E-Infrastructures
H2020-EINFRA-2016-2017

EINFRA-11-2016: Support to the next implementation phase of Pan-European High Performance Computing Infrastructure and Services (PRACE)

PRACE-5IP

PRACE Fifth Implementation Phase Project

Grant Agreement Number: EINFRA-730913

D7.5

Evaluation of Accelerated and Non-accelerated Benchmarks
Final

Version: 1.0
Authors: Walter Lioen (SURFsara), Miguel Avillez (UEVORA), Valeriu Codreanu (SURFsara), Dimitris Dellis (GRNET), Sagar Dolas (SURFsara), Andrew Emerson (CINECA), Jacob Finkenrath (CyI), Cédric Jourdain (CINES), Martti Louhivuori (CSC), Cristian Morales (BSC), Charles Moulinec (STFC), Arno Proeme (EPCC), Andrew Sunderland (STFC)
Date: 18.04.2019

Project and Deliverable Information Sheet

PRACE Project	**Project Ref. №: EINFRA-730913**	
	Project Title: PRACE Fifth Implementation Phase Project	
	Project Web Site: http://www.prace-project.eu	
	Deliverable ID: < **D7.5**>	
	Deliverable Nature: <DOC_TYPE: Report>	
	Dissemination Level: PU*	**Contractual Date of Delivery:** 30 / April / 2019
		Actual Date of Delivery: 30 / April / 2019
	EC Project Officer: Leonardo Flores Añover	

* - The dissemination level is 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 Accelerated and Non-accelerated Benchmarks**	
	ID: D7.5	
	Version: <1.0>	**Status:** ***Final***
	Available at: http://www.prace-project.eu	
	Software Tool: Microsoft Word 2013	
	File(s): D7.5.docx	
Authorship	**Written by:**	Walter Lioen (SURFsara), Miguel Avillez (UEVORA), Valeriu Codreanu (SURFsara), Dimitris Dellis (GRNET), Sagar Dolas (SURFsara), Andrew Emerson (CINECA), Jacob Finkenrath (CyI), Cédric Jourdain (CINES), Martti Louhivuori (CSC), Cristian Morales (BSC), Charles Moulinec (STFC), Arno Proeme (EPCC), Andrew Sunderland (STFC)
	Contributors:	Giannis Koutsou, CyI Srijit Paul, CyI
	Reviewed by:	Florian Berberich, JUELICH Nikos Nikoloutsakos, GRNET
	Approved by:	MB/TB

Document Status Sheet

Version	Date	Status	Comments
0.1	27/February/2019	Draft	Skeleton
0.2	10/March/2019	Draft	First integrated version
0.3	17/March/2019	Draft	
0.4	24/March/2019	Draft	
0.5	31/March/2019	Draft	
0.6	01/April/2019	Draft	For PRACE internal review
0.7	12/April/2019	Draft	
1.0	18/April/2019	Final version	For MB/TB approval

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

[2] 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

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

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

[5] 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

[6] SPECFEM3D_Globe GitHub repository: https://github.com/geodynamics/specfem3d_globe

[7] SPECFEM3D_Globe website: http://www.geodynamics.org/cig/software/specfem3d-globe

[8] Alya website: https://www.bsc.es/research-development/research-areas/engineering-simulations/alya-high-performance-computational

[9] 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

[10] UEABS, the Unified European Application Benchmark Suite. Original location on PRACE RI website: http://www.prace-ri.eu/ueabs/. This repository is obsoleted by [12].

[11] PRACE Accelerator Benchmark Suite. Original GitLab location: https://misterfruits.gitlab.io/ueabs/ms33.html. This repository is obsoleted by [12].

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

[13] 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. http://www.prace-ri.eu/IMG/pdf/Identification_and_Categorisatio_of_Applications_and_Initial_Benchmark_Suite_final.pdf

[14] 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. http://www.prace-ri.eu/IMG/pdf/D6-3-1.pdf

[15] Peter Michielse, Lukas Arnold, Olli-Pekka Lehto, and Walter Lioen. *Final Benchmark Suite*. PRACE-PP Deliverable D6.3.2. June 18, 2010. http://www.prace-ri.eu/IMG/pdf/D6-3-2-extended.pdf

[16] 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. http://www.prace-ri.eu/IMG/pdf/d7.4.2_1ip.pdf

[17] Mark Bull. *Unified European Applications Benchmark Suite*. PRACE-2IP Deliverable 7.4. July 26, 2013. http://www.prace-ri.eu/IMG/pdf/d7.4_3ip.pdf

[18] Mark Bull. *UEABS Benchmarking Results*. PRACE-3IP Deliverable D7.3.2. February 20, 2014. http://www.prace-ri.eu/IMG/pdf/d7.3.2_3ip.pdf

[19] 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. http://www.prace-ri.eu/IMG/pdf/WP212.pdf

[20] Victor Cameo Ponz. *Application performance on accelerators*. PRACE-4IP Deliverable D7.5. March 24, 2017. http://www.prace-ri.eu/IMG/pdf/D7.5_4ip.pdf

[21] Victor Cameo Ponz. *Performance and energy metrics on PCP systems*. PRACE-4IP Deliverable D7.7. January 8, 2018. http://www.prace-ri.eu/IMG/pdf/D7.7_v1.2_4ip.pdf

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

[23] *Best Practice Guide – Modern Interconnects.* http://www.prace-ri.eu/best-practice-guide-modern-interconnects

[24] Network Cray Fabric Aries: https://www.hlrn.de/twiki/pub/NewsCenter/ParProgWorkshopFall2017/03_Networks_Cray_Aries.pdf

[25] ExaNoDe Deliverable D2.5: http://exanode.eu/wp-content/uploads/2017/04/D2.5.pdf

[26] Hazel Hen, Tier-0 system at HLRS, Germany: https://www.hlrs.de/systems/cray-xc40-hazel-hen/

[27] Irene, Tier-0 system at CEA, France: http://www-hpc.cea.fr/en/complexe/tgcc-Irene.htm

[28] JUWELS, Tier-0 system at JSC, Germany: http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUWELS/JUWELS_node.html

[29] Marconi, Tier-0 system at CINECA, Italy: http://www.hpc.cineca.it/hardware/marconi

[30] MareNostrum, Tier-0 system at BSC, Spain: https://www.bsc.es/MareNostrum/MareNostrum

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

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

[33] DAVIDE PCP prototype at CINECA, Italy: https://www.e4company.com/en/?id=press&section=1&page=&new=davide_supercomputer

[34] Frioul PCP prototype at CINES, France: https://www.cines.fr/le-supercalculateur-frioul/

[35] DEEP-ER prototype SDV at JSC, Germany: https://www.deep-projects.eu/hardware/prototypes.html

[36] Mont-Blanc 3 prototype Dibona: https://www.montblanc-project.eu/prototypes

[37] Sisu Tier-1 system at CSC, Finland: https://research.csc.fi/sisu-supercomputer

[38] GADGET website: https://wwwmpa.mpa-garching.mpg.de/gadget/

[39] Alig, C., Schartmann, M., Burkert, A., & Dolag, K., 2013, ApJ, 771, 119

[40] Springel V., Yoshida N., White S. D. M., 2001, New Astronomy, 6, 51

[41] Springel V., 2005, MNRAS, 364, 1105

[42] Nils Meyer, Peter Georg, Dirk Pleiter, Stefan Solbrig, and Tilo Wettig. *SVE-enabling Lattice QCD Codes*. arXiv:1901.07294 [cs.DC]. DOI: 10.1109/CLUSTER.2018.00079. 2018 IEEE International Conference on Cluster Computing (CLUSTER), p. 623.

[43] Peter A. Boyle, Guido Cossu, Azusa Yamaguchi, and Antonin Portelli. *Grid: A next generation data parallel C++ QCD library*. DOI: 10.22323/1.251.0023. PoS LATTICE2015 (2016) 023.

[44] https://sc18.supercomputing.org/proceedings/tech_poster/tech_poster_pages/post149.html

[45] 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

[46] Stephen Booth. Technical lessons learnt from the implementation of the joint PCP for PRACE-3IP. PRACE-3IP Deliverable D8.3.4, January 10, 2018. http://www.prace-ri.eu/IMG/pdf/d8.3.4-3ip.pdf

[47] https://www.cp2k.org/dev:compiler_support

[48] https://www.cp2k.org/howto:compile_with_cuda

[49] https://www.cp2k.org/howto:libcusmm

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

[51] Code_Saturne website: https://www.code-saturne.org/cms/

[52] PRACE CodeVault: https://repository.prace-ri.eu/git/PRACE/CodeVault

[53] PCP Conclusion Report. PRACE. To appear April 2019.

[54] *Best Practice Guide – Knights Landing*. http://www.prace-ri.eu/IMG/pdf/Best-Practice-Guide-Knights-Landing.pdf

List of Acronyms and Abbreviations

AISBL	Association International Sans But Lucratif (legal form of the PRACE-RI)
ADMM	Auxiliary Density Matrix Method
AMBER	Assisted Model Building with Energy Refinement
APS	Application Performance Snapshot (Intel)
ARM	previously Advanced RISC Machine, originally Acorn RISC Machine
AVX	Advanced Vector Extensions
BCO	Benchmark Code Owner
BDW	Broadwell (Intel)
BEO	Bull Energy Optimizer
BLAS	Basic Linear Algebra Subprograms
BXI	Bull eXascale Interconnect
CAPMC	Cray Advanced Platform Monitoring and Control
CFD	Computational Fluid Dynamics
CG	Conjugate Gradients
CHARMM	Chemistry at HARvard Macromolecular Mechanics
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture (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
DFE	DataFlow Engine
DFT	Density Functional Theory
DFTB	Density Functional based Tight Binding
DIMM	Dual In-line Memory Module
DoA	Description of Action (formerly known as DoW)
DoW	Description of Work
DP	Double Precision
DRAM	Dynamic Random-Access Memory
EC	European Commission
EDF	Électricité de France R&D
EDR	Enhanced Data Rate
ELPA	Eigenvalue Solvers for Petaflop Applications
EoI	Expression of Interest
FCC	Face-Centred Cubic
FFT	Fast Fourier Transform

FFTW	Fastest Fourier Transform in the West
flop	floating-point operation
FPGA	Field-Programmable Gate Array
FP32	32-bit Floating-Point
FP64	64-bit Floating-Point
FWI	Full Waveform Imaging
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
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
GPFS	General Parallel File System
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
HDEEM	High Definition Energy Efficiency Monitoring
HDEEMVIZ	HDEEM VIZualization
HDF5	Hierarchical Data Format 5
HFI	Host Fabric Interface
HPC	High Performance Computing; Computing at a high performance level at any given time; often used synonym with Supercomputing
HPL	High Performance LINPACK
HSW	Haswell (Intel)
HT	HyperThreading
IB	InfiniBand
IBM	International Business Machines
IO	Input/Output
IPMB	Intelligent Platform Management Bus/Bridge
JUWELS	Jülich Wizard for European Leadership Science (Tier-0 system)
KB	Kilo (= 2^{10} ~10^3) Bytes (= 8 bits), also KByte
KNC	Knights Corner (Intel)
KNL	Knights Landing (Intel)
LAPACK	Linear Algebra PACKage
LDA	Local Density Approximation
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
MD5	Message-Digest 5
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)
MoU	Memorandum of Understanding.
MPI	Message Passing Interface
NAMD	Nanoscale Molecular Dynamics
NEB	Nudged Elastic Band
NEMO	Nucleus for European Modelling of the Ocean
NetCDF	Network Common Data Form
NFS	Network File System
NIC	Network Interface Controller
NVM	Non-Volatile Memory
OMP	OpenMP
OPA	Omni-Path (Intel)
OpenACC	Open Accelerators
OpenCL	Open Computing Language
OpenMP	Open Multi-Processing
OpenMPI	Open MPI
PA	Preparatory Access (to PRACE resources)
PABS	PRACE Application Benchmark Suite
PAPI	Performance Application Programming Interface
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
PLE	Parallel Locator Exchange
POSIX	Portable Operating-System Interface
PRACE	Partnership for Advanced Computing in Europe; Project Acronym
PRACE 2	The PRACE Research Infrastructure following the initial five year period.
pthreads	POSIX threads
PWscf	Plane-Wave Self-Consistent Field
RI	Research Infrastructure
QCD	Quantum ChromoDynamics
QE	Quantum Espresso
QUDA	A library for QCD on GPUs
RAM	Random-Access Memory
RUR	Resource Utilisation Reporting
ScaLAPACK	Scalable LAPACK
SCF	Self-Consistent Field method
SDV	Software Development Vehicle (DEEP-ER prototype)

SEM	Spectral-Element Method
SHOC	Scalable HeterOgeneous Computing
SIMD	Single Instruction Multiple Data
SKL	Skylake (Intel)
SLURM	Simple Linux Utility for Resource Management (batch system)
SM	Streaming Multiprocessor
SMT	Simultaneous MultiThreading
SPH	Smoothed Particle Hydrodynamics
SSD	Solid-State Drive
SSE	Streaming SIMD Extensions
STMV	Satellite Tobacco Mosaic Virus
SuSE	Software und System-Entwicklung
SVE	Scalable Vector Extension (ARM)
targetDP	target Data Parallel
TB	Technical Board (group of Work Package leaders)
TB	Tera (= 2^{40} ~ 10^{12}) Bytes (= 8 bits), also TByte
TCO	Total Cost of Ownership. Includes recurring costs (e.g. personnel, power, cooling, maintenance) in addition to the purchase cost.
TDP	Thermal Design Power
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
TreePM	Tree Particle–Mesh
UEABS	Unified European Applications Benchmark Suite
VSX	Vector Scalar Extension
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 UYBHM)
BSC	Barcelona Supercomputing Center - Centro Nacional de Supercomputación, Spain
CaSToRC	Computation-based Science and Technology Research Center, Cyprus
CCSAS	Computing Centre of the Slovak Academy of Sciences, Slovakia
CEA	Commissariat à l'Énergie Atomique et aux Énergies Alternatives, France (3rd Party to GENCI)
CESGA	Fundación 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 (3rd Party to GENCI)
CNRS	Centre National de la Recherche Scientifique, France (3rd 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 PSNC)
CyI	The Cyprus Institute, Cyprus
EPCC	EPCC at The University of Edinburgh, UK
ETHZurich (CSCS)	Eidgenössische Technische Hochschule Zürich – CSCS, Switzerland
FIS	FACULTY OF INFORMATION STUDIES, Slovenia (3rd Party to ULFME)
GCS	Gauss Centre for Supercomputing e.V., Germany
GENCI	Grand Équipement National de Calcul Intensif, France
GRNET	Greek Research and Technology Network, Greece
INRIA	Institut National de Recherche en Informatique et Automatique, France (3rd Party to GENCI)
IST	Instituto Superior Técnico, Portugal (3rd Party to UC-LCA)
IT4Innovations	IT4Innovations National supercomputing centre at VŠB-Technical University of Ostrava, Czech Republic
IUCC	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)
LiU	Linkoping University, Sweden (3rd Party to SNIC)
NCSA	NATIONAL CENTRE FOR SUPERCOMPUTING APPLICATIONS, Bulgaria
NTNU	The Norwegian University of Science and Technology, Norway (3rd Party to SIGMA)
NUI-Galway	National University of Ireland Galway, Ireland
PRACE	Partnership for Advanced Computing in Europe aisbl, Belgium
PSNC	Poznan Supercomputing and Networking Center, Poland
RISCSW	RISC Software GmbH
RZG	Max Planck Gesellschaft zur Förderung der Wissenschaften e.V., Germany (3rd Party to GCS)
SIGMA2	UNINETT Sigma2 AS, Norway
SNIC	Swedish National Infrastructure for Computing (within the Swedish Science Council), Sweden
STFC	Science and Technology Facilities Council, UK (3rd Party to EPSRC)
SURFsara	Dutch national high-performance computing and e-Science support center, part of the SURF cooperative, Netherlands
UC-LCA	Universidade de Coimbra, Laboratório de Computação Avançada, Portugal
UCPH	Københavns Universitet, Denmark
UHEM	Istanbul Technical University, Ayazağa Campus, Turkey
UiO	University of Oslo, Norway (3rd Party to SIGMA)
ULFME	UNIVERZA V LJUBLJANI, Slovenia
UmU	Umea University, Sweden (3rd Party to SNIC)
UnivEvora	Universidade de Évora, Portugal (3rd Party to UC-LCA)
UPC	Universitat Politècnica de Catalunya, Spain (3rd Party to BSC)
UPM/CeSViMa	Madrid Supercomputing and Visualization Center, Spain (3rd Party to BSC)

USTUTT-HLRS	Universität Stuttgart – HLRS, Germany (3rd Party to GCS)
WCNS	Politechnika Wrocławska, Poland (3rd Party to PSNC)

Executive Summary

The Unified European Application Benchmark Suite (UEABS) provides a publicly available benchmark suite. One of the key results of this activity is the re-unification of the UEABS and the accelerator benchmark suite so that the UEABS lives up to its Unified name again. This new release is migrated to the PRACE GitLab server (next to the CodeVault repository). We present benchmark results and performance analyses on PRACE Tier-0 systems, on two PRACE PCP prototypes, on a DEEP-ER prototype, and on a Mont-Blanc 3 prototype. (If you want to select the optimal system/architecture for a given UEABS application, please have a look at these results.) 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 energy efficiency of the two benchmarked PCP prototypes strongly depends on the application benchmark / data set / problem size / node count. Overall, the GPU based system (DAVIDE) is somewhat more energy efficient than the KNL system (Frioul). If we add the GPU based Piz Daint system to the comparison, then Piz Daint clearly is the most energy efficient system.

As expected, the optimal system/architecture also strongly depends on the application benchmark / data set / problem size / node count. Overall the most recent Intel Skylake systems are the most performant, JUWELS being the fastest. For applications that can exploit GPUs, Piz Daint is most performant. On the other end of the spectrum, the systems based on the discontinued Knights Landing in general are least performant. 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) [12] is a set of currently 13 application codes taken from the pre-existing DEISA Benchmark Suite [9], the PRACE Application Benchmark suite (PABS) [15], and the PRACE Accelerator Benchmark Suite [20]. 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 [13][14][15] and the benchmark activities continued in PRACE-1IP [16]. The UEABS itself was only publicly released (Version 1.0) by the PRACE-2IP project [17]. Benchmarking activities continued in PRACE-3IP

resulting in a new release (Version 1.1) and a benchmark report [18]. 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 [19] was based on a subset of the UEABS Version 1.2, where some applications were removed because of lack of accelerator potential; and one application and a synthetic benchmark have been added. The Accelerator Benchmark Suite was published as GitLab repository [11] and a benchmark report targeting GPUs and Xeon Phi has been produced [20]. Finally, in the PRACE-4IP extension, a benchmark report targeting the PCP prototypes has been produced [21].

1.2 Work Described in this Report

In the PRACE-5IP DoA we committed the following: "This task will also update and maintain the Unified European Applications Benchmark Suite (UEABS), so that it can be used in future procurements and to help European researchers choose systems that are appropriate for their computational requirements. The accelerator versions of the benchmarks developed under PRACE-4IP will be re-integrated, and we will investigate putting the benchmark suite into CodeVault[1]. We will evaluate the results on PRACE systems from the standard benchmarks to the accelerated benchmarks, compare where both are available, and will strive to identify reasons for, and patterns in, the performance." The original benchmark scope has been extended by including two PRACE-3IP PCP [46] prototype systems: DAVIDE and Frioul; the Mont-Blanc 3 prototype system Dibona; and the DEEP-ER prototype system SDV.

In May 2018, we released UEABS version 2.0: an integrated version of the accelerated and non-accelerated version and published it on the PRACE git repository (where also CodeVault can be found but in a separate repository). For this we reconstructed a versioned git repository from a "flat" website representation and merged back the (UEABS Version 1.2 based) Accelerator Benchmark Suite. In April 2019 we will release UEABS version 2.1, an updated version that reflects the applications and datasets as used in this report.

1.3 Outline

Section 2 describes the application benchmarks, the test problems and data sets. 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.

[1] PRACE CodeVault [52] is an open repository containing various high-performance computing code samples. The project aims to support self-learning of HPC programming and will be used as an Open platform for the HPC community to share example code snippets, proof-of-concept codes and so forth.

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 [8] is a Computational Mechanics code capable of solving different types of physics, each one with its own modelisation 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 GPU 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. It is expected to scale up to 1500 MPI tasks.
- Test Case B: A 1056 million element mesh representing the flow around a sphere. It is expected to scale up to 12000 MPI tasks.
- Test Case C: A 68.8 million element mesh representing the flow around a sphere. It is expected to scale up to 750 MPI tasks.

2.2 Code_Saturne

2.2.1 Code Description

Code_Saturne [51] is open-source multi-purpose CFD software, primarily developed by EDF R&D and maintained by them. It 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. 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 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 Fortran 95 for most of the physics implementation. MPI is used on distributed memory machines and OpenMP pragmas have been added to the most costly parts of the code to handle potential shared memory. The version used in this work (also freely available) relies also on 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 two test cases chosen for the benchmark suite have been designed to assess the velocity-pressure coupling computation, and rely on the same configuration, with a mesh 8 times larger for Test Case B than for Test Case A, the time step being halved to ensure a correct Courant number.

2.2.2 Test Cases

Two test cases are dealt with, where only the mesh size has been changed. Depending on the architecture run on and the type of physics investigated, it is expected that 10,000 (IBM BlueGene/Q) to 25,000 cells (classical CPU-based machine) 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. This case is supposed to scale up to 1,300 (resp. 520) MPI tasks for 10,000 (resp. 25,000) cells per MPI task, depending on the machine.
- Test Case B: A 111 million tetrahedral cell mesh to simulate a laminar flow in a 3-D lid-driven cavity. This case is supposed to scale up to 11,100 (resp. 4,440) MPI tasks for 5,000 (resp. 25,000) cells per MPI task, depending on the machine.

2.3 CP2K

2.3.1 Code Description

CP2K is a quantum chemistry and solid-state physics software package that can perform atomistic simulations of solid state, liquid, molecular, periodic, material, crystal, and biological systems. CP2K provides a general framework for different modelling methods such as DFT using the mixed Gaussian and plane waves approaches GPW and GAPW. Supported theory levels include DFTB, LDA, GGA, MP2, RPA, semi-empirical methods (AM1, PM3, PM6, RM1, MNDO, ...), and classical force fields (AMBER, CHARMM, ...). CP2K can do simulations of molecular dynamics, metadynamics, Monte Carlo, Ehrenfest dynamics, vibrational analysis, core level spectroscopy, energy minimisation, and transition state optimisation using NEB or dimer method.

CP2K is written in Fortran 2008 and can be run in parallel using a combination of multi-threading, MPI, and CUDA. All of CP2K is MPI parallelised, with some additional loops also being OpenMP parallelised. It is therefore most important to take advantage of MPI parallelisation, however running one MPI rank per CPU core often leads to memory shortage. At this point OpenMP threads can be used to utilise all CPU cores without suffering an overly large memory footprint. The optimal ratio between MPI ranks and OpenMP threads depends on the type of simulation and the

system in question. CP2K supports CUDA, allowing it to offload some linear algebra operations including sparse matrix multiplications to the GPU through its DBCSR acceleration layer. FFTs can optionally also be offloaded to the GPU. Benefits of GPU offloading may yield improved performance depending on the type of simulation and the system in question.

CP2K strictly requires BLAS, LAPACK and ScaLAPACK and benefits strongly from FFTW. The application can furthermore make use of a number of further performance-enhancing and functionality-extending libraries. For the purpose of performing the benchmarks reported here on a range of systems CP2K was linked to libint and libxc in addition to the abovementioned strictly required libraries and FFTW.

2.3.2 Test Cases

This section details the CP2K benchmark test cases that were run. None of the test cases make use of large input data, so initialisation cost is minimal. Inputs consist of CP2K-format input files and potential and basis set input files.

Test Case A: H2O-512

Ab-initio molecular dynamics simulation of liquid water using the Born-Oppenheimer approach, via Quickstep DFT. Production quality settings for the basis sets (TZV2P) and the planewave cutoff (280 Ry) are chosen, and the Local Density Approximation (LDA) is used for the calculation of the Exchange-Correlation energy. The configurations were generated by classical equilibration, and the initial guess of the electronic density is made based on Atomic Orbitals. The system contains 512 water molecules (1536 atoms, 4096 electrons) in a 12.4 $Å^3$ cell and MD is run for 10 steps.

Test Case B: LiH-HFX

This is a single-point energy calculation using Quickstep GAPW (Gaussian and Augmented Plane-Waves) with hybrid Hartree-Fock exchange. It consists of a 216 atom Lithium Hydride crystal with 432 electrons in a 12.3 $Å^3$ cell. These types of calculations are generally around one hundred times the computational cost of a standard local DFT calculation, although this can be reduced using the Auxiliary Density Matrix Method (ADMM). Using OpenMP is likely to be of benefit here as the HFX implementation requires a large amount of memory to store partial integrals. By using several threads, fewer MPI processes share the available memory on the node and thus enough memory is available to avoid recomputing any integrals on-the-fly, improving performance.

Test Case C: H2O-DFT-LS

This is a single-point energy calculation using linear-scaling DFT. It consists of 6144 atoms in a 39 $Å^3$ box (2048 water molecules in total). An LDA functional is used with a DZVP MOLOPT basis set and a 300 Ry cut-off. For large systems the linear-scaling approach for solving Self-Consistent-Field equations will be much cheaper computationally than using standard DFT and allows scaling up to 1 million atoms for simple systems. The linear scaling cost results from the fact that the algorithm is based on an iteration on the density matrix. The cubically-scaling orthogonalisation step of standard Quickstep DFT using OT is avoided and the key operation is sparse matrix-matrix multiplications, which have a number of non-zero entries that scale linearly with system size. These are implemented efficiently in the CP2K-internal DBCSR library.

2.4 GADGET

2.4.1 Code Description

GADGET [38] is a cosmological, parallelised N-body and Smoothed Particle Hydrodynamics (SPH) code [40][41] that is tailored to solve a wide range of astrophysical problems, e.g., large-scale structure formation (formation of galaxies in the Universe), colliding and merging galaxies, studying the dynamics of the gaseous intergalactic medium, formation of the stars and its regulation, tidal disruption events by massive black holes. In all these types of simulations, GADGET follows the evolution of a self-gravitating collisionless N-body system, and solves the Euler equations by means of SPH.

The code major components are the time integration model, the tree-code module (hierarchical tree algorithm, optionally in combination with a particle-mesh scheme for long-range gravitational forces) to compute gravitational forces, the communication scheme for gravitational and SPH forces, a domain decomposition strategy based on orthogonal bisection, the entropy-based formulation of SPH (conserving energy and entropy in regions free of dissipation, while allowing for fully adaptive smoothing length), and the TreePM functionality. Both the force computation and the time stepping of GADGET are fully adaptive, with a dynamic range which is, in principle, unlimited.

The code uses an explicit communication model and is parallelised with MPI. The domain decomposition scheme ensures that the results of forces depend on the number of used processors/cores, which is usually obtained by using orthogonal bisection in domain decomposition. The scheme uses a Peano-Hilbert spatial filling fractal curve to become a three-dimensional space in the one-dimensional curve. This is then simply divided into parts that define the different domains. This scheme has several advantages, such as the fact that points that are close in the one-dimensional curve usually are close in the three-dimensional space. For some code parts, GADGET-3 can also use either Pthreads or OpenMP for a hybrid MPI/shared-memory parallelisation.

The latter is important to overcome the slowdown the overall performance of the code in calculations that put a great stress on the domain decomposition, e.g., the case of extreme high particle velocities in the vicinity of black holes (see, e.g., [39]). After a few iterations, particle properties need to be communicated to other domains, which could reside on non-local CPUs. This strongly increased need for communication slows down the overall performance. The hybrid OpenMP-MPI implementation in GADGET helps to overcome this problem by reducing the number of MPI tasks to the number of physical CPUs on each compute node and for every MPI task spawning additional OpenMP tasks corresponding to the number of cores on each of the CPUs. With this approach, a larger number of particles can be processed locally without the need of MPI communication. This can have an effect up to a factor of 4 in performance over the standard MPI implementation [39].

The code can in principle be started using an arbitrary number of processors, but the communication algorithms will be most efficient for powers of 2. It is also possible to use a single processor only, in which case the code behaves like a serial code, except that GADGET will still

go through some of the overhead induced by the parallelisation algorithms, so the code will not quite reach the same performance as an optimal serial solution in this case.

GADGET, as well as the included initial conditions generator N-GENIC (after the initial conditions are generated they are folded into GADGET), is written in C and requires the open-source GSL (GNU Scientific Library, which is needed for a few cosmological integrations at start-up, and for random-number generation), FFTW (Fastest Fourier Transform in the West; It is only needed for simulations that use the TreePM algorithm. Note that the MPI-capable version 2.x of FFTW is required, and that FFTW needs to be explicitly compiled with parallel support enabled), and HDF5 (Hierarchical Data Format ver. 5; GADGET can be compiled without this library, but then the HDF5 format is not supported) libraries.

The code originally developed by Volker Springel was first publicly released in 2000 (GADGET-1 and GADGET-1.1, the latter corrected a bug in the forcetree.c file) was followed by GADGET-2 (2005), and later GADGET-3. Version 4 is expected to be released during 2019.

2.4.2 Test Cases

Three test cases are considered in order to determine weak and strong scaling of the GADGET code by using a simulation of a cosmological structure formation in a periodic box with adiabatic gas physics. The cosmological density field is modelled with dark matter and gas. The initial conditions of the simulations are generated by the N-GenIC code running in parallel using 8, 128, and 1024 cores for the test cases A, B, and C, respectively.

Test Case A: Small size problem with 2×128^3 particles calculated with 1 through 256 cores. The initial conditions for the same number of particles have a total size of 65 MB, and the total peak memory needed is ~2.2 GB.

Test Case B: Medium size problem with 2×512^3 particles calculated with 64 through 2048 cores. The initial conditions (with 2×512^3 particles) have a total size of 4.1 GB, and the total peak memory need is ~140 GB.

Test Case C: Large size problem with 2×2048^3 particles calculated with 64 through 2048 cores. The initial conditions (with 2×512^3 particles) have a total size of 257 GB, and the total peak memory needed is ~8.8 TB.

2.5 GPAW

2.5.1 Code Description

GPAW [1] 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 and ScaLAPACK. Parallelisation is based on

message-passing using MPI with no support for multithreading. Development branches for GPGPUs and MICs include support for offloading to accelerators using either CUDA or pyMIC/libxsteam, respectively. GPAW is freely available under the GPL license.

2.5.2 Test Cases

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 10 nodes and/or 100 MPI tasks.

Case M: Copper filament

A ground state calculation for a copper filament in vacuum. By default, uses a 2×2×3 FCC lattice with 71 atoms (freely adjustable) and ScaLAPACK for parallelisation. Expected to scale up to 100 nodes and/or 1000 MPI tasks.

Case L: Silicon cluster

A ground state calculation for a silicon cluster in vacuum. By default, the cluster has a radius of 15Å (freely adjustable) and consists of 702 atoms, and ScaLAPACK is used for parallelisation. Expected to scale up to 1000 nodes and/or 10000 MPI tasks.

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 in support all the different kinds of SIMD support 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.

2.6.2 Test Cases

- 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-4IP projects. It is reported to scale efficiently up to 1000+ cores on x86 based systems.

- Test Case B: Lignocellulose

  A model of cellulose and lignocellulosic biomass in an aqueous solution. This system of 3.3 million atoms is inhomogeneous. This system uses reaction-field electrostatics instead of PME and therefore scales well on x86. This test case was used as the "Large" test case in previous PRACE-2IP-4IP projects. It is reported in previous PRACE projects to scale efficiently up to 10000+ 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 for 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 data sets 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. This data set 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. The system contains roughly 28 million atoms and is expected to scale efficiently up to few tens of thousands x86 cores.

2.8 NEMO

2.8.1 Code Description

NEMO (Nucleus for European Modelling of the Ocean) [22] 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 Format) 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 jp_cfg as follows:

$$Jpiglo = 30 \times jp\_cfg + 2$$

$$Jpjglo = 20 \times jp\_cfg + 2$$

In this configuration, we use a default value of 30 ocean levels, depicted by jpk=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:

Test Case A:

- jp_cfg = 128 suitable up to 1000 cores
- Number of Days: 20
- Number of Time steps: 1440
- Time step size: 20 mins
- Number of seconds per time step: 1200

We performed scalability test on 512 cores and 1024 cores for test case A.

Test Case B:

- jp_cfg = 256 suitable up to 20,000 cores.
- Number of Days (real): 80
- Number of time step: 4320
- Time step size(real): 20 mins
- Number of seconds per time step: 1200

We performed scalability test for 4096 cores, 8192 cores and 16384 cores for test case B.

Both these test cases can give us quite good understanding of node performance and interconnect behaviour. We switch off the generation of mesh files by setting the flag nn_mesh = 0 in the namelist_ref file. Also using_server = false is defined in io_server file.

We report the performance in step time which is the total computational time averaged over the number of time steps for different test cases. This helps us to compare systems in a standard manner across all combinations of system architectures. The other main reason for reporting time per computational time step is to make sure that results are more reproducible and comparable.

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. To measure the step time, we inserted a patch which includes the MPI_wtime() functional call in nemogcn.f90 file for each step which also cumulatively adds the step time until the second last step. We then divide the total cumulative time by the number of time steps to average out any overhead.

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 [45]. 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. In order 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 properties of the sector Hamiltonian matrices are dense, real, and symmetric. For electron-atom or electron-ion calculations (e.g. Test Case 1), 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. Electron-molecule calculations (e.g. Test Case 2) do not produce such fine resonances and therefore require only a single 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, memory and 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 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, Cray Libsci and ARM Performance Libraries 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 cores in a node.

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.

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.

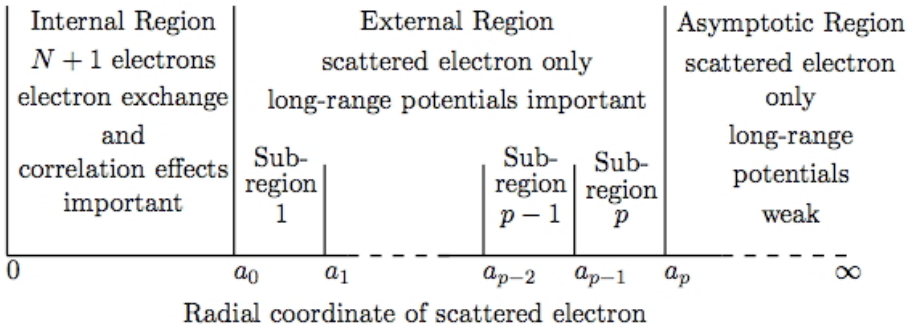


Figure 1: Partitioning of Configuration Space in PFARM

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) in order to maintain accuracy across the sector and vice-versa. Memory limits on the target hardware are also an input parameter that is used for determining a preferred configuration.

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 relevant computational characteristics for this problem are:

- *FINE* mesh calculation: 16 sectors are defined with 22 Legendre functions in the sector Hamiltonian assembly, leading to sector matrices of dimension 25982.
- *COARSE* mesh calculation: 16 sectors are defined with 10 Legendre functions in the sector Hamiltonian assembly, leading to sector matrices of dimension 11810.

Test Case 2 (Molecular)

This dataset is an electron-molecule scattering case with 1361 channels calculating electron scattering with Methane. No fine mesh is required, so a single mesh is defined. The relevant computational characteristics of this problem are:

- *SINGLE* mesh calculation: 64 sectors are defined with 10 Legendre functions in the sector Hamiltonian assembly, leading to sector matrices of dimension 13610.

Due to the greater number of sectors in Test Case 2, this dataset will scale to a higher number of nodes.

2.10 QCD

2.10.1 Code Description

The QCD benchmark is, unlike the other benchmarks in the PRACE application benchmark suite, not a full application but a set of 5 kernels which are representative of some of the most compute-intensive parts of QCD calculations.

The benchmark kernels of the QCD UEABS are based on codes widely used by the lattice QCD community. Namely, the 5 different QCD kernels in UEABS, taken from software packages of major European QCD collaborations, were extended in the PRACE-4IP project to include kernels capable of using accelerators. In this deliverable, 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. [4]). 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. [5]). Part 2 includes kernels from the library QPhiX[2], optimised for Intel architectures such as Skylake and KNL Xeon Phi cards, and QUDA[3], for NVIDIA GPUs. In addition, we use the software package GRID[42][43] for obtaining performance result on the Mont-Blanc 3 system Dibona, making use of its ARM optimised kernel.

In all cases, the benchmark kernels repeatedly apply the so-called Wilson Dirac operator on an iteratively updated vector. For all cases, with the exception of GRID, 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. For the case of GRID, the benchmark kernel used includes only the operator application, i.e. the other linear algebra operations and reductions are not included. 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.

2.10.2 Test Cases

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 (QE) 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. For the benchmarking activity we opted to test the most commonly used package in the suite, PWscf, which performs plane wave self-consistent field calculations (e.g. for calculating the ground state energy or structure optimisations).

The program has been written mainly in Fortran 90, and parallelised with both MPI and OpenMP in a sophisticated multiple communicator scheme. The result is a highly optimised application where the performance is strongly dictated by the linear algebra implementation. Memory requirements are directly related to the input size while the I/O load is usually negligible.

For the CPU-based calculations we used versions 6.x, with x=1–3, depending on availability on the host architecture. The differences between these minor versions are mainly functional improvements or bugfixes so we do not expect significant performance variations. For the GPU

architectures, we used the 6.3 CUDA Fortran version recently developed by F. Spiga. For those systems where QE was not available, or not present at a sufficiently high version, the package was compiled from the source code.

2.11.2 Test Cases

Small

The small dataset is based on a standard benchmark called AUSURF and consists of the optimisation of a surface composed of 112 gold atoms. The optimisation threshold is set such that convergence is normally achieved in exactly 21 iterations. On most x86 systems the benchmark scales up to about 200 MPI tasks. Notice that since the structure presents 2 k-points the QE -npool parameter should be set to 2. We did not use OpenMP threads for this input, except for the GPU runs.

Large

The large benchmark is based on a structure of tantalum oxide and is labelled as TA2O5. Like AUSURF this also performs a geometry optimisation but CPU and memory requirements are much higher. For this reason, we also used OpenMP threads. The system exhibits 26 k-points so the -npool parameter should be 13 or 26 or multiples thereof.

2.12 SHOC

2.12.1 Code Description

The Benchmark Suite also includes a series of synthetic benchmarks. For this purpose, we choose the Scalable HeterOgeneous Computing (SHOC) benchmark suite, augmented with a series of benchmark examples developed internally. SHOC is a collection of benchmark programs testing the performance and stability of systems using computing devices with non-traditional architectures for general purpose computing. Its initial focus is on systems containing GPU and multi-core processors, and on the OpenCL programming standard, but CUDA and OpenACC versions were added. Moreover, a subset of the benchmarks is optimised for the Intel Xeon Phi coprocessor.

The SHOC benchmark suite currently contains benchmark programs categorised by complexity. Some measure low-level ‘feeds and speeds’ behaviour (Level 0), some measure the performance of a higher-level operation such as a Fast Fourier Transform (FFT) (Level 1), and the others measure real application kernels (Level 2).

The SHOC benchmark suite has been selected to evaluate the performance of accelerators (GPUs) on synthetic benchmarks, mostly because SHOC provides CUDA/OpenCL/Offload/OpenACC variants of the benchmarks. This allowed us to evaluate NVIDIA GPUs (with CUDA/OpenCL/OpenACC) and Intel Xeon Phi KNC (with both Offload and OpenCL). However, on the latest Xeon Phi processor (codenamed KNL) none of these 4 models is supported. Thus, benchmarks on the KNL architecture cannot be run at this point, and there isn't any news of Intel supporting OpenCL on the KNL.

Code implementation description

Offloading for accelerators is implemented through CUDA and OpenCL for GPU and through OpenMP for MIC (Intel Xeon Phi). For selected benchmarks OpenACC implementations are provided for GPU.

SHOC is written in C++ and is open-source and freely available.

2.12.2 Test Cases

The benchmarks contained in SHOC currently feature 4 different sizes for increasingly large systems. The size convention is as follows:

1. CPU / debugging
2. Mobile/integrated GPU
3. Discrete GPU (e.g. GeForce or Radeon series)
4. HPC-focused or large memory GPU (e.g. Tesla or FireStream Series)

2.13 SPECFEM3D

2.13.1 Code Description

The software package SPECFEM3D_GLOBE [7] 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, 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. 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.

It has very good accuracy and convergence properties. It is also very well suited to parallel implementation on very large supercomputers as well as on clusters with GPU accelerating graphics cards. All SPECFEM3D_GLOBE software is written in Fortran 2003 with full portability in mind, and conforms strictly to the Fortran 2003 standard. It uses no obsolete or obsolescent features of Fortran. The package uses parallel programming based upon the Message Passing Interface (MPI). The package includes support for GPU graphics card acceleration and also supports OpenCL.

2.13.2 Test Cases

Test Case A is designed to run on Tier-1 sized systems (up to around 1,000 x86 cores, or equivalent), Test Case B is designed to run on Tier-0 sized systems (up to around 10,000 x86 cores,

or equivalent) and finally the test case C is designed to run on PCP prototypes (up to around 100 cores, or equivalent).

The test cases simulate the earthquake of June 1994 in Northern Bolivia at a global scale with the global shear-wave speed model named s362ani. The solver calculates seismograms for 129 stations, and simulations are run for a record length of 3 minutes 30 for test case A, 10 minutes for test case B and one minute for test case C.

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 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.

Test case A runs with 96 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=4 and NEX_XI=384.

Test Case B runs with 1536 MPI tasks using hybrid parallelisation and has the following mesh characteristics: NCHUNKS=6, NPROC_XI=16 and NEX_XI=384.

Test Case C B runs with 6 MPI tasks using hybrid parallelisation and has the following mesh characteristics: NCHUNKS=6, NPROC_XI=1 and NEX_XI=64.

3 Benchmark Systems

3.1 Tier-0 systems

PRACE hosting members Tier-0 systems:

3.1.1 Hazel Hen

Hazel Hen [26] is the Tier-0 system hosted by HLRS, Germany. Hazel Hen is a Cray XC40 system composed of 7712 compute nodes with a total of 185,088 Intel Haswell E5-2680 v3 compute cores. Hazel Hen has an Aries [23] interconnect and features 965 TB of Main Memory and a total of 11 PB of storage. The peak performance is 7.42 Pflop/s.

3.1.2 Irene

Irene (Joliot-Curie computer) [27] is the Tier-0 system hosted by CEA, France. Irene is a Bull Sequana X1000 supercomputer, and has 2 compute partitions:

- Irene SKL:

  - 1,656 Intel Xeon Platinum 8168 (Skylake) dual processors nodes @ 2.7 GHz, 24 cores/CPU,
  - 79,488 compute cores for 6.86 Pflop/s peak performance,
  - 192 GB of DDR4 memory / node,
  - InfiniBand EDR interconnect.
- Irene KNL:
  - 666 many cores Intel Xeon Phi 7250 (Knights Landing) nodes @ 1.4 GHz, 68 cores/CPU,
  - 45,288 compute cores for 2 Pflop/s peak performance,
  - 96 GB DDR4 memory/node + 16 GB MCDRAM memory/node,
  - Bull eXascale Interconnect (BXI).

3.1.3 JUWELS

JUWELS, the Jülich Wizard for European Leadership Science [28], is the Tier-0 system hosted by the Jülich Supercomputing Centre, Germany. JUWELS is an Atos Bull Sequana X1000 system with dual 24-core Intel Xeon Platinum 8168 (Skylake) CPUs @ 2.7 GHz and an EDR-InfiniBand. The peak performance is 9.89 Pflop/s.

3.1.4 Marconi

Marconi [29] is the Tier-0 system, co-designed by CINECA and based on the Lenovo NeXtScale platform, which replaced the previous IBM BlueGene/Q system (Fermi) in June 2016. The supercomputer has been upgraded by adding new partitions with the result that as of March 2019 there are two partitions:

1. A2 containing 3,600 nodes with 1 × 68-cores Intel Xeon Phi 7250 (Knights Landing) @ 1.4 GHz processor per node
2. A3 composed of ca. 3,000 nodes with 2 × 24-cores Intel Xeon Platinum 8160 @ 2.1 GHz (Skylake) processors per node.

The A2 partition, available as part of CINECA's PRACE Tier-0 offer, has a peak performance of 11 Pflop/s while the A3 partition has a peak performance of ca. 8 Pflop/s. The network fabric is based on the Intel Omni-Path architecture while data storage is provided by the IBM Spectrum Scale™ (GPFS) file system.

3.1.5 MareNostrum4

MareNostrum4 [30] 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 6.2272 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 1.880 GB/core (216 nodes high memory, 10368 cores with 7.928 GB/core)
- 100 Gbit/s Intel Omni-Path HFI Silicon 100 Series PCI-E adapter (in a full fat tree topology)
- 10 Gbit Ethernet
- 200 GB local SSD available as temporary storage during jobs

3.1.6 Piz Daint

Piz Daint [31] 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) [50].

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 [50].

3.1.7 SuperMUC-NG

SuperMUC-NG [32] is the Tier-0 system hosted by LRZ, Germany. SuperMUC-NG is a Lenovo system:

- 6336 Thin compute nodes each with 48 cores Intel Xeon Platinum 8174 (Skylake) @ 3.1 GHz and 96 GB memory
- 144 Fat compute node each with 48 cores Intel Xeon Platinum 8174 (Skylake) @ 3.1 GHz and 768 GB memory

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).

Unfortunately, SuperMUC-NG will become available only on 1 April 2019. This is too late to be able to include results in this report.

3.2 PCP prototypes

3.2.1 DAVIDE

D.A.V.I.D.E. (Development of an Added Value Infrastructure Designed in Europe) [33] is an energy-aware petaflops Class High Performance Cluster based on the IBM POWER Architecture coupled with NVIDIA Tesla Pascal GPUs (P100) using NVLink. The innovative design of DAVIDE has been developed by E4 Computer Engineering for PRACE, with the aim of producing a leading edge HPC cluster showing higher performance, reduced power consumption and ease of use.

DAVIDE entered the TOP500 and Green500 list in June 2017 in its air-cooled version, while the current version features liquid cooling and an innovative technology for monitoring and capping the power consumption.

The current configuration consists of 45 × (2 POWER8 + 4 Tesla P100) nodes with NVLink and coupled with an InfiniBand (2×IB EDR) network. The peak performance is approximately 1 Pflop/s.

3.2.2 Frioul

The Frioul PCP prototype [34] was designed by Atos/Bull and has been hosted by CINES, France since 2016. Jointly financed by PRACE (PCP) and GENCI (Frioul), part of the PCP machine was dismantled before being shipped to the TGCC; the other part was merged into Frioul to become a single homogeneous machine. The Frioul configuration has changed from 48 to 54 compute nodes and has changed the file system (MooseFS I/O to BeeGFS I/O). Since late 2018, the energy measurement tools (BEO & HDEEVIZ) initially present on the PRACE PCP prototype are no longer available on Frioul. It is now made of 18 Bull Sequana X1210 blades, each including 3 Intel Xeon Phi KNL nodes. In total, it has a theoretical peak performance of 172 Tflop/s.

- 54 nodes with
  - 1× Intel Xeon Phi 7250 processor (KNL), 68 cores at 1.4 GHz with SMT4
  - 16 GB MCDRAM, 192 GB DDR4 DIMMs
- InfiniBand EDR
- BeeGFS I/O

3.2.3 JUMAX

The JUMAX PCP prototype [46] is based on Maxeler's MAX5 DataFlow Engines (DFE) cards. A DFE card comprises an FPGA, which is used to implement part of the application following a data-flow paradigm. JUMAX is hosted by JSC, Germany. Since porting UEABS applications to JUMAX was completely out of scope of the PRACE-5IP benchmarking activities, we do not present any results on JUMAX. JUMAX results will be presented in the PCP Conclusion Report [53].

3.3 Partner prototype systems

3.3.1 DEEP-ER SDV

The DEEP-ER Prototype SDV [35], Software Development Vehicle, is hosted by JSC, Germany

Cluster

- 16 dual-socket Intel Xeon E5-2680v3 nodes
- Each node: 128 GB DRAM, 400 GB NVM

Booster

- 8 Adams Pass Intel Xeon Phi 7210 CPU
- Each node: 16 GB on-package memory, 96 GB DRAM, 200 GB NVM

System

- EXTOLL fabric using TOURMALET NICs with six links of 100 Gbit/s each
- Aggregate performance approx. 40 Tflop/s

Storage

- 2 storage servers (spinning disks, 57 TB)
- 1 metadata server (SSDs)
- BeeGFS file system

3.3.2 Mont-Blanc 3 Dibona

The Mont-Blanc 3 prototype Dibona [36] is hosted by Atos, France. Dibona is a Bull Sequana X1000 system. It has 48 nodes, each node includes:

- Dual socket Marvell ThunderX2 (32 cores per CPU, 64 cores per node, each core at 2 GHz, 32 MB L3 cache)
- 256 GB of main memory per node (16 DDR4-2666 DIMM slots, 8 channels per CPU)
- 256 GB local storage e (+ 8 TB NFS)

The Mont-Blanc 3 prototype Dibona is equipped with a fat-tree with a pruning factor of 1:2 at L1 level interconnect topology with InfiniBand EDR 100 Gb/s.

4 Benchmark Results per Application

4.1 Alya

The Alya benchmarks have been performed on systems with different architectures, Skylake (JUWELS and MareNostrum4), Haswell (DEEP-ER SDV), KNL (Marconi, Frioul), GPU (Piz Daint, DAVIDE) and ARM (Dibona).

Due to the similarities with other architectures tested, benchmarks were not performed on the Hazel Hen, Irene-KNL and Irene Skylake systems.

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.

We have only used Test Case B on the Tier-0 systems since its size is too large for the smaller systems. Conversely, we have only used Test Case C on the PCP Prototypes since its size is too small for the larger systems.

4.1.1 Performance on Skylake: JUWELS and MareNostrum4

Table 1 and Table 2 present the results for the Skylake systems, JUWELS and MareNostrum4 for Case A and B respectively. We can observe better performance on JUWELS for all the cases because the CPU frequency on JUWELS (2.7 GHz) is higher than on MareNostrum4 (2.1 GHz).

Additionally, the parallel efficiency on all the cases is better than for MareNostrum4. One of the causes of this difference on the efficiency is the network of each system, Mellanox EDR-InfiniBand on JUWELS and Intel Omni-Path on MareNostrum4.

Number of cores	Time (s)	SpeedUp	Efficiency	Time (s)	SpeedUp	Efficiency
	JUWELS			MareNostrum4		
192	124.24	1.0	100%	129.45	1.0	100%
384	62.56	2.0	99%	67.45	1.9	96%
768	31.24	4.0	99%	33.93	3.8	95%
1536	16.45	7.6	94%	18.28	7.1	89%

Table 1: Test Case A – Skylake

Number of cores	Time (s)	SpeedUp	Efficiency	Time (s)	SpeedUp	Efficiency
	JUWELS			MareNostrum4		
1152	372.52	1.0	100%	451.38	1.0	100%
2304	196.48	1.9	95%	262.32	1.7	86%
4608	99.65	3.7	93%	124.98	3.6	90%
9216	61.34	6.1	76%	86.61	5.2	65%

Table 2: Test Case B – Skylake

4.1.2 Performance on Marconi-KNL

Table 3 and Table 4 show the performance results of Alya on Marconi-KNL for Test Case A and Test Case B. We observe that for the same number of nodes (68 cores is one node on Marconi-KNL and 48 cores is one node in Skylake systems) the performance is similar, despite of KNL has

more cores per node. Also, the parallel efficiency is worse on KNL due to each node has more MPI tasks, so each node has more communication. We obtained the best performance using the Quadrant mode for the Clustering settings and the Cache mode for the MCDRAM [54]. The performance results of Table 3 and Table 4 were obtained using these modes.

Number of cores	Time (s)	SpeedUp	Efficiency
272	130.66	1.0	100%
544	68.31	1.9	96%
1088	37.94	3.4	86%
2176	22.78	5.7	72%

Table 3: Test Case A – Marconi-KNL

Number of cores	Time (s)	SpeedUp	Efficiency
2176	486.54	1.0	100%
4352	281.05	1.7	87%
8704	157.55	3.1	77%
13056	122.24	4.0	66%

Table 4: Test Case B – Marconi-KNL

4.1.3 Performance on GPU: Piz Daint

Table 5 and Table 6 show the performance results obtained in the executions in Piz Daint of cases A and B respectively. Alya's code was compiled using the compilers PGI 18.5 for using OpenACC in the assembly of the matrix system, and CUDA 9.2 in the linear solver of the Poisson equation.

The parallel efficiency for the case A is 84% when increasing the number of nodes 8 times from the initial setting. As expected, the strong scalability is lower than in the CPU runs, because reducing the local workload negatively affects the GPU occupancy, resulting in a slowdown in the performance. Moreover, the relative weight of the MPI communications increases when using a larger number of nodes. Consequently, the overlapping strategy used in the GPU implementation reduces its efficiency for hiding the communications.

This behaviour is observed more intensively in Table 6 In such case, the initial setting has half of the local workload than case A, and therefore, the parallel efficiency slows down 11%. Despite the decrease in parallel performance, the GPU still runs in average 2.5 times faster than the pure CPU implementation.

Number of nodes	Time (s)	SpeedUp	Efficiency
8	123.01	1.0	100%
16	64.49	1.91	95%
32	34.49	3.79	95%
64	18.28	6.73	84%

Table 5: Test Case A – Piz Daint

Number of nodes	Time (s)	SpeedUp	Efficiency
128	67.01	1.0	100%
256	34.81	1.93	96%
512	20.47	3.27	82%
1024	11.54	5.81	73%

Table 6: Test Case B – Piz Daint

4.1.4 Performance and Energy Consumption on PCP prototypes

As we can see on Table 7 and Table 8, Frioul is two times slower than DAVIDE in a node to node comparison of absolute times. However, note that the last ones are composed of 4 GPUs and 2 POWER8 CPUs. Roughly speaking, we could say that currently for Alya the execution in an Intel Xeon Phi 7250 is as fast as two NVIDIA P100 GPUs. For the strong speedup test, an ideal acceleration and a linear increase of energy cost would result in a constant energy cost per job. Both conditions are not true in practice on the PCP prototypes. Energy consumption grows between 1.4 and 2.6 times, this increase being more notorious for the DAVIDE system. The executions on Frioul are 3.7 times more energy efficient than the ones on DAVIDE system. However, note that on the energy measurements for DAVIDE are also considered the POWER8 hosts that are not in Alya's calculations, but only to carry out intra-node communications.

Number of full nodes	Performance	Energy	Speed-Up	Performance Efficiency	Normalised energy
4	92.80	898222	1.0	100%	1.0
8	47.83	1441304	1.9	97%	1.6
16	26.65	1722932	3.5	87%	1.9
32	14.77	1821975	6.3	79%	2.0

Table 7: Test Case C – PCP prototype DAVIDE

Number of full nodes	Performance	Energy	Speed-Up	Performance Efficiency	Normalised energy
4	220.07	332128	1.0	100%	1.0
8	108.18	389952	2.0	102%	1.2
16	55.81	380000	3.9	99%	1.1
32	28.68	470000	7.7	96%	1.4

Table 8: Test Case C – PCP prototype Frioul

4.1.5 Performance on DEEP-ER SDV

In Table 9 we observe that the Alya scalability on the DEEP-ER SDV prototype slows down in 40% with respect to MareNostrum4 with 16 nodes on each system. This result is expected due the network differences. In terms of CPU performance, ignoring the communication load, it is observed that MareNostrum4 is 50% faster than the DEEP-ER SDV prototype.

Number of cores	Time (s)	SpeedUp	Efficiency
48	772.64	1.0	100%
96	421.31	1.8	92%
192	278.47	2.8	69%
384	176.28	4.4	55%

Table 9: Test Case A – DEEP-ER SDV

4.1.6 Performance on ARM: Mont-Blanc 3 Dibona

Alya's code was compiled on Mont-Blanc 3 Dibona using GCC 7.2.1 and OpenMPI 3.1.2. We see from Table 10 that the performance of Alya on Mont-Blanc 3 Dibona is slower if we compare the ARM system node to node, with the Skylake systems. However, the scalability in Dibona is on the expected range, if we compare with a system with InfiniBand network, for example, JUWELS.

Number of cores	Time (s)	SpeedUp	Efficiency
128	1202.7	1.0	100%
256	658.85	1.8	91%
512	332.72	3.6	90%
1024	182.57	6.6	82%

Table 10: Test Case A – Mont-Blanc 3 Dibona

4.2 Code_Saturne

The tests have been carried out on: 4 machines, using CPUs only, namely Hazel Hen, Irene-SKL, JUWELS and MareNostrum4; on 3 machines with KNLs, Irene-KNL, Frioul and Marconi; and 3 extra machines, Piz Daint (CPUs and GPUs), Dibona (ARM) and DAVIDE (POWER8). Two series of benchmark tests have been run, for 10 time steps each, with 13M cells (Test Case A) and 111M cells (Test Case B), respectively, and the time per time step and efficiency were used for comparison. It was decided to stick to efficiency and not present also the speed-up behaviour, as the efficiency shows more insight than the speed-up, the speed-up can easily be derived from the efficiency, and finally for a practical reason, to keep a large-enough font size in the tables below. The partitioning was performed on-the-fly using a Space Filling Curve algorithm which relies on Morton's configuration.

4.2.1 Performance on CPU-based machines: Hazel Hen, Irene-SKL, JUWELS and MareNostrum4

MPI only is used on all these 4 machines, without any hyperthreading at all, as it was found that there was no benefit to have it on. Consequently, Irene-SKL, JUWELS and MareNostrum4 have been used with 48 MPI tasks per node, whereas Hazel Hen has been used with 24 MPI tasks per node.

Table 11 shows time per time step and efficiency for each of the Test Case A simulations. Nearly perfect efficiency is observed on the four machines up to 16 nodes of Hazel Hen and 8 nodes of the 3 other machines (384 MPI tasks in total). Beyond these node counts, the performance deteriorates, which can be explained by the fact that either each core is not loaded enough (about 17,000 cells per MPI tasks when 768 cores are used) or MPI communications are taking over one-task computation. This alteration in performance could also be explained by the fact that the Space Filling Curve partitioner does not provide an optimised edge-cut which results in an increase in the number of MPI tasks involved in point-to-point communications.

Among the three Tier-0 machines, both Skylake-based computers show best results, with JUWELS giving the best performance up to 16 nodes, in terms of timing and efficiency. On MareNostrum4, Code_Saturne is slightly slower than on Skylake-based computers, which might be explained by the higher CPU frequency of the latter (2.1 GHz vs 2.7 GHz). The node-to-node comparison shows that the simulations ran on JUWELS are always at least twice as fast as on Hazel Hen, which was to be expected given the type of processors used on Hazel Hen (Haswell).

#nodes	Hazel Hen			Irene-SKL			JUWELS			MareNostrum4		
	#cores	T(s)	Eff	#cores	T(s)	Eff	#cores	T(s)	Eff	#cores	T(s)	Eff
1	24	50.40	100%	48	23.98	100%	48	22.60	100%	48	29.02	100%
2	48	25.24	100%	96	12.05	100%	96	11.09	102%	96	13.63	106%

#nodes	Hazel Hen			Irene-SKL			JUWELS			MareNostrum4		
	#cores	T(s)	Eff	#cores	T(s)	Eff	#cores	T(s)	Eff	#cores	T(s)	Eff
4	96	12.72	99%	192	6.21	96%	192	5.32	106%	192	6.78	107%
8	192	6.08	104%	384	3.33	90%	384	2.69	105%	384	3.46	105%
16	384	3.17	99%	768	2.22	68%	768	1.52	93%	768	2.18	83%
32	768	2.01	78%									

Table 11: Test Case A: Performance of Code_Saturne on the 4 CPU-based machines

#nodes	Hazel Hen			Irene-SKL			JUWELS			MareNostrum4		
	#cores	T(s)	Eff	#cores	T(s)	Eff	#cores	T(s)	Eff	#cores	T(s)	Eff
8	192	81.42	100%	384	38.75	100%	384	34.52	100%	384	45.41	100%
16	384	39.91	102%	768	20.50	95%	768	17.79	97%	768	24.99	91%
32	768	22.87	89%	1,536	12.01	81%	1536	9.54	90%	1,536	12.99	87%
64	1,536	11.33	89%	3,072	7.40	65%	3072	6.69	65%	3,072	8.96	63%
128	3,072	8.53	60%									

Table 12: Test Case B: Performance of Code_Saturne on the 4 CPU-based machines

Table 12 presents Code_Saturne's timings and efficiency for Test Case B. The same trend as for Test Case A is observed, e.g. the fastest simulations are on JUWELS, being at least twice as fast as on Hazel Hen.

4.2.2 Performance on KNL-based machines: Irene-KNL, Frioul and Marconi

Three KNL-based machines are used to assess the performance of Code_Saturne. All of them rely on 68 thread-nodes. For Test Case A, it was found that simulations using 34 MPI tasks and 2 OpenMP threads per node were faster than simulations using 68 MPI tasks per node from 4 nodes on (these comparisons are not shown here), on Irene-KNL and Frioul, the best timing being obtained on Frioul (4.23 s) using 32 nodes, even if the best efficiency was achieved on Irene-KNL. However, using MPI only on Marconi with 68 MPI tasks per node (as opposed to 34 MPI tasks and 2 OpenMP threads per node simulation, which results are not shown here) exhibited the best timing and performance on that machine.

#nodes	Irene-KNL			Frioul			Marconi		
	#threads	T(s)	Eff	#threads	T(s)	Eff	#cores	T(s)	Eff
1	68	63.15	100%	68	47.74	100%	68	43.85	100%
2	136	33.75	100%	136	24.75	96%	136	23.89	92%
4	272	18.58	99%	272	13.32	90%	272	12.76	86%
8	544	8.30	104%	544	8.75	68%	544	7.52	73%
16	1,088	7.07	99%	1,088	5.70	52%	1,088	6.10	45%
32	2,176	5.10	78%	2,176	4.23	35%			

Table 13: Test Case A: Performance of Code_Saturne on the 3 KNL-based machines (2 OpenMP threads per MPI task are used on Irene-KNL and Frioul and MPI only on Marconi)

For Test Case B interestingly (see Table 14), using MPI only on the full nodes of Marconi shows better performance than combinations of 34 MPI tasks and 2 OpenMP threads per on Irene-KNL.

#nodes	Irene-KNL			Marconi		
	#threads	T(s)	Eff	#threads	T(s)	Eff
8	544	98.00	100%	544	75.31	100%
16	1,088	56.13	87%	1,088	40.47	93%

#nodes	Irene-KNL			Marconi		
	#threads	T(s)	Eff	#threads	T(s)	Eff
32	2,176	32.68	75%	2,176	22.67	83%
64	4,352	23.70	52%	4,352	13.90	68%
128	8,704	13.84	44%	8,704	11.54	41%

Table 14: Test Case B: Performance of Code_Saturne on Irene-KNL (2 OpenMP threads per MPI task) and Marconi (MPI only)

4.2.3 Performance on other architectures: Piz Daint, Dibona, DAVIDE

Many simulations have been carried out to select the best combination of MPI tasks, OpenMP threads, and GPUs (when available) on Piz Daint, Dibona and DAVIDE. Table 15 shows that the best timings obtained for Test Case A using Piz Daint are for 8 MPI tasks and 2 OpenMP threads and 1 GPU per node, Dibona for 32 MPI tasks and 2 OpenMP threads per node and DAVIDE for 16 MPI tasks and 4 OpenMP threads and 4 GPUs per node. If we consider former tests carried out for Code_Saturne on POWER8/9 machines, each GPU requires at least 300,000 cells to demonstrate good performance (see [44]). Given the relatively modest size of the test case (13M cells) and the 300,000-cell threshold, a meaningful comparison is carried out up to 4 nodes for the 3 machines. In this case, the simulations ran on DAVIDE are the fastest of the 2 GPU machines, by at least a factor of about 2. Simulations on Dibona are slightly slower than on DAVIDE, but the compute time per time step is still good (37.86 s vs 30.03 s, 21.28 s vs 16.75 s, and 12.13 s vs 8.52 s using 1, 2 and 4 nodes of Dibona and DAVIDE, respectively). The simulations on Dibona are clearly faster than on Piz Daint up to 16 nodes. Note that it was not possible to run simulations using 32 nodes and more on Dibona.

#nodes	Piz Daint			Dibona			DAVIDE		
	#threads	T(s)	Eff	#threads	T(s)	Eff	#threads	T(s)	Eff
1	16	80.25	100%	64	37.86	100%	64	30.03	100%
2	32	41.66	96%	128	21.28	89%	128	16.75	90%
4	64	22.29	90%	256	12.13	78%	256	8.52	88%
6							384	6.70	74%
8	128	12.60	80%	512	8.55	55%			
16	256	7.79	64%	1,024	6.40	37%			
32	512	5.08	49%						
64	1,024	3.55	35%						
128	2,048	2.19	28%						

Table 15: Test Case A: Performance of Code_Saturne on the 3 extra machines (Piz Daint, 8 MPI tasks and 2 OpenMP threads and 1 GPU per node, Dibona, 32 MPI tasks and 2 OpenMP threads per node, DAVIDE, 16 MPI tasks and 4 OpenMP threads and 4 GPUs per node)

#nodes	Piz Daint		
	#threads	T(s)	Eff
8	128	119.65	100%
16	256	63.17	95%
32	512	40.60	74%
64	1,024	24.07	62%
128	2,048	15.36	49%
256	4,096	10.32	36%
512	8,196	8.25	23%

Table 16: Test Case B: Performance of Code_Saturne on Piz Daint (8 MPI tasks and 2 OpenMP threads and 1 GPU per node)

Table 16 shows the timings obtained for Test Case B on Piz Daint. The scalability is very good up to 32 nodes, and some speed-up is still observed up to 512 nodes.

4.2.4 Cross comparison for all the machines/architectures

For Test Case A, the best timings and performance are observed on JUWELS, a time step taking about 1.52 s on 16 nodes and 768 cores, being about 3.75 times faster than the fastest simulations on a KNL-machine (Frioul), about 4.21 times faster than the simulation on the ARM cluster (Dibona) and about 5.125 times faster than the simulation on the GPU-based machine (Piz Daint). This might be explained by the very high speed of the Skylake nodes, but also because of the nature of the problem to solve in Code_Saturne, where the Navier-Stokes equations are solved implicitly using sparse matrices (sparse linear algebra). However, the limitation in terms of number of available nodes on DAVIDE (6 nodes) and the fact that the processors are now a relatively old technology (POWER8) do not help making a state-of-the-art comparison. It would be good in the future to compare the results on JUWELS to results obtained on a POWER9 machine with NVLINK 2.0 enabled.

For Test Case B, less data were available, because of the limitation in the number of available nodes on some of the machines (Frioul, Dibona and DAVIDE, for instance). Once again, JUWELS is the machine were the simulations run the fastest.

4.2.5 Energy consumption

The energy consumption is computed on DAVIDE (Test Case A) and Piz Daint (Test Case A and B). It was not possible to get the energy consumption on Frioul as the tools to compute it were no longer available on the machine, when the MPI/OpenMP/GPU version of the code used in this work was made available in late 2018.

The results presented for DAVIDE (CPU), (see 1st column of Table 17), were run using 16 MPI tasks and 2 OpenMP threads per node, and for DAVIDE (GPU), (see 2nd column of Table 17), using 16 MPI tasks, 2 OpenMP threads and 4 GPUs per node. Not enough nodes were available to run meaningful tests for Test Case B. Consequently, Test Case A results only are presented on that machine. Note as well that the results are presented for a different configuration than the ones shown in Section 4.2.2, but for cases (16 MPI tasks and 2 OpenMP threads per node) where the energy measurements are available and happen to be consistent.

The results obtained for Test Case A on Piz Daint (CPU), (see 3rd column of Table 17) have been obtained using 12 MPI tasks per node, and on Piz Daint (GPU), (see 4th column of Table 17) using 8 MPI tasks, 2 OpenMP threads and 1 GPU per node. The same conditions are used for the results shown in Table 18 and Test Case B.

For Test Case A, the energy consumed on DAVIDE (CPU) is larger than for the three other configurations. It also shows a decreasing trend, when the number of nodes increases, which is the opposite of the three other cases. This large energy spent could be explained by the fact that the time per time step is about twice as large as for DAVIDE (CPU) for two and four nodes, and the

simulations then last longer. The simulations on DAVIDE (GPU) are much faster than on Piz Daint (CPU and GPU) on two and four nodes, however, the energy consumption is very similar in these three cases (225 kJ, 215 kJ, 208 kJ respectively on 4 nodes), making DAVIDE the best machine for ratio time to solution/energy consumption.

If the number of nodes is increased (Piz Daint CPU and GPU), so does the energy consumption, but it remains very similar for both configurations.

#nodes	DAVIDE (CPU)			DAVIDE (GPU)			Piz Daint (CPU)			Piz Daint (GPU)		
	#threads	T(s)	E(kJ)	#threads	T(s)	E(kJ)	#cores	T(s)	E(kJ)	#threads	T(s)	E(kJ)
1	16	61.36		16	32.66		12	102.63	214	16	80.26	170
2	32	29.84	513	32	17.87	194	24	50.10	210	32	41.66	181
4	64	14.53	406	64	9.01	225	48	24.36	215	64	22.29	208
6	96	9.73	363	96	6.85	344						
8							96	11.78	224	128	12.60	239
16							192	6.63	282	256	7.79	315
32							384	4.09	400	512	5.08	465
64							768	3.44	763	1,024	3.55	772
128							1,536	3.08	1,550	2,048	2.19	1,230

Table 17: Test Case A: Energy consumption comparison between DAVIDE and Piz Daint, using CPU or GPU configurations

#nodes	Piz Daint (CPU)			Piz Daint (GPU)		
	#cores	T(s)	E(kJ)	#threads	T(s)	E(kJ)
8	96	144.89	2,450	64	119.65	2,180
16	192	72.82	2,530	128	63.17	2,390
32	384	36.62	2,710	256	40.60	3,000
64	768	19.99	3,330	512	24.07	3,830
128	1,536	11.71	4,710	1,024	15.36	5,150
256	3,072	9.04	7,850	2,048	10.32	8,040
512	6,144	7.33	15,810	4,096	8.25	14,330

Table 18: Test Case B: Energy consumption on Piz Daint, CPUs and GPUs

For Test Case B on Piz Daint (CPU and GPU), the same trend is observed, e.g. using GPUs does not seem to impact the total energy, as it is smaller on 8 and 16 nodes, when the time per time step is smaller than for the CPU only cases. From 32 to 256 nodes, using GPUs is marginally more energy consuming, mainly because the simulations last then longer. The only exception is for 512 nodes, when the GPU simulation uses less energy, and is slower than the CPU simulation.

4.3 CP2K

This section reports CP2K benchmarking results, analyses performance and energy efficiency for the three test cases described in Section 2.3, which have been run on a number of systems including PRACE Tier-0, PRACE PCP prototype, Mont-Blanc 3 and DEEP-ER project prototype machines. The porting effort consisting of the installation and execution work done as part of the CP2K benchmarking effort is also reported on.

With regards to the performance and energy efficiency of CP2K a comparison is made for two of the three test cases between the PCP prototype systems and one of the Tier-0 systems that also provides energy measurements (Piz Daint).

4.3.1 General remarks regarding installation

In addition to the strict requirements for LAPACK, BLAS and ScaLAPACK (often satisfied by an MKL installation or in the case of Dibona by the ARM Performance Libraries), FFTW3, and an MPI library in order to run in parallel, CP2K can make use of several other libraries to improve performance and extend functionality. Some of these, especially the autotuning library libgrid which computes products of Gaussians as well as the small matrix-matrix multiplication libraries libsmm/libxsmm can take very significant effort to build, especially on systems with unusual configurations and pre-production prototype systems with limited documentation and software stacks. For this reason and to ensure that performance and energy comparisons were nonetheless done on a fair footing between the various systems, on each system CP2K was linked only to the Libint (version 1.1.4) and Libxc (usually version 4.2.3) libraries required to run the benchmarks, in addition to the strictly required libraries. Libint and Libxc were therefore built first on each system, and in some cases also FFTW (version 3.3.x) and ScaLAPACK (version 2.0.2) where these were not present already.

Gfortran is the (strongly) recommended compiler for CP2K, and CP2K is known to generate a range of compile-time and run-time issues in conjunction with many different particular major and minor versions of the Intel compiler – see [47] for a partial listing of known issues. Wherever possible CP2K was therefore built using gfortran. In addition, CP2K needs to be linked to a version of an MPI library that was built using the same compiler as used to build CP2K itself. The same is advisable for other libraries. In several cases these requirements placed significant barriers to porting CP2K, e.g. to the prototype systems given their limited software stacks and on which building an MPI library with the desired compiler would be especially problematic given novel interconnects and/or platform configurations.

For execution on GPUs, CP2K was compiled using the flags `-D__ACC` and `-D__DBCSR_ACC` to offload sparse matrix multiplication.

4.3.2 General remarks regarding execution

To begin, hybrid MPI+OpenMP runs for fixed core count for a range of different combinations of threads per rank and ranks per node were performed for small and larger node counts in order to determine the combination giving optimal performance for each test case prior to runs exploring strong scaling for that test case.

There is no reading of any large amount of input data for any of the test cases hence initialisation cost is ignored in timings –the application's self-reported runtimes are extracted from output logs.

The maximum allowed memory per process for Test Case B (LiH-HFX) was adjusted in the input file for each different choice of numbers of MPI ranks per nodes, with a safe limit that avoids the application running out of memory and being killed found to consist of roughly 0.75× the total on-node memory on each machine.

Test Case C (H2O-DFT-LS) generates large data structures that for a few machines do not fit into memory for smaller total number of nodes, which explains the absence of measurements for these small node counts.

4.3.3 Performance Results

Results for each test case run on each machine are given below, see the section after for analysis. In each case the choice of hybrid MPI + OpenMP runtime threading configuration that was found to be optimal and used in runs to generate scaling data is given.

4.3.3.1 Performance on JUWELS

Nodes	Time (s)	Speedup	Parallel efficiency (%)
1	1353.46	1.00	100.00
2	786.46	1.72	86.05
4	434.60	3.11	77.86
8	308.03	4.39	54.92
16	213.11	6.35	39.69
32	319.18	4.24	13.25

Table 19: Test Case A (using 24 MPI × 2 OpenMP)

Nodes	Time (s)	Speedup	Parallel efficiency (%)
1	1343.94	1.00	100.00
2	677.98	1.98	99.11
4	344.97	3.90	97.40
8	180.89	7.43	92.87
16	93.09	14.44	90.23
32	50.69	26.51	82.85
64	28.26	47.56	74.31
128	23.35	57.56	44.97

Table 20: Test Case B (using 2 MPI × 24 OpenMP)

Nodes	Time (s)	Speedup	Parallel efficiency (%)
2	576.26	1.00	100.00
4	313.96	1.84	91.77
8	188.36	3.06	76.48
16	109.37	5.27	65.86
32	81.36	7.08	44.27
64	56.78	10.15	31.72
128	95.84	6.01	9.39

Table 21: Test Case C (using 24 MPI × 2 OpenMP)

4.3.3.2 Performance on Piz Daint (XC50 partition, with GPU)

Nodes	Time (s)	Speedup	Parallel efficiency (%)	Energy (kJ)
1	3426.61	1.00	100.00	792
2	1805.93	1.90	94.87	832
4	1079.81	3.17	79.33	975
8	620.42	5.52	69.04	1140
16	422.55	8.11	50.68	1510
32	319.49	10.73	33.52	2200
64	294.02	11.65	18.21	3980
128	360.43	9.51	7.43	9300

Table 22: Test Case A (using 12 MPI × 1 OpenMP)

Nodes	Time (s)	Speedup	Parallel efficiency (%)	Energy (kJ)
1	5273.17	1.00	100.00	1050
2	2644.66	1.99	99.69	1080
4	1349.94	3.91	97.66	1110
8	691.98	7.62	95.26	1120
16	350.30	15.05	94.08	1140
32	185.36	28.45	88.90	1230
64	98.74	53.40	83.44	1370
128	57.19	92.20	72.03	1660
256	34.79	151.57	59.21	2300
512	27.43	192.24	37.55	3590
1024	26.63	198.02	19.34	8140

Table 23: Test Case B (using 12 MPI × 1 OpenMP)

Nodes	Time (s)	Speedup	Parallel efficiency (%)	Energy (kJ)
8	326.75	1.00	100.00	517
16	203.52	1.61	80.27	566
32	178.70	1.83	45.71	756
64	65.88	4.96	62.00	889
128	50.41	6.48	40.51	1330
256	34.65	9.43	29.47	1940
512	28.31	11.54	18.03	3330
1024	27.11	12.05	9.42	6620
2048	27.31	11.96	4.67	14450

Table 24: Test Case C (using 6 MPI × 2 OpenMP)

4.3.3.3 Performance on Piz Daint (XC50 partition, without GPU – CPU only)

Nodes	Time (s)	Speedup	Parallel efficiency (%)	Energy (kJ)
1	5643.44	1.00	100.00	1100
2	2719.30	2.08	103.77	1130
4	1567.81	3.60	89.99	1290
8	777.30	7.26	90.75	1310
16	485.85	11.62	72.60	1650
32	317.29	17.79	55.58	2100
64	241.21	23.40	36.56	3150
128	198.21	28.47	22.24	5060
256	214.57	26.30	10.27	10790
512	209.43	26.95	5.26	20850

Table 25: Test Case A (using 12 MPI × 1 OpenMP)

Nodes	Time (s)	Speedup	Parallel efficiency (%)	Energy (kJ)
1	5213.40	1.00	100.00	1020
2	2652.29	1.97	98.28	1040
4	1354.34	3.85	96.24	1050
8	690.03	7.56	94.44	1110
16	358.49	14.54	90.89	1140
32	178.59	29.19	91.23	1160
64	93.96	55.49	86.70	1260
128	50.46	103.32	80.72	1560
256	30.73	169.65	66.27	1860
512	23.28	223.94	43.74	2750
1024	24.26	214.90	20.99	6590
2048	35.80	145.63	7.11	17920

Table 26: Test Case B (using 12 MPI × 1 OpenMP)

Nodes	Time (s)	Speedup	Parallel efficiency (%)	Energy (kJ)
1	1306.53	1.00	100.00	1090
2	668.76	1.95	97.68	1130
4	379.41	3.44	86.09	1310
8	226.36	5.77	72.15	1550
16	147.12	8.88	55.50	2050
32	86.25	15.15	47.34	2480
64	58.09	22.49	35.14	3600
128	43.10	30.31	23.68	5200
256	27.10	48.21	18.83	7370
512	32.72	39.93	7.80	15940

Table 27: Test Case C (using 6 MPI × 2 OpenMP)

4.3.3.4 Performance on Frioul

Nodes	Time (s)	Speedup	Parallel efficiency (%)	Energy (kJ)
2	5917.00	1.00	100.00	1417.40
4	3737.00	1.58	79.17	1631.30
8	1922.00	3.08	76.96	1596.20
16	794.00	7.45	93.15	1520.20
32	424.00	13.96	87.22	1603.60
64	231.00	25.61	80.05	1795.50
128	147.00	40.25	62.89	2343.40

Table 28: Test Case B (using 8 MPI × 8 OpenMP)

Nodes	Time (s)	Speedup	Parallel efficiency (%)	Energy (kJ)
1	2963.00	1.00	100.00	1410.20
2	1210.00	2.45	122.44	1396.00
4	729.00	4.06	101.61	1531.00
8	383.00	7.74	96.70	1616.00
16	226.00	13.11	81.94	1857.00
32	139.00	21.32	66.61	2427.00

Table 29: Test Case C (using 8 MPI × 8 OpenMP)

4.3.3.5 Performance on DAVIDE (without GPU – CPU only)

Nodes	Time (s)	Speedup	Parallel efficiency (%)	Energy (kJ)
1	4686.00	1.00	100.00	2825.00
2	2344.00	2.00	99.96	2833.00
4	1194.00	3.92	98.12	2926.00
8	612.00	7.66	95.71	2978.00
16	323.00	14.51	90.67	3166.00

Table 30: Test Case B (using 16 MPI × 1 OpenMP)

Nodes	Time (s)	Speedup	Parallel efficiency (%)	Energy (kJ)
1	24573.00	1.00	100.00	15504.00
2	12502.00	1.97	98.28	15684.00
4	6380.00	3.85	96.29	16217.00
8	3295.00	7.46	93.22	16777.00
16	1695.00	14.50	90.61	17314.00

Table 31: Test Case C (using 16 MPI × 1 OpenMP)

4.3.3.6 Performance on DAVIDE (with GPU)

Nodes	Time (s)	Speedup	Parallel efficiency (%)	Energy (kJ)
2	4657	1.00	100.00	3458
3	2337	1.99	99.64	3484
16	320	14.55	90.96	3963

Table 32: Test Case B (using 16 MPI × 1 OpenMP)

4.3.3.7 Performance on DEEP-ER SDV

Nodes	Time (s)	Speedup	Parallel efficiency (%)
1	3036.90	1.00	100.00
2	1773.79	1.71	85.60
4	1006.19	3.02	75.46
6	750.04	4.05	67.48

Table 33: Test Case A (using 24 MPI × 1 OpenMP)

Nodes	Time (s)	Speedup	Parallel efficiency (%)
1	2882.87	1.00	100.00
2	1509.32	1.91	95.50
4	785.97	3.67	91.70
8	433.14	6.66	83.20
16	256.84	11.22	70.15

Table 34: Test Case B (using 12 MPI × 2 OpenMP)

Nodes	Time (s)	Speedup	Parallel efficiency (%)
2	1464.77	1.00	100.00
4	779.10	1.88	94.00
8	403.13	3.63	90.84

Table 35: Test Case C (using 12 MPI × 2 OpenMP)

4.3.3.8 Performance on Dibona

Nodes	Time (s)	Speedup	Parallel efficiency (%)
1	2402.36	1.00	100.00
2	1334.29	1.80	90.02
4	769.59	3.12	78.04
8	477.77	5.03	62.85
16	336.71	7.13	44.59

Table 36: Test Case A (using 64 MPI × 1 OpenMP)

Nodes	Time (s)	Speedup	Parallel efficiency (%)
1	1800.09	1.00	100.00
2	918.89	1.96	97.95
4	468.31	3.84	96.10
8	244.09	7.37	92.18
16	132.92	13.54	84.64

Table 37: Test Case B (using 64 MPI × 1 OpenMP)

Nodes	Time (s)	Speedup	Parallel efficiency (%)
1	2909.43	1.00	100.00
2	1397.04	2.08	104.13
4	739.62	3.93	98.34
8	406.22	7.16	89.53

Nodes	Time (s)	Speedup	Parallel efficiency (%)
16	222.92	13.05	81.57

Table 38: Test Case C (using 64 MPI × 1 OpenMP)

4.3.4 Performance comparisons

We present in Figure 2, Figure 3 and Figure 4 comparisons of the performance of Test Cases A, B and C on the range of systems on which benchmarking was performed. Scaling runs for each system and test case were performed after first determining hybrid MPI + OpenMP runtime configurations that yield optimal performance across a range of node sizes on each system, with the comparisons between systems being made between these optimal runs to more realistically reflect production usage of the respective machines.

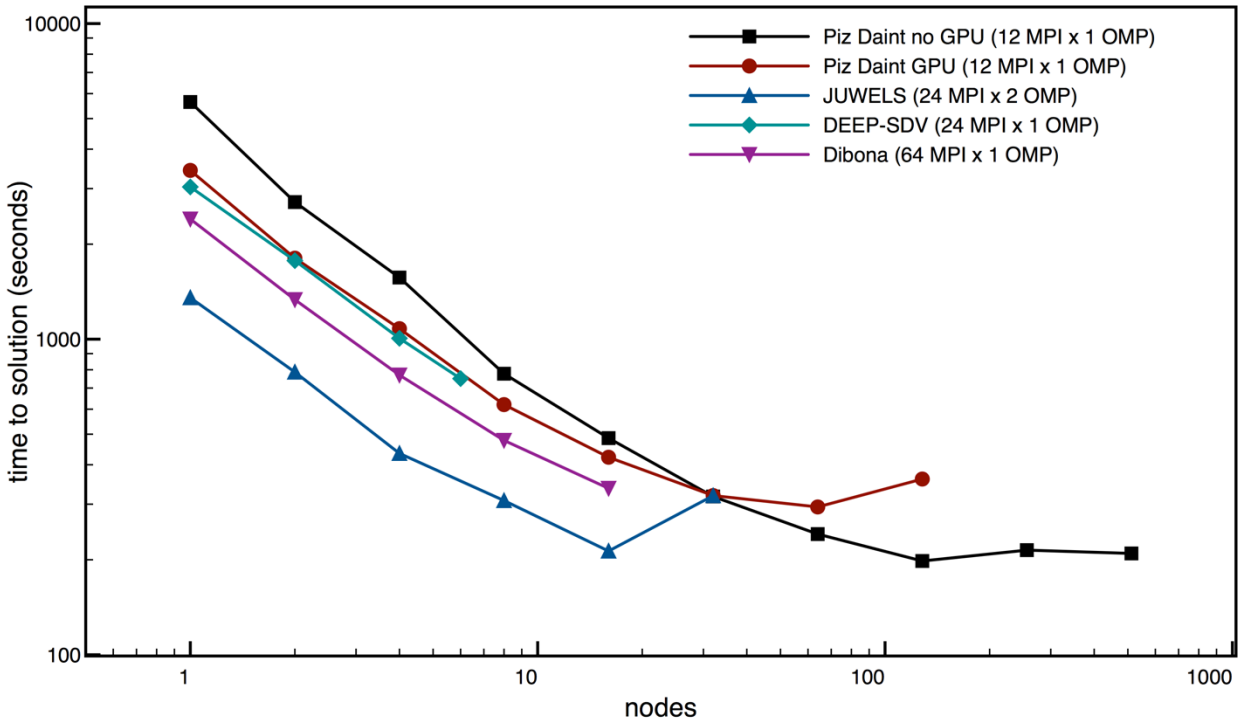


Figure 2: time to solution for Test Case A (H2O-512)

Runs on Piz Daint with and without usage of the P100 GPUs were all performed on the XC50 GPU partition, thereby taking advantage of the opportunity to compare the performance of CP2K with and without GPU whilst keeping all else – CPU, memory, interconnect and software stack – constant. For test case B (LiH-HFX) we see no benefit from GPU usage at any scale. For test case A (H2O-512) we see significant benefit from the GPU for fewer than 32 nodes but with CPU-only runs winning out for larger node counts, possibly due to strong scaling diminishing the amount of node-local work that can be offloaded, meaning data transfer overheads become too costly. For test case C (H2O-DFT-LS) there is significant benefit up to large node counts. Examining the logs from GPU-enabled runs of all three test cases and analysing the statistics of calls to CP2K's DBCSR acceleration layer, it is clear that a much higher percentage of small matrix multiplication calls are successfully offloaded to the GPU in test cases A and C, which are likely benefiting from the GPU

as a result. This suggests custom autotuning work to generate kernels to enable more efficient GPU offloading as described at [48] and [49] could improve performance of test case B on GPU-equipped machines. It is worth noting that given the current state of DBCSR, this would require significant effort from HPC users and/or support staff (which was beyond scope of the porting and benchmarking effort undertaken in this project). However, development work on DBCSR currently being undertaken is expected to bypass the need for custom autotuning in future, which would enable better offloading by default for a wider range of simulations.

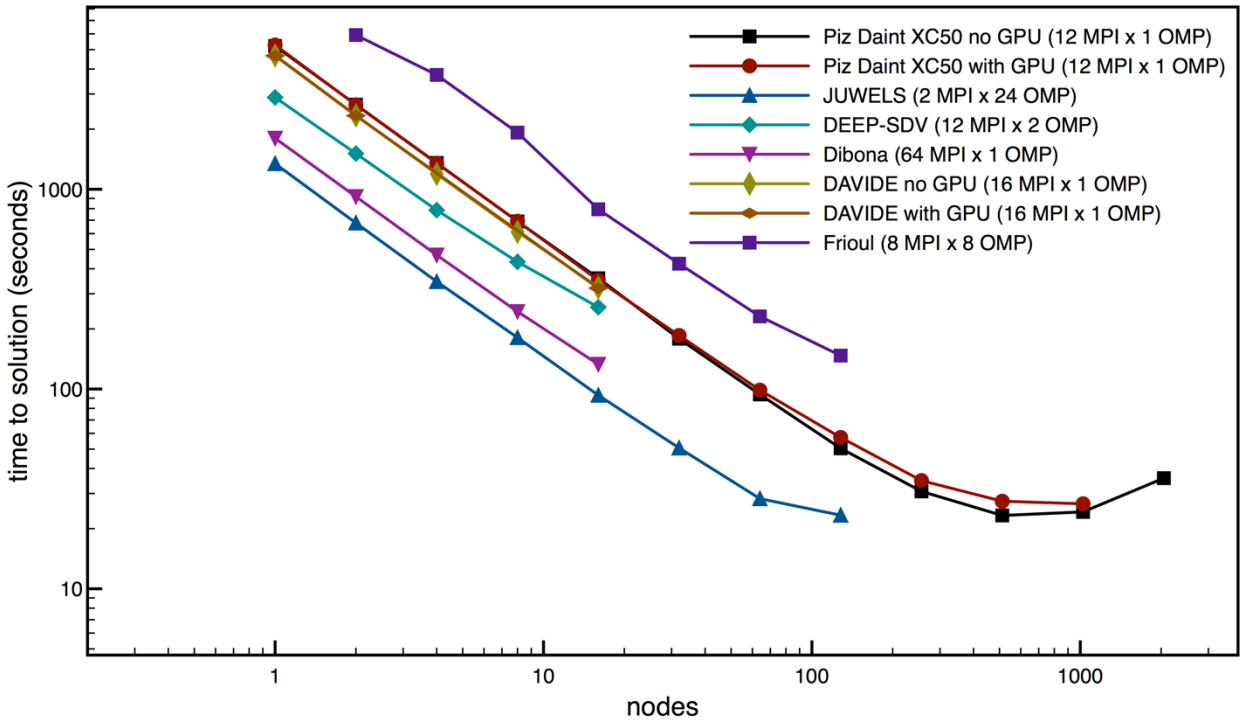


Figure 3: time to solution for Test Case B (LiH-HFX). Note that runtimes on Dibona, DAVIDE (with and without GPUs) and Piz Daint (with and without GPUs) are all very similar for 1–16 nodes.

With regards to performance of CP2K using the same P100 GPUs on the DAVIDE PCP prototype system, which provides four such GPUs per node, it was found that the benefit for offloading was equally limited for test case B and that it provided no observable performance benefit. Test cases A and C were not run successfully on DAVIDE GPUs due to various errors thrown by the CUDA runtime and linear algebra library respectively. These errors could not be resolved during the available access time to the prototype system. Based on experience with CP2K GPU offloading in this project and publicly available guidance from developers it is expected that although test cases A and C would be likely to benefit from the GPUs available on DAVIDE in a similar way as on Piz Daint, the amount and type of compute offload generated by these test cases would be far from enough to exploit four on-node GPUs concurrently with good parallel efficiency.

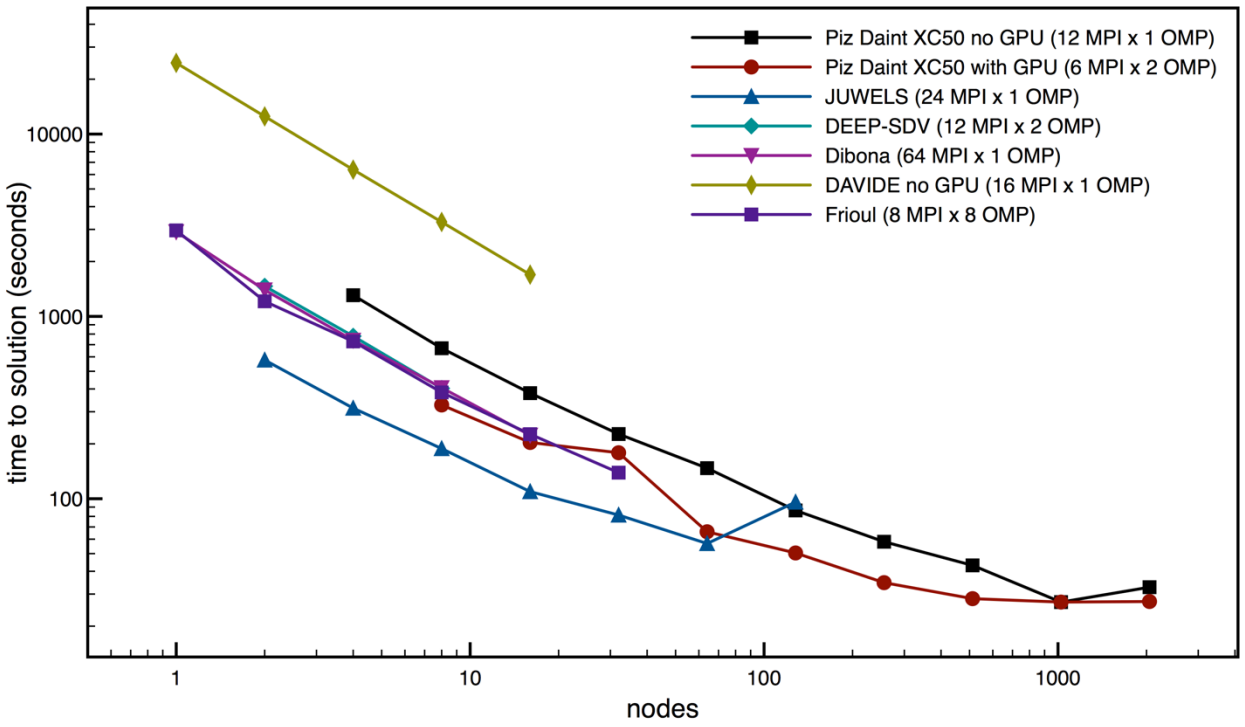


Figure 4: time to solution for Test Case C (H2O-DFT-LS). Note that runtimes on Dibona, Frioul, and DEEP-ER SDV and to a lesser extent Piz Daint with GPU are all very similar for 1–32 nodes

It is important to note that in general not only the absolute performance of CP2K as reported for these benchmarks but also the rate at which the application's parallel scaling efficiency diminishes do not accurately reflect the full potential of the code when it is linked against all possible performance libraries and carefully tuned, however this should not invalidate the cross-system comparison which we undertake here and the conclusions drawn.

Scaling behaviour on all the PRACE PCP prototypes as well as the Mont-Blanc 3 and DEEP-ER prototypes appears roughly on par with that of large established systems such as JUWELS and Piz Daint. In fact, JUWELS appears to lose parallel scaling efficiency rather earlier than e.g. Piz Daint, especially for test cases A and C, negating its node-for-node basis performance advantage thanks to a more recent generation Intel processor. This is reflected in the CP2K logs, which show that for the largest node count benchmarked in the latter two cases, for which there is an increase in runtime, the application spends a much larger percentage of its runtime in MPI operations, specifically waiting and collective operations.

In general, on a node-for-node basis, performance on JUWELS where each node has two 24-core Skylake processors is consistently fastest, exceeding performance on machines equipped with older generation Intel processors or with ThunderX2 or POWER8-based nodes, apart from at the largest scale where JUWELS' poorer MPI performance for CP2K negates this raw processing power advantage. Performance on ThunderX2-based Dibona appears competitive across the board, especially for test case B where it starts to approach JUWELS' Skylake-based performance. For test case C, Dibona and DEEP-ER SDV nodes match Piz Daint's CPU+GPU performance node for node. The same applies for DEEP-ER SDV in test case A, where Dibona comfortably exceeds

it as well as Piz Daint. Frioul performs very similarly to Dibona, DEEP-ER SDV and Piz Daint CPU+GPU in test case C, but not competitively with even Piz Daint's CPU-only 12-core Haswell nodes in test case A. Finally, while DAVIDE nodes match Piz Daint nodes in test case B, DAVIDE's performance in test case B is significantly slower than all the other systems.

As well as relying strongly on high-performance linear algebra libraries (which were provided on most benchmark systems by an MKL installation), CP2K is heavily dependent for performance on efficient MPI communication and hence a low latency, high bandwidth interconnect for decomposing and reconstituting variables onto grids, calling Fourier transforms, etc. Test cases for which results from several different systems exhibit very similar runtimes despite running on nodes with significantly different raw processing power may therefore reflect instances of communication-bound computing which, if the respective machines' communication layers do not differ significantly in their performance, causes similar performance bottlenecks to appear.

4.3.5 Energy consumption comparisons

Having considered performance across the various systems we would like to add to this comparison an evaluation of energy consumption in an attempt to better understand the possibilities offered by the PRACE PCP prototypes and future technologies. In performing this comparison, we will also make use of the rough energy measurements provided for jobs on Piz Daint which, although recorded with significantly lower sampling frequency and specificity than the specialised hardware and software developed for DAVIDE and Frioul, nonetheless gives us a context within which to better understand results from these prototype systems. In doing this comparison we aim to highlight findings from benchmarking CP2K that should help inform how best to trade off performance and energy consumption.

To provide some quantitative context, it has been confirmed by Piz Daint support staff that a job strongly using the GPU will use 30.97 kJ during the same time that a job not using the GPU at all uses 7.98 kJ. However, it is important to note that the estimated job energy usage measurements provided on Piz Daint do not include the interconnect or any cabinet-level hardware. Where we make a comparison to the PCP prototypes, in the case of Frioul we can provide information on the breakdown of energy on that system into node energy and switch energy in order to compare it to Piz Daint fairly.

Figure 5 shows the energy to solution for test case A, which was only run on Piz Daint and not on either of the PCP prototype systems. Considering also the corresponding performance shown in Figure 2 and summarising calculations based on the raw data, we can state the following with regards to CP2K running test case A:

- Performance on Piz Daint with GPU for one node is 1.65× that without GPU. This advantage steadily decreases with more nodes, and switches over somewhere between 16 and 32 nodes to benefit running without GPU. At 128 nodes, performance with GPU is 0.55× that without GPU.
- Total energy consumption with GPU for one node is 0.72× that without GPU. This advantage steadily decreases with more nodes and switches over somewhere between 16

and 32 nodes to benefit running without GPU. At 128 nodes, total energy consumption with GPU is 1.84× is that without GPU.

- Average power consumption with GPU for one node is 1.2× that without GPU (symptomatic of the GPU being used somewhat) and decreases steadily to being almost identical to running without GPU for 128 nodes, symptomatic of little use of GPU.

Conclusion for test case A: this test case benefits from using the GPU for fewer than 16–32 nodes on the Piz Daint architecture both from the perspective of performance and from the perspective of reduced total energy consumption, however for large number of nodes the opposite is true and it is both quicker and more energy efficient not to use the GPU. The desired crossover point to this decision from either performance or energy perspective will depend on prioritisation. Average power drawn by the simulation varies only about 20% over the range of number of nodes considered, so may not have a big impact on the choice of what scale to run at, though this information could be utilised as part of power capping planning.

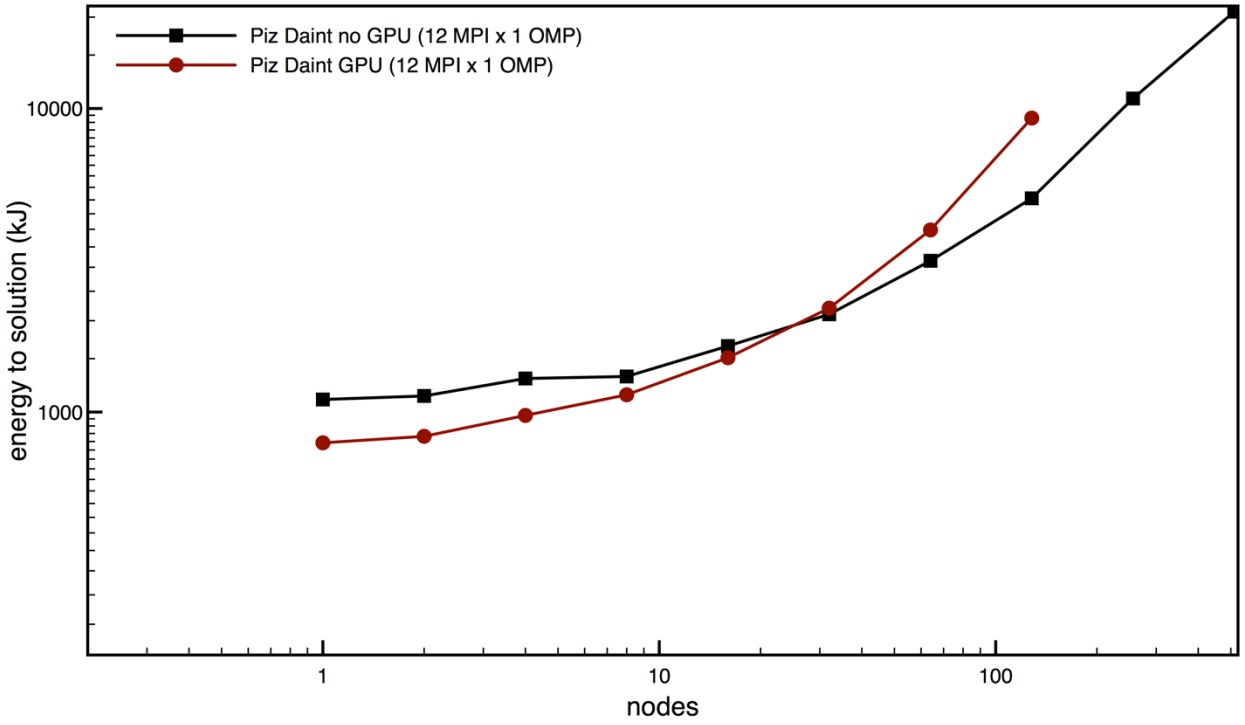


Figure 5: Energy to solution for Test Case A (H2O-512)

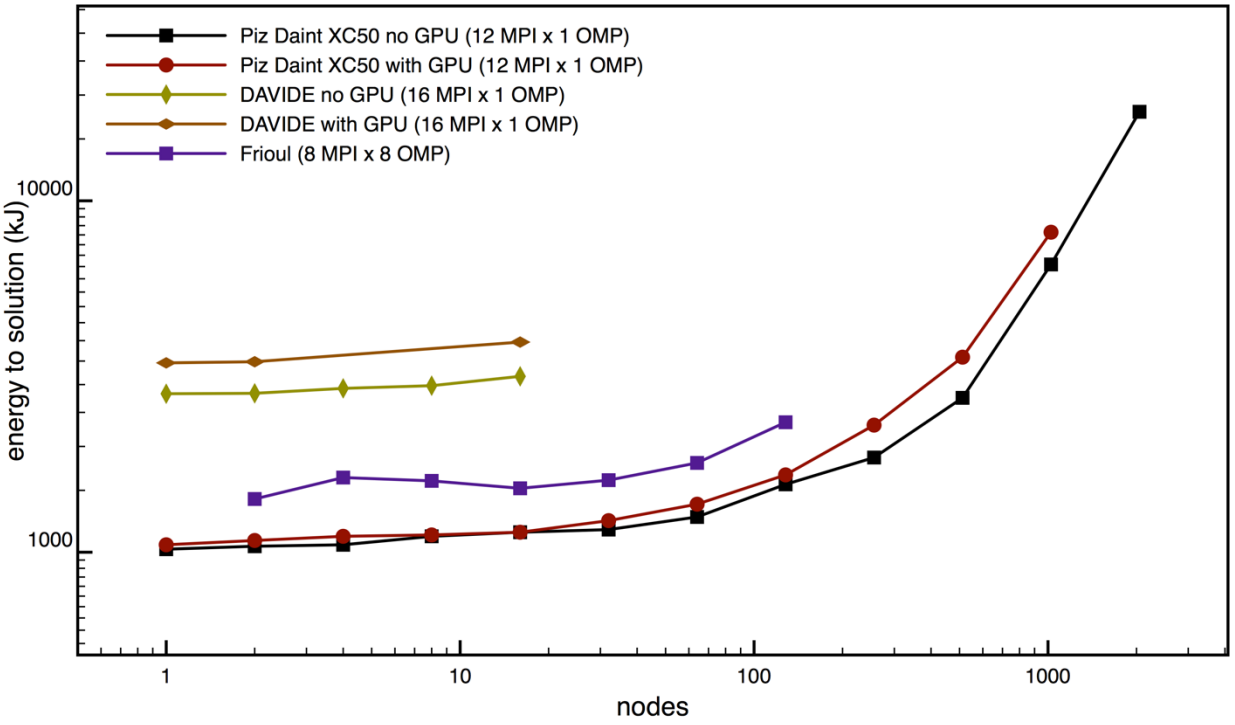


Figure 6: Energy to solution for Test Case B (LiH-HFX)

Figure 6 shows scaling of total energy consumption with job size for test case B and including both PCP prototype systems as well as Piz Daint. Considering just Piz Daint, we can see there is very little difference between both total energy consumption and runtime running with or without GPU. In fact, we can calculate that it is slightly worse on both accounts to run with GPU. Frioul has both higher energy consumption (1.3× – 1.4× that of Piz Daint for the same number of nodes) and (from results shown earlier) longer runtime (2.2× – 2.5× that of Piz Daint). To determine whether this is a fair comparison given that the Piz Daint energy measurements are node-level only and do not include networking or cabinet-level hardware, we can examine the breakdown of total job energy

on Frioul into node-level energy and switch energy. This is shown below for test case C, but is also representative of test case B.

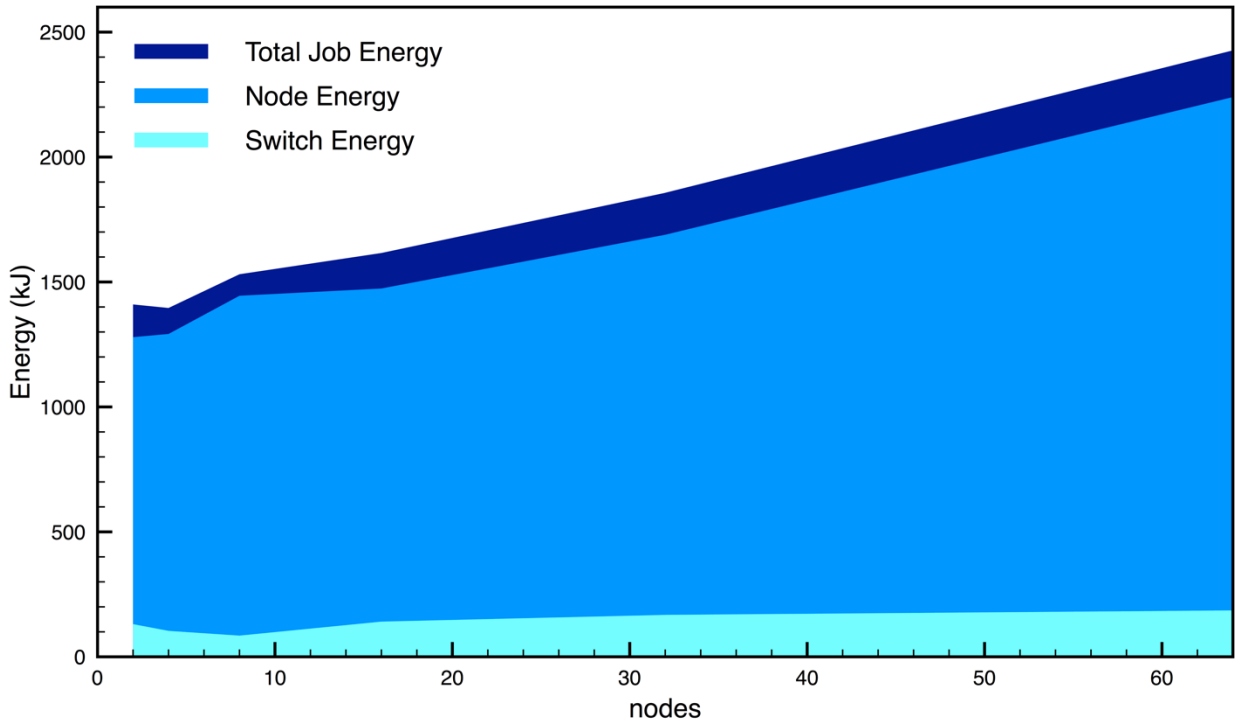


Figure 7: breakdown of node and switch energy contributions to total job energy on Frioul for test case C (H2O-DFT-LS)

This shows that the contribution from the switch energy is not significant and that the conclusions about Piz Daint's greater energy efficiency hold once compared purely with Frioul's node energy.

With regards to DAVIDE, whilst DAVIDE both with and without GPU is marginally faster on a node-for-node basis than Piz Daint (runtime 0.9× that of Piz Daint), DAVIDE without GPU uses

2.7× – 3.5× more energy than Piz Daint and DAVIDE with GPUs uses 4.4× – 3.5× that of Piz Daint.

Piz Daint is therefore considerably more energy efficient than either of the PCP prototype systems when it comes to running CP2K test case B.

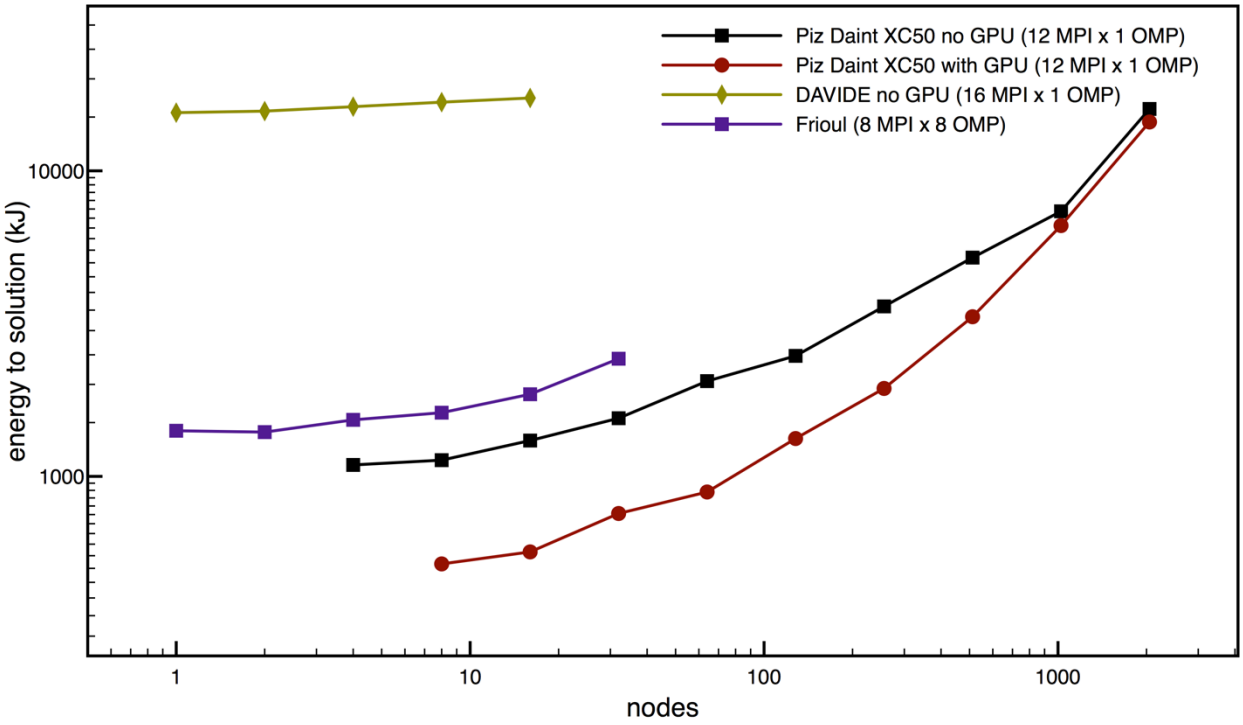


Figure 8: Energy to solution for Test Case C (H2O-DFT-LS)

Considering first Piz Daint, Figure 9 in combination with performance results shows that:

- Performance with GPU for one node is more than two times that without GPU, and running with GPU continues to provide better performance for more nodes.
- Total energy consumption with GPU is always lower than without GPU, but increases from 0.46× that without GPU for 1 node to 0.9× that without GPU for 2048 nodes.
- Average power consumption with GPU is always lower (except for at around 2048 nodes), drawing between 0.62× and 0.97× the average power consumption without GPU.

Clearly, this test case greatly benefits from the use of the GPU on the Piz Daint architecture regardless of how many nodes are being used; performance, total energy consumption and average power consumption are all better with GPU.

Concerning the PCP prototype systems: as for test case B, Frioul has higher total energy consumption than Piz Daint in test case C (~3.2× that of Piz Daint with GPU) and takes ~1.2× as long to complete the simulation. Finally, DAVIDE running test case C without use of the GPU appeared to exhibit a surprising 32× – 85× greater energy consumption than Piz Daint reproducibly across runs and for different node counts. It is possible that the CUDA runtime mistakenly engaged all four GPUs, contributing to greater than expected power draw. As previously mentioned, test

cases A and C were not run successfully on DAVIDE GPUs due to various errors thrown by the CUDA runtime and linear algebra library respectively. These errors could not be resolved during the available access time to the prototype system. For test case C, as for test case B, Piz Daint appears to be considerably more energy efficient running CP2K. A recurring insight from these energy and performance comparisons appears to be that energy efficiency, high performance, and especially high parallel efficiency are not only complementary but in fact highly correlated.

4.3.6 Analysis of threading and energy on Frioul

Results on Frioul were obtained as part of the preceding PRACE-4IP project extension and analysed here in more depth, including with direct comparison to performance and energy consumption results on other systems in order to provide new insight. No new benchmarking data were obtained as part of the PRACE-5IP project since the newer software stack on the system did not include a CP2K-compatible version of the Intel compiler nor any MPI library version built with gcc, and because the Bull Energy Optimizer (BEO) software tool required to collect energy measurements was not available for a significant part of the machine access time during PRACE-5IP.

Tests were run with KNLs configured in Quad/Flat mode. The command **numactl --preferred=1** was used to allow memory allocation in MCDRAM until exhausted. All benchmarks were run with one hyperthread per physical core as previous KNL benchmarking showed no benefit in using multiple hyperthreads per core. The process pinning options **I_MPI_DOMAIN=auto** and **I_MPI_PIN_RESPECT_CPUSET=0** were used, and thread affinity controlled with **KMP_AFFINITY=compact,1,0,granularity=fine,verbose** and **KMP_HW_SUBSET=64C,1T**.

Figure 9 illustrates the effect the choice of MPI ranks per node and OpenMP threads per rank can have on the time to solution by showing results for Test Case C (H2O-DFT-LS) on Frioul. Similar behaviour was found for Test Case B (LiH-HFX). Pure MPI performs worst. Adding multithreading improves performance, until ~ 8 threads per process (the best case) at which point the time to solution is almost 2× faster than for pure MPI. Performance deteriorates for > 8 threads per process, but is never quite as bad as for < 8 threads and as for pure MPI.

These results are not typical for CP2K running on conventional multicore processors, where one or two threads per process give best performance for a variety of problems and where higher thread counts are often used primarily as a way of alleviating memory requirements, allowing the simulation to run at all or potentially faster as more intermediate results can be stored in memory rather than recomputed on the fly. On Intel's MIC architecture CP2K clearly benefits from increased threading. This may in part be due to the smaller amount of memory available per core on this architecture, meaning that more cores should be used jointly for a single MPI rank requiring a certain minimum amount of memory (or benefiting from additional available memory).

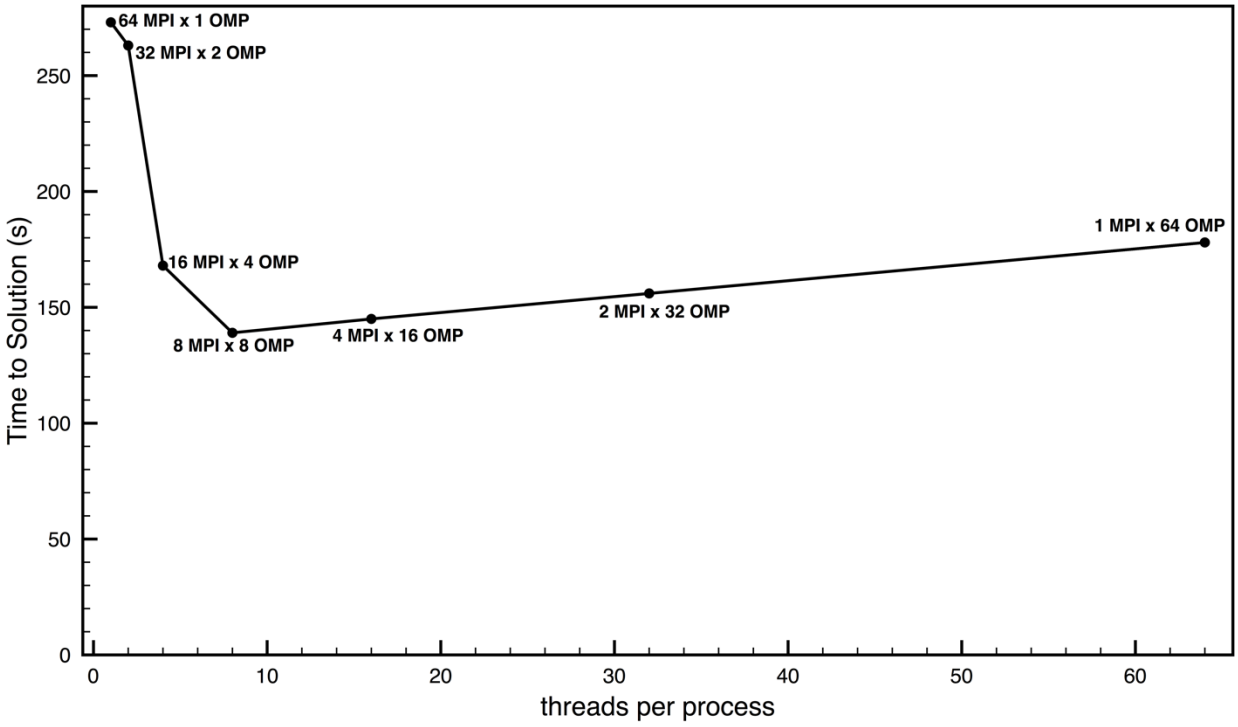


Figure 9: Time to solution for Test Case C (H2O-DFT-LS) on 64 nodes of Frioul for different choices of number of MPI processes per node and number of OpenMP threads per process

When we look at the total energy to solution for the same test case (Figure 10) we see it directly mirrors the trend in the time to solution. This suggests energy usage for a fixed number of nodes is simply proportional to runtime, with the same power drawn independent of the degree of threading in code execution. In other words, runs use more energy simply because they last longer, not because they cause the hardware to draw more power over the same (or even shorter) time than more energy-efficient runs. To understand this better, we plot in Figure 11 the average power (total job energy / runtime) for this test case running on different numbers of nodes, which will also show if average power drawn for a given threaded execution mode differs depending on the number of nodes in the job.

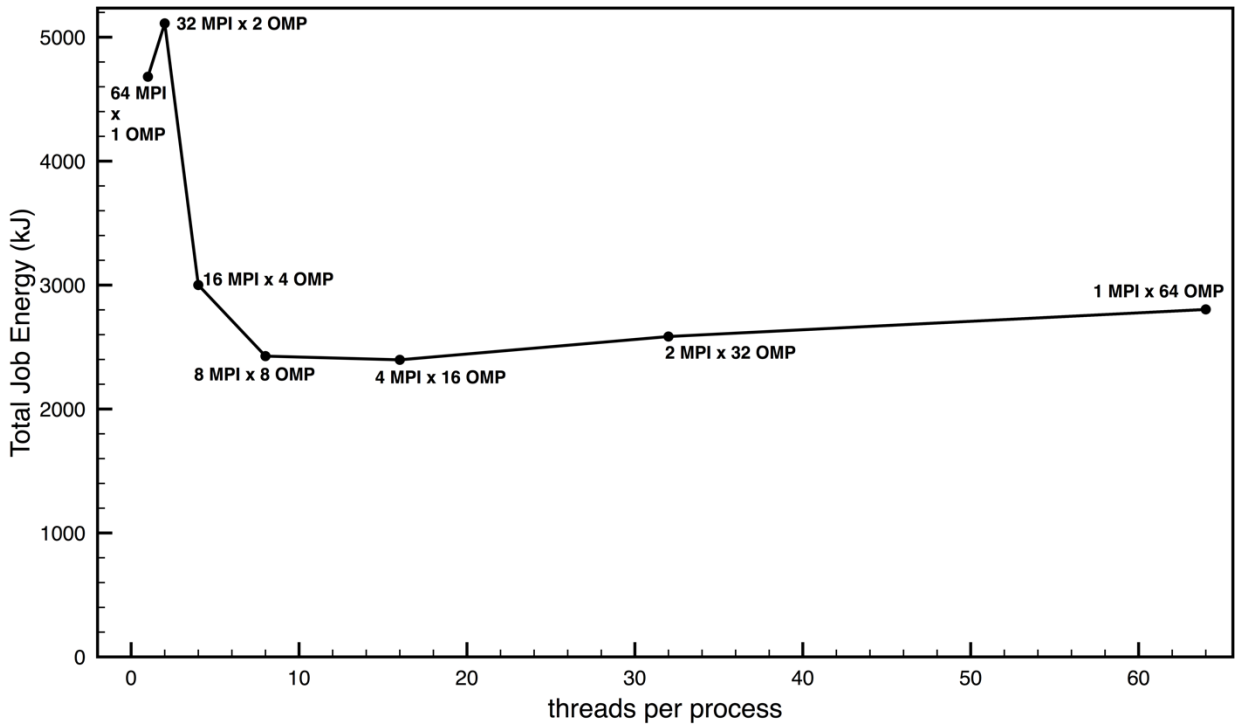


Figure 10: Energy to solution for Test Case C (H2O-DFT-LS) on 64 nodes of Frioul for different choices of number of MPI processes per node and number of OpenMP threads per process

Figure 11 shows that average power drawn by nodes is not completely independent of threading but follows a regular pattern. More threads per process leads to somewhat lower average power draw. For a given threading choice, average power draw grows roughly linearly with number of nodes. The fastest, lowest-energy execution modes use 8–16 threads per process, and draws middling power on average but completes quickly enough to still win overall on energy. Runs with > 8–16 threads draw less power on average but suffer performance penalty and so incur a higher overall energy cost as a result of longer runtimes. The outlier is pure MPI (64 MPI × 1 OMP) execution, which on 32 nodes draws by far the most power of all execution modes, on 64 nodes draws middling power, but which always loses on total energy consumption due to more than proportionally increased runtime.

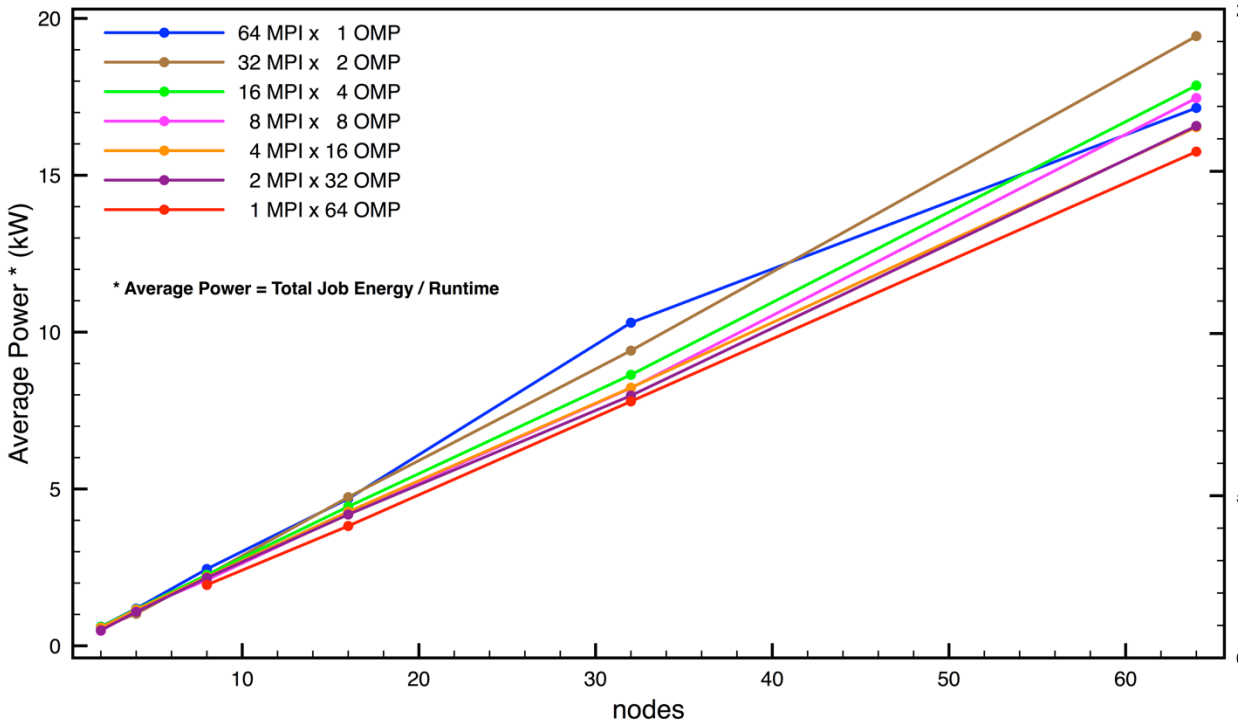


Figure 11: Average power drawn during runs for Test Case C (H2O-DFT-LS) on 64 nodes of Frioul for different choices of number of MPI processes per node and number of OpenMP threads per process

4.3.7 Conclusions

We have used CP2K benchmarks to quantify some of the performance and energy characteristics of PRACE PCP and other prototypes with reference to established Tier-0 HPC systems in a way that has allowed us to cross compare between these systems and draw some conclusions regarding what they offer from the perspective of running an important scientific application. In doing so we have found a recurring insight gained from these energy and performance comparisons to be that energy efficiency and high (parallel) performance and efficiency are not only complementary but in fact highly correlated, and that focusing on the one is likely to help improve the other.

Disregarding the impact of poorer MPI performance at large scale on JUWELS, its Intel Skylake-based nodes were found to offer superior node-for-node CP2K performance compared to older generation Haswell-equipped Piz Daint and DEEP-SDV nodes, KNL-equipped Frioul nodes, ThunderX2-based Dibona nodes, and Power8-based DAVIDE nodes. However, Dibona's ThunderX2-based nodes were found to offer competitive performance, especially compared to Haswell-based Piz Daint and DEEP-SDV. Offloading to a single GPU per node on Piz Daint was found to provide significant performance benefit for two out of three test cases, with no significant benefit for the third test case. It was not possible to ascertain the benefit from using four GPUs per node as available on DAVIDE for the two test cases that were found to benefit from single GPUs per node, however the guidance from developers is that a minority of simulations and systems are

expected to generate enough computational work to overcome offloading overheads and efficiently make use of four GPUs per node as available on DAVIDE.

With regards to energy: energy consumption on Piz Daint – especially with use of GPU – was found to be significantly more energy efficient than the PCP prototype systems Frioul and especially compared to DAVIDE. A detailed investigation on Frioul of the effect of choice of OpenMP threading on performance and energy consumption with all else being equal showed that both vary strongly with the number of threads per MPI rank. It was found that for CP2K the choice that uses least energy on the Frioul architecture is the same as yields the shortest runtime.

As well as allowing us to compare performance and energy usage, the process of benchmarking across established Tier-0 and novel prototype architectures has highlighted the importance for successful porting and efficient usage of scientific applications such as CP2K of having a software stack that includes MPI and performance libraries that are maximally compatible with a range of compilers and compiler versions.

4.4 GADGET

We carried out tests using the PRACE Tier-0 facilities (JUWELS [28], MareNostrum4 [30], Marconi-KNL [29], and the Mont-Blanc 3 prototype Dibona [36]) in order to benchmark GADGET-3 and determine its weak and strong scaling. However, we were only able to successfully run Test Cases A and B. GADGET was compiled with the optimisation level O2. The tests were carried out with one MPI-task per core and the code ran for 50 timesteps.

4.4.1 System and software environment

MareNostrum4: Software modules FFTW/2.1.5, GSL/2.4 and HDF5/1.10.1. Compiler: Intel C Compiler Ver.17.0.4.

JUWELS: Software modules: GSL/2.5 and HDF5/1.10.1. Compiler: Intel C Compiler Ver. 19.0.0.117. We installed FFTW.2.1.5 as it is not available in JUWELS.

Marconi-KNL: Software modules: profile/advanced, intel/pe-xe-2018, intelmpi/2018, GSL/2.5 and hdf5/1.10.4. Compiler: Intel C Compiler Ver. 18.0.5. We installed FFTW.2.1.5.

Mont-Blanc 3 prototype Dibona: Software modules: openmpi4.0.0/arm19.0. Compiler: ARM Compiler 19.0. We installed FFTW.2.1.5 and GSL/2.5.

4.4.2 Modifications carried out in GADGET-3

The function forcefree.c includes the MPI_Allgatherv call with the overlap of the input and output buffers (MPI_Allgatherv(&DomainMoment[DomainStartList[ThisTask * MULTIPLEDOMAINS + m]],revcounts [This Task],…). This is not allowed by the MPI standard and Intel MPI detects this error and stops the program with a message about aliased buffers (“PMPI_Allgatherv(1379):

Buffers must not be aliased"). A way out is using MPI_IN_PLACE in the call, that is, MPI_Allgatherv(MPI_IN_PLACE,recvcounts [This Task], …).

4.4.3 Dynamic analysis

Performance analysis was carried out in order to determine the timings of the different functions

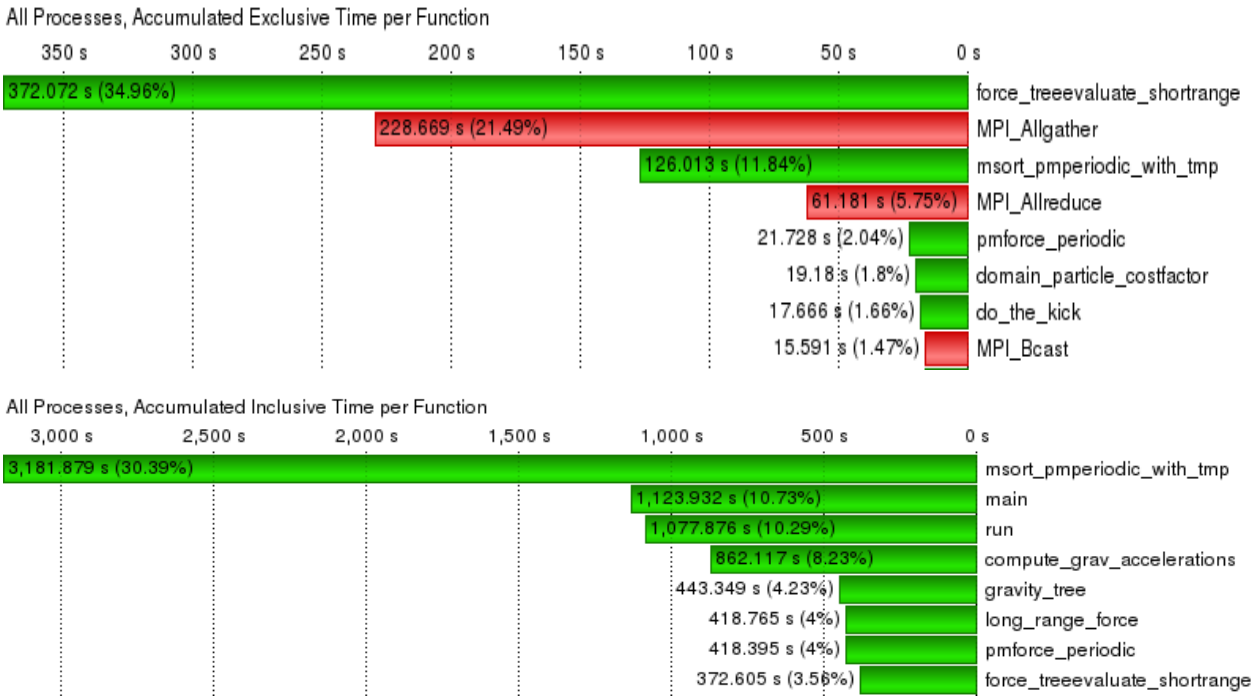


Figure 12: Accumulated exclusive (top panel) and inclusive (bottom panel) times per function

and the MPI calls. This analysis was carried out in JUWELS loading the modules Score-P/4.1 and Vampir/9.5.0. The latter was used for event trace visualisation. The GADGET code was compiled with the Score-P wrappers for mpicc and mpic++. The relative inclusive (that is, the amount of time spent in a function and all of its subroutines) and exclusive (that is, the amount of time spent in a function) times per function shown in Figure 12 (top and bottom panels) were measured for Test Case A running for 50 timesteps and using 16 cores with 1 MPI task per core. The execution time of the code is dominated, excluding the MPI_Allgather, by the force_freeevaluate_shortrange (35%), and msort_pmperiodic_with_tmp (11.8%) functions. The code may be accelerated through the insertion of OpenMP directives in these two functions.

4.4.4 Performance Results

Test Case A – Results and Analysis

The results of this test case A are shown in Table 39 and displayed in Figure 13. The different systems show an increase in computing time as we move from the SKL CPUs to ARM and the KNL in Marconi. MareNostrum4 shows a speedup a little bit larger than the remaining systems,

except for 128 cores where Dibona has the largest speedup. Overall the SKL systems are faster in this test and their clock speed determines the observed timings.

MPI Tasks per CPU	# Nodes	# Cores	JUWELS		MareNostrum4		Dibona	
			Time (s)	Speedup	Time (s)	Speedup	Time (s)	Speedup
8	1	8	206	1.00	259	1.00	322	1.00
16	1	16	107	1.93	129	2.01	168	1.92
16	1	32	56	3.68	67	3.87	86	3.74
16	2	64	32	6.44	37	7.00	51	6.31
16	4	128	19	10.84	22	11.77	26	12.38
16	8	256	15	13.73	12	21.58	17	18.94

MPI Tasks per KNL	# Nodes	# Cores	Marconi	
			Time (s)	Speedup
8	1	8	1055	1.00
16	1	16	514	2.05
32	1	32	302	3.49
64	1	64	169	6.24
64	2	128	99	10.66
64	4	256	63	16.75

Table 39: Small size problem computing times and speedup

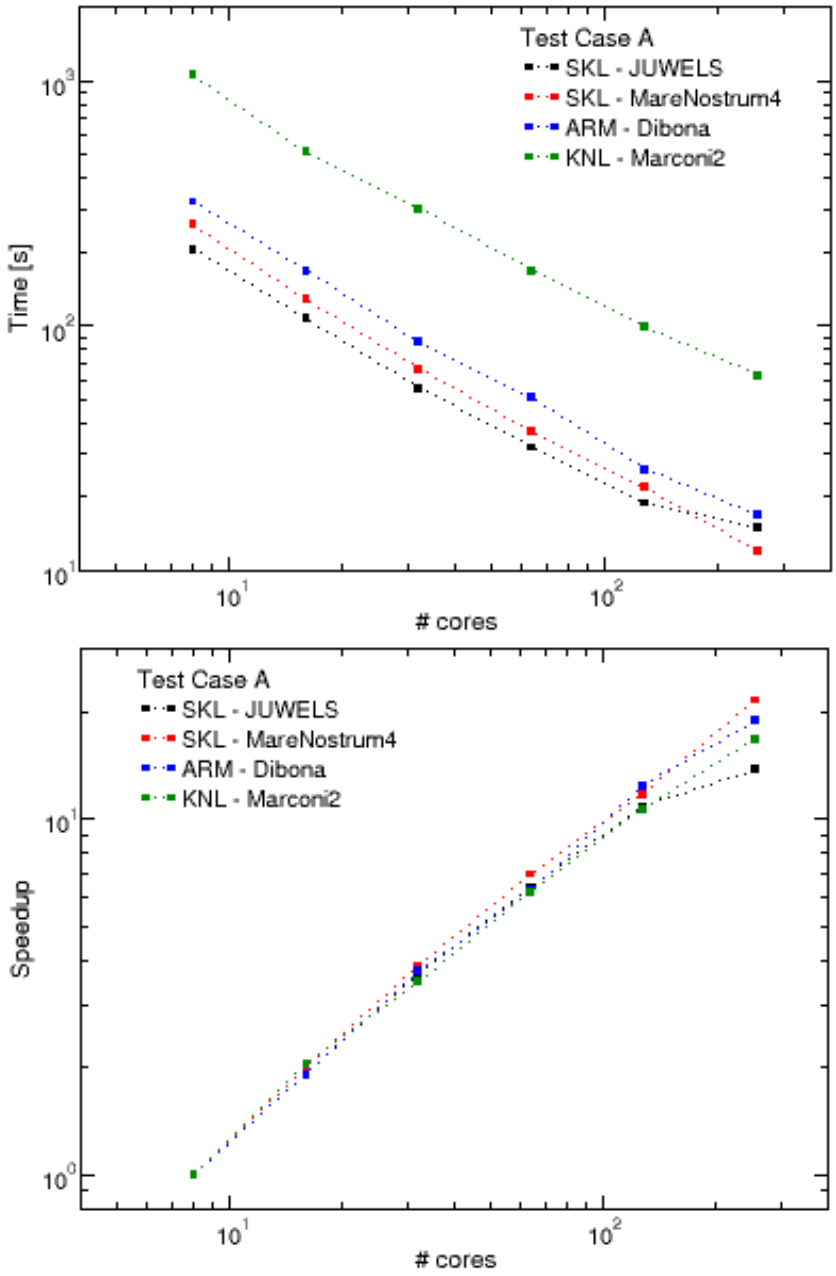


Figure 13: Variation of the computing time vs. number of cores (top panel) and speedup (bottom panel) for Test Case A.

Test Case B – Results and Analysis

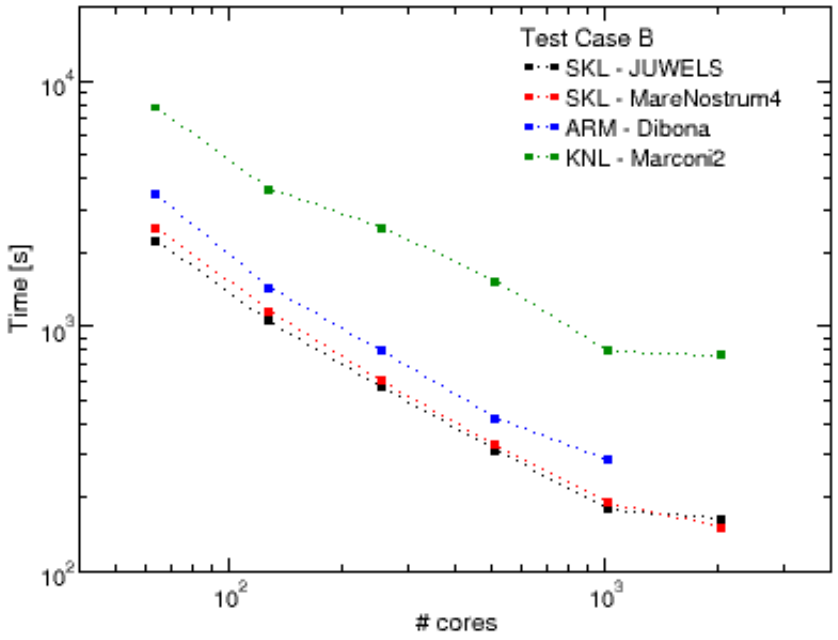
The results of this test case are shown in Table 40 and displayed in Figure 14. Contrary to the clear separation between computing times in Test Case A, here both JUWELS and Marenostrum4 have similar timings, while the remaining systems lag behind in the timing. Clearly the ARM CPU is faster than the KNL, although the explanation for such a large difference resides not only in the size of the problem, but also on the architecture of the two processors and their clock speeds. The

speedup between the SKL (JUWELS and MareNostrum4) and the ARM systems are similar for the present test case.

MPI Tasks per CPU	# Nodes	# Cores	JUWELS		MareNostrum4		DIBONA	
			Time (s)	Speedup	Time (s)	Speedup	Time (s)	Speedup
16	2	64	2235	1.00	2519	1.000	3425	1.00
16	4	128	1053	2.12	1148	2.194	1431	2.39
16	8	256	567	3.94	598	4.212	794	4.31
16	16	512	312	7.16	330	7.633	422	8.12
16	32	1024	180	12.42	191	13.188	285**	12.02**
16	64	2048	165	13.55	151	16.682		

MPI Tasks per KNL	# Nodes	# Cores	Marconi	
			Time (s)	Speedup
8	1	64	7787	1.00
32	2	128	3610	2.16
32	4	256	2508	3.10
64	8	512	1530	5.09
64	16	1024	791	9.84
64	32	2048	763	10.21

Table 40: Medium size problem computing times and speedup



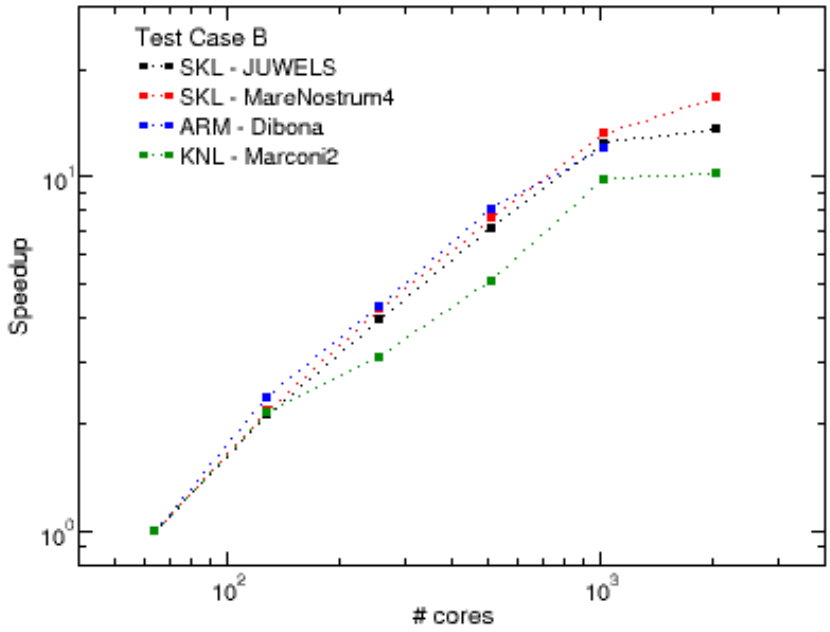


Figure 14: Variation of the computing time vs. number of cores (top panel) and speedup (bottom panel) for Test Case B

4.4.5 Conclusion

We have carried out tests for the GADGET code by running a cosmological simulation in established Tier-0 SKL and KNL systems and in an ARM prototype system. With the exception of Test Case A, the remaining tests are highly demanding in terms of computing resources. Hence, these tests provide an indication regarding the adequate systems to run such heavy simulations. It is clear that the SKL and ARM systems are better suitable to handle complex N-body and SPH cosmological simulations with the GADGET code.

4.5 GPAW

The performance of GPAW for the benchmarks described in Section 2.5 was measured and compared on different systems with a range of parallel job sizes (Table 41, Table 42, and Table 43). Only the time spent in the main computational loop (SCF-cycle) was used as the runtime in the comparison to exclude any differences in initialisation overheads.

The systems and architectures covered included: JUWELS, MareNostrum4, Frioul, DAVIDE, and Sisu.

- Sisu [37] is a Cray XC40 system at CSC, similar to Hazel Hen at HLRS with dual 12-core x86 Haswell CPUs (Intel Xeon E5-2690v3), and an Aries interconnect using a Dragonfly topology.

GPAW is mainly compute bound for all the benchmarks, but generates also periodically high levels of MPI communication, especially for the benchmarks *Case M: Copper filament* and *Case L: Silicon cluster*. Memory usage of GPAW is also quite high, setting limits to the minimum number of nodes needed for the benchmarks.

4.5.1 Performance Results

Nodes	Sisu	JUWELS	MareNostrum4	Piz Daint	DAVIDE / P100	DAVIDE / POWER8	Frioul
1	239.1	111.2	136	566	114.3	517.7	527.3
2	136.7	58.9	73.3	305.8	77.3	282.8	307.2
4	78.9	32.5	44	175.7	56.2	263.5	187.3
8	44.3	20.8	30.0	106.3	46.3	84.7	140.8
16	31.5	14.2	21.2	70.3		172.6	114.8
32	21.9	11.3	22	54.8			118.3
64	19.0			50			
128				43.2			
256				41.5			
512				41.6			

Table 41: Total runtime (in seconds) for benchmark Case S: Carbon nanotube

Nodes	Sisu	JUWELS	MareNostrum4	DAVIDE / POWER8	Frioul
1		1821	2073	15595	456.5
2		1059.3	1213.9	7891	214.8
4	1531	623.0	740	3928	128.7
8	932.2	309.2	354.0	2087	72.0
16	439.0	159.3	208	3477	49.5
32	251.1	92.4	130		36.0
64	150.5	58.2	87		
128	93.4	38.8	64		
256	60.6	31.6	56		
512	42.0	29.7			
768	42.6				

Table 42: Total runtime (in seconds) for benchmark Case M: Copper filament

Nodes	Sisu	JUWELS	MareNostrum4	DAVIDE / POWER8
1				
2				26290
4		1986		17720
8		936	1264	15734
16	1217	486.1	628	11049
32	646.5	267.8	376	
64	377.1	189.6	269	
128	226.4	156.2	183	
256	140.3			
512	115.4			
768	123.7			

Table 43: Total runtime (in seconds) for benchmark Case L: Silicon cluster

Nodes	Sisu	JUWELS	MareNostrum4
32		1592	2043
64	2398	888	1214
128	1328		633
256	684		
512	502		

Nodes	Sisu	JUWELS	MareNostrum4
768	375.9		

Table 44: Total runtime (in seconds) for benchmark Case L: Silicon cluster using a larger system with a radius of 20Å

4.5.2 Performance Cross comparison

Skylake vs. Haswell Based on the benchmark runtimes on different systems (Table 41, Table 42, Table 43, and Table 44), one can conclude that compared to the older Haswell CPUs, the next generation Skylake CPUs are roughly twice as fast, with speed-ups ranging from no speed-up to three times faster. On JUWELS, the performance increase is consistently more than two times until the scalability limits for the benchmarks are reached.

KNL vs. Haswell Looking at the data for the Frioul system, Knights Landing (KNL) MICs are significantly slower than the same generation Haswell CPUs. Scalability is also worse on KNLs, which is actually not surprising considering that GPAW uses only MPI for parallelisation. Since the number of cores, and thus MPI tasks, is larger on the KNL, this generates a higher communication load on the node that could only be mitigated by not using all cores, which would of course then lower the overall performance.

Interestingly, next generation Skylake CPUs have a similar number of cores per node, but are better able to handle the increased communication load.

POWER8 vs. Haswell The performance of POWER8 CPUs seems to depend greatly on the benchmark in question. For *Case S: Carbon nanotube* (Table 41), POWER8 CPUs offer a similar level of performance than KNLs do. But for *Case M: Copper filament* (Table 42), the performance of POWER8 CPUs is an order of magnitude lower. It is likely that this is due to the higher MPI load generated by the latter benchmark.

P100 vs. Skylake The performance of a single P100 GPU (on a POWER8 host CPU) seems to be similar to a node with two 24-core Skylake CPUs (Table 41). When using multiple GPUs, the performance does not scale as well as, though. The execution model of the experimental GPGPU-version of GPAW is such that for each GPU only a single host CPU core is used. On one hand, this means that the majority of computing is done on the GPU, and on the other hand, that the MPI communication load is significantly lower per node than e.g. when using only the host CPUs as was done for the POWER8 CPU results. Nevertheless, despite the lower MPI load, the scalability is still not as good as e.g. on JUWELS or MareNostrum4.

JUWELS vs. MareNostrum4 A comparison of the two Skylake systems (JUWELS and MareNostrum4) gives an interesting chance for direct comparison between roughly similar systems. The CPUs on JUWELS have a slightly higher frequency (2.7 GHz) than those on the MareNostrum4 (2.1 GHz). The two systems also have different interconnects, with an Intel Omni-Path on MareNostrum4 and a Mellanox EDR-InfiniBand on JUWELS.

The single-node performance difference of the two systems is approximately equivalent to the slightly higher CPU frequency on JUWELS. When using multiple nodes, the performance difference increases in favour of JUWELS. Since single-node performance was roughly equivalent (if one takes into account the CPU frequencies), this likely means that the interconnect on JUWELS is better suited for the MPI communication load from GPAW.

Piz Daint vs. DAVIDE Another possibility for direct comparison is offered by the two P100 GPGPU systems (Piz Daint and DAVIDE). A major difference between the systems is that on Piz Daint each node has a single GPU and a single Haswell host CPU whereas on DAVIDE each node has four GPUs and two POWER8 host CPUs. In other respects, the two systems are also quite different, e.g. Piz Daint has an Aries interconnect in a Dragonfly topology and on DAVIDE the interconnect is an InfiniBand EDR network.

Since only a single CPU core is used per GPU in the current GPGPU-version of GPAW, this means that the number of GPUs is the dominant factor when comparing the performance between systems. Even after taking into account the fact that each DAVIDE node is the rough equivalent of four Piz Daint nodes (4 × P100 on DAVIDE compared to Piz Daint), it is clear from Table 41 that Piz Daint has lower overall performance than DAVIDE for GPAW. Since the host CPUs on Piz Daint have better performance, this is most likely due to communication overhead coming either from differences in the interconnect or from a less dense packing of the allocated nodes. It is possible, that by requesting a more densely packed allocation of nodes (similar to DAVIDE), the performance difference could be mitigated.

4.5.3 Energy consumption

PRACE PCP prototypes (DAVIDE and Frioul) have built-in energy measurement hardware and tools to estimate the energy usage of jobs. These tools were used to estimate the total energy consumption of two benchmarks (Table 45).

For **KNLs** (on Frioul), the energy consumption seems to be a more or less linear function of the total usage of processors, i.e. the number of nodes used times the runtime. There is a slight improvement in the coefficient of the linear function for larger number of nodes, but since the total energy consumption is nevertheless growing, the most optimal energy-to-solution is to use only a single KNL.

For **P100 GPGPUs** and for **POWER8 CPUs** (on DAVIDE), the total energy consumption decreases when using more nodes. In light of the fact that also the runtime decreases, it seems that for P100s and POWER8s it is beneficial to use as many nodes as possible, within scalability limits, to reach an optimal energy-to-solution.

Nodes	Case S: Carbon nanotube			Case M: Copper filament
	DAVIDE / P100	DAVIDE / POWER8	Frioul	Frioul
1	91.9	451.5	153.8	159.3
2	72.2	226.3	242.5	204.9
4	52.6	245.7	306.5	255.0
8		79.5	490.8	352.2
16			826.1	614.9
32			1800	1200

Table 45: Total energy consumption (in kJ) for benchmarks Case S: Carbon nanotube and Case M: Copper filament in PRACE PCP prototypes

4.6 GROMACS

GROMACS is a package that uses multilevel optimisations. The main difference from other codes is the use of low-level CPU accelerated kernels. For mainstream processors the computationally expensive parts are written using specific CPU features like SSE, AVX, VSX etc. Of course, the catch-up C version of kernel is still present. The comparison between C and CPU accelerated builds lead to a speedup of 4–10, depending on the CPU features. Its memory footprint is quite small compared to other Molecular Dynamics packages.

It uses hybrid MPI/OpenMP parallelisation. GROMACS does domain decomposition on MPI tasks and can use OpenMP on each cell. 3D domain decomposition quality is sensitive on the number of MPI tasks, for example on 68 KNL cores this can be done as 2×2×17 that lead in many cases to unbalanced distribution of computation and in some cases, it rejects to run. This is the reason why in the reported results on KNL we used 64 instead of 68 cores/node. Using hybrid MPI/OpenMP parallelisation, one can achieve better domain decomposition and use OpenMP on each cell. Parallelisation using OpenMP is efficient when using small number of threads per task, in general in the range 1–10. At high numbers of nodes, using hybrid MPI/OpenMP parallelisation scheme, we may extend its scalability when pure MPI scalability is exhausted, by decreasing the number of MPI packets/increasing the packet size. It also has pinning capabilities.

For NVIDIA GPUs, it includes the corresponding CUDA kernels to offload work on GPU. Speedup from GPUs is not as high as with other packages because it is already CPU optimised. In most cases, speedup using GPUs is analogous to the theoretical performance ration of the GPU/CPU.

On all systems, trial runs with some combinations of MPI tasks/threads task were performed using small, intermediate and high number of nodes in order to find the combination that yields the higher performance as function of number of nodes on each system. In results tables, only the combination that yields the higher performance is reported.

4.6.1 Performance on KNL systems

GROMACS ran on three KNL systems, Frioul, Irene-KNL and Marconi-KNL with their own characteristics like KNL memory configuration, interconnect and batch system limitations. Marconi-KNL seems to be faster when using 1 or 2 nodes, but scalability is limited at higher number of nodes. Frioul system had limited number of nodes with the same memory configuration available during benchmarks and although its performance seems good there is not sufficient data to have a clear picture of scalability. On the other hand, Irene-KNL had limitation on the number of tasks/threads per node, but reached higher performance, with efficiency 55.3% at 64 nodes. A direct comparison of three KNL systems using 16 nodes, shows large deviation in performance between systems.

Frioul				Irene-KNL				Marconi-KNL			
Nodes	**Tasks/ Node**	**Threads/ Task**	**Performance [ns/day]**	**Nodes**	**Tasks/ Node**	**Threads/ Task**	**Performance [ns/day]**	**Nodes**	**Tasks/ Node**	**Threads/ Task**	**Performance [ns/day]**
1	66	4	1.84	1	64	1	1.57	1	136	2	2.05
2	66	4	3.57	2	64	1	2.94	2	136	2	3.62
6	66	4	9.33	4	64	1	5.56	4	136	2	5.97

Frioul				Irene-KNL				Marconi-KNL			
Nodes	Tasks/ Node	Threads/ Task	Performance [ns/day]	Nodes	Tasks/ Node	Threads/ Task	Performance [ns/day]	Nodes	Tasks/ Node	Threads/ Task	Performance [ns/day]
8	66	4	12.05	8	64	1	10.08	8	136	2	8.82
16	66	4	20.10	16	64	1	10.00	16	68	4	12.67
20	66	4	24.40	32	64	1	16.42	32	34	8	11.62
				64	64	1	37.50	64	68	4	11.89
				128	64	1	55.60				

Table 46: GROMACS Performance on KNL systems (in ns/day units) for Test Case B: Lignocellulose

4.6.2 Performance on GPU accelerated systems

Benchmarks with GPUs were performance on two machines, the POWER8/P100 with NVLink based DAVIDE and Haswell/P100/PCIe Piz Daint. On DAVIDE, initially machines were configured with SMT off. Results from previous PRACE implementation phases exhibit significant performance gain when using SMT with hybrid applications. Machines were configured for some period with SMT on, where the results were obtained. For comparison, performance and energy to solution are reported with both SMT on and off. Using up to 16 nodes, DAVIDE with four P100/node is roughly 5–20% faster than Piz Daint of one P100/node. Performance on DAVIDE without SMT is significantly lower (~60%) with respect the performance with SMT. The efficiency with 16 nodes is 51.4% for DAVIDE and 69.8% for Piz Daint. GROMACS performance is increasing up to 800 nodes although with decreasing efficiency.

DAVIDE 16 physical cores/Node, SMT=4				Piz Daint 12 Physical Cores/Nodes, HT=ON			
Nodes	Tasks/Node	Threads/Task	Performance [ns/day]	Nodes	Tasks/Node	Threads/Task	Performance [ns/day]
1	16	4	3.90	4	12	2	10.88
2	16	4	6.07	8	6	4	13.49
4	16	4	12.50	16	12	2	30.40
8	16	4	20.84	32	12	2	55.94
16	16	4	32.11	64	12	2	83.29
32	16	4	52.09	128	6	4	99.79
				300	12	2	128.31
				800	12	2	143.05

Table 47: GROMACS Performance on systems with GPUs (in ns/day units) for Test Case B: Lignocellulose

4.6.3 Performance on Haswell/Skylake systems

There are four systems with different interconnect technologies and CPUs. Hazel Hen is Haswell based with Aries network, while the rest are based on Skylake and also different interconnect and CPU settings – Hyperthreading and probably power management. On a single node, JUWELS with Hyperthreading enabled has higher performance, MareNostrum4 and Irene-SKL have comparable performance taking into account the different frequency of CPUs. Older Haswell of Hazel Hen have less than half the performance of Skylake Machines. At 128 nodes the efficiency drops down to 48.1%, 30%, 32.8%, 29.8% from Hazel Hen, MareNostrum4, Irene-SKL and JUWELS respectively. Finally, the DEEP SDV Haswell have similar single node performance to this of Hazel Hen. Scaling on DEEP SDV is not comparable to this of same CPU Hazel Hen.

Hazel Hen: 24 Physical Cores/Node, HT=ON				MareNostrum4: 48 Physical Cores/Node, HT=OFF			
Nodes	**Tasks/Node**	**Threads/Task**	**Performance [ns/day]**	**Nodes**	**Tasks/Node**	**Threads/Task**	**Performance [ns/day]**
1	24	2	1.182	1	12	4	2.384
2	24	2	2.329	2	12	4	4.43
4	24	2	4.542	4	12	4	8.125
8	24	2	8.711	8	12	4	15.11
16	24	2	14.582	16	24	2	27.089
32	24	2	24.252	32	24	2	44.26
64	24	2	44.762	64	12	4	63.646
128	24	2	72.841	128	48	1	94.493
256	24	2	125.076				
320	24	2	129.302				
Irene-SKL: 48 Physical Cores/Node, HT=OFF				**JUWELS: 48 Physical Cores/Node, HT=ON**			
1	8	6	2.761	1	48	2	3.288
2	8	6	5.363	2	48	2	6.408
4	8	6	10.244	4	48	2	12.328
8	8	6	19.112	8	48	2	22.186
16	8	6	33.255	16	24	4	40.06
32	8	6	57.876	32	12	8	57.55
60	8	6	92.095	64	24	4	100.528
128	12	4	116.108	128	12	8	125.491
192	12	4	164.861	256	12	8	101.508
384	12	4	205.703				
DEEP-ER SDV: 24 Physical Cores/Node, HT=ON							
1	24	2	1.095				
2	24	2	1.995				
4	24	2	3.244				
8	24	2	4.848				
16	24	2	5.152				

Table 48: GROMACS Performance on x86 systems (in ns/day units) for Test Case B: Lignocellulose

4.6.4 Energy consumption

PRACE PCP prototypes DAVIDE and Frioul have built-in high frequency energy measurement hardware and tools to estimate the total energy usage of all involved components (Node, network and storage).

Since parallel efficiency decreases by increasing the number of nodes for both machines, the energy to solution is also increasing. The energy to solution for Frioul and DAVIDE with SMT=4 is very close for the same number of nodes except single node, On the other hand, the performance on DAVIDE with SMT on is roughly double of this of the same number of Frioul nodes. This indicates that with DAVIDE, consuming the same amount of energy we have the solution in half of time. Without SMT on DAVIDE, the energy to solution is roughly 50% higher than with SMT enabled.

DAVIDE					Frioul		
Nodes	**SMT=1**		**SMT=4**		**Nodes**	**Performance [ns/day]**	**Energy [kJ]**
	Performance [ns/day]	**Energy [kJ]**	**Performance [ns/day]**	**Energy [kJ]**			
1	2.36	641.6	3.90	436.0	1	1.48	829.7

DAVIDE					Frioul		
Nodes	SMT=1		SMT=4		Nodes	Performance [ns/day]	Energy [kJ]
	Performance [ns/day]	Energy [kJ]	Performance [ns/day]	Energy [kJ]			
2	4.55	667.4	6.07	554.5	4	4.89	533.9
4	9.25	682.9	12.50	508.9	8	9.43	603.5
8	14.13	900.4	20.84	620.9	16	14.18	817.4
16	21.46	1264.1	32.11	859.2	32	22.35	1200.0
32	38.54	1723.0	52.09	1180.0	48	29.25	1700.0
40	39.77	2186.5					

Table 49: Performance and total energy consumption (in kJ) for GROMACS benchmarks Case B: Lignocellulose, on PRACE PCP prototypes

4.7 NAMD

NAMD is a molecular dynamics package for mainly biomolecular systems. It supports parallelisation with MPI, Threads, CUDA for GPUs, as well as other architectures. It has a large memory footprint, and uses a dynamic load balancer to redistribute computation among tasks. In PRACE benchmarks the special memory optimised build was used that enables the simulation of very large systems without the need for huge amounts of memory in the master process. The input data for PRACE benchmark are 2.3 GB in size. The initialisation stage except read/distribute input data, optimise FFTW parameters based on input data, setups load balancer etc. The initialisation time is usually very short – order of few seconds, but on some systems, it was found to be of order of one minute. Also, this time happens to have large deviations on the same system. In the reported data, the setup time, that is reported by NAMD is subtracted and the pure simulation time was taken into account. For each machine a number of trial runs was performed using small, intermediate and large number of nodes in order to obtain the combination of MPI tasks/ threads per task that yields the best performance. It should be noted that in the hybrid version, one thread is used as controller while the rest of the threads are used for computation. This means that with N threads / task this results to N – 1 threads for computation.

4.7.1 Performance on KNL systems

NAMD ran on two KNL systems, Frioul and Marconi-KNL with their own characteristics like KNL memory configuration. Both systems have similar performance with Frioul being slightly faster for the same number of nodes.

Frioul				Marconi-KNL			
Nodes	Tasks/Node	Threads/Task	Wall Time [sec]	Nodes	Tasks/Node	Threads/Task	Wall Time [sec]
2	4	64	55607.1	2	4	64	65889.2
4	4	64	30210.7	4	4	64	34234.6
8	4	64	17340.0	8	4	64	17720.9
16	4	64	10144.6	16	4	64	9224.7
32	64	4	6479.4	32	64	4	7053.6
64	64	4	5262.1	64	64	4	3232.3
				128	64	4	1934.8
				192	64	4	1595.2

Table 50: NAMD Execution Time on KNL systems (in seconds) for Test Case B: STMV.28M

4.7.2 Performance on GPU accelerated systems

Benchmarks with GPUs were performance on two machines, the POWER8/P100 with NVLink based DAVIDE and Haswell/P100/PCIe Piz Daint. On DAVIDE, initially machines were configured with SMT off. Results from previous PRACE implementation phases exhibit significant performance gain when using SMT with hybrid applications. Machines were configured for some period with SMT on, where the results obtained.

NAMD benchmark case B is large enough and does extensive GPU use. DAVIDE with four P100 is significantly faster compared to the same number of nodes of Piz Daint. Scaling is almost linear up to the available 40 DAVIDE nodes.

DAVIDE 16 physical cores/Node, SMT=4				Piz Daint 12 Physical Cores/Nodes, HT=ON			
Nodes	**Tasks/Node**	**Threads/Task**	**Wall Time [sec]**	**Nodes**	**Tasks/Node**	**Threads/Task**	**Wall Time [sec]**
6	16	4	2408.1	4	1	12	12433.4
8	16	4	1826.8	8	1	12	5227.5
12	16	4	1260.4	16	1	12	2907.1
16	16	4	1078.3	32	1	12	2431.2
24	16	4	608.4	64	1	12	783.7
32	16	4	529.7	128	1	12	807.2
40	16	4	484.3	256	1	12	586.8

Table 51: NAMD Execution Time on systems with GPUs for Test Case B: STMV.28M

4.7.3 Performance on Haswell/Skylake systems

On Haswell/Skylake systems, NAMD case B exhibits almost linear scaling up to the available number of nodes on each system except JUWELS, where scaling drops down when using more than 64 nodes. Performance is comparable taking into account the differences.

Hazel Hen: 24 Physical Cores/Node, HT=ON				MareNostrum4: 48 Physical Cores/Node, HT=OFF			
Nodes	**Tasks/Node**	**Threads/Task**	**Wall Time [sec]**	**Nodes**	**Tasks/Node**	**Threads/Task**	**Wall Time [sec]**
32	2	24	2944.4	2	2	24	44195.7
64	2	24	1545.3	4	2	24	21815.4
128	2	24	792.3	8	2	24	11041.7
192	2	24	557.7	16	2	24	5683.5
256	2	24	421.6	32	2	24	2937.5
320	2	24	344.9	64	2	24	1501.2
400	2	24	303.1	128	2	24	808.8
				256	2	24	443.5
				320	2	24	367.8
				400	2	24	297.5
Irene-SKL: 48 Physical Cores/Node, HT=OFF				**JUWELS: 48 Physical Cores/Node, HT=ON**			
4	8	6	15994.0	1	4	24	43660.5
16	8	6	4355.7	2	4	24	21222.1
32	8	6	2249.5	4	4	24	10478.1
64	8	6	1073.3	8	4	24	5534.6
128	8	6	548.8	16	4	24	2671.3
216	8	6	343.4	32	4	24	1395.3

				64	4	24	906.8
				96	4	24	914.2
				144	4	24	951.9
DEEP-ER SDV 24 Physical Cores/Node, HT=ON							
4	2	12	36165.1				
8	2	12	25713.1				
16	2	12	17705.3				

Table 52: NAMD Performance on x86 systems for Test Case B: STMV.28M

4.7.4 Energy consumption

PRACE PCP prototypes DAVIDE and Frioul have built-in high frequency energy measurement hardware and tools to estimate the total energy usage of all involved components (Node, network and storage). Since parallel efficiency decreases by increasing the number of nodes for both machines, the energy to solution is also increasing.

Performance on DAVIDE with SMT on is significantly higher on DAVIDE compared to Frioul. This results to a significantly lower energy to solution on DAVIDE.

DAVIDE (SMT=4)			**Frioul**		
Nodes	**Wall Time [sec]**	**Energy [kJ]**	**Nodes**	**Wall Time [sec]**	**Energy [kJ]**
16	1078.3	19434.9	2	55607.1	38700
32	608.4	20224.8	4	30210.7	41200
40	529.7	22896.6	8	17340.0	46200
			16	10144.6	52300
			32	6479.4	65200
			64	5262.1	97400

Table 53: Performance and total energy consumption (in kJ) for NAMD benchmark Case B: STMV.28M, on PRACE PCP prototypes

4.8 NEMO

Comparative benchmarking of NEMO has been performed on homogenous CPU only system comprising of JUWELS, MareNostrum4 and Hazel Hen. Since Irene-SKL and JUWELS are similar in nodal configuration and interconnect, we skipped the Irene-SKL and also were not able to use SuperMUC-NG because of its unavailability until April 2019.

We also committed benchmarking NEMO on DEEP-ER SDV and Dibona but did not proceed further due to following reasons:

DEEP-ER SDV booster modules consist of Intel Xeon Phis and we decided not to benchmark NEMO on Xeon Phi or any other accelerator type of nodes. Since we did not utilise booster nodes and also wanted to benchmark NEMO on a 1024 CPU core count to make results comparable on different systems, this was not possible since the highest core count available on DEEP-ER SDV cluster partition is 768 and therefore we did not proceed with DEEP-ER SDV.

4.8.1 Installation

We have installed NEMO version 3.6 with XIOS 2.0 on JUWELS and MareNostrum4 using the Intel compiler toolchain and NEMO version 3.6 with XIOS-1.0 on Hazel Hen also with the Intel compiler tool chain.

4.8.2 Performance Results

Test Case A

SubDomain size per core	CPU Cores	Average Time(s) per computational step and relative speedup					
		MareNostrum4	Speed Up	JUWELS	Speed Up	Hazel Hen	Speed Up
122 × 162	512	1.15	1	1.14	1	0.99	1
122 × 82	1024	0.59	1.94	0.58	1.96	0.51	1.97

Table 54: NEMO Test Case A performance

Test Case B

Subdomain size per core	CPU Cores	Average Time(s) per computational step and relative speedup					
		MareNostrum4	Speed Up	JUWELS	Speed Up	Hazel Hen	Speed Up
122 × 82	4092	0.62	1	0.55	1	0.51	1
61 × 82	8192	0.39	1.5	0.29	1.9	-	-
62 × 42	16384	0.22	2.8	0.15	3.6	-	-

Table 55: NEMO Test Case B performance

4.8.3 Performance Cross Comparison

The three-systems involved (JUWELS, MareNostrum4 and Hazel Hen) represent the widespread system characteristics involving latest generation Intel CPUs and modern interconnect technologies (Intel Omni-Path, Mellanox InfiniBand EDR, and Aries with Dragonfly Topology).

NEMO is a memory bound code [25] and also involves a lot of point to point and collective communication due to underlying numerical constructs. NEMO is a representative of some of the MPI applications which are heavily influenced by memory subsystem and interconnect performance especially when experiments are conducted on a very large number of cores.

The PRACE best practice guide for modern interconnects [23] shows theoretical maximum nodal bandwidth for JUWELS, Hazel Hen and MareNostrum4. The numbers are summarised in Table 56.

	JUWELS – Intel Omni-Path	MareNostrum4 – InfiniBand EDR	Hazel Hen – Dragonfly Aries
Theoretical Max Bandwidth (GB/s)	12.8	12.8	15

Table 56: Theoretical Max Bandwidth (GB/s)

According to Table 56 there exists a clear trend for maximal theoretical bandwidth per node where Hazel Hen has the highest performance and this combination with modest CPU clock frequency should provide the relatively better performance for memory and communication bound applications. Also, [24] documents the detailed performance comparison of different interconnects for latency and bandwidth. The authors conclude that for latency and bandwidth the Aries interconnect outperforms both InfiniBand EDR and Intel Omni-Path with upper hand for InfiniBand EDR compared to Intel Omni-Path. We should perhaps see qualitatively similar trends in performance for NEMO as well.

Results from test case A show quite similar performance characteristics with slightly upper hand for JUWELS and for Hazel Hen. The average time taken for each computational step is lowest for Hazel Hen and the relative speed-up also shows a very similar trend. Since the speedup can only help us to evaluate application scalability on a single system, its use for comparing performance with other system should not be practiced, therefore it is important to not only look at the speedup but also time taken on different systems.

Results from test case B depicts scalability and time perspective up to 16000 cores. In some cases, 16000 cores represents almost a third of the full production system under consideration. On 16384 cores and 8192 cores we can see a clear trend, NEMO on JUWELS achieve relative speedup of 3.6 and 1.9 compared to 2.8 and 1.5 on MareNostrum4. The time taken on JUWELS is less than that on MareNostrum4 and this is perhaps due to a better interconnect performance on JUWELS since nodal performance is almost identical. We can also see time taken for test case B on Hazel Hen for 4096 cores and it outperforms other system in comparison.

Since NEMO is a memory bound application, we can clearly see that throwing more computational power per node in this case (MareNostrum4 and JUWELS) does not help but a better interconnect and a balanced system does. Summarizing, in case of NEMO benchmarking, the above-mentioned trends in interconnect performance clearly augment the experiments conducted here on three representative systems and support the numbers of system performance.

4.9 PFARM

PFARM benchmark runs were undertaken on PRACE Tier-0 and PCP prototypes. The parallelisation and test cases are described in Section 2.9

The performance of PFARM from the four main CPU architectures benchmarked – Intel Xeon Skylake, Intel Xeon Phi Knights Landing, Intel Xeon Haswell/NVIDIA P100 and ARM ThunderX2 is analysed in this section. The compilers used, compiler optimisation flags and numerical libraries used are summarised Table 57.

Machine	Compiler	Compiler Optimisation Flags	Numerical Libraries
JUWELS	Intel Fortran v2019.0.117	-mtune=skylake -mkl=parallel -Ofast	Intel MKL v2019.0.117
Marconi	Intel Fortran v2018.0.5	-xMIC-AVX512 -mtune=knl -mkl=parallel -Ofast	Intel MKL v2018.5.274
Piz Daint	Cray Fortran v8.7.3	-O3	Magma 2.3.0 & Cray Libsci v18.07.1 (third-party)

Machine	Compiler	Compiler Optimisation Flags	Numerical Libraries
Dibona	ARM Fortran Compiler v 19.0	-Ofast	ARM Performance Libs 19.0.0

Table 57: Summary of Programming Environments

4.9.1 Performance Results

The PFARM EXDIG MPI/OpenMP code is designed to match individual sector calculations to nodes in order to minimise inter-node communication and maximise the number of threads available for computation. One MPI task is assigned to each compute node. The number of sectors in the benchmark is therefore an upper bound on the number of nodes used in the benchmark run – 16 sectors/nodes for Test Case 1 and 64 sectors/nodes for Test Case 2. If the calculation is relatively small and memory limits permit, more than one sector calculation, and therefore more than one MPI task, can be placed on a node, but this is not a scheme investigated here. Each MPI task uses OpenMP/CUDA to parallelise the computationally intensive sector Hamiltonian eigensolvers via highly-optimised vendor-supplied numerical libraries (Table 57).

Parallel performance results for both datasets are summarised for the four main architectures tested in Table 58 and Table 59. JUWELS is the Skylake system with the highest CPU frequency (2.7 GHz) and this platform provides the fastest results overall for both test cases. The Mellanox EDR-InfiniBand network on JUWELS also contributes to the best overall parallel efficiency results, though inter-node communication in the PFARM parallelisation is designed to be very low. The ability of the GPFS parallel filesystem to handle multiple simultaneous outputs of large volumes of data from multiple nodes is of more importance to parallel efficiency here. All the modern filesystems tested here perform relatively well in this respect. However, for Test Case 2 all systems show a marked reduction in parallel efficiency going from 32 nodes (~90% to 98%) to 64 (~77% to 87%) nodes.

Number of nodes	Number of cores	Time (s)	SpeedUp	Efficiency	Time (s)	SpeedUp	Efficiency
		JUWELS (Intel Skylake)			Marconi (KNL)		
1	48	2379.45	1.0	100%	5184.94	1.0	100%
2	96	1199.83	1.98	99.15%	2599.42	1.99	99.73%
4	192	594.84	4.00	100%	1307.55	3.96	99.13%
8	384	302.42	7.87	98.35%	673.28	7.70	96.26%
16	768	153.07	15.54	97.16%	355.78	14.57	91.08%

Number of nodes	Number of cores	Time (s)	SpeedUp	Efficiency	Time (s)	SpeedUp	Efficiency
		Piz Daint (Intel Haswell & NVIDIA P100)			Dibona (ARM)		
1	48	2404.05	1.0	100%	13540.67	1.0	100%
2	96	1206.64	1.99	99.15%	6773	1.99	99.96%
4	192	605.12	3.97	100%	3386.5	3.99	99.96%
8	384	305.11	7.87	98.35%	1700.97	7.96	99.50%
16	768	154.66	15.54	97.16%	853.11	15.87	99.20%

Table 58: Summary of Results from PRACE systems for full runs of the PFARM EXDIG Benchmark (Test Case 1). Runs undertaken with one compute thread per core.

Number of nodes	Number of cores	Time (s)	SpeedUp	Efficiency	Time (s)	SpeedUp	Efficiency
		JUWELS (Intel Skylake)			Marconi (KNL)		
1	48	1580.71	1.0	100%	4213.81	1.0	100%
2	96	802.45	1.97	98.49%	2155.14	1.96	97.76%
4	192	401.22	3.93	98.49%	1080.55	3.90	97.49%
8	384	203.34	7.77	97.17%	544.22	7.74	96.79%
16	768	101.28	15.61	97.55%	273.13	15.43	96.42%
32	1536	50.36	31.39	98.09%	139.60	30.19	94.33%
64	3072	28.24	55.97	87.46%	75.48	55.83	87.23%

Number of nodes	Number of cores	Time (s)	SpeedUp	Efficiency	Time (s)	SpeedUp	Efficiency
		Piz Daint (Intel Haswell & NVIDIA P100)			Dibona (ARM)		
1	48	1882.07	1.0	100%	7587.51	1.0	100%
2	96	945.26	1.99	99.55%	3820.20	1.99	99.31%
4	192	474.77	3.96	99.11%	1935.25	3.92	98.02%
8	384	239.23	7.87	98.34%	972.10	7.81	99.57%
16	768	124.98	15.05	94.12%	490.58	15.47	96.67%
32	1536	65.52	28.72	89.77%			
64	3072	37.94	49.61	77.51%			

Table 59: Summary of Results from PRACE systems for full runs of the PFARM EXDIG Benchmark (Test Case 2). Runs undertaken with one compute thread per core.

4.9.2 Detailed Performance Analysis

4.9.2.1 Timing Breakdown

A PFARM (EXDIG) calculation takes place in several distinct stages. The time spent in each stage is reported in Table 60. JUWELS is used as the example platform here and timing breakdowns do not differ significantly from platform to platform. Firstly, in the SETUP stage, H file data produced from a preceding inner region calculation is read from disk by all the nodes. This non-optimised parallel input from one file to multiple nodes is potentially a bottleneck on large node counts, but all systems cope relatively well here. The results show that the relative cost of SETUP rises from 0.02% on 1 node to 5.42% on 64 nodes. The Legendre basis functions are applied and the sector Hamiltonian matrices are filled in the stage MATRIX ASSEMBLY stage. This is a sequential cost within each node for each sector calculation and therefore its proportion of runtime remains fairly constant throughout at approximately 10%. The parallel eigensolution of the sector Hamiltonian matrix using OpenMP threads on the CPUs is undertaken in DIAG using optimised numerical library routines e.g. DSYEVD. This stage is relatively onerous and parallelises well across the range of node counts, averaging roughly 70% of runtime. This is mainly thanks to the inherent parallelism in the R-matrix calculation construction (a single node run will repeat 64 sector calculations whilst a 64 node run will undertake the compete set in parallel). The final stage AMPS is dominated by output costs from each node. Again, this follows the inherent coarse-grained parallelism and the relative cost remains fairly constant at 15–20%.

Nodes	Total Runtime (s)	SETUP (s)	SETUP (%)	MATRIX ASSEMBLY (s)	MATRIX ASSEMBLY (%)	DIAG (Threaded) (s)	DIAG (Threaded) (%)	AMPS (s)	AMPS (%)
1	1580.71	0.37	0.02	171.52	10.85	1190.04	75.29	218.78	13.84
2	802.45	0.31	0.04	86.40	10.77	594.56	74.09	121.18	15.10

Nodes	Total Runtime (s)	SETUP (s)	SETUP (%)	MATRIX ASSEMBLY (s)	MATRIX ASSEMBLY (%)	DIAG (Threaded) (s)	DIAG (Threaded) (%)	AMPS (s)	AMPS (%)
4	401.22	0.65	0.16	43.36	10.81	298.08	74.29	59.13	14.74
8	203.34	0.63	0.31	21.92	10.78	148.64	73.10	32.15	15.81
16	101.28	0.94	0.93	10.72	10.58	75.64	74.68	13.98	13.80
32	50.36	1.54	2.80	5.34	10.31	37.82	73.01	7.19	13.88
64	28.24	1.81	5.42	2.67	9.45	18.91	66.96	5.13	18.17

Table 60: Breakdown of timings within distinct computational stages of PFARM EXDIG for Test Case 2 (JUWELS)

Table 60 shows that the eigensolver calculation contributes significantly to overall computation costs. The computational complexity of the eigensolver, where both eigenvalues and eigenvectors are required is of order $O(N^3)$, where N is the dimension of the matrix. This means that as the problem size increases, the proportion of runtime reduces in stages of the code with either O(N) costs, e.g. SETUP, or $O(N^2)$ costs, e.g. MATRIX ASSEMBLY and AMPS. Table 61 shows the high proportion of runtime spent in the parallel eigensolver for Test Case 2 on the test platforms. Similar results are obtained for Test Case 1. On 64 nodes, all 64 sector Hamiltonian calculations take place simultaneously in one batch. Therefore the sequential (within a node) properties of the stages outside the eigensolver begin to impact more on overall runtime and the proportion of runtime spent in the parallel eigensolver decreases significantly on all the platforms, see Table 60 and Table 61.

Nodes	JUWELS	Marconi	Piz Daint	Dibona
1	75.31	64.12	73.21	87.41
2	74.09	64.68	73.09	88.70
4	74.29	64.57	72.66	86.81
8	73.10	64.44	72.37	86.02
16	73.97	64.32	70.96	85.81
32	75.42	63.84	67.40	
64	66.96	62.07	58.38	

Table 61: Percentage of total runtime in the eigensolver routine DSYEVD (Test Case 2)

4.9.2.2 Intra-Node Parallel Performance

Evidently, eigensolver performance is highly important to PFARM EXDIG efficiency. In this version of EXDIG it is the intra-node performance of the eigensolver routine DSYEVD that determines performance. The parallel eigensolver performance within nodes for Test Case 2 for different numerical libraries – MKL, ARM PL – on three different CPUs is shown in Table 62. Based on the optimal number of threads, intra-node parallel efficiency is fairly constant for the different CPUs (~35%). On JUWELS and Frioul, the optimal thread count corresponds to the maximum number of physical cores. However, on Dibona, using 32 threads per node is faster than using 64 and this configuration is generally preferred.

Only the DAVIDE benchmark platform provides multiple GPUs per node. MAGMA eigensolvers are currently preferred to CuSolver eigensolvers as these can automatically parallelise across GPUs on a node. Table 63 shows performance results on a single DAVIDE node with 1 – 4 GPUs for a

DSYEVD eigensolver routine from MAGMA using Test Case 2 data. Parallel efficiency is approximately 30% using 4 GPUs.

JUWELS (Intel Skylake 48 cores)				**Frioul (**Intel KNL 64 cores)			
Threads (1 per core)	**DSYEVD (MKL) (s)**	**Efficiency (%)**	**Speed-up**	**Threads (1 per core)**	**DSYEVD (MKL) (s)**	**Efficiency (%)**	**Speed-up**
1	300.15	100.00	1.00	1	1037.72	100.00	1.00
2	189.00	79.40	1.59	2	598.23	86.73	1.73
4	90.16	83.23	3.33	4	300.00	86.48	3.46
8	49.74	75.43	6.03	8	139.87	92.74	7.42
12	35.79	69.89	8.39	16	75.75	85.62	13.70
24	22.78	54.90	13.18	32	46.99	69.01	22.08
48	18.38	34.02	16.33	64	42.90	37.80	24.19

Threads	**Dibona: ARM ThunderX2 with 64 cores**		
	DSYEVD (ARM PL) (s)	**Efficiency %**	**Speed-up**
1	1320.07	100.00	1.00
2	714.20	92.42	1.85
4	394.33	83.69	3.35
8	219.15	75.29	6.02
16	144.47	57.11	9.14
32	120.85	34.14	10.92
64	125.40	16.45	10.53

Table 62: Single node parallel eigensolver performance on CPUs (Test Case 2)

Nodes × GPUs	**DAVIDE: Intel Haswell with 4 P100 cards**		
	DSYEVD (MAGMA) (s)	**Efficiency %**	**Speed-up**
1 × 1	3305.98	100.00	1.00
1 × 2	2874.26	57.51	1.15
1 × 4	2556.89	32.32	1.29

Table 63: Single node parallel eigensolver performance on DAVIDE with multi GPU acceleration (Test Case 2)

4.9.3 Energy Consumption

Energy monitoring tools have been used to collect power consumption data on DAVIDE and Piz Daint, which are both GPU-accelerated platforms. The tool was unavailable on Frioul when these benchmark runs were undertaken. These energy consumption results are shown in Table 64. Reported energy costs on Piz Daint are fairly constant with node count, whilst on DAVIDE they vary quite widely with node count. Test Case 1 generally consumes more energy than Test Case 2 – this is to be expected, as the problem size is larger and the runs take longer. The four node DAVIDE value looks somewhat of an outlier. Restricted access to larger node counts on DAVIDE meant that repeated tests for verification were not possible.

Nodes	Test Case 1		Test Case 2	
	DAVIDE	Piz Daint	DAVIDE	Piz Daint
	Total Energy (kJ)	Total Energy (kJ)	Total Energy (kJ)	Total Energy (kJ)
1	2361.49	347.10	1627.99	257.92
2	1944.36	342.51	829.72	254.15
4	497.53	352.48	968.06	250.85
8		357.44		261.54
16		378.48		284.45
32		347.10		301.27
64		342.51		420.36

Table 64: Energy Consumption comparison

4.10 QCD

The QCD Benchmark runs are performed on all PRACE Tier-0 systems, except Hazel Hen and in case of part 2 Irene-KNL, the PCP Prototypes Frioul and DAVIDE, the DEEP-ER SDV system and the Mont-Blanc 3 Dibona system.

4.10.1 Performance Results for QCD part 1

In Table 65 we show strong scaling results for the QCD benchmark kernel part 1. The kernel scales very well for the Skylake machines on up to 128 nodes. For larger volumes the local lattice size becomes too small and communications dominate the kernel. For the case of accelerated systems with NVIDIA Pascals and Intel KNLs (with the exception of Marconi-KNL) the strong scaling is not as good. In the case of Piz Daint, the communication bottleneck begins to dominate starting from 32 nodes, while for the KNLs on Irene, good scaling is observed up to 128 nodes. Interestingly, the total performance per node on Mont-Blanc 3 is similar to that observed on Skylake systems.

Nodes	Irene-KNL	Irene-SKL	JUWELS	Marconi-KNL	MareNostrum4	Piz Daint	DAVIDE	Frioul	DEEP-ER	Mont-Blanc 3
1	155.94	219.68	182.49	133.38	186.4	53.73	53.4	151	656.41	206.17
2	81.87	114.22	91.83	186.14	94.63	32.38	113	86.9	432.93	93.48
4	48.01	58.11	46.58	287.17	47.22	19.13	21.4	52.7	277.67	49.95
8	26.83	32.09	25.37	533.49	25.86	12.78	14.8	36.5	189.83	25.19
16	15.32	14.35	11.77	1,365.7	11.64	9.2	10.1	27.8	119.14	12.55
32	8.83	7.28	5.43	2,441.2	5.59	6.35	6.94	15.6		
64	7.18	4.18	2.65		2.65	6.41		11.7		
128	5.48		1.39		2.48	5.95				
256			1.38			5.84				
512			0.89							
	MPI=1 omp=64	MPI=8 omp=6	MPI=8 omp=6	MPI=1 omp=64	MPI=8 omp= 6		4 GPUs	MPI=1 omp=68	MPI=4 omp=3	MPI=64 omp=1

Table 65: Time-to-solution of benchmark kernel part 1, given in seconds, for lattice size V=8×64×64×64

4.10.2 Performance Results for QCD part 2

For the kernel in part 2, we use two different problem sizes, namely: V=96×32×32×32 shown in Table 66 and V=128×64×64×64 shown in Table 67. We use three different software packages which implement the same functionality but optimised for different target systems. For NVIDIA

GPU machines, we use QUDA, for the Intel machines, we perform the tests using the QPhiX library, while for the ARM system, we utilise GRID. As mentioned, QUDA and QPhiX apply the Wilson Dirac operator within a CG solver, while the GRID benchmark only applies the operator, without additional linear algebra operations included in the CG. We note that GRID also compiles for Intel architectures, and after comparing its performance with QPhiX on JUWELS we found a similar performance pattern for parallelisations up to 512 nodes, from which we conclude that the performance results for Mont-Blanc 3 are representative and can be used to compare the performance of the different HPC systems.

For the smaller test size of V=96×32×32×32, the performance on the Skylake machines scales very well until 64 nodes. For larger partitions the local lattice size becomes too small, the communications dominate, and the scaling stagnates. Here we also observe large fluctuations of the performance of up to 60%. In case of the GPU machines, we see that scaling on Piz Daint starts to flatten-out after 4 nodes, while on DAVIDE it continues up to 32 nodes when using one GPU per Node. This shows that NVLINK is able to sustain the strong scaling for a larger number of nodes. The strong scaling of the KNL machines flattens-out after 8 nodes, and we observe large fluctuations for all parallelisations up to 40%. The scaling on DEEP-ER and Mont-Blanc 3 is good while the performance of Mont-Blanc 3 is comparable to the Skylake machines.

For the lager test size of V=128×64×64×64 we perform strong scaling tests on larger PRACE Tier-0 systems. We found for the Skylake systems a good scaling on up to 512 nodes, although the performance results start to have larger deviations from around 128 nodes especially on Irene. On Piz Daint, the scaling is weaker however the total performance per node is larger compared to the Skylake nodes on up to 128 nodes. The KNL machines show scaling on up to 16 nodes, however with larger fluctuations and not exceeding a performance of around 4100 Gflop/s.

Overall, we found that JUWELS shows the best scaling results with moderate fluctuations for larger partitions starting from 128 nodes and reaching a maximal performance of around 41 Tflop/s on 512 nodes using double precision. Nevertheless, for smaller partitions with up to 128 nodes, Piz Daint shows the best total performance per node, with up to 13 Tflop/s on 128 nodes using double precision.

In Table 68 we show the performance and energy consumption on the PCP prototypes, DAVIDE and Frioul. The timings are for two CG application in case of QUDA given by 24 iteration each and five CG application in case of QPhiX given by 250 iteration each. For the runs on DAVIDE 4 GPUs per node were used, while the runs on Frioul are performed on one KNL CPU in flat mode per node.

Nodes	Irene-SKL	JUWELS	Marconi-KNL	Mare-Nostrum	Piz Daint	DAVIDE	Frioul	DEEP-ER	Mont-Blanc 3
1	135.84	132.25	152.11	142.44	387.66	392.76	184.73	41.78	99.64
2	241.34	245.77	264.59	264.56	755.31	773.90	269.71	40.77	214.55
4	451.61	457.74	393.37	486.43	1,400.06	1,509.46	441.53	59.63	410.90
8	757.83	866.57	607.10	899.43	1,654.21	2,902.83	614.47	67.34	715.70
16	1,265.89	1,688.55	584.51	1,668.48	2,145.69	5,394.16	644.30	91.51	1,165.66
32	2,691.25	3,458.24	730.26	3,013.55	2,923.98	9,650.91	937.76	--	--
64	4,920.97	6,208.53	611.61	4,601.35	2,332.71	--	800.51	--	--
128	8,493.40	10,234.19	156.86	4,415.97	--	--	--	--	--
256	--	10,042.6	--	--	--	--	--	--	--

Nodes	Irene-SKL	JUWELS	Marconi-KNL	Mare-Nostrum	Piz Daint	DAVIDE	Frioul	DEEP-ER	Mont-Blanc 3
512	--	5,309.96	--	--	--	--	--	--	--
	MPI =8 omp = 6	MPI =8 omp = 6	MPI=4 omp=68	MPI =8 omp = 6	1 GPU	1 GPU per node	MPI=1 omp=68	MPI=1 omp=64	MPI=64 omp=1

Table 66: Performance of the kernel of part 2, in Gflop/s as reported by the benchmark, using problem size V=96×32×32×32

Node	Irene-SKL	JUWELS	Marconi-KNL	MareNostrum4	Piz Daint
1	141.51	135.23	64.43	144.32	--
2	266.70	263.74	154.70	280.68	--
4	504.36	496.94	423.19	514.96	--
8	936.21	954.09	916.72	930.95	2,694.10
16	1,662.49	1,791.72	1,496.82	1,778.23	5,731.56
32	3,061.35	3,301.38	2,430.12	2,635.74	7,779.29
64	4,680.76	5,979.32	2,457.25	5,264.16	10,607.20
128	5,890.96	10,577.84	2,273.16	7,998.56	13,560.50
256	15,520.32	19,702.56	--	--	--
512	20,095.67	36,079.10	--	--	--
	MPI =8 omp = 6	MPI =8 omp = 6	MPI=4 omp = 68	MPI =8 omp = 6	1 GPU per Node

Table 67: Performance of the kernel of part 2, in Gflop/s as reported by the benchmark, using problem size V=128×64×64×64

Nodes	DAVIDE time (s)	DAVIDE performance (Gflop/s)	DAVIDE Energy (kJ)	Frioul time (s)	Frioul performance (Gflop/s)	Frioul energy (kJ)
1	3.76	1533.13	14.901	81.9	184.729	34.1
2	4.88	3005.07	19.813	56.1	269.705	39.9
4	3.72	5409.18	26.466	34.3	441.534	49.8
8	4.04	7248.57	43.078	24.6	614.466	65.8
16	4.86	3490.27	88.145	23.5	644.303	117.0
32	4.86	4570.13	288.513	16.1	937.755	171.2
64	--	--	--	18.9	800.514	375.0

Table 68: Performance and energy consumption of kernel of part 2 on the PCP Prototypes using problem size V=96×32×32×32

4.11 Quantum Espresso

In this section we describe the benchmark activity carried out for the QE application on various Tier-0 and Tier-1 systems available in PRACE. For each system we report how the application was used and installed, the performance data obtained, and an analysis of the results. On some computer systems we employed a profiling or tracing tool (e.g. Scalasca, Intel APS and Extrae) to give further insights on the benchmark results. In addition, for Piz Daint and the PCP prototypes (i.e. DAVIDE and Frioul), we report the corresponding energy-to-solution data.

4.11.1 Performance on Hazel Hen

Performance benchmarks were not run on this architecture – instead we show data for the Broadwell partition of Marconi (see later). It should also be noted that Hazel Hen will be replaced with a completely different system which is expected to occur in Q1 of 2019.

4.11.2 Performance on Irene

Due to the similarities between this and other architectures tested, benchmarks were not run on the Irene systems. For Intel KNL and Skylake data the reader is referred to the sections on Marconi.

4.11.3 Performance on JUWELS

4.11.3.1 Installation and execution

The QE application was not available on this system so it was compiled from source. On JUWELS we note that it is essential to unset the ARCH environment available otherwise the QE configure script does not recognise the operating system. In addition, we observed runtime problems with the Intel19 MPI library, so this was substituted with Parastation MPI. With this installation procedure, the application compiles, installs and executes correctly. The small dataset was run without using OpenMP threads, while the large data set was run with 13 or 26 MPI tasks per node and four OpenMP threads per task.

4.11.3.2 Results

Strong parallel scaling curves for both datasets are given in Figure 15, with the corresponding data in Table 69.

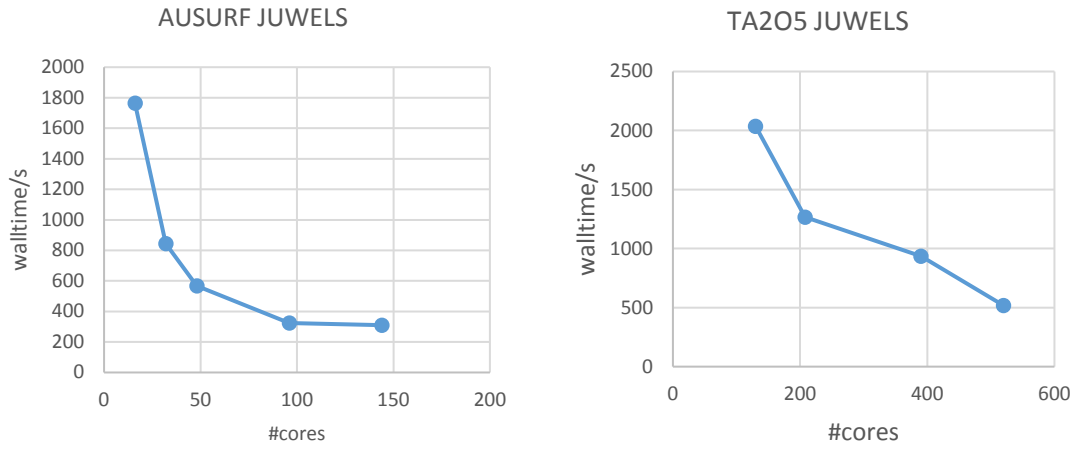


Figure 15: Strong scaling for small (left) and large (right) datasets on JUWELS

Small test case		Large test case	
Cores	**Wall time/s**	**Cores**	**Wall time/s**
16	1765	130	2037
32	843	208	1266
48	567	390	936
96	324	520	518
144	310		

Table 69: Performance Data for JUWELS

4.11.3.3 Analysis

The scaling and performances are as we expected for this architecture. On JUWELS we did a further performance analysis using the Scalasca tool for the AUSURF benchmarks (single node). A snapshot from this analysis is shown in Figure 16 and from the leftmost panel we see that about 20% of the elapsed time is consumed in MPI calls (this result is confirmed by other analyses on different systems, see later). The middle panel, on the other hand, shows that most of the CPU time

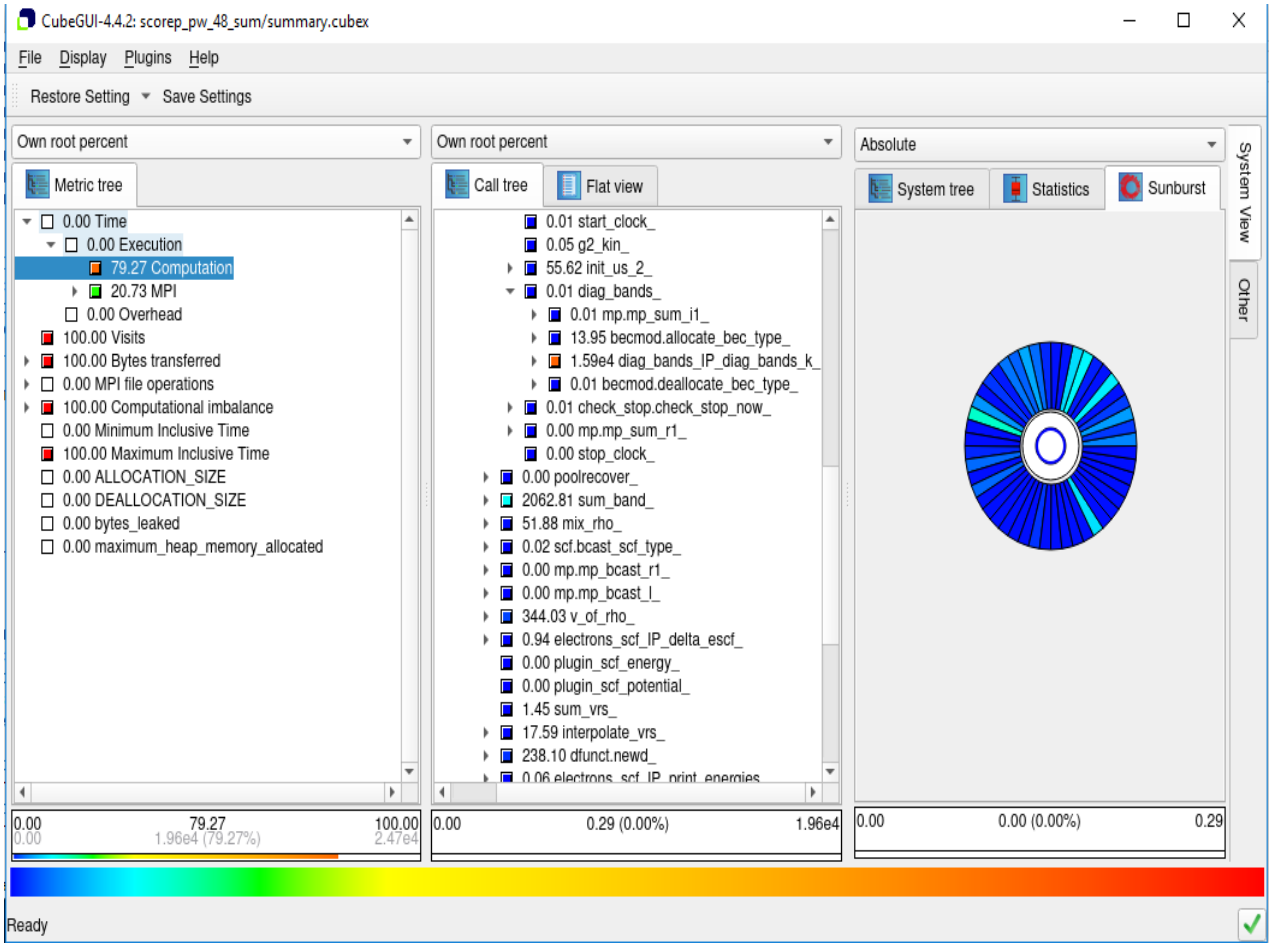


Figure 16: Scalasca Analysis of AUSURF for 1 node on JUWELS

is used in the diagonalisation routines for the calculation of the electronic SCF energy (mainly in the MKL library). The third panel shows the almost perfect load balancing of the 48 tasks as demonstrated by the fact that the slices, each representing an MPI task, have similar colours.

4.11.4 Performance on Marconi (KNL and Skylake and Broadwell)

4.11.4.1 Installation and execution

Optimised installations of QE for both KNL and Skylake and Broadwell partitions are present on Marconi and were used in the benchmarks. For the KNL partition 64 MPI tasks/node were used for the small test case while 13 per node were used for the large test case. For Marconi Skylake 48 tasks were used for the former and 26 for the latter, for Broadwell 36 and 13 tasks respectively

were employed. We note that while the Marconi-KNL nodes have 68 cores, using 64 tasks is more convenient for the parallelisation strategy of QE.

4.11.4.2 Results

The strong scaling curves for small and large test cases on all Marconi partitions are shown in Figure 17, Figure 18 and Figure 19 with the corresponding data in Table 70.

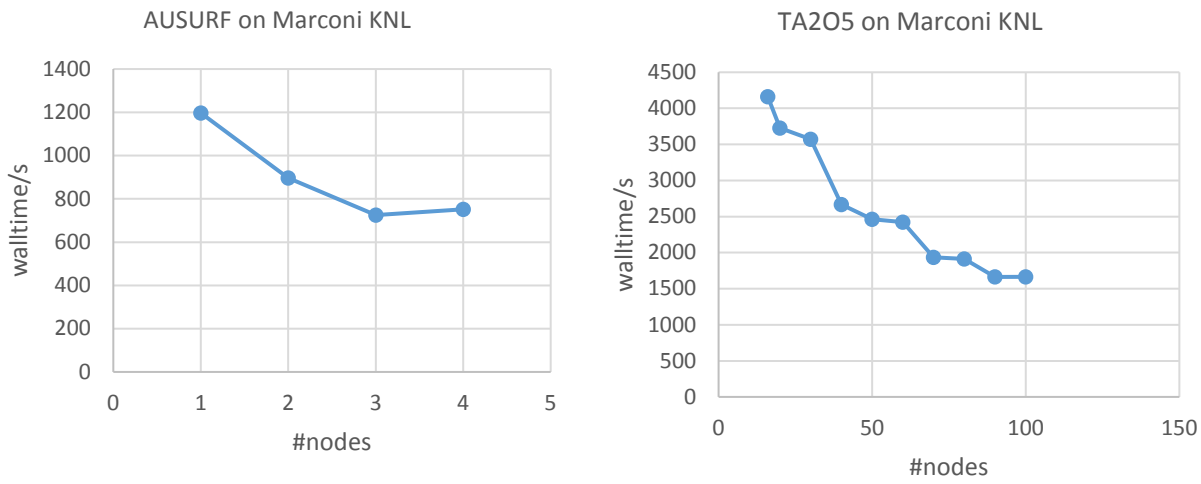


Figure 17: Strong scaling curves for small (left) and large (right) test cases on Marconi-KNL

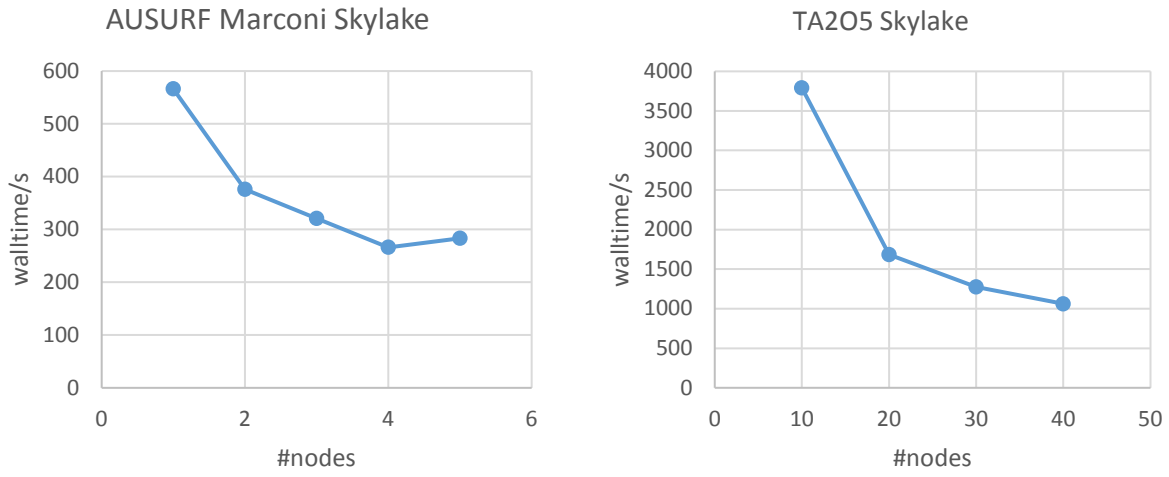


Figure 18: Strong scaling of small (left) and large (right) test cases on Marconi Skylake

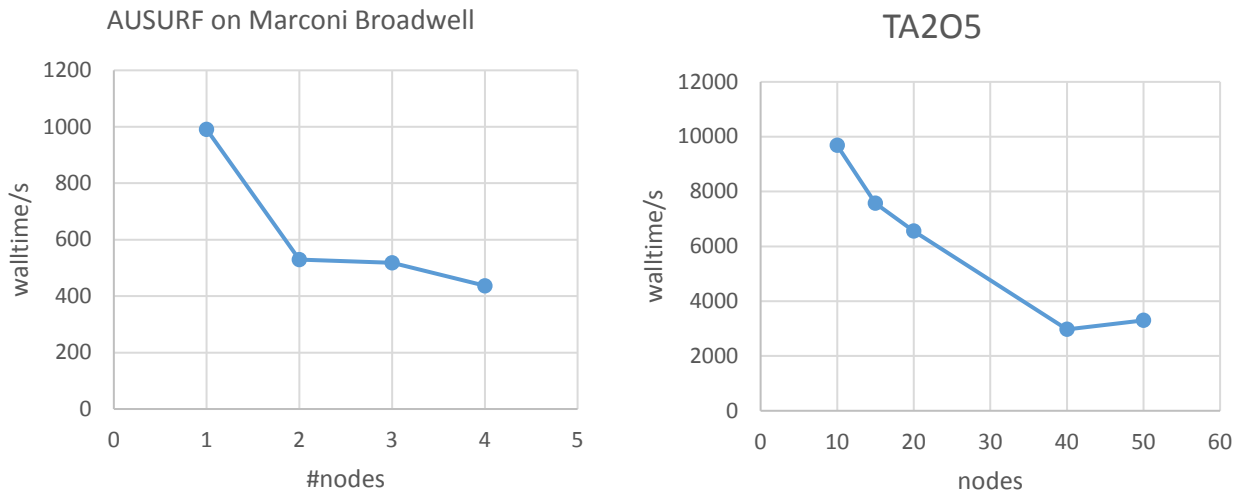


Figure 19: Performance on Marconi Broadwell

Broadwell				KNL				Skylake			
Small		Large		Small		Large		Small		Large	
Nodes	Wall time/s	Nodes	Wall time/s	Nodes	Wall time/s	Nodes	Wall time/s	Nodes	Wall time/s	nodes	Wall time/s
1	991	10	9681	1	1197	16	4164	1	566	10	3791
2	529	15	7567	2	897	20	3726	2	376	20	1685
3	518	20	6550	3	725	30	3573	3	321	30	1274
4	437	40	2969	4	751	40	2667	4	266	40	1062
		50	3295			50	2463	5	283		
						60	2421				
						70	1936				
						80	1911				
						90	1663				
						100	1664				

Table 70: Performance data for the Broadwell, KNL and Skylake partitions on Marconi

4.11.4.3 Analysis

We see from the graphs that for both datasets, the performance of Marconi Skylake is about twice that of KNL and Broadwell. In order to understand better the performance and scaling on the two architectures, we ran the Intel Application Snapshot tool for the small test case on KNL and Skylake

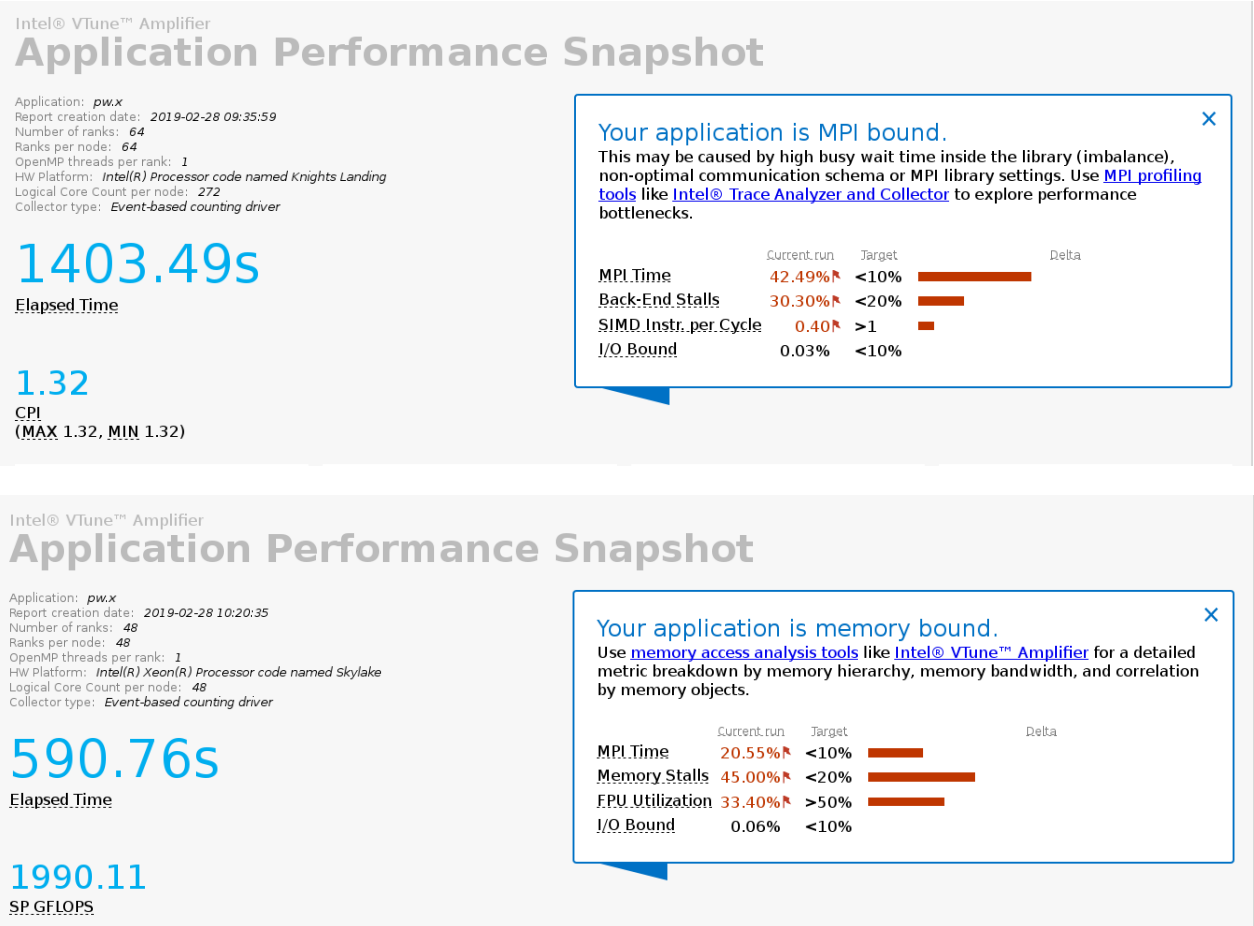


Figure 20: Performance Analyses using the Intel APS tool for the small dataset on Marconi-KNL (top) and Marconi Skylake (bottom)

- snapshots of the graphical outputs are show in Figure 20 We notice that on Skylake, just like on JUWELS, about 20% of the elapsed time is spent in MPI calls but the application is classed as memory bound by the profiler, i.e. dictated by the time needed to access main memory. For KNL, on the other hand, the application is MPI bound with more than 40% of the elapsed time being consumed in the MPI calls. The poor MPI performance on KNL is in fact one reason for the poor performance of MPI-only programs on this processor.

4.11.5 Performance on MareNostrum4

4.11.5.1 Installation and execution

QE version 6.2 was present on MareNostrum4 but given the standard x86 architecture of the system we opted to install the latest version 6.3 from source. No problems were revealed during installation

and the application was benchmarked similarly to other systems (e.g. JUWELS or Marconi SKL), although due to time constraints only the small dataset was benchmarked.

4.11.5.2 Results

The variation of elapsed time with MPI tasks for the small dataset are reported in Figure 21 and Table 71.

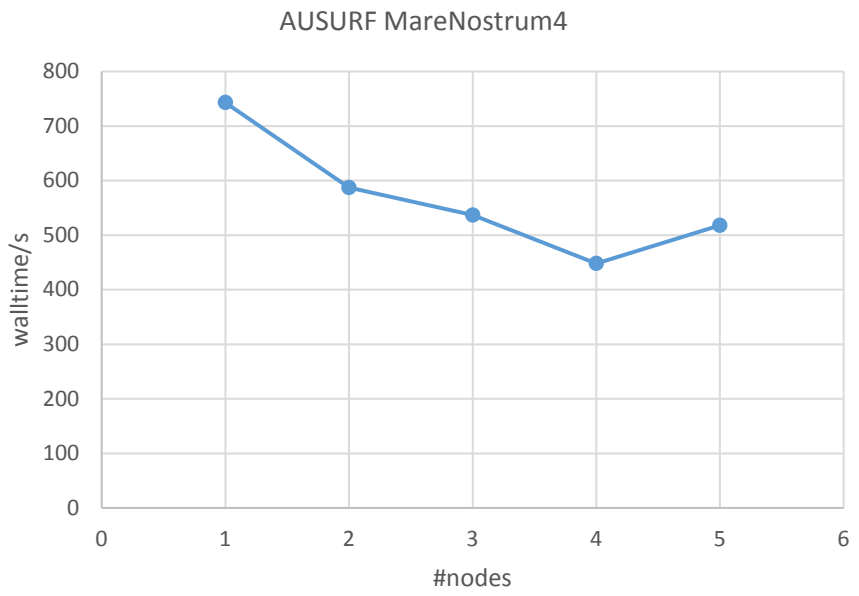


Figure 21: Strong scaling of the small dataset on MareNostrum4

Nodes	Wall time/s
1	743
2	587
3	537
4	448
5	518

Table 71: Performance data on MareNostrum4

4.11.5.3 Analysis

Only limited benchmarks were collected but the wall times show the expected trend for the AUSURF small benchmark case, even though the absolute values are higher than equivalent results for Marconi Skylake or JUWELS. To understand this difference, we have compared the timings output by QE for the two architectures for one node (=48 cores) and found that the major difference lies in the calculation of the electron energy (see Figure 22). Here the most expensive function is the c-bands routine which requires matrix operations and are performed in the Intel MKL library. One reason for the increased wall time could then be the different versions of the MKL used for compilations; MKL 2017 in the case of MareNostrum4 as opposed to MKL 2018 for the Marconi

version. Unfortunately, we haven't been able to confirm this hypothesis by recompiling the application on MareNostrum4 with later versions of the Intel suite.

Extrae Tracing

The availability of the Extrae tracing tool on MareNostrum4 has allowed us to perform a very detailed performance analysis on the system. Because trace files can be very large, we have conducted an analysis of only a very short run, based on one iteration of the AUSURF input and using 8 MPI tasks. The 1.6 GB trace file was then further reduced with the Paraver tool to about 500 Mb to allow further analysis with Paraver itself. A snapshot of the trace together with a window detailing only the MPI calls is shown in Figure 23. Here, the upper window shows all the activity during the iteration, while the lower shows only the MPI-related processes. In this tool colours are used to indicate important activity states so in the upper window we see processes which are "Running" (blue), in MPI barriers (red) or communications (orange), while the yellow lines link tasks involved in communications. In the lower pane, tasks are identified as involved in calls to MPI_Barrier (red), MPI_AlltoAll (violet), MPI_Rsend (green), MPI_Allreduce (pink) and MPI_comm_size (blue), while the rectangles have been added to indicate the MPI_Barrier synchronisation for the k-point calculation. We see very clearly from the figure that an important bottleneck is the MPI_Barrier needed to synchronise the two groups of MPI tasks corresponding to the two k-points. Other communications, such as MPI_AlltoAll or MPI_Allreduce, are also important although we stress that we have focussed on a communication intensive region of the program and with just eight MPI tasks we have only a limited view of the more general parallel communications pattern.

```
MareNostrum4
Called by electrons:
   c_bands    :   644.60s CPU    655.89s WALL (     22 calls)
   sum_band   :    52.31s CPU     53.94s WALL (     22 calls)
   v_of_rho   :     4.74s CPU      4.95s WALL (     22 calls)
   newd       :     4.40s CPU      5.52s WALL (     22 calls)
   mix_rho    :     1.07s CPU      1.12s WALL (     22 calls)
```

```
Marconi
Called by electrons:
   c_bands    :   455.48s CPU    472.78s WALL (     22 calls)
   sum_band   :    52.06s CPU     54.25s WALL (     22 calls)
   v_of_rho   :     4.38s CPU      4.62s WALL (     22 calls)
   newd       :     4.36s CPU      5.55s WALL (     22 calls)
   mix_rho    :     1.14s CPU      1.22s WALL (     22 calls)
```

Figure 22: Output timings from the small test case on MareNostrum4 (upper) and Marconi Skylake (lower)

Figure 23: Snapshots of an Extrae trace file of the first iteration of Quantum Espresso on MareNostrum4

4.11.6 Performance on Piz Daint

4.11.6.1 Installation and execution

Since Piz Daint is equipped with one NVIDIA Tesla Pascal P100 per node we decided to run the CUDA Fortran version of Quantum Espresso, already present on the system. For the runs we used one MPI task per node (corresponding to the number of GPUs) and 12 OpenMP threads per task. We note that because nearly all the program memory is assigned to the GPU memory (16 GB per device), it is difficult to obtain a large number of benchmark data points with this version of the code. The reason is due to the fact that you need a large number of GPUs in order to have sufficient memory to actually run the program, but then you are already likely to have reached the performance limit.

4.11.6.2 Results

In Figure 24 and Table 72 we show the scaling data we have obtained for both the small and large test case.

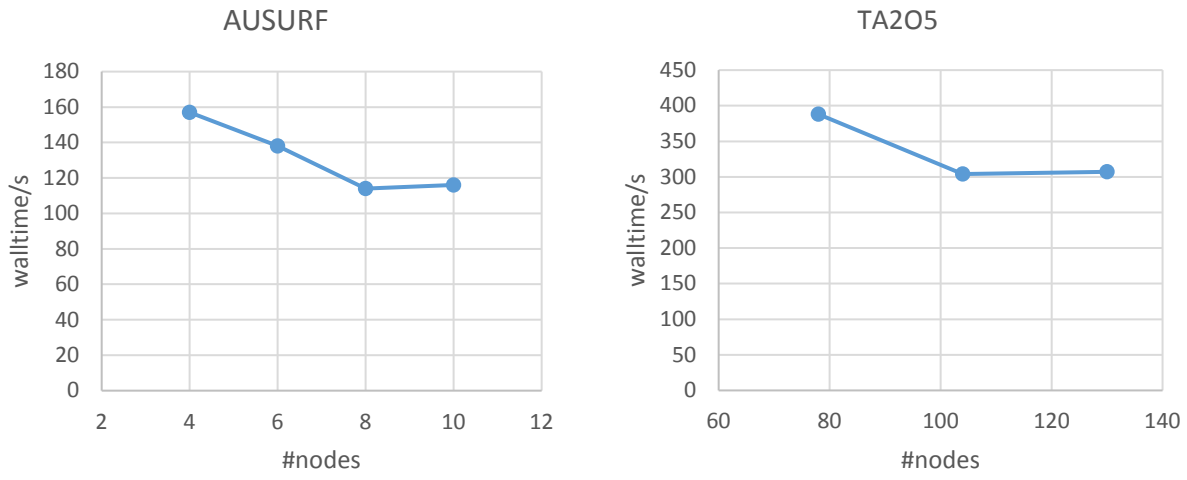


Figure 24: Benchmarks for the small and large test cases on Piz Daint

Small Test Case		Large Test Case	
Nodes	**Wall time/s**	**Nodes**	**Wall time/s**
4	157	78	388
6	138	104	304
8	114	130	307
10	116		

Table 72: Performance data for Piz Daint

4.11.6.3 Analysis

For the reasons cited above only a limited number of data points was collected. We note however that the program performance is very high on this architecture. The SLURM scheduler also provides energy data per node, but this analysis we leave until the end of the section on Quantum Espresso.

4.11.7 Performance on SuperMUC-NG

Not available during the benchmark period.

4.11.8 Performance on DAVIDE and Frioul

4.11.8.1 Installation and Execution

For the DAVIDE POWER8+GPU cluster we used the CUDA Fortran version of QE compiled with the OpenMPI installation for the IBM POWER8 architecture. When using the GPU all four accelerators per node were allocated, employing four MPI tasks and one OpenMP thread per task. We note that for the large test case, which has 26 k-points, these parameters are not optimal because we should use multiples of 13 or 26 for the MPI tasks. Since for the GPU version we can allocate only one MPI task/GPU, running over 26 k-points would have made it impossible to get meaningful benchmarks (we recall that the DAVIDE cluster has only 45 nodes). For the Frioul system the

application was compiled and run as for Marconi-KNL (see above), although it should be noted that only KNL devices in flat mode were available. For both systems the energy consumed for each benchmark, as reported by the appropriate tool, is also reported.

4.11.8.2 Results

The wall times for the two test cases are given in Figure 25 and Table 73 and show results for accelerated and non-accelerated (i.e. only POWER8) runs on DAVIDE together with the simulations launched on Frioul. In Figure 26 and Table 74 we report the corresponding energies.

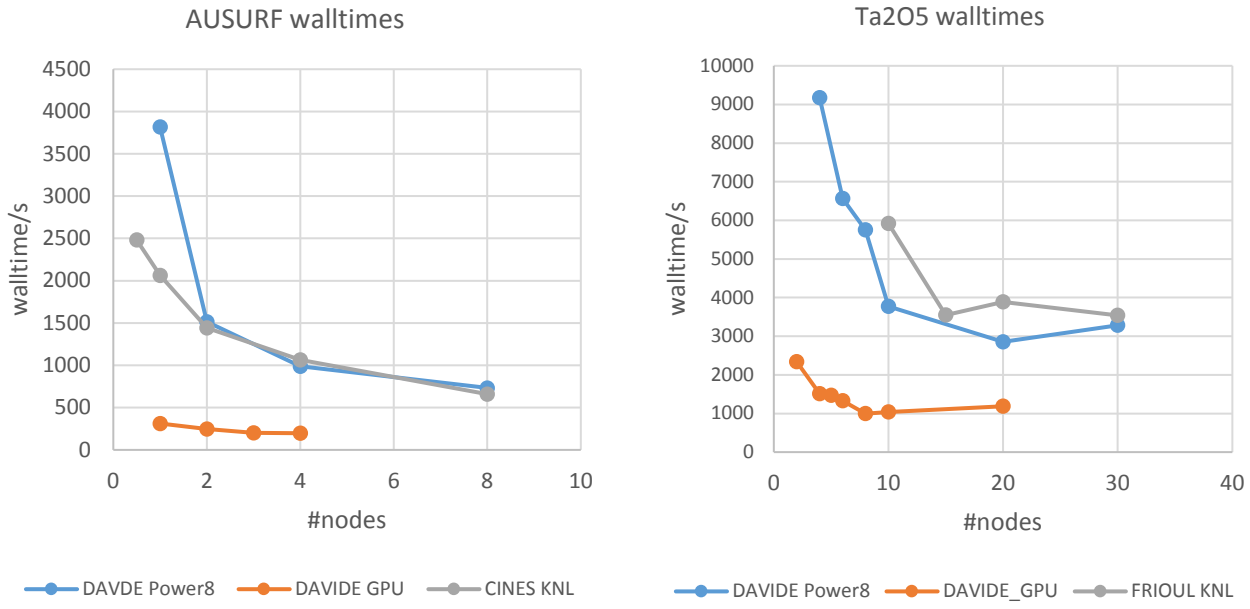


Figure 25: Performances for the small (right) and large (left) datasets on the PCP prototypes, DAVIDE and Frioul. For DAVIDE both accelerated and non-accelerated results (POWER8) are shown

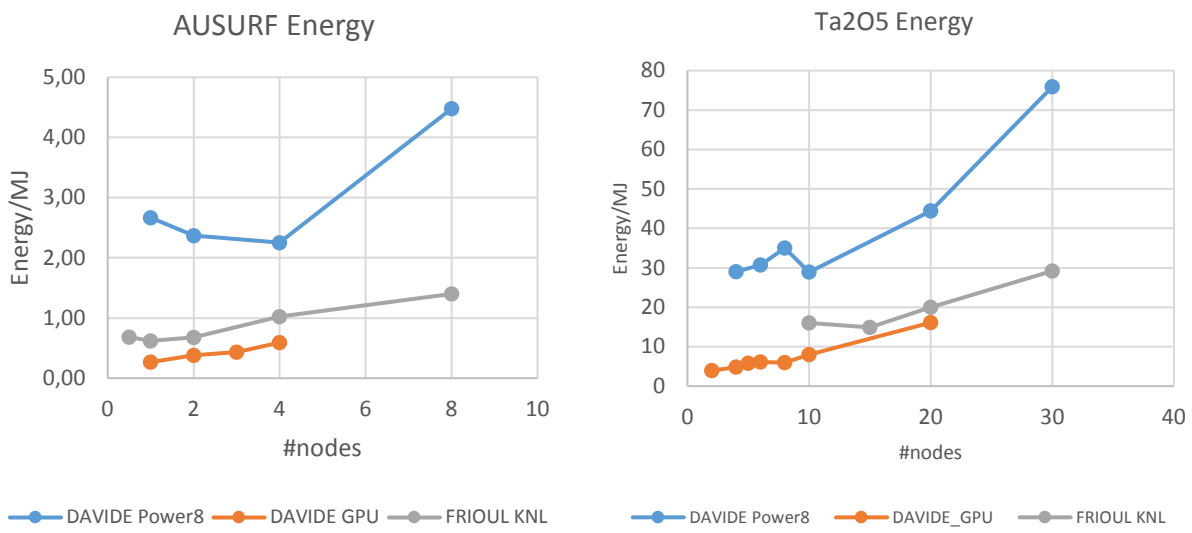


Figure 26: The energy consumed by the batch jobs for the two datasets on the PCP prototypes

DAVIDE POWER8				DAVIDE POWER8 + GPU				Frioul			
Small		Large		Small		Large		Small		Large	
Nodes	Wall time/s	Nodes	Wall time/s	Nodes	Wall time/s	Nodes	Wall time/s	Nodes	Wall time/s	nodes	Wall time/s
1	3817	4	9169	1	312	2	2337	1 (32 cores)	2481	10	5916
2	1514	6	6560	2	248	4	1511	1	2062	15	3549
4	990	8	5753	3	200	5	1470	2	1442	20	3886
8	733	10	3771	4	197	6	1324	4	1063	30	3539
		20	2855			8	995	8	659	40	4732
		30	3285			10	1041				
						20	1189				

Table 73: Performance data on DAVIDE and Frioul

DAVIDE POWER8				DAVIDE POWER8 + GPU				Frioul			
Small		Large		Small		Large		Small		Large	
Nodes	Energy /MJ	Nodes	Energy /MJ	Nodes	Energy/MJ	Nodes	Energy /MJ	Nodes	Energy /MJ	nodes	Energy /MJ
1	2.7	4	29.0	1	0.3	2	3.9	0.5	0.7	10	16.0
2	2.4	6	30.7	2	0.4	4	4.8	1	0.6	15	14.9
4	2.3	8	35.0	3	0.4	5	5.8	2	0.7	20	20.0
8	4.5	10	28.9	4	0.6	6	6.1	4	1.0	30	29.2
		20	44.4			8	6.0	8	1.4		
		30	75.9			10	8.0				
						20	16.1				

Table 74: Energy data for DAVIDE and Frioul

4.11.8.3 Analysis

We see from Figure 25 that for both datasets using the Tesla P100 GPUs on the DAVIDE system gives the best performances while the wall times on Frioul's KNL nodes are similar to the

performances on the POWER8 processors. Comparing with the other GPU system tested, Piz Daint, we note that the GPU performances for the small dataset on DAVIDE are similar. For the larger dataset the application is slower than Piz Daint, but this may be partly due to the fact that OpenMP threads were not used on DAVIDE. However, an advantage of the latter system is that with four GPUs per node, we require only one node to perform the small dataset whereas with Piz Daint we need at least four. The performances for the Frioul system are lower and, in addition, lower than those reported for Marconi-KNL (see Figure 17). One reason may be the fact that on Marconi-KNL devices were used in cache mode, whilst on Frioul flat mode was employed. Since the QE application has not been programmed to exploit the MCDRAM on KNL a difference in performance would be expected since the application can use the additional memory available in cache mode. As regards the energies to solution shown in Figure 26, we see that for both datasets the simulations with the GPUs require the least energy, while the POWER8 runs require the most energy. The KNL simulations use considerably less energy than POWER8 alone, but still higher than the GPUs.

4.11.9 Performance on DEEP-ER SDV

4.11.9.1 Installation and Execution

The program was installed with the Intel compiler and options suitable for KNL execution, but using the Parastation implementation of MPI instead of Intel MPI. We remark that benchmarking this platform was challenging due to the small number of KNL nodes available and, as the results appear to show, an issue relating to communications between three nodes.

4.11.9.2 Results

Performances for the small dataset are shown in Table 75.

Nodes	#tasks	Wall time (s)
1	64	2669
2	128	1806
3	192	6420

Table 75: Benchmarks for the DEEP-ER SDV

4.11.9.3 Analysis

For this system we have been able to obtain only very few data. The performances for 1–2 nodes are lower than other similar data for KNL (e.g. Marconi-KNL), while it has been very difficult to perform runs for more than three nodes, presumably due to a communication issue at the time the benchmarks were performed.

4.11.10 Summary of performance and energy analyses

For the small test case, the benchmark data on all architectures effectively reflect the relative clock speeds of the processor cores since the computationally expensive calculations all take place in the linear algebra library (e.g. Intel's MKL). This is true for the large test case as well but the data are more difficult to interpret because the high memory requirements mean that large numbers of nodes

are often needed and the UEABS architectures have different memory per core specifications. In addition, the fact that this test case has 26 k-points means that to exploit the best parallelism we need groups of cores which are multiples of 13 or 26, i.e. cores will be unused if we can only ask for whole nodes (by contrast we recall that the small test case has 2 k-points which is more convenient). In addition, depending on the cores/node available sometimes we can benefit from OpenMP threads. For both inputs, GPU acceleration has a big impact but since the CUDA version of the program does not use the host memory, many GPUs are required in order to provide sufficient memory.

The more detailed performance analyses using profilers has confirmed that while single core performance cannot probably be improved any further, there is a scaling bottleneck due to the MPI barriers needed to synchronise the tasks involved for each k-point calculation.

4.11.10.1 Energy Consumption

In addition to the performance analysis in terms of elapsed times, we have also recorded the energy used per job when available. This has already been described for the PCP prototypes but we can also extend the analysis to a Tier-0 system, i.e. Piz Daint. A comparison of the energies obtained from the PCP prototypes and Piz Daint using the GPU for the test cases is shown in Figure 27 and Table 76 (we have excluded the energies from the non-accelerated runs of DAVIDE). For the small test case Piz Daint seems to be more energy efficient than DAVIDE, despite the fact that for the Tier-0 system more nodes are required, although we recall that DAVIDE has four NVIDIA Tesla P100 GPUs/node. We have limited data for the large test case on Piz Daint, but the energy required is similar to that of DAVIDE within the strong scaling regime (i.e. between 1–10 nodes for DAVIDE).

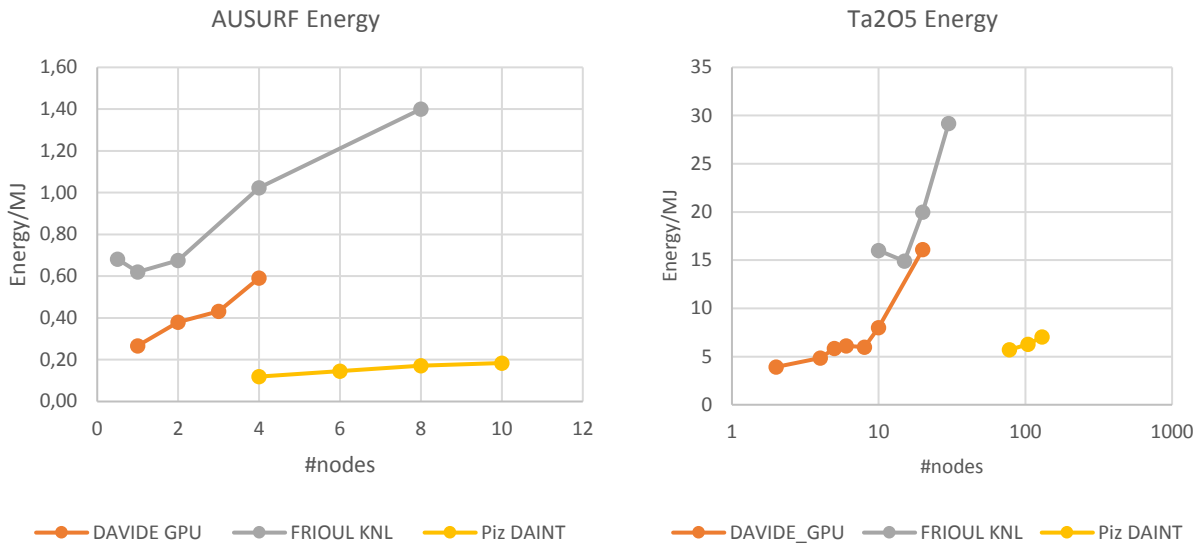


Figure 27: Comparison of the energy consumed per job for the two test cases on DAVIDE (GPU), Frioul (KNL) and Piz Daint (GPU)

Piz Daint				DAVIDE POWER8 + GPU				Frioul			
Small		Large		Small		Large		Small		Large	
Nodes	Energy /MJ	Nodes	Energy /MJ	Nodes	Energy /MJ	Nodes	Energy /MJ	Nodes	Energy /MJ	nodes	Energy /MJ
4	0.12	78	5.70	1	0.27	2	3.92	0.5	0.68	10	16.00
6	0.15	104	6.27	2	0.38	4	4.84	1	0.62	15	14.90
8	0.17	130	7.05	3	0.43	5	5.84	2	0.68	20	20.00
10	0.18			4	0.59	6	6.13	4	1.02	30	29.20
						8	5.98	8	1.40		
						10	8.01				
						20	16.11				

Table 76: Comparison of energy-to-solution for Piz Daint, DAVIDE and Frioul

4.12 SHOC

In order to run all benchmarks from SHOC at once, we use the size 3, as mentioned in Section 2.12. The command to run all benchmarks is:

./shocdriver –s 3 –cuda

The SHOC benchmark has been run on Cartesius, Ouessant, DAVIDE, Piz Daint, and JUWELS. The results are presented in Table 77.

4.12.1 Performance Results

Device/Bench	K40 (Cartesius)	POWER8 + P100 (Ouessant)	POWER8 + P100 (DAVIDE)	P100 (Piz Daint)	V100 (JUWELS)
BusSpeedDownload (GB/s)	10.5	32.23	32.9	12.47	12.38
BusSpeedReadback (GB/s)	10.5	34	34	13.21	13.15
maxspflops (Gflop/s)	3716	10424	10475	9322	15539
maxdpflops (Gflop/s)	1412	5315	5318	4735	7802
gmem_readbw (GB/s)	177	575.16	574.53	574	894.93
gmem_readbw_strided (GB/s)	18	99.15	98.65	97.84	476.83
gmem_writebw (GB/s)	175	436	436	431	777.69
gmem_writebw_strided (GB/s)	7	26.3	26.15	25.1	60.18
lmem_readbw (GB/s)	1168	4239	4245	4166	9413
lmem_writebw (GB/s)	1194	5488	5485	5221	10125
BFS (MEdges/s)	49.2	91.9	90.2	110	122.7
FFT_sp (Gflop/s)	523	1472	1467	1498	2255
FFT_dp (Gflop/s)	262	733	734	742	1131
SGEMM (Gflop/s)	2921	8604	8732	8255	14065
DGEMM (Gflop/s)	1032	3635	3654	3546	5399
MD (SP) (Gflop/s)	188	483	522	456	863
MD5Hash (GH/s)	3.38	15.77	15.87	14.03	34.2
Reduction (GB/s)	144	271	270	250	331
Scan (GB/s)	53.8	99.2	98.5	106	184.8
Sort (GB/s)	3.22	12.54	12.52	9.87	20.98
Spmv (FP64/no padding) (Gflop/s)	4	23	23	19	56
Spmv (FP32/with padding) (Gflop/s)	23	65	65	60	151
Stencil2D (Gflop/s)	126	465	470	431	661
Stencil2D_dp (Gflop/s)	60	258	258	250	322

Device/Bench	K40 (Cartesius)	POWER8 + P100 (Ouessant)	POWER8 + P100 (DAVIDE)	P100 (Piz Daint)	V100 (JUWELS)
Triad (GB/s)	13.5	43	41.3	15.6	16.31
S3D (level 2) (Gflop/s)	95	294	292	289	423

Table 77: SHOC performance

We can see clear differences between the GPU families (Kepler, Pascal, Volta), but also between the hosts that drive these GPUs. The difference amongst these results sits in the fact that the Ouessant and DAVIDE feature NVLink connections between the hosts and accelerators, as opposed to PCIe in all other cases. The performance of this can be easily noticed in the BusSpeedDownload/BusSpeedUpload/Triad benchmarks, outperforming both the Piz Daint results based on P100 as well as JUWELS with the newer Volta architecture.

For the other workloads, we can see the clear evolution in terms of compute and memory performance between the GPU families. The peak performance difference in terms of FP32/FP64 compute between Kepler and Pascal architectures is of a factor of around 2.5. The same goes for the peak memory bandwidth between the two cards (288 GB/s vs 732 GB/s). Thus, this is the main reason why the P100 results are generally around a factor of three times faster than the K40 ones.

It is interesting to compare the DAVIDE/Ouessant results to the ones obtained on Piz Daint. The results on Piz Daint are on average inferior, and this is mostly because there is still communication happening between the CPU hosts and the GPU, NVLink outperforming PCIe transfers.

When comparing the P100 accelerator with the V100 one from JUWELS, we see that again NVIDIA has improved the architecture, this time with around 50% (both in terms of memory bandwidth and FP32/FP64 peak compute). The 50% performance difference is reflected in most benchmark results, with the exception of those that benefit from NVLink connectivity on the host side (POWER8). Another interesting exception is the gmem_readbw_strided benchmark, that now reaches 50% of the peak memory bandwidth (in the case of V100), whereas for the Kepler GPU generation it was reaching around 10%, and for the Pascal GPU around 20%. Thus, V100 has much better architectural support for strided access.

4.12.2 Energy Consumption

Being a synthetic benchmark, SHOC does not really fit the time and energy to solution paradigm as the other scientific benchmarks. However, it has been included for completeness (although “solution” does not represent much in this case) on some representative benchmarks. The results have been obtained on DAVIDE. As an interesting note, all compute-bound workloads draw around 1200W of power on average, whereas the memory-bound ones only around 750W. The test setup to measure energy consumption was as follows:

- GEMM with size 4 benchmark and executed 1000 times
- FFT with size 4 benchmark and executed 10000 times
- MD5Hash with size 4 and executed 1000 times

Test	Number of GPUs	Time to solution (s)	Energy to solution (kJ)	Average node power consumption (W)
GEMM	1	193	140	722
GEMM	4	226	289	1274
FFT	1	54	34.7	642
FFT	4	166	126	758
MD5Hash	1	104	70.7	680
MD5Hash	4	106	125	1176

Table 78: SHOC time and energy to solution

The tests that are run on 4 GPUs are actually performing four times the work of the ones that are using one GPU. Thus, when the relative time difference between the one GPU and four GPU runs is small, almost perfect scaling efficiency is observed, as it can be observed for the compute-bound examples such as GEMM. On the other side, if we take the FFT example, it needs 3.07× more time to perform four times the work, while using 1.18× more power on average. The energy to solution metric merges these two metrics together, showing that this implementation of FFT used 3.63× more energy to perform 4× the work.

We have also used the same "size 3" benchmarks as outlined below, and we report the energy metrics for a full benchmark suite run as presented by the DAVIDE system. The results are below; however, they are less insightful compared to the per-benchmark energy measurements.

```
./shocdriver –s 3 –cuda
```

Cumulative (all nodes)

```
Mean power (W): 577.757197279
Total energy (J): 169282.858803
Mean GPUs power (W): 148.426778011
Total GPUs energy (J): 43785.8995132
```

4.13 SPECFEM3D

The code has been tested and timed on several architectures, it has been compiled on almost all systems with Intel compilers (between version 17.0 and version 19.1 depending on their availability on the systems) except on Cray systems (Hazel Hen, Piz Daint: PrgEnv-cray+cray-mpich), OpenPOWER (DAVIDE: gnu+openmpi) and ARM (Dibona: arm-hpc-compiler). So far, it has only been possible to run on one fixed core count for each test case, so scaling curves are not available.

4.13.1 Performance Results by Test Case

4.13.1.1 Test case A

Machine		Solver (s)	Threads OpenMP
Tier-1 Systems	Occigen	1082	12
	SuperMUC-Haswell	1210	12
	SuperMUC-Sandy Bridge	3367.2	12
PRACE Tier-0 Systems	Hazel Hen	2389	6

Machine		Solver (s)	Threads OpenMP
	Joliot Curie-KNL	1639	16
	Joliot Curie-SKL	734	12
	JUWELS	658	12
	Marconi-KNL	1653	17
	MareNostrum4	744	12
	Piz Daint	195	6
PRACE PCP Prototypes	Frioul	1963.5	17
Mont-Blanc 3 Prototype	Dibona	3921.2	16

Table 79: Time to solutions for SPECFEM3D Globe on test case A

4.13.1.2 Test case B

Machine		Solver (s)	Threads OpenMP
Tier-1 Systems	Occigen	248.2	12
	SuperMUC-Haswell	240.36	12
PRACE Tier-0 Systems	Joliot Curie-KNL	330	16
	Joliot Curie-SKL	169	12
	JUWELS	193	6
	Marconi-KNL	1211	17
	MareNostrum4	156	12
	Piz Daint	50	6

Table 80: Time to solutions for SPECFEM3D Globe on test case B

4.13.1.3 Test case C

4.13.1.3.1 Run on one node

Machine		Solver (s)	Threads OpenMP
Tier-1 Systems	Occigen	143.5	4
	SuperMUC-Haswell	196.2	4
PRACE Tier-0 Systems	Joliot Curie-KNL	160.3	10
	Joliot Curie-SKL	83.9	8
	JUWELS	83.2	8
	Marconi-KNL	3440.3	12
	MareNostrum4	88.2	8
	Piz Daint	26.7	2
PRACE PCP Prototypes	DAVIDE	113	2
	Frioul	944.4	12
DEEP-ER Prototype	SDV	134.6	4
Mont-Blanc 3 Prototype	Dibona	474.9	10

Table 81: Time to solutions for SPECFEM3D Globe on test case C using one node

4.13.1.3.2 Run on two nodes

Machine		Solver (s)	Threads OpenMP
Tier-1 Systems	Occigen	109	4
	SuperMUC-Haswell	123.678	8
PRACE Tier-0 Systems	Joliot Curie-KNL	468.4	22
	Joliot Curie-SKL	68.2	16
	JUWELS	66.7	16
	Marconi-KNL	1246.2	22
	MareNostrum4	74.25	16
	Piz Daint	16.6	4

Machine		Solver (s)	Threads OpenMP
PRACE PCP Prototypes	DAVIDE	51.7	4
	Frioul	170.7	22
DEEP-ER Prototype	SDV	99	8
Mont-Blanc 3 Prototype	Dibona	656	20

Table 82: Time to solutions for SPECFEM3D Globe on test case C using two nodes

4.13.2 Comparison methodology

In order to compare the efficiency of the SPECFEM3D code on the different supercomputers, we chose to test the efficiency of the code for a fixed number of compute nodes (and therefore a variable number of cores) as it allows us to easily observe the efficiency of the code according to the architecture used. Indeed, current heterogeneous and/or hybrid (CPU+GPU) architectures make it difficult for us to evaluate the performance for a fixed number of cores used since the number of cores per node is very variable depending on the architectures used; 68 cores for Intel KNL architectures compared to eight for Intel Sandy Bridge nodes for example.

In order to highlight the theoretical and experimental performance accelerations according to the architecture used, we compared the theoretical peak performance per node of each machine with that of the MareNostrum4 supercomputer (ranked 25 in the TOP500 November 2018) then we did the same for the solver times for SPECFEM3D compiled on the different machines compared to the solver time we obtained on the MareNostrum4 supercomputer. Here is the definition of the two metrics used in the graph below:

$$Theoretical\ performance\ speedup\ compared\ to\ MareNostrum4 = \frac{Peak\ performance\ per\ node\ supercomputer\ X}{Peak\ performance\ per\ node\ MareNostrum4}$$

$$Solver's\ resolution\ time\ speedup\ compared\ to\ MareNostrum4 = \frac{Solver\ time\ Supercomputer\ X}{Solver\ time\ MareNostrum4}$$

4.13.3 Comparative results of systems

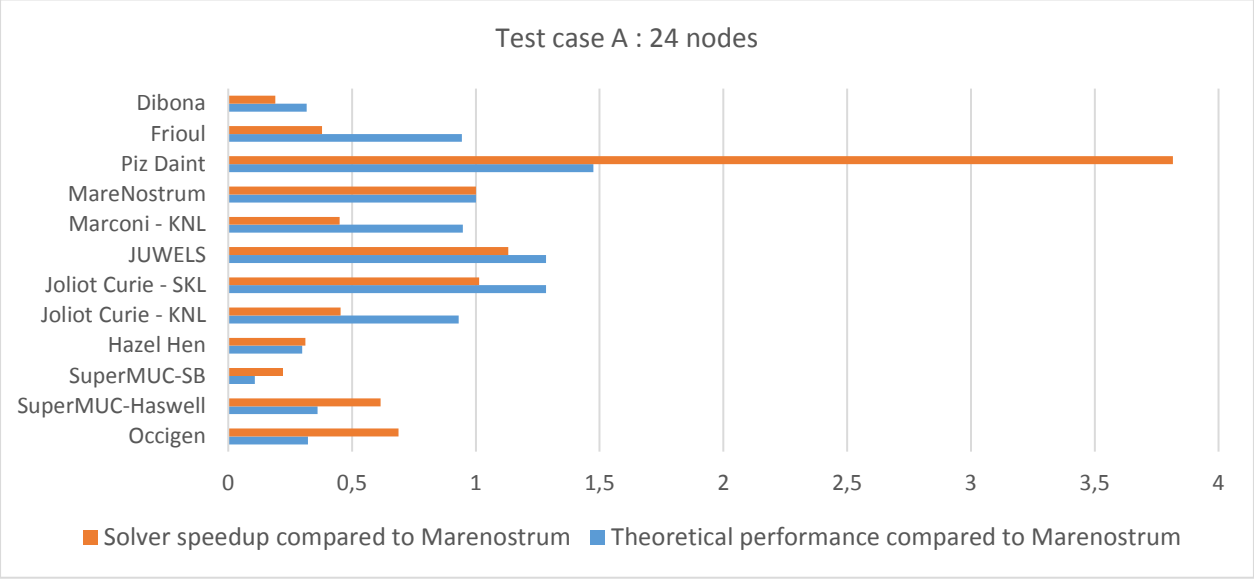


Figure 28: Solver speedup and theoretical performance compared to MareNostrum4 on 24 nodes

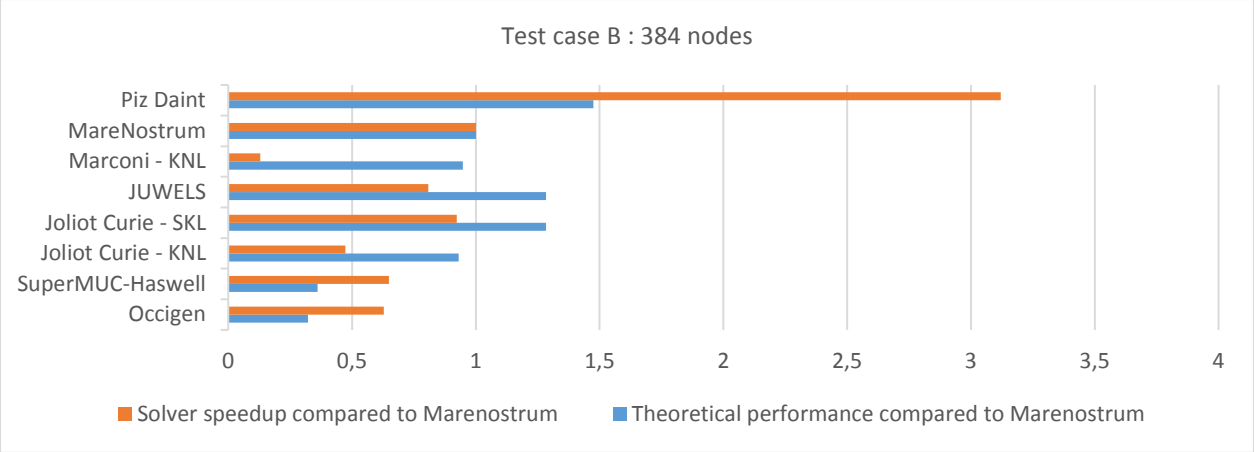


Figure 29: Solver speedup and theoretical performance compared to MareNostrum4 on 384 nodes

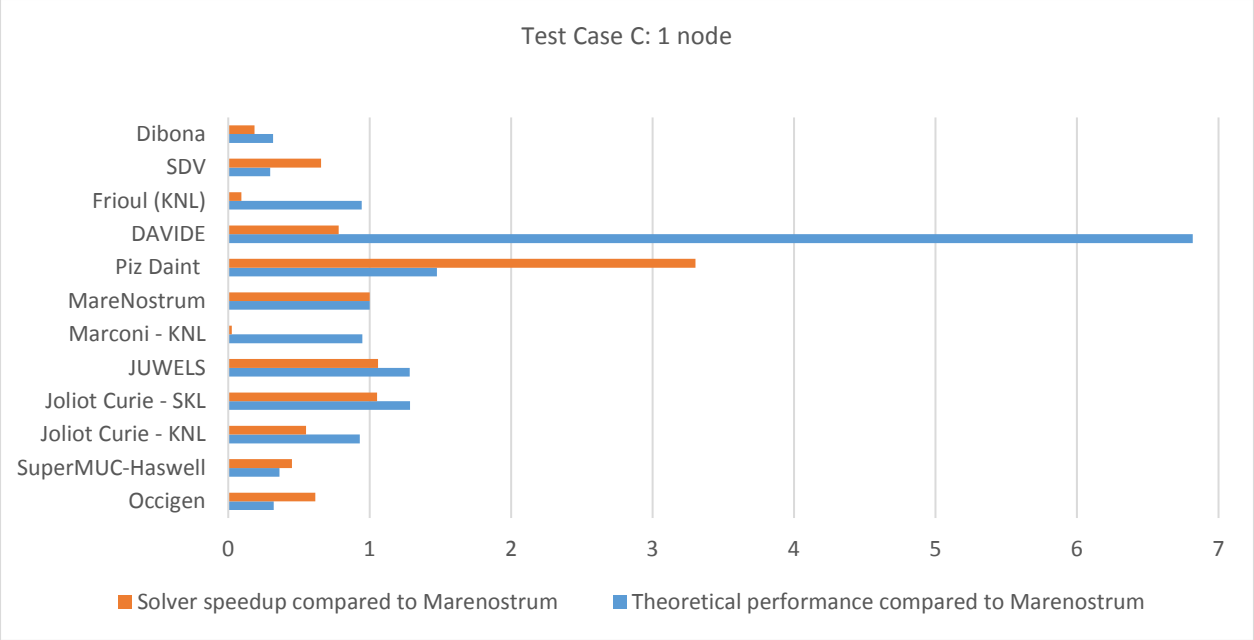


Figure 30: Solver speedup and theoretical performance compared to MareNostrum4 on 1 node

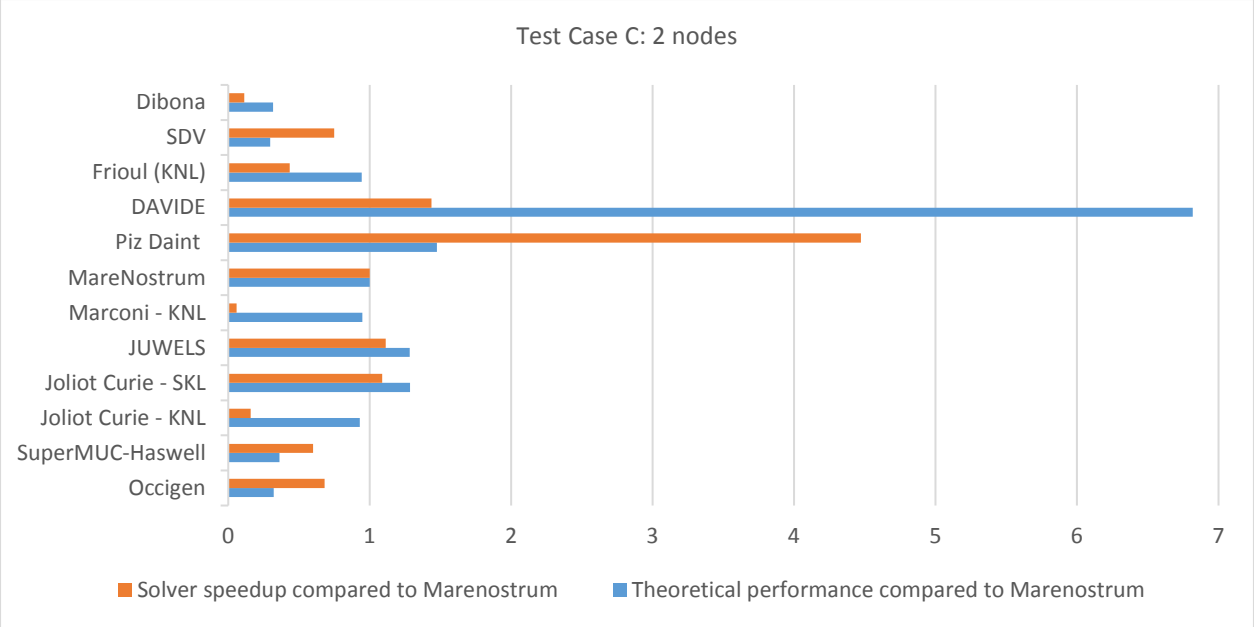


Figure 31: Solver speedup and theoretical performance compared to MareNostrum4 on 2 nodes

The results obtained on Skylake microarchitecture supercomputers are consistent, we obtain decent and homogeneous performance on all three machines. This newer (and therefore more expensive) architecture is an interesting solution for performing simulations with SPECFEM3D.

By comparing the theoretical peak performance of the machines to the performance of the solvers, we can see that the SPECFEM3D code is relatively efficient on Haswell and Sandy Bridge nodes (Occigen, Piz Daint, Hazel Hen, SuperMUC). These architectures are therefore good candidates for obtaining correct performance with less expensive systems.

On Knights Landing (KNL) machines (Joliot Curie-KNL, Marconi-KNL) comparing to PRACE-4IP runs [20], performance for the most part systems more than twice as slower. This is due to the fact that a code modified by Intel were used but as Intel didn't release the code publicly, the current public code [6] have been used to carry these performances. It is recommended to avoid using SPECFEM3D with this type of architecture.

GPU-based computations significantly outperform the CPU-based calculations, the code is very well adapted to accelerators to achieve such performance it is essential to use this type of hybrid architecture (CPU+GPU). Performance on Piz Daint (on all test cases) is at least three times better than that of any supercomputer tested.

4.13.4 Energy Efficiency

We collected the power consumed on the Piz Daint and DAVIDE computers because only these two computers had energy measurement tools during the benchmarks. These energy consumptions are shown in Table 83.

Problem	Nodes	Piz Daint - Energy (kJ)	Piz Daint - Performance	DAVIDE - Energy (kJ)	DAVIDE - Performance
Test case C	1	8.7	26.7	89.114241	113
Test case C	2	9.45	16.6	83.03966382	51.7
Test case A	24	1260	195		
Test case B	384	5180	50		

Table 83: Energy to Solution on DAVIDE and Piz Daint

The energy comparison for this code is not obvious since the test cases run on a number of cores depending on the available memory size and a fortiori, we cannot vary the number of processors used as we wish. To facilitate this comparison, we normalised the energy consumption by dividing by the number of nodes used multiplied by the solver time and multiplying by the calculation time for test case C on one node.

Problem	Nodes	Piz Daint - Normalised energy	DAVIDE - Normalised energy
Test case C	1	8.7	89.114241
Test case C	2	7.599849398	41.51983191
Test case A	24	7.188461538	
Test case B	384	7.2034375	

Table 84: Normalised energy consumption on DAVIDE and Piz Daint

The results in Table 83 and Table 84 highlight the consistency of Piz Daint's energy consumption, where consumption remains linear for an increasing number of nodes. For DAVIDE, the consumption goes from simple to double for the test case C; indeed, on Table 83 we notice that the consumption remains identical whether we turn on one or two nodes.

5 Conclusions

The whole purpose of benchmarking is providing a metric for comparing systems. Clearly, one single (application) benchmark won't 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 85. 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 Skylake systems, the Broadwell and Haswell systems, the Knights Landing systems. Piz Daint is the only GPU based system and is most relevant for applications that can exploit the GPUs. SuperMUC-NG, the latest and greatest Skylake system, even has a higher relative core performance than Piz Daint. Unfortunately, it was not yet available for our benchmarking activities.

PRACE Tier-0 system	R_{peak} (Pflop/s)	R_{max} (Pflop/s)	Cores	R_{max} per core (Gflop/s/core)	Relative core performance
SuperMUC-NG	26.874	19.477	305,856	63.68	1.00
Piz Daint	27.154	21.230	387,872	54.73	0.86
JUWELS	9.891	6.178	114,480	53.96	0.85
Irene-SKL	6.636	4.066	79,488	51.15	0.80
MareNostrum4	10.296	6.471	153,216	42.23	0.66
Marconi Broadwell	2.003	1.724	54,432	31.67	0.50
Hazel Hen	7.404	5.640	185,088	30.47	0.48
Marconi-KNL	18.816	10.385	348,000	29.84	0.47
Irene-KNL	2.340	1.311	56,304	23.29	0.37

Table 85: 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 / Data Set / 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. To 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 86. We did not include SHOC in these results since these synthetic benchmarks are all single node and GPU only. To get a fair comparison of the GPU based Piz Daint and DAVIDE results, we used performance results with the same number of GPUs where possible.

Application	Test Case	Piz Daint-HSW	JUWELS	MareNostrum4	Sisu	Hazel Hen	Irene-SKL	Piz Daint-P100	Marconi-SKL	DAVIDE-P8	Marconi-BDW	DAVIDE	Dibona	Frioul	Marconi-KNL	SDV	Irene-KNL
Alya	A		1.00	0.93				0.36					0.14		0.65	0.35	
	B		1.00	0.80				0.79							0.33		
	C											0.67		1.00			
Code_Saturne	A	0.91	1.00	0.69		0.96	0.68	0.14				0.14	0.18	0.19	0.18		0.15
	B	1.00	0.55	0.39		0.69	0.48	0.13							0.18		0.15
CP2K	A	1.00	0.57					0.20					0.27			0.32	
	B	1.00	0.95					0.18		0.85		0.06	0.53	0.06		0.30	
	C	1.00	0.89					0.36		0.15			0.31	0.31		0.31	
GADGET	A		1.00	0.84									0.65		0.19		
	B		1.00	0.95									0.71		0.23		
GPAW	S		1.00	0.69	0.94			0.14		0.74		0.22		0.10			
	M		0.44	0.34	0.32					0.06				1.00			
	L		1.00	0.77	0.80					0.13							
GROMACS	B		1.00	0.68		0.73	0.83	0.54				0.25		0.25	0.22	0.10	0.18
NAMD	B		1.00	0.47		0.95	0.62	0.41				0.61		0.15	0.14	0.12	
NEMO	G		0.89	1.00		0.82											
PFARM	1		1.00	0.84		0.38	0.82	0.69				0.14	0.13	0.33	0.32	0.29	0.24
	2		1.00	0.83		0.46	0.79	0.60				0.13	0.16	0.27	0.26	0.28	0.24
QCD	1		0.99	1.00			0.81	0.89				0.92	0.70	0.30	0.01	0.07	0.48
	2v1		1.00	0.99			0.75	0.90				0.72	0.52	0.27	0.24	0.04	
	2v2		0.60	0.59			0.56	1.00							0.31	0.12	
Quantum Espresso	S		0.96	0.50				1.00		0.67	0.76	0.57		0.24		0.18	
	L		1.00						0.56	0.98	0.21	0.51		0.17	0.18		
SPECFEM3D	C		0.35	0.32			0.34	1.00				0.08	0.03	0.10	0.01	0.18	0.04
	A		0.42	0.37		0.23	0.38	1.00					0.05	0.10	0.12		0.12
	B		0.37	0.45			0.42	1.00							0.04		0.15

Table 86: Selected relative speed per application-dataset combination

Looking at the green colours, the clear overall winners are:

1. The Skylake based JUWELS and MareNostrum4.
2. The Haswell based Sisu and Hazel Hen.
3. The GPU based Piz Daint is a clear winner for SPECFEM3D and is competitive for QCD.

The few CPU only results on the Haswell based Piz Daint show an even higher performance per core – probably because of the available interconnect bandwidth per core – but they leave the GPUs unused, which is a waste of resources.

For the GPU based systems, in general Piz Daint shows a higher performance than DAVIDE when using the same number of GPUs.

The POWER8 based DAVIDE (CPU only) results for CP2K (Test Case B) and Quantum Espresso (Large Test Case) are competitive with the JUWELS and Piz Daint results.

The ARM based Dibona system shows a good performance for GADGET and QCD (part 1). We are looking forward to the future ARM CPUs with Scalable Vector Extension (SVE) that will make the ARM CPUs even more competitive.

The KNL performance per core is consistently lowest, which is due to the low clock speed. Nowadays, Skylake also has AVX-512, and the only remaining true Xeon Phi differentiator is the 16 GB MCDRAM High Bandwidth Memory (HBM). Furthermore, KNL has been discontinued by Intel. Nevertheless, the KNL based Frioul system is the winner for GPAW Test Case M.

The full picture strongly corresponds with the LINPACK results presented in Section 5.1.1 where Piz Daint is marginally faster per core than JUWELS and Irene-SKL. (As stated earlier: SuperMUC-NG was not yet available for benchmarking.) 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.

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 87. In the last column we provide the relative energy efficiency (normalised using the maximum value). The ranking is as expected. The GPU based system Piz Daint is at least twice as energy efficient as all other non-accelerated systems. Next are the recent Skylake systems and the Knights Landing system. The oldest Haswell and Broadwell based systems are least energy efficient.

PRACE Tier-0 system	R_{max} (Pflop/s)	Power (kW)	Power Efficiency (Gflop/J)	Relative power efficiency
Piz Daint	21.230	2384	8904	1.00
JUWELS	6.178	1361	4539	0.51
Irene-SKL	4.066	917	4434	0.50
Irene-KNL	1.311	326	4022	0.45
MareNostrum4	6.471	1632	3965	0.45
Hazel Hen	5.640	3615	1560	0.18
Marconi Broadwell	1.724	1360	1268	0.14

Table 87: Green500 energy efficiency of PRACE Tier-0 systems

5.2.2 Energy to Solution on DAVIDE and Frioul (in their original PCP-configuration)

As mentioned in Section 3.2.2., the energy measurements tools initially present on Frioul are no longer available since late 2018. Because of this, we were not able to provide Frioul energy measurements for some of the applications. To nevertheless give some insight on the relative energy efficiency of DAVIDE and Frioul in their original PCP-configuration, we give another presentation of the energy to solution as published in the Wrap-up table in PRACE-4IP D7.7 (cf. [21]). (The original table presents a summary of all measurements and the authors refer to the relevant benchmark results in the preceding sections for an interpretation.) In Table 88 we present the energy to solution for the test cases where both DAVIDE and Frioul results were provided. 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, CP2K Test Case 1 is an outlier. This was due to excessive I/O.

Application	Test Case	Relative Energy to Solution	
		DAVIDE	Frioul
Alya	1	4.11	1.00
	2	3.88	1.00
Code_Saturne	1	1.00	4.27
CP2K	1	1.76	1.00
	2	9.32	1.00
GROMACS	1	1.36	1.00
	2	1.00	1.19
NAMD	1	3.38	1.00
	2	1.00	3.14
PFARM	1	1.00	1.96
QCD part 1	1	1.00	2.20
	2	1.00	5.50
QCD part 2	1	1.00	3.34
Quantum Espresso	1	1.00	3.84
	2	1.00	3.08
SPECFEM3D Globe	1	1.00	2.08
Average		2.11	2.29
Average w/o CP2K		1.63	2.37

Table 88: Relative energy to solution on DAVIDE and Frioul in their original PCP-configuration

Overall, it is shown that the most energy efficient system is heavily application dependent. Giving all benchmarks an equal weight and taking the average Relative Energy to Solution, shows that DAVIDE overall is slightly more energy efficient than Frioul. The difference is somewhat larger if we exclude the CP2K outlier.

5.2.3 Energy to Solution on DAVIDE, Frioul and Piz Daint

In Table 89 we selected Energy to Solution measurements from Section 4 combined with the measurements also presented in Table 88. We only selected benchmarks with at least two measurements where at least one of them is on a non-PCP prototype. To give a good comparison we selected benchmarks with the largest system coverage. Since the energy efficiency also depends on the number of nodes, we also tried to cover different node counts. Similarly, to what has been done in Table 88, we normalised and colour coded the energy to solution to come to a relative energy to solution.

Application	Test Case	nodes	DAVIDE	DAVIDE CPU only	Frioul	Piz Daint	Piz Daint CPU only
Code_Saturne	A	4	1.02				1.00
CP2K	A	1				1.00	1.39
CP2K	A	128				1.84	1.00
CP2K	B	16	3.48	2.78	1.33	1.00	1.00
CP2K	C	16		30.59	3.28	1.00	2.31
GPAW	S	1	1.52	7.48	2.55	1.00	
GPAW	S	8		1.00	6.17	1.08	
PFARM	1	1	6.80			1.00	
PFARM	1	4	1.41			1.00	
PFARM	2	1	6.31			1.00	
PFARM	2	4	3.86			1.00	
Quantum Espresso	S	4	4.96	18.89	11.74	1.00	
Quantum Espresso	L	10	1.00	3.61	2.00		
SPECFEM3D	C	2	8.79			1.00	

Table 89: Selected relative energy to solution measurements

Overall, Piz Daint is the most energy efficient system. There are only two exceptions: CP2K on 128 nodes runs more efficiently using only the CPUs. This is to be expected since in this case the GPUs are underutilised but still using a lot of energy. For GPAW on 8 nodes, the CPU only run on DAVIDE wins. This is somewhat comparable to the CP2K case. The runner up is the other GPU system: DAVIDE. The reason is clear: DAVIDE uses the same NVIDIA P100 GPUs but it has 4 GPUs (and two CPUs) per node whereas Piz Daint only has one GPU (and one CPU) per node. For perfectly scaling codes, the four GPUS in a DAVIDE node are expected to use four times the energy of one GPU in a Piz Daint node. Except for PFARM we see that the smallest problem sizes on DAVIDE use some 4 times or more energy compared to Piz Daint.

Acknowledgements

We gratefully acknowledge the support of the DEEP-ER project. We are especially grateful for the support we got from Sebastian Lührs and Estela Suarez from the Jülich Supercomputing Centre (Forschungszentrum Jülich). The results presented here have been (partially) performed on the DEEP-ER SDV prototype, which was built with funding from the European Commission's FP7 Programme, under Grant Agreements n° 610476.

We gratefully acknowledge the support of the Mont-Blanc 3 project. We are especially grateful for the support we got from Etienne Walter and Joël Wanza-Weloli from Atos France. The results presented here have been (partially) performed on the Mont-Blanc 3 Dibona prototype, which was built with funding from the European Commission’s Horizon 2020 Research and Innovation Programme under Grant Agreement n° 671697.