SEVENTH FRAMEWORK PROGRAMME
Research Infrastructures

INFRA-2011-2.3.5 – Second Implementation Phase of the European High Performance Computing (HPC) service PRACE

PRACE-2IP

PRACE Second Implementation Project

Grant Agreement Number: RI-283493

D8.1.2
Performance Model of Community Codes
Final

Version: 1.0
Author(s): Claudio Gheller, Will Sawyer, Thomas Schulthess, CSCS; Fabio Affinito, CINECA; Ivan Girotto, Alastair McKinstry, Filippo Spiga, ICHEC; Laurent Crouzet, CEA; Andy Sunderland, STFC; Giannis Koutsou, Abdou Abdel-Rehim, CASTORC; Fernando Nogueira, Miguel Avillez , UC-LCA; Georg Huhs, José María Cela, and Mohammad Jowkar, BSC.
Date: 24.11.2011

Project and Deliverable Information Sheet

PRACE Project	**Project Ref. №: RI-283493**	
	Project Title: PRACE Second Implementation Project	
	Project Web Site: http://www.prace-project.eu	
	Deliverable ID: **D8.1.2**	
	Deliverable Nature: Report	
	Deliverable Level: PU	**Contractual Date of Delivery:** 30 / 11 / 2011
		Actual Date of Delivery: 30 / 11 / 2011
	EC Project Officer: Bernhard Fabianek	

* - The dissemination level are indicated as follows: **PU** – Public, **PP** – Restricted to other participants (including the Commission Services), **RE** – Restricted to a group specified by the consortium (including the Commission Services). **CO** – Confidential, only for members of the consortium (including the Commission Services).

Document Control Sheet

Document	**Title: Performance Model of Community Codes**	
	ID: **D8.1.2**	
	Version: 1.0	**Status:** Final
	Available at: http://www.prace-project.eu	
	Software Tool: Microsoft Word 2007	
	File(s): D8.1.2.docx	
Authorship	**Written by:**	Claudio Gheller, Will Sawyer
	Contributors:	Thomas Schulthess, CSCS; Fabio Affinito, CINECA; Ivan Girotto, Alastair McKinstry, Filippo Spiga, ICHEC; Laurent Crouzet, CEA; Andy Sunderland, STFC; Giannis Koutsou, Abdou Abdel-Rehim, CASTORC; Fernando Nogueira, Miguel Avillez , UC-LCA; Georg Huhs), José María Cela, and Mohammad Jowkar, BSC
	Reviewed by:	Aad van der Steen; Dietmar Erwin
	Approved by:	MB/TB

Document Status Sheet

Version	Date	Status	Comments
0.1	04/10/2011	Draft	Document skeleton
0.2	19/10/2011	Draft	Draft distributed to task leaders
0.3	24/10/2011	Draft	First version of the performance modelling methodology
0.4	25/10/2011	Draft	First benchmarks collected
0.6	28/10/2011	Draft	Introduction and Section1 improved
0.8	5/11/2011	Draft	Most benchmarks collected
0.9	9/11/2011	Draft	Extensive proofreading
1.0	30/11/2011	Final version	

Document Keywords

Keywords:	PRACE, HPC, Research Infrastructure, scientific applications, libraries, performance modelling.

Table of Contents

List of Figures

List of Tables

References and Applicable Documents

[1] http://www.prace-project.eu
[2] Deliverable D8.1.1: “Community Codes Development Proposal”
[3] High Performance and High Productivity Computing Initiative, http://hp2c.ch.
[4] DeRose, L.; B. Homer, D. Johnson, S. Kaufmann and H. Poxon: Cray Performance Analysis Tools. In: Tools for High Performance Computing, 191-199. Springer-Verlag. 2008.
[5] Schende, S.S.; A.D. Malony: The TAU Parallel Performance System. Int. J. High Perf. Comput. Appl. 20, 287-311. 2006.
[6] Wolf, F.; B.J.N. Wylie, E. Ábrahám, D. Becker, W. Frings, K. Fürlinger, M. Geimer, M.-A. Hermanns, B. Mohr, S. Moore, M. Pfeifer, and Z. Szebenyi: Usage of the SCALASCA toolset for scalable performance analysis of large-scale parallel applications. In: Tools for High Performance Computing, 157—167. Springer-Verlag. 2008.
[7] http://web.me.com/romain.teyssier/Site/RAMSES.html
[8] http://user.cscs.ch/hardware/rosa_cray_xt5/index.html
[9] https://hpcforge.org/projects/pkdgrav2/
[10] J. Barnes and P. Hut (December 1986). "A hierarchical O(N log N) force-calculation algorithm". Nature 324 (4): 446-44
[11] Ewald P. (1921) "Die Berechnung optischer und elektrostatischer Gitterpotentiale", Ann. Phys. 369, 253–287.
[12] P G Burke, C J Noble and V M Burke, Adv. At. Mol. Opt. Phys. 54 (2007) 237-318.
[13] K L Baluja, P G Burke and L A Morgan, CPC 27 (1982), 299-307.
[14] A G Sunderland, C J Noble, V M Burke and P G Burke, CPC 145 (2002), 311-340.
[15] Future Proof Parallelism for Electron-Atom Scattering Codes with PRMAT, A. Sunderland, C. Noble, M. Plummer, http://www.hector.ac.uk/cse/distributedcse/reports/prmat/
[16] Single Node Performance Analysis of Applications on HPCx, M. Bull, HPCx Technical Report HPCxTR0703 (2007), http://www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0703.pdf
[17] Rockel, B.; A. Will and A. Hense: The Regional Climate Model COSMO-CLM (CCLM). Meteorologische Zeitschrift 17(4), 347-348. 2008.
[18] S. Valcke; Directions for a community coupler for ENES. CERFACS internal report, 2011.
[19] A. Gassmann, and H.-J. Herzog, Towards a consistent numerical compressible non-hydrostatic model using generalized Hamiltonian tools, Q.J.R.Meteorol.S., 134, 1597-1613, 2008.
[20] Hazeleger W., et al., EC-Earth V2: description and validation of a new seamless Earth system prediction model. Submitted. http://ecearth.knmi.nl/Hazelegeretal.pdf
[21] Collins, M.; Tett, S.F.B., and Cooper, C.: "The internal climate variability of HadCM3, a version of the Hadley Centre coupled model without flux adjustments". Climate Dynamics 17: 61–81. 2001.
[22] Giorgetta, M.A.; G. P. Brasseur, E. Roeckner, and J. Marotzke, Preface to Special Section on Climate Models at the Max Planck Institute for Meteorology, J. Climate, 19, 3769-3770, 2006.
[23] Prusa, J.M.; P.K. Smolarkiewicz, and A.A. Wyszogrodzki: EULAG, a computational model for multiscale flows. Comput. Fluids., 37, 1193-1207. 2008.
[24] J. M. Dennis and J. Edwards and R. Loy and R. Jacob and A. A. Mirin and A. P. Craig and M. Vertenstein, 2011: "An Application Level Parallel I/O Library for Earth System Models", Int. J. High Perf. Comput. Appl., Accepted.
[25] Craig, A.; M. Vertenstein and R. Jacob: “A new flexible coupler for earth system modeling developed for CCSM4 and CESM1”, Int. J. High Perf. Comput. Appl.. In press.
[26] ICON testbed; https://code.zmaw.de/projects/icontestbed
[27] ICOMEX project; http://wr.informatik.uni-hamburg.de/research/projects/icomex

[28] Williams, S.; A. Waterman, and D. Patterson, "Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures", Communications of the ACM (CACM), April 2009.
[29] Conti, C; W. Sawyer: GPU Accelerated Computation of the ICON Model. CSCS Internal Report, 2011.
[30] Skamarock, W.C.; J.B. Klemp, M.G. Duda, L.Fowler, S.-H. Park and T.D. Ringler: A Multi-scale Nonhydrostatic Atmospheric Model Using Centroidal Voronoi Tesselations and C-Grid Staggering. Submitted to Mon. Wea. Rev., 2011.
[31] Satoh, M.; T. Matsuno, H. Tomita, H. Miura, T. Nasuno, S. Iga: Nonhydrostatic icosahedral atmospheric model (NICAM) for global cloud resolving simulations. J. of Comp. Phys. 227(7), 3486-3514. 2008.
[32] DYNAMICO project: http://www.lmd.polytechnique.fr/~dubos/DYNAMICO
[33] Gung-Ho project: http://www.metoffice.gov.uk/research/areas/dynamics/next-generation
[34] Lauritzen, P.H.; C. Jablonowski, M. Taylor and R.D. Nair: Rotated versions of the Jablonowski steady-state and baroclinic wave test cases: A dynamical core intercomparison. J. Adv. Model. Earth Syst., Vol. 2 Art. 15, 2010.
[35] Madec, G: NEMO ocean engine, Note du Pole de modélisation, Institut Pierre-Simon Laplace (IPSL), France, No 27 ISSN No 1288-1619, 2008.
[36] Pain, C.C.; M.D. Piggot, A.J.H. Goddard, F. Fang, G.J. Gorman, D.P. Marshall, M.D. Eaton, P.W. Power, and C.R.E. de Oliveira: Three-dimensional unstructured mesh ocean modelling. Ocean Modelling, 10(1-2), 5-33, 2005.
[37] Rew, R; G. Davis: NetCDF: an interface for scientific data access. Computer Graphics and Applications, IEEE 10(4), 76-82, 1990.
[38] Li, J.; W.-K Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale: Parallel netCDF: A High-Performance Scientific I/O Interface. In *Proceedings of the 2003 ACM/IEEE conference on Supercomputing* (SC '03), 39-49, 2003.
[39] Next Generation Weather & Climate Prediction, accessed 9th Nov. 2011: http://www.nerc.ac.uk/research/programmes/ngwcp/
[40] HARNESS Fault Tolerant MPI, accessed 9th Nov. 2011: http://icl.cs.utk.edu/ftmpi/
[41] Madec,G; M. Imbard: A global ocean mesh to overcome the North Pole singularity, Climate Dynamics, 12,6381-388
[42] Bridging Performance Analysis Tools and Analytic Performance Modeling for HPC, T. Hoefler, Proceedings of Workshop on Productivity and Performance (PROPER 2010), Springer, Dec. 2010.
[43] A Framework for Performance Modeling and Prediction. Allan Snavely , Laura Carrington , Nicole Wolter , Jesus Labarta, Rosa Badia , Avi Purkayastha, Proceedings of the 2002 ACM/IEEE conference on Supercomputing.
[44] Performance Modeling: Understanding the Present and Predicting the Future. Bailey, David H.; Snavely, Allan. http://escholarship.org/uc/item/1jp3949m
[45] How Well Can Simple Metrics Represent the Performance of HPC Applications? Laura C. Carrington, Michael Laurenzano, Allan Snavely, Roy L. Campbell, Larry P. Davis; Proceedings of the 2005 ACM/IEEE conference on Supercomputing, 2005, IEEE Computer Society
[46] http://www.hp2c.ch/
[47] http://www.enes.org
[48] http://icl.cs.utk.edu/ftmpi/overview/index.html
[49] http://www.deisa.eu/science/benchmarking/codes/nemo
[50] http://qe-forge.org/
[51] http://www-zeuthen.desy.de/~kjansen/etmc/
[52] http://www.exciting.physics.at

[53] http://exciting-code.org
[54] http://elk.sourceforge.net
[55] http://www.netlib.org/scalapack/
[56] http://www.netlib.org/blacs/
[57] http://netlib.org/blas/
[58] https://verc.enes.org/models/software-tools/oasis
[59] http://www.unidata.ucar.edu/software/netcdf/
[60] http://www.grib.us/
[61] http://www.cs.virginia.edu/stream/
[62] http://icl.cs.utk.edu/ftmpi/
[63] http://www-meom.hmg.inpg.fr/Web/Projets/DRAKKAR/
[64] http://www.hector.ac.uk/
[65] http://www.mcs.anl.gov/petsc/
[66] http://www.pwscf.org/home.htm
[67] http://www.quantum-espresso.org/
[68] http://www.icmab.es/dmmis/leem/siesta/
[69] http://www.abinit.org/
[70] http://exciting-code.org/
[71] http://elk.sourceforge.net/
[72] http://www.tddft.org/programs/octopus/wiki/index.php/Main_Page
[73] http://www.yambo-code.org/
[74] http://www.gnu.org/copyleft/gpl.html
[75] http://icl.cs.utk.edu/magma/
[76] http://www-ccrt.cea.fr/fr/moyen_de_calcul/titane.htm
[77] http://software.intel.com/en-us/articles/intel-mkl/
[78] http://inac.cea.fr/L_Sim/BigDFT/
[79] http://www.fftw.org/
[80] http://www.gnu.org/s/gsl/

List of Acronyms and Abbreviations

AMR	Adaptive Mesh Refinement
API	Application Programming Interface
BLAS	Basic Linear Algebra Subprograms
BSC	Barcelona Supercomputing Center (Spain)
CAF	Co-Array Fortran
CCLM	COSMO Climate Limited-area Model
ccNUMA	cache coherent NUMA
CEA	Commissariat à l'Energie Atomique (represented in PRACE by GENCI, France)
CERFACS	The European Centre for Research and Advanced Training in Scientific Computation
CESM	Community Earth System Model, developed at NCAR (USA)
CFD	Computational Fluid Dynamics
CG	Conjugate-Gradient
CINECA	Consorzio Interuniversitario, the largest Italian computing centre (Italy)
CINES	Centre Informatique National de l'Enseignement Supérieur (represented in PRACE by GENCI, France)
CNRS	Centre national de la recherche scientifique

COSMO	Consortium for Small-scale Modeling
CP	Car-Parrinello
CPU	Central Processing Unit
CSC	Finnish IT Centre for Science (Finland)
CSCS	The Swiss National Supercomputing Centre (represented in PRACE by ETHZ, Switzerland)
CUDA	Compute Unified Device Architecture (NVIDIA)
DEISA	Distributed European Infrastructure for Supercomputing Applications. EU project by leading national HPC centres.
DFPT	Density-Functional Perturbation Theory
DFT	Discrete Fourier Transform
DGEMM	Double precision General Matrix Multiply
DKRZ	Deutsches Klimarechenzentum
DP	Double Precision, usually 64-bit floating-point numbers
DRAM	Dynamic Random Access memory
EC	European Community
ENES	European Network for Earth System Modelling
EPCC	Edinburgh Parallel Computing Centre (represented in PRACE by EPSRC, United Kingdom)
EPSRC	The Engineering and Physical Sciences Research Council (United Kingdom)
ESM	Earth System Model
ETHZ	Eidgenössische Technische Hochschule Zürich, ETH Zurich (Switzerland)
ETMC	European Twisted Mass Collaboration
ETSF	European Theoretical Spectroscopy Facility
ESFRI	European Strategy Forum on Research Infrastructures; created roadmap for pan-European Research Infrastructure.
FFT	Fast Fourier Transform
FP	Floating-Point
FPGA	Field Programmable Gate Array
FPU	Floating-Point Unit
FT-MPI	Fault Tolerant Message Passing Interface
FZJ	Forschungszentrum Jülich (Germany)
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
GCS	Gauss Centre for Supercomputing (Germany)
GENCI	Grand Equipement National de Calcul Intensif (France)
GFlop/s	Giga (= 10^9) Floating-point operations (usually in 64-bit, i.e., DP) per second, also GF/s
GGA	Generalised Gradient Approximations
GHz	Giga (= 10^9) Hertz, frequency =10^9 periods or clock cycles per second
GNU	GNU's not Unix, a free OS
GPGPU	General Purpose GPU
GPL	GNU General Public Licence
GPU	Graphic Processing Unit
HDD	Hard Disk Drive
HMPP	Hybrid Multi-core Parallel Programming (CAPS enterprise)
HPC	High Performance Computing; Computing at a high performance level at any given time; often used synonym with Supercomputing

HPL	High Performance LINPACK
ICOM	Imperial College Ocean Model
ICON	Icosahedral Non-hydrostatic model
IDRIS	Institut du Développement et des Ressources en Informatique Scientifique (represented in PRACE by GENCI, France)
IEEE	Institute of Electrical and Electronic Engineers
IESP	International Exascale Project
I/O	Input/Output
IPSL	Institut Pierre Simon Laplace
JSC	Jülich Supercomputing Centre (FZJ, Germany)
KB	Kilo (= 2^{10} ~10^3) Bytes (= 8 bits), also KByte
LBE	Lattice Boltzmann Equation
LINPACK	Software library for Linear Algebra
LQCD	Lattice QCD
LRZ	Leibniz Supercomputing Centre (Garching, Germany)
NEMO	Nucleus for European Modeling of the Ocean
NERC	Natural Environment Research Council (United Kingdom)
NCAR	National Center for Atmospheric Research
MB	Mega (= 2^{20} ~ 10^6) Bytes (= 8 bits), also MByte
MB/s	Mega (= 10^6) Bytes (= 8 bits) per second, also MByte/s
MBPT	Many-Body Perturbation Theory
MCT	Model Coupling Toolkit, developed at Argonne National Lab. (USA)
MD	Molecular Dynamics
MFlop/s	Mega (= 10^6) Floating-point operations (usually in 64-bit, i.e., DP) per second, also MF/s
MHz	Mega (= 10^6) Hertz, frequency =10^6 periods or clock cycles per second
MIPS	Originally Microprocessor without Interlocked Pipeline Stages; a RISC processor architecture developed by MIPS Technology
MKL	Math Kernel Library (Intel)
MPI	Message Passing Interface
MPI-IO	Message Passing Interface – Input/Output
MPIM	MPI for Mathematics
MPP	Massively Parallel Processing (or Processor)
MPT	Message Passing Toolkit
NCF	Netherlands Computing Facilities (Netherlands)
NEGF	non-equilibrium Green's functions,
NERC	Natural Environment Research Council
OpenCL	Open Computing Language
Open MP	Open Multi-Processing
OS	Operating System
PAW	Projector Augmented-Wave
PGI	Portland Group, Inc.
PGAS	Partitioned Global Address Space
PIMD	Path-Integral Molecular Dynamics
POSIX	Portable OS Interface for Unix
PPE	PowerPC Processor Element (in a Cell processor)
PRACE	Partnership for Advanced Computing in Europe; Project Acronym
PSNC	Poznan Supercomputing and Networking Centre (Poland)
PWscf	Plane-Wave Self-Consistent Field
QCD	Quantum Chromodynamics

QR	QR method or algorithm: a procedure in linear algebra to factorise a matrix into a product of an orthogonal and an upper triangular matrix
RAM	Random Access Memory
RDMA	Remote Data Memory Access
RISC	Reduce Instruction Set Computer
RPM	Revolution per Minute
SGEMM	Single precision General Matrix Multiply, subroutine in the BLAS
SHMEM	Share Memory access library (Cray)
SIMD	Single Instruction Multiple Data
SM	Streaming Multiprocessor, also Subnet Manager
SMP	Symmetric MultiProcessing
SP	Single Precision, usually 32-bit floating-point numbers
SPH	Smoothed Particle Hydrodynamics
STFC	Science and Technology Facilities Council (represented in PRACE by EPSRC, United Kingdom)
STRATOS	PRACE advisory group for STRAtegic TechnOlogieS
TB	Tera ($=2^{40} \sim 10^{12}$) Bytes (= 8 bits), also TByte
TDDFT	Time-dependent density functional theory
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
UPC	Unified Parallel C

Executive Summary

This document presents the results achieved by PRACE-2IP [1] Work Package 8 at PM3. Scientific communities, selected during the first project month, proposed a number of codes relevant for the scientific domain and promising in terms of potential performance improvement on the coming generation of supercomputing architectures. In order to analyse the performance features of these codes, a methodology, based on the performance modelling approach, has been defined and adopted. This methodology relies on the analytic modelling of the main algorithms (characterising the dependency from the critical model parameters) and on the performance analysis, based on the usage of performance tools. The performance modelling approach allows studying the current behaviour of a code, emphasising performance and bottlenecks. But it is also a predictive tool, allowing the estimation of the code's behaviour on different computing architectures, and identifying the most promising areas for performance improvement.

The first step of the modelling is represented by performance analysis. This analysis was accomplished for all the proposed codes, with detailed data generated and collected. The overall results are presented for each scientific domain and code. Most of the data was generated for "real cases", i.e., running the codes for real scientifically meaningful cases, in order to evaluate performances and bottlenecks in daily usage configurations, and to achieve a performance impact through code refactoring and optimisation of the crucial sections of each code. In the following step of WP8, this data will be synthesised and modelled, in order to identify the most promising numerical kernels for performance improvement. This will be the subject of the coming deliverable D8.1.3 "Prototype Codes Exploring Performance Improvements"

1. Introduction

In the first month of Work Package 8 (hereafter WP8), four scientific domains, Astrophysics, Climate, Material Science and Particle Physics, have been identified as areas on which WP8 can have a extraordinary impact. The final objective was to select, within these domains, a number of representative communities, i.e., research groups acting jointly in a given research field, developing some of the most popular scientific codes, and willing to actively invest in software refactoring and algorithm re-engineering in synergy with PRACE-2IP partners within the framework of WP8. In this way, scientific teams and HPC experts cooperate in order to design and implement a new generation of software tools with outstanding scientific features and, at the same time, capable of effectively exploiting the coming HPC systems.

A sound and proper selection of the communities was a key achievement for the work package. Since they have the best grasp of applications and algorithms, these communities not only specify the scientific challenges, but also address the selection and the implementation of the simulation codes. Such selection had also to be prompt, in order to leave as much time as possible to the development phase, where most of the WP effort has to focus.

A successful selection was accomplished at the end of PM1, as reported in deliverable D8.1.1 [2]. The first immediate step made by the communities was the proposal of a number of relevant codes that could be interesting for WP8. Among these codes, only those deemed ready for a refactoring effort were selected.

For an objective and quantitatively motivated selection, a detailed performance analysis was necessary. Due to the broad spectrum of applications under investigation, a general and powerful methodology had to be specified. An appropriate methodology was defined, based on the "Performance Modelling" [42][43][44][45] approach. Performance modelling has the goal of gaining insight into a application's performance on a given computer system. This is achieved first by measurement and analysis, and then by the synthesis of the application characteristics in order to understand the details of the performance phenomena involved, and to project performance to other systems. Therefore, performance modelling not only allows to study the current behaviour of a code, emphasising performances and bottlenecks, but represent a predictive tool, estimating the behaviour on a different computing architecture and identifying the most promising areas for performance improvement.

The adopted Performance Modelling approach will be described in more details in Section 2.

The first step to model the proposed codes was the analysis of the performances, using standard performance tools, collecting all the information that are necessary to understand the behaviour of the main code's algorithms and their dependencies from the relevant model parameters (e.g. the number of cells of the computational mesh) and from the hardware. The performance analysis phase and its results are described in details in Sections 3 to 6, where codes from the four selected scientific domains are considered.

For all the codes, it is important to note that:

1. In this document we can present only a synthesis of all the performance data available, in general those that characterise the "coarse grain" behaviour of each application and that can be of interest for the non-expert reader. More specific and detailed information is available and will be exploited in the subsequent modelling phase.
2. Due to the variety of involved application areas, algorithms, numerical approaches, compilers, computing environments, libraries etc., each code was analysed according to its most proper specific methodology, using the most appropriate tools and collecting the most meaningful data. This makes the presentation of the results somehow "untidy" and inhomogeneous, but guarantees that all necessary data was produced.

At the end of this step, collected data will be synthesised and performance-modelled, in order to identify the most promising numerical kernels for performance improvement. This will be the subject of the coming deliverable D8.1.3.

The present deliverable is organised as follows. The adopted performance modelling methodology is presented in Section 2, together with a simple case that exemplifies how the methodology works.

Sections 3 to 6 are dedicated to the presentation of the overall results of the performance analysis. Section 3 is dedicated to Astrophysics codes (RAMSES, PKDGRAV, PFARM); Section 4 is for Climate (OASIS, CDI, XIOS, PIO, ICON, NEMO, ICOM); Section 5 is focused on Material Science codes (ABINIT, Quantum ESPRESSO, Yambo, Siesta, Octopus, EXCITING). Finally, Section 6 describes Particle Physics algorithms performances.

In Section 7, the next steps of WP8 are summarised and the conclusions drawn.

2. The Performance Analysis Methodology

A thorough understanding of the application code performance is crucial for the ultimate success of this work package. While the chosen codes differ greatly in their size, complexity and preparedness for HPC, a common methodology for analysing their performance can be formulated. This methodology relies on the "performance modelling" approach.

The goal of performance modelling is to gain understanding of applications' performance, by means of measurement and analysis, and then to synthesise these results in order to gain greater understanding of the performance phenomena involved and to project performance to other system/application combinations.

Performance modelling of scientific codes is usually performed in three phases: (1) identify the performance-critical input parameters (e.g., the number of particles or cells, the number of iterations, etc.), (2) formulate and test a hypothesis about the performance as function of the performance-critical input parameters, and (3) parameterise the function. Empirical modelling strategies that benchmark parts of the code (kernels) on the target architecture are often employed to maintain human-manageable performance models. Steps (2) and (3) of developing analytic performance models are often performed with the help of performance tools, that allow deep insights into the behaviour of machines and their performance by displaying the performance characteristics of executed applications. Tools allow the determination of bottlenecks and tune applications. They can also guide re-engineering of applications and they are often used to collect the data to design application models. Many codes already contain their own profiling timers. For the codes that do not, either timers can be inserted, or common performance analysis tools like CrayPat [4], TAU [5] or Scalasca [6] can be employed.

Analytic modelling and performance analysis tools cooperate in the performance model. Analytic performance modelling can be seen as top-down approach where the user formulates an expectation based on an algorithm or implementation and tries to validate and parameterise it to predict performance. Performance analysis tools can be seen as a bottom-up approach that records performance artefacts and strive to trace the artefacts back to the original implementation or algorithm.

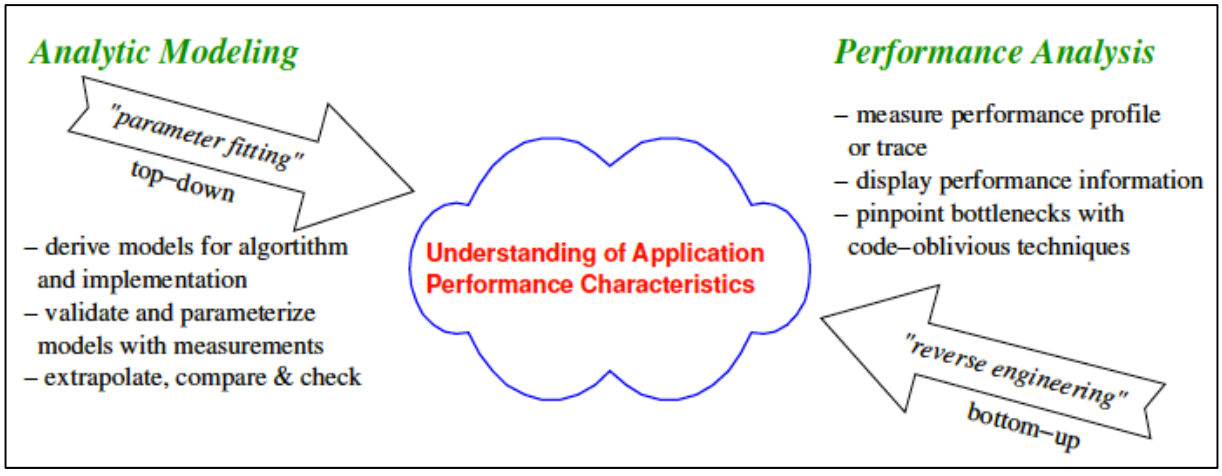


Figure 1: The performance modelling methodology.

We intend to apply the performance model approach for each individual community code chosen, in order to give a precise indication of areas of the code requiring performance improvement. We do not expect that the performance models for all codes have the same degree of sophistication, or that there can be the same degree of profiling presented for each, however a minimum of information necessary to complete the modelling has to be collected.

From the analytical point of view, performance-critical input parameters must be identified. This has to be done by an application expert. Performance-critical input parameters ("critical parameters") can be for example the size of the simulated system or parameters that influence convergence. Other parameters, such as initial starting values (e.g., heats or masses) might not change the runtime of the algorithm and are thus not critical in performance models. More complex parameters such as the shape of the input systems need to be approximated into a single value by the application expert.

For performance analysis we focused two major factors: 1) single processor performance and 2) use of parallel architectures (other factors, although present, are assumed to be negligible for the applications we deal with). Therefore we expect to collect information on:

1. single processor traces and profiles, focusing on floating-point work and usage of the memory sub-system,
2. shared- or distributed-memory parallel performances, focusing on communication, scalability, access to shared memory, hybrid approaches.

Analytical and performance information concurs in defining, for each code, an extended analytical model that can predict performance on a multiple-node and/or multi-core platform, given a minimal number of descriptive parameters, allowing to predict execution time on new systems, given the critical parameters and the characteristic of the target platform (e.g., memory bandwidth, network cross-sectional bandwidth, Flop/s, etc.).

This will allow the selection of the numerical kernels to work on in WP8 and a quantitatively estimation of the benefits of code refactoring on the target architectures, maximising the impact on the community codes of interest.

2.1 Performance Modelling Example

Within the context of the HP2C initiative [46], a pilot project was started to model and subsequently optimise for single node performance. First a highly simplified model was developed for the performance of the fast-wave solver, which is the main kernel of the dynamical core of a European regional numerical weather prediction model COSMO.

The ideas behind the performance model are the following:

- Performance is dominated by the memory access,
- The memory access in the small time steps dominates,
- Only accesses to the 3d arrays are considered,
- The read and write accesses to the variables are counted,
- Assume that all accesses within the inner 2 loops are in L2 cache.

Task	# accesses	# runs per large step	# accesses per large step
Update tendencies	11	3	33
Horizontal integration	17	10	170
Vertical integration	41	10	410
Pre-calculation	50	1	50 (estimation)

Table 1: Number of memory accesses are listed for the calculation of a given 3-D field in different portions of the atmospheric dynamics fast-wave solver. These portions have various calling frequencies.

The number of accesses for portions of the fast-wave solver is listed in **Table 1**. Their weighted sum gives the number of accesses per field element per time step. We assume that the local domain – the horizontal 2-D cross-section of all 3-D fields in the computation – can fit into level 2 cache on the individual core, an often unjustified assumption. If one ignores all computation and considers only the time needed to move data between L2 cache and the FPU, a simple lower bound for execution time can be derived. This lower bound can be tight if the cache assumption is valid, if not, the timings can be off by a large factor.

Due to its simplicity, this model can be quickly modified to represent a different architecture, by changing the cache memory bandwidth parameter. It was the chief motivator for the refactoring of the fast wave solver to recalculate intermediate quantities "on the fly" instead of storing them in intermediate arrays and then reusing them.

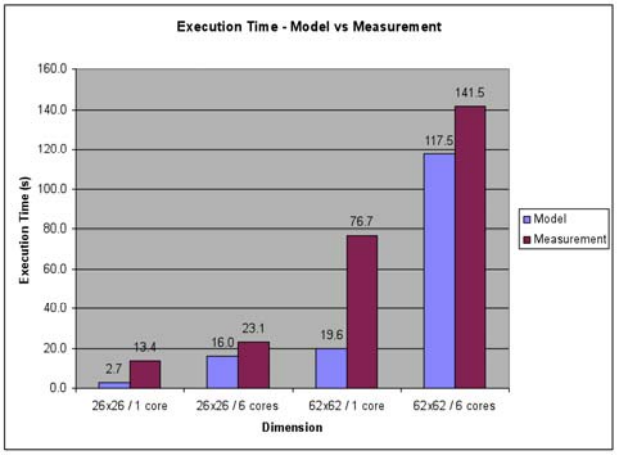


Figure 2: An analytic model based on the memory bandwidth to L2 cache was derived from the number of memory accesses given in Table 1**. The predicted execution time can be considered a lower bound. If the assumption is valid that all arrays associated with the local domain fit into L2 cache, this lower bound is quite tight (see 6 core results), if not, the predicted times can be off by a large factor (e.g., 1 core results).**

3. Performance Analysis of Community Codes: Astrophysics

Several codes have been proposed by the Astrophysics community for the refactoring of the most time consuming routines and their enabling to innovative HPC systems. All of these codes are open source. RAMSES and PKDGRAV are widely used in the astrophysiscs community, the first focusing on problems of galaxy and cluster of galaxies formation and evolution, the second being an outstanding tool for the study of the large scale structure of the universe and for precision cosmology. The PFARM suite is, instead, usually used in material science but, describing electron-atom and electron-ion scattering, it can be effectively adapted and exploited in the astrophysics framework, where the precise and efficient description of atomic physics, an extreme computationally demanding task, is necessary to get, from simulations, results comparable to observational data.

3.1 RAMSES

3.1.1 Description of the code

The RAMSES code was developed in Saclay [7] to study the evolution of the large-scale structure of the universe and the process of galaxy formation. RAMSES is an adaptive mesh refinement (AMR) multi-species code, describing the behaviour of both the baryonic component, represented as a fluid on the cells of the AMR mesh, and the dark matter, represented as a set of collisionless particles. The two matter components interact via gravitational forces. The AMR approach makes it possible to get high spatial resolution only where this is actually required, thus ensuring a minimal memory usage and computational effort.

The main features of the RAMSES code are the following:

1. The AMR grid is built on a tree structure, with new refinements dynamically created (or destroyed) on a cell-by-cell basis, where high spatial resolution is required by the physical problem. This allows greater flexibility to match complicated flow geometries. This property appears to be especially relevant to cosmological simulations, since clumpy structures form and collapse everywhere within the hierarchical clustering scenario; different refinement strategies are implemented: e.g. the “quasi-Lagrangian” criterion, in which the number of dark matter particles per cell remains roughly constant, minimising two-body relaxation and Poisson noise, or criteria based on matter overdensities.
2. The hydrodynamic solver is based on several different shock capturing methods, all ensuring exact total energy conservation, and relying on Riemann solvers, without any artificial viscosity.
3. The dark matter particles dynamics is calculated according to a N-body approach with a Cloud-in-Cell force calculation schema.
4. The gravitational field is calculated solving the Poisson equation with Dirichlet boundary conditions on a Cartesian grid with irregular domain boundaries. This scheme was developed in the context of the AMR schemes based on a graded-octree data structure. The Poisson equation is solved on a level-by-level basis, using a “one-way interface” scheme in which boundary conditions are interpolated from the previous coarser level solution. Such a scheme is particularly well suited for self-gravitating astrophysical flows requiring an adaptive time stepping strategy.
5. Time integration is performed for each level independently, with an adaptive time step algorithm, the time interval being determined by a level dependent stability condition.
6. Magnetic and radiative fields are supported and can be turned on for specific applications.
7. The code is parallelised adopting a MPI-based approach. Domain decomposition is accomplished by mesh partitioning techniques, inspired by parallel tree codes. Several

cell ordering methods (based on space filling curves) are implemented in order to achieve an optimal work-load balancing and to minimise the communication

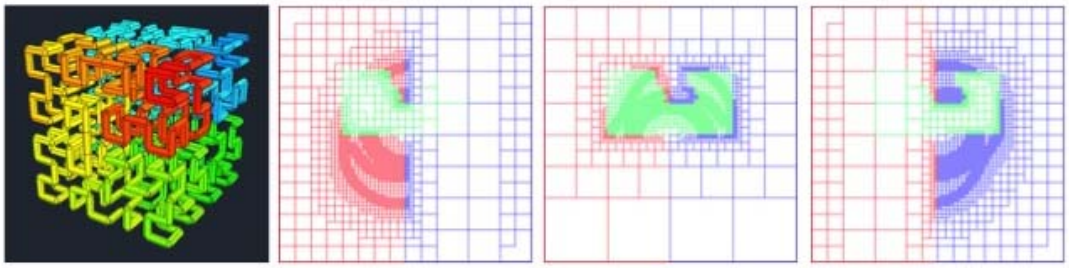


Figure 3: RAMSES domain decomposition based on the Peano-Hilbert curve for AMR based data structure: different colours are assigned to different processors.

3.1.2 Performance Analysis

All the tests presented in this section were run on a 1844 nodes CRAY XT5 [8] system. Each of the compute nodes consists of two 6-core AMD Opteron 2.4 GHz Istanbul processors giving 12 cores in total per node with 16 GBytes of memory.

For each test we present the total time required by the code to complete the test and the fraction of the work spent in the different parts of the code ("sections"), grouped as follows:

- *Hydro*: all the functions needed to solve the hydrodynamic problem are included. Within these functions, we have those that collect from grids at different resolutions the data necessary to update each single cell, those that calculates fluxes to solve conservation equation, the Riemann solver, and the finite-volume solver.
- *Gravity*: this group comprises functions needed to calculate the gravitational potential at different resolutions using a multigrid-relaxation approach
- *N-body*: functions needed to update particles' position and velocity and to evaluate the gravitational force acting on each particle
- *I/O*: functions that read/write data from/to the disk
- Time-stepping: function needed to manage the AMR hierarchy and to control the multiple time step integration sweep
- *MPI* (only in the parallel tests): comprises all the communication related MPI calls (data transfer, synchronisation, management)

Critical parameters analysis

The critical parameters for RAMSES are represented by the number of particles describing the Dark Matter component (Np), the number of cells of the AMR, base mesh, where fluid dynamics data are initialised (Nc), the number of AMR cells, generated during the simulation (N_{AMR}) and the number of refinement levels (N_L). The two N_{AMR} and N_L parameters are clearly related, however it is impossible to find a precise dependency between the two, the AMR grid refining according to the evolution of the system.

Due to the adaptive time stepping, *Hydro* (finite differences) and *Gravity* (multigrid) scale linearly with the total number of cells of the computational mesh ($Nc\backslash N_{AMR}$) at a given level:

$T = A \times N_{AMR}$,

where $A = 2^L$, L being the current AMR Level (L=0 is the base level).

The same behaviour is exhibited by the *N-body* (Particle-Mesh) algorithm that scales as:

$T = A \times N_{(p,a)}$,

$N_{(p,a)}$ being the number of particles active at a given refinement level.

Single core profiling, UNIGRID mode

In this first test, we analyse the behaviour of RAMSES when executed sequentially on a single core of our computing system. This allows the modelling the performance of the code without any influence of communication and network. AMR is switched off (UNIGRID mode), in order to keep memory requirements constant (both the number of particles and the number of cells do not change with time). Three different configuration are analysed: in all the tests the two critical parameters, i.e., the number of particles, Np, is set equal to the number of cells Nc. For the Small test, we set Np = Nc = 32^3, in the Medium test Np = Nc = 64^3, while for the large test, we set: Np = Nc = 128^3.

Problem size	32^3	64^3	128^3
Total Time (sec)	19.3	87.3	733.4
Hydro (%)	37.4	41.9	40.2
Gravity (%)	27.3	31	33.6
N-body (%)	4.3	5.4	5.6
I/O (%)	1	1	1
Time-stepping (%)	5.8	9.9	10.3
Others (%)	24.2	10.8	9.3

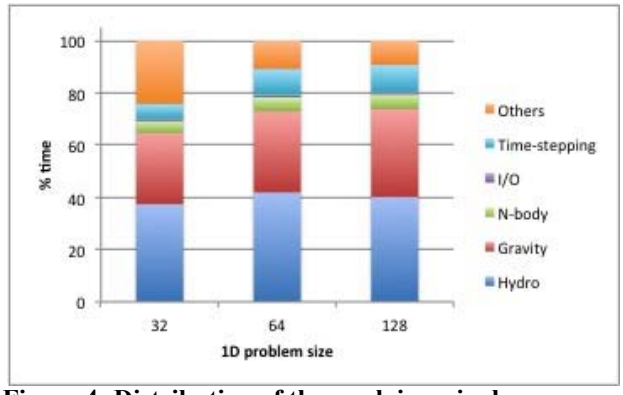


Figure 4: Distribution of the work in a single core run for a UNIGRID setup.

The medium and the large tests are consistent with each other, the total time icreasing by a factor approximately of 8 (as expected by the problem sizes) and the percentage of time spent in each section being roughly the same. The small case is strongly affected by the initial conditions setup (the corresponding times being included in "Others"). However, once "cleaned" from such contributions, the results are consistent with the two larger configurations. Hence, on a single processor no specific dependency from the problem size seems to appear.

Single core profiling, AMR code

The same initial configurations adopted in the previous section, are used with AMR switched on. In this case, the number of particles is constant, while the number of cells changes with time, leading to a variable memory load with memory access performances changing with time. We have performed two different experiments, setting Np=128^3 a base grid with 128^3 cells and 2 and 4 refinement levels respectively:

	2 levels	4 levels
Total Time (sec)	1555.3	1567
Hydro (%)	37.2	37.8
Gravity (%)	41.8	38.9
N-body (%)	2.9	3.2
I/O (%)	1	1
Time-stepping (%)	4.6	5.2
Others (%)	12.5	13.9

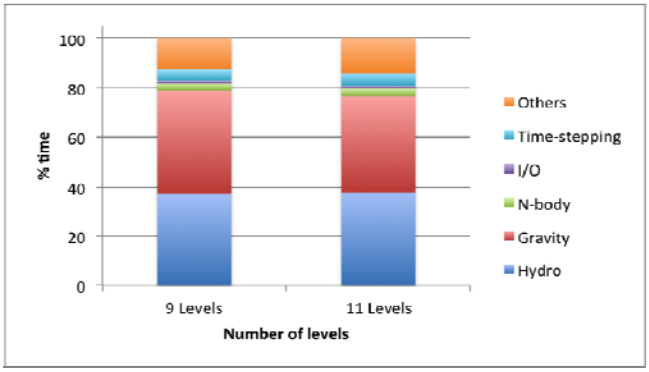


Figure 5: Distribution of the work in a single core run for a AMR setup.

It is interesting to notice that the time to solution is roughly the same in the two cases, although a different number of levels of refinement is adopted. This is mainly due to the particular data structure of RAMSES, that allows to access data irrespectively to the resolution level it belongs to. Furthermore, it is worth noticing that, the total number of refined cells in the "4 levels" test is, at the final step, equal to 199853, while in the "2 levels" test it is 285474, unlike one would expect. This means that, when high resolution is enforced, some processes that at lower resolution are spread over a broad volume, concentrate on narrow regions, strongly decreasing the number of cells necessary to their description.

When AMR is active, *Gravity* becomes the most demanding part of the code, although *Hydro* times tend to increase with the resolution. The *N-body* part, is instead always negligible.

The Time-stepping section appears to be less demanding than in the UNIGRID case. This, however, is true only from a percentage point of view. The wall clock time spent in this section by the AMR, in fact, is, as expected, larger than in the UNIGRID case.

Parallel Profiling, Single Node test

In this test we compare the previous 128^3 sequential runs, UNIGRID and AMR, with the corresponding parallel realisations. Since the test is still small, it can be run on a single node, using up to 8 cores. This preliminary set of tests is needed to verify the impact of parallelism on the application: larger configurations in fact, cannot be run on a single processor, hence a direct comparison with the sequential performances is prevented. For the AMR we adopted 4 levels of refinement.

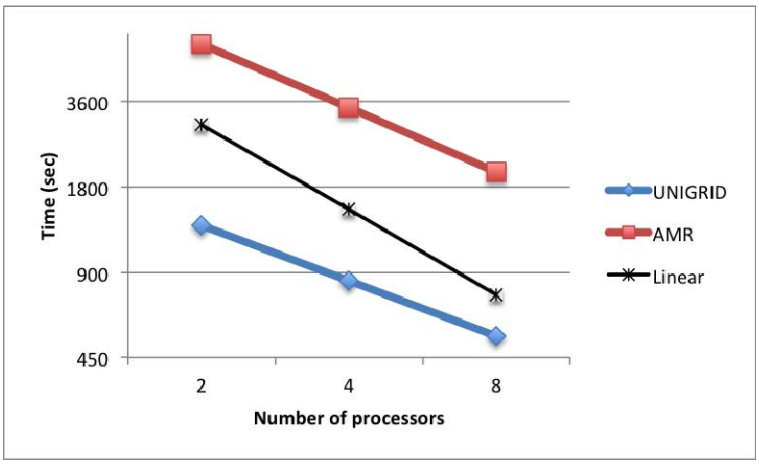


Figure 6: Scalability of a small test both for UNIGRID and AMR set-up described above. Liner scalability (black line) is shown for comparison).

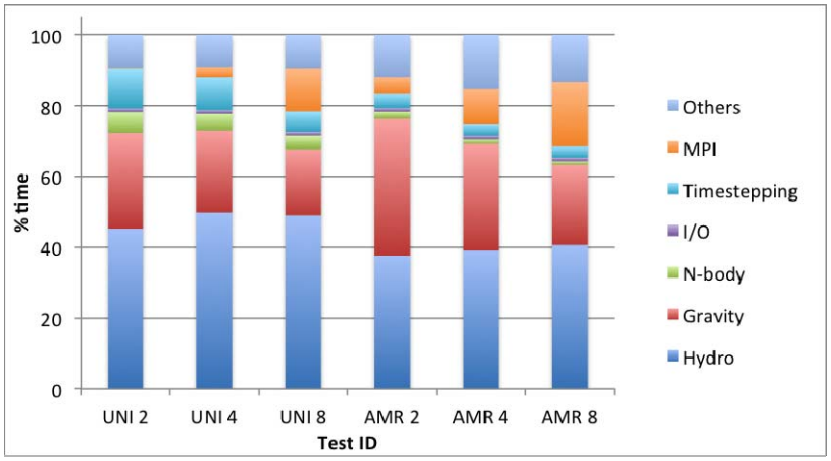


Figure 7: Distribution of the work for the small test as a function of the number of processors.

In Figure 6 we show the scalability of the UNIGRID and AMR cases from 2 to 8 processors. The uniform grid test deviates from the linear behaviour, since, due to the limited grid size, the amount of computation tends to be small with respect to communication, even when 4 cores are used. This is slightly less relevant for the AMR, where the data size is larger due to the presence of refined regions, but, as shown in Figure 7, the MPI overhead is larger due to the unbalanced distribution of work related to the adoption of the multiple time-step approach, that leads to a slower integration of the high resolution region, whose distribution is not perfectly balanced among the processors. This problem in fact grows with the number of cores. It is interesting to notice that for the parallel execution, the *Hydro* part tends to be larger than the gravitational one. The two are instead comparable on a single processor and on two processors for the AMR case. The *N-body* part is always much smaller than *Hydro* and *Gravity*, and becomes completely negligible in the AMR runs, being not subject of any refinement procedure.

Parallel Profiling, Production test

In the last series of test, we analyse the performances of the code in "production" conditions, i.e., with a problem size typical of scientific cosmological simulations. In this case the number of processors is typically larger than 512, in order to have both enough memory and to keep the simulation time (wall clock time) reasonably low (the former being usually the most severe constraint).

We have run a cosmological simulation of a box of 50^3 Mpc, with a base mesh of 512^3 cells and the same number of particles. We have analysed the performances considering 15 time steps starting from an evolved configuration, when AMR is already active and its nested structure populated (see Figure 8, where Level 9 is the last level with a uniform grid - the base grid – while at higher levels AMR is active and cells are generated at increasing resolution)

```
Mesh structure
Level  1 has          1 grids
Level  2 has          8 grids
Level  3 has         64 grids
Level  4 has        512 grids
Level  5 has       4096 grids
Level  6 has      32768 grids
Level  7 has     262144 grids
Level  8 has    2097152 grids
Level  9 has   16777216 grids
Level 10 has   20041282 grids
Level 11 has    7129893 grids
Level 12 has    1822461 grids
Level 13 has     212640 grids
```

Figure 8: AMR structure at the final time step of the production test

In Figure 9 we show the performance analysis of the different code sections when 256, 512 and 1024 cores are used. Due to the memory requirements of this set up, it was not possible to use less than 256 processors. It extremely interesting to notice how, for such large configurations, the MPI communication overhead is large, becoming dominant when 1024 processors are used. A more accurate analysis shows that most of the MPI time is spent in MPI_wait calls, so in synchronising the different MPI threads. In order to give the right interpretation to this result, it is useful to consider the strong scalability curves of the whole code and of its main sections (MPI, *Hydro*, *Gravity*, see Figure 10). It is also interesting to note that, both *Hydro* and *Gravity* scales linearly with the number of processors (*Hydro*, due to cache effects even better than linearly between 512 and 1024 processors). This means that the domain decomposition is well balanced and each processor works approximately on the same number of cells for each level (due to the multiple time stepping approach, higher levels cells requires more work to be integrated). Therefore, the synchronisation overhead cannot be interpreted as an effect of the multiple time stepping approach, together with the difficulty in getting ideal domain decomposition, intrinsic to AMR. Instead, it can be explained as a result of the continuous need for each processor to access information that are stored remotely, both for hydrodynamics and for the gravity. For the former, data locality is not ensured, due to the AMR structure that, although optimised, leads to frequent requests for cells data stored in remote memories. For the latter, long-range forces are involved, hence remote information are necessary to perform any calculation. The number of remote accesses is not predictable and changes along time and among different processors. The result can be detected as the MPI overhead shown above. This is confirmed by the fact that when few processors are used, and the domain decomposition is much coarser, MPI impact is less relevant, becoming more and more important as the number of AMR (hence its complexity) increases (see Figure 7).

In Figure 10, also the computational efficiency is presented. For this specific set up, the efficiency is good for the 512 case, still reasonable for the 1024 case, but the tendency is to drop to unacceptable values above 1024 processors.

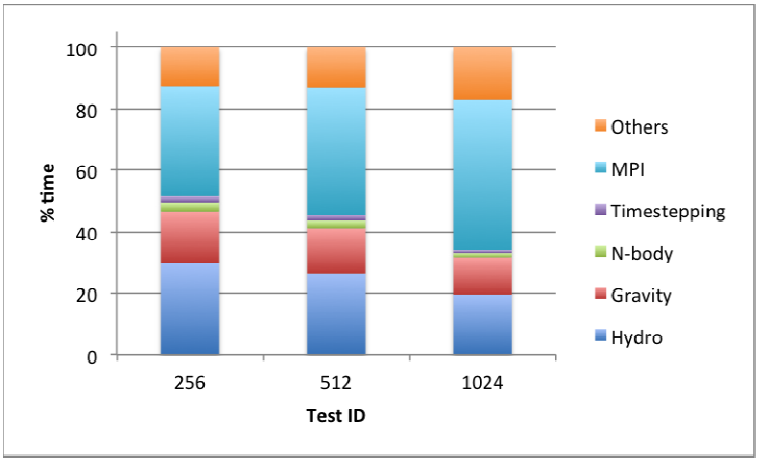


Figure 9: Distribution of the work for the production test as a function of the number of processors.

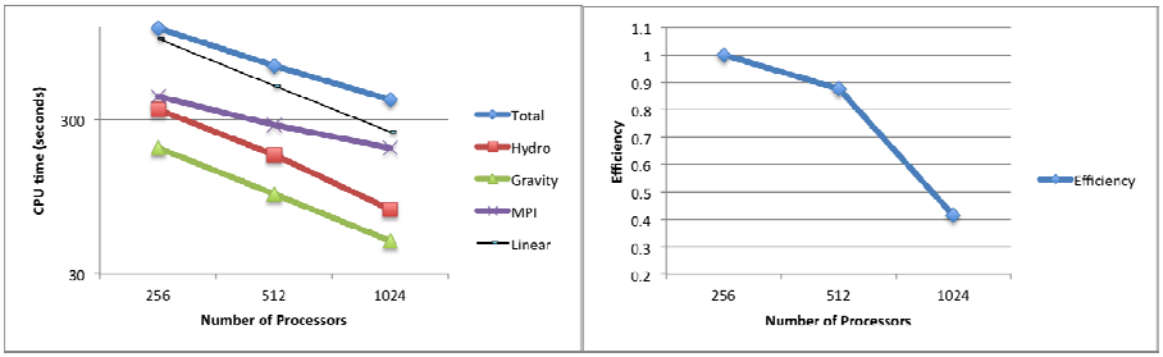


Figure 10: Scalability in the production test (left) for the whole code, the principal sections and MPI (linear scalability is shown for comparison – black line). Efficiency of the code as a function of the number of processors is shown in the right image (efficiency=1 for the 256 case).

3.2 PKDGRAV

3.2.1 Description of the code

PKDGRAV [9] is a Tree-N-Body code, designed to accurately describe the behaviour of the Dark Matter in a cosmological framework. The central data structure in PKDGRAV is a tree structure, which forms the hierarchical representation of the mass distribution. Unlike the more traditional oct-tree, which is used in the Barnes-Hut algorithm [10], PKDGRAV uses a k-D tree, which is a binary tree. The root-cell of this tree represents the entire simulation volume. Other cells represent rectangular sub-volumes that contain the mass, centre-of-mass, and moments up to hexadecapole order of their enclosed regions. PKDGRAV calculates the gravitational accelerations using the well-known tree-walking procedure of the Barnes-Hut algorithm. Periodic boundary conditions are implemented via the Ewald summation technique [11]. PKDGRAV uses adaptive time stepping. It runs efficiently on very large parallel computers and has produced some of the world's highest resolution simulations of cosmic structures. Although designed for cosmological and galaxy simulations, PKDGRAV has been successfully used also on much smaller spatial scales, for studies on the evolution of planetary systems.

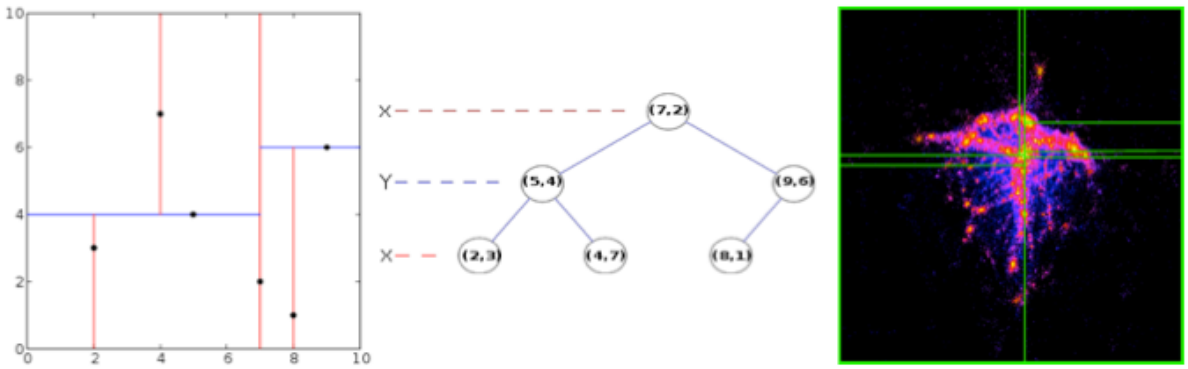


Figure 11: The tree structure of PKDGRAV.

3.2.2 Performance Analysis

All the tests presented in this section were run on a 1844 nodes CRAY XT5 [8] system. Each of the compute nodes consists of two 6-core AMD Opteron 2.4 GHz Istanbul processors giving 12 cores in total per node with 16 GBytes of memory.

For each test we present the total time required by the code to complete the test and the fraction of the work spent in the different parts of the code ("sections"), grouped as follows:

- *Gravity*: gravitational forces are calculated for each particle by direct point-to-point sum, for particles that are close to each other (where high accuracy is needed) and via Fast Multipole Method (5^{th}-order expansion of the potential) for long-range interactions.
- *Tree build*: the K-d tree is built according to the distribution of particles, in order to speed-up the calculation of gravitational forces.
- *Kick*: equation of motion are integrated with multiple time-steps, depending on the dynamical status in which the particle is (e.g. high density regions are more "active", hence the time-step for particles in those region are shorter than those for particle in "empty" volumes)

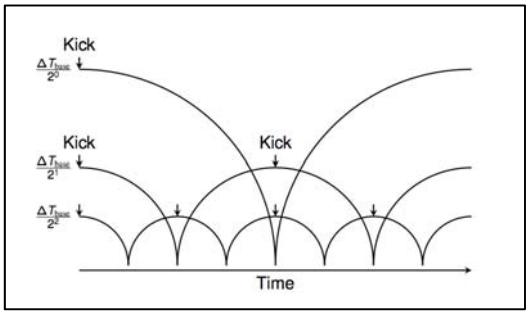


Figure 12: Multiple time step integration scheme.

Critical parameters analysis

The critical parameters for the PKDGRAV code are the number of particles used for the simulation (Np) and the number of particles active at a given sub-timestep level (Na).

- *Gravity*: the computing time needed for the calculation of the gravitational forces, scales linearly with the active particles O(Na) if Na >~ 100000; otherwise it scales as O(Na log(Np)). Communication is estimated to scale linearly with the number of particles ~O(Np).
- *Tree Build*: the construction of the K-d tree can be split in two main phases, a sort function, which scales as O(Np log(Np)) and the calculation of forces and moments, that scales as O(Np). Communication scales linearly with Np.
- *Boundary* Conditions: no communication involved in this step. The computation scales as O(Na)
- *Kick*: particle positions update requires no communication and scales as O(Na)

Single Timesteps test

In a first series of tests, we have analysed PKDGRAV performances when adaptive time step is switched off. This allows to focus on the force calculation part, emphasizing the performances of the *Boundary*, *Gravity* and *Tree Build* sections.

The test set-up consists in about 1 billion particles (1024^3), whose dynamics is followed from the homogeneous initial conditions, to an evolved, clumpy, configuration, after about 50 integration steps.

Figure 13 shows the distribution of work between different sections of the code as a function of the number of processors. PKDGRAV spends approximately the same fraction of time for calculating gravity, exploring the tree and imposing periodic boundary conditions. This last function is the most demanding for 512and 1024 processors. Communication overhead, represented by the MPI section, is less or around the 10% in all cases but the 4096 processors one, for which it grows to the 18%. This is due to the size of the problem that starts to be too "small" when 4096 processors are used.

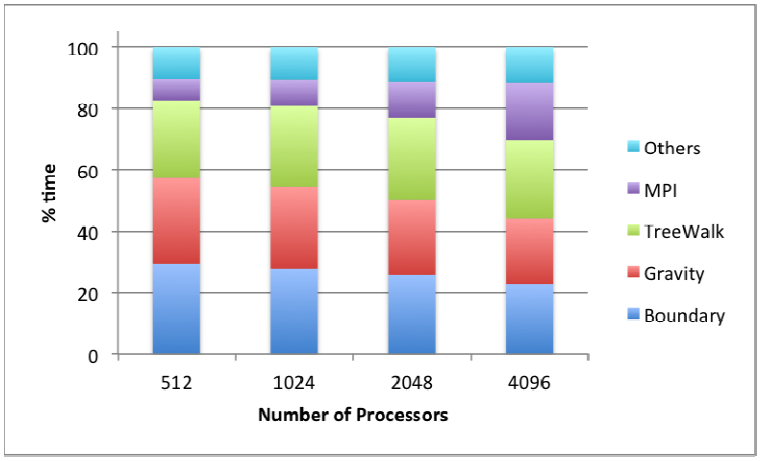


Figure 13: Distribution of work for the different PKDGRAV section in the Single Timestep test described above.

In Figure 14, we present the strong scalability of the whole code and of its main sections as a function of the number of processors. For comparison, the linear scalability is shown as well. A slight deviation from the linear scalability is measured, due mainly to the highest communication overhead, stressed by the MPI overhead trend, the other sections scaling linearly (or with NlogN behaviour) as expected by the analytic performance modelling.

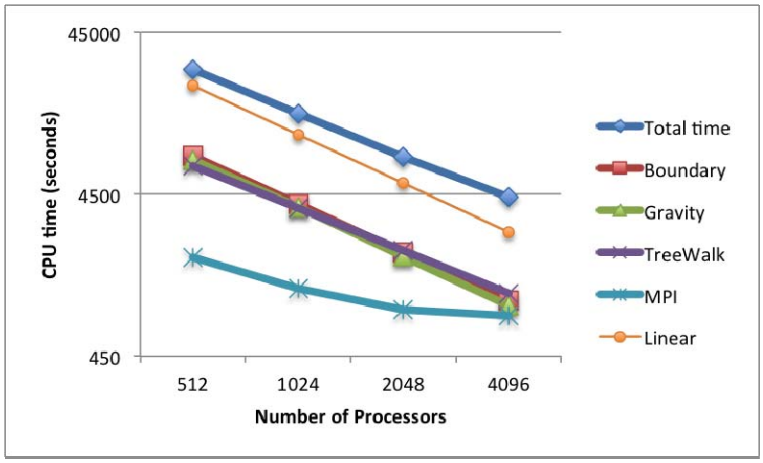


Figure 14: Scalability of PKDGRAV in in the Single Timestep test described above..

The efficiency curve, calculated with respect to the 512 processors case, confirms the good scalability of the code in this configuration. Efficiency is above 0.75 even when 4096 processors are used.

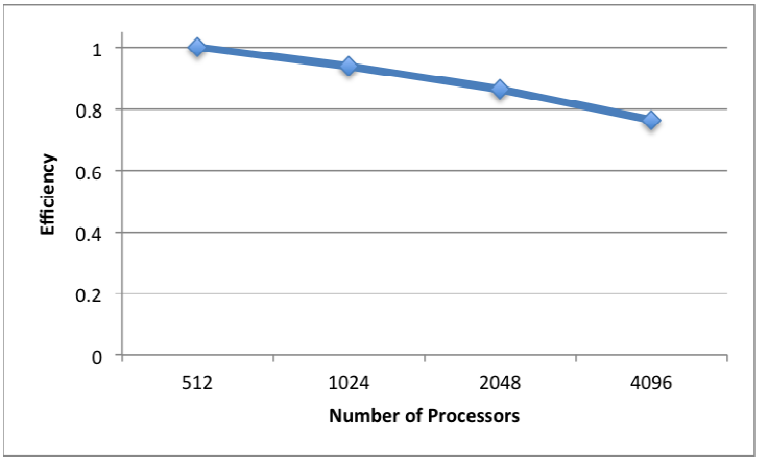


Figure 15: Efficiency of PKDGRAV for the Single Timestep test (efficiency=1 is set for the 512 processors test)

Dynamic Timesteps test

In this testcase, the PKDGRAV adaptive timestep is switched on. This allows coping with the large dynamical range typical of cosmological simulations. In this situation, each particle, depending on its dynamical state, evolves on different timescales. This means that long timesteps can be adopted for particles lying in "quite" (i.e. underdense) regions, using a small timestep only for those particles that really need it for numerical accuracy, strongly reducing the computational effort, that is mainly due to small timesteps particles. Figure 16 shows for a test with 9 timestep levels ("Rungs") the distribution of particles adopting different timesteps. At the beginning of the simulation, most of the particles use the same timestep, the particle distribution being homogeneous. The situation changes as soon as gravitational collapse starts and particles tend to concentrate in small clumps, where timestep can be extremely short. Note that less than 5% of the particles end up in these clumps, but they account for almost half of the total work. In a sequential code, this is a good achievement. In fact, it means that the code saves a lot of time that would be needed to integrate particles that can adopt large timesteps, but that with the single timestep approach would be integrated at the maximum time resolution (i.e. with a tiny timestep). However, this creates a relevant load balancing problem in parallel, expecially when a large number of processors is used. High (time) resolution particles, in fact, are distributed randomly among processors, and a few processors could be in charge of their integration.

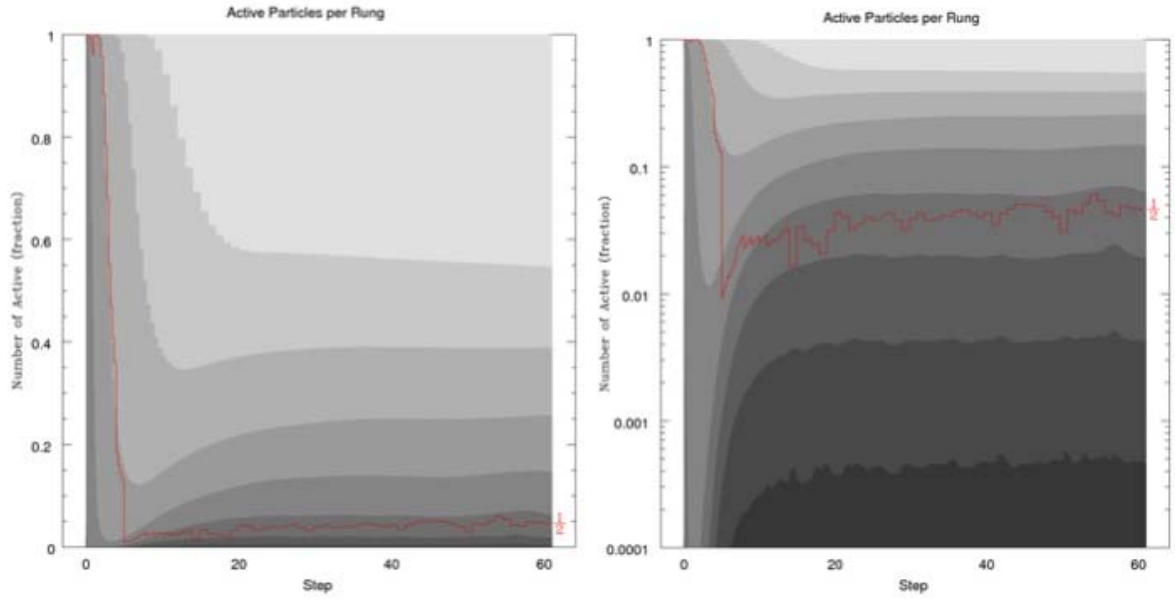


Figure 16: Fraction of active particles with different timesteps, linear (left) and logarithmic (right) scales as a function of evolutionary time. Darker zones are characterized by shorter dynamic timesteps. Red line shows where half of the CPU work is spent.

This leads to a computational overload of these few processors, leaving the others almost free, hence, to an overall drop of the performance.

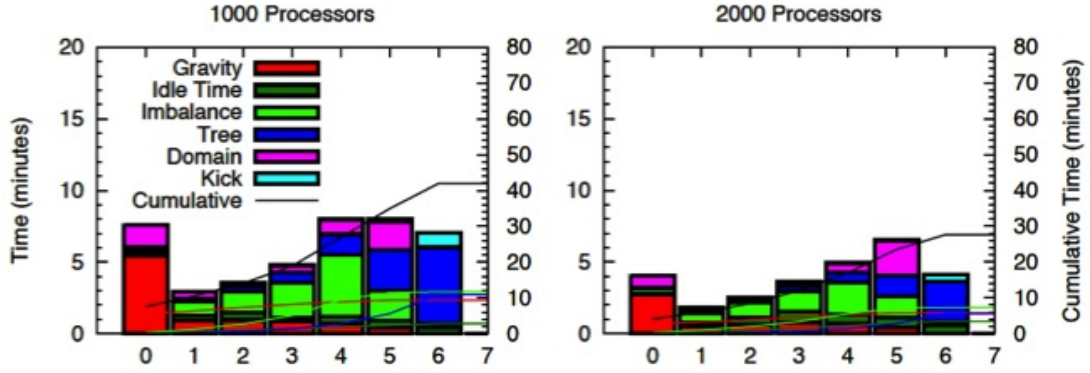


Figure 17: Distribution of absolute time spent in different parts of the codead different timestep levels in runs with 1000 (left) and 2000 (right) processors. The solid lines show the time for the various sections integrated on the various time levels.

The behaviour of the code on a use case characterised by about 1 billion particles is shown in Figure 17. In this test eight timestep levels are active. The absolute time spent in different parts of the code is shown for each level. As expected, most of the time is spent at levels 4, 5 and 6, that performs 16, 32, and 64 timesteps respectively for each level 0 “large” step. The unbalance problem is extremely clear and dominates the computing time at various levels. It is not relevant at level 0, where a homogeneous distribution of the weakly interacting particles is intrinsic to the physical problem. For level 0 it is the force calculation the most demanding part of the algorithm. The unbalance is also low or even negligible at the highest levels. For those levels in fact, few particles are present (although each particle accounts for 128 and 256 steps with respect to level 0) and most of the overhead is due to the access to the distributed tree.

An overall parallel efficiency of the 25% is measured in this (representative) test, due mainly to the load unbalance and to some MPI communication overhead (Idle time). These aspect will be the subject of specific care in the further steps of WP8.

3.3 PFARM

3.3.1 Description of the code

PFARM is part of a suite of programs based on the 'R-matrix' ab initio approach to variational solution of the many-electron Schrödinger equation for electron-atom and electron-ion scattering [12]. Relativistic extensions have been developed and have enabled much accurate scattering data to be produced. The package has been used to calculate data for electron collisions with various ions of Fe, Ni, Sn and neutral O. It is also being used for studies of intermediate energy scattering by light atoms.

PFARM divides configuration space into radial sectors and solves for the Green's function within each sector using a basis expansion: the BBM method [13]. The parallel calculation takes place in two distinct stages, with a dedicated MPI-based program for each stage. Firstly, parallel sector Hamiltonian Diagonalisations are performed using a domain decomposition approach with the ScaLAPACK-based code EXDIG. The energy-dependent propagation (EXAS stage) across the sectors is then performed using systolic pipelines with different processors handling different sectors [14]. This partitioning allows us to optimise sector length for each region: generally smaller numbers of sectors with a larger number of basis functions in the fine region and larger numbers of sectors with a smaller number of basis functions in the coarse region. In most cases the vast majority of energy points lie within the fine region and therefore this is the stage where most compute time is spent.

3.3.2 Performance Analysis

EXDIG Stage

The first stage of a calculation involves the parallel diagonalisation of large sub-region (or sector) Hamiltonian matrices that are computed independently of scattering energies. Evidently this approach is highly beneficial for calculations involving hundreds or thousands of scattering energies. The code is structured to take advantage of the parallel symmetric diagonalisation methods available in ScaLAPACK [55]. These routines are built upon the BLACS [56] communication library (itself based on MPI) and the highly optimised vendor BLAS [57] libraries, thereby ensuring highly efficient performance. Figure 18 shows the parallel performance of EXDIG using the ScaLAPACK diagonaliser PDSYEVD for a range of data matrix sizes out to 8192 cores of the Cray XT4.

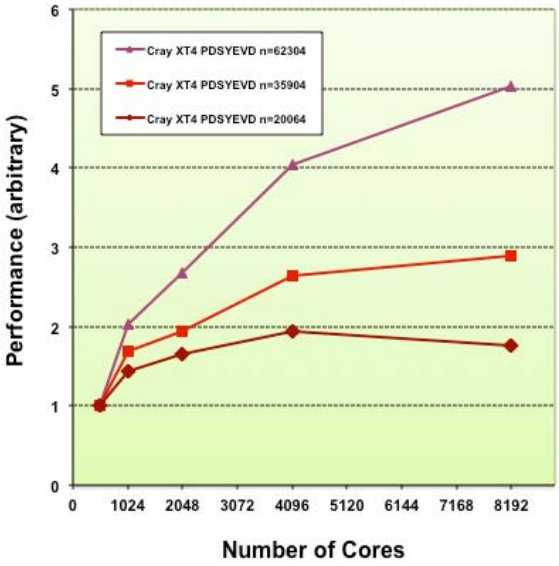


Figure 18: Parallel Performance of Diagonalisation Stage EXDIG on the Cray XT4

A characteristic of the EXDIG code is that *n* independent Hamiltonian matrix diagonalisations are required, where *n* is the number of sub-regions defined in the calculation. A recent optimisation [15] applied to the EXDIG code involves introducing a further level of parallelisation by splitting the global number of parallel tasks into *n* sub-groups of tasks, each with a unique BLACS communicator (or BLACS sub-grid). Each sub-group is then allocated a sector Hamiltonian matrix and undertakes the parallel matrix diagonalisations concurrently within these sub-groups. This can improve performance of EXDIG markedly, as shown by the timings obtained on the Cray XT4 and depicted in Figure 19.

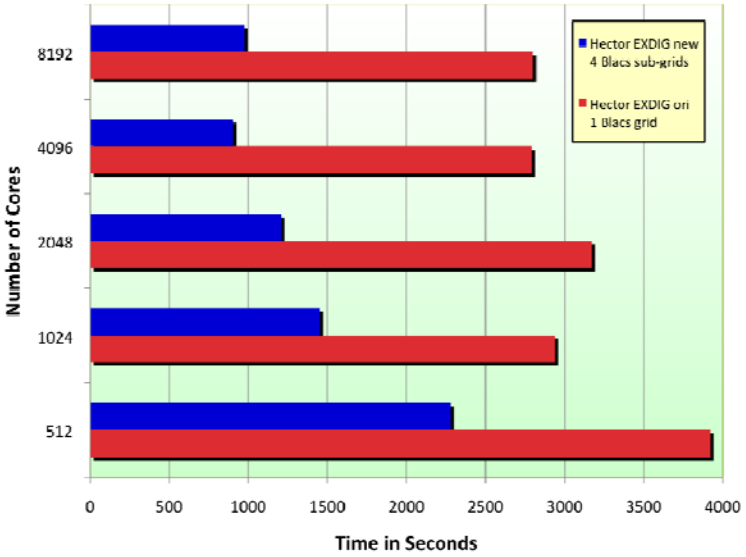


Figure 19: Parallel Performance of optimised EXDIG (new) using BLACS sub-groups compared to the original EXDIG (ori). FeIII, JJ coupling calculations.

EXAS Stage

For large-scale calculations involving several thousands of scattering energies, the parallel run time is generally dominated by the R-matrix propagation stage calculations undertaken in the EXAS code. In this stage of the calculation the majority of the processors available are arranged in arrays of processor pipelines, where each 'node' of the pipeline represents one sector. These pipelines are supplied with initial R-matrices (one for each scattering energy) from the inner region boundary by an R-matrix production group of processors (domain decomposition calculation). The final R-matrices produced by the propagation pipelines are passed on to a final group of processors for a task-farmed asymptotic region calculation, before results such as collision strength results are written to disk. The significant advantage of this 'hybrid' decomposition of tasks in EXAS is that much of the initial R-matrix and sector R-matrix propagation calculation on each node of the pipeline can be based upon highly optimised level 3 BLAS routines, leading to highly efficient usage of the underlying HPC architecture. A further reduction in compute time can usually be obtained by undertaking EXAS runs in two stages. Firstly a fine region propagation involving scattering energies residing in the extremely complex scattering resonance region followed by a coarse region propagation for scattering energies above this region.

The parallel code also scales very well up to large number of cores as more processor pipelines are added (Figure 20). However on very large core counts e.g. 16384, overheads such as IO start to become significant and at this level of parallelism the strong scaling wanes. A more detailed discussion of the parallel performance of PFARM can be found in the report from the Distributed CSE Support Project Report [15].

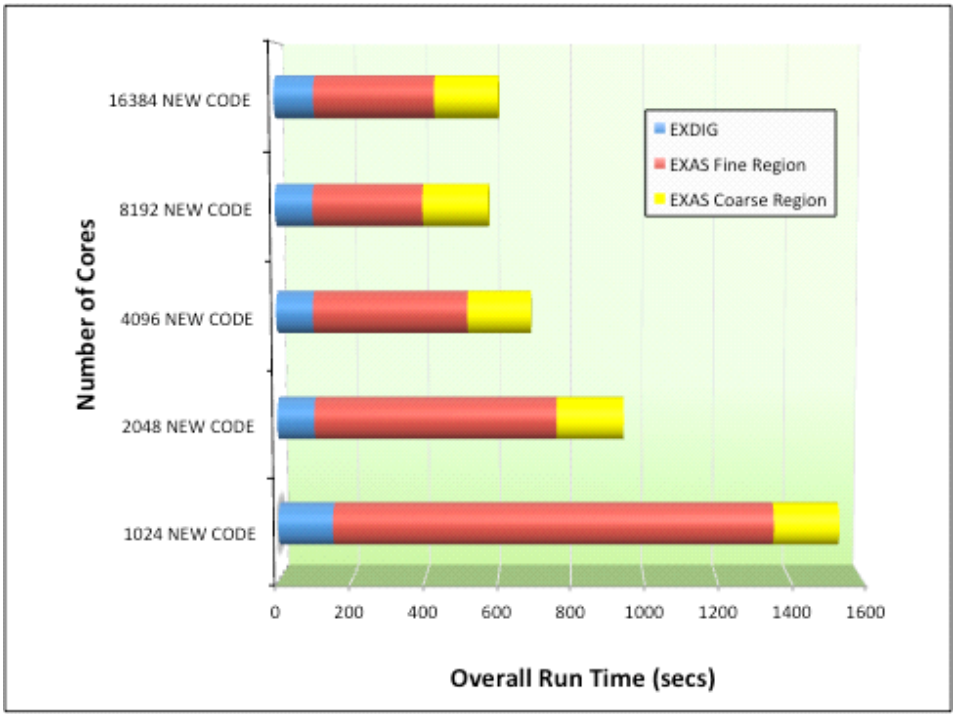


Figure 20: Parallel Performance of EXAS R-matrix propagation code. The graph reports the strong scaling behaviour on the Cray XE6 for a FeIII calculation with JJ coupling involving 10678 scattering energies.

Serial Performance

As shown in Figure 20, the bulk of the computational time in large-scale parallel runs is spent in the sub-region propagation calculations. These calculations make much use of Level 3 serial BLAS matrix-multiply routines. The highly optimised BLAS library routines are typically designed to take advantage of the underlying microprocessor architecture, and therefore attain near peak performance. The figure below (Figure 21), taken from a single-node efficiency study in the UK on an IBM Power5+ series architecture [16], shows that PFARM (here referenced as PRMAT) is one of the fastest academic application codes in common usage in the UK on HPC architectures, averaging >35% of peak performance.

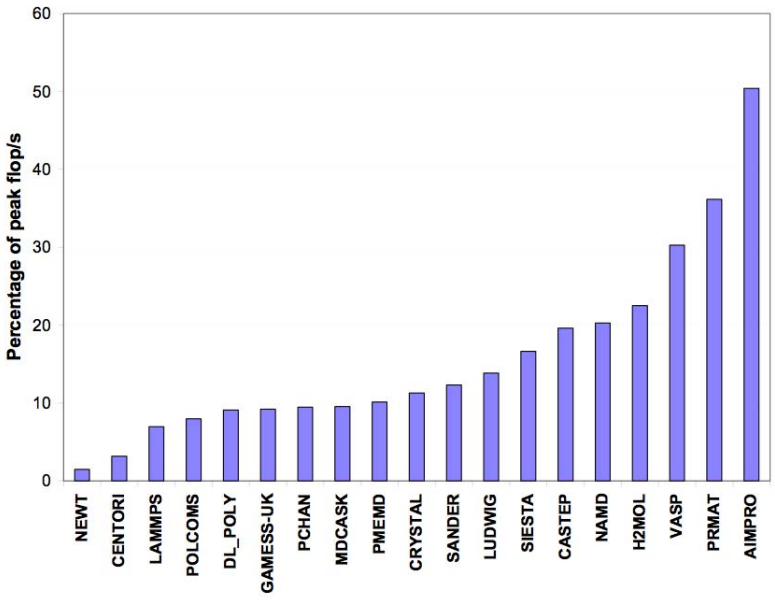


Figure 21: Performance of different codes

4. Performance Analysis of Community Codes: Climate

In deliverable D8.1.1, four areas have been identified as being central to the improvement of performance in climate modelling. These are:

1) Couplers between different earth system components,
2) Input/Output – reading and writing data in parallel from numerous threads,
3) Dynamical cores – the solution of the equations of fluid motion,
4) Ocean models – solvers of the oceanic dynamics.

The selection of the individual software components and their performance analysis is given in the subsequent sections.

4.1 OASIS

4.1.1 Description of Code

In Europe, OASIS [58] is the most widely employed coupler to build a full climate model on the basis of individual realm components (ocean, atmosphere, sea-ice, etc.). It is used in six out of the seven Earth System Models (ESMs) involved in ENES [47] to exchange and interpolate the coupling information (the coupling "fields") between their individual components. The widely used OASIS-3 version of the coupler offers only a limited field-by-field "pseudo-parallelism", and will soon become a central bottleneck for high resolution ESMs running on massively parallel platforms.

4.1.2 Performance Analysis

In spite of the impending bottleneck, OASIS-3 is still being used successfully, for example, in the EC-Earth model, whose underlying atmospheric model, IFS, was increased to ~25km, or 843,000 points, and 62 levels, along with an 0.25-degree (1.5M points), 75 depth-level configuration of the NEMO ocean model [35]. This was run on the Ekman cluster at the PDC Centre for High-Performance Computing (1268 nodes of dual-socket quad-core AMD Opteron processors, i.e., a total of 10144 cores) with different numbers of cores for each component and OASIS-3. Different combinations were tested, and **Table 2** illustrates the benefit of the OASIS-3 pseudo-parallelisation.

IFS-NEMO-OASIS number of cores	512-128-1	512-128-10	800-256-1	800-256-10
1-IFS standalone	41.	41.	29.9	29.9
2-EC-Earth3	45.7	42.3	33.2	30.3
2.1-IFS component	41.8	n/a	32.7	n/a
2.2-NEMO component	38.5	n/a	24,6	n/a
2.3-OASIS	5.5	n/a	6	n/a
Coupling overhead (2-1)	4.7 (13.4%)	1.3 (3%)	3.3 (11%)	0.4 (1.3%)

Table 2: 2-hour simulation response time (in seconds) for the different components and for EC-Earth3 coupled model. The configuration (top row) indicates the number of cores used for IFS, NEMO and OASIS respectively. The coupling overhead is calculated as the difference between EC-Earth and IFS standalone elapse time. IFS and NEMO run in parallel, not sequentially.

The OASIS elapsed time is non-negligible when it runs in single-core mode (respectively 5.5 seconds and 6 seconds for the 512-128-1 and the 800-256-1 configurations). In this case, the

coupling induces significant overhead in elapsed time with respect to the IFS standalone run (respectively 13.4% and 11%); this is true even if OASIS-3 interpolates the fields when the fastest component waits for the slowest, as the cost of OASIS-3 is larger than the component imbalance. When the parallelism of OASIS-3 increases (from 1 to 10 cores), the OASIS-3 elapsed time decreases and can almost be "hidden" in the component imbalance. Even if we do not have direct measures of the OASIS elapse time in these cases, this can deduced by the EC-Earth3 elapsed time, which decreases from 45.7 to 42.3 seconds (512-128-1 -> 512-128-10 configurations) and from 33.2 to 30.3 seconds (800-256-1 -> 800-256-10 configurations). Therefore, it can be concluded that OASIS-3 pseudo-parallelisation can be an efficient way to reduce the coupling overhead (which goes from 13.4% to 3% in the 512-128 configuration and from 11% to 1.3% in the 800-256 configuration).

Of course, this way of overlapping the cost of OASIS-3 works only if there is some imbalance of the components elapsed times, which allows OASIS-3 to interpolate the fields when the fastest component waits for the slowest. If the components were perfectly load balanced, then the OASIS-3 cost, even if lower when OASIS-3 is used in the pseudo-parallel mode, would be directly added into the coupled model elapsed time.

So, it is expected that for more than O(1000) cores – already close to the low limit of cores required to run on PRACE Tier-0 platforms, or for other configurations, the limited parallelism of OASIS-3 will soon become a bottleneck in coupled simulations. For example, the ARPEGE-NEMO climate model jointly developed by Météo-France and CERFACS, has been compiled and run on more than 1000 cores on the PRACE Tier-0 "CURIE" Bullx supercomputer, at relatively high resolution (50km-atmosphere, 1⁄4 degree-ocean), to study regional scale / large scale interactions. The ocean model is used in a 1D mode, as a mixed layer model (this configuration is called NEMIX), to simplify and better understand coupled processes. On 1024 cores (500 for the atmosphere, 512 for the ocean, 12 for the coupler), it was observed that OASIS-3 takes up to 20% of total elapsed time to perform interpolations and communications between coupled components.

Two approaches are currently followed to introduce true parallelism into OASIS. The first is the modification of OASIS-3 to use the Model Coupling Toolkit (MCT [25]), which also couples components in the American National Center for Atmospheric Research (NCAR) Community Earth System Model (CESM). MCT requires that the regridding weights are pre-computed offline, but then implements fully parallel regridding and exchanges of the coupling fields.

MCT has proven parallel performance. For example, a key operation is the interpolation between fields defined on atmospheric and on oceanic grids. This is essentially a sparse, rectangular matrix-vector multiplication, which scales well on parallel platforms if programmed correctly (see Figure 22). The production nature of MCT and its documented performance imply that an OASIS3-MCT implementation will be a wise investment.

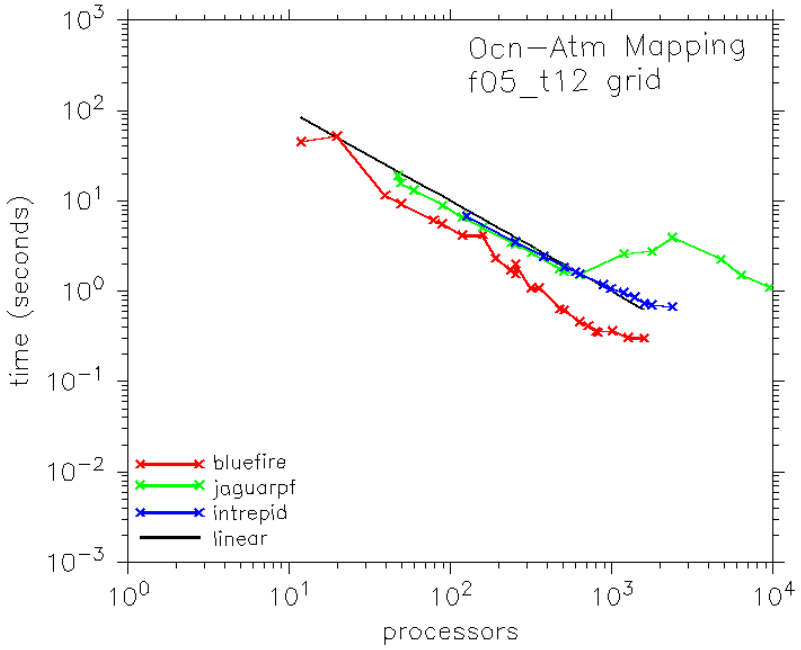


Figure 22: The MCT ocean-to-atmosphere benchmark performs an interpolation between a 0.47x0.63 degree oceanic grid and a 0.47x0.63 degree atmospheric grid. The operation scales well to large numbers of cores on an IBM PowerSeries ("bluefire") and IBM BlueGene/P ("intrepid"), though there are some scalability limitations on the Cray XT5 ("jaguarpf"). Credit: [25].

The second approach is to use the OASIS-4 coupler, which should be considered a separate product to OASIS-3, that is to say, OASIS-4 is not a direct evolution of OASIS-3. OASIS-4 is a product of a collaboration between the Nippon Electric Company (NEC) IT Research division, CERFACS, CNRS and DKRZ. The conceptual strength of OASIS-4 is its fully parallel online calculation of regridding weights (the so-called "neighbourhood search"). However, in an evaluation [18] of a beta version of OASIS-4, extensive bugs were found. These were difficult to fix because the code is large (180K lines of code) and because the original developers no longer work on the project. Moreover, key deficiencies in the design of the parallel neighbourhood search library were determined. The OASIS-4 design is based on structured grids, and completely new routines would have to be developed for unstructured grids.

A short-term solution is to retrofit OASIS-4 for pre-computed regridding weights, thereby circumventing most of the bugs, and the limitations on structured grids. This solution is conceptually equivalent to the OASIS3-MCT solution. Our current plan is to implement both OASIS3-MCT and OASIS-4 with pre-computed weights, and to perform a comparison.

4.2 Input/Output

Many scientific challenges in climate research, e.g. the direct simulation of small scale processes like clouds and precipitation, will require models with increasing spatial resolution as well as very high temporal resolution. The large volume of data produced by these simulations and the time needed to write (and subsequently access and process) these datasets will greatly impact the scientific productivity, and limit the effective exploitation of multi-Petascale and Exascale systems.

Today's output in many climate models is still serial: data are collected on a single CPU or core, which packs the data and writes them to disk. In a typical climate model run, about 2%

of time steps are output steps, one output step taking at least on order of magnitude longer than non-output steps. Thus, often 30% or more of the of a model's wall clock time is spent in (serial) I/O. This gets worse with model resolution and scaling of any given resolution to higher thread counts.

Trivial parallel I/O concepts proved to have severe limitations, e.g. parallel I/O using many separate files just postpones the problem to the post-processing stage. Parallel I/O using native direct access, e.g. MPI-IO, leads to problems with compressed data and is often not implemented efficiently.

Within the climate modelling community, several approaches exist to solve this problem through the development of specific I/O server solutions, which ideally provide at the same time interfaces to on-the-fly post-processing capabilities. However, a single, tested and generally applicable solution is not yet available.

We propose therefore, within PRACE2IP WP8, to compare several of these tools with respect to functionality and efficiency, and propose efficient generic solutions. The tools to be considered here are CDI developed at MPIM in Hamburg, XIOS developed at IPSL in Paris and PIO developed at NCAR in Boulder, USA. It is also proposed to compare these solutions with a standard library, e.g. parallel netCDF [59].

4.2.1 Description of Code: CDI

The Climate Data Interface (CDI) is an Interface to access Climate and NWP model Data. The interface is independent from a specific data format and has a C and Fortran API. CDI was developed for a fast and machine independent access to GRIB [60] and NetCDF [59] datasets with the same interface. Local MPI-MET data formats are also supported.

The production version of CDI runs serially on MPI process 0. File writing with this version is depicted in **Figure 23**.

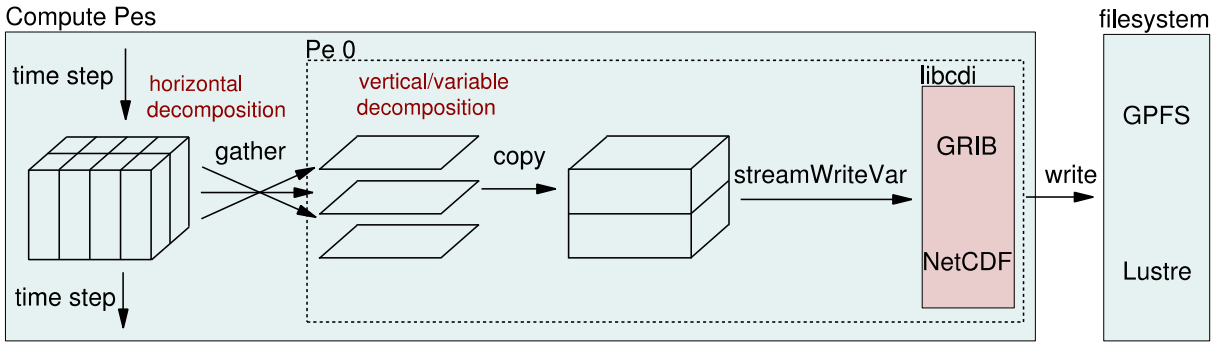


Figure 23: File writing procedure with serial CDI version, running on MPI process 0.

The CDI version currently under development will run on multiple I/O PEs. A file write operation proceeds along the following lines:

1. decompose I/O in a way that all variables are distributed over the collector/concentrator processes ("I/O PEs"),
2. store each compute PEs data in buffers to be collected by collector PEs (I/O PEs) - the buffer should reside in RDMA capable memory areas,
3. rather than doing I/O, copy data to buffer and continue simulation,
4. the collectors collect their respectable data (gather) via one-sided (RDMA based) MPI calls and do the transpose,
5. compress each individual record, and
6. write the file.

The parallel write procedure is illustrated in Figure 24. In this case there is a pool of processes to collect and to write the data, completely independent of the compute processes.

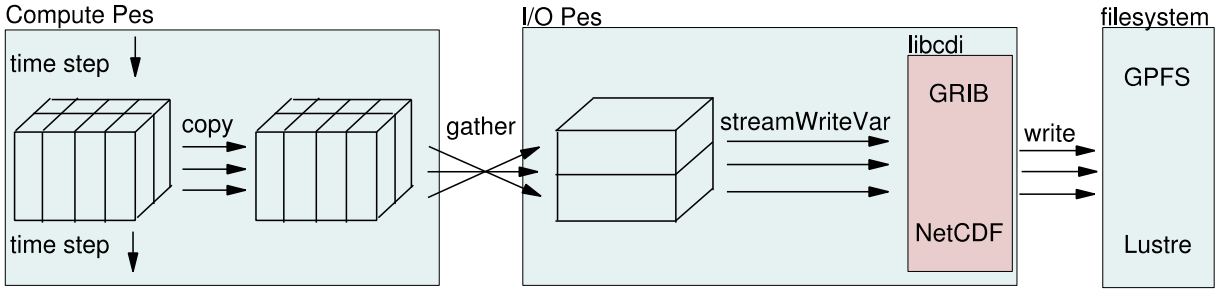


Figure 24 File write procedure for the new CDI version, running on a set of I/O processes, completely separate from the compute processes.

4.2.2 Performance Analysis: CDI

In the parallel CDI, a number of different algorithmic strategies have been tested for the write operation, which is known to be more critical in climate applications than the read operation.

1. Classic Serial: collect and write serially using *fwrite.* Advantages: no communications needed, simplicity, simple multi-file handling. Disadvantage: does not scale.
2. MPI Writer: collect and write in parallel using *MPI_File_write_all*. Advantages: No user-visible communication between I/O processes, straightforward implementation, performs best of all MPI file-writing routines. Disadvantages: collective call, does not work for inhomogeneous allocation of variables per process/file.
3. *MPI_File_iwrite_shared*: collect and write in parallel using *MPI_File_iwrite_shared.* Advantages: no user-visible communication between I/O processes. Disadvantage: very bad performance on GPFS
4. Offset sharing: collect and write in parallel, communicate file offsets using MPI RMA with passive target. Advantages: all collectors write, use of POSIX AIO possible. Disadvantages: complex locking is needed, performance is bad.
5. Offset guard: collect and write in parallel using one process to administrate file offsets. Advantages: performs best of testbed versions, use of POSIX AIO possible. Disadvantages: one process is idle most of the time, problems if I/O processs span over different nodes.
6. POSIX writer: collect in parallel and write serially using *fwrite.* Advantages: use of POSIX AIO possible, use of double buffering possible, one writer process for each file possible (this might gain performance, and the processes could be on different nodes). Disadvantages: the buffer has to be communicated, the speedup is limited if only one writer for all files is used.

These write strategies are depicted in Figure 25.

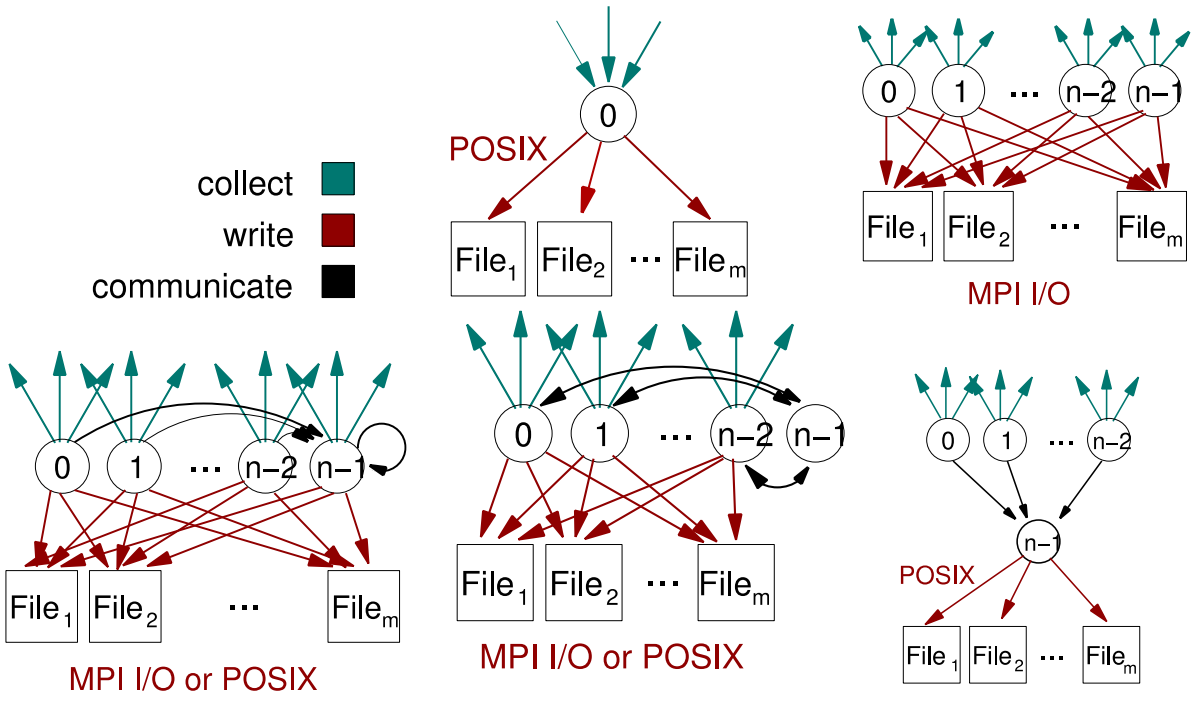


Figure 25: CDI testbed write strategies: Classic Serial (top center), MPI Writer, including MPI_File_iwrite_shared (top right), Offset Sharing (bottom left), Offset Guard (bottom center), POSIX Writer (bottom right).

Write Strategy	# PEs	# Collectors	# Writers	MB/s
Classic Serial	1	1	1	148
MPI_File_iwrite_shared	4	4	4	234
	8	8	8	281
	16	16	16	338
	32	32	32	322
Offset Guard	4	3	3	240
	8	7	7	498
	16	15	15	829
	32	31	31	946
POSIX Writer, fwrite	4	3	1	257
	8	7	1	450
	16	15	1	411
	32	31	1	482
POSIX Writer, aio_write	4	3	1	122
(5 output streams)	8	7	1	120

Table 3: Throughputs (in MB/s) for four of the six write strategies tested for parallel CDI.

The write throughputs of four of the six strategies are given in Table 3. “Offset Guard” seems to be the most promising strategy, however not all strategies have been fully optimised. All of the strategies will be implemented in the upcoming release, and these will all be tested in the I/O intercomparison.

4.2.3 Description of Code: XIOS

XIOS is the parallel I/O software, written in C++, developed at IPSL for climate codes. The software has two aims: i) a parallel output system, ii) a simple configuration of model outputs

through XML files. A draft version has been developed, mainly to test the main concepts. This version 0 has been put in production in the NEMO ocean model [35]. It has been used in several climate models for the CMIP5 climate model intercomparison. XIOS version 1 is now in beta state, and will be delivered to users in a few weeks.

The system can work in two modes: online mode or server mode. In either case, one can choose to output multiple files, with each process writing its domain on its own file, or output in a single global file using parallel file system access using the NetCDF4 library. In the case of multiple files, the global output file should be built after execution.

- **Connected mode**. The software is linked as a library in the model executable. Each process has simultaneous access to the file system so time access cannot be overlapped. With this mode it easier to manage the execution, but performance may suffer drastically from the file system time access.
- **Server mode**. The software is divided between server(s) and client(s). The client is linked to the model executable. The server runs as a separates executable, on its own processor. In each model parallel process, the client sends data to the output server. If the number of model processes is large, several servers can be used. The number of servers is adjusted to equilibrate the load balancing between the model and the servers. Communications between clients and servers are asynchronous: the clients buffer the data, and send them asynchronously while the model runs. The main advantage is to reduce the number of processes simultaneously accessing the file system, and time access is overlapped with computation. This is the mode used in production runs. In server mode, time means could either be done on the server side, producing a heavy load of communication, or on the client side, producing more CPU load on the model processors.

Performance has not been extensively evaluated as of this time.

4.2.2 Description of Code: PIO

PIO is a thin layer placed on top of existing, more general parallel I/O libraries. This layer clearly separates the concerns of earth science application, e.g., the desire for a simple mapping between the process-local and global depiction of a field, and those of the I/O library, which are generally associated with efficiency. The target application for PIO is the Community Earth System Model (CESM), however the API would support a wide spectrum of applications, e.g., also those employing non-rectangular grids. The back-end libraries supported are MPI-IO, NetCDF-3 and NetCDF-4 [37], and pNetCDF [38].

4.2.3 Performance Analysis: PIO

In [24]**,** an I/O benchmark "POPD", derived from the parallel ocean program (POP), was implemented with PIO with several different configurations (see **Table 4**). In the table, the test configurations for the POPD benchmark are defined in terms of different output formats (either NetCDF3 or binary), different backend libraries (NetCDF3 or pNetCDF), varying numbers of I/O tasks (denominator of 12 yields one I/O task per socket of the test machine, a Cray XT5), whether user-level collective buffering and/or flow control was employed. While NetCDF3 is inherently a sequential library, the "parallel" C-n configuration was achieved by the I/O tasks reading/writing in turn from/to the file. C-n can be expected to reduce memory usage by roughly a factor of #iotasks, but will, if anything, yield an I/O bandwidth less than the sequential approach. The D-b configuration with a binary format is meant for comparative purposes only, as climate models would consistently require their data in NetCDF, or a similar, self-describing metadata format.

Config. name	Format	Backend lib.	# I/O tasks	coll. buffering	flow control
A-n	NetCDF3	NetCDF3.6.2	1	yes	no
B-n	NetCDF3	NetCDF3.6.2	1	yes	yes
C-n	NetCDF3	NetCDF3.6.2	2(ncores/12)	yes	yes
D-n	NetCDF3	pNetCDF1.2.0	2(ncores/12)	yes	yes
D-b	binary	MPI-IO	2(ncores/12)	yes	yes
E-n	NetCDF3	pNetCDF1.2.0	ncores	no	yes

Table 4: Test configurations for the POPD benchmark defined in terms of different output formats

The performance results for memory usage and read/write bandwidth are given in **Figure 26**. The results can be explained by fairly simple analytical arguments: Memory usage for the sequential versions (A-n, B-n) should be roughly constant, since the fields must be collected on one process. For the parallel scenarios (C-n, D-n, D-b, E-n), memory usage should decrease monotonically to the point where buffering becomes an issue for large number of cores (e.g., in E-n). For A-n and B-n, the maximum achievable single node bandwidth can be expected. In C-n, the I/O readers/writers take turns in a round-robin fashion with some overhead, necessitating a bandwidth less than the sequential case. Configurations D-n and D-b compare the effect of different backend libraries, MPI-IO and pNetCDF. Since pNetCDF uses MPI-IO in turn as a backend, the D-n bandwidth is logically similar, though generally less, than D-b performance. Note: the D-b configuration is not usable in production, since climate data are always required in a metadata format. Using all *ncores*, the E-n bandwidth can be expected to decay for large numbers of cores.

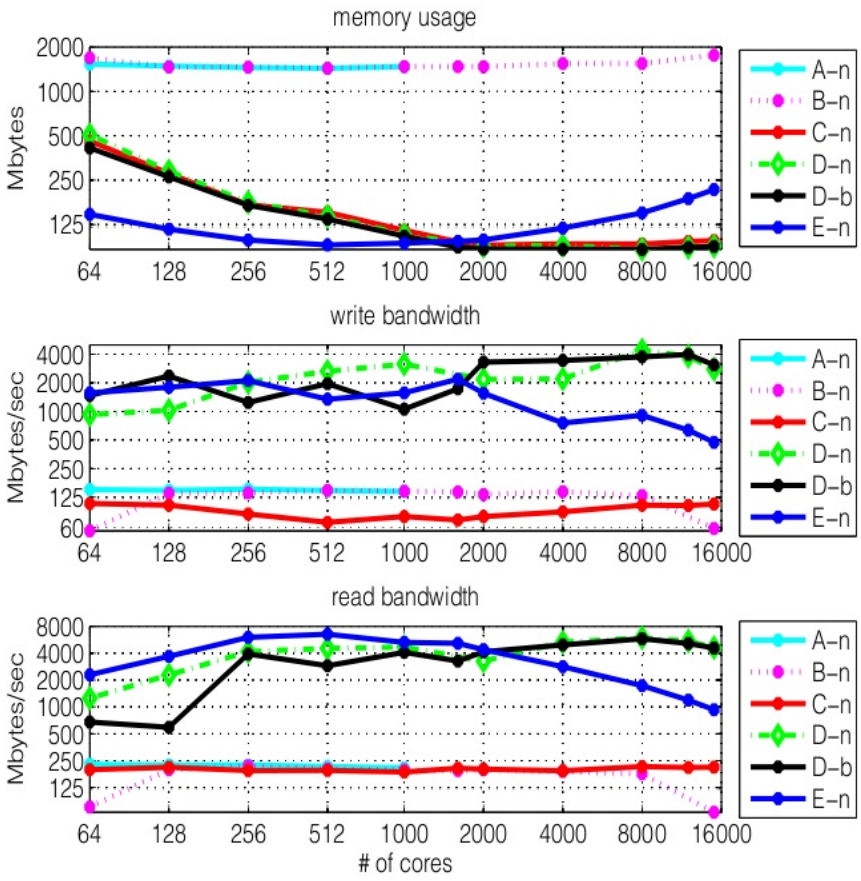


Figure 26: PIO performance results from [24] for the collective reading and writing of fields, which are distributed over a given number of cores.

4.3 Dynamical Cores

4.3.1 Description of Codes

Within this project there are potentially two ways to assist the climate community in the field of atmospheric dynamics solvers (known as dynamical cores). First would be to facilitate the port of existing production dynamical cores to emerging architectures, as was done, for example, in the HP2C COSMO project [3]. For climate models, the candidates for such an effort would by the cores in the IFS model (used in the EC-EARTH [20] climate model), or the ECHAM [22] and HadCM3 [21] models. The problem here is the limitations on the scalability of the underlying spectral atmospheric dynamics solver. Extensive effort has already been put into scaling these cores on previous generations of supercomputers, and it is generally agreed that the prospects to make a meaningful contribution on emerging platforms are limited.

The second possible contribution is to participate in the development of new and emerging dynamical cores for climate models, which are specifically targeting massively parallel platforms from the outset. Most of these global dynamical cores do not employ latitude-longitude grids, as such grids cause numerical instabilities, and limit inherent parallelism, due to converging meridians at the poles. A G8-funded ICOMEX project [25] concentrates on four of these cores: MPAS [30], NICAM [31], ICON [19], and DYNAMICO [32], where the latter two are European developments. ICON is close to official release and has already been evaluated in an inter-dycore comparison [34]. ICON employs a grid of spherical triangles which is derived from an icosahedral grid, and which can be (statically) refined in regions of interest (see Figure 27).

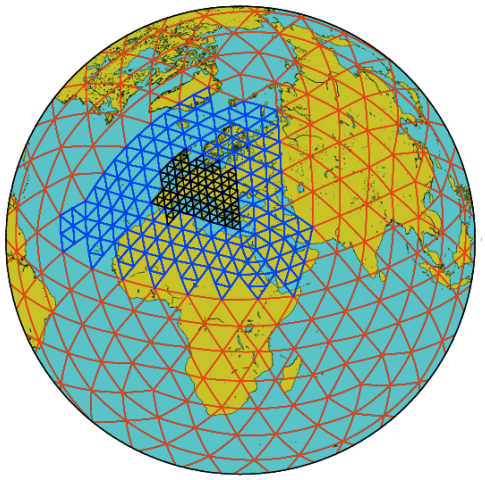


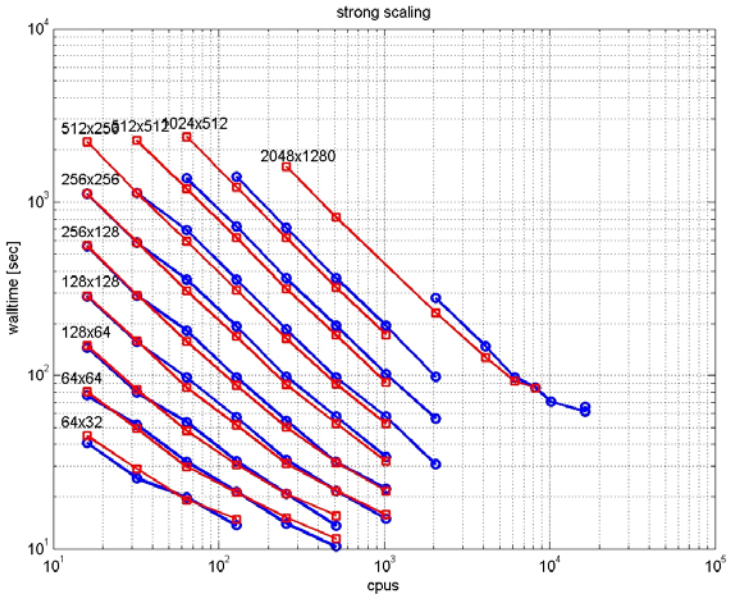
Figure 27: The ICON grid consists of spherical triangles at a base resolution (red), which have been derived by recursively bisecting the edges of an icosahedron. In areas of particular interest, some triangles can be further refined (blue) by subdivided triangles into four. This procedure can be repeated recursively (e.g., black triangles).

The UK MetOffice, STFC and other UK academic institutions, funded through NERC, are working on the Next Generation Weather and Climate Prediction (NGWCP) programme [39], also known as "GUNG-HO!" The aim of this project is to develop a new dynamical core for an operational forecast model to eventually replace the current Unified Model in about ten years time and also be suitable for climate research scenarios. A major objective is to replace the current latitude-longitude grid with a quasi-uniform grid to avoid problems due to convergence of longitude lines at the poles and enable better scaling on massively parallel systems. A wide range of grids and associated implicit, semi-implicit and explicit solvers are currently being evaluated with a view to reducing the number of candidates in March 2012, allowing more detailed investigations and parallel implementations to proceed. At that stage PRACE 2IP-WP8 support could contribute to the implementation and evaluation of algorithms on current and near-future systems, e.g. by focusing on PRACE prototype systems.

There is currently a study supported by the COSMO Consortium to evaluate the efficacy of the EULAG [23] solver for geophysical flows in the regional COSMO CLM climate model.

4.3.2 Performance Analysis: EULAG, ICON

Of the above-mentioned European developments – DYNAMICO, GUNGHO, EULAG, and ICON – the first two are in the design phase. There is a clear opportunity for PRACE centres to offer their background on emerging architectures to assist in the co-design of these cores. EULAG has proven scalability (Section 28) but its rectangular grid is not applicable for global climate applications.



Section 28: All EULAG-HS strong-scaling benchmarks except the horizontal domain grid 2048x1280 were performed on a BlueGene/L at the National Center for Atmospheric Research. The vertical has 41 levels. The red curves result when the benchmark is run in coprocessor mode, the blue lines in virtual mode. The 2048x1280 domain size was run on a BlueGene/P at IBM/Watson, and indicates excellent scaling to about 7000 cores in either mode. Credit: Andrzej Wyszogrodzki, NCAR.

ICON is the only near-production European emerging dynamical core appropriate for global climate to which PRACE centres can actively contribute to the evolving code base. ICON has several dynamical cores: several variants of a hydrostatic solver and a non-hydrostatic solver. The latter is of particular importance, as the trend to high resolution (10 km. and less) requires this. The ICON development branch contains the MPI-enabled version of the entire ICON model, but it is not publicly available. However, preliminary benchmarking has been performed at DKRZ. The speedup results with respect to the 64 MPI-process version are depicted in **Figure 29**.

The tail-off in scalability in the MPI implementation implies that other parallelisation paradigms need to be investigated as well. An OpenMP-multithreaded version of the non-hydrostatic core is a component of the freely available ICON testbed [26], and was extensively benchmarked in the first months of WP8 project, with the midterm goal of porting the code to a GPU.

In order to evaluate ICON's single-node, shared memory performance, the Roofline Model [28] was employed. The roofline model consists in a log-log plot of the operational intensity of the kernel measured in flops/byte (horizontal axis) versus the maximum floating-point performance measured in GFlop/s (Figure 30). For a given computational component, the operational intensity is defined as the ratio of the number of floating-point instructions per byte of memory traffic. The plot is divided in two distinct regions: the area underneath a left sloping line defines all operational intensities for which the performance is bound by the memory bandwidth. The sloping line itself represents the maximum bandwidth achieved in the STREAM benchmark [61]. Secondly, the area underneath the right horizontal line denotes values of the operational intensity with performance limited by the computing capacity of the underlying hardware. The horizontal line denotes the maximum achievable performance of a computationally intensive code, such as the DGEMM (matrix-multiply) benchmark [57].

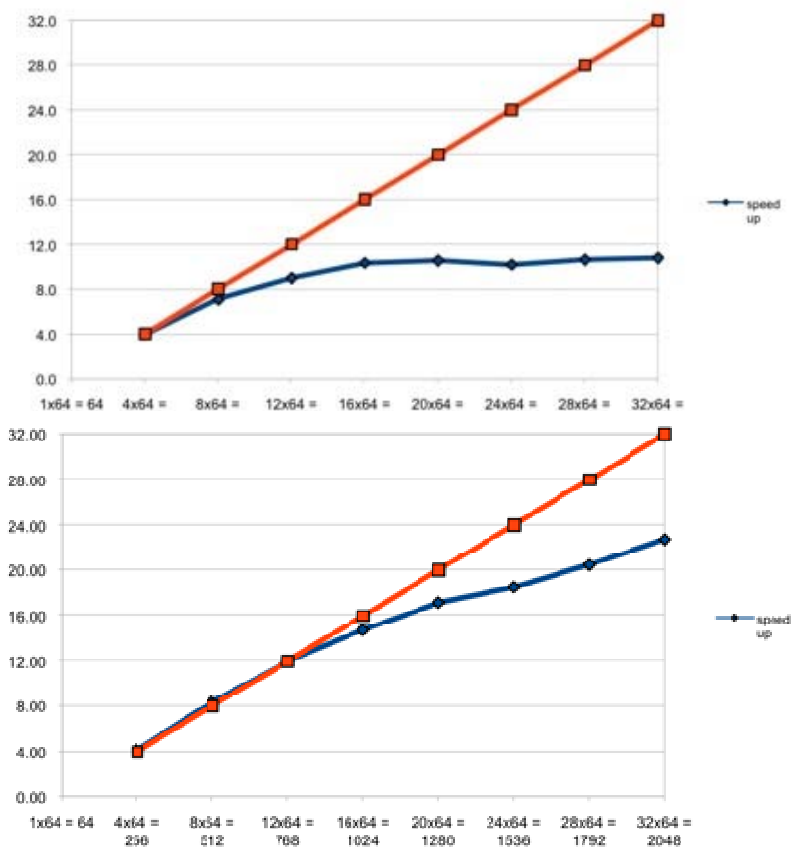


Figure 29: The speedup of the MPI-only version of ICON for the R2B04 resolution (roughly 139 km. - upper panel) and for the R2B05 resolution (roughly 69 km) – lower panel. with respect to the 64-process execution. The strong scaling plateaus at about 10 for this medium resolution test case. Credit: Hendryk Bockelmann, DKRZ.

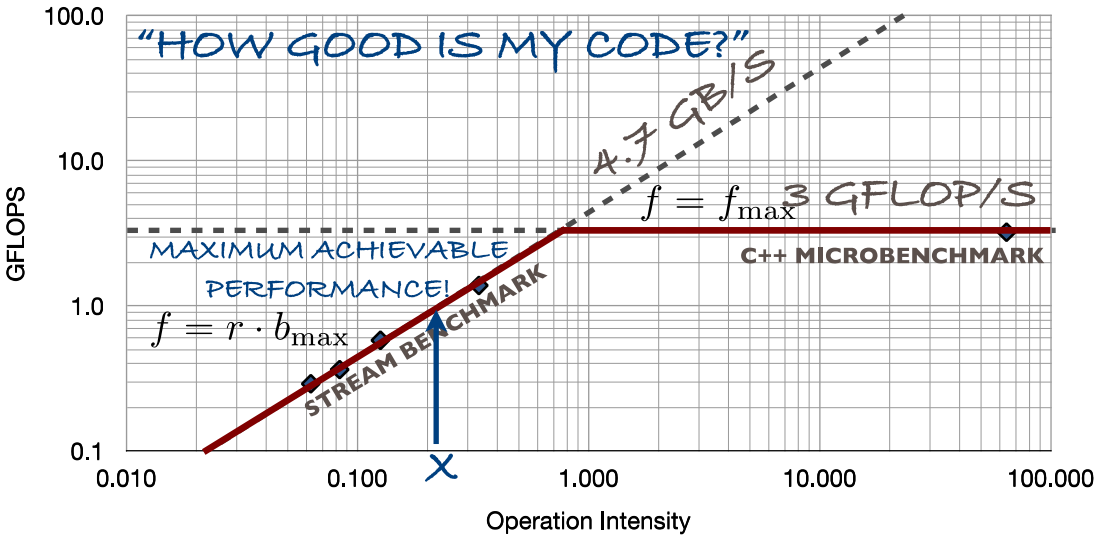


Figure 30: The roofline model [28], distinguishes between low and high computational intensities (floating-point operations per byte accessed). For low intensities, the overall performance is limited by memory bandwidth in a roughly linear relationship: the higher the intensity the more performance since the bandwidth is constant. At a certain intensity, memory speed becomes sufficient to fully occupy the floating-point unit, whose performance is now the limiting factor. The "X" indicates roughly the location of most finite difference or finite volume dynamical cores, such as the ICON non-hydrostatic solver.

In the preliminary WP8 investigations, several CPU and GPU architectures were evaluated and their roofline models derived (see Figure 31).

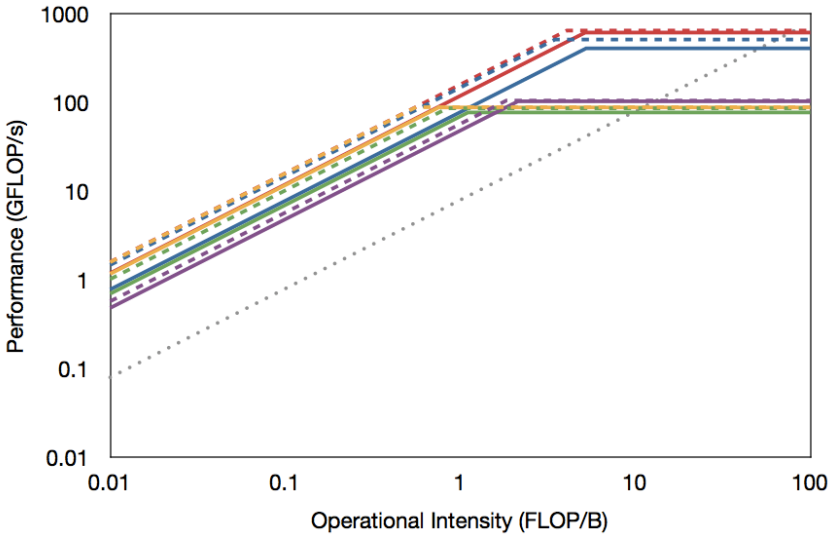


Figure 31: Double precision rooflines for AMD Magny-Cours (purple), NVIDIA Tesla M2050 (blue), NVIDIA Tesla T10 (green), NVIDIA GeForce GTX285 (yellow) and AMD Cayman (red). The theoretical rooflines are represented with dashed lines and the measured ones are shown with solid lines. The grey dotted line represents the theoretical PCI-e bandwidth. Credit: Christian Conti, ETHZ.

In a student project [29], the computational components of the non-hydrostatic solver were rewritten into roughly 60 OpenCL kernels, which are portable to both CPU and GPU. The kernel intensities (defined as the number of floating point operations per used Byte of memory) range from 0.1 to 1.0, entirely within the bandwidth-limited region of the roofline. The performances generally cluster around the STREAM benchmark limits, but are occasionally above them, implying some benefit from cache or thread-local memory ("shared memory" in CUDA terminology).

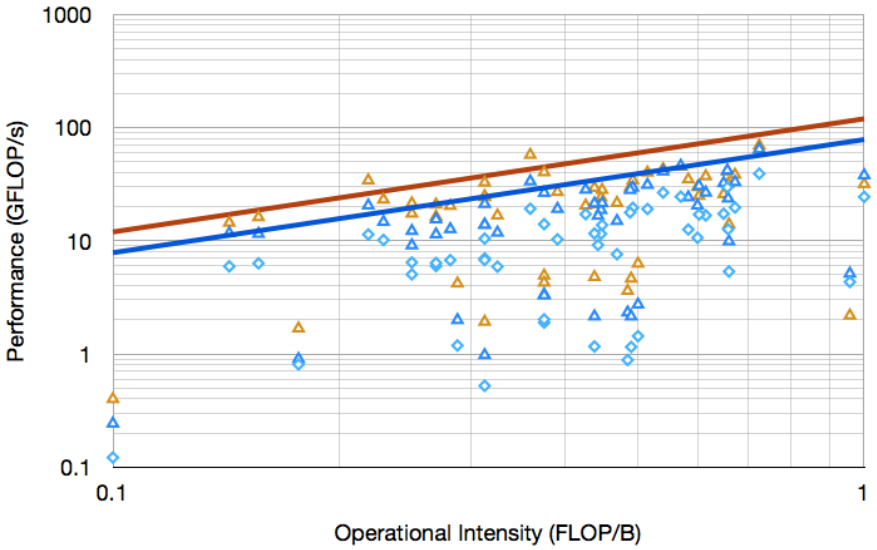


Figure 32: Operational intensities of the various kernels implemented with expected peak achievable performance (solid lines). Three different cases are depicted for each kernel: the R2B3 resolution (5120 grid triangles) on Tesla M2050 (blue diamonds) and R2B4 (20480 grid triangles) on a Tesla M2050 (blue triangles) and on a Cayman (orange triangles). The expected performance is based on the operational intensity only and does not consider the performance degradation caused by the size of the data structures on which the kernels operate. About ten kernels perform far worse than expected, due to poor utilisation of the data structures, and/or dependencies between loop iterations (such as for the vertical integration). Several kernels perform above the STREAM performance due to fortuitous cache effects. Most kernels cluster just below the maximum performance.

To test this the applicability of the STREAM benchmark to the ICON non-hydrostatic solver further, we ran the existing OpenMP testbed NH solver on a number of different multi-core architectures (**Table 5**).

Platform	Threads	R2B3 (5120 tri)		R2B4 (20480 tri)		STREAM	Achievable
		s.	GFlop/s	(s.)	GFlop/s	GB/s	GFlop/s
IBM BG/P	1	57.2	0.8	229.5	0.8	2.2	0.9
IBM BG/P	4	15.0	3.1	60.4	3.1	8.9	3.6
AMD Magny-Cours	1	46.6	1.0	195.4	1.0	8.1	3.24
AMD Magny-Cours	12	5.4	8.7	23.1	8.1	29.6	11.8
AMD Interlagos	1	33.5	1.4	135.0	1.4	12.7	5.1
AMD Interlagos	16	4.3	11.0	16.3	11.6	70.0	28.0
Intel Westmere	1	19.7	2.4	82.0	2.3	8.39	3.4
Intel Westmere	7	3.1	15.2	12.9	14.6	50.22	20.1

Table 5: The performance of the OpenMP multi-threaded version of the ICON non-hydrostatic solver is compared over a number of multi-core architectures. The memory throughput (GB/s) for the STREAM benchmark is also supplied. The "achievable GFlop/s" is defined as the STREAM throughput (GB/s) times the solver's average computational intensity of 0.4. Credit: CSCS.

Thus, it appears that the STREAM benchmark offers a fairly tight upper bound for ICON non-hydrostatic solver. Furthermore, we predict that the roofline model can be applied to most ICON components (e.g., the physical parameterisations), and, indeed, in other finite-difference or finite-volume-based climate codes. The open task is to then to concentrate development efforts on bottleneck components, which are more than an order of magnitude slower than the achievable STREAM performance. In the above-mentioned pilot project, the implicit vertical solver, which contains a small tridiagonal matrix inversion, is one such component. The isolation of such components will be a topic of subsequent work.

4.4 Ocean Models

4.4.1 Description of Code: NEMO

NEMO [35] is a widely used, highly portable numerical platform for simulating ocean dynamics, biochemistry and sea-ice, for both operational and research purposes. It is written in Fortran90 and parallelised using MPI with a regular domain decomposition in latitude/longitude. The governing equations are solved in finite-difference form upon a tri-polar 'ORCA' grid [45] to get rid of the north fold singularity. Thus, unlike atmospheric models, polar filtering isn't needed and parallelisation is well balanced. The development of new oceanic dynamical cores (Section 4.3) is then of a lower priority, even if NEMO could benefit at middle term of such improvements. Considering the large spectrum of options and applications (global, regional, with/without ice, with/without biogeochemistry, etc.), the focus should be more on improving the structural paradigm of parallelisation, rather than on specific routines. This is also confirmed by the flat profile of the code (Section 4.4.2).

Other parallelisation initiatives

Fine grain parallelism (loop-based) has already been tested in hybrid mode. Unfortunately it did not show real speed-up and became even inefficient on a large number of processors (O(1000)). Tests on other finite difference ocean models (POM, ROMS) or NAS parallel benchmarks strongly suggest the need to go with coarse grain OpenMP multitasking with

each MPI domain being sub-divided in tiles at a high level.

A prospective study on porting NEMO on GPGPUs (gNEMO) is underway, led by STFC Daresbury Laboratory. At that time, different routines have been tested on GPUs with various success in terms of speed-up. It is worth noting that, as pointed out by A. Porter, the analysis of the most significant NEMO routines has emphasised their low computational intensity and resulting dependency on memory bandwidth. Therefore the question is; how do we make more effective use of what memory bandwidth we have? Bandwidth requirements could be solved by GPUs but also by using different approaches to algorithms or different data structures.

Another perspective concerns data output. For realistic experiments, NEMO, like all ocean/atmospheric models, spends a large amount of time writing ouput. It has been interfaced with a first version of a dedicated output server developed at IPSL. This implementation is technically working but shows disappointing speed-up, mainly due to unnecessary MPI communications. A second version of an IO server has been provided (November 2011) and tests are underway.

NEMO Fault Tolerance

The French collaboration program (ANR "SPADES") has given CERFACS the opportunity to start working on the fault tolerance issues. This preparatory step consisted in an evaluation of the amount of work which is be necessary to adapt the different climate models components or tools (atmosphere, ocean, coupler, etc.) and make them compliant to a fault tolerant FT-MPI [62] environment. This work is long-term: its full validation is not expected before the availability of Exascale supercomputers, but a first prototype will be delivered at the end of 2012 through ANR SPADES collaboration.

The first step was to choose an application from the different models available. The NEMO ocean model [36] is the best candidate, given its relative simplicity: the recent F90 code holds about 100.000 lines, and MPI routines calls are located on a single routine, which simplifies the error handling. The code has been compiled and launched on a supercomputer using ICL University of Tennessee Fault Tolerant OpenMPI environment (FT-MPI) [40]. The error handling is going to be implemented on the model, and the different choices and steps of the implementation documented.

Within WP8, we intend to test the implemented error handling with a test case necessarily different from the ultimately targeted 1,000,000-core configuration. This test case will be defined according to the FT-MPI 'communicator mode' choice. With a “BLANK” communicator mode (fault tolerant FT-MPI supported mode, see documentation for details [48]), our implementation allows that surviving model process keep calculating until the end of the simulation. In order to evaluate our error handling robustness, the test case must be able to reproduce the error structure of the targeted configuration. Present regional configurations, which better reproduce spatial resolution of the future global configuration, could be chosen for such experiment. Ideally, both NEMO fault compliant test case and FT-MPI should be tested on a platform able to reproduce real failures.

4.4.2 Performance Analysis: NEMO

Test Case A is a global, 0.25-degree resolution simulation with sea-ice initially developed by DRAKKAR project [63]. It uses a grid-size of 1442x1021, and 46 depth levels. This is now a standard resolution, and some centers are runnings 0.1 degree in forced mode.

Figure 33 shows scaling results for Test Case A taken from Deliverable D5.4 of the PRACE Preparatory Phase project.

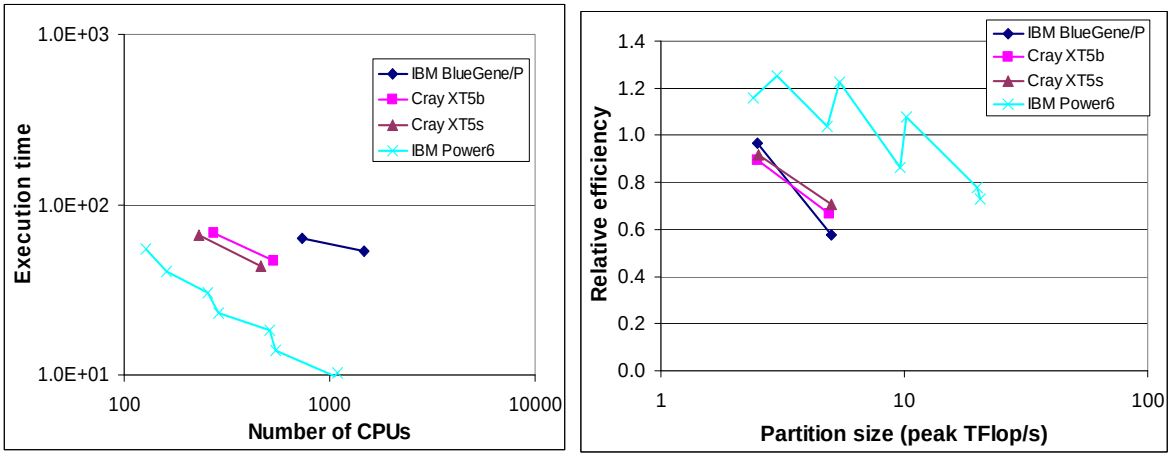


Figure 33: Execution time (l.) and relative efficiency (r.) for NEMO, Test Case A. Credit: A. Porter, STFC.

This test case does not scale well on any of the systems used. Relative efficiency is similar on the Cray XT5 and IBM BlueGene/P, and is lower on these systems than on the IBM Power6.

Profiling Results

All of the following profile data was obtained for NEMO version 3.3 running the smaller ORCA2_LIM configuration (a two-degree resolution global model with a grid of just 182 x 149 with 31 vertical levels). Table 6 contains profile data obtained using the craypat tool for a run on 12 MPI processes of a Cray XE6. In common with many environmental-science codes, the profile is rather flat in that no one routine accounts for a large percentage of the total runtime.

Key to NEMO's poor scaling performance is the large amount of time spent in MPI as seen in in Table 6. To further illustrate this, Figure 34 shows summary profile data for 12- and 24-process count jobs. It is clear that the time spent in MPI increases significantly in doubling the number of MPI processes. Even worse, since HECToR IIb [64] has 24 cores per node, the MPI communications in these jobs are all intra-node. Once a job spans more than one node we can expect the MPI communications to be even more time consuming.

% of total runtime	Imbalance %	Routine
74.4	**USER routines**	
13.4	2.6	limrhg_lim_rhg
5.2	7.8	lib_mpp_mpp_lnk_3d
5.1	2.6	traldf_iso_tra_ldf_iso
4.3	3.9	traadv_tvd_tra_adv_tvd
4.2	3.9	ldfslp_ldf_slp
3.3	7.5	traadv_tvd_nonosc
3.3	2.1	trazdf_imp_tra_zdf_imp
3.2	4.6	zdftke_tke_tke
2.6	2.8	dynzdf_imp_dyn_zdf_imp
2.2	5.0	zdftke_tke_avn
2.1	7.9	field_bufferize_bufferize_field
1.9	6.8	mathelp_moycum
1.6	34.4	solpcg_sol_pcg
1.5	7.5	field_bufferize_init_field_bufferize
1.2	6.5	traadv_eiv_tra_adv_eiv
1.2	6.3	eosbn2_eos_bn2
1.1	7.7	eosbn2_eos_insitu_pot
1.1	8.3	traswp_tra_unswap
1.1	12.7	dynspg_flt_dyn_spg_flt
1.0	6.7	zdfddm_zdf_ddm
1.0	7.8	eosbn2_eos_insitu
15.2	**MPI**	
6.1	23.4	mpi_allreduce
5.9	39.7	mpi_recv
2.5	60.5	mpi_allgather
10.4	**ETC**	
3.5	7.5	_wordcopy_fwd_aligned
1.4	23.8	__c_mcopy8
1.0	28.8	__c_mzero8

Table 6: A profile of NEMO running the ORCA2_LIM configuration on 12 MPI processes on HECToR Phase IIb.

Key to NEMO's poor scaling performance is the large amount of time spent in MPI seen in profile in Table 6. To further illustrate this, Figure 34 shows summary profile data for 12- and 24-process count jobs. It is clear that the time spent in MPI increases significantly in doubling the number of MPI processes. Even worse, since HECToR IIb has 24 cores per node, the MPI communications in these jobs are all intra-node. Once a job spans more than one node we can expect the MPI communications to be even more time consuming.

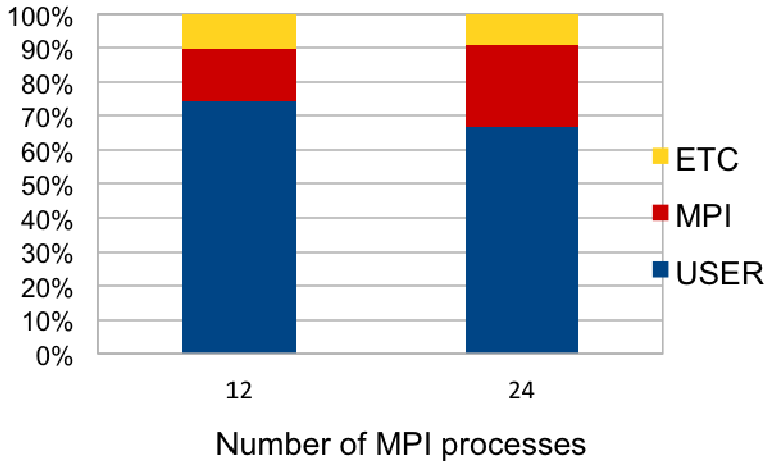


Figure 34: NEMO profile as a function of MPI process count.

We also used the craypat tool [4] on HECToR to measure the computational intensity of NEMO's more significant subroutines. *Table 7* contains another profile of NEMO running ORCA2_LIM but this time including the computational intensity of each subroutine. This shows that, in general, the routines accounting for a greater percentage of run-time have relatively low computational intensities, emphasising the dependence of NEMO on available memory bandwidth.

Routine	% of wall-clock time	Computational intensity (ops/ref)
tra_adv_tvd	6.5	0.82
sol_pcg	6.4	0.78
tra_ldf_iso	5.9	0.88
ldf_slp	5.6	0.98
lim_rhg	3.8	1.17
tra_qsr	3.5	0.81
tra_adv_eiv	3.3	0.42
dyn_zdf_imp	2.8	0.57
tra_zdf_imp	2.6	0.70
mathelp_moycum	2.6	0.62
zdf_ddm	2.3	0.71
eos_insitu_pot	2.0	1.63
eos_bn2	2.0	1.69
eos_insitu	1.6	1.74

Table 7: Profile of NEMO run in serial on a single core of HECToR IIb for the ORCA2_LIM configuration.

4.4.3 Description of Code: ICOM

The Fluidity-ICOM (Imperial College Ocean Model) [36] is built on top of Fluidity, an adaptive unstructured finite element code for computational fluid dynamics. It consists of a three-dimensional non-hydrostatic parallel multiscale ocean model, which implements various finite element and finite volume discretisation methods on unstructured anisotropic adaptive meshes, Fluidity-ICOM uses state-of-the-art and standardised 3rd party software components whenever possible. For example, PETSc [65] is used for solving sparse linear systems while Zoltan is used for many critical parallel data-management services both of which have compatible open source licenses. Python is widely used within Fluidity-ICOM at run time for user-defined functions and for diagnostic tools and problem setup. It requires in total about 17 other third party software packages and use three languages (Fortran, C++, Python).

4.4.4 Performance Analysis: ICOM

Profiling is the best way to address both the serial execution of the code (such as cache usage, vectorisation) and parallel aspects, such as parallel efficiency, load balancing and communications overheads. Profiling using CrayPAT and Vampir on HECToR has been performed on the gyre benchmark [49] test case, which is of particular relevance to GFD applications. CrayPAT is the main tool being used for Fluidity-ICOM profiling. As a starting point, we use simple timing hooks in the code to get a coarse grain profile of code performance, then to use these results as a basis for more fine grain profiling with the CrayPAT API in the identified areas of interest.

Following the above procedure, the first target of this work was to analyse Fluidity-ICOM on a large number of processes (cores) with the gyre test case. The number of processes ranges from 1024 to 4096. In order to identify the issues for this scale of processes, a 10 million node mesh has been generated. We focus on the solution of the momentum equation in combination with the incompressibility constraint given by the continuity equation, as this is by far the main cost of the simulation, and dominates the overall scaling of the simulation. The solution process consists of the assembly of the linear systems representing the discretised momentum equation and the pressure equation, and their solution. The scaling analysis of the momentum equation is naturally broken down into 4 parts: assembly of the pressure matrix, linear solve for the pressure equation, assembly of the discretised momentum (velocity) equation, and linear solve of the momentum (velocity) equation.

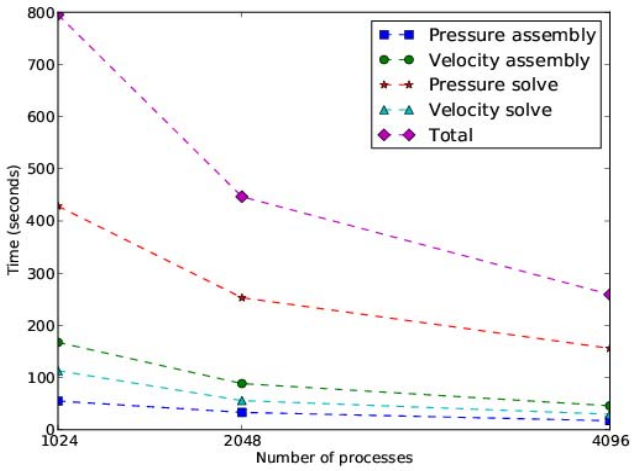


Figure 35: Wall time for the assembly and solve of the momentum and pressure equation on the Hector Cray XE6.

From Figure 35, we can infer that matrix assembly for pressure and velocity can take more than 30% of the total simulation time with 1024 cores, where the pressure solver occupies nearly 53.9% of the total simulation time. The matrix assembly phase is expensive for a number of reasons, including: significant loop nesting, where the innermost loop increases in size with increasing quadrature; indirect addressing (due to unstructured meshes) and cache re-use.

Communication overhead and load balance analysis

Using CrayPAT [4], we obtained the statistics of three groups of functions, namely MPI functions, USER functions and MPI_SYNC functions. MPI_SYNC is used in the trace wrapper for each collective subroutine to measure the time spent waiting at the barrier call before entering the subroutine. Therefore, MPI_SYNC statistics can be a good indication of load imbalance. The time percentage of each group is shown in Figure 36.

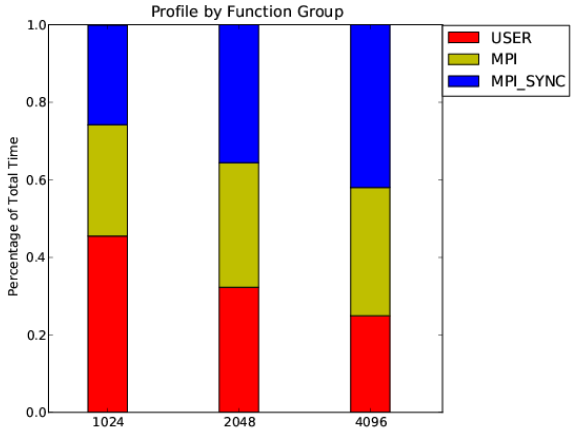


Figure 36: Profile by function group

With core counts from 1024 to 4096, we can see that the time percentage spent in MPI increases from 28.7% to 33.1% while USER functions drop from 45.5% to 24.9%, and time percentage of MPI_SYNC increase from 25.7% to 42.0%. This lead us to identify the top time consuming functions in each group along with their calling hierarchy**.**

Top time consuming functions in each group

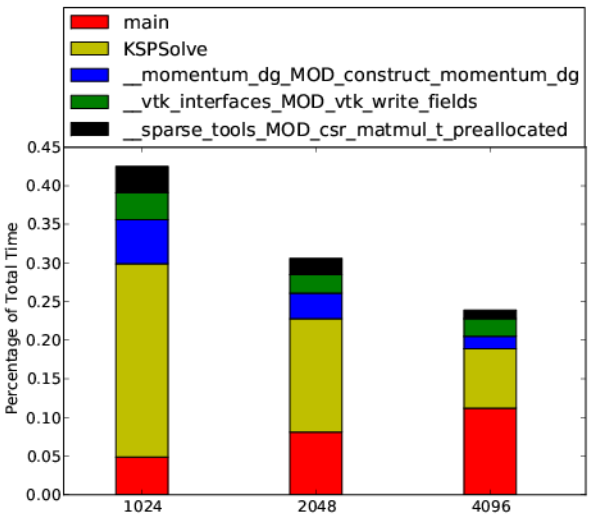


Figure 37: Top time consuming user functions got from CrayPAT.

The subsequent figures give the top time consuming functions in each group. In Figure 37 the speed up of the linear solver KSPSolve is about 3.5 with 4096 cores comparing with 1024 cores according to the CrayPAT tracing results. The function main represents the functions that have not been traced in the code. These functions are outside of momentum solver.

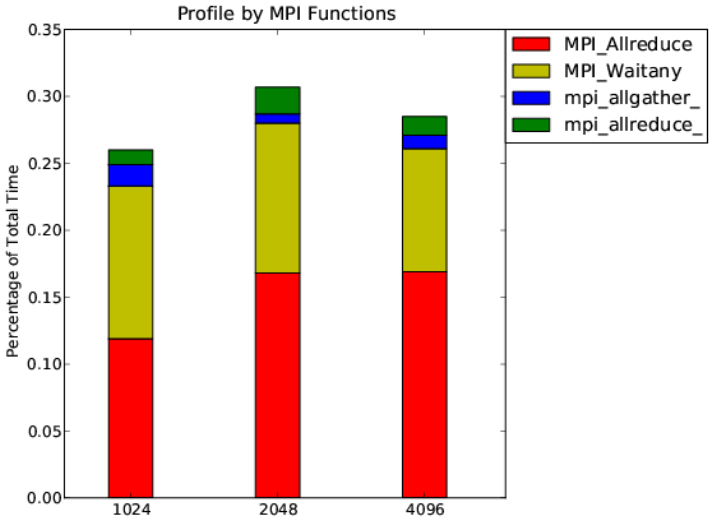


Figure 38: Top time consuming MPI functions.

The most time consuming of the MPI groups is **MPI_Allreduce**. It is expected that this collective operation does not scale well. However on the HECToR the scaling is relatively good from 2048 to 4096 cores. From the call tree generated by CrayPAT, it becomes clear that this function is called from PetscMaxSum within PETSc. MPI_Waitany is indicative of the quality of the load balancing. Given that this amount does not increase significantly between runs on 1024 to 4096 cores in Figure 39, it does not appear that load-balancing is worsening noticeably as the core count increases.

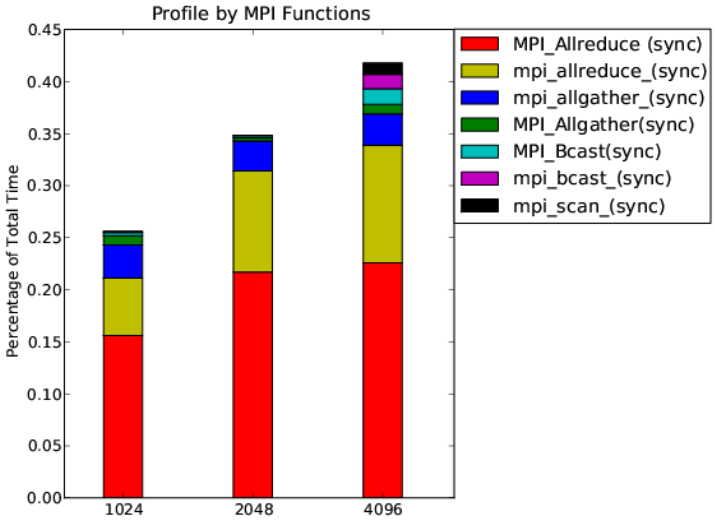


Figure 39: Top time consuming MPI SYNC functions.

In Figure 39, MPI_Allreduce accounts the most part of waiting time spent in the barrier, it is worth to check if it is possible to combine several MPI_Allreduce together. MPI_Bcast and MPI_SCAN are becoming more significant on 4096 cores, compared to runs on 1024 and 2048 cores.

Conclusions

Presently the code is now scaling well up to 4096 cores on the HECToR Cray XE6. Runs on even larger core counts could be achieved if suitably partitioned datasets existed. The ongoing project of hybridising fluidity-ICOM with MPI and OpenMP will further improve scaling of fluidity-ICOM.

Profiling the real world application has turned out to be a big challenge. It required a considerable understanding of profiling tools and extensive knowledge of the software itself. The introduction of manual instrumentation was required in order to focus on specific sections of the code. Determining a suitable way to reduce the profiling data size without losing the fine grain details was critical for successfully profiling. Inevitably this procedure involved much experimentation requiring large numbers of profiling runs.

5. Performance Analysis of Community Codes: Material Science

The codes discussed in the present chapter are representative set of the most used ab initio electronic structure codes in Europe that are freely distributed under open source licenses. The particular selection is based only on community input. Initial input was gathered during the community selection phase prior to the submission of the first deliverable of this work package. During a subsequent face-to-face meeting as well as several conference calls, representatives from the European Theoretical Spectroscopy Facility (ETSF) and the Quantum ESPRESSO community decided to select the following set of codes that are to be further investigated: (1) a suite of codes from the ETSF; (2) Quantum ESPRESSO [67] including PWscf [66]; and (3) Siesta [68]. Siesta was included as a representative of localized numerical basis set codes that are used mostly in chemistry and complement the plane wave pseudopotential codes developed by the ETSF and Quantum ESPRESSO communities. The ETSF suite of codes includes ABINIT [69], EXCITING [70], ELK [71], OCTOPUS [72], and YAMBO [73].

The code descriptions and performance data presented bellow were provided by the communities that maintain the codes. It has to be emphasized that the benchmarks used for the performance analysis represent typical workloads for which the codes have been developed (in contrast to codes and problems selected by centres to emphasis scalability of a code or performance of a supercomputer). Hence the benchmarks are not uniform. No attempt has been made to polish the results. In some cases, such as ABINITI, teams have in the past invested into scalability of parts of the code base. In other cases, such as EXCITIN/ELK, the code has not been running on much more than large workstations and small clusters. The purpose of subsequent work in the present work package will be to improve the performance of these codes and to map them onto future supercomputing platforms, such as hybrid multi-core systems. The intent is to eventually make these codes fit for PRACE Tier 1 and possibly even Tier 0 systems.

5.1 ABINIT

5.1.1 Global description of ABINIT

ABINIT is a package whose main program allows one to find from first principles the total energy, charge density, electronic structure and miscellaneous properties of systems made of electrons and nuclei (molecules and periodic solids) using pseudo-potentials and a plane-wave or wavelet basis. The basic theories implemented in ABINIT are Density Functional Theory (DFT), density-functional perturbation theory (DFPT), Many-Body Perturbation Theory (the GW approximation and Bethe-Salpeter equation), and Time-Dependent Density Functional Theory. The main ABINIT program includes options to optimise the geometry according to the DFT forces and stresses, to perform molecular dynamics simulations using these forces, to determine transition states (string method), to perform path-integral molecular dynamics. It can also directly generate dynamical matrices, Born effective charges, dielectric tensors, and other linear and non-linear coupling quantities, based on Density-Functional Perturbation Theory. Excited states computations from Many-Body Perturbation Theory (the GW approximation) delivers band gaps generally in excellent agreement with experiment, unlike with DFT. Accurate Optical properties are obtained with excitonic effects within the Bethe-Salpether equation.

ABINIT is delivered under the GNU General Public Licence (GPL [74]), and freely distributed on the Web. The documentation is extensive, also provided on the Web. More than 800 automatic tests are integrated in the package, allowing to verify the development by different groups worldwide.

The functional structure of ABINIT, at the highest level, is represented by Figure 40.

Depending on the input parameters, ABINIT will call one (or several) processing unit(s) in turn. These processing units are rather independent, implement different algorithms, and their parallelisation is to be addressed separately.

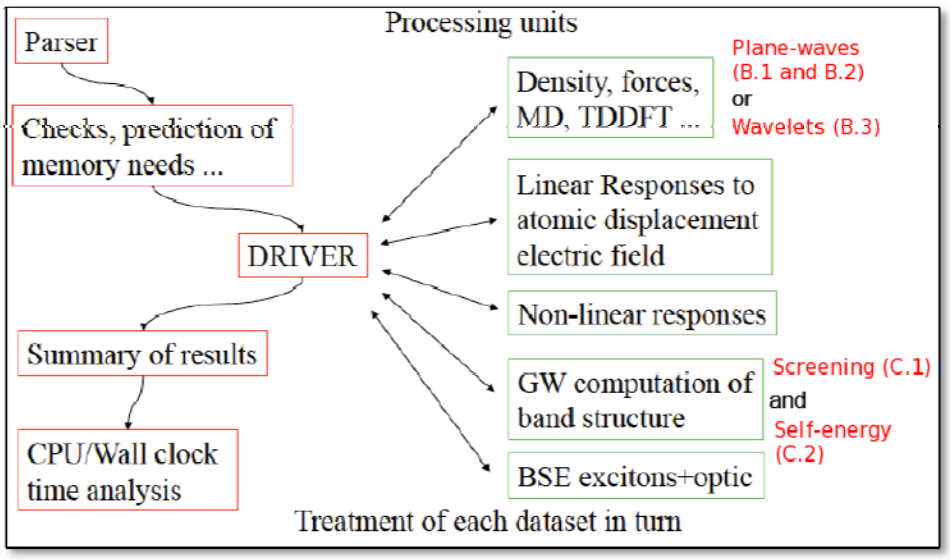


Figure 40: Functional structure of ABINIT.

Section 5.1.2 will address performance issues for the basic DFT calculations using plane-waves and wavelets. In section 5.1.3, the two major steps, based on plane-waves, for the calculation of the electronic structure using MBPT will be detailed: the screening calculation and the self-energy calculation.

In the different sections, several physical or numerical parameters are to be considered in order to understand the computational load and memory scaling, and also the possibilities to distribute it on different processors.

The size of the physical cell (in real space) will usually scale directly with N_{atom}, the number of atoms to be represented, although much larger cells will be needed to host systems placed in vacuum (like molecules, or nanotubes, or slabs) when treated with plane-waves.

The basis set size will directly depend on this size of the cell. It will also depend on the spatial resolution, measured by the kinetic energy cut-off E_{cut} (plane-waves) or the grid spacing (wavelets).

The number of points in real space, usually connected to a Fast Fourier treatment (plane-wave case), will be represented by N_{fft}. The number of plane-waves N_{pw} to be used is usually a fixed fraction of N_{fft}. The number of wavelets will be represented by N_{wvl}.

In case of plane-waves, the resolution (small wavelength details) is governed by the kinetic energy cut-off E_{cut}. Roughly, N_{fft} or N_{pw} are proportional to N_{atom} times $E_{cut}^{3/2}$. In Ground-state DFT as well as MBPT, different resolution grids might coexists (e.g. for the representation of the screening matrix in GW)

The sampling of electronic velocities (or wave-vectors) is described by the number of "*k*-points" in the Brillouin zone, N_{kpt}. Usually it scales inversely proportional to the size of the system, although one cannot go below one *k* point. The number of electronic states, or energy bands N_{band}, is also usually proportional to the number of atoms N_{atom}. In DFT or DFPT, one often treats explicitly only the occupied electronic states or also the low-lying unoccupied states. For MBPT calculation (screening or self-energy), the number of unoccupied states to be treated to converge the results can be much larger than the number of occupied states (a factor 100 is not unusual).

Finally, there are two additional physical quantities that can lead to a distribution of the computational load and memory:

(1) for spin-polarised systems, the two spins can be treated separately, to a large extend. N_{sppol}=2 in this case.

(2) for spinorial wave-functions, the two spinor components can be treated also separately. N_{spinor}=2 in this case.

5.1.2 Ground-State calculations: performances

Historically, ABINIT uses plane-waves to describe the electronic wave functions; it makes an intensive use of Fourier transforms, in particular when applying the local part of the Hamiltonian.

ABINIT parallelisation is exclusively performed using the MPI library for the current stable version and for ground-state calculations. In a beta version, several time consuming code sections of the ground-state part have been ported to GPU. Even if it is already useable, this level of parallelisation is in progress…

In recent years, a development of wave functions on a wavelet basis has been introduced (for the ground state calculations), using wavelet transforms and a specific Poisson operator in real space. The implementation of wavelets has been achieved in the project named "BigDFT" [78]. During this project, a library of functions devoted to wavelets has been produced. It is used by ABINIT and can also be called from a standalone executable. The library and the standalone code are inseparable parts of the ABINIT project.

This section devoted to ground-state calculations with ABINIT is divided in three subsections: 1-plane-waves using MPI, 2-plane-waves using CUDA, 3-wavelets.

5.1.2.1 Electronic ground state calculations using planes-waves; performances using MPI

Parallelisation levels

Several levels of parallelisation have been introduced in ABINIT; they can be used separately or simultaneously:

- Parallelisation over *k points*: this is a classical level of parallelisation in DFT codes. Several terms of the total energy are obtained by integration over the wave-vector space (*k* points). Each contribution to the integral can be computed separately. A final reduction (global communication) is done to get the total energy. As the scaling of this parallelism level is almost linear, it is not checked here.
- Parallelisation over *independent spins*: in the case of spin-polarised systems, each spin component of the density can be computed independently. As the scaling of this parallelism level is almost linear, it is not checked here.
- Parallelisation over *bands*: parallelisation over *plane-waves*: *These two levels of parallelisation are linked.* To solve the Kohn-Sham DFT equations, an eigenvalue problem has to be solved (only the lowest eigenvalues are computed); N_{band} eigenvalues have to be identified, expressing the Hamiltonian matrix on a plane-wave basis (N_{pw} basis elements). In ABINIT, an iterative "by block" algorithm is used (LOBPCG= "*Locally Optimal Block Preconditioned Conjugate Gradient*"). It can be parallelised over N_{band} and N_{pw}. Results concerning this level of parallelisation are presented here.
- Parallelisation over *spinorial components*: in some specific cases (non-collinear magnetism, spin-orbit coupling) each electronic wave function has to be expressed as

a “spinor” (two vectors). This level of parallelisation is implemented in ABINIT but not detailed here.

- Parallelisation over *replicas of the unit cell*: this is a high-level parallelism level. For some specific applications, the simulation cell has to be replicated several times: “*Minimal Energy Path*” research or inclusion of the quantum effect of atomic nuclei (PIMD=“*Path-Integral Molecular Dynamics*”). Although it is needed (especially for PIMD), this distribution of workload can be considered as “*embarrassingly parallel*”. We just verifired that the scaling is linear.

Definition of main time consuming parts:

- *Hamiltonian application*: this routine applies Hamiltonian **H** (and overlap matrix **S**) to the wave-functions. It is divided in three parts:
  - Application of local operator (*Fast Fourier Transform*)
  - Application of non-local operator
  - MPI communications (“*alltoall*”)
- *LOBPCG algorithm:* this routine solves the eigenvalue problem by minimisation using *LOBPCG* algorithm; it mainly uses linear algebra.
- *Diagonalisation/orthogonalisation of wave functions:* this routine solves the eigen-problem in wave-functions subspace; it mainly uses linear algebra.
- *Local Potential:* this routine computes the evolving parts of the local potential (*Hartree* + exchange-correlation).
- *Forces:* this routine computes forces on atoms.

Band-FFT strong scaling

In the following, we test how the increasing of cores at fixed load (i.e., in a strong scaling regime) affects the performance of each of the functions. We verify that the code scales linealry with the number of cores and try to find at what number of cores a « plateau » is reached. Such tests help to check if the core work is well balanced.

Test case: one vacancy in a 108 Gold atoms unit cell (i.e., 107 atoms). *Gamma k*-point calculation. The (unrelaxed) vacancy breaks symmetries and induces large forces. “Projector Augmented-Wave” (PAW) method is used.

Libraries used: MPI and ScaLAPACK

Keeping fixed the number of atoms and increasing the number of CPU cores.

We check here that, mixing band and FFT level, the performance model is not simple. For a given total number of CPU cores, performances directly depend on their distribution on bands and plane-waves.

Varying only plane-wave CPU cores

CPU total clock time (s) ABINIT part	65 (band) x 2 (npw) *130 cores*	65 (band) x 4 (npw) *260 cores*	65 (band) x 6 (npw) *390 cores*	65 (band) x 8 (npw) *520 cores*
Hamiltonian application: local operator (FFT)	2538,0	4547,9	8342,9	8720,9
Hamiltonian application: non-local operator	20568,9	19451,4	18698,3	21086,7
Hamiltonian application: *alltoall* communications	3167,0	6613,9	12453,1	14518,3
LOBPCG algorithm (without Hamiltonian application)	29385,1	74593,0	108648,9	157520,6
Diagonalisation/orthogonalisation of wave-functions	1308,0	3570,6	9277,6	15312,9
Local potential computation (*Hartree* + XC)	8601,4	12986,4	13081,0	15773,6
Forces computation	3272,8	4861,3	4918,0	5596,9
Others	6972,4	10932,3	11858,8	21830,1
Total	**75813,6**	**137556,8**	**187278,6**	**260360,0**

Table 8: CPU total clock time of ABINIT varying the number of plane-wave CPU cores.

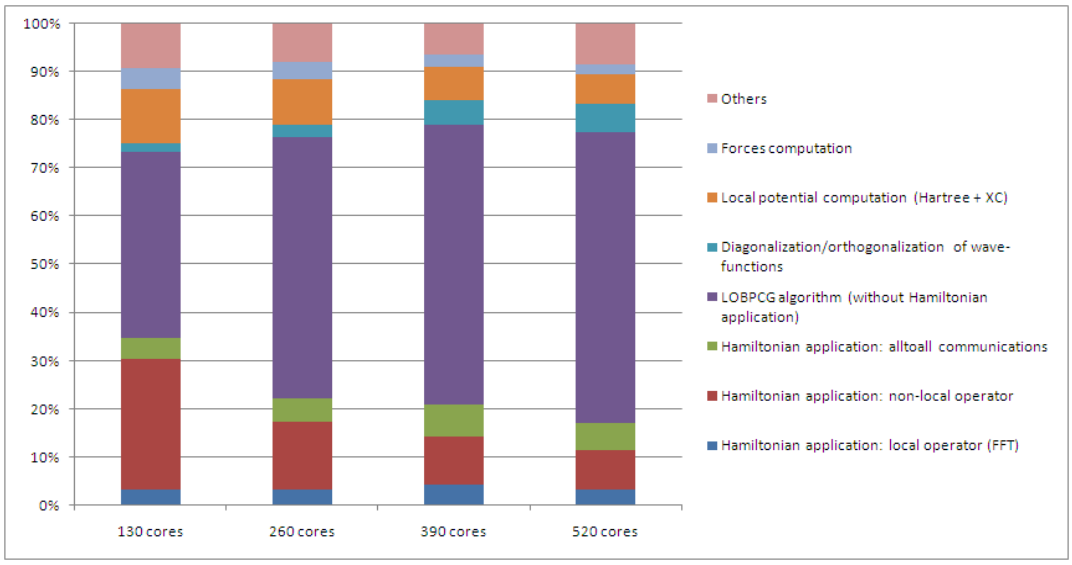


Figure 41: Repartition of time in ABINIT routines varying the number of plane-wave CPU cores. While some parts of the code scale linearly (ex: non-local operator), others become predominant. A plateau is observed at 390 cores.

Varying only band cores

CPU total clock time (s) ABINIT part	9 (band) x 6 (npw) *72 cores*	18 (band) x 6 (npw) *144 cores*	36 (band) x 6 (npw) *216 cores*	72 (band) x 6 (npw) *432 cores*
Hamiltonian application: local operator (FFT)	4601,9	5085,0	5196,1	6461,3
Hamiltonian application: non-local operator	20891,1	20957,1	20982,7	23122,3
Hamiltonian application: *alltoall* communications	3248,7	4196,8	4708,0	10966,4
LOBPCG algorithm (without Hamiltonian application)	4581,1	27232,0	50083,6	361190,1
Diagonalisation/orthogonalisation of wave-functions	767,4	1731,3	2736,7	9520,1
Local potential computation (*Hartree* + XC)	1890,1	4063,6	6405,4	14779,9
Forces computation	3272,8	4861,3	4918,0	5596,9
Others	-683,7	714,2	3912,3	20955,2
Total	38569,4	68841,3	98942,8	452592,2

Table 9: CPU total clock time of ABINIT varying the number of band CPU cores.

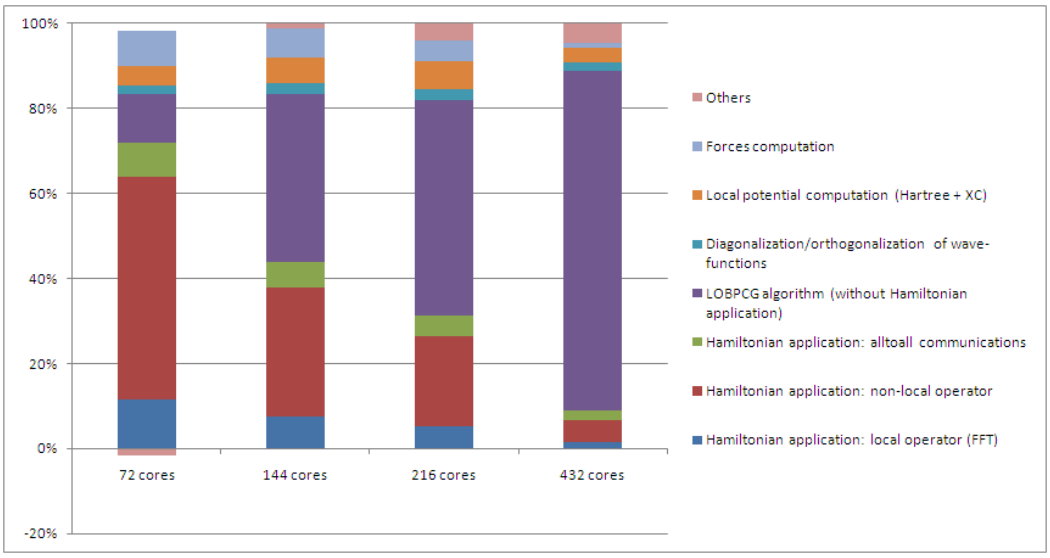


Figure 42 Repartition of time in ABINIT routines varying the number of band CPU cores. While some parts of the code scale linearly (ex: non-local operator, forces), others become predominant. On 432 cores, the codes clearly has no more a linear behavior.

Band-FFT weak scaling

Differently from the previous two cases presented (strong scaling), we keep here the number of cores fixed observing the performance when the number of atoms changes. In this case the problem size (workload) assigned to each processing element stays constant and additional elements are used to solve a larger total problem.

Test case: a 108 (or 54) Gold atoms unit cell; atomic positions obtained from a Molecular Dynamics simulation at 500K. *Gamma k*-point calculation. The "Projector "Projector Augmented-Wave" (PAW) method is used.

Libraries used: MPI and ScaLAPACK

Varying the number of atoms keeping the number of CPU cores constant

CPU total clock time (s) ABINIT part	55 (band) x 8 (npw) *108 atoms* *440 cores*	55 (band) x 8 (npw) *54 atoms* *440 cores*
Hamiltonian application: local operator (FFT)	6571,1	1852,1
Hamiltonian application: non-local operator	23775,7	1831,9
Hamiltonian application: *alltoall* communications	8037,5	3691,3
LOBPCG algorithm (without Hamiltonian application)	378221,4	32112,2
Diagonalisation/orthogonalisation of wave-functions	9912,5	5485,0
Local potential computation (*Hartree* + XC)	15262,4	3569,5
Forces computation	5379,0	854,5
Others	21087,2	11831,3
Total	**468246,8**	**61227,8**

Table 10: CPU total clock time of ABINIT varying the number of atoms.

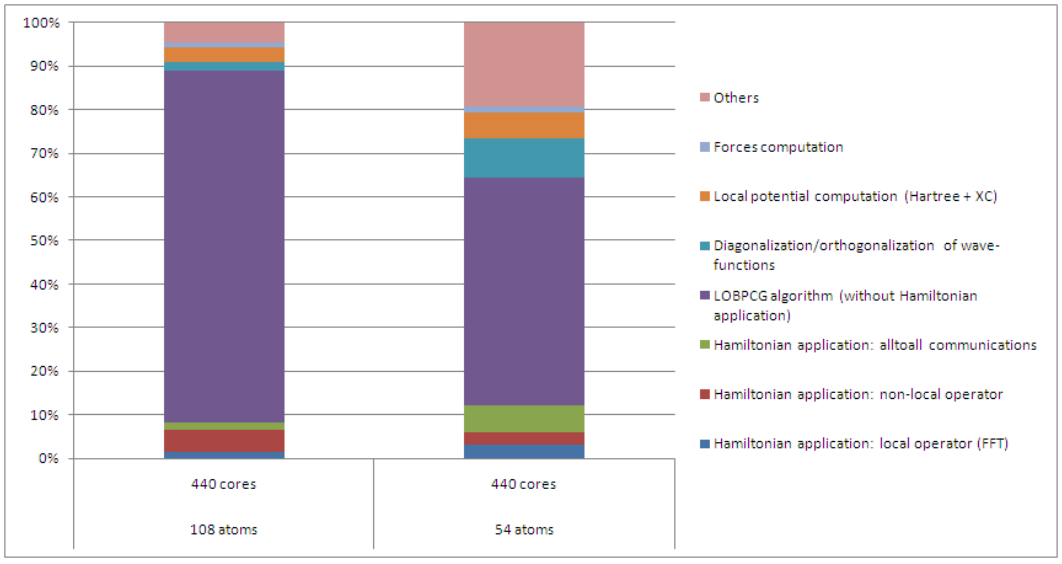


Figure 43: Repartition of time in ABINIT routines varying the number of atoms. This test case is not a full « weak scaling » performance test as the number of cores is kept fixed. When only the size of the system is increased, the resolution of the eigenvalue problem becomes the predominant part.

Varying the number of atoms and the number of CPU cores

CPU total clock time (s) ABINIT part	55 (band) x 8 (npw) *108 atoms* *440 cores*	55 (band) x 4 (npw) *54 atoms* *220 cores*
Hamiltonian application: local operator (FFT)	6571,1	874,5
Hamiltonian application: non-local operator	23775,7	1759,2
Hamiltonian application: alltoall communications	8037,5	1145,8
LOBPCG algorithm (without Hamiltonian application)	378221,4	14544,8
Diagonalisation/orthogonalisation of wave-functions	9912,5	1234,5
Local potential computation (Hartree + XC)	15262,4	2901,4
Forces computation	5379,0	744,0
Others	21087,2	3370,2
Total	**468246,8**	**26574,4**

Table 11: CPU total clock time of ABINIT varying the number of atoms and number of cores.

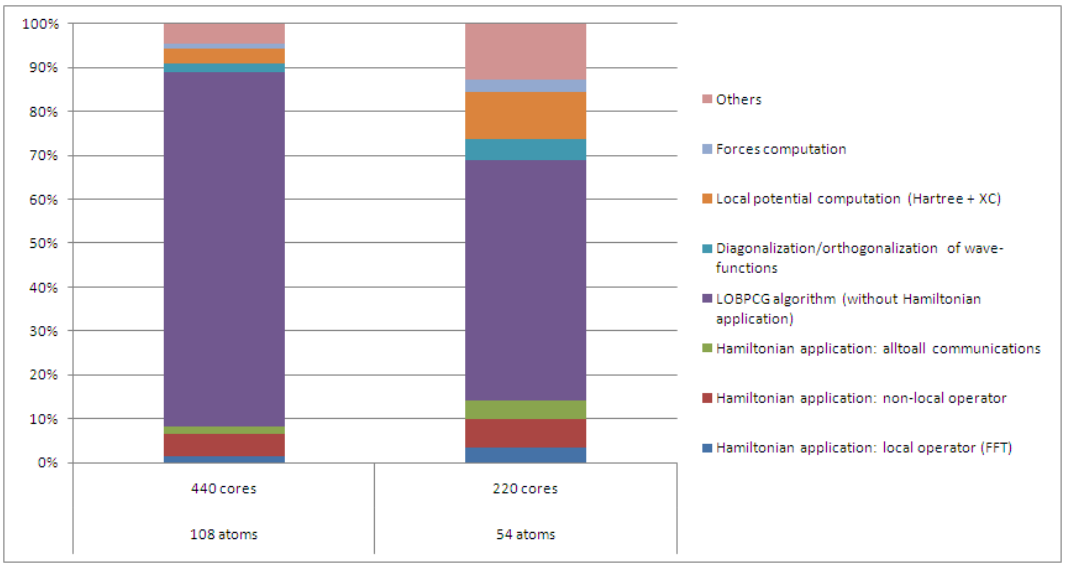


Figure 44: Repartition of time in ABINIT routines varying the number of atoms and the number of cores. This weak scaling performance test clearly shows that the code does not scale linearly which is an expected behavior for a DFT code. As the size of the simulation cell increases, the number of plane waves increase as the cube of the cell size.

Influence of the CPU cores distribution in the parallelisation levels

Test case: PuO_2 surface: 60 atoms (correlations, magnetism, *f* electrons, vacuum…). 400 bands; *gamma k*-point calculation

"Projector Augmented-Wave" (PAW) method is used.

We check here how the distribution of CPU cores in the (N_{band} x N_{pw} x N_{kpt}) levels of parallelisation influences the performance.

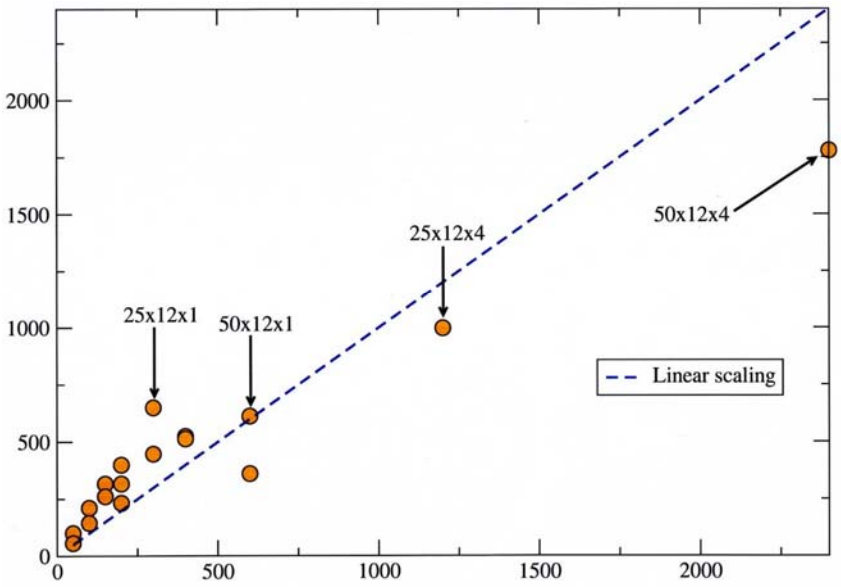


Figure 45: Scaling of ABINIT wrt the distribution of (Nband x Npw x Nkpt) CPU cores

Parallelisation over replicas of the simulation cell

Test case: calculation of the energy barrier between two positions of a silicon interstitial atom; 65 silicon atoms; 4 *k-points*; 130 bands; PAW method.

We check here that this level of parallelisation scales almost linearly.

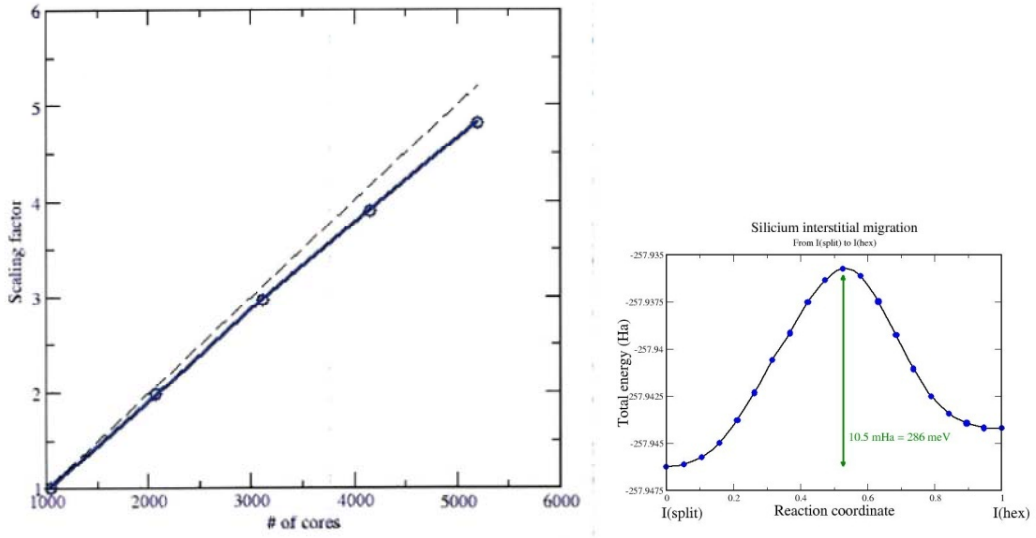


Figure 46 Scaling of ABINIT wrt the CPU cores distributed on the replicas of the cell.

5.1.2.2 Electronic ground state calculations using planes-waves; performances using CUDA

Use of Graphic Processing Units (GPU) will be available in the 6.12 version ABINIT; this implementation is in beta stage. It uses NVIDIA CUDA library and is still evolving. The following performance profiling has to be considered as a step toward the full implementation.

Three parts of the code have been parallelised:

- Application of the local Hamiltonian (Fast Fourier Transform); it has been chosen to use the *cuFFT* library (included in CUDA package).
- Application of the non-local Hamiltonian; specific CUDA *kernels* have been written for the non-local operator and its derivatives (forces, stresses…).

- Linear algebra used in the LOBPCG algorithm and diagonalisation-othogonalisation in the wave-functions subspace; for that purpose we use *cuBLAS* library delivered in CUDA package and GNU-GPL MAGMA library (LAPACK on GPU [75]).

The GPU implementation is fully compatible with all MPI levels of parallelisation except the FFT level. Only results obtained with one CPU cores are shown here in order to isolate the GPU performances only.

Test cases (all done using PAW):

- ***Test Au***: 107 gold atoms (vacancy in gold crystal)
- ***Test $BaTiO_3$***: 39 atoms (one vacancy in 8 $BaTiO_3$ units)
- ***Test Cu***: 20 copper atoms

Application of local Hamiltonian
Use of cuFFT library for the Fast Fourier Transform of wave functions.

Test case	***FFT CPU time (sec)***	***FFT GPU time (sec)***
Test Au	246,1	156,2
Test $BaTiO_3$	164,2	120,3
Test Cu	19,6	25,4

Table 12: Comparison of elapsed time for the wave function FFT.

As shown in **Table 12**, executing FFT on GPU is only profitable when the size of wave functions is large enough. However, according to the simulated system, it is possible to send several wave functions together to the graphic card, increasing the FFT efficiency. **Table 13** shows the results obtained for the smallest test case (Test Cu), varying the number of wave functions sent to the GPU:

# of wave-functions sent to the GPU	***FFT GPU time (sec)***
1	25,4
2	20,8
4	14,1
84	9,6

Table 13: Elapsed time for the wave function FFT w.r.t. the number of WF sent.

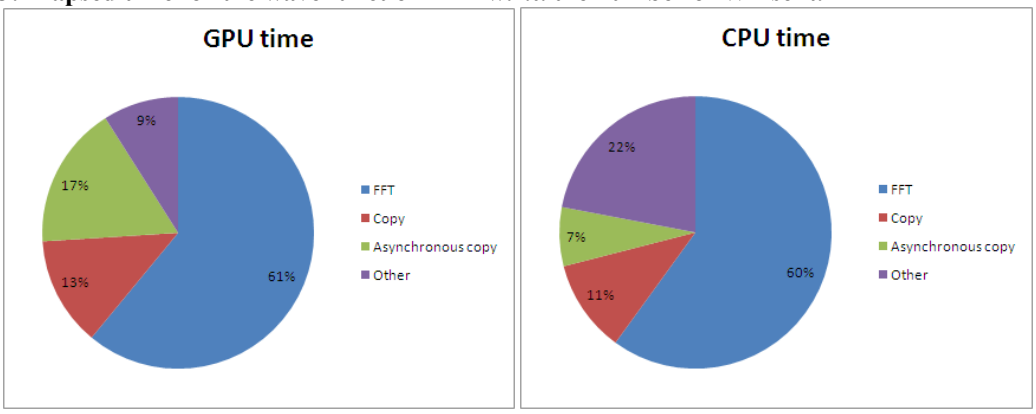


Figure 47: Profiling of elapsed time for the application of FFT to one wave function, in the "Test Cu" test case; "GPU time" corresponds to the bare GPU time needed by the graphic card to execute the FFT task; "CPU time" corresponds to the total elapsed time, including kernel latencies and synchronisations.

Application of non-local Hamiltonian

Specific CUDA kernels have been implemented to compute application of non-local operator and its contribution to energy, forces and stress tensor.

Test case	***NL op. CPU time (sec)***	***NL op. GPU time (sec)***
Test Au	3142,0	1172,9
Test $BaTiO_3$	185,1	160,2
Test Cu	85,1	42,4

Table 14: Comparison of elapsed time for the application of non-local operator.

As shown in **Table 14**, the efficiency of the adopition of the non-local Hamiltonian based approach on the GPU strongly depends on the number and type of treated electrons (*s*, *p*, *d* or *f*).

Application of linear and matrix algebra

All vector/matrix multiplication in LOBPCG algorithm are done using the cuBlas library. Orthogonalisation and diagonalisation of the Hamiltonian in the wave-function subspace uses MAGMA package. It can be shown that this use of MAGMA is only profitable when the size of matrixes is large enough. A threshold value has been introduced to call MAGMA routines only when they are efficient. It can be shown that this use of MAGMA is only profitable when the size of matrixes is large enough. A threshold value has been introduced to call MAGMA routines only when they are efficient. This value has been empirically estimated to 100 (size of matrix) and can also be determined « on the fly » by launching a small lapack routine at the start of the code.

Test case	***LOBPCG CPU time (sec)***	***LOBPCG GPU time (sec)***
Test Au	761,9	593,1
Test $BaTiO_3$	709,0	342,1
Test Cu	21,3	12,3

Table 15: Comparison of elapsed time for the LOBPCG algorithm.

Global profiling on GPU

Performances are subject to change as the development of the GPU code is a work in progress. In the following table we show performances of the ABINIT running on GPU in the present state of the code (v6.12 to be released in december 2011). Improvements of the GPU implementation are ongoing and have not been included.

Test case	**Curie GPU Fermi** ***Time (sec)***	**Curie CPU** ***Time (sec)***	**Titane GPU Tesla** ***Time (sec)***	**Titane CPU** ***Time (sec)***
Test Au	609,8	1528,0	1856,8	4230,7
Test $BaTiO_3$	681,1	865,1	810,11	849,0
Test Cu	67,5	235,6	102,2	153,0

Table 16: Comparison of total elapsed times using (or not) GPU on two different architectures; Curie: CPU=Intel Westmere, GPU=NVidia Fermi M2090; Titane: CPU=Intel Nehalem, GPU=NVidia Tesla S1070

5.1.2.3 Electronic ground state calculations using wavelets: BigDFT profiling

BigDFT is a project that should be considered inseparable from ABINIT. It consists of two (connected) parts: a library, which is used by ABINIT, and a standalone code. We present here some performance of the standalone code.

Two data distribution schemes are used in the parallel version of the program. In the orbital distribution scheme, each processor works on one or a few orbitals for which it holds all its scaling function and wavelet coefficients. In the coefficient distribution scheme each processor holds a certain subset of the coefficients of all the orbitals. Most of the operations — such as applying the Hamiltonian on the orbitals and the preconditioning — are done in the orbital distribution scheme. This has the advantage that we do not have to parallelise these routines with MPI. The calculation of the Lagrange multipliers that enforce the orthogonality constraints onto the gradient as well as the orthogonalisation of the orbitals is done in the coefficient distribution scheme. A global reduction sum is then used to sum the contributions to obtain the correct matrix. Such sums can easily be performed with BLAS-LAPACK routines. Switch back and forth between the orbital distribution scheme and the coefficient distribution scheme is done by the MPI global transposition routine *MPI ALLTOALL(V)*. For parallel computers where the cross sectional bandwidth scales well with the number of processors this global transposition does not require much CPU time. Another time-consuming communication is the global reduction sum required to obtain the total charge distribution from the partial charge distribution of the individual orbital.

MPI parallelisation performances. Architecture dependence

The parallelisation scheme of the code is tested since its first version. Since MPI communications do not interfere with calculations, as far as the computational workload is more demanding than time needed for communication, the overall efficiency is always higher than 88%, also for large systems with a large number of processors.

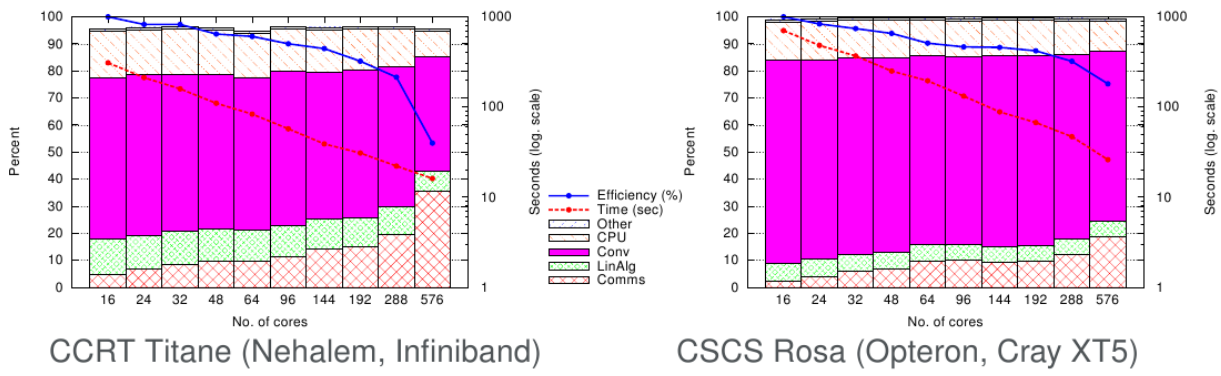


Figure 48: Comparison of the performances of BigDFT on different platforms.

Runs on CCRT machine are worse in scalability but better in performances than runs on CSCS one (1.6 to 2.3 times faster). French CCRT Titane platform (Bull Novascale R422 [76]) is compared to Swiss Rosa Cray XT5 [8]. The latter have better performances for communication, and the scalability performances are quite good. However, from the « timeto-solution » viewpoint, the former is about two times faster. This is mainly related to better performances of the linear algebra libraries (Intel MKL [77] compared to Istanbul linear algebra) and of the processor.

OpenMP parallelisation

In the parallelisation scheme of the BigDFT code another level of parallelisation was added via *OpenMP* directives. All the convolutions and the linear algebra part can be executed in multi-threaded mode. This adds further flexibility to the parallelisation scheme. Several tests and improvements have been performed to stabilise the behaviour of the code in multilevel

MPI/OpenMP parallelisation. At present, optimal performances can be reached by associating one MPI process per CPU, or even one MPI per node, depending on the network and MPI library performances. This has been possible also thanks to recent improvements of the *OpenMP* implementation of the compilers.

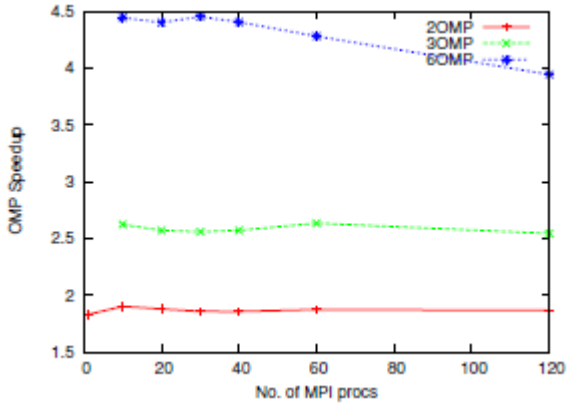


Figure 49: Speedup of OMP threaded BigDFT code as a function of the number of MPI processes. The test system is a B80 cagem and the machine is Swiss CSCS Palu (Cray XT5, AMD Opteron).

GPU acceleration

The BigDFT code is well suited for GPU acceleration. On one hand the computational nature of 3D separable convolutions may allow to write efficient routines, which may benefit of GPU computational power. On the other hand, the parallelisation scheme of BigDFT code is optimal in this sense: GPU can be used without affecting the nature of the communications between the different MPI process. This is in the same spirit of the multi-level *MPI/OpenMP* parallelisation. Porting has been done within the Kronos OpenCL standard, which allows for multi-architecture acceleration.

In the following figure, systems of different sizes have been run in different conditions. The response of the code in the case of an under-dimensioned calculation (where the amount of communication is of the same order as the calculation) has been tested. This may happen if the system is too small, or if the ratio between the runtime GigaFlop/s of the computations and the cross-sectional bandwidth of the network is high.

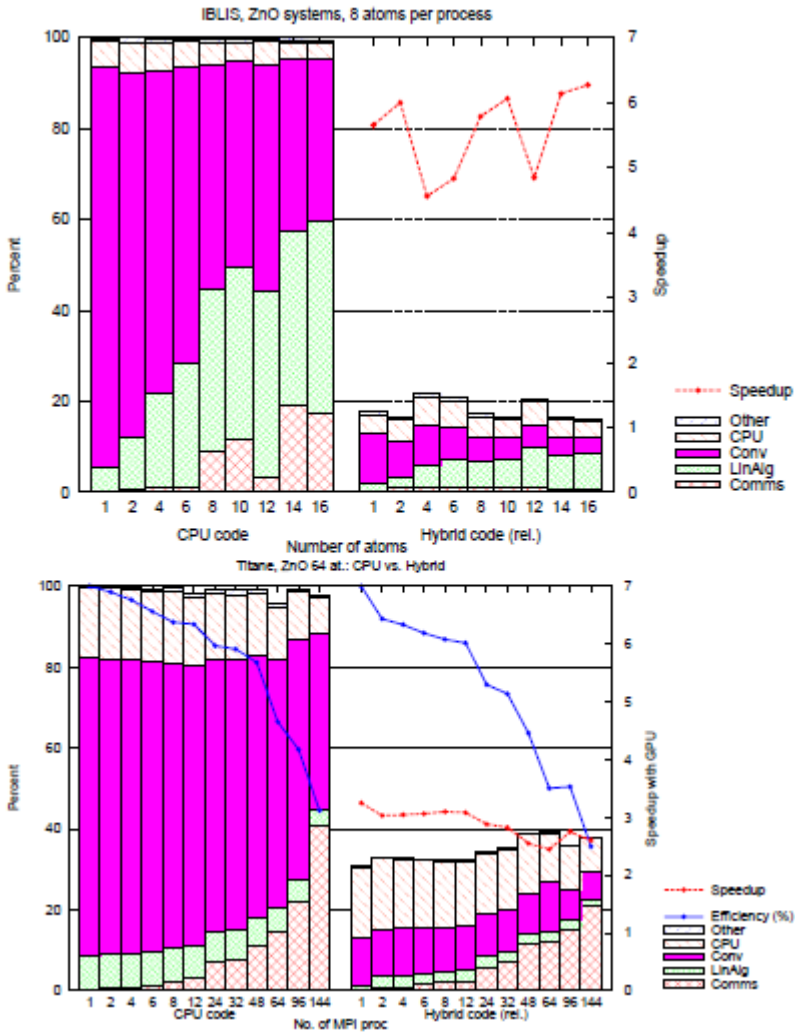


Figure 50: Relative speedup of the hybrid DFT code wrt the equivalent pure CPU run. In the top panel, different runs for systems of increasing size have been done on a Intel X5472 3GHz (Harpertown) machine. In the bottom panel, a given system has been tested with increasing number of processors on an Intel X5570 2.93GHz (Nehalem) machine. The scaling efficiency of the calculation is also indicated. It presents poor performances due to the fact that the system is too little for so many MPI processes. In the right side of each panel, the same calculation have been done by accelerating the code via one Tesla S1070 card per CPU core used, for both architectures. The speedup is around a value of six for a Harpertown, and around 3.5 for a Nehalem based calculation.

5.1.3 Excited States calculations: performance

Description of the test

The test case is a relaxed 2x2x1 *supercell* of *wurtzite* ZnO with an *oxygen* defect (one oxygen removed). The cell contains 31 atoms, corresponding to 205 occupied electronic bands. The Projector Augmented-Wave (PAW) method is used for all-electron precision, with a 2x2x2 k-point grid in reciprocal space. A *plasmon-pole* model is used for the screening calculation.

For optimal load balancing the number of bands to be calculated was set to 717 and 1229 for the screening calculation, and to 1024 for the self-energy.

5.1.3.1 Screening

The most time-consuming parts of the screening calculation are:

setup: Initialisation of run

This section reads and checks the header of the KSS file containing the wave functions and electronic band energies and performs the setup of the basic objects needed for computing the screening (pseudo-potentials, PAW objects, GW objects, etc.) This part is not parallelised.

rdkss: Reading of the Kohn-Sham orbitals

This routine reads KSS file employing plain Fortran-IO. Each node opens the file and reads a subset of bands. This routine does not scale and it is expected to have a detrimental effect on the scaling.

qloop: Matrix inversion and write to file

This routine calculates the inverse dielectric matrix via matrix inversion and writes the SCR file. The inversion is done in serial and the writing is performed by the master node using Fortran-IO primitives. This component does not scale.

cchi0q0: Computation of the polarisability for q = 0

cchi0: Computation of the polarisability for non-zero q

These routines are parallelised over the empty bands. The implementation scales optimally with the number of processors provided that the number of CPUs divides the number of conduction states used in the calculation.

The tests were executed with ABINIT version 6.5.0.

To decrease the high demand of memory *gwmem*=10 was used, but still at least 16 nodes (8GB memory each) were needed.

Results

The functions *cchi0* and *cchi0q0* scale well with the number of processes. This scaling is nearly independent of the number of bands to be calculated, which is different for the total speedup, as to be seen in Figure 51. Apparently the relative cost of the non-scaling functions is higher when calculating fewer bands. Figure 52 compares the partitioning of the workload for different numbers of bands. One can see that the fraction of the well scaling functions (*cchi0* and *cchi0q0*) is 99% when using 16 processes. On 512 processors this decreases to 79% in the 1229 bands case, and even to 64% for 717 bands.

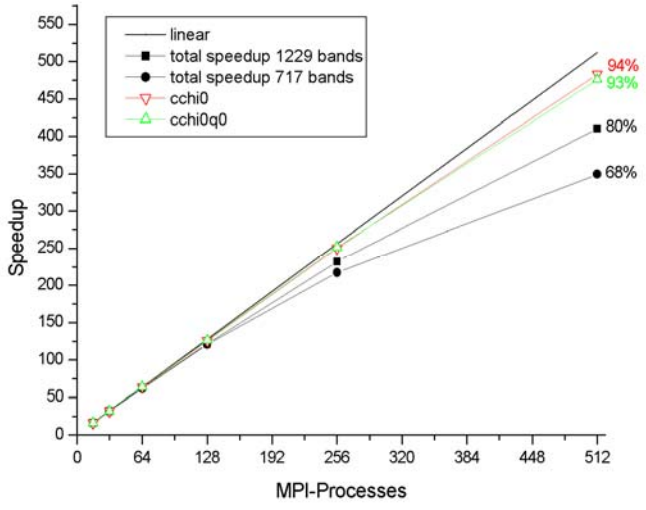


Figure 51: Speedup for the scaling parts of the screening calculation and total speedup for different numbers of bands

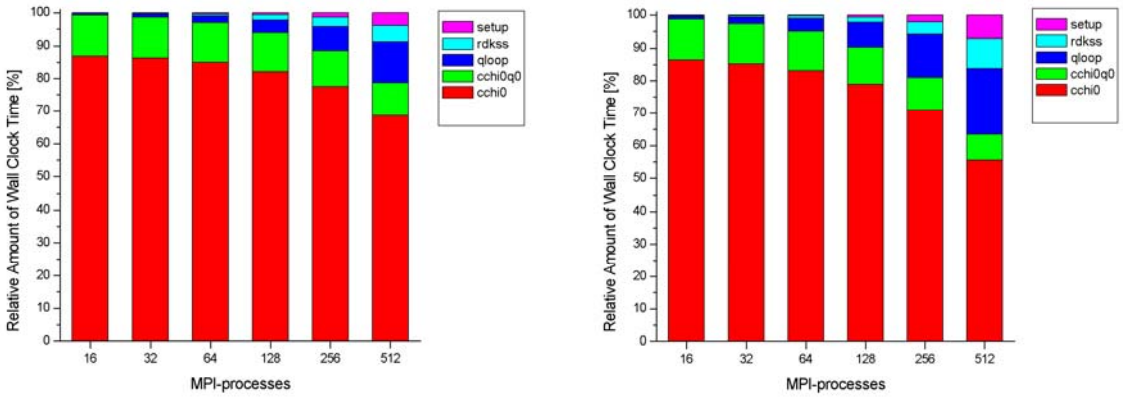


Figure 52: Relative cost of the most time-consuming code sections On the left for 717 bands, on the right for 1229 bands.

5.1.3.2 Self-Energy

The calculations in the sigma-part can be decomposed into:

Init: Initialisation of the run

*setup*_`sigma`: Initial setup of the self-energy (not parallelised)

rdkss: Reading of the Kohn-Sham orbitals (wave function file)

This routine reads KSS file employing plain Fortran-IO. Each node opens the file and reads a subset of bands. This routine does not scale and it is expected to have a detrimental effect on the scaling.

csigme: Calculation of the self-energy matrix elements

There are two components to be calculated, the exchange contribution and the correlation part. The computation of the correlation (the most time consuming part) should scale up to the total number of bands *occupied+unoccupied* (N_{band}). For an optimal distribution the number of processors should divide N_{band}.

Note, however, that the calculation of the exchange term will not scale anymore when the number of processor exceeds the number of occupied bands (205 in our case). This should explain part of the degradation of the speedup in csigme when Ncpu >= 128, this limitation can be lifted by implementing a new algorithm that distributes the computation over transitions instead of distributing bands.

Further there is the function *Init2*, whose contribution is negligible and thus will not be discussed here.

The ABINIT version 6.10.1. was used.

For this test the parameter *gwmem*=11 was used, yielding faster but more memory-demanding calculations. As a result it was not possible to run the problem on less than 4 nodes, each featuring 8GB of memory.

Results

As to be seen in Figure 53, the only function scaling well is *csigme*, which takes most of the computation time when using a small number of processes (Figure 54). This changes rapidly, decreasing from 99% to 33% when going from 4 to 512 processes. In particular *rdkss* takes over a lot of time.

The resulting total speedup is quite good up to 64 processes, where still an efficiency of 67% can be reached, but decays quickly beyond that point, reaching only 16% at 512 processes.

This result can be expected due to the small number of bands (1024) to be calculated, giving only two bands per processor. Note that *rdkss* (which acts in serial) starts to take up a significant amount of the calculation when the number of processor is comparable to the number of bands.

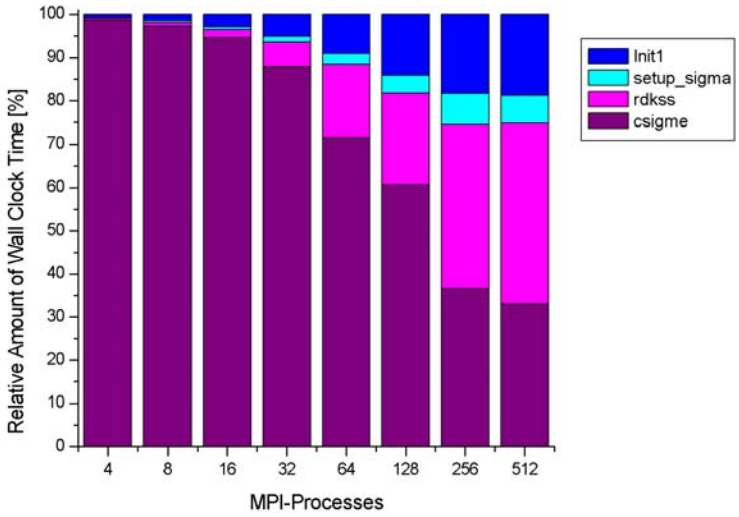


Figure 53: Speedup for the screening part and its most costly sections

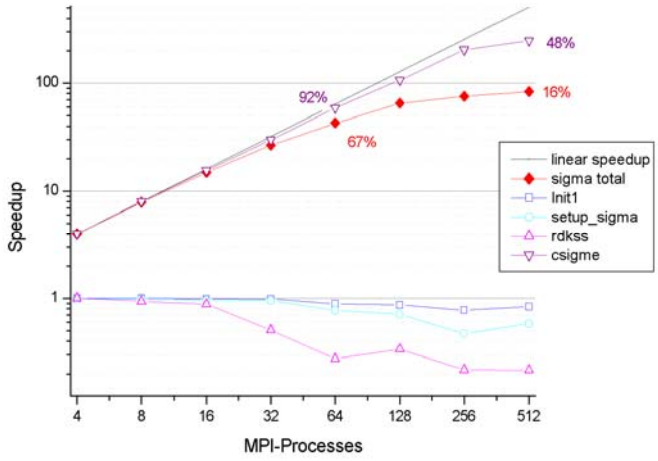


Figure 54: Relative amount of wall clock time for the partitioning of the sigma calculation.

5.2 Quantum ESPRESSO

5.2.1 Description of the code

Quantum ESPRESSO is an integrated suite of computer codes based on density-functional theory, plane waves, and pseudo-potentials - separable, norm-conserving and ultrasoft - and projector-augmented waves. The acronym ESPRESSO stands for opEn Source Package for Research in Electronic Structure, Simulation, and Optimisation. It is freely available under the terms of the GNU General Public License (GPL). It builds upon newly restructured

electronic-structure codes that have been developed and tested by some of the original authors of novel electronic-structure algorithms and applied in the last twenty years by hundreds of materials modelling groups.

Quantum ESPRESSO is a modular software package developed and presented with two goals: 1) to enable state-of-the-art materials simulations, and 2) to foster methodological innovation in the field of electronic structure and simulations by providing highly efficient, robust, and user-friendly open source codes containing most recent developments in the field. This approach blurs the line separating development and production codes and engages and nurtures the user community by inviting their software contributions. These are included in the distribution after being verified, validated, and made fully inter-operable with other modules. A web portal, the qe-forge [50], was set up as a dynamic space for content creation and sharing to facilitate the coordination and integration of the programming efforts of various groups and to easy the dissemination of software tools. It is the main tool by which QE end users and external contributors maintain QE-related projects and make them available to the community.

Two of the main codes contained in the suite Quantum ESPRESSO are PW and CP.

PW – This is the total energy code using an iterative approach to self-consistency and iterative diagonalisation techniques in the framework of the plane-wave pseudo-potential method. All forms of pseudo-potentials, exchange and correlation functionals, and extended functionals mentioned above are implemented in this code. Spin-polarisation, and non-collinear magnetism as induced by spin–orbit interactions are also implemented. Self-consistency is achieved via the modified Broyden method. Crystal symmetries are automatically detected and exploited to restrict the BZ (k-point) sampling to the irreducible wedge. BZ integrations in metallic systems can be performed using a variety of smearing/broadening techniques. The finite-temperature Mermin functional is also implemented. The calculation of minimum-energy paths, activation energies, and transition states uses the nudged elastic band (NEB) method. Potential energy surfaces as a function of suitably chosen collective variables can be studied using metadynamics. Microcanonical (NVE) MD is performed on the Born-Oppenheimer surface using the Verlet algorithm. Canonical (NVT) dynamics can be performed using velocity rescaling, or Anderson's or Berendsen's thermostats. Constant pressure (NPT) MD is performed by adding additional degrees of freedom for the cell size and volume, using either the Parrinello–Rahman or the invariant Lagrangian by Wentzcovitch. A quantum fragment can also be embedded in a complex electrostatic environment such as a model solvent.

The main bottlenecks in PWscf code are the calculations that utilise 3D-FFTs, linear algebra (matrix-matrix multiplications), space integrals and point-function evaluations.

CP - The CP code is a specialised module performing Car–Parrinello ab initio MD. It is quite efficient for large-scale calculations. CP implements all exchange-correlation functionals but the hybrids. A simplified one-electron self-interaction correction (SIC) and Hubbard U corrections are implemented. k-point sampling is restricted to the Γ-point. CP can also treat metallic systems. Microcanonical (NVE) MD on electronic and nuclear degrees of freedom use the Verlet algorithm. Constant-pressure (NPH) uses the Parrinello–Rahman Lagrangian. Nose'–Hoover thermostat and chains allow simulations for different canonical ensembles. CP can also be used to achieve electronic self-consistency or to perform structural minimisations using the 'global minimisation' approaches, damped dynamics, or conjugate-gradients on all degrees of freedom. It can also perform NEB and metadynamics calculations. Finite homogeneous electric fields are implemented using the Berry-phase method. This feature can be used in combination with MD to obtain the infrared spectra of liquids, low- and

high-frequency dielectric constants and coupling factors required to calculate vibrational properties, including infrared, Raman and hyper-Raman spectra.

5.2.2 Performance: PW.X – plane-wave self consistent calculations

Test Case 1: 216 atoms MnGeSbTe (gamma point calculation) [1]

The profiling for this Test Case has been performed keeping fixed the number of atoms and increasing the number of cores. No ScaLAPACK were used in these tests. These runs were performed by using only MPI tasks. The following table summarises the time spent in each of the main functions of the code.

CORES	electrons	sum_bands	v_of_rho	newd	h_psi	s_psi	*diaghg	other
48	13616,83	4399,21	210,85	3532,4	3125,58	255,09	1113,64	980,06
96	9783	2663,58	136,42	1823,38	3252,4	136,86	1171,41	598,95
144	8264,01	2016,56	136,8	1133,45	3251	87,98	1123,54	514,68
192	5067,04	1134,44	154,71	760,17	1319,68	67,74	1124,5	505,8
288	3384,76	565,46	101,26	401,42	653,15	45,97	1122,11	495,39
384	3199,27	450,26	126,91	248,47	698,15	35,83	1119,27	520,38

Table 17: Time (seconds) spent in each of the main functions of the code.

The routines for PW are described here:

1. *electrons*: this routine is a driver of the self-consistent cycle. It uses the routine c_bands for computing the bands at fixed Hamiltonian, the routine sum_band to compute the charge density, the routine v_of_rho to compute the new potential and the routine mix_rho to mix input and output charge densities.
2. *sum_bands*: calculates the symmetrised charge density and sum of occupied eigenvalues
3. *v_of_rho*: this routine computes the Hartree and Exchange and Correlation potential and energies which corresponds to a given charge density. The XC potential is computed in real space, while the Hartree potential is computed in reciprocal space.
4. *newd:* it calculates the augmented density of the ultrasoft pseudo-potential.
5. *h_psi:* this routine computes the product of the Hamiltonian matrix with the wave-functions
6. *s_psi:* this routine applies the S matrix to m wave-functions psi
7. **diaghg*: this routine calculates eigenvalues and eigenvectors of the generalised problem Hv=eSv, with H symmetric matrix, S overlap matrix (*rdiaghg* is the version of this routine in double precision).

In the following plot, we show the relative time spent on each of the above-described subroutines. Each column refers to a different number of cores. Here it's possible to compare how the increasing of cores at fixed load (i.e., in a strong scaling regime) affects the performance of each of the functions.

[1] The extension to k-points is quite straightforward, being the calculation over k-points almost "embarassingly parallel"

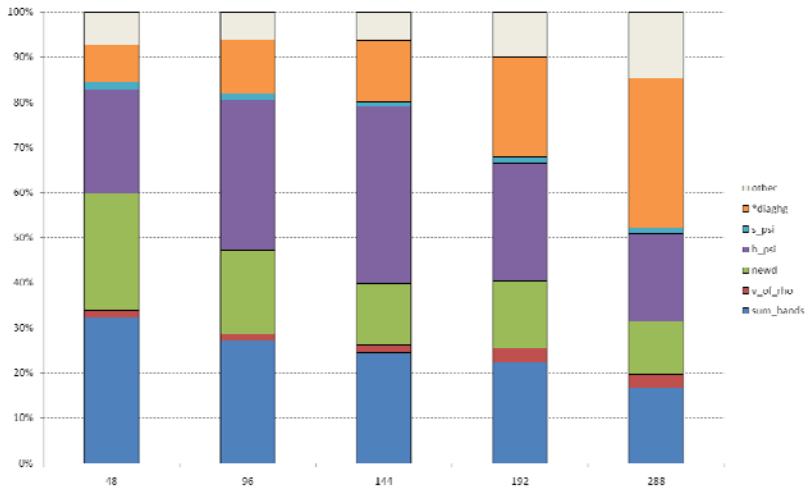


Figure 55: Relative time spent in the main code's subroutines.

From this plot it is quite evident that the function **diaghg* gains more weight when the number of cores rises. Without ScaLAPACK, the standard algorithm uses a serial LAPACK and every time the diagonalisation of the reduced Hamiltonian is performed.

In the following plot, we show absolute time values in seconds. In this way it is possible to better identify the overall scaling behaviour of the Quantum ESPRESSO for this Test Case.

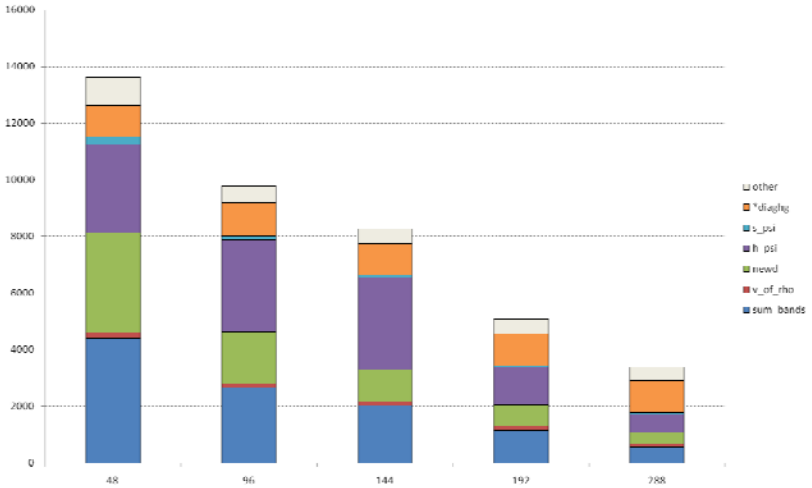


Figure 56: Absolute performances of the various sections of the code.

Test Case 2: 875 atoms GeSbTe (gamma point calculation)

We report this scientific case where a large number of atoms is used. In this case, to exploit all the necessary memory we used both MPI and OpenMP. More precisely, we used 128 MPI tasks and 6 threads per task, in order to fulfil a cluster node capability.

In the following plot and table, we report the absolute times for two cases.

MPIxOMP	sum_bands	v_of_rho	newd	h_psi	s_psi	*diaghg	*other*
64x6	255,15	2,42	98,81	476,96	118,82	568,05	300,79
128x6	128,31	2,81	38,82	269,03	62,24	529,25	208,54

Table 18: Time spent in each of the main functions of the code.

The scaling trend of the single functions and the overall performance are comparable to those obtained in the Test Case 1. Again, the most time consuming function is **diaghg*.

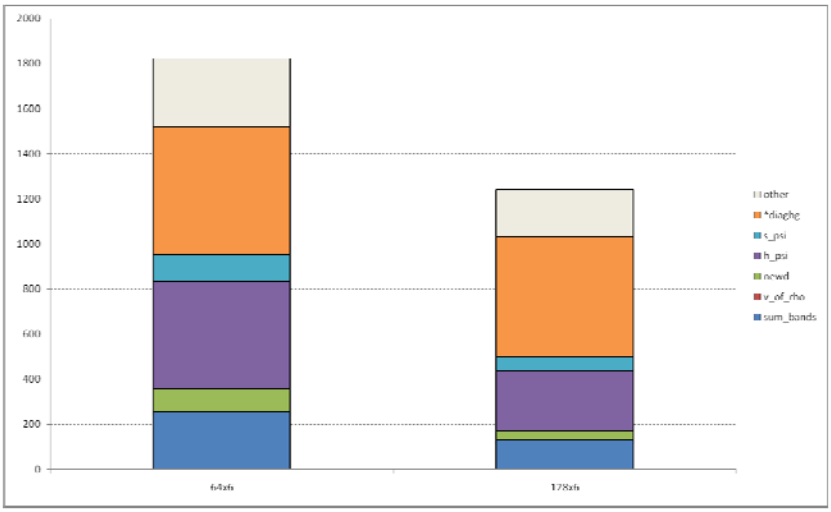


Figure 57: Distribution of time between the main functions in the two cases.

Test Case 3: nanodots of CdSe atoms

Differently from the previous two cases presented, we kept the number of cores fixed observing the performance when the number of atoms changes (*weak-scaling* analysis).

In the following table we report the relative times for the above described functions of two cases (275 and 489 atoms) using 64 MPI tasks with 6 OpenMP threads each.

atoms	h_psi	s_psi	rdiaghg	sum_band	vofrho	newd
275	34,98287	6,129886	6,271327	34,35897	0,495973	17,7609
489	34,81832	9,032691	13,74204	24,85920	0,215780	17,33195

Table 19: Time spent in the main code's subroutines.

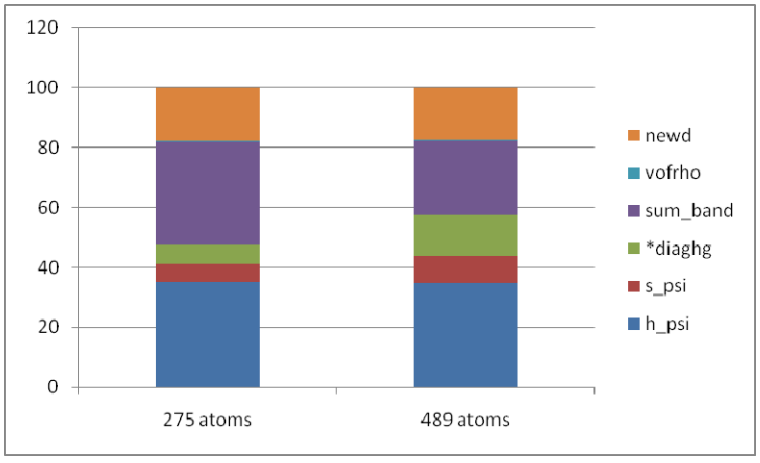


Figure 58: Distribution of time between the main functions in the two cases.

Again, from the stacked columns, one can notice that the most increasing contribution comes from the **diaghg* function, as observed in the previous Case Studies. At the same time, the function *sum_band* decreases his weight. The relative amount of the all other functions studied here does not change significantly.

5.2.3 Performances: CP.X – Car Parrinello MD

Test Case 4: nanodots of CdSe

For the Car Parrinello simulation we choose to use the same system used in the Test Case 3. Here, just like in that case, we keep fixed the number of cores (64 MPI tasks x 6 threads) and we increase the load, i.e., the number of atoms. (weak scaling).

In the following table, we report the data obtained for 4 different systems of different size.

atoms	rhoofr	vofrho	dforce	other	newd	Ortho	updatc
275	5,18	1,53	74,96	2,52	3,38	10,42	1,98
489	3,16	0,82	80,00	2,81	1,03	10,09	2,06
922	2,44	0,43	82,03	2,89	0,37	9,62	2,19
1214	1,82	0,36	82,57	3,02	0,21	9,88	2,09

Table 20: Time spent in the main code's subroutines.

Definition of subroutines

- *rhoofr*: it computes the electron density in real space
- *vofrho*: it computes the one-particle potential v in real space, the total energy and the forces acting on the ions
- *dforce*: it computes the generalised force acting on the i-th electron state at the gamma point of the Brillouin zone
- *newd*: it calculates the augmented density of the ultrasoft pseudo-potential
- *ortho*: it makes wave functions orthogonal
- *updatc*: updates the wave functions

Now if we look at the following plot we can notice that the compute intensive part is the *dforce* routine. This could be expected because in the dynamics this part is invoked many times to calculate the generalised force acting on all the atoms. Moreover, this function contains a FFT over all the bands and this is of great on the overall performance.

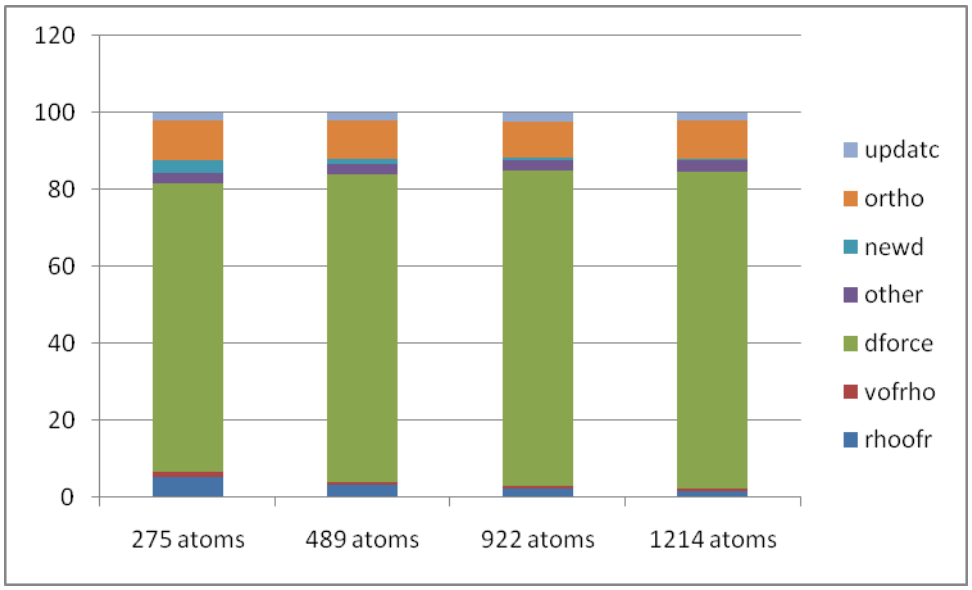


Figure 59: Relative time spent in the main code's subroutines.

If we compare the first two columns of the above plot with the corresponding plot of the Test Case 3, we can make a consideration on the two analogues routines: *ortho* and **diaghg*. We can observe that in the PW case the latter increases his weight when passing from 275 to 489 atoms. Differently, in the CP case the weight relative to the *ortho* routine keep constant or it slightly decreases. The trend, when the number of atoms increases, is that that in CP the prevalent part is for the FFT calculation, while on the PW is for the linear algebra (*GEMM and eigenvalues/eigenvectors solvers*).

In order to study the scaling at fixed number of atoms, we report the results obtained for a system of 922 atoms with 64 and 128 MPI tasks. In both cases, 6 OpenMP threads for each task were used.

	rhoofr	**vofrho**	**dforce**	**calphi**	**newd**	**ortho**	**updatc**
64task*6threads	361	63,9	12131	428	55,33	1424	325
128task*6threads	218,1	33,9	6097	220	57,06	771	245

Table 21: Time spent in the main code's subroutines.

If we plot the above data, we can see that all the routines, with the exception of the ortho routine, scale up almost linearly.

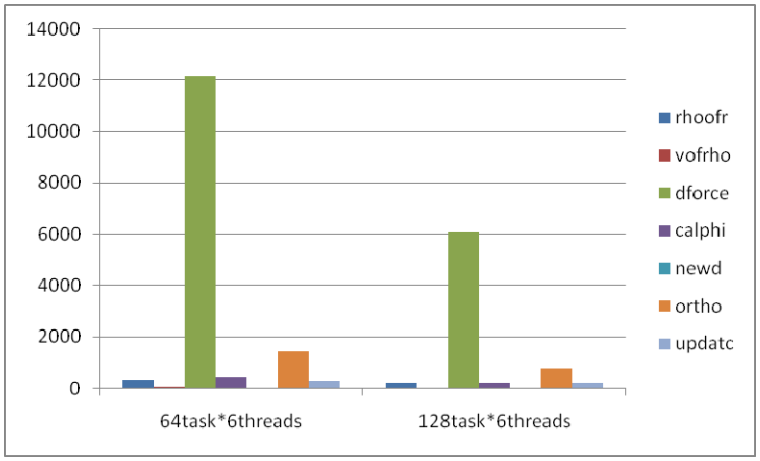


Figure 60: Absolute time spent in the main code's subroutines.

Finally, to profile the efficiency of the threading, we made a calculation by fixing the number of MPI tasks. In this case, we have 275 atoms, 64 MPI tasks, and we report the obtained timings for 1 thread and 6 threads.

	rhoofr	**vofrho**	**dforce**	**calphi**	**newd**	**ortho**	**updatc**
1 thread	152,24	21,05	1701	92,4	108,52	334,68	61,55
6 thread	32,59	9,68	471,43	15,88	21,29	65,55	12,46

Table 22: Time spent in the main code's subroutines.

The scaling factor is 3.6 for the *dforce* routine and 5.1 for the *ortho* routine.

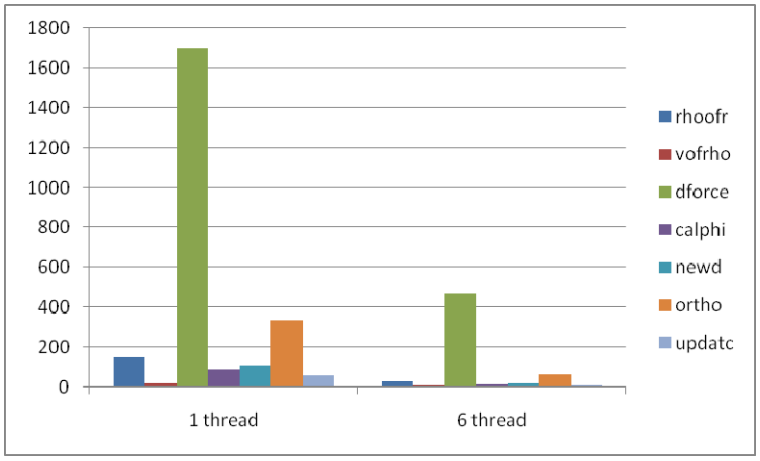


Figure 61: Absolute time spent in the main code's subroutines.

5.3 Yambo

5.3.1 Description of the code

Yambo is an *ab initio* code for calculating quasiparticle energies and optical properties of electronic systems within the framework of many-body perturbation theory (MBPT) and time-dependent density functional theory (TDDFT). Quasiparticle energies are calculated within the GW approximation for the self-energy. Optical properties are evaluated either by solving the Bethe–Salpeter equation or by using the adiabatic local density approximation.

The distinct feature of Yambo when compared with other electronic excitation codes is the accounting for excitonic effects (electron–hole interactions) through the solution of the Bethe–Salpeter equation for the electron–hole Green's function, within the MBPT framework. As an example, for wide bandgap insulators these effects are crucial as spectra calculated within non-interacting theories do not resemble experiment at all.

The excitations that can be calculated with Yambo are quasiparticles, excitons and plasmons. These excitations are ubiquitous in the *ab initio* description of the electronic and optical properties of any physical system. The Yambo code uses as input the result of standard DFT calculations obtained with other codes, such as abinit and Quantum ESPRESSO. This input enables Yambo to build the non-interacting Green's function. To obtain the exact Green's function, one has then to solve the Dyson equation. This is a very complicated equation and Yambo resorts to a widely used approximation in order to solve it: the GW approximation.

The key ingredient for the excitation calculations is the density response. Within the GW approximation, obtaining the density response amounts to building the non-interacting response function and then inverting it. As Yambo is interfaced with plane-wave codes, the non-interacting response function is written as a matrix in reciprocal space, transforming the key problem of excitation calculations to a matrix inversion procedure. TDDFT and the Bethe-Salpeter equations can be used afterwards to obtain the optical response functions.

Yambo is a plane-wave code that, although particularly suited for calculations of periodic bulk systems, has been applied to a large variety of physical systems. Yambo relies on efficient numerical techniques devised to treat systems with reduced dimensionality, or with a large number of degrees of freedom. The code has a user-friendly command line based interface and a flexible I/O procedure. It requires BLAS and LAPACK libraries and can use some other libraries: MPI, BLACS, SCALAPACK, FFTW [79] and netCDF. The GPL version of yambo can be obtained at http://www.yambo-code.org/.

5.3.2 Test cases

The problem used to profile Yambo was a 64 atoms slab (16 layers) of Si, periodically repeated. The goal was to obtain the optical response function of Si(100) c(2x4) surface. In this calculation, 256 electrons were explicitly considered (128 occupied states) and the non-interacting density response function was a 3000x3000 matrix.

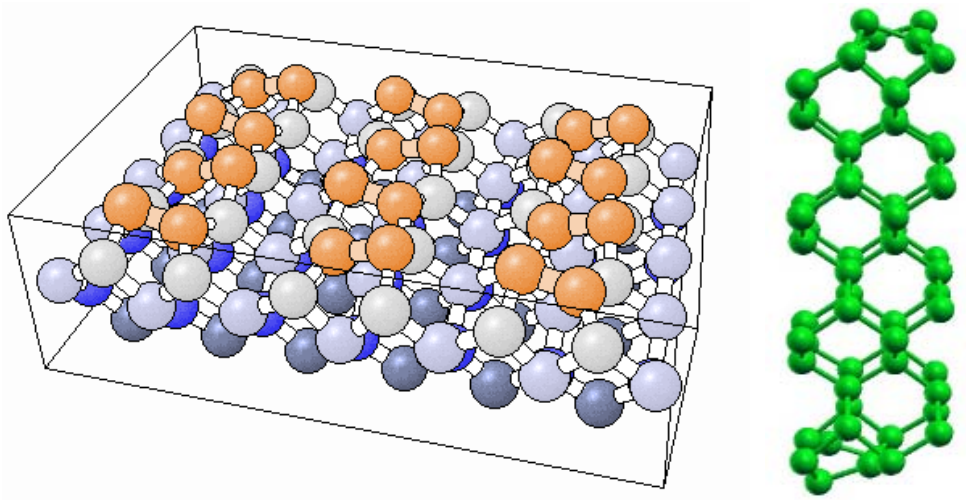


Figure 62: The test system: Si(100) c(2x4) surface (left) and the 64 atoms slab used to represent it (right).

Performance data

Test runs were performed in Barcelona's Supercomputing Centre MareNostrum computer. For these runs, the code did not use SCALAPACK. The scaling is hindered by the matrix inversion step, as can be seen in Figure 63. The other key step in the run is the setup of the non-interacting matrix, but this a trivially parallel procedure, as every node in the run can independently calculate some matrix elements.

The use of SCALAPACK does not seem to bring any scaling improvements, although some runtimes are smaller than when using standard LAPACK (see Figure 64). In the case shown in Figs. 3 and 4, a slightly different system was used: a 16 layers slab of Si(100) c(4x4), again with 128 occupied bands, but with a different number of k-points (20 instead of 42). The non-interacting response matrix size was 9000x9000. The runs were performed at the IBM SP6 of CINECA in SMT mode. Figure 65 shows the speedup of the two main tasks of Yambo for this second test case. At least for this dataset and this (small) number of CPUs, the scaling of the matrix set up is almost ideal.

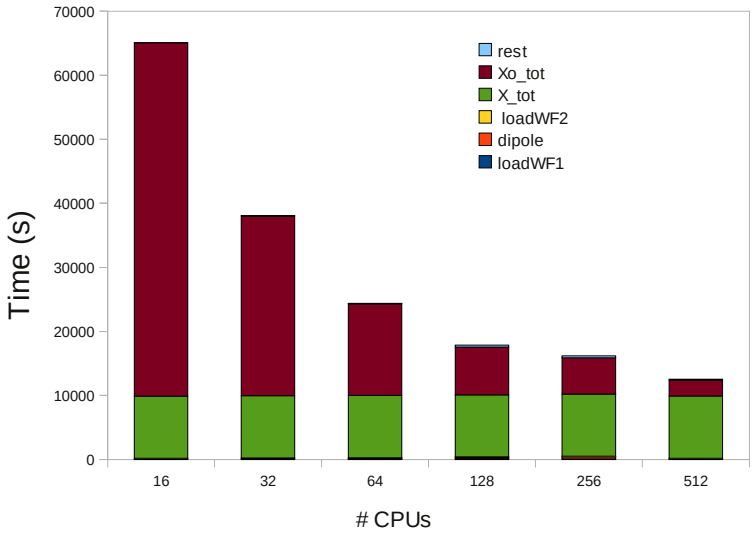


Figure 63: Scaling analysis of the Si 64-atoms slab run. Xo_tot is the matrix setup step that is very well distributed among the nodes. X_tot is the matrix inversion step that does not show any sign of parallelism. Other steps in the calculation are unimportant.

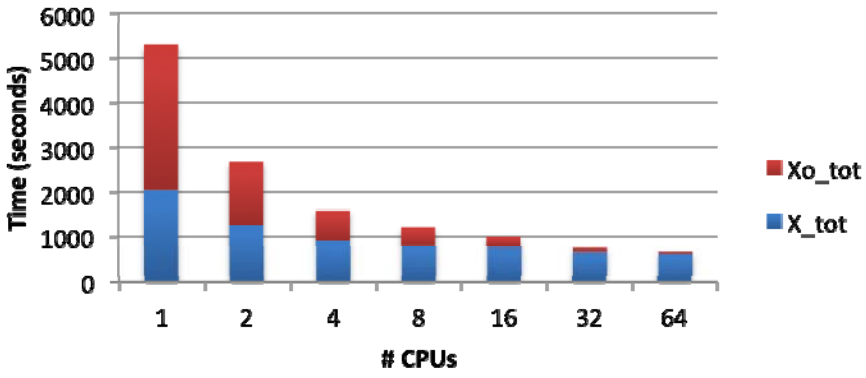


Figure 64: Scaling analysis of a run that uses SCALAPACK. Inversion step remains essentially non-parallelised.

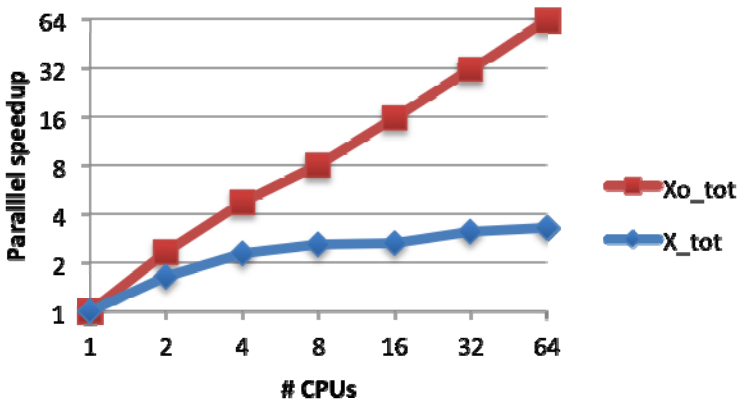


Figure 65: Same as previous figure, but showing parallel speedup instead of computing time.

5.4 Siesta

5.4.1 Description of the code

SIESTA (Spanish Initiative for Electronic Simulations with Thousands of Atoms) is both a method and its implementation as a computer program, to perform electronic structure calculations and ab-initio molecular dynamics simulations of molecules and solids. Its main characteristics are:

- It uses the standard Kohn-Sham self-consistent density functional method in the local density (LDA-LSD) or generalised gradient (GGA) approximations.
- It uses norm-conserving pseudo-potentials in their fully nonlocal (Kleinman-Bylander) form.
- It uses atomic orbitals as a basis set, allowing unlimited multiple-zeta and angular momenta, polarisation and o-site orbitals. The radial shape of every orbital is numerical and any shape can be used and provided by the user, with the only condition that it has to be of nite support, i.e., it has to be strictly zero beyond a user-provided distance from the corresponding nucleus. Finite-support basis sets are the key for calculating the Hamiltonian and overlap matrices in O(N) operations.
- Projects the electron wave-functions and density onto a real-space grid in order to calculate the Hartree and exchange-correlation potentials and their matrix elements.
- Besides the standard Rayleigh-Ritz eigenstate method, it allows the use of localised linear combinations of the occupied orbitals (valence-bond or Wannier-like functions), making the computer time and memory scale linearly with the number of atoms. Simulations with several hundred atoms are feasible with modest workstations.
- It is written in Fortran 95 and memory is allocated dynamically.
- It may be compiled for serial or parallel execution (under MPI).

It routinely provides:

- Total and partial energies
- Atomic forces
- Stress tensor
- Electric dipole moment
- Atomic, orbital and bond populations (Mulliken)
- Electron density

And also (though not all options can be used together):

- Geometry relaxation, xed or variable cell
- Constant-temperature molecular dynamics (Nose thermostat)
- Variable cell dynamics (Parrinello-Rahman)
- Spin polarised calculations (collinear or not)
- k-sampling of the Brillouin zone
- Local and orbital-projected density of states
- COOP and COHP curves for chemical bonding analysis
- Dielectric polarisation
- Vibrations (phonons)
- Band structure
- Ballistic electron transport (through TranSiesta)

Starting from version 3.0, SIESTA includes the TranSiesta module. TranSiesta provides the ability to model open-boundary systems where ballistic electron transport is taking place. Using TranSiesta one can compute electronic transport properties, such as the zero bias

conductance and the I-V characteristic, of a nanoscale system in contact with two electrodes at different electrochemical potentials. The method is based on using non-equilibrium Green's functions (NEGF), that are constructed using the density functional theory Hamiltonian obtained from a given electron density. A new density is computed using the NEGF formalism, which closes the DFT-NEGF self-consistent cycle.

5.4.2 Implementation details concerning performance

SIESTA uses a set of strictly localised pseudo-atomic orbitals as basis (non-orthogonal). Charge densities and potentials are represented on a real-space grid, and the Poison equation is currently solved using an FFT algorithm. The performance depends on the following parameters:

1. The cardinality of the basis (*norbs*). This is the total number of basis orbitals in the unit cell. In diagonalisation mode, SIESTA currently builds dense Hamiltonian (H) and overlaps (S) matrices, and solves the generalised eigenvalue problem. At the moment it uses ScaLAPACK, distributing the matrices in 2D block-cyclic form.
2. If k-points are used, this procedure is repeated for each k-point, building H(k) and S(k) with the appropriate phases. There is an option to parallelise over k-points instead of over orbitals, useful for smaller systems. (Work is ongoing to have both levels of parallelisation at the same time).
3. The degree of sparsity of the H and S matrices. This depends on the spatial extent of the orbitals ("big" orbitals will overlap with more neighbours). The number of non-zero matrix elements of H and S (*nnz*) is a key parameter for the workload involved in setting up H and S. Due to the locality of the basis set, this setup is approximately O(N) in the size of the system (except for a very small O(NlogN) term due to the FFT needed to set up the potential from the charge density).
4. The degree of sparsity will be large for large systems, while for small systems most pairs of orbitals will have non-zero matrix elements. As an example, we show tables for a DNA fragment (approx. 750 atoms) with different basis sizes. The number of orbitals (*norbs*), the number of non zeros (*nnz*) and the sparsity are presented. The size of the orbitals can be parameterised in SIESTA by indicating the "energy shift" involved in confining the free atom (the larger the shift the more confined, and the smaller the orbital radii).

	norbs	*nnz*	sparsity
OUT.sz.100:	2218	75462	1,53%
OUT.sz.200:	2218	61954	1,26%
OUT.sz.300:	2218	54648	1,11%
OUT.sz.400:	2218	49062	1,00%
OUT.sz.500:	2218	44680	0,91%

Double-zeta plus polarisation basis set:

	norbs	*nnz*	sparsity
OUT.dzp.100:	7702	771951	1,30%
OUT.dzp.200:	7702	640884	1,08%
OUT.dzp.300:	7702	567209	0,96%
OUT.dzp.400:	7702	510423	0,86%
OUT.dzp.500:	7702	463914	0,78%

For this large system we get about 1% of non-zero entries in H and S. This DNA example was used in 1998 to demonstrate the capabilities of the linear-scaling solver in SIESTA: the calculation was done in a desktop workstation. In diagonalisation mode the nearly 8000 orbitals for the DZP basis represent a big computational load. Work is underway at BSC to produce an eigensolver that exploits sparsity. It would be a major breakthrough.

5. The fineness of the real-space grid. The grid point separation is expressed as a MeshCutoff in units of energy (resembling the use of an energy cutoff in plane-wave codes - in SIESTA, this cutoff is the "density cutoff" and not the "wave-function cutoff"). The major operations involved are the computation of the charge density from the one-particle density matrix and the orbital info (*rhoofd*), the calculation of the potential from the charge density (*poison*), the calculation of the exchange-correlation energy and potential (*cellxc*), and the computation of the contribution of the potential to the Hamiltonian matrix elements (*vmat*). The *rhoofd* and *vmat* parts actually couple the grid to the orbitals.

A given SIESTA calculation might be dominated by the setting up of H and S (an operation collectively known in the code as *DHSCF*) or by diagonalisation (*diagon*). Since *diagon* scales as O(N3) with the size of the system, and *DHSCF* approximately as O(N), it is clear that systems with a large number of atoms are dominated by diagonalisation. (In some cases, such as large QM-MM systems with smallish quantum parts but large unit cells, the *DHSCF* part will dominate).

The *diagon* part consists of the actual diagonalisation routines *cdiag* (complex valued) or *rdiag* (real valued), which use ScaLAPACK routines, and the computation of the density matrix in *c-buildD* and *r-buildD*.

5.4.3 The tests

Performance analysis of three examples, taken from two different physical problems, is presented. All data was gathered with SIESTA Version 3.0-b-11 running on MareNostrum, which is based on PowerPC 970 processors, Myrinet network, and Suse Linux. Each of its 2560 nodes features four cores and 8 GB of memory.

The first example, called **CNT transport**, is based on the examination of electronic transport through carbon nanotubes covalently bound to graphene layers for different geometric configurations, presented in [15]. These evaluations were done with TranSiesta, which builds on SIESTA ground state calculations. The data presented here describes the performance of the SCF calculation for one of the configurations.

For examining the influence of the number of atoms, a second system twice as big was created by using two unit cells per supercell. To point out this influence, the results of both systems are discussed in parallel.

The second physical system is given by a part of a DNA strand, solved using diagonalisation. The system size is similar to the smaller CNT transport example, but due to a higher number of orbitals the resulting matrix size is even larger than in the scaled CNT problem.

These systems differ in the number of k-points calculated – eight for CNT and one for DNA. When calculating only the gamma-point, as in the DNA example, SIESTA uses real arithmetic instead of complex. Due to the selection of examples both modes are represented here, but some direct comparisons, e.g. total running time, are not reasonable.

Problem	Atoms	*n*	*nnz*
CNT transport 1	780	3048	~ $5*10^5$
CNT transport 2	1560	6096	~ 10^6
DNA	776	7752	~ $2.4*10^6$

Table 23: Parameters describing the systems examined.

5.4.3.1 CNT-Transport Test

Scalability

The following diagrams show the speedup for the most time-consuming code sections and for the total ground state calculation.

The total speedup apparently depends mainly on the scaling of the diagonalisation part, whose efficiency rises with the system size. For the small system the speedup quickly saturates at around 30, whereas with the big system a total speedup of about 60 can be reached on 256 processes.

The scaling of *DHSCF* and *c-buildD* is less correlated to the system size, but never exceeds a value of 20. The matrix construction speedup even decreases beyond 128 processes.

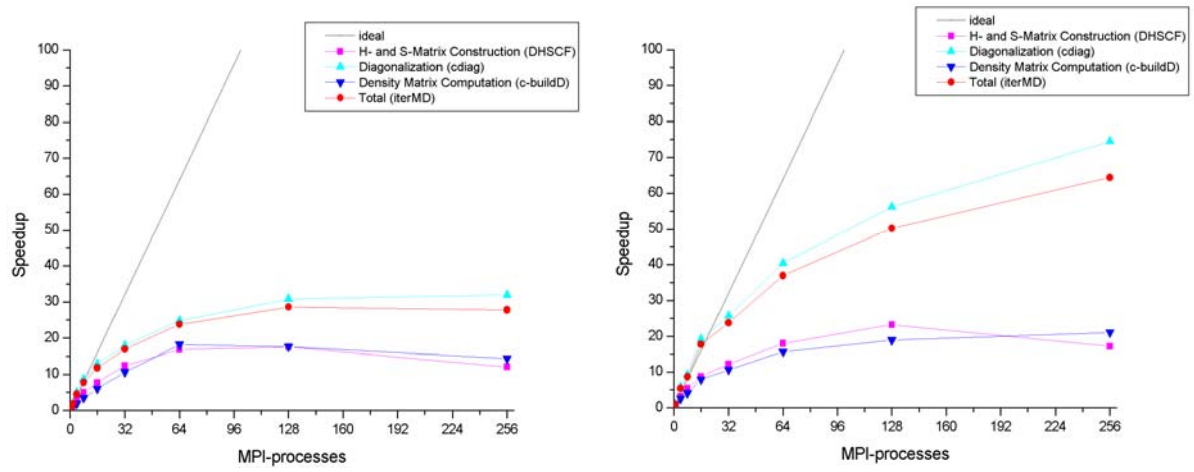


Figure 66: Speedup graphs for the CNT transport examples with one (left) and two (right) unit cells

Time Distribution

In order to visualize the importance of the different code sections, their relative amount of

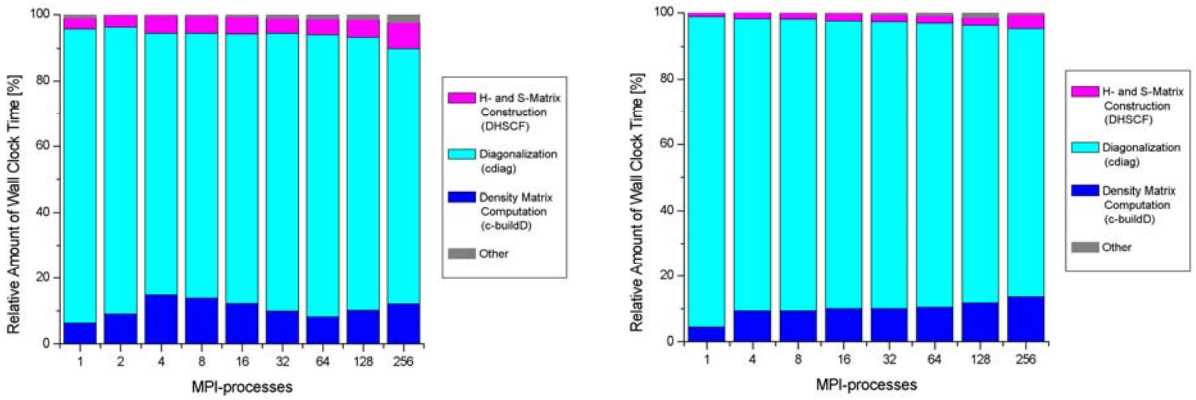


Figure 67: Relative amount of time spent in the most costly functions depending on the number of processes. The left image shows the results for the small, the right for the big example.

wall clock time is shown.

As expected from the scalability results, the diagonalisation is always the by far most expensive step, taking up to over 90% of the total effort. For the small problem there is no clear dependence on the number of processors observable, whereas in the other case the time fraction of *DHSCF* and *c-buildD* rises with the number of MPI-processes due to the bad scaling of these functions.

Total time

The times for solving the systems are as follows:

Processes	1	*4*	*16*	*64*	*256*
CNT transport 1	44645	9892	3777	1878	1608
CNT transport 2	302672	55430	17015	8190	4698

Table 24: Total wall clock time in seconds for different numbers of processes.

5.4.3.2 DNA Test

Scalability

The speedup graph clearly shows that *DHSCF* and *r-buildD* scale very bad. The diagonalisation scales much better, but beyond 128 processes its efficiency drops. In total the speedup converges to about 70.

Compared to the CNT data the diagonalisation scales better, but has less influence on the total speedup.

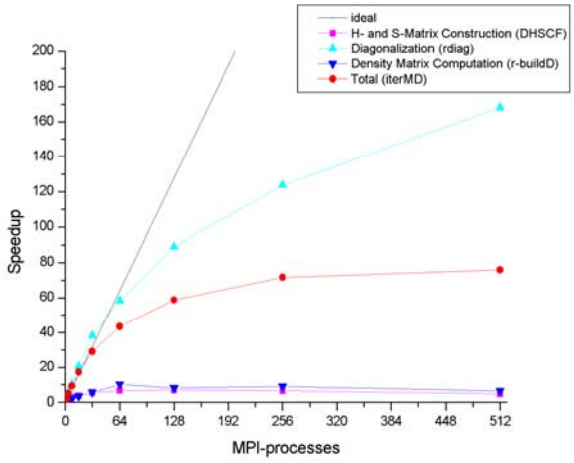


Figure 68: Speedup graph for the DNA example

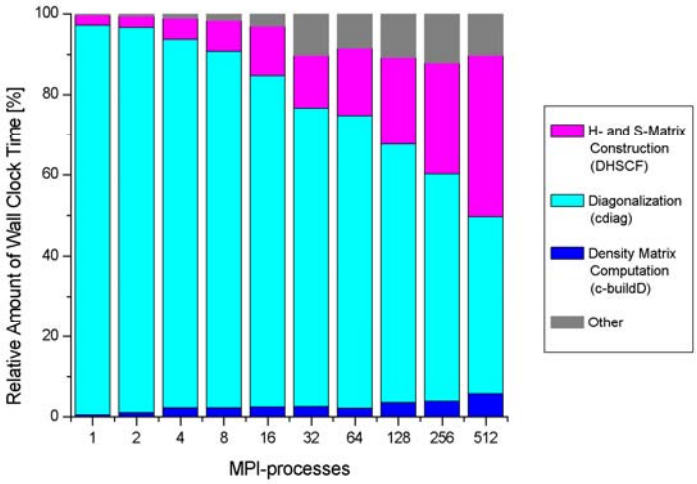


Figure 69: Relative amount of time spent in the most costly functions depending on the number of processes.

Similar to the CNT examples the diagonalisation is the most costly part, but due to its lack of scalability *DHSCF* takes over a good part of the computation time when increasing the number of processes.

Total time

The total running time for different numbers of processes is given below. The times are smaller compared to the CNT results because only one k-point is calculated, real arithmetic is used, and the number of iterations in the SCF loop is much smaller.

Processes	**1**	***4***	***16***	***64***	***256***
DNA	33411	6338	1916	762	463

Table 25: Total wall clock time in seconds for different numbers of processes.

5.4.4 Conclusions

In all cases the diagonalisation routine is the most costly part. It uses ScaLAPACK, which also has the disadvantage of working with dense matrices, so an unnecessary high amount of memory is needed by not taking advantage of sparseness.

As the DNA example shows, also the construction of the Hamiltonian and overlap matrices can take a good fraction of the computation time for certain types of problems when using many processes. The scalability of this part has already been improved by optimising the load balance and tests show much better behaviour, but this code was not implemented in the SIESTA version used for the analysis.

5.5 Octopus

5.5.1 Description of the code

Octopus is a computer code to calculate excitations of electronic systems, i.e., to simulate the dynamics of electrons and nuclei under the influence of external time-dependent fields. The code relies on Density Functional Theory (DFT) to accurately describe the electronic structure of finite 1-, 2- and 3-dimensional systems, like e.g. quantum dots, molecules and clusters. The initial implementation relied on finite-system boundary conditions, but the most recent version also enables the user to specify periodic boundary conditions, opening the door to the simulation of 1D, 2D and 3D infinite systems.

Although it is used to calculate the DFT ground-state of electronic systems, Octopus was written mainly for the calculation of electronic excitations, using the Time-Dependent formulation of DFT (TDDFT). The response properties of electronic systems can be calculated using either perturbative techniques (Casida's or Sternheimer's equations) or real-time propagation of the Kohn-Sham wavefunctions when subject to an external perturbation. There is also the possibility of allowing the ions to move (classically) and thus perform a very realistic simulation of the ionic and electronic dynamics of, e.g., a molecule under the influence of a time-dependent perturbation.

In Octopus the functions are represented in a real space grid. The differential operators are approximated by high-order finite differences. The propagation of the time-dependent Kohn-Sham equation is done approximating the exponential of the Hamiltonian operator by a Taylor expansion.

The code is released under the GNU Public License and is freely available at: http://www.tddft.org/. The code is mainly written in FORTRAN 90, but it contains some parts written in C and it also relies on several Perl scripts. The code consists of approximately 120k lines of FORTRAN 90 code and 20k lines of C code.

Besides an MPI library, the code requires some standard external libraries: FFTW, BLAS, LAPACK, and GSL [80]. Other libraries are also required, but they are currently bundled with the code. Optional libraries include BLACS and SCALAPACK for parallel linear algebra, netCDF and ETSF_IO for ETSF standardised output, and PFFT or LIBFM for the solution of Poisson's equation. Octopus uses auto-tools to ease the compilation process and is thus very easy to compile for different architectures/systems.

Octopus plain ASCII input file is parsed by an engine that allows for the use of variables. It also automatically assumes default values for all input parameters that are not explicitly assigned a value in the input file. The output is plain text for summary information, and platform-independent binary for wave functions.

5.5.2 Test cases

The typical problem used to profile Octopus was the optical response of chlorophyll complexes of different sizes. A realistic simulation of this system would involve at least a complex with 5879 atoms, but calculations did not converge for systems larger than 1365 atoms. The 650-atom chlorophyll complex is shown in Figure 70. This system has 650 atoms

and 1654 electrons. It was simulated considering 827 occupied states and 53 unoccupied states. The grid had 1144073 points.

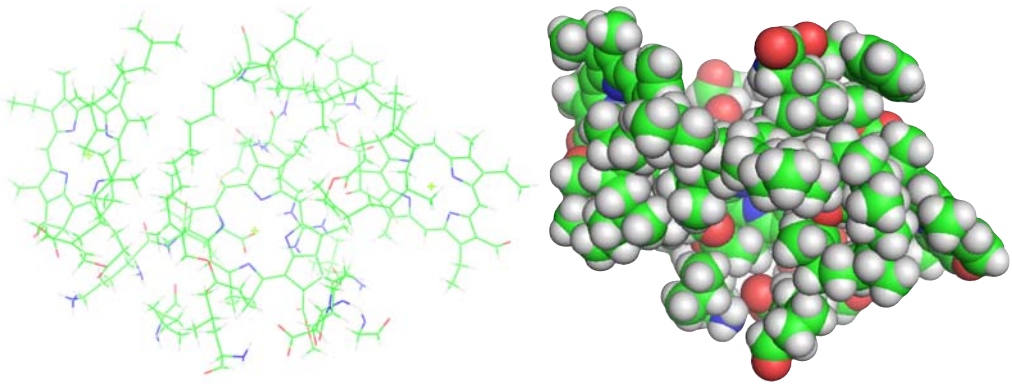


Figure 70: 650-atom chlorophyll complex represented in two different ways.

Parallelisation strategies in the code

Octopus has a multilevel parallelisation: states parallelisation, where the processors are divided in groups, and each one gets assigned a number of orbitals; and domain parallelisation, where real space is divided in domains assigned to different processors. When using domain parallelisation, the application of differential operators requires that the boundary regions be communicated. This is done asynchronously, overlapping computation and communication. For periodic systems, k-point parallelisation is also available. Besides these parallelisation strategies, each process can run several OpenMP threads and use OpenCL if a GPU is available (see Figure 71).

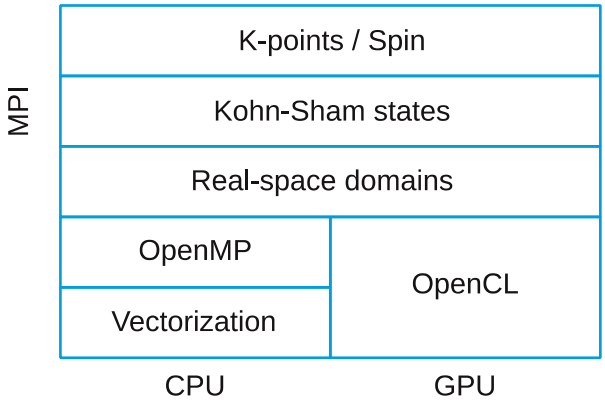


Figure 71: Scheme of the multi-level parallelisation of Octopus. The main parallelisation levels are based on MPI and include state- and domain-parallelisation. For a limited type of systems, additionally K-point or spin parallelisation can be used. In-node parallelisation can be done using OpenMP threads and hand-vectorisation using compiler directives, or by using OpenCL parallelisation for GPUs and accelerator boards.

Ground-state and time-propagation runs present different scaling challenges. In a time-dependent simulation with state parallelisation there is minimal communication, but in the ground state mode this parallelisation strategy requires a considerable amount of communication due to the orthogonalisation and subspace diagonalisation procedures that mix different states. Not using this parallelisation forces the ground-state runs to be only parallel in domains, and this puts a limit to the number of processors involved in ground-state calculations, because small domains imply a large domain surface to volume ratio, and this incurs in a communication overhead, as there are too many boundary points. This limitation is easily seen in Figure 72.

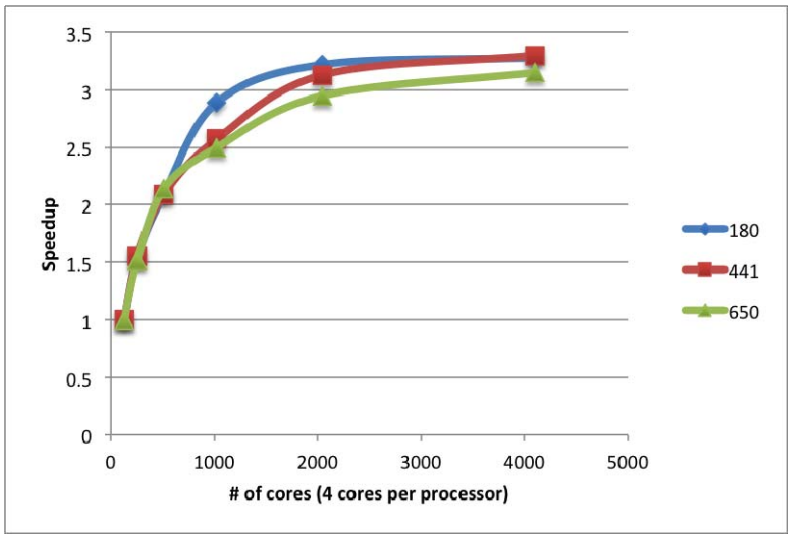


Figure 72: Parallel speedup of a ground-state calculation for 3 different chlorophyll complexes, with 180, 441 and 650 atoms, run on Jugene.

In order to circumvent this problem, state parallelisation was recently implemented in the ground-state mode of Octopus, based on the ScaLAPACK library. This parallelisation has not been profiled and tuned yet. Given that the main objective of using Octopus is performing TDDFT simulations, the ground-state part of the calculation represents a small part of the total computational time. In fact, the most important objective of the use of ScaLAPACK is to have the capability of simulating large systems with a modest amount of memory per node, and this applies to both ground-state and time-dependent runs.

Time propagation runs present very different scaling problems. As the propagation of each orbital is almost independent, the partitioning of the orbitals among processor groups is extremely efficient. This scaling strategy is in principle only limited by the number of available states, which increases linearly with the system size. Domain parallelisation is limited by the number of points in the grid that as in the previous case, also increases linearly with the system size. When both parallelisation strategies are used, and as the number N of processors increases, the number of processors dedicated to each strategy is proportional to the square root of N. There are, however, tasks like the calculation of the Kohn-Sham potential, that do not depend on the number of electrons and that are not affected by the state-parallelisation. Hence their computational time is only reduced by the domain parallelisation, so they will have a scaling proportional to the square root of N. Based on this idea, Amdahl's law can be generalised to the case of Octopus by dividing the computational time in three parts: a fully parallelizable one, P, a fully serial one, S, and the rest that is partially parallelizable and scales with the square root of the number of processors. This one corresponds to tasks that can only be divided in domains. For this division in three parts the parallel speed-up takes the form:

Figure 73 shows scaling data obtained on Jugene for the 1365-atom chlorophyll complex. The scaling model fits very nicely to the data, with P=97.930% and S=0.0058%, hence the part that scales with the square root of N corresponds to about 2% of the serial execution time. An Amdahl's law fit of the data yields a parallelisable fraction P=99.995%, however, the curve does not properly reproduce Octopus speed-up for large number of cores.

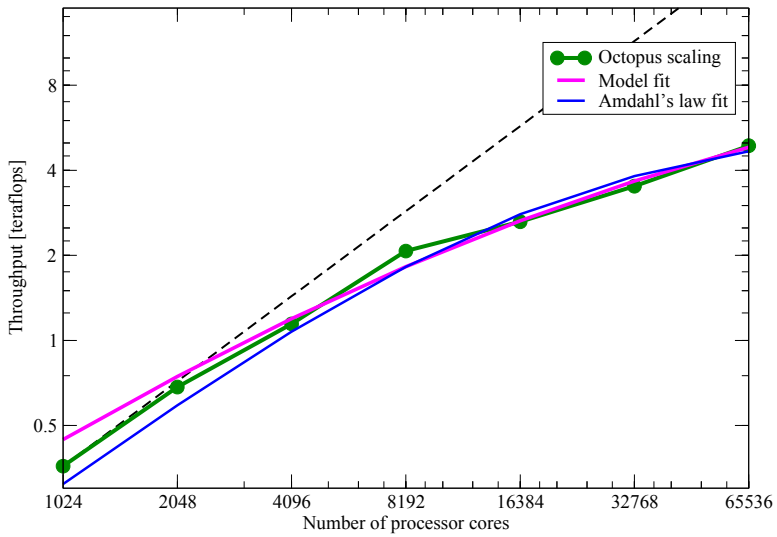


Figure 73: Parallel speedup of a real-time propagation run for a 1365 chlorophyll complex on Jugene.

These curves show that, although the vast majority of the code seems to be fully parallel, there is something hindering its performance, as the parallel efficiency drops markedly with the number of processors.

This point can be studied deeper. In a time propagation run, the evolution of the states is completely independent, but at each time-step it is necessary to recompute the Hamiltonian. The Hartree potential is therefore recomputed at each step and this computation is performed obtaining an updated electronic density (built with updated orbitals) and then solving Poisson's equation. In Figure 74 and Figure 75 the time taken by each block of routines is plotted against the number of nodes. It is apparent that most of the routines scale reasonably well, but the Poisson solver doesn't. For small numbers of processors its share of the total runtime is negligible, but for large numbers of processors it takes almost 50% of the total step time. This corresponds to the part of limited scaling in our model.

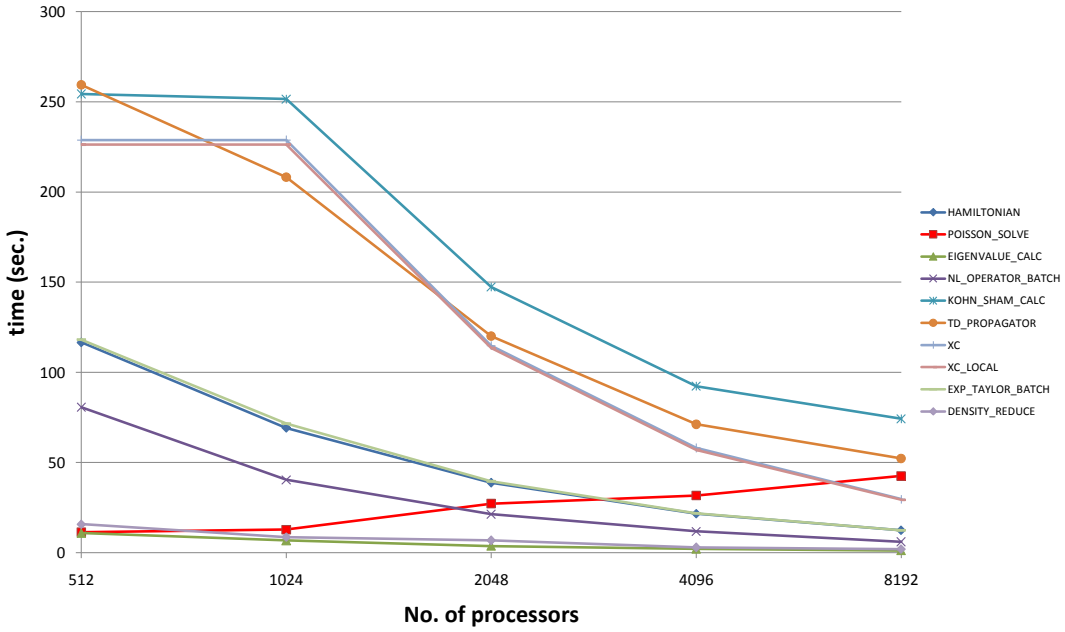


Figure 74: Cumulative times, on Jugene, of a time propagation run for the 1365-atom chlorophyll complex.

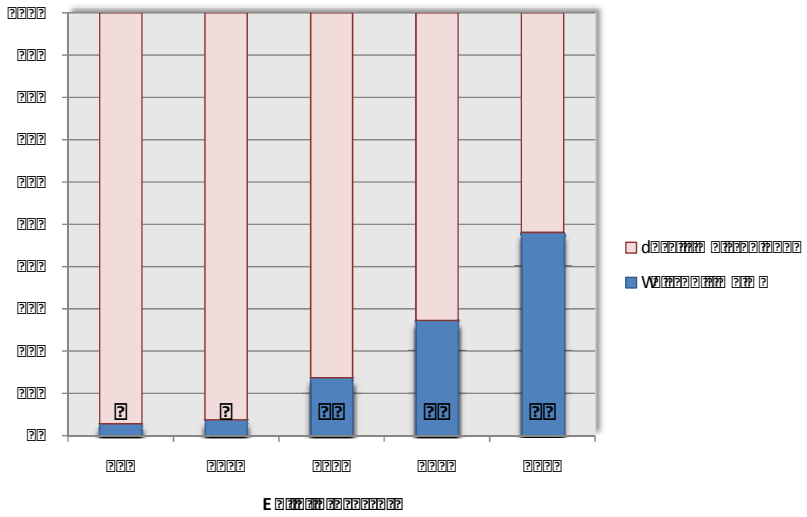


Figure 75: Percentage of time taken by each TDDFT propagation step, on Jugene, for the 1365-atom chlorophyll complex.

5.6 Exciting / ELK

5.6.1 Description of the code

The EXCITING FP-LAPW Code was an open-source full-potential linearized augmented-plane wave (FP-LAPW) code originally developed at Karl-Franzens-Universität Graz as a milestone of the EXC!TiNG EU Research and Training Network [52]. In 2009 the code development split into two branches: Exciting [53] which is particularly focused on excited state properties, within the framework of time-dependent DFT (TDDFT) as well as within many-body perturbation theory (MBPT) and Elk [54], which is mainly focused on the ground-state properties with some experimental implementation of excited state properties (within the TDDFT framework).

Both Exiting/Elk are Fortran90 MPI+OpenMP hybrid codes, freely available under the terms of GNU General Public License.

Main features:

Though Exiting/Elk codes are diverging, they still bear the core features of the original Exciting code:

- high precision all-electron DFT code based on the FP-LAPW method including local-orbitals – no approximation is made on the form of potential, charge density and wave-functions
- most general treatment of the magnetism (collinear, non-collinear, spin-orbit, spin-spirals) is implemented using the “second-variational” technique
- calculation of forces and structural optimization
- “beyond DFT” calculations including L(S)DA+U, exact exchange (EXX) and Hartree-Fock (HF)
- The following important features were added to the both codes after splitting:
- time-dependent density functional theory (TDDFT) for the linear response calculations (for finite momentum transfer q-vectors)
- Bethe-Salpeter equation (BSE) for the optical response

5.6.2 Performance analysis

Exciting/Elk codes perform many tasks. The most important among them is the ground state total energy calculation using the Kohn-Sham iterative minimization procedure. Ground state calculation involves three major steps:

i) Construction of the dense Hamiltonian and overlap matrices in the LAPW basis (the basis size is ~100 functions/atom and construction time scales as ~O(N^3), where N is the number of atoms) with the consequent call to the Lapack generalized eigenvalue solver (~O(N^3) scaling).
ii) Construction and symmetrization of the charge density (~O(N^3) scaling).
iii) Construction of the new potential (~O(N2log(N)) scaling). The obtained Kohn-Sham eigen-values and eigen-functions are then used in various tasks as an input data.

Performance tests:

Below we present the ground-state profiling of the Elk-1.3.31 code (obtained at http://elk.sourceforge.net/), which is representative of the EXCITING code as well. Test platform is Cray XT5 at CSCS with the threaded BLAS/LAPACK libraries provided by Cray's LibSci. Default PGI compiler (with the options '-O3 -fast -fastsse -Munroll -mp=nonuma') is used. The tests are done with the default set of LAPW parameters and species files. MT spheres are scaled up to a maximally allowed radii (autormt=.true.). Default LSDA functional is used. Number of empty bands is 50. Number of ground-state iterations is 6. Restart from STATE.OUT is done to skip the initial charge-density setup.

Test case 1: Antiferromagnetic $La_4Cu_2O_8$ (21 irreducible k-points)

The test is done with 1,7,11 and 21 single-threaded MPI tasks and with 21 six-threaded MPI tasks.

	Time (sec)									
MPIxOMP	Total	allatoms	ground state	one iteration	seceqnfv (total)	seceqnfv (solve)	seceqnfv (setup)	seceqnsv	rhomag	other
1x1	3843.103	72.885	3770.219	628.370	1521.967	283.955	1238.012	1307.096	852.597	88.559
7x1	1187.275	65.587	1121.688	186.948	553.424	90.789	462.635	218.680	215.838	133.746
11x1	851.831	65.150	786.681	131.114	345.103	55.031	290.072	142.662	158.198	140.719
21x1	529.882	72.813	457.069	76.178	190.862	31.646	159.216	76.106	95.511	94.590
21x6	510.286	123.493	386.794	64.466	73.403	10.956	62.447	63.006	70.394	179.991

Table 26: Aggregative time of the various parts of the Elk code for the 6 iterations of the ground state run for La4Cu2O8.

Definition of the columns:

- *Total* – total time (initialization + 6 ground-state iterations)
- *allatoms* – search the initial core energies; this is done once during the initialization
- *ground state* – difference between Total and allatoms; this is the pure time for 6 iterations
- *one iteration* – time for one iteration (ground state divided by 6)
- *seceqnfv* (total) – total (setup + diagonalization) time for the first-variational eigen-value equation
- *seceqnfv* (solve) – generalized eigen-value problem for the first-variational states (done in reals because of the existing inversion symmetry)
- *seceqnfv* (setup) – setup of the LAPW Hamiltonian and overlap matrices
- *seceqnsv* – total (setup + diagonalization) time for the second-variational eigen-value equation (most of the time is spent in the setup)
- *rhomag* – generation of charge density and magnetization

- *other* – remaining stuff (MPI sync, OMP threads sync, density symmetrization, effective potential, mixing, linearization)
-

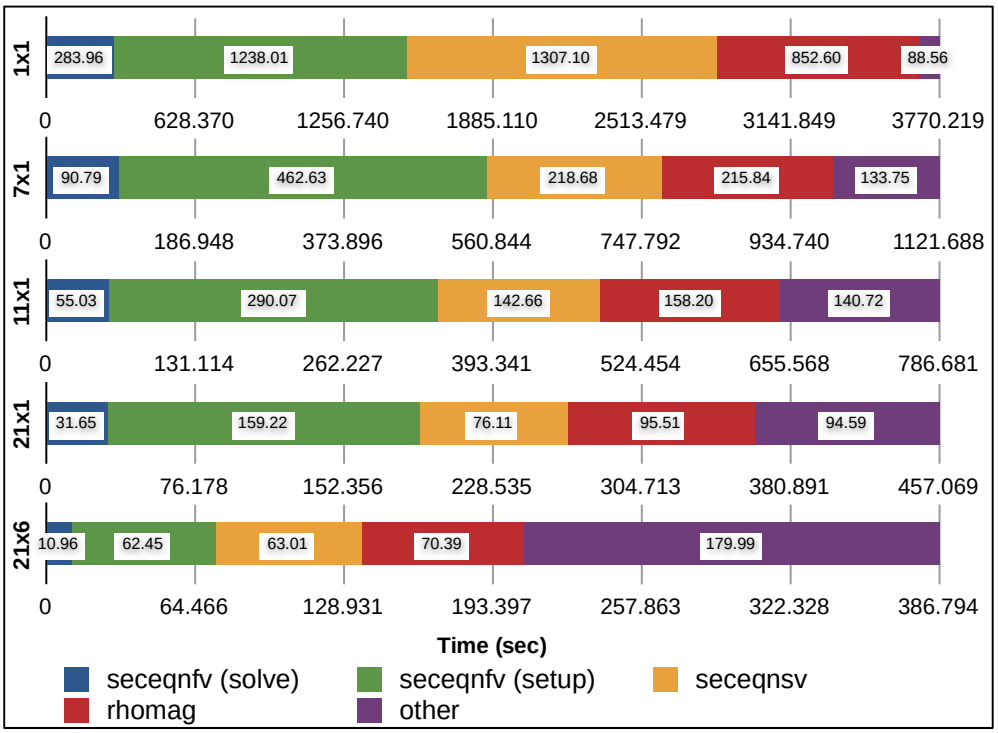


Figure 76: Graphical representation of results shown in Table 26

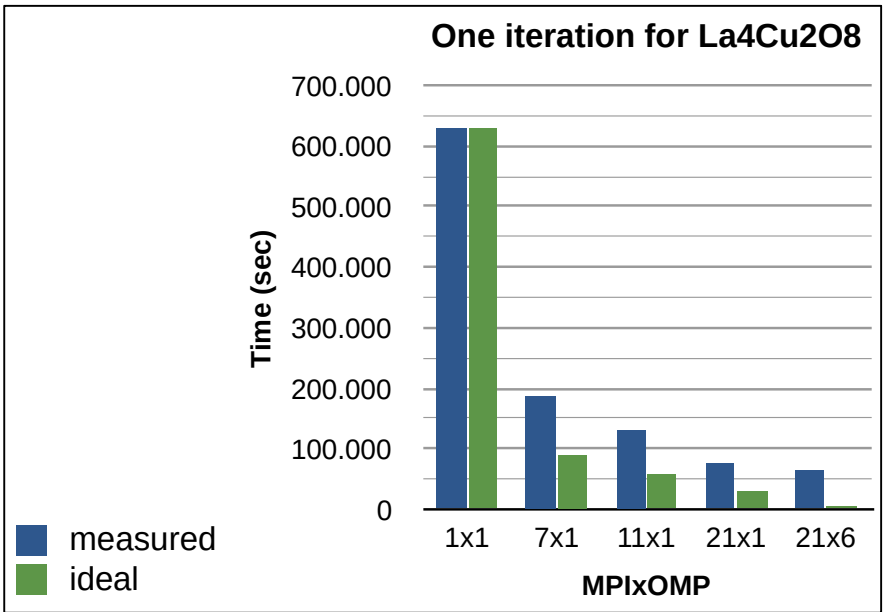


Figure 77: Graphical representation of time to solution (shown in table 26) versus number of MPI task times OpenMP threads. We compare measure versus ideal scaling.

Test case 2: antiferromagnetic $La_7SrCu_4O_{16}$ (40 irreducible k-points)

The test is done with 10,20 and 40 single-threaded MPI tasks and with 40 six-threaded MPI tasks.

	Time (sec)									
MPIxOMP	Total	allatoms	ground state	one iteration	seceqnfv (total)	seceqnfv (solve)	seceqnfv (setup)	seceqnsv	rhomag	other
10x1	13585.172	85.287	13499.885	2249.981	8214.087	3443.511	4770.576	1343.405	2414.338	1528.055
20x1	7096.498	95.888	7000.610	1166.768	4409.032	1848.320	2560.711	702.856	1225.473	663.249
40x1	3759.015	94.215	3664.800	610.800	2256.937	943.958	1312.979	352.155	626.789	428.919
40x6	2441.522	112.326	2329.196	388.199	895.182	411.618	483.565	217.010	348.879	868.125

Table 27: Aggregate time of the various parts of the Elk code for the 6 iterations of the ground state run of La7SrCu4O16.

Note that there is no inversion symmetry in this case. Hence, the Hamiltonian matrix is complex and consequently the diagonalization is a factor 3 slower.

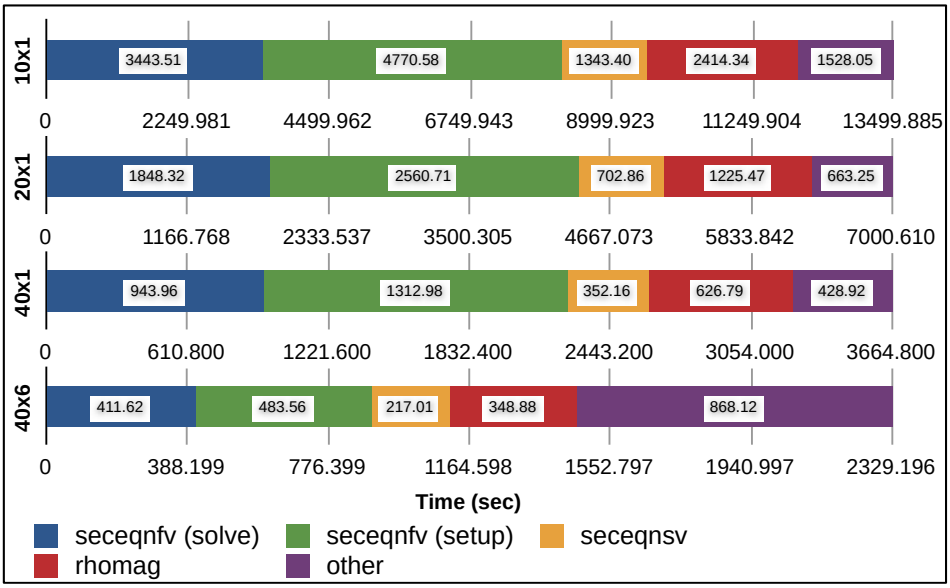


Figure 78: Graphical representation of the results shown in table 27.

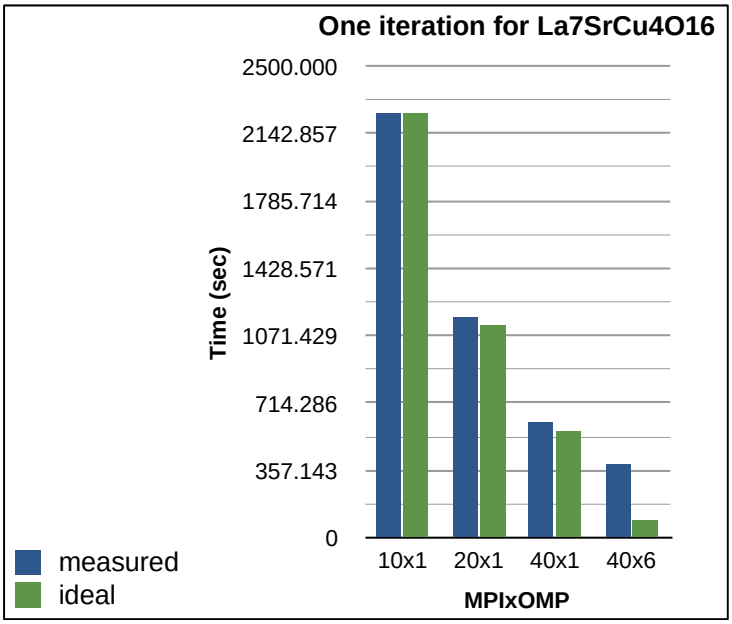


Figure 79: Graphical representation of time to solution (shown in table 27) versus number of MPI task times OpenMP threads. We compare measure versus ideal scaling.

6. Performance Analysis of Community Codes: Particle Physics

6.1 Overview

In this area we shall focus on optimising a key component of almost every lattice QCD application code, the so-called Wilson Dirac operator. Although many versions of this operator in its discretised form exist, the part that involves communication and is therefore more susceptible to architecture characteristics such as network bandwidth and latency, is almost identical across different lattice formulations.

We will focus on the tmQCD package, which is the production code of the European Twisted Mass QCD collaboration or ETMC [51]. Twisted Mass (TM) QCD refers to a particular discretisation of the fermion action of continuum QCD in which quarks are arranged as doublets with a modified mass term called the twisted-mass term. This formulation has two important advantages. The first advantage is that it avoids the problem of exceptional configurations that limits the lattice simulations with Wilson fermions to a relatively large quark mass. Consequently, simulations with light quark masses close to the physical values become possible. The second advantage is the automatic removal of leading errors due to the finite value of the lattice spacing, or what is known as O(a) improvement, where "a" is the lattice spacing. QCD Gauge configurations with a dynamical degenerate doublet of up and down quarks on large physical volumes, small lattice spacings, and small up and down quark masses have been generated by the ETMC and made available for researchers through the ILDG. In addition, configurations with dynamical up, down, charm, and strange quarks are currently being generated. These configurations are being used to compute important hadronic observables including the spectrum of hadrons as well as parameters of the standard model of particle physics with great success.

As mentioned, we focus on the operation of applying the Dirac matrix to a vector. For a lattice with L points in the spatial directions x,y,z and T points in the time direction t, a vector has dimension 12 L^3 T, where 12 corresponds to 3 colours and 4 spins. The Dirac matrix has a structure:

$$M = A + c\,H \qquad (6.1)$$

where A is a diagonal matrix, c is a constant, and H is called the hopping matrix with non-zero elements connecting nearest neighbour sites. Dividing the lattice sites into even and odd sites, the Dirac matrix will have the structure

$$M = \begin{vmatrix} A_{ee} & c\,H_{eo} \\ c\,H_{oe} & A_{oo} \end{vmatrix} \qquad (6.2)$$

where the subscripts ee,eo,oe,oo mean even-even, even-odd, odd-even and odd-odd respectively. For the parallel implementation with MPI, one needs to communicate the boundary sites when applying the hopping matrix. In turn this affects the performance of the code and it becomes important to optimise how this communication is performed and how it is integrated with computation.

An important part of lattice QCD calculations is the solution of the linear system:

$$M\,x = b \qquad (5.3)$$

where b is an input vector and x is the unknown solution vector. Many such systems need to be solved in the course of lattice calculations within the hybrid Monte Carlo simulation or when requiring the quark propagator to obtain the hadronic spectrum and structure related

observables. These systems are usually solved using iterative solvers such as the Conjugate-Gradient (CG) algorithm. In the CG algorithm, the hopping matrix needs to be applied to a vector O(1000) times. This illustrates the need to optimise this part of the code which we will call the kernel.

6.2 Performance analysis

In order to identify the bottlenecks of the code we performed a profiling test. For this test, we solved the system in Equation (5.3) using CG. Our test lattices are described in **Table 28**.

L	T	beta	kappa	mu	n_f
16	32	3.9	0.16090	0.0075	2
48	96	1.9	0.16124	0.0035	2+1+1

Table 28: Parameters of the test configurations. beta is a gauge coupling parameter that determine the lattice spacing. kappa and mu are two mass parameters and nf is the number of sea quarks. nf=2 means two degenerate light quarks corresponds to the up and down quarks and nf=2+1+1 means two light quarks and two heavy quarks corresponds to the strange and charm quarks.

The profiling tests are done on a Cray XE6 machine where each node has two 12-core AMD 'MagnyCours' 2.1-GHz processors (or a total of 24 cores per node). The machine has 6000 nodes with 32 GB DDR3 1333-MHz memory per node and 384 nodes with 64 GB DDR3 1333-MHz. The machine has a peak performance of 8.4 GFlop/s per core. Profiling was done using CrayPat.

The relevant user functions we want to profile are:

Qtm_pm_psi : the application of $Q^+ Q$ where Q is the even-odd preconditioned Dirac matrix.

Hopping_Matrix : application of H_{eo} or H_{oe}, called within *Qtm_pm_psi()*.

We'd like to note at this point that although *Qtm_pm_psi* calls *Hopping_Matrix,* in what follows the time indicated by the profiler for *Qtm_pm_psi* is in fact the time spent in this function *excluding* the time spent in *Hopping_Matrix.*

6.2.1 Single core performance:

We first run the 16^3x32 lattice on a single core. In this case there is no communication required (even though MPI-calls still occur in the code). Since the hopping matrix has the same sparsity in every application, the number of floating-point operations per iteration is directly measurable and therefore the floating-point rate can be computed simply by dividing with the total time spent in *Qtm_pm_psi*.

In this run 1516 iterations of CG where performed, requiring 679 seconds. This means a sustained floating-point performance of 797.5 GFlop/sec double precision.

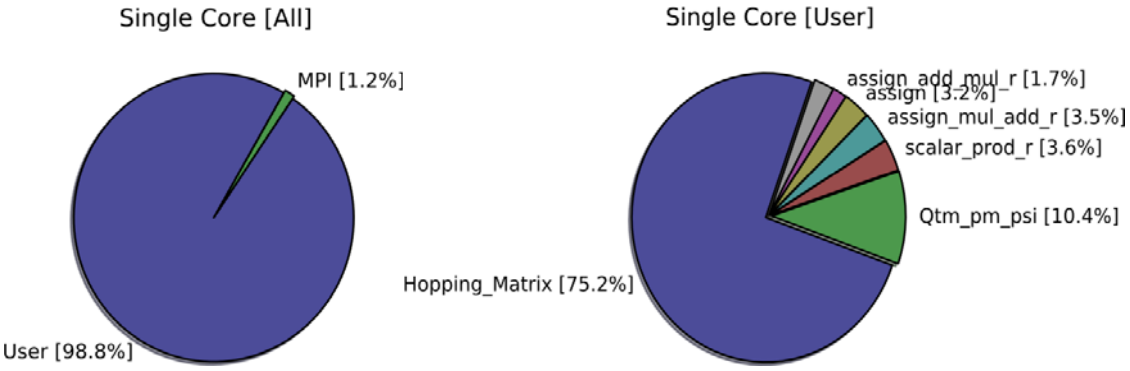


Figure 80: Profiling of the twisted mass inverter code. The left chart compares User and MPI functions, while the right chart compares the User functions (percentages are with respect to the total time spent in User functions).

In Figure 80 we show profiling results of this single core run. In this case where no communication is required, we see that about three quarters of the run-time are spent in the *Hopping_Matrix* function, which involves coupling the nearest neighbors, while all other functions shown are local, linear operations, which don't require complicated memory accesses.

6.2.2 Single node performance

For this test, we run the 16^3x32 lattice on 8 cores on a single node. In this case, MPI communication will take place on a single node. The code required 1521 iterations to converge in 151 seconds. This means a sustained performance of 797.5 GFlop/sec double precision, which is consistent with the single-core run.

In Figure 81 we plot the profile data for the run on a single node. As can be seen about 15% of the time is now spent in MPI functions. 65% of this time is in the blocking *MPI_Waitall()* function, which is called in the communication part of *Hopping_Matrix()*.

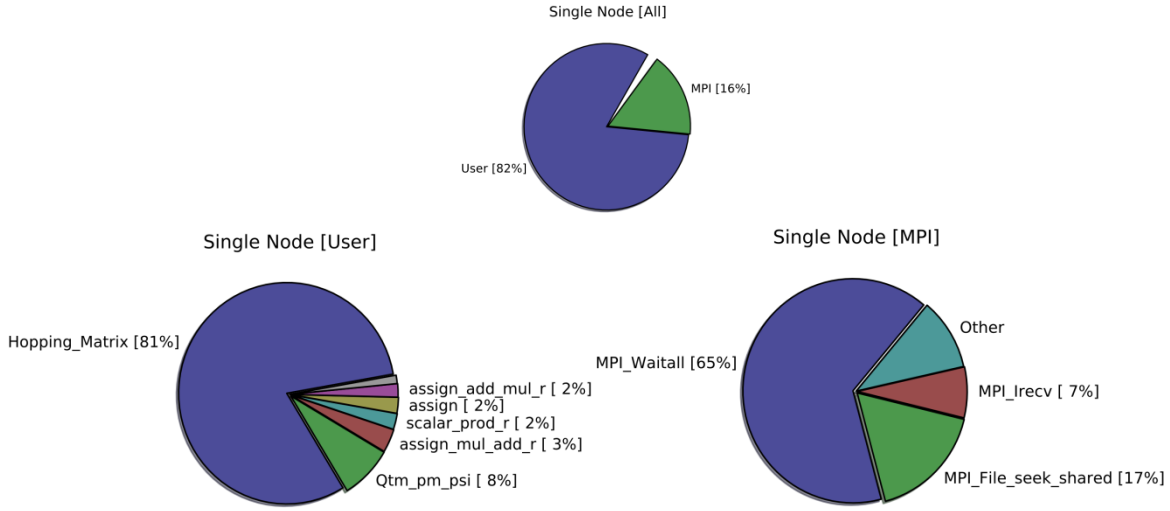


Figure 81: Profiling of the twisted mass inverter code on a single node. Centre for User and MPI functions with respect to the total time. The left chart is a break-down of the User functions (percentages are with respect to the total time spent in User functions) and the right chart is a break-down of the MPI functions (percentages are with respect to the total time spent in MPI functions)

6.2.3 Many nodes performance with a large lattice:

We perform a test on many nodes using the 48^3x96 lattice. We run this test on 576 cores with 24 cores per node. For the 5299 iterations the solver required 601 seconds, meaning a sustained performance of 442.4 GFlop/s in double precision.

In Figure 82 we show profiling charts of the run on 24 nodes. The conclusions are the same as in the case of Section 6.2.2, namely that the bulk of the time spent in MPI functions is spent in the *MPI_Waitall()* function which blocks for the communication in the *Hopping_Matrix()* function.

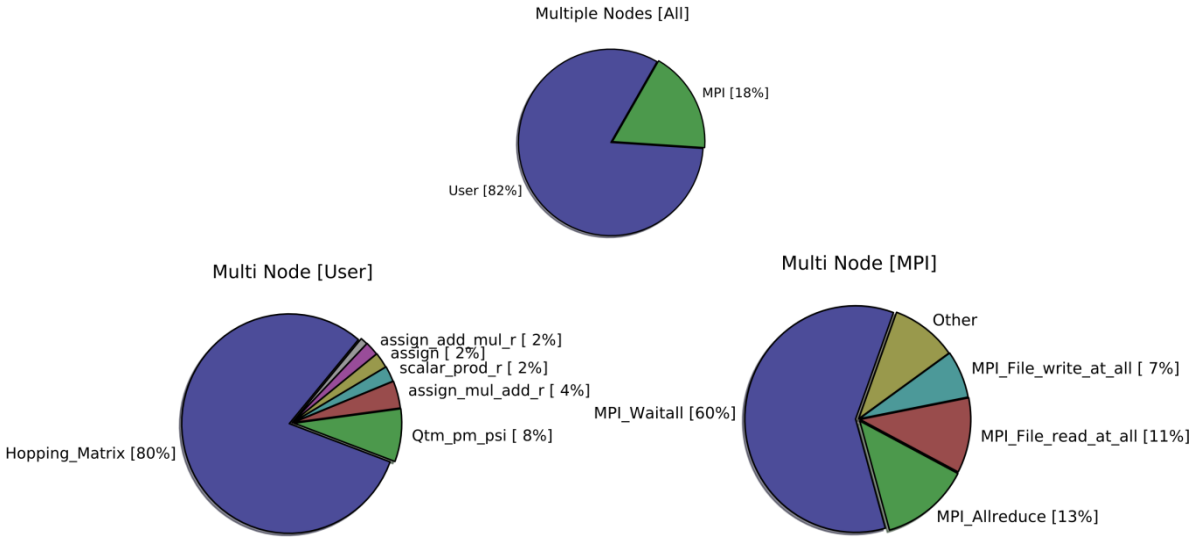


Figure 82: Profiling of the twisted mass inverter code on a 24 nodes. Notation is the same as in the previous figure.

6.2.4 Strong scaling:

The second area of possible improvement is related to the fact that the parallelisation in the current implementation is done using MPI only. Given that future machines will probably be equipped with many cores per node it will be advantageous to use a hybrid implementation with MPI and OpenMP. This will take advantage of the shared memory feature of those multi-core/node machines. We performed a strong scaling test of the 48^3x96 lattice on a Cray XE6 and a BlueGene/P machines.

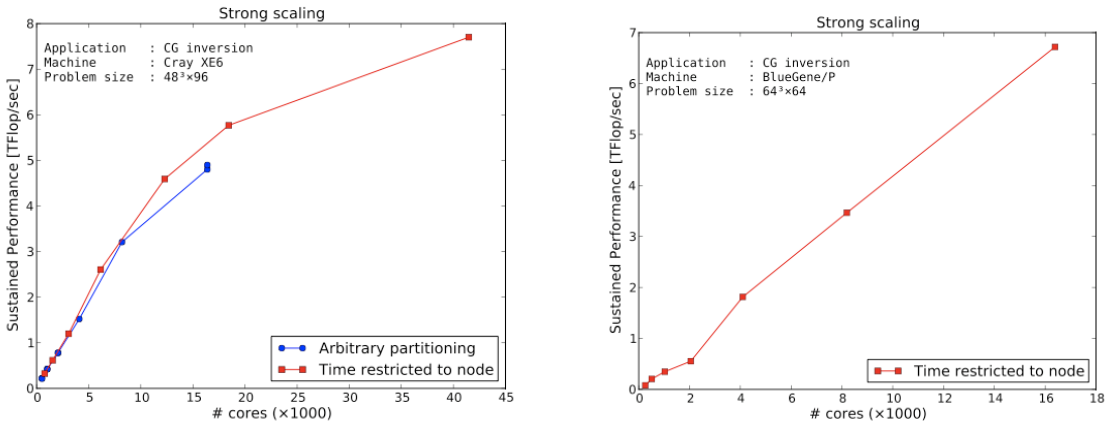


Figure 83: Strong scaling test of the twisted mass inverter on a CrayXE6 (left) and a BlueGene/P (right). The points labeled "Time restricted to node" refer to scaling tests carried out where care was taken so that the spatial lattice sites where mapped to the physical 3D torus topology of the machine's network, which restricts the time-dimension partitioning to a node.

6.3 Discussion:

Our profiling tests have corroborated what is well known about lattice QCD codes; that the majority of the wall time is spent in the application of the Kernel. For the specific code we have seen that the code requires optimisation in terms of the floating-point performance as well as the way communication is done.

In terms of the floating-point performance, lattice QCD codes are rather balanced in the ratio of floating-point operations per memory access. The exact knowledge of this ratio allows for an approximate estimate of the sustained performance one should be able to reach on a given architecture. The number of floating-point operations in an application of the kernel on a single lattice site is of the order of 1.3 thousand, while the bytes of memory needed to be read or written in order to apply the kernel on a single site is 960 bytes (in double precision). This gives a ratio of 1.4 flops to bytes I/O for the kernel. To see what this means in terms of sustainable performance, we need to compare with the characteristics of a given machine, say the CrayXE6. This machine has 24 cores per node, which have a theoretical peak of 8.4 GFlop/core and all 24 cores share a common bandwidth to memory, which has a theoretical peak of 85.3 GB/s. This machine has therefore a flop-to-bytes of I/O ratio of about 2.4 flops/byte. Therefore, to a first approximation, we can say that the maximum floating-point performance a lattice QCD kernel could achieve on this architecture is ~60% of peak performance. We stress that this number is a first approximation, since the performance of the kernel can only asymptotically reach this number. To achieve this, our kernel would have to read each site once, which is practically unfeasible due to the imposing of boundary conditions, and we would have to physically saturate the memory bandwidth, which in practice we know is technically unachievable.

However the above calculation indicates that the code is not well optimised in terms of the floating-point performance we are currently seeing; the fact that we achieve around 800 MFlop/sec, which is about 10%, means there is room for improvement. This is the first point we would like to address in terms of our effort in this work package.

Apart from the computational efficiency of the kernel, it seems the communication strategy is also sub-optimal the way it is currently implemented. Indeed, the profiling shows that the bulk of the time spent in MPI functions is spent in *MPI_Waitall()*. With careful inspection of the source code of *Hopping_Matrix()* we were able to identify that this blocking call is made for waiting for boundary data communicated between nearest neighbouring processes. Specifically, the communication schedule follows the algorithm:

1. Prepare boundaries (communication buffers)
2. Start communication of boundaries
3. Wait for boundaries (block)
4. Apply operator

This structure could be improved by overlapping communications and computations such that the code performs part of the computation while part of the data is being communicated. There is no unique way to do this, depending on the latency and bandwidth of the architecture; less "aggressive" overlapping means less tolerance to latencies but are usually simpler to implement and require less memory. More "aggressive" schemes require more temporary buffers and are more complicated with the upshot that smaller bandwidths and larger latencies can be hidden. Using test kernels we will investigate this issue starting from the simplest, less aggressive overlapping schemes and evolve towards more aggressive ones if required.

Apart from the aforementioned issues we have identified directly through the profiling work done, we would also like to list several modification that would also benefit this implementation, and we which we intend to focus on in the future:

- **Thread level parallelism:** there is currently only process-level parallelism implemented in the code (MPI). On major improvement would be to add thread-level parallelism via OpenMP pragmas for instance.
- **Refactoring of right-hand-sides loop:** Better data reuse can be achieved by modifying Hopping_Matrix() to act on (and return) multiple vectors rather than one. This would also allow for better thread-level parallelism.
- **Eigenvalue deflation:** Shorter time-to-solution can potentially be achieved if the lowest few eigenvalues of the kernel are removed before the inversion.
- **Domain Decomposition:** This is a preconditioner which effectively allows for less frequent communication of boundaries, thus yielding an algorithm more tolerable to network bandwidth and latency.
- **Using Poisson brackets to tune Monte Carlo parameters:** A method to improve the scaling of the Monte Carlo algorithm with respect to the problem size "V" from V5/4 to V9/8.
- **Parallel Landau and Coulomb gauge fixing:** These are lattice QCD operations which are difficult to optimise since they rely on an efficient parallel FFT.

Some of these are purely algorithmic improvements which can be investigated in the context of their suitability for massively parallel architectures while others are technical modifications which require code refactoring for better scaling to massively parallel architectures.

7. Conclusions and Next Steps

In this document we presented the results of the performance analysis of the codes proposed by the scientific communities as targets of the re-design and refactoring action addressed by WP8. This is the first step of the performance modelling methodological approach, and its outcomes will be used to evaluate code's behaviour in an analytical and quantitative way and to estimate potential performance improvements on the next generation HPC systems, highlighting critical numerical kernels that can benefit greatly from the adoption of novel algorithmic/computational approaches that allows the exploitation of innovative hardware/software (e.g. GPU/CUDA) solutions. Such analytical study will be the subject of the next step of WP8, and will be reported in deliverable D8.1.3 "Prototype Codes Exploring Performance Improvements".

According to the outcomes of our analysis, all the codes under investigation are suitable candidates to the WP8 refactoring work. Relevant, computing intensive, parts that can potentially benefit of new software/hardware, can be clearly identified. A quantitative estimate of the impact of the proposed refactoring work will be provided as a result of the analytical study phase, together with a precise specification of the numerical kernels that needs to be re-designed and re-implemented.

It is worth pointing out that the analysis phase could be accomplished only thanks to a successful synergy between the scientific community code developers and the HPC experts, blending the deep knowledge of codes and algorithms of the former -- necessary to properly set up the benchmarks, to provide the analytical modelling, and to check the results -- with the expertise in HPC systems, parallel programming and performance analysis tools of the latter. The same approach will be adopted in WP8 also for the next steps.