.2
32	8 - 16	181.0	8 - 16	240.0
64	8 - 16	93.4	8 - 16	136.7
128	4 - 32	65.5	8 - 16	83.3
256	4 - 32	38.3	16 - 8	38.6
512			16 – 8	29.6
Test Case B				
8	8 - 16	2224.9	8 - 16	2476.2
16	8 - 16	1144.0	8 - 16	1291.8
32	8 - 16	618.4	8 - 16	693.2
64	8 - 16	285.7	16 - 8	335.8
128	8 - 16	154.6	16 - 8	182.6
256	8 - 16	84.7	8 - 16	105.5
512	8 - 16	54.1	16 - 8	64.1
1024			16 - 8	47.4
Test Case C				
8	8 - 16	2375.0	16 - 8	2421.9
16	8 - 16	1180.4	16 - 8	1191.1
32	8 - 16	589.2	8 - 16	609.0
64	8 - 16	322.4	8 - 16	306.9
128	8 - 16	161.0	8 - 16	166.5
256	8 - 16	90.8	8 - 16	96.1
512	8 - 16	45.6	16 - 8	48.7
1024			16 - 8	34.3

Table 110: NAMD performance on AMD EPYC based Systems

	SuperMUC-NG			MareNostrum4			JUWELS	
Nodes	**Tasks / Node – Threads / Task**	**Wall time [s]**	**Energy to solution [kJ]**	**Tasks / Node – Threads /Task**	**Wall time [s]**	**Energy to solution [kJ]**	**Tasks / Node - Threads / Task**	**Wall time [s]**
Test Case A								
1	2 - 48	13157.9	4996	2 - 24	15429.5			
2	2 – 48	6629.1	5367	2 - 24	7913.8			
4	2 - 48	3597.0	5151	2 - 24	3991.4			
8	2 - 48	2243.0	7194	2 - 24	2013.2			
16	2 - 48	1308.3	7783	2 - 24	1050.9		2 - 48	737.9
32	24 - 4	637.0	8878				2 - 48	475.4
64	24 - 4	447.9	11744				2 - 48	291.2
128	24 - 4	383.3	19211				2 - 48	262.1
Test Case B								
2	8 - 12	23413.2	18280	2 - 24	26790.4			
4	8 - 12	11497.8	19343	2 - 24	13478.0			
8	8 - 12	6518.0	18956	2 - 24	6823.9		2 - 48	5234.1
16	8 - 12	2920.0	20112	2 - 24	3519.3		2 - 48	2447.4
32	8 - 12	1839.3	19855	2 - 24	1756.8		2 - 48	1271.6
64	8 - 12	940.2		2 - 24	921.2		2 - 48	827.5
128	8 - 12	624.0	32672	2 - 24	548.8	23134	2 - 48	564.3
256	24 - 4	489.4	53203	4 - 12	330.0			
Test Case C								
16	8 - 12	2985.9	19344	2 - 24	3512.3	19368	2 - 48	2523.8
32	2 - 48	1412.1	18184	2 - 24	1934.5	19323	2 - 48	1288.0
64	2 - 48	726.1	19098	2 - 24	887.0	20063	2 - 48	640.5
128	2 - 48	383.8	21191	2 - 24	452.0	21253	2 - 48	339.0
256	2 - 48	208.4	26463	2 - 24	238.1	23556		
512	2 - 48	136.2	37460					
1024	8 - 12	104.1						

Table 111: NAMD performance on Skylake based Systems

	Piz Daint			MARCONI100	
Nodes	**Tasks / Node - Threads / Task**	**Wall time [s]**	**Energy to solution [kJ]**	**Tasks / Node - Threads / Task**	**Wall time [s]**
Test Case A					
1	1 - 24	4295	1010	4 - 32	5639.2
2	1 - 24	2189	1060	4 - 32	1034.0
4	1 - 24	1152	1100	4 - 32	766.6
8	1 - 24	628	1240	4 - 32	1641.4
16	1 - 24	351	1340		
32	1 - 24	205	1500		
64	1 - 24	122	1700		
Test Case B					
2				4 - 32	5050.4
4	1 - 24	4573.2	4070	4 - 32	2441.5
8	1 - 24	2040.3	4080	4 - 32	1984.3
16	1 - 24	1029.6	4050	4 - 32	2559.7
32	1 - 24	579.7	4370		
64	1 - 24	323.2	4640		
128	1 - 24	192.5	7610		
256	1 - 24	140.4	6730		
Test Case C					
6				4 - 32	1059.7
8	1 - 24			4 - 32	735.6
16	1 - 24			4 - 32	376.2
32	1 - 24	578.6	4700	4 - 32	224.6
64	1 - 24	285.3	4760		
128	1 - 24	146.3	5280		
256	1 - 24	87.1	6240		
512	1 - 24	82.2	9870		

Table 112: NAMD performance on GPU-based Systems

4.7.3 NAMD Performance Comparison

Benchmark results for NAMD are presented in Table 110 – Table 112 grouped by system type. The two AMD EPYC based systems have similar performance as function of number of nodes, with Hawk being slightly faster.

The three Skylake based systems have also similar performance given the differences in CPU frequency and interconnect.

Results from GPU accelerated systems start from a number of nodes that provide the necessary amount of GPU memory. This is quite different between Piz Daint and MARCONI100 due to the number of GPUs per node. The minimum number of nodes to run Test Case C is 6 for MARCONI100 and 32 for Piz Daint.

Comparing the three types of systems, with their relatively small differences in performance for each type, we see that the AMD EPYC based nodes are 2 to 2.5 faster than Skylake based nodes. AMD EPYC based systems exhibit with all Test Cases similar performance with Piz Daint using the same number of nodes. Finally, although the number of performance measurements is small on MARCONI100 compared to the other systems, it seems to be roughly 2–3 times faster than AMD EPYC based machines at low node counts with Test Cases B and C.

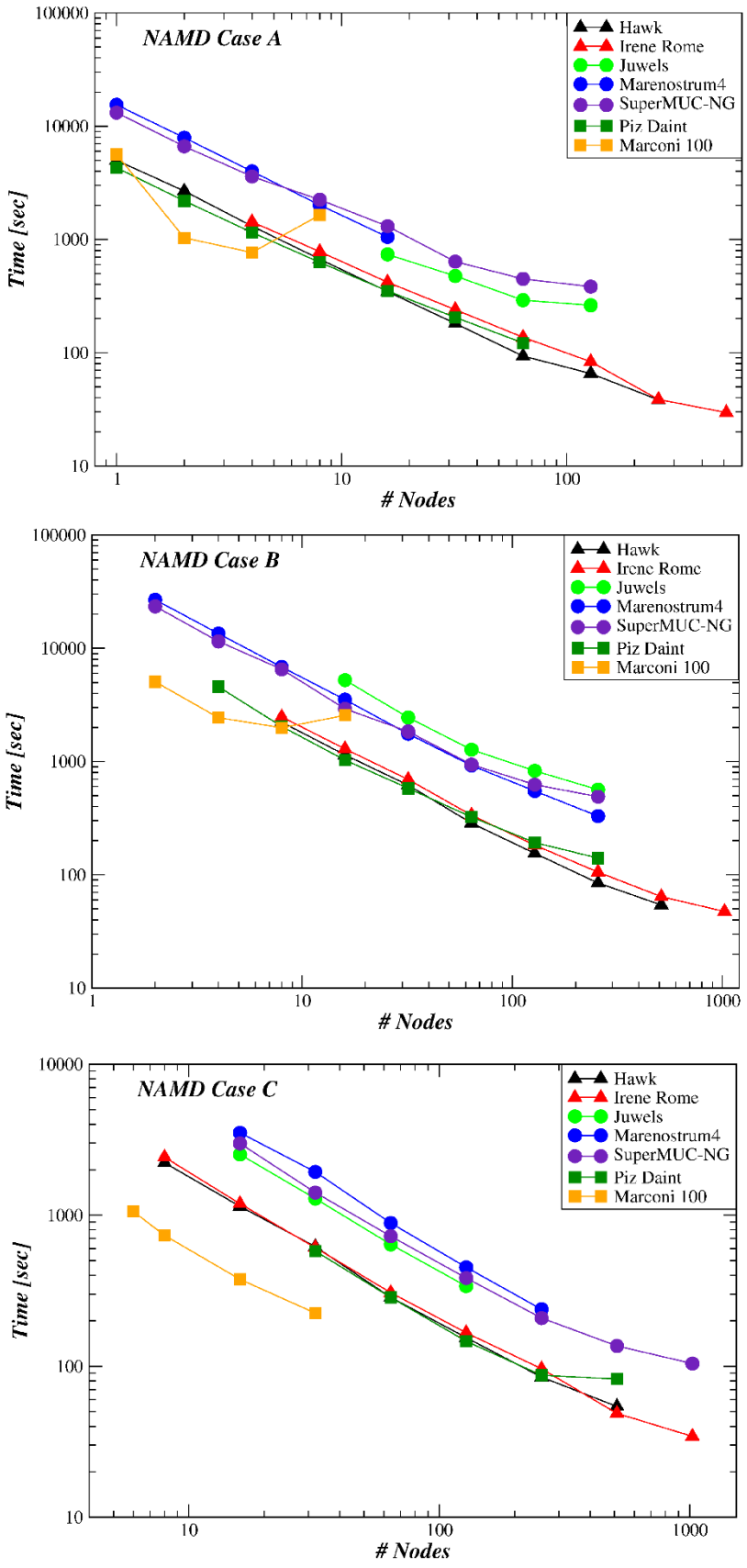

Figure 16: NAMD Performance as function of number of nodes.

Job energy accounting was available for all jobs only on Piz Daint and partially on SuperMUC-NG. It seems from these measures that Piz Daint is roughly 3–5 times more energy efficient than SuperMUC-NG. This is expected since Piz Daint is GPU accelerated.

4.8 NEMO

Comparative benchmarking of NEMO has been performed on the homogenous CPU system Hawk, Irene (both Skylake and Rome partitions), JUWELS, MARCONI100 (CPU only), MareNostrum4, and SuperMUC-NG. We did not use Piz Daint since it is only available to users who use GPU.

4.8.1 Installation

We have installed NEMO version 4.0 with XIOS 2.5 on all machines. We used Intel compilers on Irene, JUWELS, MareNostrum4, SuperMUC-NG; and GNU compilers on Hawk and MARCONI100. We used Intel MPI on MareNostrum4 and SuperMUC-NG, OpenMPI on Irene, ParaStation MPI on JUWELS, MPT MPI on Hawk, and IBM Spectrum MPI on MARCONI100.

4.8.2 Performance Results

We report the performance in terms of total time to solution as well as total consumed energy to solution whenever possible. This helps us to compare systems in a standard manner across all combinations of system architectures.

As we mentioned NEMO supports both attached and detached mode of the IO server. In the attached mode all cores perform both computation and IO, whereas in the detached mode each core performs either computation or IO. It is reported that NEMO performs better with detached mode for especially large number of cores [3]. Therefore, we performed benchmarks for both attached and detached modes. We utilise 15:1 ratio for the detached mode [3]. That is, we divide 1024 cores as 960 compute cores and 64 IO cores for Test Case A, whereas we divide 10240 cores as 9600 compute cores and 640 IO cores for Test Case B.

Performance comparison between Test Cases A and B run on 1024 and 10240 processors, respectively, can be considered as something between weak and strong scaling. That is, number of processors are increased ten times, however the increase in the mesh size is approximately 16 times, when we go from Test Case A to B. We should note here that MARCONI100 does not allow us to use more than 8192 cores, so we used 8192 cores for attached and 7680+512 cores for detached mode of Test Case B. Table 113 shows the number of allocated nodes for each test case and for each Tier-0 system. Note that as we will discuss later, we allocate more than enough nodes for some cases.

Tier-0 System	Test Case A		Test Case B (16 times larger than Test Case A)	
	attached (1024 cores)	detached (960+64 cores)	attached (10240 cores)	detached (9600+640 cores)
Hawk	8	8	120	80
Irene-Rome	8	8	128	80
Irene-Skylake	22	22	214	214
JUWELS	22	22	320	214
MARCONI100	32	32	256	256
MareNostrum4	22	22	300	214
SuperMUC-NG	22	22	285	214

Table 113: Number of allocated nodes for NEMO for each Test Case and for each machine.

Table 114 shows the time to solution and energy to solution values for both attached and detached modes obtained on various Tier-0 systems. In the table "-" denotes not applicability. For example, Hawk does not log energy.

Tier-0 System	Test Case A				Test Case B (16 times larger than Test Case A)			
	attached (1024 cores)		detached (960+64 cores)		attached (10240 cores)		detached (9600+640 cores)	
	Time (s)	Energy (kJ)	Time (s)	Energy (kJ)	Time (s)	Energy (kJ)	Time (s)	Energy (kJ)
Hawk	13.98	-	17.63	-	41.03	-	62.05	-
Irene-Rome	21.41	167.61	18.45	44.09	73.89	5,674.85	43.01	2,903.77
Irene-Skylake	18.11	327.72	18.72	393.43	81.32	9,509.90	46.27	6,570.76
JUWELS	14.92	-	20.82	-	474.89	-	272.36	-
MARCONI100	15.03	-	14.84	-	117.98	-	76.96	-
MareNostrum4	18.96	366.01	17.63	357.07	112.76	54,357,76	62.05	29,649.80
SuperMUC-NG	19.54	-	29.58	-	115.33	-	62.58	-

Table 114: Time and energy to solution of NEMO for both test cases on 1024 and 10240 cores of each machine.

Results obtained from Test Case A attained with attached mode show that systems display comparable performance (between about 15 and 20 seconds), where Hawk, JUWELS and MARCONI100 attain relatively better performance (about 15 seconds). Results attained with detached mode show that Hawk, Irene (both Rome and Skylake clusters) and MareNostrum4 display very close performance (about 18 seconds), whereas MARCONI100 shows considerably better performance (about 15 seconds) and SuperMUC-NG shows considerably worse performance (about 30 seconds). When attached and detached modes are compared, mostly detached mode attains better performance than the attached mode as expected. Detached mode performs worse than the attached mode for Hawk, JUWELS, and SuperMUC-NG. We believe that this stems from the distribution of the IO and computation cores among nodes.

Results obtained from Test Case B show variance across Tier-0 systems, where Hawk is the best for attached mode, whereas Irene-Rome partition is the best for the detached mode. Recall that time to solution values displayed in Table 114 contain IO times. We observed that the application becomes IO bound as the number of cores increases. This is because each core writes its output to a file periodically. The most notable example for the IO bound anomaly is the running of Test Case B with attached mode on JUWELS. The running time increases to 474.89 seconds, which is more than the 10 times of the fastest machine (Hawk). Therefore, for a better comparison of the Tier-0 systems, we also present solution times excluding IO times in Table 115. As seen in Table 115, JUWELS achieves the fastest running time on Test Case B with attached mode excluding IO times.

Tier-0 System	Test Case A		Test Case B (16 times larger than Test Case A)	
	attached (1024 cores)	detached (960+64 cores)	attached (10240 cores)	detached (9600+640 cores)
	Time (second)	Time (second)	Time (second)	Time (second)
Hawk	7.92	11.20	15.62	35.22
Irene-Rome	10.27	11.72	23.64	20.94
Irene-Skylake	8.92	9.46	20.66	18.32
JUWELS	7.54	13.15	10.17	41.64
MARCONI100	6.39	9.18	15.16	64.17
MareNostrum4	7.32	11.21	12.27	35.22
SuperMUC-NG	9.78	25.07	14.99	36.83

Table 115: Time to solution (excluding IO time) of NEMO for both test cases on 1024 and 10240 cores of each machine.

Regarding the above reported experimental results, we should first note that the amount of work done for both attached and detached modes are the same. Therefore, the amount of work per core is higher for the detached mode, since some of the cores are only responsible for the IO operations. That is the increase in the time values of the detached mode compared to attached mode is expected. We should also note here that Test Case B has the larger grid size to be solved. That is these values show a kind of weak scaling results. We increase the problem size about $4 \times 4 = 16$ times, whereas we increase the number of processors 10 times. As a result, we achieve about 10 times speed-up on JUWELS and SuperMUC-NG, 8 times speed-up on Hawk, and 7 times on Irene for the attached mode. These speed-up values are obtained by dividing the runtime of large test by 16, in order to obtain speed-up values like strong scaling. On the other hand, although the detached mode achieves generally much less actual times, it does not scale as good as attached mode.

We should also note that Test Case B using attached mode, main memory of a single node becomes insufficient for some Tier-0 systems if we use all cores of each node. In order to overcome this problem, we allocate more than enough nodes and use some of the cores in each node. For example, on Hawk, there are 128 cores on a node, but we allocated 120 nodes (as seen in Table 113) instead of 80 nodes to reach total number of cores of 10240. Similarly, we allocated 128 nodes on Irene-Rome. For JUWELS, MareNostrum4 and SuperMUC-NG each node has 48 cores. As also seen in Table 113, we allocated 320, 300 and 285 nodes on JUWELS, MareNostrum4 and SuperMUC-NG, respectively. We tried to select the minimum number of nodes, which enables running the application.

4.9 PFARM

4.9.1 Benchmarking Setup

PFARM benchmark runs were undertaken on a range of PRACE Tier-0 systems. The compilers and numerical libraries used are summarised Table 117. Compilation optimisation is undertaken through -Ofast options (also -mtune=skylake where appropriate).

4.9.1.1 Hybrid MPI / OpenMP Configurations on CPUs

Performance experiments with alternative placement of threads and bindings have been undertaken on a range of compute nodes. It has been established that the default thread placements usually give optimal performance for PFARM (EXDIG). For hybrid MPI/OpenMP runs involving 1 MPI task per node it is often beneficial to performance to under-populate the

node with the number of runtime cores used (mainly due to memory bandwidth saturation). We found this to be the case on the two AMD EPYC Rome-based systems tested (32/128 available cores utilised) and two of the Intel Xeon-based systems (24/48 cores utilised), see Table 116. It was found that configuring the hybrid runs with multiple MPI tasks per node was always advantageous to performance, though memory limits on compute nodes necessitate that the maximum number of MPI tasks per node is restricted to 4 (each MPI task needs to store the data for a complete sector calculation). Due to the computational characteristics of the dense linear algebra involved, hyperthreading is found to be never advantageous for this code.

System (CPU Node Architecture)	Number of physical CPU cores per node	Node config. for 1 MPI task / Cores per MPI task	Node config. for multiple MPI tasks / Cores used per MPI task
Hawk (AMD EPYC Rome)	128	1 / 32	4 / 16
Irene-Rome (AMD EPYC Rome)	128	1 / 32	4 / 16
Irene-SKL (Intel Xeon Skylake)	48	1 / 48	4 / 12
JUWELS (Intel Xeon Skylake)	48	1 / 24	4 / 12
MareNostrum4 (Intel Xeon Skylake)	48	1 / 48	4 / 12
SuperMUC-NG (Intel Xeon Skylake)	48	1 / 24	4 / 12

Table 116: Hybrid MPI/OpenMP configurations used for PFARM (EXDIG)

4.9.1.2 GPU Node Configurations

Benchmark runs on GPU accelerated nodes use a combination of MPI and CUDA (CUDA within the MAGMA library) for the parallelisation. Computations involving MAGMA have the capacity to distribute the matrix computations across multiple GPU devices, if required. We found this feature to be beneficial, rather than assigning specific matrix computations to specific single GPU devices. Setups involving 1 MPI task per node and 4 MPI tasks per node were used for the benchmarking. Both these configurations use the 4 GPU devices available on MARCONI100 and JUWELS Booster. Using 4 MPI tasks per node proved to be universally beneficial to performance for the benchmark datasets.

4.9.1.3 Numerical Libraries used for the Eigensolver Calculation

Previous benchmarking exercises have determined that the overwhelming bulk of the compute time is usually spent undertaking symmetric sector Hamiltonian matrix diagonalisations [34] inside the numerical library eigensolver routine DSYEVD. DSYEVD uses a highly efficient and robust algorithm that exploits a divide-and-conquer approach for determining eigenvectors and eigenvalues of a symmetric matrix [44]. Implementations of this routine are available in LAPACK, MKL, ESSL, and MAGMA numerical libraries. The recommended higher-level dense linear algebra library for AMD architecture, LibFLAME [45] does not support a DSYEVD implementation, therefore a combination of Intel MKL and the recommended BLIS library for BLAS functionality was used [45]. The Intel MKL library has not been specifically tuned for AMD processors, but the environment settings MKL_DEBUG_CPU_TYPE=5 and

MKL_ENABLE_INSTRUCTIONS=AVX2, as advised by the Hawk machine documentation, were set for both benchmarked AMD platforms.

Performance tests were run for all Test Cases 1a, 1b, 1c and 1d, however this section will analyse results from Test Cases 1c and 1d, as these are the most appropriate for modern large-scale HPC architectures and systems.

Machine	Compiler	Numerical Libraries
Hawk	GNU Fortran, gcc v9.2.0	Intel MKL, BLIS
Irene-Rome	Intel Fortran v2020	Intel MKL
Irene-Skylake	Intel Fortran v2020	Intel MKL
JUWELS	Intel Fortran v2020	Intel MKL
JUWELS Booster	Intel Fortran v2020	Magma v2.5.4 / Intel MKL
MARCONI100	GNU Fortran, gcc v8.4.0	Magma v2.5.3, ESSL v6.2.1 LAPACK v3.9.0, OpenBLAS 0.3.9
MareNostrum4	Intel Fortran v2020	Intel MKL
SuperMUC-NG	Intel Fortran v2020	Intel MKL

Table 117: Numerical Libraries used for PFARM (EXDIG)

4.9.2 PFARM Performance Results

All performance charts included show the PRACE Tier-0 systems grouped by architecture, i.e. from left to right, two AMD Rome-based systems, four Intel Xeon-based systems and two NVIDIA GPU-accelerated systems. All the runs are MPI/OpenMP parallelised apart from the runs on the two GPU accelerated systems which use MPI and CUDA.

4.9.2.1 Test Case 1c

Figure 17 shows the performance results for Test Case 1c on the PRACE Tier-0 systems. The timings from JUWELS Booster are significantly faster than any other systems. This is primarily due to the performance of the new NVIDIA A100 devices. MAGMA has now been optimised for NVIDIA A100 GPUs [46] and performance 2–3 times faster than on comparable NVIDIA V100 GPU systems is to be expected, depending on the problem size. This is evident here when comparing JUWELS-Booster performance against the V100-based MARCONI100, though it should also be noted that the CPU hosts differ between the two machines, which will also affect overall performance. As expected, the four Xeon Skylake-based architectures tested produce similar performance figures, with JUWELS marginally fastest. The two AMD EPYC Rome-based systems, Hawk and Irene-Rome are slowest for this calculation, mainly due to the relatively slow performance of the Sector Hamiltonian eigensolver calculation (see Figure 19 and Figure 20). Single node performance on both AMD machines is quite variable and generally relatively slow compared to more highly parallelised runs (hence parallel efficiencies of over

often 100% reported in Table 120). This appears to be due to the variability of I/O overheads for the lengthy single node runs. These overheads become less variable for the shorter runs involving more nodes. For Test Case 1c, parallel efficiency, defined as $(\frac{Speed-up}{Ideal\ Speed-up} \times 100)$, is generally excellent on all the systems tested, ranging from 84.9% on JUWELS Booster to around 88–96% on the Xeon-based systems and to around 100% on Hawk and Irene-Rome.

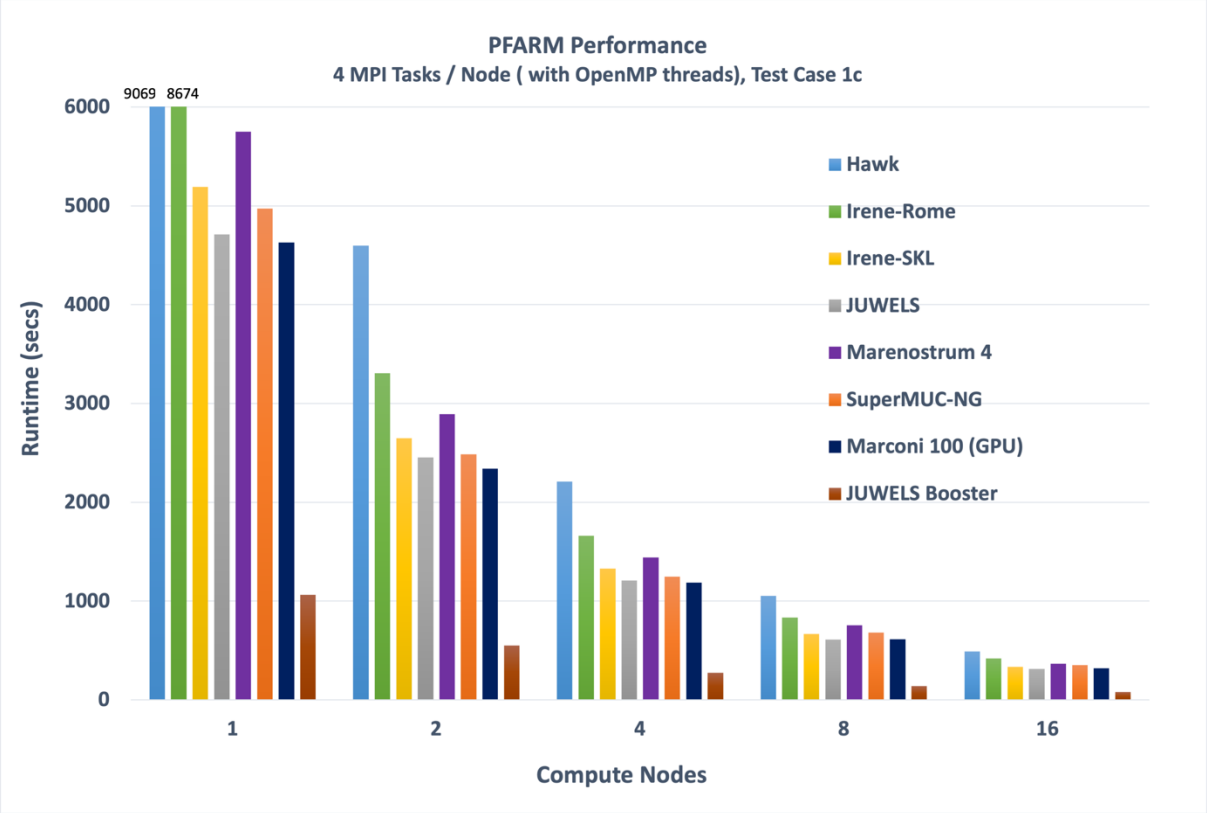

Figure 17: Parallel performance of PFARM (EXDIG) on PRACE Tier-0 systems for Test Case 1c

4.9.2.2 Test Case 1d

Figure 18 shows the performance results for Test Case 1d on the PRACE Tier-0 systems. Here each sector Hamiltonians is of smaller dimension than 1c, but the number of sectors is increased to 1024. The dataset is suitable for scaling to large numbers of compute nodes and due to the smaller matrix size a relatively smaller proportion of time is spent in the eigensolver (matrix eigensolver operations required are of order n^3). For example, on JUWELS around 78% time is spent in DSYEVD for Test Case 1c vs 92% for Test Case 1d). Matrix data transfer between host and device (order n^2) also relatively impacts more on performance when n is smaller (i.e. surface area to volume ratio).

Again, JUWELS Booster is the fastest machine by a significant margin, though performance scaling is stalling between 128 and 256 nodes, where only a 7% improvement is achieved. This scaling slowdown is mainly due to I/O becoming more of a relative overhead, as two large output files are produced for each sector. MARCONI100 performance was again slower, but at the largest scale was only 2.6× slower than JUWELS Booster, down from 4.1× with Test Case 1c. Although the parallel efficiency for this range of node counts on the CPU-only systems

is once again very good (79–100%), the AMD platforms are again slowest, but by a less significant margin, especially for Irene-Rome. The Intel Xeon Skylake machines all produce similar performance results, though parallel efficiency for these node counts varies between 77% on JUWELS to 95% on MareNostrum4 indicating superior I/O performance on the latter. SuperMUC-NG was the fastest Xeon-based machine at scale.

Figure 18: Parallel performance of PFARM (EXDIG) on PRACE Tier-0 systems for Test Case 1d

4.9.2.3 Performance Benchmark Data

For completeness, the raw performance data from the benchmarked machines is listed in Table 118 – Table 121. Parallel efficiency is computed as $(\frac{Speed-up}{Ideal\ Speed-up} \times 100)$.

Nodes (Total MPI Tasks)	Cores Used	Time (s)	Speed-up	Efficiency	Time (s)	Speed-up	Efficiency
		JUWELS (Intel Xeon)			MareNostrum4 (Intel Xeon)		
Test Case 1c							
1 (4)	48	4710.84	1.00	100 %	5897.28	1.00	100 %
2 (8)	96	2452.57	1.92	96.04 %	2968.36	1.99	99.33 %

Nodes (Total MPI Tasks)	Cores Used	Time (s)	Speed-up	Efficiency	Time (s)	Speed-up	Efficiency
		JUWELS (Intel Xeon)			MareNostrum4 (Intel Xeon)		
4 (16)	192	1207.97	3.90	100 %	1485.79	3.96	99.23 %
8 (32)	384	611.25	7.71	96.34 %	756.16	7.80	96.49 %
16 (64)	768	313.63	15.02	93.87 %	380.34	15.51	96.91 %
Test Case 1d							
64 (256)	3072	541.03	1.00	100.00 %	619.21	1.00	100.00 %
128 (512)	6144	367.32	1.47	73.64 %	330.85	1.87	93.58 %
256 (1024)	12288	175.59	3.08	77.03 %	170.74	3.63	90.67 %

Table 118: Parallel Performance of PFARM (EXDIG) on Xeon-based systems (i) for Test Cases 1c and 1d

Nodes (Total MPI Tasks)	Cores Used	Time (s)	Speed-up	Efficiency	Time (s)	Speed-up	Efficiency
		SuperMUC-NG (Intel Xeon)			Irene-SKL (Intel Xeon)		
Test Case 1c							
1 (4)	48	4972.55	1.00	100.00 %	5191.65	1.00	100.00 %
2 (8)	96	2486.06	2.00	100.01 %	2646.55	1.96	98.08 %
4 (16)	192	1246.05	3.99	99.77 %	1329.86	3.90	97.60 %
8 (32)	384	682.37	7.29	91.09 %	666.51	7.79	97.37 %
16 (64)	768	350.84	14.17	88.58 %	335.87	15.46	96.61 %
Test Case 1d							
64 (256)	3072	525.57	1	100 %	574.97	1.0	100.00 %
128 (512)	6144	265.07	1.98	99.14 %	305.83	1.88	94.00 %
256 (1024)	12288	135.80	3.87	96.75 %	188.11	3.06	76.41 %

Table 119: Parallel Performance of PFARM (EXDIG) on Xeon-based systems (ii) for Test Cases 1c and 1d

Nodes (Total MPI Tasks)	Cores Used	Time (s)	Speed-up	Efficiency	Time (s)	Speed-up	Efficiency
		Hawk (AMD Rome)			Irene-Rome (AMD Rome)		
Test Case 1c							
1 (4)	64	9734.95	1.00	100.00 %	8674.32	1.00	100.00 %
2 (8)	128	4809.14	2.02	101.21 %	3305.60	2.62	131.21 %
4 (16)	256	2387.8	4.08	101.92 %	1662.62	5.21	130.43 %
8 (32)	512	1178.34	8.26	103.27 %	831.83	10.43	130.43 %
16 (64)	1024	507.61	19.18	119.86 %	418.35	20.73	129.95 %
Test Case 1d							
64 (256)	4096	1002.84	1.00	100.00 %	680.81	1.00	100.00 %
128 (512)	8192	442.63	2.27	113.28 %	350.93	1.94	97.00 %
256 (1024)	16384	246.81	4.06	101.58 %	214.28	3.17	79.43 %

Table 120: Parallel Performance of PFARM (EXDIG) on AMD-based systems for Test Cases 1c and 1d

Nodes (Total MPI Tasks)	Cores Used*	Time (s)	Speed-up	Efficiency	Time (s)	Speed-up	Efficiency
		Marconi100 (IBM Power & V100 GPU)			Juwels-Booster (AMD Rome & A100 GPU)		
Test Case 1c							
1 (4)	N/A*	4630.74	1.00	100.00 %	1061.39	1.00	100.00 %
2 (8)	N/A*	2340	1.98	98.93 %	548.95	1.93	99.33 %
4 (16)	N/A*	1207.97	3.90	97.49 %	275.53	3.85	99.23 %
8 (32)	N/A*	611.25	7.53	94.18 %	141.42	7.51	96.49 %
16 (64)	N/A*	313.63	14.43	90.24 %	78.15	13.58	96.91 %
Test Case 1d							
64 (256)	N/A*	541.03	1.00	100.00 %	164.83	1.00	100.00 %
128 (512)	N/A*	367.32	1.94	96.96 %	80.00	2.06	103.02 %
256 (1024)	N/A*	175.59	3.70	92.49 %	74.76	2.21	55.12 %

Table 121: Parallel Performance of PFARM (EXDIG) on GPU-accelerated systems for Test Cases 1c and 1d. (* The number of cores to be used in MAGMA calculations is not user-specified, though MAGMA uses pthread parallelism for some CPU-based operations)

4.9.2.4 Eigensolver Performance

PFARM (EXDIG) calculations primarily involve assembling and diagonalising symmetric real sector Hamiltonian matrices to determine all eigenvalues and associated eigenvectors to then calculate sector surface amplitude matrices of the same dimension for output. The performance of the numerical library-based eigensolver is therefore central to the performance of the overall calculation and is worthy of some analysis. Matrix diagonalisations take place within compute nodes so shared memory routines are used. Figure 19 and Figure 20 summarise the intra-node performance of DSYEVD for two different Hamiltonian matrices in EXDIG from test Case 1c. The relative performance of DSYEVD on the different machines maps closely to the overall run times from Figure 17 and Figure 18. For example, in a calculation involving 4 sector Hamiltonian matrices, the figures show that using 4 concurrent MPI tasks per node each with ¼ of physical cores as OpenMP threads is preferable to using 1 MPI task with all compute node resources 4 times in succession for each sector calculation. However, for very large matrices it may be necessary to utilise all the memory on a single node via 1 MPI task. The 4 MPI tasks, 4 NVIDIA A100 performance of MAGMA is particularly impressive for the larger matrix, comparatively around 4× faster than the NVIDIA V100 and around 6× faster than the Xeon-based MKL eigensolver. For the smaller matrix, MAGMA performance is relatively lower due to higher (order n^2) host-device data transfer overheads relative to matrix computation operations (order n^3).

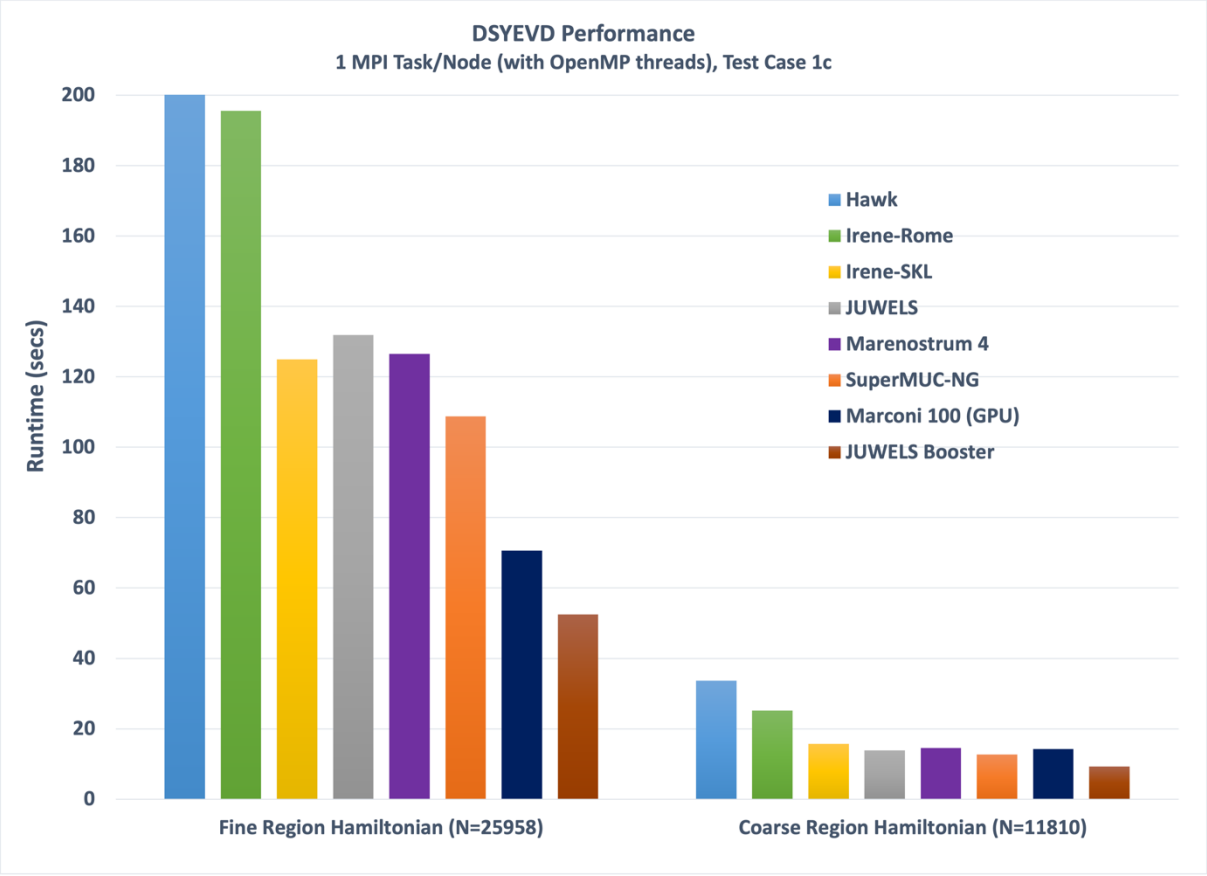

Figure 19: Sector Hamiltonian Eigensolver performance using DSYEVD with 1 MPI task per node.

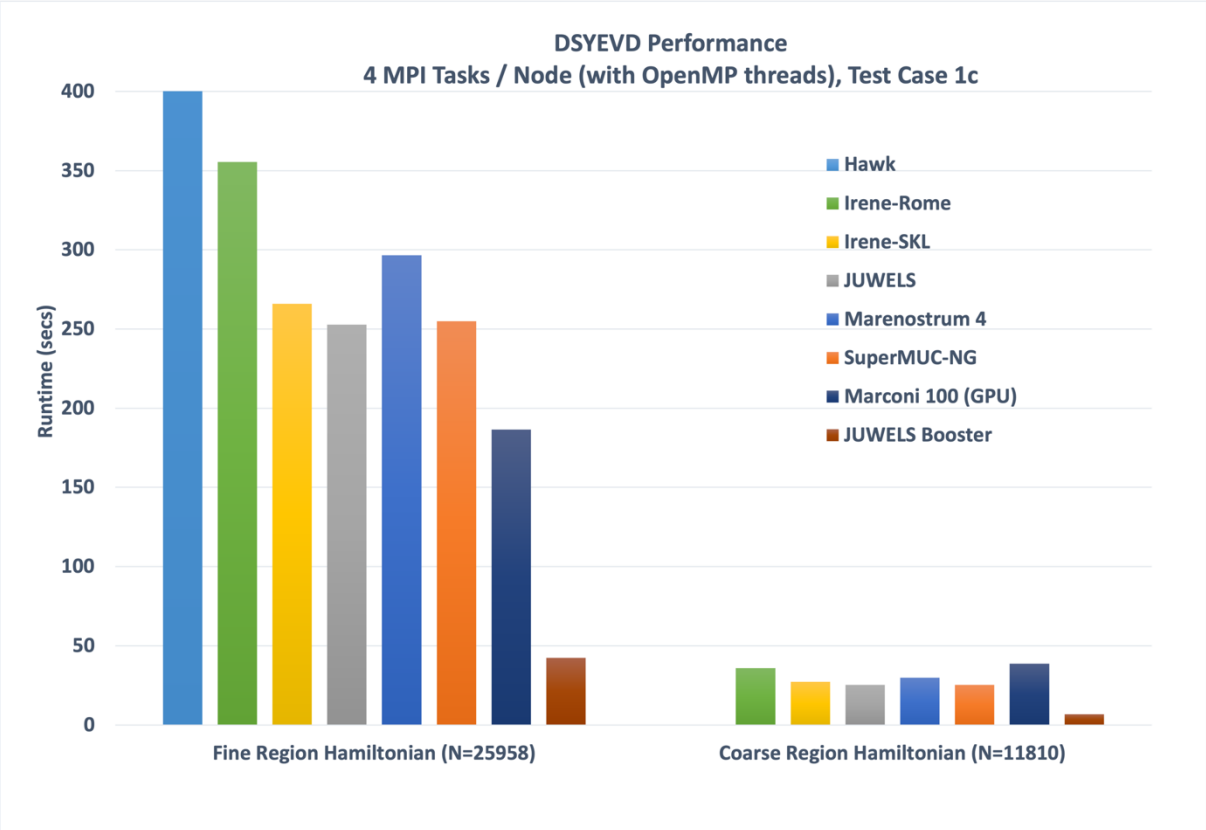

Figure 20: Sector Hamiltonian Eigensolver performance using DSYEVD with 4 MPI tasks per node

4.9.2.5 Energy Usage of Benchmark Runs

Energy monitoring has been used to collect data from four machines. Three of these are CPU-based machines – Irene-SKL, Irene-Rome, MareNostrum4 and one is a CPU/GPU-accelerated system – MARCONI100. Energy used for the benchmarking runs is summarised in Table 122 and Table 123. The energy reports from MARCONI100 only report energy usage data from the GPU devices, rather than the whole CPU/GPU node. The MARCONI100 values tabulated are therefore unsuitable for direct comparisons with the other platforms. Of the three CPU node systems, Irene-SKL is the most energy efficient with Irene-Rome only a little behind. Comparable runs on MareNostrum4 are reporting a higher level of power usage.

Generally, the pattern of energy consumption for runs using a range of nodes on a machine tends to be proportional to parallel scaling efficiency. For example, the energy consumption of Irene-SKL for Test Case 1d is 19% lower on 64 nodes than 256 nodes. The associated drop in parallel scaling efficiency (from ideal scaling) between the two node counts is 24%. Likewise, for MareNostrum4, energy consumption on 64 nodes is around 7% lower than on 256 nodes relating to a drop in parallel scaling efficiency of 5%.

Nodes	Total Energy (kJ)			
	Irene-SKL		Irene-Rome	
	1 MPI Task / 48 OpenMP threads per task	4 MPI tasks / 12 OpenMP threads per task	1 MPI Task / 32 OpenMP threads per task	4 MPI tasks / 16 OpenMP threads per task
1	1128	2507	1844	2496
2	1166	2566	1931	2567
4	1189	2580	1265	2585
8	1213	2591	1212	2582
16	1247	2605	1246	2619

Nodes	Total Energy (kJ)			
	MareNostrum4		MARCONI100 (GPU usage only)	
	1 MPI Task / 48 OpenMP threads per task	4 MPI tasks / 12 OpenMP threads per task	1 MPI Task / 4 GPU devices	4 MPI tasks / 4 GPUs devices
1	1885	2321	512	1121
2	1856	3498	479	1418
4	1879	4125	484	1417
8	1882	4143	422	1446
16	1914	4464	429	1383

Table 122: Energy Consumption of PFARM (EXDIG) benchmarking runs for Test Case 1c.

Nodes	Total Energy (kJ)			
	Irene-SKL		Irene-Rome	
	1 MPI Task / 48 OpenMP threads per task	4 MPI tasks / 12 OpenMP threads per task	1 MPI Task / 32 OpenMP threads per task	4 MPI tasks / 16 OpenMP threads per task
64	18387	17248	20709	16643
128	18485	17790	21423	17411
256	19120	20521	21941	20947

Nodes	Total Energy (kJ)			
	MareNostrum4		MARCONI100 (GPU usage only)	
	1 MPI Task / 48 OpenMP threads per task	4 MPI tasks / 12 OpenMP threads per task	1 MPI Task / 4 GPU devices	4 MPI tasks / 4 GPUs devices
64	30635	28504	7770	12833
128	30731	29005	7898	13136
256	31544	30449		13952

Table 123: Energy Consumption of PFARM (EXDIG) benchmarking runs for Test Case 1d.

4.10 QCD

The QCD kernels that are part of the UEABS Benchmark Suite are unchanged since PRACE-4IP, with the exception of adding software packages with capabilities to run on accelerator devices, like Intel Xeon Phi processors or NVIDIA GPGPUs. Since PRACE-4IP these kernels have benchmarked most of the PRACE Tier-0 systems, with the exception of

Irene-Rome, which makes it possible to compare the older Tier-0 systems with state-of-the-art machines, like JUWELS Booster.

While for most of the machines performance results are provided, energy consumption measurements of the QCD-test cases are very limited. This is due to the fact that new results within PRACE-6IP were obtained only on the newest machines, namely Hawk, SuperMUC-NG, JUWELS Booster and MARCONI100, which are all missing a log of energy usage for single jobs.

4.10.1 Test Case: Part 1

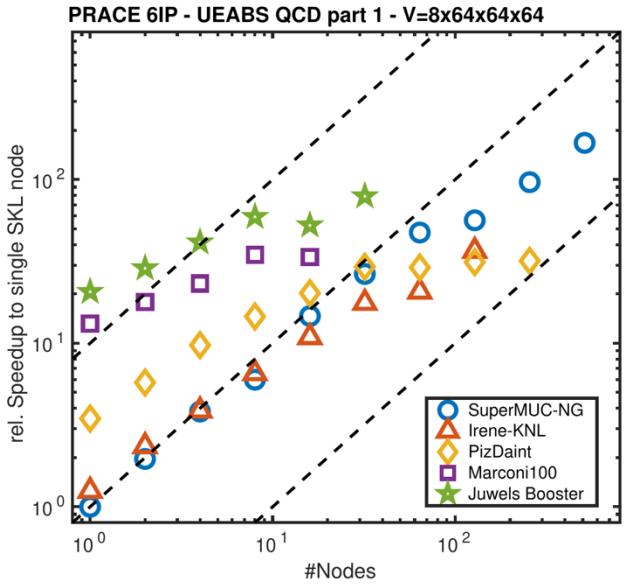

Figure 21: Relative speed-up of the performance using UEABS QCD Part 1 compare to the single node performance on SuperMUC-NG equipped with Intel Xeon Skylake chips. For the benchmark application strong scaling towards multiple nodes on PRACE Tier-0 machines with a test size of V=8×64×64×64 is used.

The test size of benchmark kernel *Part 1* is given by V=8×64×64×64, a relatively small lattice size for state-of-the-art lattice QCD simulations. In Figure 21 the speed-up relative to the performance on one Skylake node of SuperMUC-NG is shown. The test case, which consists of 1000 iterations of a conjugate gradient solver, takes 186 secs on SuperMUC-NG using a Hybrid parallelisation consisting of 8 MPI tasks each with 6 OpenMP threads to utilise all 48 Skylake cores. The performance results can be summarised as:

- **SuperMUC-NG, Skylake:** The obtained data is used as the reference to calculate the speed-up towards a larger number of nodes. The kernel scales almost perfectly on up to 64 nodes, while still solid speed-up is obtained on 512 nodes. Due the scalability of Intel Xeon Skylake, 512 SuperMUC-NG nodes outperform all other architectures, giving a speed-up of a factor 168 relative to the performance on a single node.
- **Irene-KNL:** The runs were performed using one MPI task with 68 OpenMP threads per card. The performance matches the numbers obtained from using Intel Xeon Skylake on up to 32 nodes, while for larger numbers of nodes the performance drops compared to the Skylake case matching the performance obtained on Piz Daint using 128 nodes.

- **Piz Daint, NVIDIA P100:** The runs were performed with one MPI task per node, utilising each NVIDIA P100 per node. While a single P100 node outperforms a single Skylake node by a factor 3.5, it reaches similar performance on 32 nodes.
- **MARCONI100, NVIDIA V100:** MARCONI100 nodes are each equipped with 4 NVIDIA GPGPU V100, which reaches with 4 MPI task on a single node a relative speed-up by a factor 13.2 compare to a single SuperMUC-NG Skylake node. Similar to Piz Daint P100 node scalability is slightly decreasing and stagnating for larger node counts than 8.
- **JUWELS Booster, NVIDIA A100:** The single node performance of JUWELS Booster outperforms all other systems in case 4 NVIDIA A100 GPGPU are used. The A100 performance is in all cases a factor 2× faster than the 4 NVIDIA V100 GPGPU nodes.

Nodes	Irene KNL	Irene SKL	JUWELS	JUWELS Booster	Marconi 100	MareNostrum4	SuperMUC-NG	Piz Daint
1	148.68	219.68	182.49	9.02	14.07	186.40	186.05	53.73
2	79.35	114.22	91.83	6.48	10.36	94.63	94.79	32.38
4	48.07	58.11	46.58	4.50	8.04	47.22	48.71	19.13
8	28.42	32.09	25.37	3.14	5.36	25.86	31.09	12.78
16	17.08	14.35	11.77	3.54	5.54	11.64	12.71	9.20
32	10.56	7.28	5.43	2.35		5.59	7.02	6.35
64	9.01	4.18	2.65			2.65	3.92	6.41
128	5.08		1.39			2.48	3.30	5.95
256			1.38				1.93	5.84
512			0.89				1.11	
in sec	MPI = 1, omp = 68	MPI = 8, omp = 6	MPI = 8, omp = 6	MPI = 4	MPI = 4	MPI = 8, omp = 6	MPI = 8, omp = 6	MPI = 1

Table 124: The table shows the sustained performance of the UEABS QCD Part 1 with volume V=8×64×64×64 using strong scaling in time to solution (in seconds) on the different PRACE Tier-0 machines.

4.10.2 Test Case: Part 2 - V = 32×32×32×96

Within *Part 2* of the UEABS QCD case state-of-the-art lattice QCD kernels are used, which implement the sparse matrix stencil given by Wilson Dirac applications within the conjugate gradient method. The first test case is given by a lattice volume of V=32×32×32×96, a relatively small volume but it fits to the memory of a single Intel Xeon Phi KNC. The size limits the expected strong scaling window roughly to 16 nodes, where most of the machines show some degradation from ideal scaling. On a single node the sparse matrix stencil is bandwidth bound, roughly given by a theoretical ratio of 1:1 for compute to memory requirements. Using a consistent test case since PRACE-4IP enables us to compare different computing hardware of the last decade. This can be used to illustrate the progress of increasing node performance, due to hardware and software developments. We depict the sustained performance achieved on a single node and on 16 nodes in dependence of the theoretical peak memory bandwidth, as provided by the different vendors, in Figure 22. Overall, the performance improvements over the years scales very well with the increase in bandwidth of the different computing devices. Moreover, as shown by the scaling dependence on the peak bandwidth the kernel on a single

node is bandwidth bound while this is changing on multiple nodes, being for some architectures communication bound. Note that the obtained performance results are shown in Table 124.

In the following, we will discuss in detailed some of the key observations of the different HPC computing devices shown in Figure 22

- **Intel Xeon (Haswell, Skylake, Cascade Lake):** The shown performance results of Intel Xeon chips are obtained on SuperMUC-NG with Haswell architecture (4 MPI task with 7 OpenMP threads), SuperMUC-NG, JUWELS Cluster, MareNostrum4, Irene SKL with Skylake architecture (8 MPI task with 6 OpenMP threads) and Frontera at TACC, US with Cascade Lake (8 MPI tasks with 7 OpenMP threads). The Intel Xeon architecture shows very good scalability for the selected test case, while no major difference between architectures without and with AVX512 capability is observed.
- **Intel Xeon Phi (KNC, KNL):** Performance results are obtained on a local Xeon Phi cluster (CaSToRC) equipped with Intel Xeon Phi KNC cards (1 MPI task with 60 OpenMP threads) and on Marconi-KNL and Frioul (CINES test system) equipped with Intel Xeon Phi KNL. The single node performance of KNLs can vary due to the memory configuration, booted here in cache mode, and could be tuned to higher values if a different memory configuration is selected. As shown, scalability is difficult to achieve, already breaking down for 16 nodes in case of KNLs.
- **ARM (Marvell ThunderX2):** The benchmark was performed within PRACE-5IP on the Mont Blanc system Dibona using a Wilson Dirac kernel, optimised for ARM architectures from the software package grid. The performance is below the one obtained on Intel Skylakes, although more theoretical peak bandwidth is provided by Marvell's chip. In general, the results show that there is potential for additional optimisation. Moreover, a European Fujitsu's ARM A64fx is missing and with-it performance results, but the architecture will find its entry with the Portuguese EuroHPC-JU system Deucalion into the European HPC scene.
- **AMD EPYC (Rome):** The depicted results are obtained on HLRS Hawk using the AMD EPYC Rome processors (32 MPI tasks with 8 OpenMP threads with QPiX). The single node performance matches Intel Xeon Phi KNL ones and shows perfect scaling on 16 nodes, outperforming the Intel Xeon chips. Due to the more complex memory structure AMD EPYCs are trickier to tune and can show performance improvements at larger node numbers in strong scaling tests due to cache effects. In general AMD EPYC's performance trends are similar to Intel Xeon's, while showing higher overall performance.
- **NVIDIA GPGPU (K20, K40m, P100, V100, A100):** The results are obtained on a local cluster (CaSToRC) equipped with NVIDIA K20, on SURF's Cartesius with K40m, on Piz Daint and Davide (CINECA GPU test system) with P100**,** on MARCONI100 with V100 (each node is equipped with 4 GPUs each) and JUWELS Booster with A100 (similar to Marconi, each node comes with 4 GPUs) using QUDA (version 0.7–1.0.1). For all generations the performance is depicted on a single GPU, while for the latter two machines, MARCONI100 and JUWELS Booster, also the performance is shown utilising all 4 GPUs. However, in this case the number for theoretical peak bandwidth is not increased within the figure because they would overlap with the Intel Xeon results on 16 nodes. The single node performance numbers show clearly the performance improvements on the architecture but also on the software side. Namely the performance is not only increasing with the increase of the theoretical

memory bandwidth but also with newer versions of QUDA, where version 0.7 was used for K40m and version 1.0.1 for the A100. Moreover, these improvements on the software side also impact scalability, where for the later generation direct communication between GPUs via GDR is enabled, enhancing scalability performance. The single node performance of JUWELS Booster using 4 A100 tops all other architectures with 3840 Gflop/s in double precision, while reaching 15,950 Gflop/s on 16 nodes. Note that reduction in scalability is expected for the relatively small test size on 64 GPUs.

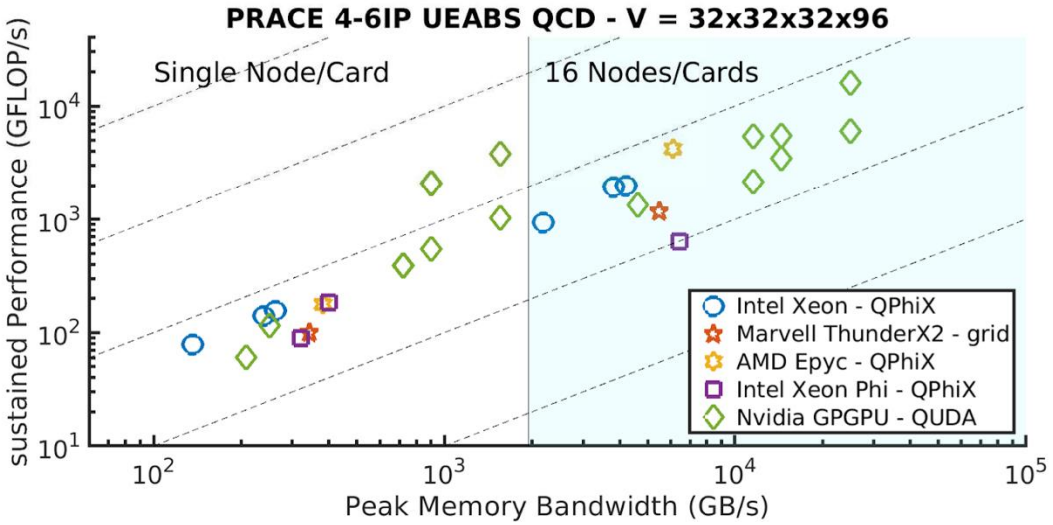

Figure 22: Sustained performance of the UEABS QCD Part 2 with the smaller volume of V=32×32×32×96 on a single node and on 16 node in dependece of the theoretical peak memory bandwith of the corresponding architecture. The figure shows obtained results within PRACE-4IP to PRACE-6IP. Note that all numbers were obtained using double precision.

Nodes	Hawk	Irene SKL	JUWELS	JUWELS Booster	MARCONI100	MareNostrum4	SuperMUC-NG	Piz Daint
1	177,83	134.38	132.26	3836.90	2112.99	142.34	146.92	387.66
2	408.84	240.85	245.60	6261.74	3861.76	263.36	275.61	755.31
4	857.49	460.04	456.23	11875.60	4628.68	480.52	515.15	1400.06
8	1980.97	754.66	864.96	17751.30	6570.62	895.28	951.26	1654.21
16	4200.71	1366.21	1700.95	15947.40	3475.38	1632.87	1820.73	2145.69
32	11702.00	2603.90	3199.98	18345.90	4737.90	2923.7	3517.53	2923.98
64		4122.76	5167.48		1109.96	4118.70	5501.95	2332.71
128		4703.46	7973.90			4050.41	7217.27	
256			3130.42					
512			3421.25					
in Gflop/s	MPI = 32 omp = 8	MPI = 8 omp = 6	MPI = 8 omp = 6	4 GPU per node	4 GPU per node	MPI = 8 omp = 6	MPI = 8 omp = 6	1 GPU

Table 125: Sustained performance of the UEABS QCD Part 2 test size V=96×32×32×32 using strong scaling on the current PRACE Tier-0 machines. The number obtained are collected using double precision kernels.

Note that the scalability of the QPhiX kernel is hard limited by the local volume per MPI task. This is reached in case of 32 Nodes on Hawk, thus limit the scaling towards larger number of nodes in the chosen parallelisation.

4.10.3 Test Case: Part 2 - V = 64×64×64×128

The second volume of the Part 2 of the UEABS QCD is a factor 10.7 larger than the smaller set and extends the strong scaling capability to up to 500 nodes. Moreover, it represents currently common QCD partition sizes. Here, we will show results achieved on the newer PRACE Tier-0 systems, namely SuperMUC-NG, Hawk, MARCONI100 and JUWELS Booster, all equipped with different hardware. As depicted in Figure 23 the A100 nodes of JUWELS Booster outperforms the CPU machines, already reaching 61 Tflop/s in double precision on 32 Nodes with 4 GPUs. Here, GDR via the QUDA option '*export QUDA_ENABLE_GDR=1*' was enabled to utilise direct GPU to GPU communication. This improves scalability drastically, namely finding improvements of 1.8 on 32 nodes. The performance of Marconi's V100 is roughly the half of JUWELS Boosters A100 and strong scaling breaks down after reaching 16 nodes with 64 GPUs. Here, enabling direct GPU communication does not significantly improve the scalability. The CPU machines Hawk and SuperMUC-NG are showing excellent scalability, which is mildly stagnating from 128 nodes.

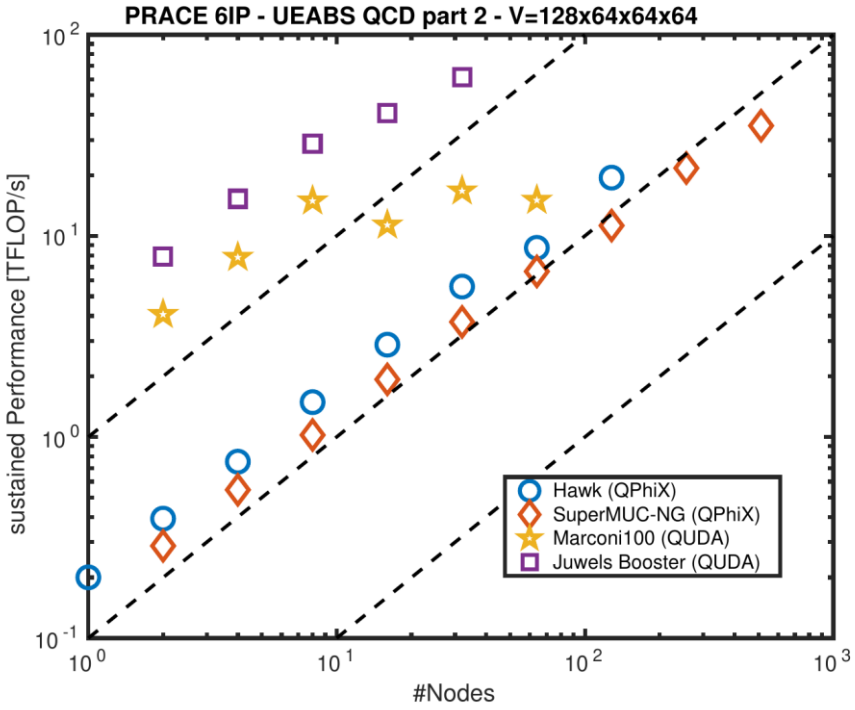

Figure 23: Strong scaling of QCD Part 2 Test Case 2: The sustained performance on the newer PRACE Tier-0 machines is shown for Test Case 2 with volume 128×64×64×64 is shown in dependence on the number of nodes. In all cases the double precision benchmark kernels were used.

Nodes	Hawk	Irene SKL	JUWELS	JUWELS Booster	MARCONI100	MareNostrum4	SuperMUC-NG	Piz Daint
1	200.45	141.31	134.97			144.32		
2	392.57	267.28	263.64	7904.11	4070.01	280.68	287.38	
4	753.75	503.04	496.47	15265.00	7819.70	514.96	545.89	
8	1489.70	922.19	954.66	28671.60	14923.80	930.95	1022.88	2694.00
16	2876.30	1607.92	1787.43	40877.70	11319.50	1778.23	1932.51	5731.56
32	5596.72	3088.02	3289.02	61469.80	16737.10	2635.74	3732.18	7779.29
64	8717.53	4787.89	5952.80		15038.20	5264.16	6652.35	10607.20
128	19467.90	5750.35	10315.30			7998.56	11247.60	13560.50
256		15370.90	18177.20				21725.90	
512			26972.60				35291.50	
In Gflop/s	MPI = 32 omp = 8	MPI = 8 omp = 6	MPI = 8 omp = 6	4 GPU per node	4 GPU per node	MPI = 8 omp = 6	MPI = 8 omp = 6	1 GPU

Table 126: Strong scaling of QCD Part 2 using the larger test size of V=128×64×64×64. The quoted numbers are sustained performance in Gflop/s using double precision.

4.10.4 Comments on Future Developments

New and future architectures will not be supported by the QPhiX kernel, such as the ARM SVE instruction sets, due to that support beyond PRACE-6IP will need to be shifted towards QUDA. Here, within the DoE exascale project and thanks to NVIDIA's support the package QUDA is currently under optimisation to support up-coming hardware, such as deployed within US exascale machines and Europe's EuroHPC-JU machines. Due to that the QPhiX kernel of Part 2 should be replaced in a future release by the *generic* QUDA kernel (still under development), which will guarantee a continuation of the performance metric used within QUDA since PRACE-4IP.

The current set of test cases is missing a large-scale set, which has the potential to scale to a full PRACE Tier-0 machine. In light of so-called master-field simulations such a large set might be relevant in the near future but could be easily added via a weak-scaling test case or a large set such as V=64×64×64×128.

Furthermore, the computing intensive workload of QCD applications are shifted from purely large-scale sparse matrix stencil operation towards more dense and smaller sparse matrix stencil operation, needed in multigrid approaches. Nowadays for all common lattice operators, multigrid methods are known and outperform traditional methods by more than an order of magnitude (e.g. [16]). Moreover, the needed more dense matrix applications also shift the technical limits of the benchmark kernels from bandwidth towards latency bound. To cope with this development future extension of the QCD part of UEABS should be extended to include coarse matrix vector stencils, which are available within DDalphaAMG [17] and QUDA [18] Note that a simple access to these kernels can be provided by the community python API *lyncs* which is under development under the PRACE-6IP WP8 project LyNcs [19].

4.10.5 Conclusion

In all three test cases JUWELS Booster is showing the best overall performance. Compared to older NVIDIA hardware, scalability is widely improved on the new generation of NVIDIA A100 due to direct communications between GPUs. The CPU machines show for most of the cases excellent scalability; however, they cannot reach the overall performance provided by an NVIDIA A100. For a final conclusion the energy consumption of the different architectures would be an important metric, namely to estimate the energy consumption of 32 nodes each with 4 NVIDIA A100 compared to 512 Skylake nodes. These measurements are not available on the machine where the QCD benchmarks within PRACE-6IP are conducted. To conclude, the results of the UEABS QCD part confirm that QCD workloads will perform also efficiently on the up-coming EuroHPC-JU petascale and pre-exascale systems, which will be mostly equipped with NVIDIA A100, such as CINECA's Leonardo or IT4I's Karolina.

4.11 Quantum ESPRESSO

The PWscf module of Quantum ESPRESSO has been benchmarked on several European systems by measuring the wall time (in seconds) of a single iteration of an electronic structure optimisation (i.e. "time-to-solution") as a function of computer resources, i.e. testing strong parallel scaling. We note that weak scaling experiments cannot be performed with this application. The version used varied from 6.5 to 6.8, depending on the installation available on the benchmark platform, but we do not expect this to influence the results in so far as software updates between minor versions of Quantum ESPRESSO involve bug fixes or added functionality, rather than performance enhancements.

4.11.1 MARCONI100

In this section, we analyse the GPU performance of the application with the medium test case on MARCONI100 system and compare with the non-accelerated version on the same system. To reduce the memory per task for the CPU tests, we used 8 OpenMP threads per task and, for the CPU-only runs, 8 tasks per node. For the GPU tests we used 1 MPI task per GPU, and therefore 4 tasks per node. The time-to-solution data are given in Table 127 and Table 128, and are plotted in Figure 24. As can be seen from the data, the GPU version of Quantum ESPRESSO shows a high performance for small numbers of nodes compared to the CPU-only tests, but it does not scale. The highest performance in fact can be obtained at with large numbers of CPU-only nodes which exhibit higher parallel efficiencies.

Nodes	Time-to-solution (s)	Parallel Efficiency (%)
4	296	100
8	251	59
16	247	30
32	283	13

Table 127: Performance and parallel efficiency of Quantum ESPRESSO for the medium test case on MARCONI100 GPU nodes

Nodes	Time-to-solution (s)	Parallel Efficiency (%)
10	606	100
20	305	99
30	209	97
40	161	94

Table 128: Performance and parallel efficiency of Quantum ESPRESSO for the medium test case on MARCONI100 using only CPUs

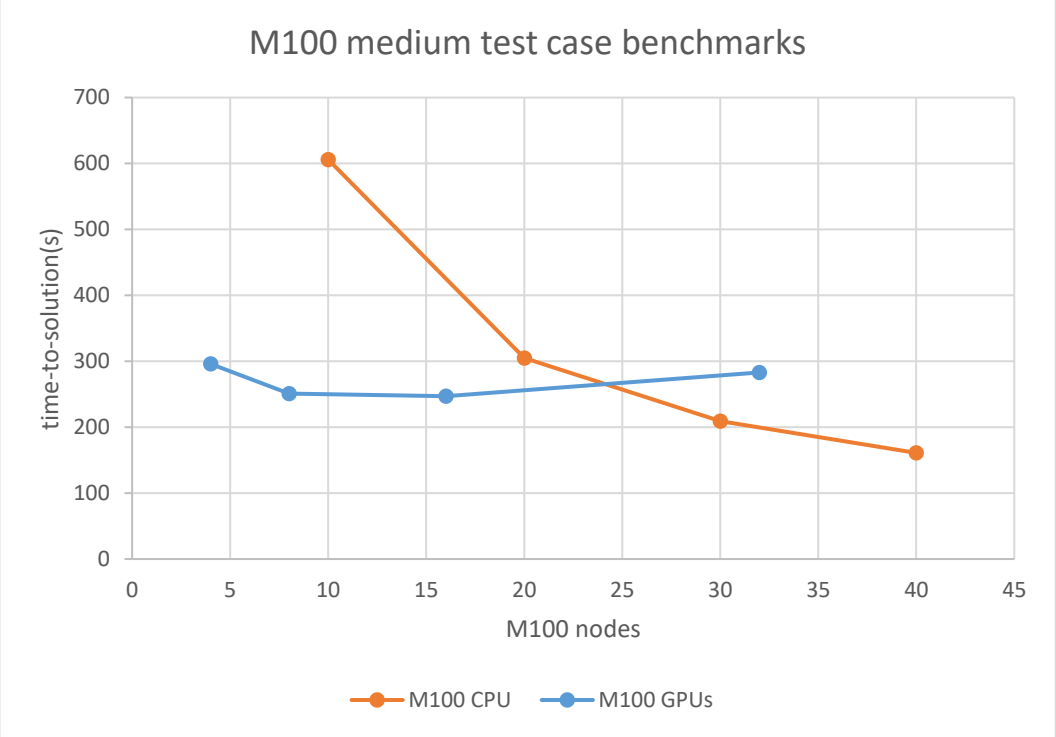

Figure 24: Time-to-solution as a function of MARCONI100 nodes for the medium test case of Quantum ESPRESSO. We show data for both the CPU and GPU versions of the application.

4.11.2 Benchmarks for MareNostrum4, JUWELS, and SuperMUC-NG

In the following subsection we compare results for the CPU partitions of three major supercomputers in Europe: MareNostrum4, JUWELS, and SuperMUC-NG. For each benchmark we measure the time-to-solution for both the medium test case and the large test case. We note that in some cases it was necessary to use large-memory partitions (often called "fat nodes") to run the large test case results. For each run the hybrid mode of Quantum ESPRESSO was run, using 12 MPI tasks per node and 4 OpenMP threads/task.

The performance data for the benchmarks are given in Table 129 and Table 130 while Figure 24 and Figure 25 show the corresponding plots. We can observe for both test cases that the data show very similar trends but with MareNostrum4 giving slightly higher performances overall.

It is not clear why the MareNostrum4 performances are higher than those of JUWELS and SuperMUC-NG, particularly as the latter two machines have processors which have clock frequencies higher than those of the Barcelona supercomputer. On the other hand, at high node counts, the results are essentially identical.

	Medium Test Case Time-to-solution (s)		
#nodes	MareNostrum4	JUWELS	SuperMUC-NG
10	482	460	547
20	153	263	196
30	122	197	213
40	107	157	169
50	125	143	147
60		134	131
70		133	108

Table 129: Comparison of the performance of Quantum ESPRESSO for the medium test case on MareNostrum4, JUWELS, SuperMUC-NG.

	Large Test Case Time-to-solution (s)		
#nodes	MareNostrum4	JUWELS	SuperMUC-NG
10	716	819	1021
20	343	470	572
30	288	375	449
40	245	319	339
50	225	303	330
60	237	309	324

Table 130: Comparison of the performances of Quantum ESPRESSO for the large test case for MareNostrum4, JUWELS and SuperMUC-NG

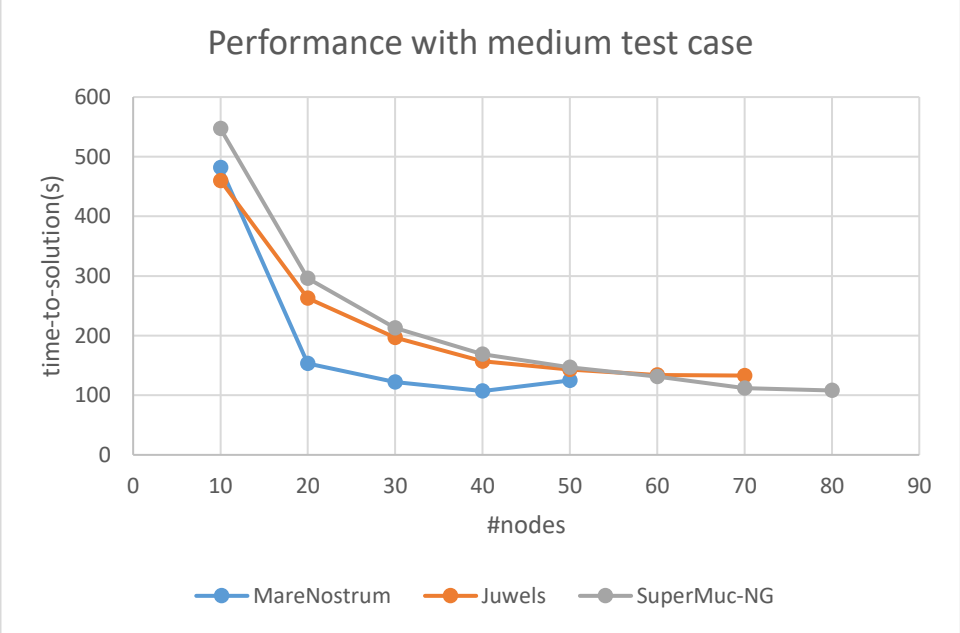

Figure 25: Performance with the medium test case on MareNostrum4, JUWELS and SuperMUC-NG

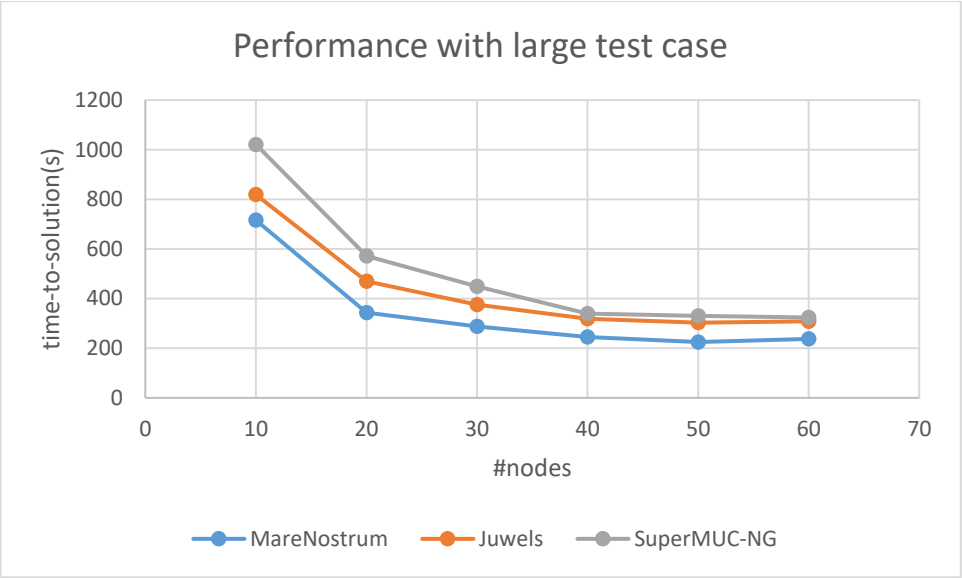

Figure 26: Performance with the large test case for MareNostrum4, JUWELS and SuperMUC-NG

4.11.3 Energy Efficiencies

It was not possible to perform energy measurements of the calculations with the architectures tested with Quantum ESPRESSO.

4.12 SPECFEM3D_GLOBE

The SPECFEM3D_GLOBE benchmarks have been performed on systems with different processors: AMD EPYC, Intel Knights Landing, Intel Skylake, Intel Broadwell and IBM Power9, some of them have different graphics processing units (GPUs): NVIDIA V100, P100 and A100. Distributed parallelism (MPI) was used on all systems. On the accelerated systems, GPU parallelisation was used at the expense of the OpenMP parallelisation model, which was not used because no performance was obtained by coupling these two parallelisation paradigms (OpenMP and GPU). Hybrid parallelisation (MPI and OpenMP) was exploited on platforms with x86 processors only.

4.12.1 System Software Environment

SPECFEM3D_GLOBE requires Fortran and C compilers, MPI and in case of GPU support a CUDA SDK installation. We used the development version of SPECFEM3D Globe 7.0.1 of October 31th, 2015 (Git commit #b1d6ba9). The MPI compiler and library used depends on the availability on each system, the choice is sometimes restricted due to instabilities in the simulation with certain MPI compiler and library combinations.

The software stack used on the machines is summarised in Table 131.

PRACE Tier-0		Fortran compiler	MPI flavour	CUDA
x86 platforms	Hawk EPYC™ 7742	GCC 9	HPE-MPI 2.23	N/A
	Joliot-Curie EPYC™ 7H12	Intel 20	Open MPI 4.0.5	N/A
	Joliot-Curie KNL 7250	Intel oneAPI 21.3.0	Intel MPI 2019.5.281	N/A
	Joliot-Curie Skylake 8168	Intel 20	Open MPI 4.0.5	N/A
	JUWELS Skylake 8168	Intel 19	Intel MPI 2019.7.217	N/A
	MareNostrum4 Skylake 8160	Intel 20	Intel MPI 2018.4	N/A
	SuperMUC-NG Skylake 8174	Intel 17.4	Intel MPI 2017.4	N/A
	Vega 2 × EPYC™ 7H12	GCC 9	Open MPI 4.0.5	N/A
GPU platforms	JUWELS Booster 2 × AMD EPYC™ 7402 4 × NVIDIA A100	Intel 21	ParaStation MPI 5.4.10	11.3
	MARCONI100 IBM POWER9 + 4 × NVIDIA V100	IBM XL	Spectrum MPI 10.3.1	10.1
	Piz Daint Broadwell E5-2695 v4 + NVIDIA P100	CRAY ftn 2.7.3	cray-mpich 7.7.16	11.0
	Vega 2 × EPYC™ 7H12 + 4 × NVIDIA A100	GCC 9	Open MPI 4.0.5	10.1

Table 131: Software environment used in SPECFEM3D_GLOBE Benchmarks

4.12.2 Results

4.12.2.1 Validation Test Case

All simulations were performed on one node, using 24 MPI tasks. Table 132 shows the results of the runtime for the mesher and solver part of the Geophysics code.

PRACE Tier-0		Mesher (s)	Solver (s)
x86 platforms	Hawk EPYC™ 7742	74	6127
	Joliot-Curie EPYC™ 7H12	52	6437
	Joliot-Curie KNL 7250	352	7783
	Joliot-Curie Skylake 8168	72	3968
	JUWELS Skylake 8168	54	1969
	MareNostrum4 Skylake 8160	56	3224
	SuperMUC-NG Skylake 8174	63	2666
GPU platforms	JUWELS Booster 2 × AMD EPYC™ 7402 4 × NVIDIA A100	40	234
	MARCONI100 IBM POWER9 + 4 × NVIDIA V100	192	176
	Piz Daint Broadwell E5-2695 v4 + NVIDIA P100	73	451
	HPC - Vega 2 × EPYC™ 7H12 + 4 × NVIDIA A100	73	175

Table 132: SPECFEM3D_GLOBE Validation Test Case

The part of the code that deals with mesh creation does not include a GPU implementation, and this is clearly seen in the table above. Thus, the mesh creation times are relatively close whether on a GPU or x86 platform. The results of this first test case highlight the spectacular performance of the code on GPU platforms, which are of the order of a hundred seconds for the solver part of the calculation, whereas they are of the order of thousands of seconds for x86 platforms.

These results make it difficult to really compare the parallelisation performances since on x86 platforms only MPI parallelisation was used while on GPU platforms both (MPI+GPU) were used. This is why the following results present this hybrid parallelisation.

4.12.2.2 Test Case A

All simulations have been performed on 24 nodes, using 96 MPI tasks. Table 133 presents the run times of the best performing configuration of OpenMP threads with this particular node count on each system.

PRACE Tier-0		Mesher (min)	Solver (s)	Threads OpenMP
x86 platforms	Hawk EPYC™ 7742	12	781	32
	Joliot-Curie EPYC™ 7H12	7	576	32
	Joliot-Curie KNL 7250	54	1662	16
	Joliot-Curie Skylake 8168	7	682	12
	JUWELS Skylake 8168	6	637	12
	MareNostrum4 Skylake 8160	9	751	12
	SuperMUC-NG Skylake 8174	9	671	12
GPU platforms	JUWELS Booster 2 × AMD EPYC™ 7402 4 × NVIDIA A100	7	25	N/A
	MARCONI100 IBM POWER9 + 4 × NVIDIA V100	28	44	N/A
	Piz Daint Broadwell E5-2695 v4 + NVIDIA P100	12	212	N/A
	Vega 2 × EPYC™ 7H12 + 4 × NVIDIA A100	13	49	N/A

Table 133: SPECFEM3D_GLOBE Test Case A

The mesh creation times are still very close whether on a GPU or x86 platform with again a mesh time at least quadrupled on the KNL compared to the other architecture and at least doubled on the Power9 of MARCONI100. Concerning the x86-only platforms, we still find that the AMD Rome partition of Joliot-Curie performs best on the Validation Test Case and Test Case A.

4.12.2.3 Test Case B

All simulations have been performed on 384 nodes, using 1536 MPI tasks. Table 134 presents the run times of the best performing configuration of OpenMP threads with this particular node count on each system.

PRACE Tier-0		Mesher (min)	Solver (s)	Threads OpenMP
x86 platforms	Hawk EPYC™ 7742	1	151	32
	Joliot-Curie EPYC™ 7H12	1	137	32
	Joliot-Curie Skylake 8168	1	101	12
	JUWELS Skylake 8168	1	128	12
	MareNostrum4 Skylake 8160	1	162	12
	SuperMUC-NG Skylake 8174	2.5	147	12

Table 134: SPECFEM3D_GLOBE Test Case B

Simulations on GPU platforms could not be performed because a 384-nodes submission is not possible on these platforms. Furthermore, this test case will not take full advantage of the CPU-GPU resources. Indeed, this test case makes sense on x86 CPU systems but on GPU systems, the solver part will compute too fast, and only the communication performance will count.

4.12.3 Performance Comparison

In this section, we compare the performance of the PRACE Tier-0 systems using scalability and strong scalability curves by also reporting the speed-up and parallel efficiency factors. Weak scalability has not been used because the SPECFEM3D_GLOBE software has numerical and software instabilities on some mesh configurations and the number of processors used, making it impossible to produce weak scalability curves.

4.12.3.1 Scalability

The results presented in this section cover the execution times of the mesher and the solver by gradually increasing the size of the meshes (which is reflected in the value of the variable NEX_XI, the number of spectral elements along one side of a piece of the cubic sphere). Figure 27 and Figure 28 show a comparison of the performance of the Joliot-Curie system partition with two AMD Rome processors, JUWELS Cluster module with its 24 core Intel Xeon Skylake processors and Vega which has two AMD Rome processors and four NVIDIA A100 graphics cards per node. The simulation is run on 24 compute nodes.

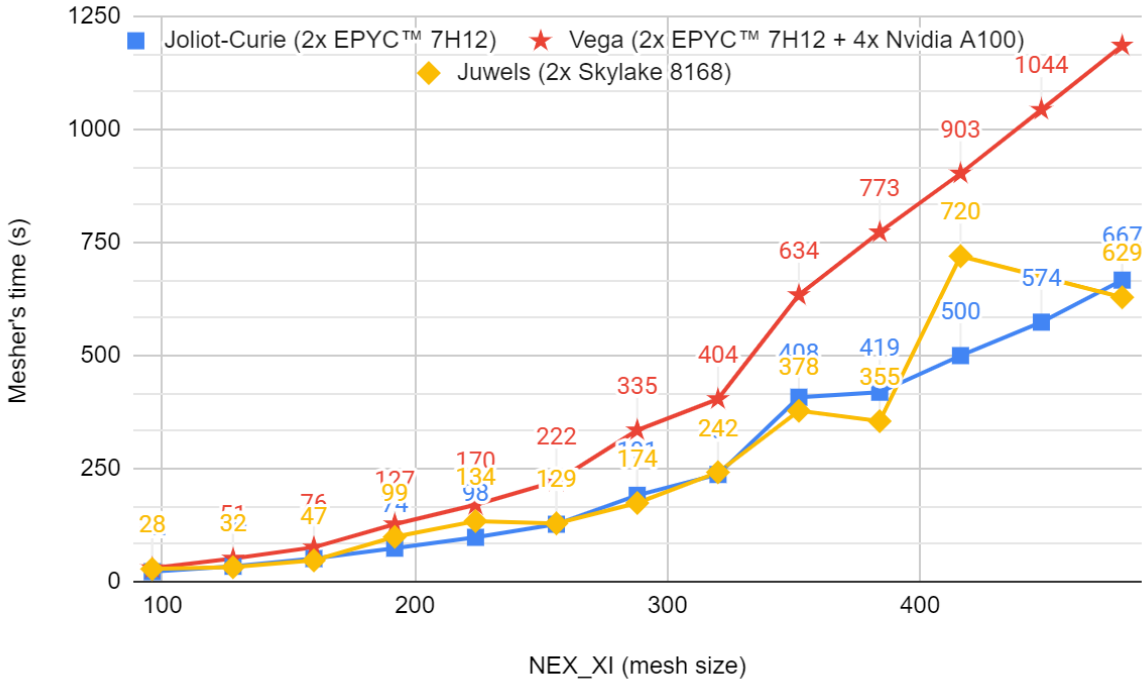

Figure 27: Mesher scaling on 24 computes nodes by increasing the NEX_XI

The Vega and Joliot-Curie systems have the same processors but the Joliot-Curie configuration allows more performance to be achieved on the CPUs only. The mesh part of the SPECFEM3D_GLOBE code does not benefit from GPU acceleration. The results for the Joliot-Curie system and the JUWELS Cluster module are of the same order for small mesh sizes and for larger mesh sizes the JUWELS system configuration performs better overall.

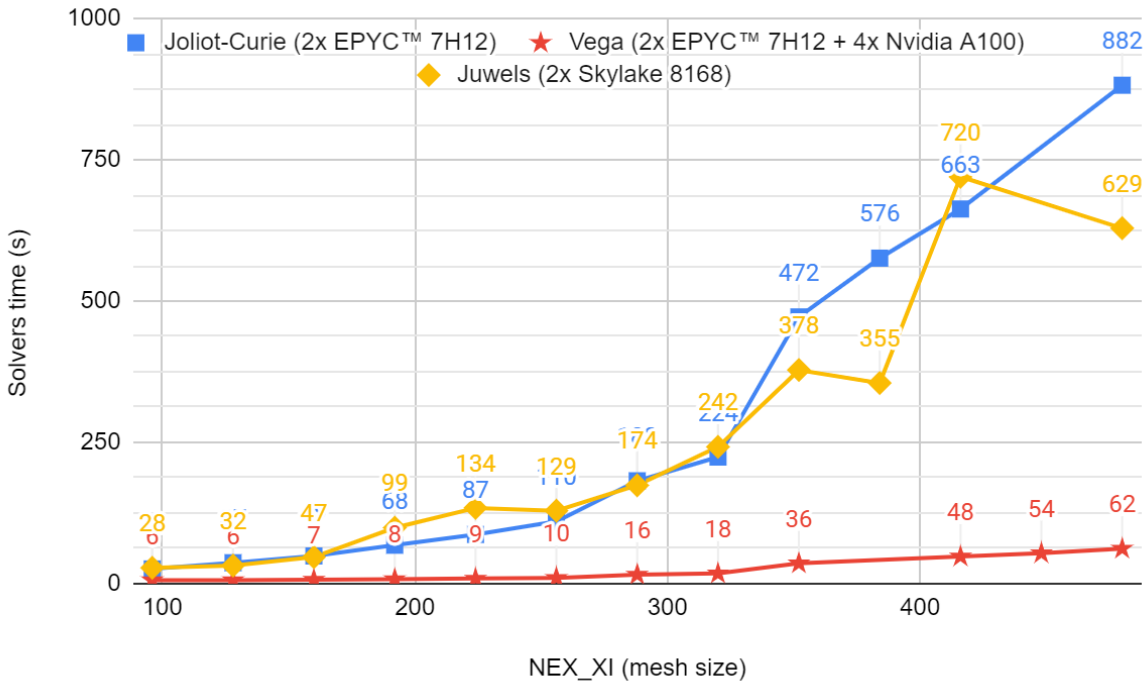

Figure 28: Solver scaling on 24 computes nodes by increasing the NEX_XI

Figure 28 highlights the efficiency of the GPU acceleration of the SPECFEM3D_GLOBE solver. While the execution times of the Joliot-Curie and JUWELS systems explode when the mesh size increases, the execution times on Vega do not exceed several tens of seconds, i.e. about ten times less execution time for fine meshes. The limiting factor of these curves is the memory required for meshes with a NEX_XI higher than 480, the memory required to run the test cases on the systems becomes insufficient.

It is surprising to see that the JUWELS Cluster module performs better than the Joliot-Curie AMD Rome partition, indeed JUWELS Cluster module has 48 cores per node against the 64 per node of Joliot-Curie; this can be partly explained by the AVX512 vectorisation present on JUWELS against the AVX2 vectorisation of Joliot-Curie AMD Rome.

4.12.3.2 Strong Scaling

In this section, we present the strong scaling results only for the solver part of the SPECFEM3D_GLOBE code.

4.12.3.2.1 Small Benchmark Run to Test More Complex Earth

On the "small benchmark to test a more complex land" native test case we tested a large scale-up, starting with a problem size of 600 MB per process on 1 node (original test design) and depopulating it on 2, 4 and 8 nodes. Only six systems were compared for this test case, the systems compared were chosen to highlight the differences in efficiency depending on the architecture used to run the code (results for more systems will be presented in the next section for Test Case A which is closer to a real simulation). The following tables present the solver execution times, speed-up and parallel efficiency of different systems. The speed-up and parallel efficiency are reported with reference to the run time on a single node.

Nodes	Time (s)	Speed-up	Parallel efficiency (%)
1	6437	1.00	100
2	4362	1.48	73
4	1997	3.22	80
8	2014	3.20	40

Table 135: SPECFEM3D_GLOBE, strong scaling Validation Test Case on Joliot-Curie Rome

Nodes	Time (s)	Speed-up	Parallel efficiency (%)
1	1969	1.00	100
2	1099	1.79	89
4	1949	1.01	25
8	1130	1.74	21

Table 136: SPECFEM3D_GLOBE, strong scaling Validation Test Case on JUWELS Cluster module

Nodes	Time (s)	Speed-up	Parallel efficiency (%)
1	3968	1.00	100
2	3105	1.27	64
4	3332	1.19	29
8	2853	1.39	17

Table 137: SPECFEM3D_GLOBE, strong scaling Validation Test Case on Joliot-Curie Skylake

Nodes	Time (s)	Speed-up	Parallel efficiency (%)
1	175	1.00	100
2	97	1.80	90
4	68	2.57	64
8	31	5.65	70

Table 138: SPECFEM3D_GLOBE, strong scaling Validation Test Case on Vega

Nodes	Time (s)	Speed-up	Parallel efficiency (%)
1	176	1.00	100
2	103	1.71	85
4	75	2.35	58
8	39	4.51	56

Table 139: SPECFEM3D_GLOBE, strong scaling Validation Test Case on MARCONI100

Nodes	Time (s)	Speed-up	Parallel efficiency (%)
1	234	1.00	100
2	118	1.98	99
4	82	2.85	71
8	44	5.32	66

Table 140: SPECFEM3D_GLOBE, strong scaling Validation Test Case on JUWELS Booster

On all machines, the runtime decreases when the number of nodes increases. For this test case, we observe in particular that on GPU platforms the parallel efficiency remains good up to 8 nodes (56% on MARCONI100 which is the worst result) while on x86 platforms only the parallel efficiency results are bad beyond 2 nodes (lower than 30%) except on the AMD Rome system of Joliot-Curie which remains above 70% up to 4 nodes.

In order to have all the tools to compare the systems, we have drawn the strong scalability curves (Figure 29), the speed-up (Figure 30) and parallel efficiency results (Figure 31).

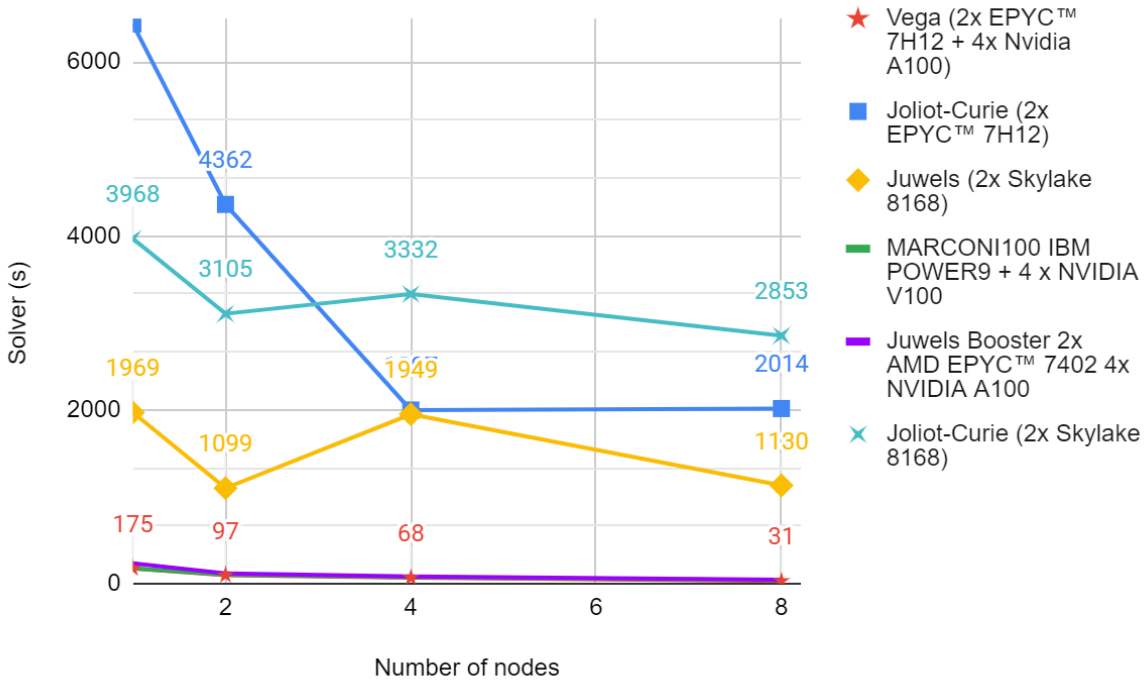

Figure 29: SPECFEM3D_GLOBE, strong scaling on Validation Test Case: small benchmark run to test on more complex earth

The results in the three tables above (Table 135, Table 136 and Table 138) suggest that the Joliot-Curie (2 × EPYC™ 7H12) system scales best among the CPU-only systems. Although the parallel efficiency of Joliot-Curie (2 × EPYC™ 7H12) is relatively good up to 8 nodes; by analysing Figure 29, we understand that this system is in fact the least suitable for this test case. Indeed, the reference time on the Joliot-Curie AMD Rome system is 3 times higher than on JUWELS Cluster module

Concerning the two benchmarked Intel Xeon Skylake systems (JUWELS and Joliot-Curie), JUWELS exceeds the performance of Joliot-Curie even though they have an almost identical hardware configuration (identical memory, CPU, interconnect and nominal clock speed), these differences can only be explained by the use of the Intel MPI library on JUWELS and OpenMPI 4 on Joliot-Curie.

Again, the performance of GPU systems greatly exceeds the performance of x86 systems. The simulation on a Vega node performs 36 times better than on Joliot-Curie (2 × EPYC™ 7H12) and 6 times better compared to the JUWELS Cluster model system.

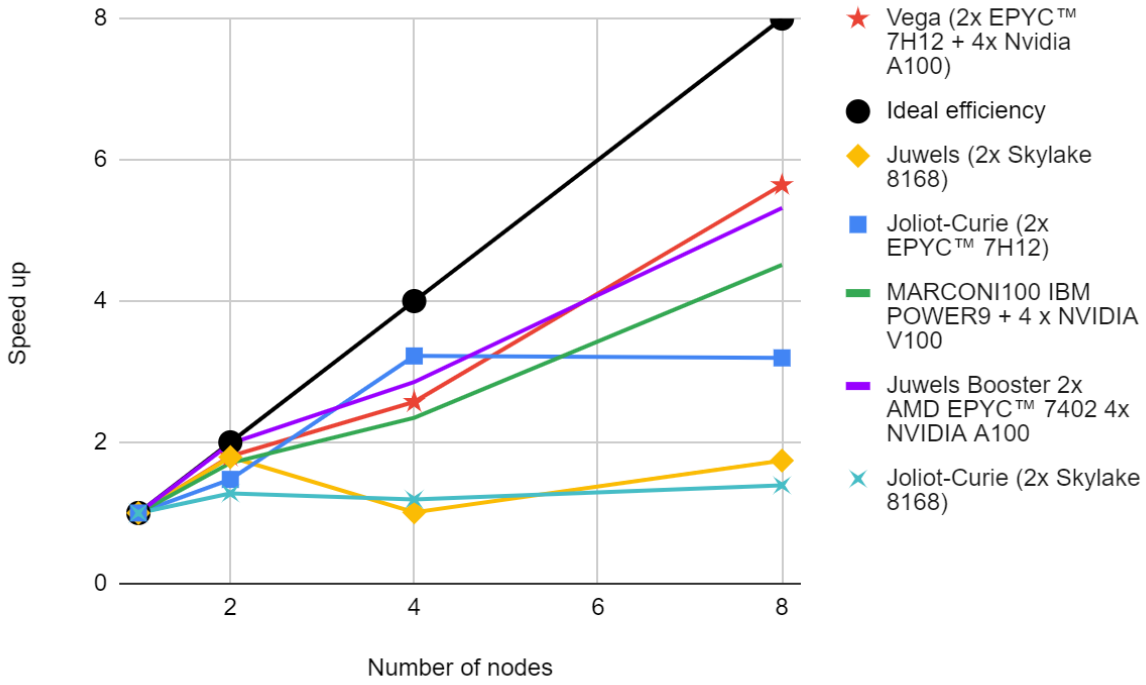

Figure 30: SPECFEM3D_GLOBE, speed-up on Validation Test Case: small benchmark run to test on more complex earth

Although the results obtained on JUWELS are better than those obtained on Joliot-Curie Skylake, the scaling trends are almost identical. The same remark applies to the Vega and JUWELS Booster systems, but this time the Vega configuration achieves the best performance. MARCONI100 with its four GPUs and Power9 processors achieves similar performance to these two other systems with four NVIDIA A100 cards and two AMD Rome processors.

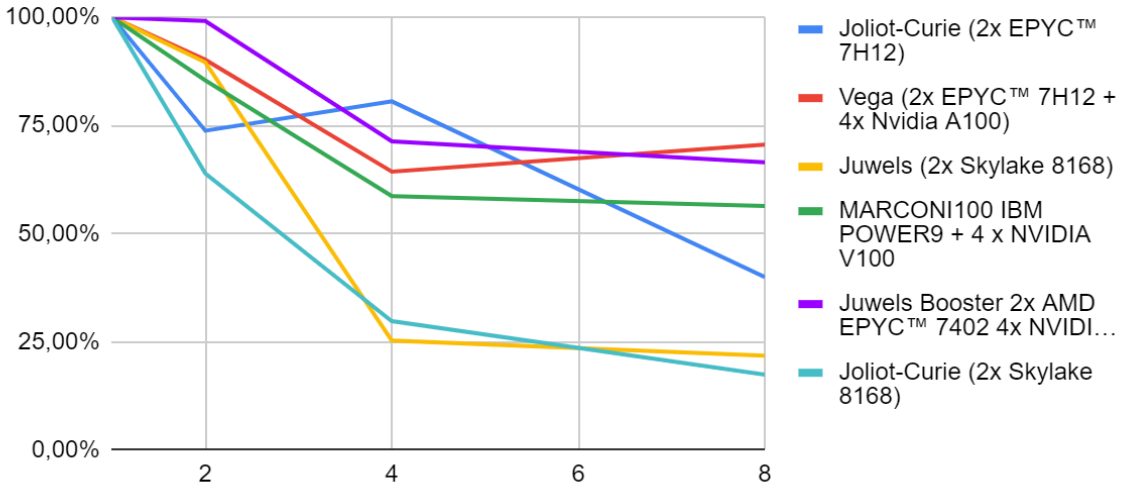

Figure 31: SPECFEM3D_GLOBE, parallel efficiency on Validation Test Case: small benchmark run to test on more complex earth

4.12.3.2.2 Test Case A

The tests were performed on 12, 24 and 48 nodes on Test Case A. As described in Section 2.12.2 the SPECFEM3D_GLOBE mesher and solver must be recompiled each time we change the mesh size because the solver uses a static loop size. Not all systems are reported because some allocations were insufficient or the software environment of some systems did not allow for a stable compilation and execution of the software to perform all the tests. The systems compared nevertheless allow us to highlight the efficiency of the code on very different architectures. The following tables present the solver execution times, speed-up and parallel efficiency of different systems. Parallel speed-up and efficiency are reported with reference to execution time on 12 nodes, except for the JUWELS Booster and Piz Daint systems where the simulation could not be run at 12 nodes. Indeed, on some systems the test case cannot be run on fewer nodes as the simulation requires a minimum amount of memory.

Nodes	Time (s)	Speed-up	Parallel efficiency (%)
12	772	1.00	100
24	576	1.34	67
48	572	1.35	33

Table 141: SPECFEM3D_GLOBE, strong scaling Test Case A on Joliot-Curie Rome

Nodes	Time (s)	Speed-up	Parallel efficiency (%)
12	737	1.00	100
24	758	0.97	48
48	732	0.63	16

Table 142: SPECFEM3D_GLOBE, strong scaling Test Case A on Hawk

Nodes	Time (s)	Speed-up	Parallel efficiency (%)
12	1041	1.00	100
24	637	1.63	81
48	433	2.4	60

Table 143: SPECFEM3D_GLOBE, strong scaling Test Case A on JUWELS Cluster module

Nodes	Time (s)	Speed-up	Parallel efficiency (%)
12	1083	1.00	100
24	682	1.58	79
48	467	2.32	58

Table 144: SPECFEM3D_GLOBE, strong scaling Test Case A on Joliot-Curie Skylake

Nodes	Time (s)	Speed-up	Parallel efficiency (%)
12	42	1.00	100
24	50	0.84	42
48	19	2.2	55

Table 145: SPECFEM3D_GLOBE, strong scaling Test Case A on Vega

Nodes	Time (s)	Speed-up	Parallel efficiency (%)
12	64	1.00	100
24	43.6	1.46	73
48	N/A	N/A	N/A

Table 146: SPECFEM3D_GLOBE, strong scaling Test Case A on MARCONI100

Nodes	Time (s)	Speed-up	Parallel efficiency (%)
12	N/A	N/A	N/A
24	25	1.00	100
48	43	1.72	86

Table 147: SPECFEM3D_GLOBE, strong scaling Test Case A on JUWELS Booster

Nodes	Time (s)	Speed-up	Parallel efficiency (%)
12	N/A	N/A	N/A
24	212	1.00	100
48	109	1.94	97

Table 148: SPECFEM3D_GLOBE, strong scaling Test Case A on Piz Daint

It is difficult to define a trend for GPU systems as we have results per system, but overall, these GPUs show good parallel efficiency (over 70%) except for Vega which has 42% parallel efficiency on 24 nodes, but this is partly explained by the better performance on the reference time on 12 nodes.

The Skylake systems (JUWELS Cluster module and Joliot-Curie) show good scaling up to 48 nodes with a parallel efficiency of about 60% (on 48 nodes) while the AMD Rome systems (Joliot-Curie and Hawk) show a scaling below 67% from 24 nodes.

In order to have all the tools to compare the systems, we have drawn the strong scalability curves (Figure 32), and the speed-up (Figure 33) and parallel efficiency results (Figure 34).

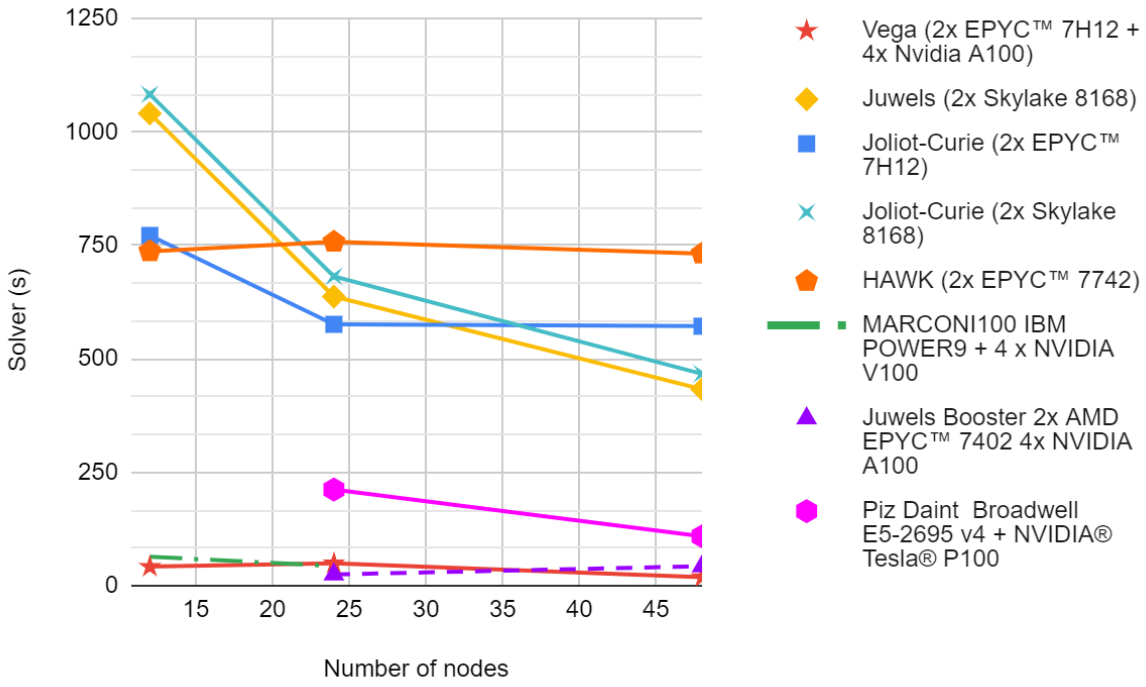

Figure 32: SPECFEM3D_GLOBE, strong scaling on Test Case A

Figure 32 highlights again the performance gain on accelerated systems (runtime is 17 times higher on 12 Vega nodes than on 12 Hawk nodes, 25 times higher than on 12 Joliot-Curie Skylake nodes). Piz Daint contains one GPU per node; the performance obtained on this system compared to other systems with four GPUs per node is consistent (in terms of proportionality). The execution time of the solver remains much lower compared to systems with only x86 processors. The AMD EPYC™ systems perform better than Skylake systems when the problem size is suited to the number of resources, but as soon as the problem becomes too small for the node, performance stalls. The JUWELS and Joliot-Curie Skylake systems achieve similar parallel performance, speed-up and parallel efficiency with again a slight lead for the JUWELS Cluster module system.

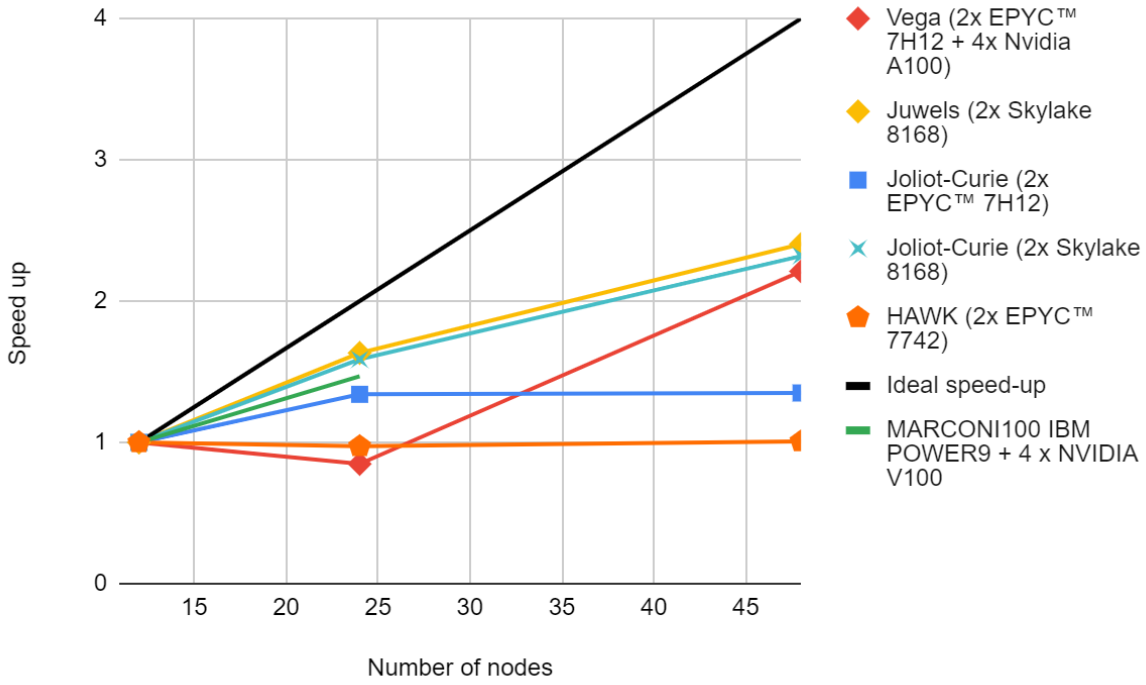

Figure 33: SPECFEM3D_GLOBE, speed-up on Test Case A

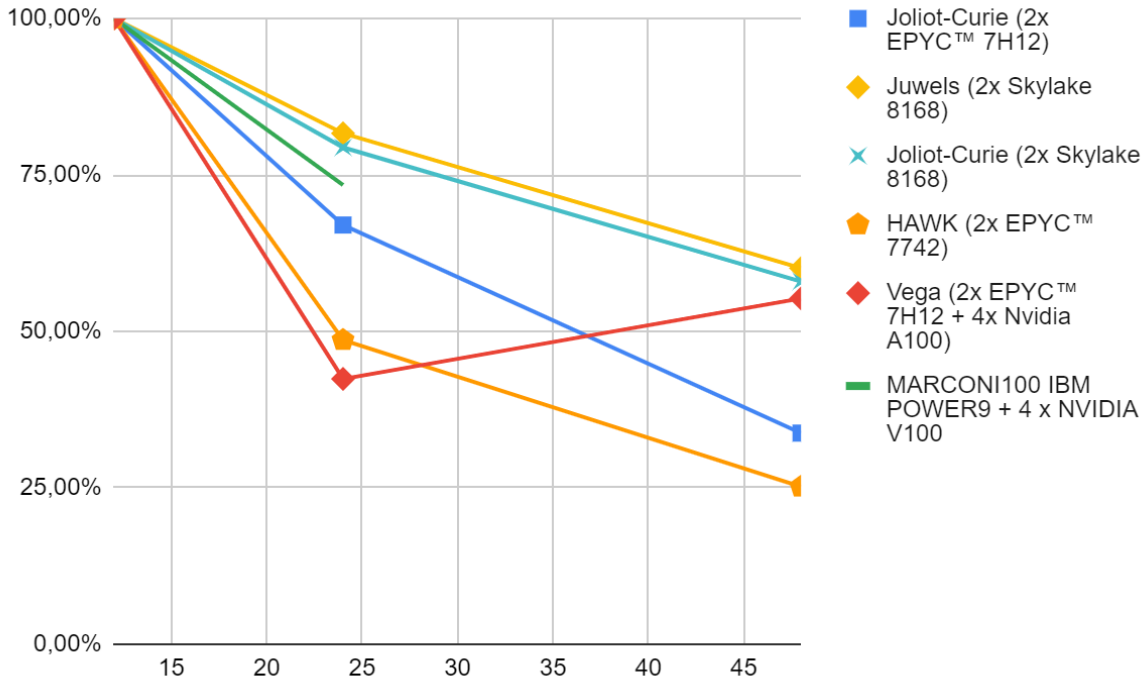

Figure 34: SPECFEM3D_GLOBE, parallel efficiency on Test Case A

4.12.3.3 Energy Consumption Comparison

In this section, we present the energy consumed by the execution of the Validation Test Case and Test Case A of SPECFEM3D_GLOBE on the Tier-0 systems. In each case, the total energy

for the execution of the task on the system is given from the workload accounting logs using the 'sacct' command or similar. This energy measure includes contributions from both node energy and switch energy.

Due to the fact that this measure is considered by many to be inaccurate (although it is the only one present on most systems), some systems do not provide this energy measure. We therefore only present results for Joliot-Curie KNL, Joliot-Curie-Rome, Joliot-Curie Skylake, Piz Daint, SuperMUC-NG, MareNostrum4 and Vega. For Test Case B, we only report the values for MareNostrum4 and Joliot-Curie-Rome, the other systems returning outliers.

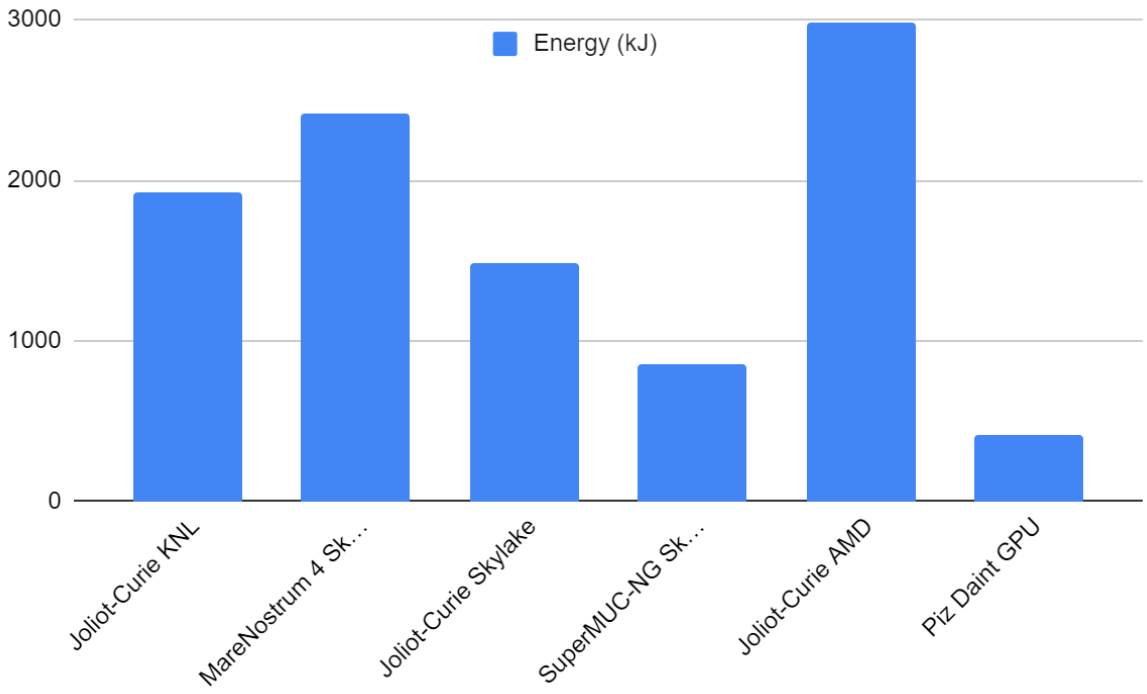

Figure 35: SPECFEM3D_GLOBE, energy consumption for the Validation Test Case

PRACE Tier-0		Solver (s)	Energy (kJ)
x86 platform	Joliot-Curie EPYC™ 7H12	576	2978
	Joliot-Curie KNL 7250	1662	1918
	Joliot-Curie Skylake 8168	682	1479
	MareNostrum4 Skylake 8160	751	2412
	SuperMUC-NG Skylake 8174	671	850
GPU platform	Piz Daint Broadwell E5-2695 v4 + NVIDIA P100	212	419

Table 149: SPECFEM3D_GLOBE, energy consumption and solver time for the Validation Test Case

Figure 35 is not very representative for the consumption of a real SPECFEM3D_GLOBE calculation since this test case was designed to run on 24 cores, and to validate the functioning of the software relatively quickly. Nevertheless, we observe that MareMostrum4 is already more power hungry than the Skylake partition of Joliot-Curie while MareNostrum4 is clocked at 2.1 GHz against 2.7 GHz for Joliot-Curie Skylake. We also notice that the simulation on the AMD Rome partition of Joliot-Curie consumed more energy than on the other platforms while

the calculation took the least time. The high power consumption of the KNL, which is only clocked at 1.4 GHz, is explained by the long simulation time on the architecture.

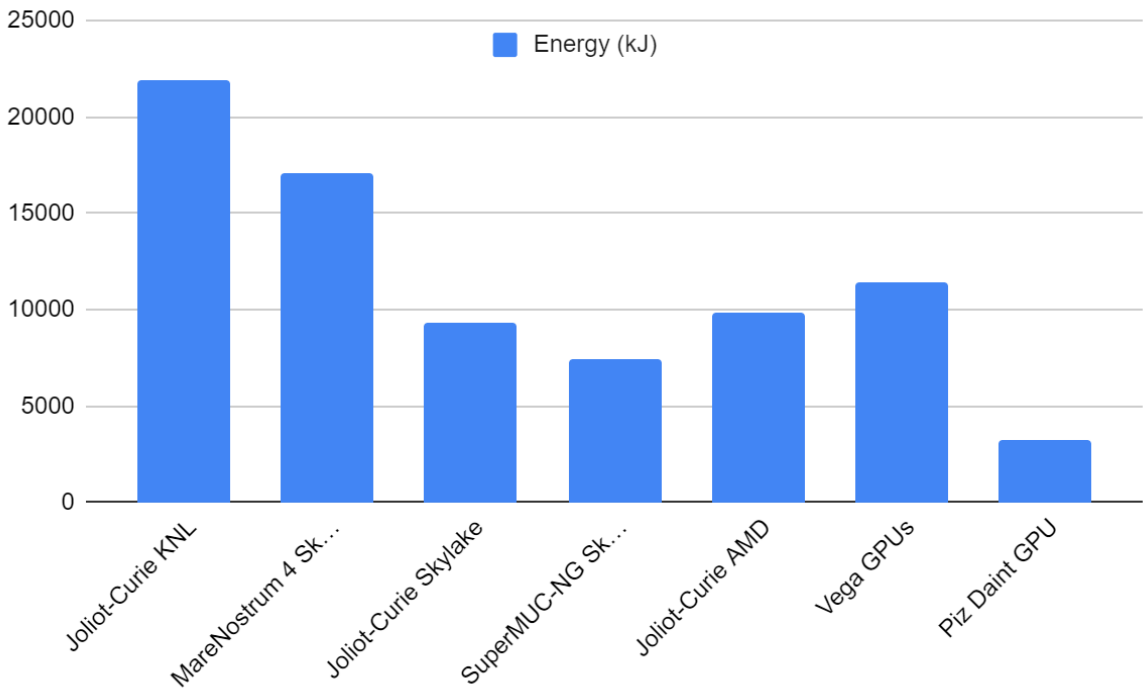

Figure 36: SPECFEM3D_GLOBE, energy consumption for Test Case A

PRACE Tier-0		Solver (s)	Energy (kJ)
x86 platforms	Joliot-Curie EPYC™ 7H12	576	9818
	Joliot-Curie KNL 7250	1662	21869
	Joliot-Curie Skylake 8168	682	9363
	MareNostrum4 Skylake 8160	751	17088
	SuperMUC-NG Skylake 8174	671	7479
GPU platforms	Piz Daint Broadwell E5-2695 v4 + NVIDIA P100	212	3280
	Vega 2 × EPYC™ 7H12 + 4 × NVIDIA A100	49	11396

Table 150: SPECFEM3D_GLOBE, energy consumption and solver time for Test Case A

Figure 36 and Table 150 above confirm part of the trend of the Validation Test Case, namely that MareNostrum4 is even more energy consuming than its Joliot-Curie Skylake counterpart. Joliot-Curie KNL is this time the most energy consuming but is also the simulation that took the longest time (about 3 times longer than on the other x86 platforms). For Test Case A, SuperMUC-NG is still the most energy-efficient x86-only system. Piz Daint with its single GPU still achieves the best result in terms of energy consumption. As for Vega with its 4 GPUs, its energy consumption remains consistent in terms of proportion compared to Piz Daint and its energy consumption is relatively moderate compared to the CPU-only system. This performance can be explained by the fact that the simulation times are much shorter than for CPU-only systems.

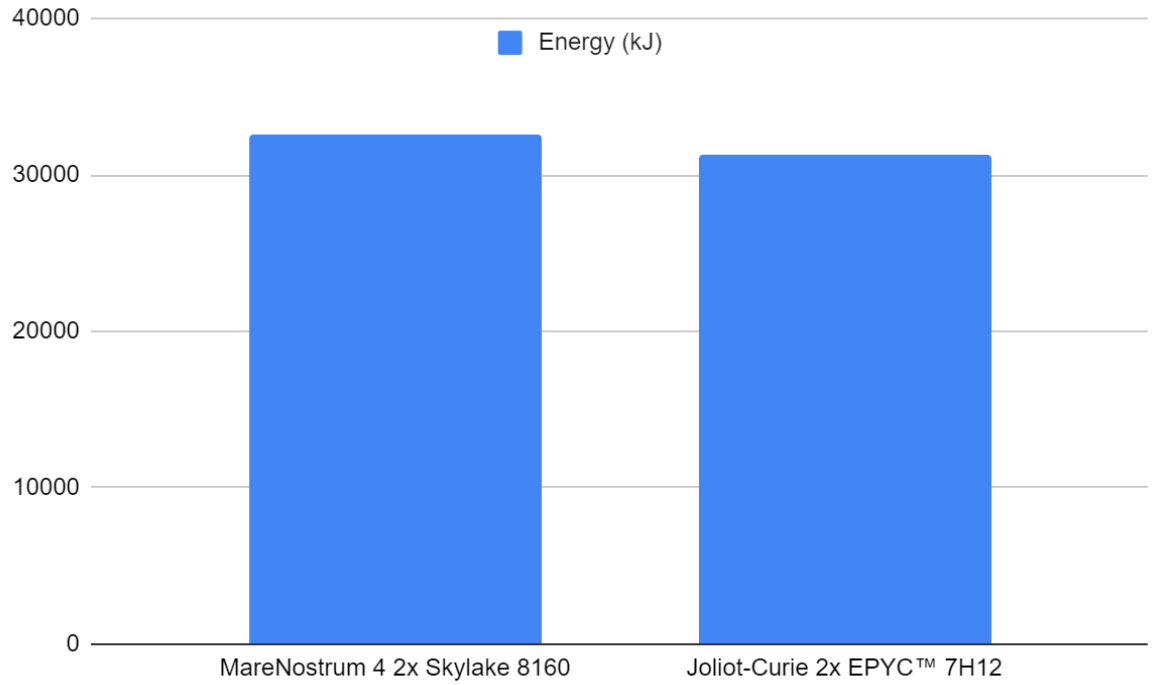

Figure 37: SPECFEM3D_GLOBE, energy consumption for Test Case B

PRACE Tier-0		Solver (s)	Energy (kJ)
x86 platforms	Joliot-Curie EPYC™ 7H12	137	31294
	MareNostrum4 Skylake 8160	162	32520

Table 151: SPECFEM3D_GLOBE, energy consumption and solver time for Test Case B

Figure 35, Figure 36 and Figure 37 show that the energy performance of SPECFEM3D_GLOBE simulations on AMD Rome processors depends on the workload given to the processors, but that when using many AMD Rome nodes, the energy consumption remains in the same order of magnitude as on Skylake architectures.

To conclude, Piz Daint obtains the best energy results of all the systems combined, SuperMUC-NG seems to be the architecture using Skylake processors with the best energy performance; and finally, the AMD system obtains correct performances but depends on the good distribution of the workload which is difficult to find on these systems.

The comparison of energy performance between current Tier-0 architectures is not very meaningful since for the SPECFEM3D_GLOBE test cases we use a constant number of nodes for each test case. AMD EPYC systems have more cores per node and are therefore more power hungry. Systems with graphics cards are supposed to consume more power because in addition to powering the CPUs the GPUs need to be powered, but some systems (Piz Daint) make up for this over-consumption because the execution times are much lower.

Another problem for the energy comparison is that the sacct command is not very reliable, sometimes it returns nonsense values; and finally, some Tier-0s do not disclose the energy information, which makes the exhaustive energy comparison impossible.

4.12.4 Conclusions

The SPECFEM3D_GLOBE code exploits the parallelism of the GPUs very well and the performance of these systems in terms of solution time is far ahead of the performance of CPU-only systems, which has a positive impact on the energy performance of the systems. Systems with four GPUs per node have performances of the same order (a few tens of seconds of runtime against several hundred for x86 platforms only) but which slightly differ (from simple to double for the runtime) depending on the configuration of the simulations.

The four Skylake systems have similar performance given their difference in CPU frequency and interconnect, with JUWELS being slightly faster (see Table 132 and Table 133).

The code fails to perform well on the KNL architecture, with times two to three times longer than on other x86 platforms without GPUs. Several OpenMP threading configurations have been tested without satisfactory results.

The SPECFEM3D_GLOBE code struggles to get performance out of AMD Rome processors, when the problem size is adapted to the number of resources the code achieves good performance, but as soon as the problem becomes too small for the node, the performance stalls. So, there is some work to do to adapt the simulation configuration of SPECFEM3D_GLOBE to these AMD Rome with 128 cores per node.

The MPI parallelisation is a bit disappointing for this code, we quickly drop below 70% parallel efficiency when we increase the amount of computing resources (from 4 to 8 nodes). The code requires a good configuration of the computing resources used (task placement, number of OpenMP threads, compiler and MPI libraries used) and a preliminary study to determine the optimal configuration (number of cores and memory per MPI process) on a given system.

The architectures are still evolving and currently are very heterogeneous; some have 4 GPUs per node, the number of cores per node varies between 12 and 128 cores. Comparing systems when there are so many variables becomes complicated. Anyway, it is still a good code for benchmarking HPC systems, as its code can and does adapt to a wide range of architectures, including CPU-only, GPU-only and even new generation AMD processors. It has a lot of MPI communication and can require a lot of memory depending on the configuration of the test cases.

4.13 TensorFlow

We carried out the TensorFlow benchmark with Test Case A, since it is small enough to train on both CPUs and GPUs. While TensorFlow and other deep learning frameworks are known to be GPU oriented, we still carry out the benchmark on both CPU and GPU systems. This is because in some cases, the training data and/or model can be too large to fit into the GPU memory, hence doing deep learning on CPU becomes a viable option.

The benchmark is carried out on Hawk, SuperMUC-NG, and Lisa (SURF). The command to run all benchmark is:

```
export OMP_NUM_THREADS=<number_of_cores_per_sockets>
 HOROVOD_FUSION_THRESHOLD=134217728 \
 mpirun --np <number_of_mpi_workers> \
 --map-by ppr:1:socket:pe=$OMP_NUM_THREADS \
```

```
 --report-bindings \
 --oversubscribe \
 -x LD_LIBRARY_PATH \
 -x HOROVOD_FUSION_THRESHOLD \
 -x OMP_NUM_THREADS=$OMP_NUM_THREADS \
 python dg_train.py -f output_bw_512.hdf5 --num-camera 3 --arch EfficientNetB4 \
--epochs 5 --batch-size <batch_size>
```

The placeholders <number_of_cores_per_sockets> and <number_of_mpi_workers> should be replaced by the number of CPU cores in a CPU socket and the number of copies of the neural network is trained in parallel. For example, if a simulation is running on 4 nodes, each of which with two CPU sockets, and each CPU has 64 cores, then number_of_cores_per_sockets = 64 and <number_of_mpi_workers> = 8 (4 nodes, 2 MPI workers per node). The <batch_size> parameter is specific to machine learning rather than HPC, but users should choose a proper batch size to make sure that the hardware resources are fully utilised but not overloaded.

This configuration results in the following MPI-OpenMP binding:

```
[BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB][../../../../../../../../../../../../../../../../../../../..]  ← Host #1, socket #1
[../../../../../../../../../../../../../../../../../../../..][BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB]  ← Host #1, socket #2
[BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB][../../../../../../../../../../../../../../../../../../../..]  ← Host #2, socket #1
[../../../../../../../../../../../../../../../../../../../..][BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB]  ← Host #2, socket #2
……
[BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB][../../../../../../../../../../../../../../../../../../../..]  ← Host #n, socket #1
[../../../../../../../../../../../../../../../../../../../..][BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB/BB]  ← Host #n, socket #2
```

In the diagram above, each "[]" indicates a CPU socket, and each core is wrapper with "/". The "BB" inside a "/BB/" indicates that the CPU supports hyperthreading. This mapping shows that an MPI process is pinned to a CPU socket, on which all cores are fully utilised (parallelised by OpenMP).

Working in a data parallel manner, the gradients calculated by each worker have to be communicated periodically so that the model can be updated collectively. In an ideal system without communication overhead, the processors are fully occupied by the computation of gradients and reduction of gradients from other processors, spending no time waiting for the gradients to be transferred. In reality, communication overhead does exist (depends on the type of communications such as point-to-point or global, the network topology, and the data placement therein), so the processors will have to spend time in the synchronization barriers, which subsequently lowers the scaling efficiency. Therefore, by plotting the scaling efficiency as a function of the number of processors (or compute nodes), one can obtain insights into the communication overheads of the underlying system.

The behaviour of a TensorFlow multi-node training can be understood from the following timeline. The training starts at t = 0 s, followed immediately by the initialisations of software libraries and hardware. Before the first epoch starts, each worker reads the corresponding part of the compressed training dataset. For instance, if the training is parallelised on 4 MPI workers, then worker #1 reads the first quarter of the dataset and decompresses in its memory, worker #2 reads the second quarter of the dataset and also does the same decompression, and so on. The dataset is balanced, i.e. each class contains the same amount of training samples, and each

worker will read the same amount of training data. This makes sure that the scaling is not hampered by an imbalanced data distribution scenario where all other workers are waiting for the last one to finish the data load and gradients calculation. By the time all training data are in place, epoch 1 starts, but between its completion and the starting of epoch 2, there is a significant gap due to the initialisation effect (TensorFlow and Horovod are allocating all kinds of tensors to facilitate the exchange of gradients. The gap is narrowed between epoch 2 and 3, and following the third epoch the gap becomes sufficiently small. By this time, the training is stabilised. The 5 blocks of operations in Figure 38 corresponds to the 5 epochs of a test run.

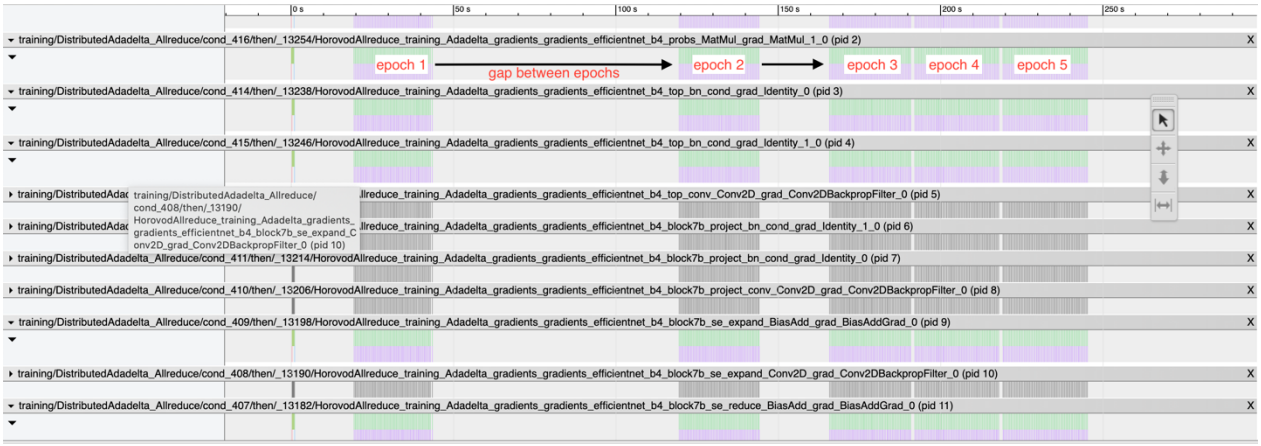

Figure 38: The Horovod timeline showing how the neural network gradients from different nodes are communicated and reduced. This figure only contains information about the communication between nodes; the actual computation time spent on individual nodes is not shown.

If we zoom-in to each epoch block, it looks like Figure 39. The `NEGOTIATE_ALLREDUCE` bar represents a phase when all workers send to rank 0 the signal that they are ready to reduce the given tensor (rank 0 is in charge of the general coordination among all workers). The `ALL_REDUCE` bars correspond to the actual phase when the hardware (CPUs or GPUs) is performing the reduction. There is also memory transfer of tensors, shown by the bars `MEMCPY_IN_FUSION_BUFFER` and `MEMCPY_OUT_FUSION_BUFFER`. The empty part of the timeline is the computational time on individual processors. Ideally, the overhead in `NEGOTIATE_ALLREDUCE`, `MEMCPY_IN` and `MEMCPY_OUT` should be minimised, which means that the time is mostly spent on the calculation on each processor. In practice, however, data transfer between processors also takes a significant amount of time, and this part depends on the size of gradient tensors to communicate.

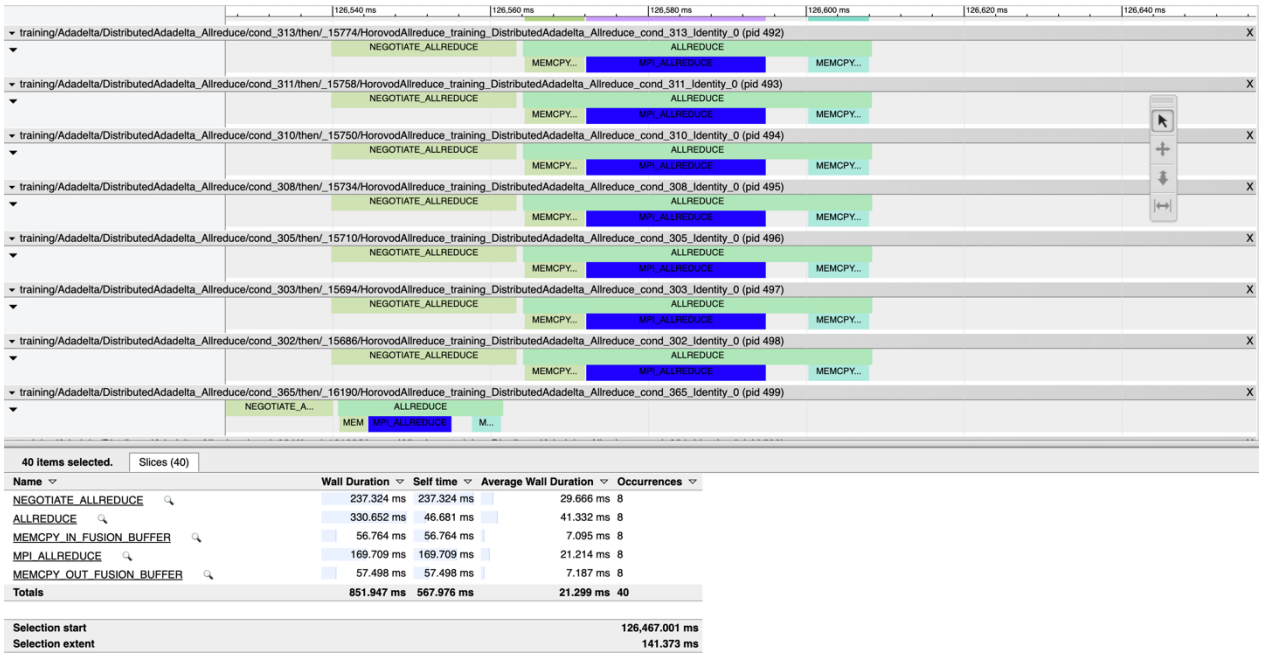

Figure 39: A zoom-in view of Figure 38 showing the communication between nodes in microsecond timescales.

In DeepGalaxy, the throughput is calculated with a timing call back. The timing starts immediately before an epoch starts, and stops immediately after an epoch finishes. That is, the preloading time is excluding in this timing, but communication overheads are included in this timing, and therefore it will affect the throughput. An example output looks like this:

```
[Performance] Epoch 0 takes 403.65 seconds. Throughput: 0.31 images/sec (per worker), 2.52 images/sec (total)

[Performance] Epoch 1 takes 328.76 seconds. Throughput: 0.39 images/sec (per worker), 3.09 images/sec (total)

[Performance] Epoch 2 takes 316.82 seconds. Throughput: 0.40 images/sec (per worker), 3.21 images/sec (total)

[Performance] Epoch 3 takes 319.10 seconds. Throughput: 0.40 images/sec (per worker), 3.18 images/sec (total)

[Performance] Epoch 4 takes 318.56 seconds. Throughput: 0.40 images/sec (per worker), 3.19 images/sec (total)
```

As mentioned, the gaps between epochs 1–2 and epochs 2–3 are substantial, which are also observed in the output of the timing call back in DeepGalaxy: the throughput is initially low, and eventually stabilised as of epoch 3. Therefore, a reliable and stable throughput is obtained by averaging the per-epoch throughput values from epoch 3 to the end of the training.

4.13.1 Performance on Hawk

Hawk is a CPU-based supercomputer, and therefore we carry out the benchmark using TensorFlow-CPU. Each node is equipped with 2 × AMD EPYC 7742 (Rome) processors, and each processor has 64 cores. For deep learning related tasks, the AVX2 instruction set is particularly useful in speeding up the tensor calculations. The CPU version of TensorFlow is compiled to take advantage of the AVX2 instruction set.

We pin each MPI worker to a CPU socket, which means that a worker should make full use of the 64 cores. Since hyperthreading is supported on each core, we set `OMP_NUM_THREADS` = 128 to make full utilisation of the processing power of the CPUs. We then alter the number of MPI workers (N_p). When $N_p > 2$, inter-node communication is needed, and the speed of the interconnect among the compute nodes is reflected in the scaling efficiency.

As shown in Figure 40,TensorFlow scales nicely on Hawk, thanks to its 200 Gbps InfiniBand HDR connectivity. The scaling is almost linear for N_p <= 64. Also, thanks to its new 64-core AMD EPYC 7742 CPUs, the throughput per MPI worker is decent.

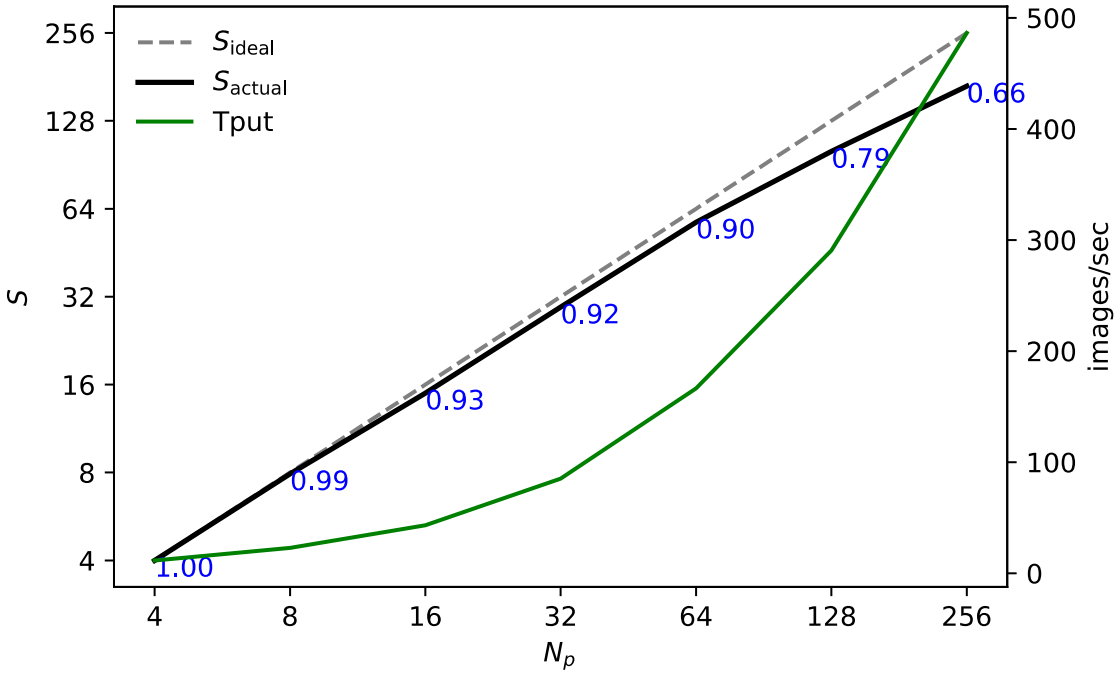

Figure 40: The scaling efficiency of TensorFlow on Hawk, annotated by blue numbers in the figure. The dashed line indicates a perfectly linear scaling where the speed-up factor (S) grows as a function of the number of MPI workers (N_p), and the black thick curve indicates the actual speed-up factor. The green curve is the throughput of the system as a whole (in the units of images per second).

We have inspected the CPU load on the compute node, and found that in general the CPUs cannot be 100% utilised. As shown below, the CPU is loaded at a capacity of 2/3. We have explored increasing the number of MPI ranks, but that doesn't make the CPU usage higher. Alternating the batch size can affect the CPU utilisation, since a larger batch size makes the TensorFlow more computationally bound than communication bound. However, if the batch size is too large, the workload becomes more memory bound, and when the local memory bandwidth becomes a bottleneck, the CPU utilisation is hampered as well. We found that a local batch size of 16 gives an optimal performance.

```
top - 16:52:29 up 2 days, 18:37,  1 user,  load average: 97.81, 24.86, 18.52
Tasks: 2795 total,   1 running, 2794 sleeping,   0 stopped,   0 zombie
%Cpu(s): 64.6 us,  1.8 sy,  0.0 ni, 32.8 id,  0.0 wa,  0.6 hi,  0.2 si,  0.0 st
MiB Mem : 257303.9 total, 219708.4 free,  35957.4 used,   1638.1 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used. 219704.1 avail Mem
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 371112 iprmacai  20   0   32.3g  14.7g 185992 S  8382   5.9  45:03.86 python
 371113 iprmacai  20   0   32.3g  14.7g 186096 S  8372   5.8  45:18.13 python
```

4.13.2 Performance on SuperMUC-NG

SuperMUC-NG is an Intel Skylake powered supercomputer, and therefore the TensorFlow benchmark is carried out on the CPUs. With the Intel Skylake architecture, advanced instruction sets such as AVX2 and AVX512 are available to speed-up the calculation of linear algebra operations in deep learning. The TensorFlow-CPU binary is compiled with these instruction sets enabled. TensorFlow makes use of oneAPI (formally known as MKL-DNN) as the low-level optimisation library.

Each SuperMUC-NG node has 2 × Xeon Platinum 8174 CPUs, each of which has 24 cores and supports hyperthreading to 48 threads in parallel. As such, we set OMP_NUM_THREADS to 48 to make full utilisation of the CPU computing power. Each MPI worker is pinned to a CPU socket. Within a CPU socket, there is no communication needed between OpenMP threads, but the gradients are communicated via MPI across CPU sockets and across compute nodes.

The scaling performance is shown in Figure 41. For the scenario of N_p <=64, the scaling is nearly linear, but this decays with larger N_p. Also, with fewer CPU cores and an older CPU architecture, the throughput of the system is less efficient. With 256 workers (6,144 cores, 12,288 OpenMP threads) the system can deliver about 380 images per second.

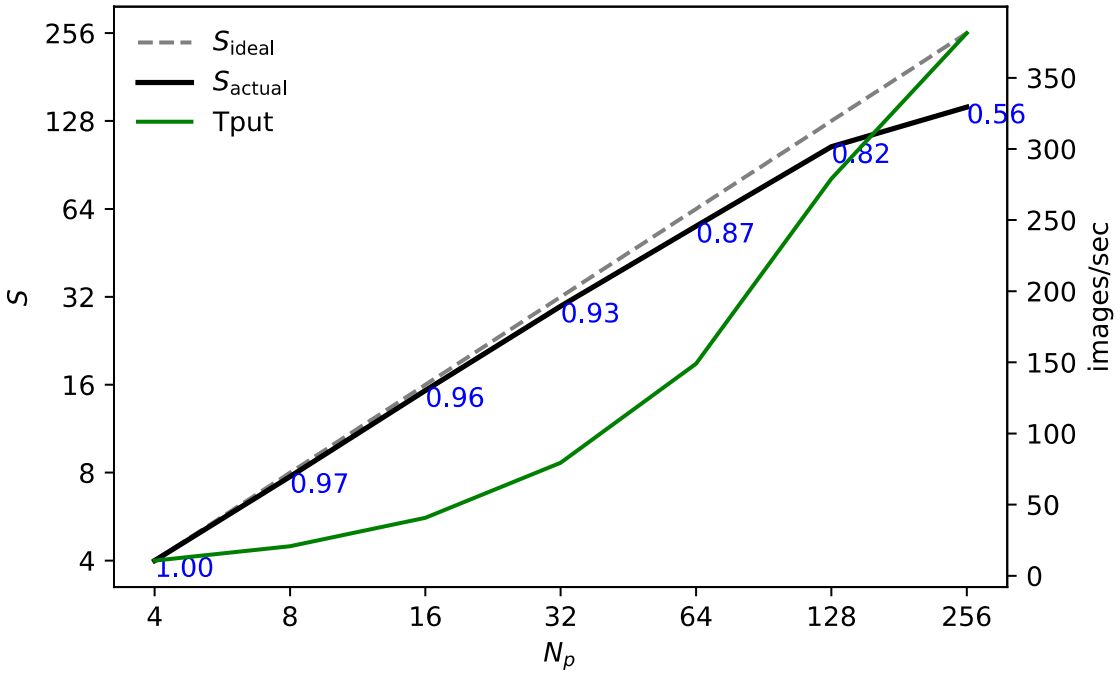

Figure 41: The scaling efficiency of TensorFlow on SuperMUC-NG, annotated by blue numbers in the figure. The dashed line indicates a perfectly linear scaling where the speed-up factor (S) grows as a function of the number of MPI workers (N_p), and the black thick curve indicates the actual speed-up factor. The green curve is the throughput of the system as a whole (in the units of images per second).

4.13.3 Performance on Lisa

Lisa is a high-performance compute cluster (located at SURF) with consumer-class GPUs. It has a GPU partition, in which each node is equipped with 4 × NVIDIA Titan RTX GPU (24 GB

GPU memory). To understand the effectiveness of GPUs for a deep learning workload, we carry out a TensorFlow-GPU benchmark on this cluster. TensorFlow invokes the linear algebra primitives with cuDNN, which is in turn accelerated with CUDA.

Figure 42 shows the scaling and throughput performance. Lisa is not originally designed for scaling, so the scaling performance beyond 4 nodes (16 workers) starts to decay. However, its throughput is excellent: a single GPU card delivers about 10× the performance of a CPU socket. This test demonstrates that GPUs are indeed a better architecture for deep learning workloads.

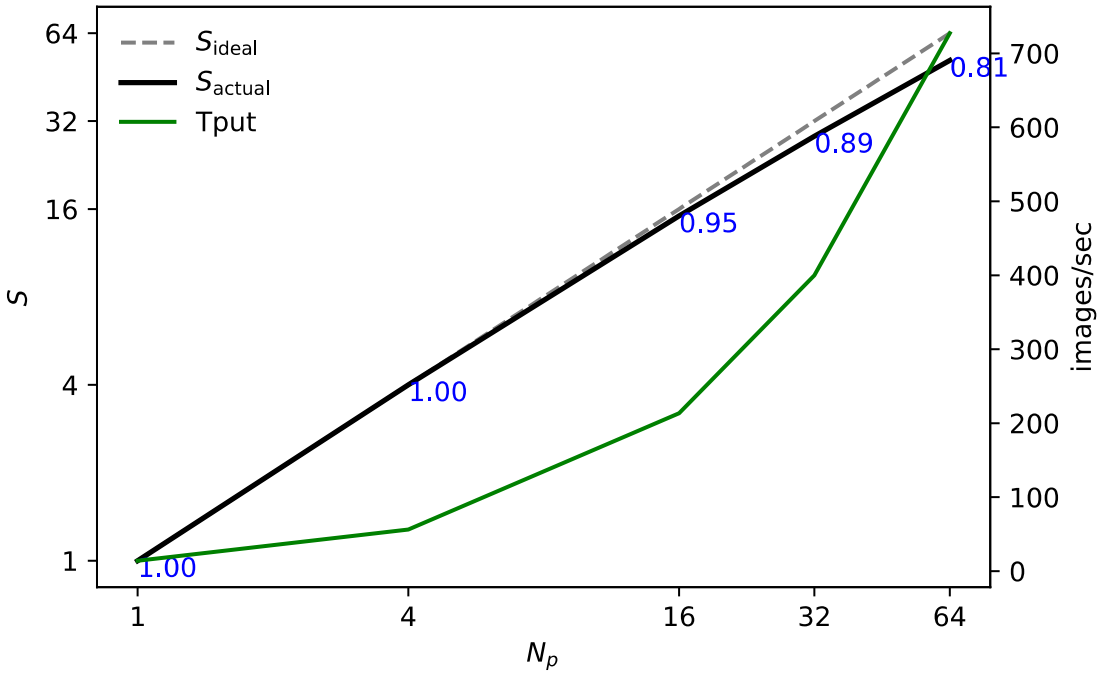

Figure 42: The scaling efficiency of TensorFlow on Lisa, annotated by blue numbers in the figure. The dashed line indicates a perfectly linear scaling where the speed-up factor (S) grows as a function of the number of MPI workers (N_p), and the black thick curve indicates the actual speed-up factor. The green curve is the throughput of the system as a whole (in the units of images per second).

5 Conclusions

The whole purpose of benchmarking is providing a metric for comparing systems. Clearly, one single (application) benchmark will not provide the answer to what the fastest/most efficient or most energy efficient system is. For this we will combine the previous results and derive a comparison of the overall performance of the systems. We will also derive a comparison of the energy efficiency for the few systems where we obtained energy measurements.

If you want to select the optimal system/architecture for a given application, please have a look at the corresponding Section 4 subsection where we present performance and energy efficiency results, analyses, and conclusions per application.

5.1 Performance Comparison of all Benchmark Systems

5.1.1 LINPACK Performance

To set a baseline, we provide the TOP500/HPL performance of the current PRACE Tier-0 systems in Table 152. In the last two columns, we provide the HPL performance per core (for GPUs we consider the SM units as cores), and a relative core performance (normalised using the maximum value). Although there is a lot to argue about the relevance of HPL for real applications performance, it still can be used as a starting point in comparing system performance. (It is relevant for dense linear algebra and other codes that can efficiently use AVX/SIMD instructions.) The ranking is more-or-less as expected, from highest to lowest: the NVIDIA A100 system (JUWELS-Booster), the highest clocked Skylake system (SuperMUC-NG), the NVIDIA V100 (MARCONI100) and P100 (Piz Daint) systems, the Skylake systems, the Rome systems and the Knights Landing system last. The Skylake systems' order correlates with their clock speed. The same is true for the two Rome systems. The lowest clocked KNL system still is close to the Rome systems since it can – like Skylake – perform AVX512 instructions, whereas Rome is limited to AVX256 instructions.

PRACE Tier-0 system	R_{peak} (Pflop/s)	R_{max} (Pflop/s)	Cores	R_{max} per core (Gflop/s/core)	Relative core performance
JUWELS-Booster	70.980	44.120	449,280	98.20	1.00
SuperMUC-NG	26.874	19.477	305,856	63.68	0.65
MARCONI100	29.354	21.640	347,776	62.22	0.63
Piz Daint	27.154	21.230	387,872	54.73	0.56
JUWELS-Cluster	9.891	6.178	114,480	53.96	0.55
Irene-SKL	6.636	4.066	79,488	51.15	0.52
MareNostrum4	10.296	6.471	153,216	42.23	0.43
Irene-Rome	12.039	6.988	197,120	35.45	0.36
Hawk	25.160	19.334	698,880	27.66	0.28
Irene-KNL	2.340	1.311	56,304	23.29	0.24

Table 152: TOP500 performance of PRACE Tier-0 systems

5.1.2 Application Performance

In Section 4 we provided a plethora of benchmark results, i.e. for many application benchmark / dataset / problem size – system combinations. If you are a PRACE user and are interested in running one of the UEABS applications, you are advised to study the relevant subsection. On the other hand, we want to provide some insight in the relative application performance of the benchmark systems presented in Section 3 and the additional systems that have been used in Section 4 for some of the applications. For this reason, we took a similar approach as in Section 5.1.1 and used selected performance results from the benchmark results in Section 4. If performance was determined as time to solution, we took the inverse value and divided this by the number of cores. If performance already was determined as some speed, we also divided this by the number of cores. Thus, we obtained an abstract speed metric per core. Subsequently, we normalised these values per application-test case combination by dividing all values by the highest abstract speed per core metric. This results in a relative application speed per core. Finally, we colour coded the relative speed per core: green for relative speed 1 (highest) and red for the lowest; and sorted the columns on their average speed. The results are presented in Table 153.

Application	Test Case	Piz Daint CPU	MARCONI100 CPU	JUWELS Cluster	Irene-SKL	SuperMUC-NG	MareNostrum4	Hawk	Irene-Rome	Lisa	Piz Daint	JUWELS Booster	HPC Vega	MARCONI100	Irene-KNL
Alya	A			0.95	0.84	1.00	0.74	0.75	0.69		0.24			0.17	
	B			1.00	0.66	0.97	0.81	0.73	0.72		0.57			0.26	
Code_Saturne	A			0.88	0.73	1.00	0.70	0.54	0.57						
	B			0.75	0.58	1.00	0.42	0.75	0.41						
	C			0.69	0.53	0.99		1.00	0.84						
	D			1.00	0.80	0.90		0.68							
CP2K	A	1.00		0.49		0.51	0.44	0.62	0.44		0.19			0.03	
	B	0.37		0.31		0.42	0.80	1.00	0.85		0.05			0.04	
	C	1.00		0.56		0.76	0.73	0.70	0.47		0.17			0.14	
GADGET	A			0.72			1.00								
GPAW	S			1.00		0.89	0.77	1.00	0.78						
	M			1.00		0.87	0.81	0.72	0.79						
	L			1.00		0.93	0.44	0.55	0.66						
GROMACS	A			0.64		1.00	0.79	0.73	0.62		0.38			0.20	
	B			0.98		1.00	0.76	0.44	0.25		0.52			0.14	
	C			0.82		0.95	0.83	1.00	0.88		0.36			0.24	
NAMD	A			0.33		0.23	0.66	0.42	0.42		1.00			0.11	
	B			0.35		0.41	0.60	0.69	0.58		1.00			0.16	
	C			1.00		0.41	0.71	0.70	0.65		0.73			0.79	
NEMO	AaX		0.93	0.94	0.77	0.72	0.74	1.00	0.65						
	AdX		1.00	0.71	0.79	0.50	0.84	0.84	0.80						
	Aa		1.00	0.85	0.72	0.65	0.87	0.81	0.62						
	Ad		1.00	0.70	0.97	0.37	0.82	0.82	0.78						
	BaX		0.35	0.09	0.50	0.36	0.36	1.00	0.56						
	BdX		0.56	0.16	0.93	0.69	0.69	0.69	1.00						
	Ba		0.67	1.00	0.49	0.68	0.83	0.65	0.43						
	Bd		0.29	0.44	1.00	0.50	0.52	0.52	0.87						
PFARM	1a			0.93	0.88	1.00	0.87	0.24	0.25			0.26		0.17	
	1b			0.89	0.84	1.00	0.84	0.20	0.24			0.20		0.18	
	1c			1.00	0.93	0.89	0.82	0.23	0.28			0.40		0.13	
	1d			0.77	0.72	1.00	0.80	0.21	0.24			0.18		0.09	
QCD	1			0.59	1.00	0.47	0.84				0.13	0.36		0.39	0.29
	2v1			0.59	0.47	0.63	0.47	1.00			0.19	0.42		0.02	
	2v2			0.27	0.31	0.36	0.33	0.30			0.39	1.00		0.16	
Quantum Espresso	M		1.00	0.46		0.50	0.69							0.06	
	L			0.77		0.73	1.00								
SPECFEM3D	V			0.32	0.16	0.24	0.20	0.04	0.04		1.00	0.27	0.31	0.47	0.06
	A			0.39	0.37	0.37	0.33	0.12	0.16		0.83	1.00	0.43	0.74	0.11
	B			0.79	1.00	0.68	0.62	0.25	0.28						
TensorFlow	A					1.00		0.48		0.54					

Table 153: Selected relative speed per core per application-dataset combination

Clearly, NEMO Test Cases BaX and BdX (B, attached and detached, including XIOS time) on JUWELS Cluster are outliers. For this reason, these values have not been taken into account while sorting the columns.

Looking at the green colours – and as expected – no single system gives the best results on all the benchmarks. It is therefore important for users to choose the best system for a given application in order to maximise the scientific output for a given amount of compute resources. Not all applications are suitable for running efficiently on GPUs. If so, the most recent GPUs in JUWELS Booster perform best. For non-accelerated codes, the fastest Skylake systems are attractive, but depending on the application characteristics also Rome systems can perform best. The least performant is the KNL system but that is no problem since Intel discontinued the Xeon Phi line. Please note that the previous comparison is based on core performance (or SM unit performance in the case of GPUs). A per node comparison or including power envelopes (see Section 5.2.2 below) will shift the picture: some system designs chose lower frequency SKUs to optimize for energy efficiency rather than single core performance, or chose a faster interconnect for application performance; Intel Skylake has AVX512 whereas AMD Rome has AVX256; an AMD Rome CPU has much more cores than an Intel Skylake CPU. These design approaches can lead to similar performance-per-cost ratios.

5.2 Energy Efficiency

5.2.1 LINPACK Energy Efficiency

To set a baseline, we provide the Green500/HPL energy efficiency – if listed – of the current PRACE Tier-0 systems in Table 154. In the last column we provide the relative energy efficiency (normalised using the maximum value). The ranking is as expected. The GPU-based systems score best ranked from most recent architecture to least recent architecture (NVIDIA A100, V100, and P100, respectively). Next are the AMD Rome systems the lowest clocked SKUs first. The Skylake are last and ordered from highest clock frequency to lowest.

PRACE Tier-0 system	R_{max} (Pflop/s)	Power (kW)	Power Efficiency (Gflop/J)	Relative power efficiency
JUWELS-Booster	44.120	1764	25,008	1.00
MARCONI100	21.640	1476	14,661	0.59
Piz Daint	21.230	2384	8,904	0.36
Hawk	19.334	3906	4,950	0.20
Irene-Rome	6.988	1436	4,866	0.19
JUWELS-Cluster	6.178	1361	4,539	0.18
Irene-SKL	4.066	917	4,434	0.18
MareNostrum4	6.471	1632	3,965	0.16

Table 154: Green500 energy efficiency of PRACE Tier-0 systems

5.2.2 Energy to Solution

In Table 155 we selected energy to solution measurements from Section 4 with (application, test case, size)-combinations having the largest system coverage but having at least measurements on two different systems. We normalised using the minimum energy for a given application-test case. Higher values mean higher energy to solution. We also added colouring

green (for the baseline, 1) – red (for the highest value). Clearly, NEMO Test Case AdX (A, detached, including XIOS time) on Irene-SKL is an outlier.

Application	Test Case	Size	Irene-Rome	Irene-SKL	MARCONI100	MareNostrum4	SuperMUC-NG	Piz Daint GPU	Piz Daint CPU
Code_Saturne	A	16 nodes	1.00	1.19		1.77			
	B	128 nodes	1.64	1.00		1.65			
	C	128 nodes	1.00	1.37					
CP2K	A	32 nodes	1.36			2.20	1.00	1.24	1.22
	B	128 nodes	1.09			1.97	1.00	1.96	1.62
	C	128 nodes	1.51			2.22	1.26	1.00	1.48
GROMACS	A	16 nodes					1.26	1.00	
	B	64 nodes					1.93	1.00	
	C	128 nodes				1.22	1.33	1.00	
NAMD	A	64 nodes					6.91	1.00	
	B	128 nodes				3.04	4.29	1.00	
	C	256 nodes				3.78	4.24	1.00	
NEMO	AaX	1024 cores	1.00	1.96		2.18			
	AdX	1024 cores	1.00	8.92		8.10			
	BaX	10240 cores	1.00	1.68		9.58			
	BdX	10240 cores	1.00	2.26		10.21			
PFARM	1a	16 nodes	3.07	2.90	1.00	4.46			
	1b	128 nodes	2.71	2.34	1.00	3.89			
	1c	16 nodes	1.89	1.88	1.00	3.23			
	1d	256 nodes	1.50	1.47	1.00	2.18			

Table 155: Selected relative energy to solution measurements

There is not sufficient data to justify general conclusions. There are only two GPU-based systems in this table. PFARM is the only application for which MARCONI100 energy to solution results have been produced. Here, MARCONI100 is the clear winner. For other GPU enabled codes, Piz Daint is the clear winner. This is completely in line with the results in Table 154. From the CPU-based systems, Irene-Rome is the most energy efficient for Code_Saturne and NEMO followed by SuperMUC-NG for CP2K. Unfortunately, we cannot relate these results to Green500 results, since they are not available for SuperMUC-NG.