
# 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**

# D8.1.3
# Prototype Codes Exploring Performance Improvements
# *Final*

| | |
|---|---|
| Version: | 1.0 |
| Author(s): | Claudio Gheller and Will Sawyer (CSCS) |
| Date: | 23.12.2011 |

## 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: D8.1.3** | |
| | **Deliverable Nature:** Report | |
| | **Deliverable Level:** PU * | **Contractual Date of Delivery:** 31 / 12 / 2011 |
| | | **Actual Date of Delivery:** 31 / 12 / 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: Prototype Codes Exploring Performance Improvements** | |
| | **ID: D8.1.3** | |
| | **Version:** | **Status:** 1.0 |
| | **Available at:** http://www.prace-project.eu | |
| | **Software Tool:** Microsoft Word 2007 | |
| | **File(s):** D8.1.3.docx | |
| **Authorship** | **Written by:** | Claudio Gheller and and Will Sawyer (CSCS) |
| | **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:** | Richard Blake, Thomas Eickermann |
| | **Approved by:** | MB/TB |

## Document Status Sheet

| Version | Date | Status | Comments |
|---|---|---|---|
| 0.1 | 18/11/2011 | First skeleton | |
| 0.2 | 23/11/2011 | Updated version with astro stuff | |
| 0.3 | 01/12/2011 | Intro and conclusions added | |
| 0.4 | 02/12/2011 | Contributions added | |
| 0.5 | 05/12/2011 | Contributions added | |
| 0.6 | 06/12/2011 | Contributions added | |
| 1.0 | 31/12/2011 | Final version | |

## Document Keywords

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

# Table of Contents

# List of Figures



# References and Applicable Documents

[1] http://www.prace-ri.eu
[2] Deliverable D8.1.1: “Community Codes Development Proposal”
[3] Bridging Performance Analysis Tools and Analytic Performance Modeling for HPC, T. Hoefler, Proceedings of Workshop on Productivity and Performance (PROPER 2010), Springer, Dec. 2010.
[4] 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.
[5] Performance Modeling: Understanding the Present and Predicting the Future. Bailey, David H.; Snavely, Allan. http://escholarship.org/uc/item/1jp3949m
[6] 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
[7] Deliverable D8.1.2: “Performance Model of Community Codes“
[8] http://web.me.com/romain.teyssier/Site/RAMSES.html
[9] http://lca.ucsd.edu/portal/software/enzoFLASH
[10] http://www.mpa-garching.mpg.de/gadget/
[11] ICON testbed; https://code.zmaw.de/projects/icontestbed
[12] ICOMEX project; http://wr.informatik.uni-hamburg.de/research/projects/icomex

[13] 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.
[14] Conti, C; W. Sawyer: GPU Accelerated Computation of the ICON Model. CSCS Internal Report, 2011. A G Sunderland, C J Noble, V M Burke and P G Burke, CPC 145 (2002), 311-340.
[15] A G Sunderland, C J Noble, V M Burke and P G Burke, CPC 145 (2002), 311-340.
[16] K L Baluja, P G Burke and L A Morgan, CPC 27 (1982), 299-307.
[17] 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/.
[18] The MRRR algorithm for multi-core and SMP systems, http://code.google.com/p/mr3smp/.
[19] The Parallel Linear Algebra for Multicore Architectures project, http://icl.cs.utk.edu/plasma/.
[20] The Matrix Algebra on GPU and Multicore Architectures project, http://icl.cs.utk.edu/magma/.
[21] Single Node Performance Analysis of Applications on HPCx, M. Bull, HPCx Technical Report HPCxTR0703 (2007).
[22] Combined-Multicore Parallelism for the UK electron-atom scattering Inner Region R-matrix codes on HECToR, HECToR Distributed CSE Support projects, http://www.hector.ac.uk/cse/distributedcse/projects/.
[23] Wolfe, M. and C. Toepfer, 'The PGI Accelerator Programming Model on NVIDIA GPUs Part 3: Porting WRF', PGI Insider Article, October 2009, (http://www.pgroup.com/lit/articles/insider/v1n3a1.htm).
[24] Madec, G: NEMO ocean engine, Note du Pole de modélisation, Institut Pierre-Simon Laplace (IPSL), France, No 27 ISSN No 1288-1619, 2008.
[25] 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.
[26] http://www.hector.ac.uk
[27] http:// www.top500.org
[28] https://hpcforge.org/projects/pkdgrav2/
[29] J. Barnes and P. Hut (December 1986). "A hierarchical O(N log N) force-calculation algorithm". Nature 324 (4): 446-44
[30] Ewald P. (1921) "Die Berechnung optischer und elektrostatischer Gitterpotentiale", Ann. Phys. 369, 253–287.
[31] Long Wang et al., Large Scale Plane Wave Pseudopotential Density Functional Theory Calculations on GPU Clusters, SC2011
[32] Spiga F. & Girotto I., phiGEMM: a CPU-GPU library for porting Quantum ESPRESSO on hybrid systems, 20th Euromicro International Conference on Parallel, Distributed and Network-Based Computing (PDP2012), Special Session on GPU Computing and Hybrid Computing, February 15-17, 2012, Garching (Germany) - accepted
[33] http://www.tddft.org
[34] 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.
[35] http://www.tddft.org/programs/octopus/wiki/index.php/Main_Page
[36] http://www.yambo-code.org/
[37] http://www.abinit.org/
[38] http://www.quantum-espresso.org/
[39] http://www.icmab.es/dmmis/leem/siesta/

[40] A. M, Khokhlov, Fully Threaded Tree Algorithms for Adaptive Refinement Fluid Dynamics Simulations, 1998, Journal of Computational Physics, 143, 519
[41] http://lca.ucsd.edu/portal/software/enzo
[42] http://www.mpa-garching.mpg.de/gadget/
[43] http://code.google.com/p/cusp-library/

## 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) |
| CUSP | CUda SParse linear algebra library |
| 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) |
| 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 |
| NCAR | National Center for Atmospheric Research |
| NCF | Netherlands Computing Facilities (Netherlands) |
| NEGF | non-equilibrium Green's functions, |
| NERC | Natural Environment Research Council |
| NEMO | Nucleus for European Modeling of the Ocean |
| NERC | Natural Environment Research Council (United Kingdom) |
| NWP | Numerical Weather Prediction |
| OpenCL | Open Computing Language |
| OpenMP | 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 |
| UML | Unified Modeling Language |
| UPC | Unified Parallel C |

# Executive Summary

In this document we describe the performance-critical numerical kernels extracted from a number of community codes, proposed by the scientific communities, which were identified in the first month of PRACE-2IP work package 8. The kernels have been selected according to the results of the performance analysis previously accomplished (and reported in deliverable D8.1.2). A number of approaches are envisaged for refactoring the kernels refactoring, in order to allow them to efficiently and effectively exploit the coming generation of HPC systems. This was part of the performance modelling methodology adopted in this work package in order to quantitatively evaluate the potential performance benefits that can be achieved on the new hardware architectures if novel software solutions are adopted. This final performance evaluation will be estimated by month six of the project (M6, )along with the specification of the final refactoring plan.

## 1 Introduction

In this deliverable we describe the performance critical numerical kernels extracted from a number of community codes, proposed by the scientific communities identified in the first month of PRACE-2IP [1] work package 8 (hereafter WP8; see deliverable D8.1.1 [2]), and the approaches that have been envisaged for their refactoring, in order to allow them to efficiently and effectively exploit the coming generation of HPC systems.

Selected codes are:

| **Domain** | **Application** | **Usage** |
|---|---|---|
| Astrophysics | RAMSES | Galaxy - cluster of galaxy evolution. |
| | PKDGRAV | Large scale structure of the universe, precision cosmology |
| | PFARM | Electron-atom scattering |
| Climate | OASIS | Full climate modelling, coupler |
| | CDI/XIOS/PIO | Efficient I/O libraries |
| | ICON | Dynamical core |
| | NEMO | Ocean models |
| Material Science | ABINIT | Density functional theory, Density-Functional perturbation theory, Many-Body perturbation theory, Time-Dependent Density functional theory |
| | Quantum ESPRESSO | Density-Functional theory, Plane Waves, and Pseudo-Potentials, Projector-Augmented waves. |
| | YAMBO | Many-Body perturbation theory, Time-Dependent Density functional theory |
| | SIESTA | Electronic structure calculations and ab-initio molecular dynamics |
| | OCTOPUS | Density Functional Theory |
| | Exciting/ELK | Full-Potential Linearized Augmented-Plane Wave |
| Particle Physics | tmQCD | Lattice QCD |

Note that, one of the codes evaluated in Deliverable D8.1.2, EULAG, was dropped since it was viewed not appropriate for further effort. It has proven scalability but its rectangular grid is not applicable for global climate applications. Note also, that the analysis on Exciting/ELK

will be presented in the next deliverable (D8.1.4), since, due to the tight deliverables schedule, the reference community could not provide their inputs on time.

The proposed codes have been the subject of a detailed performance analysis in order to identify the most promising kernels deemed ready for the refactoring effort. Due to the broad spectrum of applications under investigation, an appropriate methodology was defined, based on the "Performance Modelling" [3][4][5][6] approach. Performance modelling has the goal of gaining insight into an 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 the study of the current behaviour of a code but also represents a predictive tool, estimating the behaviour on a different computing architecture and identifying the most promising areas for performance improvement.

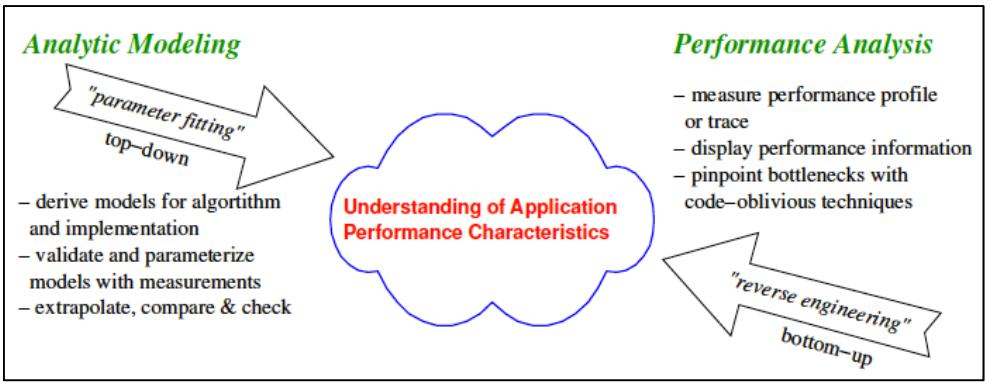


**Figure 1: The performance modelling methodology.**

The methodology has been described in details in deliverable D8.1.2 [7].

Essentially, it accounts for the two main steps sketched in Figure 1. The first step consists of the analysis of the codes' performance, using standard performance tools, collecting all the information necessary to understand the behaviour of the principal algorithms and their dependencies on the relevant model parameters (e.g. the number of cells of the computational mesh) and on the hardware. This step was accomplished during PMs 2 and 3 and the results described in D8.1.2.

In the second step, the accrued data are synthesised and performance modelled in order, first, to gain greater understanding of the performance phenomena involved, and to identify the numerical kernels that will be the subject of WP8 proposing solutions for their re-design and refactoring. Then, the model is used to project performance to other system/numerical solutions combinations. The former is the subject of the present deliverable, in which we describe the work undertaken to select the kernels extracted from the different codes We justify quantitatively these choices using the collected performance analysis data. In the next deliverable, that will complete the performance modelling phase, we will present the expected performance improvements on promising architectures (multicore, GPUs, MIC…) with the most effective programming approaches (MPI, OpenMP, CUDA, OpenCL, etc. and their combinations). We will ustify quantitatively the choices (extrapolating from the benchmarks, according to the performance modelling approach): and identify what improvements can be expected exploiting a given new architecture with a "novel" parallel programming approach. The results will lead to the specification of the detailed work plan for code refactoring and re-implementation, along with a testing and validation strategy.

This document is organized as follows. Sections 2 to 5 are dedicated to describing the kernels that have been identified as candidates for refactoring, and to show the results of the

performance analysis accomplished on these kernels. Section 2 is dedicated to codes in the domain of Astrophysics, while Climate and Material Science are the subjects of Section 3 and 4 respectively. In Section 5 we focus on Particle Physics. Finally, in Section 6, we draw the conclusions and discuss the next steps for WP8

# 2 Astrophysics

In this section, we present kernels selected for performance modelling of three codes in the Astrophysics domain. RAMSES and PKDGRAV are well known open source codes, 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 describes electron-atom and electron-ion scattering and can be effectively adapted and exploited in the astrophysics framework, where the precise and efficient description of atomic physics, is necessary to compare simulations and observations.

## 2.1 RAMSES

The RAMSES code [8] is an adaptive mesh refinement (AMR) multi-species code, describing the behaviour of both the baryonic component, represented as a fluid in the cells of the AMR mesh, and the dark matter component, represented as a set of collisionless particles. These 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.

During the performance analysis phase [7], we identified the critical parts ("sections") of the code:

- *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 calculate fluxes to solve conservation equations, a Riemann solver, and a finite-volume solver.
- *Gravity*: this group comprises functions needed to calculate the gravitational potential at different resolutions using a multigrid-relaxation approach.
- *MPI*: comprises all the communication related MPI calls (data tranfer, synchronisation, management)

Several other sections, like N-body (functions needed to update the positions and velocities of particles and to evaluate the gravitational force acting on each particle) and I/O have been investigated, but proved to be negligible compared to the other sections.

In what follows we will focus on the two critical sections, *Hydro* and *Gravity*. Most of the *MPI* overhead is due to operations performed within these two sections, hence its impact and the related solutions toward performance improvement will be included in their discussion.

### *2.1.1 Data structure*

In order to properly understand the results coming from the performance analysis, the principal data structures defined in the code must be described.

Three main data "containers" are defined, one for the fluid variables (mass and energy density, velocity), one for the gravitational forces and one for the N-body particles. The first two are defined on the AMR mesh, while the third is a meshless data structure containing particle positions and velocities. This last array is not relevant for the performance analysis and will be neglected in the following discussion.

The two AMR based data structures have to describe quantities on a variable resolution mesh. RAMSES adopts a peculiar approach, based on the "Fully Threaded Tree" [40], in which, starting from a basic Cartesian mesh, refinements are created on a cell-by-cell basis, according to proper criteria. The fundamental data object is represented by the Oct, a small grid composed, in 3D, by 8 cells (see Figure 2). Octs are organised as linked lists, fully connected with other mesh elements, pointing towards:

1. The parent cell
2. The six neighbouring parent cells
3. The six neighbouring octs
4. The eight children octs

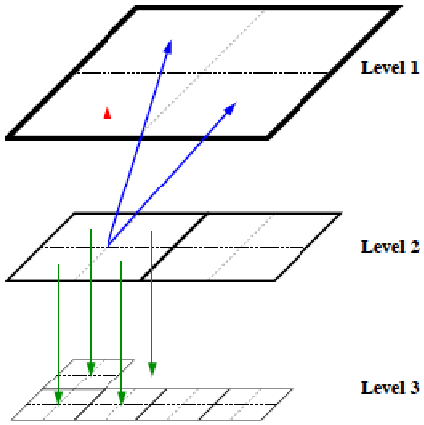


**Figure 2: Oct structure (in 2D composed by 4 cells), its possible refinements and links to parent and children octs**

With this approach, each Oct has all the necessary links to the cells relevant for the update of its physical status. This allows to avoid having to define a global tree structure (adopted in other similar codes, like Enzo [41] or Flash [9]), managing the distribution of meshes at different resolution levels. This is typically a computational demanding structure to build and, in parallel, to balance, requiring intensive data exchange between processors. Furthermore, in order to achieve acceptable performance, it is usually replicated on each processor, strongly limiting the scalability, in terms of memory usage, of the code (soon memory is filled by the global tree structure, this problem becoming worse and worse with the decrease of core memory, typical for current HPC systems). This is all avoided with the RAMSES' approach. The penalty is in terms of memory usage (due to the large number of pointers associated to each Oct), memory access performance, since Octs are distributed randomly in memory and load balancing that is hard to optimize. This last problem has already been relieved by the adoption of a domain decomposition technique based on "Mesh Partitioning" inspired by Tree N-Body codes [42]. With this approach, the usage of space filling curves arranges Octs according to a locality criterion that allows to distribute data among different processors in an effective, though not perfect (see discussion below), way (see Figure 3).

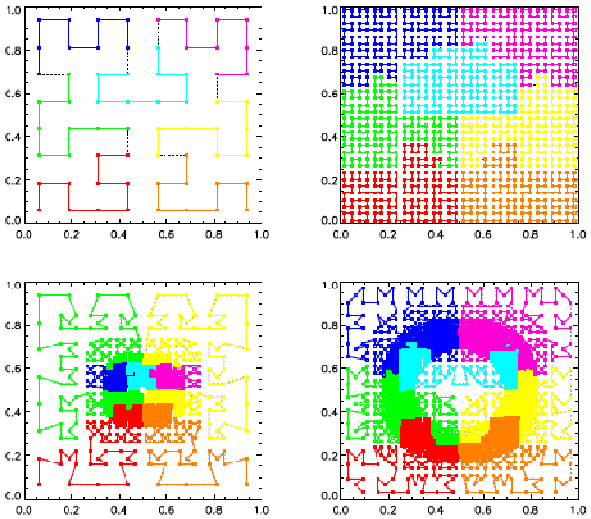

**Figure 3: Domain decompositions based on a Peano-Hilbert space filling curve.**

Octs' cells at the maximum possible resolution in a certain region (that have no children) are classified as "active". Those with children are classified as "passive". These are auxiliary cells where equations are not directly solved (values are averaged from higher resolution cells). On average, passive cells are estimated to be the 15% of the total number of cells.

### *2.1.2 Hydro Kernel*

In the *Hydro* section the equations of conservation of mass, momentum and energy, extended to include the gravitational field and other possible external sources (cooling, magnetic field etc.), are solved. A number of solvers are implemented, with different levels of accuracy, all integrating the equations on the AMR Oct-based mesh.

The *Hydro* kernel workflow can be summarized as follows:

1. Start the integration sweep updating all *ntot* active cells of the AMR structure, from time $t^n$ to $t^{n+1}$.
2. Divide the whole integration sweep in chunks, each integrating a fraction, *ncache/ntot*, of the active cells. A proper setting of *ncache* is an important choice, whose relevance will become clear later.
3. For each cell, collect all the data necessary to update its status. This means that around each cell, a 6x6x6 auxiliary mesh (hereafter AuxBox) is built, composed of cells at the same resolution, containing fluid and gravitational field data copied by neighbouring Octs at the same level, if present, or interpolated/averaged from lower/higher resolution levels. At this stage memory is increased by a factor of $6^3$. This is where *ncache* becomes important first. Its value must be set low enough in order to prevent the memory usage "exploding".
4. Each AuxBox is passed to the hydro solver and the corresponding cell updated. Here, one of five different solvers can be selected. Each solver has a different accuracy and impact on the performance.

According to our analysis, the most critical parts for the kernel performance are the creation of the AuxBoxes and the call to the hydro solver. This can be inferred already from Figure 4, where the hydro solver accounts for most of the *Hydro* time, while the AuxBoxes building contributes to the *MPI* overhead.

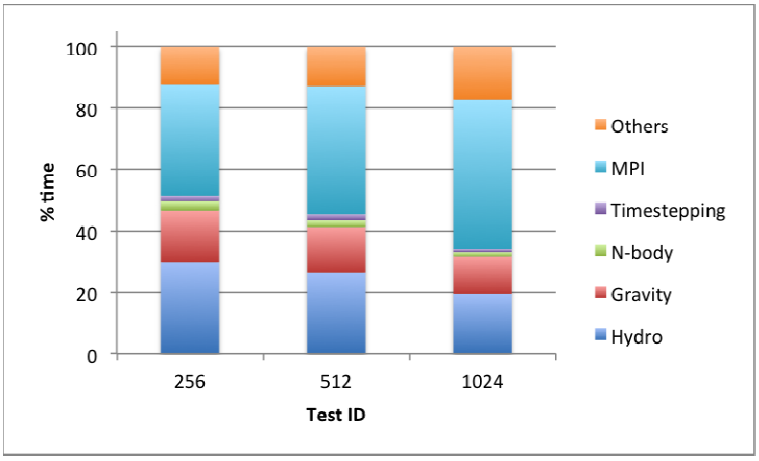


**Figure 4: Distribution of the work for a production test as a function of the number of processors.**

### AuxBoxes performance improvement model

This is the part of the *Hydro* kernel most affected by data access. In the RAMSES' AMR data structure, memory contiguity of two neighbouring Octs is not ensured. Therefore the corresponding data can be far away in the processor's memory or even in the memory of two different processors. Furthermore, despite the usage of the space filling curves, the load is not perfectly balanced between processors. This problem grows with the number of processors, since smaller chunks of data are assigned to each processor. This means that, first, the data distribution tends to be more and more heterogeneous, leading to higher imbalance of the work. Second, in order to build the AuxBoxes, each processor has to access information stored on a larger number of processors, affecting strongly the network load and the communication overhead.

In order to improve the performance, the load balancing must be improved and the communication overhead must be reduced. This can be achieved byworking on the basic algorithmic architecture, changing the AMR data structure, or working on the domain decomposition strategy, increasing data locality. A third solution is that of exploiting some specific hardware features, like multicore nodes with large shared memory available. Let us explore in more detail the three solutions.

*Data Structure:* this first approach has the largest impact on the whole code and can be considered as an extreme solution. However, it could be adopted in specific situations in which the AMR data structure is redundant. The typical case is when RAMSES is used with uniform resolution, so with AMR switched off. In these cases, a linked list based approach is highly inefficient and can be replaced by a regular rectangular mesh representation, using directional splitting integration, instead of full 3D integration (currently adopted), in order to exploit arrays data locality and decomposing the domain between processors in regular, equal size, rectangular sub-volumes with ghost-cells, for boundaries.

*Domain decomposition:* The problems that RAMSES treats are intrinsically unbalanced. Due to gravitational clustering, regions at high resolution form at unpredictable locations in the computational box, changing continuously with time. Precise load balancing algorithms can be implemented, but they tend to become the dominant part of the calculation, wiping out any benefit gained from the other parts of the code.

*Hybrid parallelization:* the previous problem can be strongly mitigated by specific hardware solutions. In particular, the coming generation of computing cores will have increasingly smaller memories, for both performance and power consumption reasons, but they will be organized (typically) in 8-16 cores per node, sharing a memory of relevant size. In a 16 core node, 16 or 32 GB of shared memory can be expected. RAMSES can exploit this architecture

by decomposing the computational domain between nodes instead of cores, thus dealing with much larger data chunks, with intra-node direct memory access. The communication overhead is consequently reduced and load balancing is much easier to optimise.

In order to exploit shared memory, the Hydro kernel has to be re-implemented with a hybrid OpenMP+MPI approach. This, in principle, could be accomplished by an OpenMP parallel loop running on all the active cells stored in each node. However, special care must be devoted to managing the access to shared memory. This in fact, is in general only logically shared, being physically distributed among the cores inside a node. Furthermore, concurrent access to the same memory location can be frequent when a large number of threads is instanced and work simultaneously to build different AuxBoxes. Therefore, OpenMP implementation must be developed:

- Preserving data locality even inside the shared memory
- Handling effectively concurrent data access

**Hydro solver performance improvement model**

The hydro solver, due to its high accuracy, is intrinsically computational intensive. Five different solvers are available, the most accurate being based on an exact Riemann solver that is extremely demanding. However, in most cases, a light approximate Riemann solver can be used.

The implementation of the hydro solver proved to be highly optimized, so no relevant improvements are expected from algorithms refactoring. Furthermore, any change could affect the accuracy of the solution, which is a crucial feature of this solver.

However, benefits can be expected by the usage of accelerator devices, like GPUs or Intel MIC. In fact, once AuxBoxes are built around the *ncache* cells, the computation is completely local and fully vectorizable: in order to calculate the new value of each cell, the code uses only the data stored in the corresponding AuxBox, with no other access to memory (local). Hence, each cell can be calculated by a different thread independently from all the other, (vectorizable) perfectly matching, e.g, the CUDA programming model.

A number of issues must be taken into account:

1. The main memory size of the device: only part of the active cells can be moved to the device, since its memory is expected to be smaller than that of the corresponding node. Here a careful setting of the *ncache* parameter becomes crucial, to fit the memory size of the device. The algorithm does not require any change, but just a careful tuning of the parameter.
2. The cache memory of each core of the device: this ideally should contain all the AuxBox data, in order to minimize main memory accesses. However, the $6^3$ cells based AuxBox structure is, from a storage point of view, rather demanding, therefore this optimization appears to be challenging and needs to be carefully investigated on different device architectures, potentially requiring deep algorithmic changes.
3. Data movement: due to its complexity the hydro solver is characterized by many successive loops. AuxBox data has to be moved to the device memory only once, at the beginning of the procedure and then it has to lie on the device till the end of the computation, when only the updated cells dataset, much smaller than the input dataset, is moved back to the processor memory. Unnecessary data copy would produce a big overhead.
4. Functions inlining: in order to run on the device, functions called within a loop, should be inlined. This is actually a requirement that could be relaxed in future device generations. However, currently it may represent an issue; due to the high number of functions called in the hydro solver.

5. The Riemann solver: the exact Riemann solver is unsuitable for GPU-like devices, being characterized by frequent branching. The adoption of the approximate solver is required. This rules out all those applications for which the exact solver is necessary.

Note that most of the code refactoring needed to run the hydro kernel on the device does not foresee any modification in the core algorithms that can be taken as “black boxes”.

### *2.1.3 Gravity Kernel*

The Gravity section calculates gravitational forces on each fluid element and particle in the simulations. Forces are computed as partial directional derivatives of the gravitational potential $\Phi$, that is evaluated solving the Poisson equation

$$\Delta\Phi = \rho$$

where $\Delta$ is the Laplacian operator and $\rho$ is the total matter density (baryon + dark matter mass).

The Poisson equation is solved by means of a multigrid scheme with Dirichlet boundary conditions on a Cartesian grid with irregular domain boundaries. This scheme was developed in the context of the AMR scheme. The Poisson equation is solved on a level-by-level basis, using a “one-way interface” approach in which boundary conditions are interpolated from the previous coarser level solution with a second order accurate reconstruction algorithm. Such a scheme is particularly well suited for self-gravitating astrophysical flows requiring an adaptive time stepping strategy.

For the one-way interface schemes implemented in RAMSES the Poisson equation is solved over the whole AMR hierarchy on a level-by-level basis, the size of the cell involved in each individual Poisson solve being uniform. Each AMR level constitutes the finest level of the multigrid hierarchy. A Cartesian grid with complex irregular boundaries therefore defines the finest multigrid level. We then build a hierarchy of coarser grids, which cover this reference domain (see Figure 5). This new hierarchy defines the multigrid structure for the current level. Although it is also based on an octree structure similar to the underlying AMR grid, it is a different structure that does not interfere with the coarse AMR levels (which are, in a way, “orthogonal” to the multigrid levels). One major advantage of using this secondary grid hierarchy for each Poisson solve is that the computational cost of the multigrid solver at a given level only scales as the number of cells in that level, as we use only a subset of the AMR coarser cells.

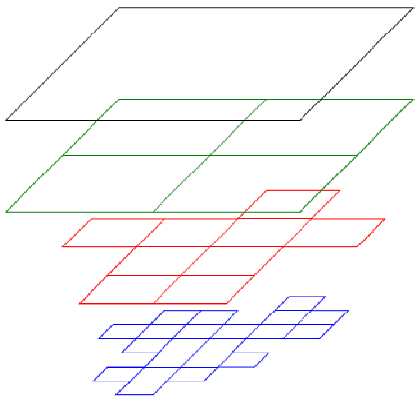

**Figure 5: Levels of the multigrid hierarchy**

Avoiding technical details, the multigrid approach can be interpreted as an iterative method to solve the Poisson equation, accelerated by the usage of a hierarchy of grids where errors at different resolutions are wiped out. This strongly improves the convergence rate to the solution. One of the crucial factors for the quality of the result is a proper reconstruction a) of

the domain at various levels/resolutions, b) the proper setting of the boundary conditions, which is accurately implemented in the code.

The parallel implementation of the gravity solver is based on a domain decomposition approach, splitting the computational domain into sub-regions (gravity patches), which are each managed and updated by a dedicated computing core. Such spatial domain decomposition techniques rely on the ability to update each CPU independently first, then address the couplings between different domains. In RAMSES, this last step is implemented using buffer regions (see Figure 6).

Each computing core manages its own cells, but also possesses a local copy of cells from other neighboring CPUs which are needed for local computation. These buffer cells need to be updated after every iteration. The update operation is done by communicating the updated values of the buffer cells from the CPUs which own them to the buffer regions in other processors. Therefore, any CPU only communicates with its direct neighbors, and the number of neighbors usually remains small. Moreover, the number of buffer cells scales only as a surface term, limiting the transfer to computation volume ratio.

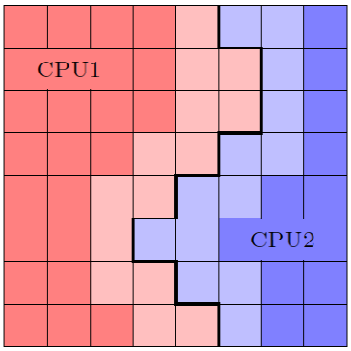


**Figure 6: Domain decomposition and buffer regions as used in RAMSES,**

Just like for the Hydro section, the data patches with all the necessary boundary data, at different levels, are built at the beginning of the step. In this case, however, the aforementioned features of the multigrid algorithm make communication less demanding, ensuring a better scalability of the kernel and a good efficiency over a large number of processors.

The two approaches proposed for the Hydro section, hybrid OpenMP+MPI parallelization and exploitation of accelerators, are expected to be effective also for the Gravity kernel. Communication is less demanding, but it can be further reduced by the usage of large shared memory nodes, that can manage larger patches, minimizing the need for internode data exchange. With respect to the Hydro section, also shared memory access is less intense, improving the performance of the OpenMP parallelization that therefore can be efficient up to the number of cores available on the node.

Once the patches are built, all the subsequent calculation is efficiently vectorizable, since it is based on local finite volumes algorithms and linear interpolation operators. Therefore each patch can be potentially solved effectively on an accelerator. For the Gravity kernel, however, the memory management is critical, since the patches are much larger than the AuxBoxes for Hydro and their size cannot be predicted in advance. Therefore, accelerator load cannot be perfectly tuned and this could result in a sub-optimal exploitation of the resource. In other words, only a certain number of patches can be copied to the accelerator memory. Due to the patches' "granularity" this could leave part of the accelerator memory unused, and require a larger number of expensive copy-in copy-out operations.

### *2.1.4 Conclusions*

We have identified the most computational demanding parts of the RAMSES code as the Hydro and the Gravity kernels together with all the related communication. We have described performance features informed by a work analysis in order to roughly predict the benefits that can be achieved by the refactoring of some of the basic algorithms and from changing the adopted parallel paradigm, currently relying on a pure MPI based strategy. A hybrid parallel implementation strategy has been envisaged, in order to exploit multi-core nodes and many-core accelerator devices.

## 2.2 PKDGRAV

PKDGRAV [28] 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. PKDGRAV calculates the gravitational accelerations using the well-known tree-walking procedure of the Barnes-Hut [29] algorithm. Periodic boundary conditions are implemented via the Ewald method [30]. Adaptive time stepping is adopted in order to improve time integration accuracy only where this is strictly necessary.

The main computational intensive parts of the code identified in PKDGRAV are:

- *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 a 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 particles in “empty” volumes).

### *2.2.1 Adaptive timestepping algorithm*

The performance analysis phase [7] showed that, when a single time step approach is adopted (all particles integrated with the same time step), the code is balanced between the three main sections, with good scalability and efficiency. However, particles are all integrated with the same time step, dictated by the dynamical conditions of the N-body system. Hence a large fraction of the particles adopts a time step much shorter than necessary. When adaptive time stepping is used, particles are integrated with their “ideal” time step, longer for particles in dynamically inactive regions, shorter in overdense regions, where forces are strong and dynamics fast. In PKDGRAV, the time step is refined by powers of 2, the timestep at “rung” 0 (the longest) being twice that at rung 1, that is twice that at rung 2 etc. Particles are assigned to the most suitable rung.

The adaptive time step approach, in principle, improves the performances of the code, which can dedicate most of the time integrating a limited number of particles, updating at lower frequency particles that require long time steps (see Figure 7). On parallel systems however this leads to a serious load balancing issue that represents the main source of inefficiency of the current implementation of the code.

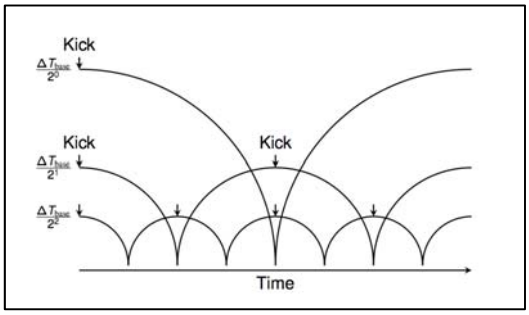


**Figure 7: Multiple time step integration scheme.**

In fact, as more and more time-stepping rungs are needed the cost of a simulation can be completely dominated by the deep rungs. In other words, the simulation cost is dominated by work done on a small subset of the particles (deep particles). This strongly affects parallel efficiency since:

- the compute to communication ratio decreases.
- load balancing is hard, since particles at different rungs are distributed "randomly" on the computational domain. Any optimization on the particle distribution leads to an increase in communication and synchronization.
- more synchronizations are required.

In general, we get about 25% parallel efficiency in typical simulations with adaptive time step due to load imbalance and idle time waiting for data. Effective load balance can be achieved, but the implemented load balancer algorithms tend to dominate the calculation. Furthermore the tree building phase gets ever more imbalanced and also begins to dominate the calculation. A compromise is represented by fully balancing the highest couple of rungs only, suffering imbalance on deeper rungs, but reducing the cost the load balancing.

**Performance improvements**

A first improvement of PKDGRAV performance can be expected by exploiting multi core shared memory systems. In fact, core calculations of PKDGRAV have been written to use "tiles" of vectors whose size can be optimized for OpenMP. The domain decomposition can be performed over nodes as opposed to cores, greatly reducing overhead and increasing parallelization for difficult cases.

Domain decomposition can be calculated for all rungs once at the beginning of a step, providing a map where each particle should be sent at each sub-step. Each rung performs the domain decomposition of its active (i.e. integrated by that rung) and inactive particles separately, thereby getting perfect load and memory balance.

The main drawback of this approach is represented by a heavy communication phase at each sub-step. This has to be carefully designed and optimized.

The same tiling can be exploited to off-load the calculation of gravitational interactions to an accelerator (GPU, MIC) where the calculation of the force per particle is completely local and efficiently vectorizable, leading to a potentially efficient exploitation of the accelerating device. For this, the tile's number and size represent critical parameters, minimizing memory transfer between host and device and leading to an ideal exploitation of the accelerator memory.

A further improvement can be achieved calculating gravity using a specific tree structure for each rung each with its own domain decomposition. This leads to a perfect load and memory balance for all phases of the calculation. Each rung's domain decomposition can be calculated in a time proportional to the number of active particles at that level, and is performed as

frequently as gravity for that rung. Multiple trees would mean more terms on interaction lists, but with the fast multipole algorithm of PKDGRAV it is possible to aggregate terms from different trees efficiently. The main drawbacks are represented first, by a high degree of non-locality of data that could be alleviated by the implementation of local data caches in available memory. Second, such reimplementation requires a deep change in the tree structure, one of the main data structures of the code.

#### *2.2.2 Conclusions*

PKDGRAV is an extremely well engineered software, optimized for HPC, whose main current performance limitations are represented by the adoption of multiple timesteps. The WP8 refactoring effort will focus on two major areas: hybridization with OpenMP, in order to exploit large multi-core systems, and support of accelerators for the calculation of gravitational interactions. Other solutions require a more invasive algorithms rewriting, thus they will be considered only if time and resources permit.

### 2.3 PFARM

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 [15]. Relativistic extensions have been developed and have enabled much accurate scattering data to be produced. The package has been used to calculate electron collision data for astrophysical applications (such as: the interstellar medium, planetary atmospheres) with, for example, various ions of Fe and Ni and neutral O, plus other applications such as plasma modelling and fusion reactor impurities (for example ions of Sn, Co, and in progress, W). PFARM performs the 'outer region' calculation extending from the R-matrix sphere boundary to the asymptotic region in which scattering matrices and (temperature-dependent) collision strengths are produced [15].

PFARM divides configuration space into radial sectors and solves for the Green’s function within each sector using a basis expansion: the BBM method [16]. The parallel calculation takes place in two distinct stages, with a dedicated MPI-based program for each stage. Firstly, parallel sector Hamiltonian diagonalizations 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 computing different sector calculations.

#### *2.3.1 EXDIG Stage*

The first stage of a calculation is dominated by the parallel diagonalizations of large sub-region (or sector) Hamiltonian matrices that are computed independently of scattering energies. The code is structured to take advantage of the parallel symmetric diagonalization methods available in the distributed-memory based numerical library ScaLAPACK. A previous project [17] has improved the performance of this stage by undertaking the *n* Hamiltonian matrix parallel diagonalizations concurrently using BLACS-defined sub-groups of processes. Of the several parallel real symmetric diagonalization routines available in the current release of ScaLAPACK, investigations have found that the ‘Divide-and-Conquer’ based method PDSYEVD performs best, both in terms of performance and scalability [17]. However ScaLAPACK is based on a distributed-memory pure MPI parallel model and therefore may not be best suited to the latest multi/many core or heterogeneous architectures. In order to explore alternative approaches that may yield performance improvements on the latest HPC architectures we suggest that the following novel methods are investigated (and incorporated into the code if appropriate).

**Selected EXDIG Kernels for refactoring and optimization:**

1. Analysis of a newly developed multi-threaded variant of the diagonalization algorithm MRRR called *mr3smp* [18].
2. Analysis of real symmetric diagonalization routines made available through the PLASMA [19] or MAGMA [20] numerical library initiatives, which are designed specifically for multi-core and GPU-based platforms.

### *2.3.2 EXAS Stage*

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 decomposition is shown in the figure below.

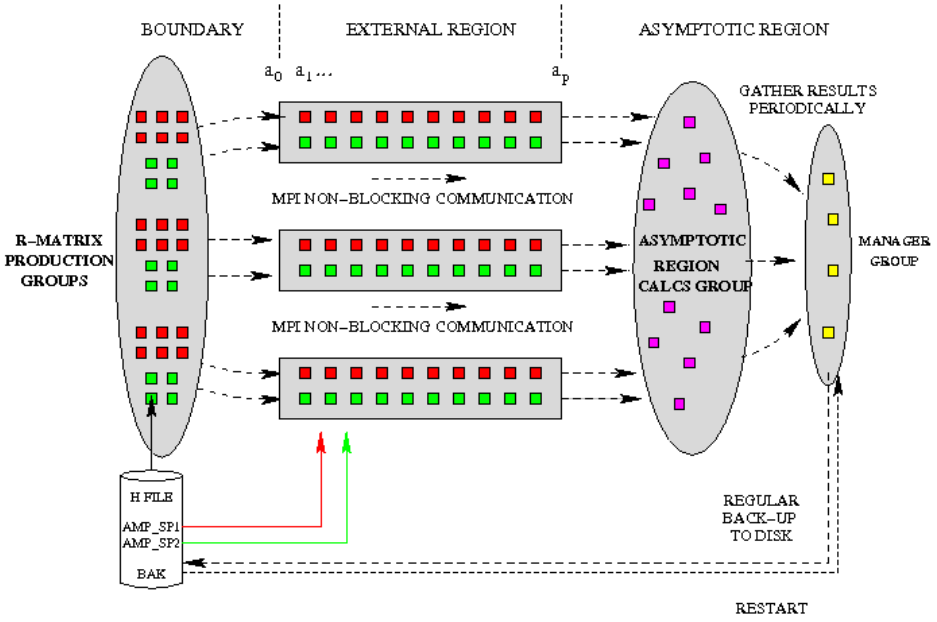


**Figure 8: Example Process Decomposition in the EXAS Stage**

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 [21]. The main priority is therefore to optimize the number of processes dedicated to each task-group, particularly the asymptotic region calculation, and minimize the runtime management (data collection) group whilst ensuring the best achievable load-balancing properties. The processor configuration is currently determined automatically via a Perl script using predictive algorithms for expected performance of each task-group [15], but this is in need of updating for the latest multi/many core and accelerator based architectures.

Many calculations using HPC electron collisions codes involve many different stages of computation involving different scales of computation (e.g. number of cores suitable). Therefore overall performance is highly dependent upon efficient portable, parallel I/O transfer between different numbers of cores and hence different HPC platforms. To help

facilitate these important requirements, the proposed work will involve parallel extension, testing and application of I/O management and wrapper systems recently developed for the R-matrix suite [22] which can automatically handle and switch between local binary and portable XDR formats.

**Selected EXAS Kernels for refactoring and optimisation:**

1. Update of auto-load-balancing script and algorithm for target architectures (0.5 Months)
2. Debugging, testing and further development and parallelization of existing I/O wrapper system. (1 Month)
3. Incorporation of parallel I/O wrapper system into PFARM electron collision codes (a software package containing several parallel programs): adjust automated sub-task core-group assignment algorithm to take account of the efficient I/O as required. (3 Months)
4. Development of self-contained parallel I/O wrapper library package for more general use and distribution. (1 Month)
5. Determination of optimal I/O settings on PRACE systems, benchmarking of optimized codes (i.e. using I/O wrapper library) on PRACE systems. (0.5 Months)

# 3 Climate

In deliverable 8.1.2 [7], seven codes were evaluated and analysed within four different areas of computational climate science:

1. Couplers: OASIS
2. Input/Output: CDI, XIOS, PIO
3. Dynamical cores: ICON
4. Ocean Models: NEMO, ICOM

The performance analysis of these codes already led to initial thoughts on the kernels which should be investigated, and the technologies (hardware, programming paradigms, etc.) which might be employed to achieve a considerable increase in performance. In this section individual kernels, which are representative for the codes as a whole, are selected for intense refactoring for emerging architectures, to which PRACE will soon have access.

## 3.1 Couplers: OASIS

OASIS was reviewed in D8.1.2 as a key component in most large European climate models, and its performance was seen to be a bottleneck. Two solutions are being examined: OASIS3-MCT and OASIS4.

The first is the modification of OASIS-3 to use the Model Coupling Toolkit (MCT [34]), 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.

OASIS3-MCT is currently being tested by CERFACS on the PRACE Tier-0 Bullx computer CURIE on the “toymodel” test up to 2048 cores, with decompositions similar to those used in the EC-EARTH model. This work is at a preliminary stage, with the coupling of the OASIS3-MCT still under development. Profiling has been done on a high-resolution test case with the atmosphere component using a IFS T799 grid and the ocean component using the ORCA 0.25 deg grid. The toymodel performs 120 coupling exchanges.

Initial analysis shows that time taken for toyocn/atm decreases from 1 to 64 cores but then increases. This is because of a poor scaling of the coupling initialization phases and because the initialization here has a relative high cost. Some work should be done to reduce the coupling initialization phases, but in all cases, the cost is small compared to the cost of the real component for a real run of the equivalent duration (i.e. 5 days if there are coupling exchanges every hour).

In the initialisation phases, some time is spent before calling the `prism_enddef_proto` and in the calling to `prism_enddef_proto`. Before calling the `prism_enddef_proto`, most of the time is spent when calling `mct_gsmap`, which defines the coupling map from distributed data. In `prism_enddef_proto`, most of the time is spent in the call to `prism_coupler_setup` and in the distributed read of the NetCDF SCRIP file.

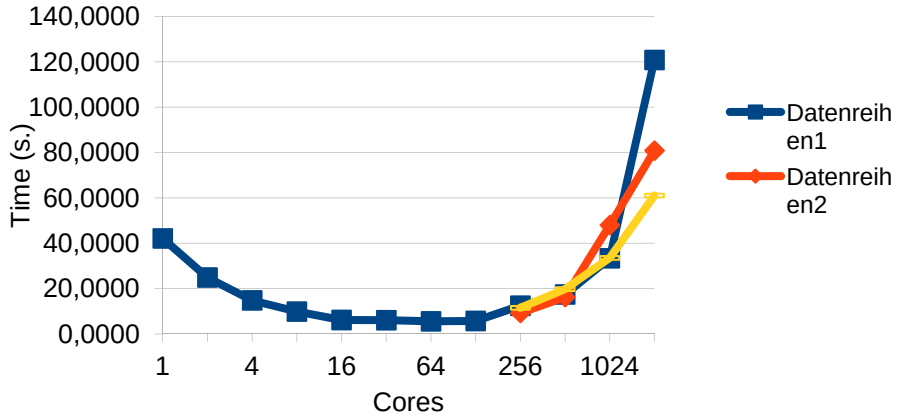


**Figure 9: time spent in OASIS3-MCT initialisation**

Most interesting is the time of the ping-pong exchanges (see Figure 10 )"toyocn/atm : Ping-pong exchange"), which represents the overhead of the coupling exchanges (not taking into account the coupling initialization overhead). This time decreases from 1 to 128 cores but then increases slightly to about 5 sec for 2048 cores. This is mainly the matrix-vector multiplication (for the remapping from the source to the target grid) and communication; an overhead of 5 sec for 120 interpolations + exchanges of 2 high-resolution fields is quite reasonable.

The increase of the time needed for the ping-pong from 128 to 2048 cores is partly linked to the increase of the time required for the matrix-vector multiplication. This is not yet understood, and some time should be devoted to understand why that is.

Hence initial targets for analysis and comparison of OASIS3-MCT and OASIS4, which are conceptually identical in precalculating the gridding, are the scaling of initialisation and time required for the matrix-vector multiplication.

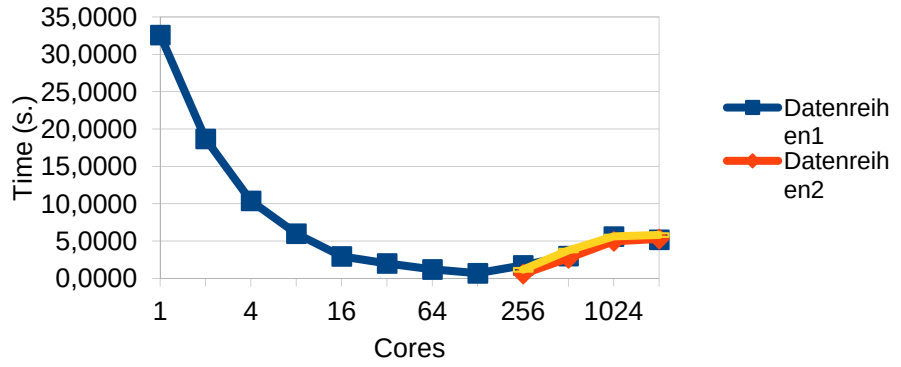


**Figure 10: OASIS3-MCT: Ping-pong (coupling exchange)**

## 3.2 Input/Output: CDI, XIOS, PIO

In D8.1.2 [2] we proposed within PRACE-2IP WP8 to compare several I/O and processing tools, with the aim of proposing and developing efficient generic solutions. The tools to be considered are CDI developed at MPIM in Hamburg, XIOS developed at IPSL in Paris, and PIO developed at NCAR in boulder, USA.

### *3.2.1 Description of Code: CDI*

Climate Data Interface, CDI has been developed at MPI Meteorology in Hamburg to provide an efficient I/O server mechanism for climate codes, in particular for the ECHAM climate

model. It implements a strategy of using separate I/O nodes responsible for doing file writing, with each of the compute nodes storing data in buffers that are RDMA-capable, so that the nodes running the model can copy data to these buffers and continue while the I/O nodes collect this data, do transpositions necessary to orientate arrays for storage, compress records and write them.

Secondly, the CDI interface provides an abstraction allowing a variety of output formats to be used, in particular both NetCDF3/4 and Grib1/2 formats. This is important for models that are used for "seamless" or "unified" prediction, where the same model is used for both weather and climate prediction. This includes models such as the IFS atmosphere model used in EC-EARTH and the UK Met Office Unified Model. For NWP, GRIB output is preferred for size reasons, but in climate modelling, the higher numerical accuracy of NetCDF is preferred. Hence it is desirable that I/O processors are capable of writing both.

In performance tests done to date, CDI has shown linear scaling up to the limits of I/O bandwidth possible at MPIM (30 GB/s total system bandwidth, but 2 GB/s bandwidth per node for 250 nodes). Work is currently underway to measure the scaling of CDI. Scaling is determined by:

- total systems hardware bandwidth,
- single node hardware bandwidth,
- available memory on I/O nodes,
- encoding and compression,
- network jitter for RDMA access,
- memory used on compute nodes,
- memory requirements for post-processing.

Principally then, the scaling of the I/O server system is set by hardware and memory, especially memory for post-processing. Hence full profiling of CDI needs to be done in the context of a plan for what post-processing may be done on the I/O nodes

### *3.2.2 Description of Code: XIOS*

XIOS is the parallel I/O software, written in C++, developed at IPSL for climate codes. This code was described in detail in [2]. The software has two aims: i) a parallel output system, and ii) a simple configuration of model outputs through XML files. A draft version has been developed, mainly to test the main concepts. XIOS Server version 1 is in beta state, with delivery to users this month.

To date no real performance testing has been done with XIOS.

A meeting has been arranged for DKRZ (German Climate Computing Centre) in February to include PRACE partners and the community developers, in particular the CDI and XIOS developers, to investigate the opportunities for a generic I/O module and scaling tests will be planned for PRACE Tier-0 systems.

### *3.2.3 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.

PIO is a mature library. Performance results were discussed extensively in [7]. There are no plans to refactor PIO, but rather to use PIO in a comparative study. Thus the only targeted kernel will be the I/O driver, which will be modified for the PIO API.

### 3.3 Dynamical Cores: ICON

The emerging ICOsahedral Non-hydrostatic (ICON) model was determined [11] to contain the only emerging, near-production European dynamical core, to which software engineers at PRACE centres could actively and beneficially contribute. Of particular long-term interest is ICON's non-hydrostatic (NH) solver, as it will play a key role in versions of ICON using mesh refinement for enhanced resolution over certain geographic regions. Detailed performance analysis and modelling [11] indicated that the NH-solver is entirely memory bandwidth-bound, and that its performance adheres well to the Roofline Model [13] for processor performance, and correlates particularly well with the STREAM benchmark for memory bandwidth. Moreover, initial results [11] of the MPI-parallel NH-solver indicate reasonable strong scaling of a moderately high-resolution test case (roughly 69 km) to 1024 MPI processes, but only modest scaling thereafter. We conclude that NH-solver should be refactored to offer additional levels of parallelism, e.g., multi-threading for (possibly heterogeneous) multi-core platforms.

A feasibility study [13] considered the ICON testbed [11] NH-solver, which has been parallelised for OpenMP multi-threading, but not MPI distributed memory parallelism. The NH-solver was rewritten in OpenCL to offer multi-platform compatibility. The study concluded that the implicit vertical solver, which solves a linear tridiagonal system of equations with the order of the number of vertical levels (approx. 60 columns), was a significant bottleneck to GPU parallelisation. In Figure 11 one vertical column (containing two triangles and one diamond) represents the same kernel run on either an NVIDIA Fermi (medium and high resolution) or AMD Cayman (high resolution). Two kernels stand out with much lower performance than predicted by the roofline model. These have loop dependencies and are both related to the implicit vertical solver.

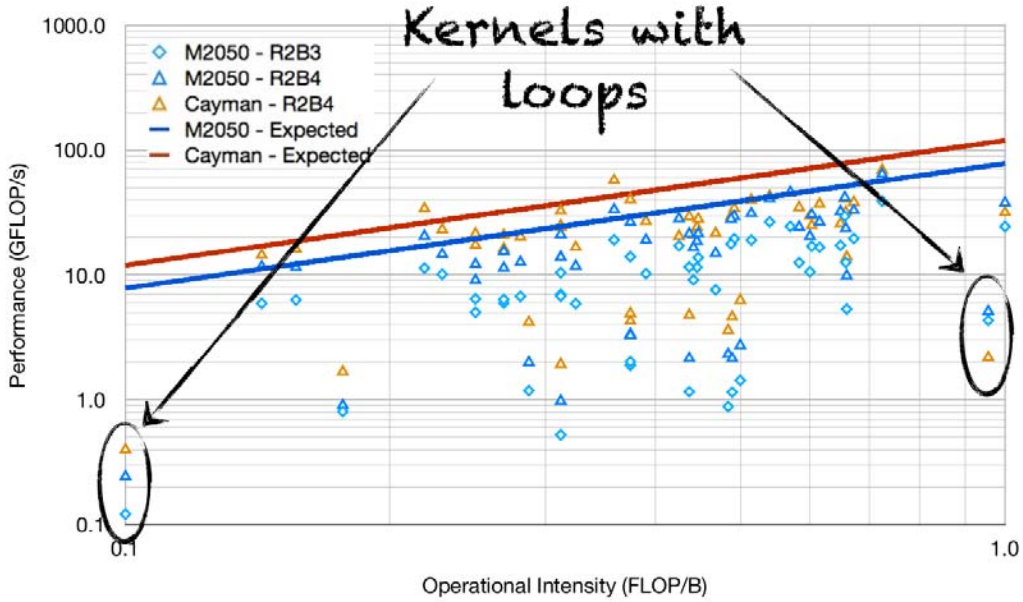


**Figure 11: Performance of individual OpenCL kernels in the ICON NH-solver multi-platform implementation.**

It seems clear that the tridiagonal solver needs to be investigated. One issue is the extensive optimisation of ICON for vector machines, in which higher dimensional temporary arrays are often introduced to avoid data dependencies.

The feasibility study has, however, clearly proven the potential of GPUs for the ICON NH-solver. In contrast to that study, the work in WP8 will be based on the MPI-enabled

development version available through collaboration with MPIM and DWD. From the analysis presented here and in [11] efforts on the following kernels could bring tangible benefits to the ICON project, and thus to the greater community:

1. Firstly, it is would be necessary to clearly define the set of kernels. One option would be to adopt the kernel taxonomy from the feasibility study [13]. Generally each nested loop of the existing solver would roughly map to one kernel. Certain kernels, such as gradient and divergence are already well-defined and currently correspond to subroutines. But for the most part, the current code is a monolithic collection of these loops, so written to perform well on vector architectures.
2. In conjunction with MPIM and DWD, optimisations to the *vertical implicit solver* need to be made, while leaving in the current optimisations for vector architectures in place (possibly utilising CPP directives). Subsequently the gain (or loss) in performance will be evaluated on various CPU and vector architectures.
3. After agreement on the proper programming paradigm for multi-core architectures (e.g., OpenCL or CUDAFortran), the kernels defined in points 1 and 2 would be programmed, optimised and incorporated into the MPI-enabled ICON development code.

## 3.4 Ocean Models

### *2.4.1 NEMO*

NEMO [24] is a widely-used, highly portable numerical platform for simulating ocean dynamics, biochemistry and sea-ice. 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 to get rid of the north pole singularity.

The two key performance-limiting factors for NEMO identified during the profiling stage [7] were the poor scaling of the amount of time spent in MPI and the high memory-bandwidth requirements of its computations. (We leave input/output performance as a separate issue tackled in Section 3.2) Here we consider two strategies for tackling the memory-bandwidth demands – porting the code to make use of a GPU with its greater memory bandwidth and porting the code to OpenMP with the hope of achieving more efficient use of on-chip cache.

Unfortunately NEMO has a rather flat profile [7] and therefore the memory-bandwith issue permeates much of the code. Despite this, we can select kernels for this work according to the relative significance of the various subroutines found during the profiling stage [7] for single node performance. Therefore, we will work on kernels based on the *traldf_iso()*, *lim_rhg()* and *traadv_tvd()* subroutines since these account for 5.1%, 13% and 4.3%, respectively, of wall-clock time when the ORCA2_LIM configuration (global model at 2 degree resolution coupled with the LIM sea-ice model) is run on 12 cores of a Cray XE6 (HECToR [26]). We anticipate that these proportions will remain similar for multi-node runs.

Although *lim_rhg()* is easily the most expensive of these routines, it is likely that this is due to load-balancing as well as to memory bandwidth since the sea-ice properties are only computed by MPI processes which have sea-ice in their domain. For this reason we will also consider the *traldf_iso()* and *traadv_tvd()* routines which deal with the lateral diffusion and advection of tracers, respectively. The first of these is more expensive and has the advantage of being relatively short and free of any MPI calls and therefore should be a straightforward test case. The second brings in the complexity of MPI calls, which will be a key issue for both OpenMP and GPU implementations. In both cases this is because the MPI calls represent thread synchronisation points. In the case of the GPU there is an additional difficulty since, with current technology, the MPI library will only be able to access data in the address space

of the CPU. Consequently, data must be transferred between the CPU and GPU to service the MPI calls. Since the current generation of many-core devices must be accessed over a PCI Express bus, the transfer of data between the CPU and GPU can be a significant bottleneck.

As listed in Table 1, there are currently several ways of writing code for execution on a GPU. Of these, three are specific to NVIDIA hardware and while NVIDIA currently has the edge in GPU hardware, both AMD and Intel are working to develop their offerings in this area. A second key issue is the fact that NEMO is written in Fortran while CUDA and OpenCL are both C-based. The performance benefits of running on a GPU would have to be substantial to warrant the effort required to re-write tried, tested and trusted parts of NEMO in C.

| Approach | Notes | Fortran support |
|---|---|---|
| PGI Accelerator Directives | Currently NVIDIA specific | Yes |
| HMPP Workbench | Can generate CUDA and OpenCL code, will support Intel MIC in 2012. | Yes |
| PGI CUDA Fortran | NVIDIA specific | Yes |
| OpenCL | Portable, open standard | No |
| CUDA C | Widely used, mature and low-level but NVIDIA specific | No |

**Table 1: The available options for programming a GPU.**

As a consequence of these considerations we will use only directives-based approaches in this work since, in principle, they allow the original Fortran code to remain unchanged and also have the potential to generate code for different GPUs. In fact, HMPP Workbench (a product from the French company CAPS Entreprise) can generate either CUDA or OpenCL code from the same instrumented source (including Fortran to OpenCL and Fortran to CUDA/C++ translators).

Inevitably there is a compromise here in that the use of a higher-level programming model often precludes some of the optimisations that might be viable when using a lower-level language such as CUDA. In their work on porting a kernel from the Weather Research & Forecast (WRF) model, the authors compared the performance achieved when using PGI Accelerator Directives with that of a hand-written CUDA version by J. Michalakes (one of the WRF authors). Although the GPU compute part of their initial version was twice as slow as the CUDA version, after manually applying some of the loop transforms used in the latter, it was actually some 26% faster. Including data transfers, their GPU version of the kernel was a factor of 2.7 times faster on an NVIDIA Tesla than on four cores of an Intel Nehalem. We can conclude that using directives is not necessarily a barrier to performance although it may prove necessary to alter the source code in order to take full advantage of the GPU architecture.

Many-core devices are widely acknowledged as being a key building block of future supercomputers. In fact, four of the top ten machines in the June 2011 Top500 List of Supercomputers [27] are already using some sort of many-core accelerator. It is therefore essential to assess the potential of this hardware for a code such as NEMO so as to inform future development effort on both it and any successors. From the work in [23] it seems reasonable to expect a speed-up in the region of two to three for a given kernel on a GPU as compared to a version using all of the cores on a CPU. Creating OpenMP versions of the

kernels will both enable realistic comparisons with GPU performance as well as inform the best way of making use of the available memory bandwidth on the now ubiquitous many-core CPUs. It also appears that OpenMP is one of the ways in which the new Intel MIC architecture can be programmed and will therefore provide a way of testing performance on this new approach to many-core hardware.

### *3.4.2 Fluidity-ICOM*

Fluidity-ICOM is an open-source partial differential equation simulator for unstructured meshes used in a diverse range of geophysical fluid flow applications. Fluidity-ICOM uses state-of-the-art and standardised 3rd party software components whenever possible. For example, PETSc 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.

Typically current computer systems are based on interconnected multi-core processing units, each of which has a non-uniform memory access (NUMA) architecture. Such multicore clusters provide a natural environment for the mixed-mode programming paradigm, whereby OpenMP can be used for sharing data between the cores that comprise a single node and MPI can be used for the communication between computing nodes.

Previous benchmarking [7] has already shown that the two dominant simulation costs are sparse matrix assembly (30%-40% of total computation) and solving the sparse linear systems defined by these equations. Recent work by the Applied Modelling and Computation Group (AMCG) at Imperial College, and Fujitsu Laboratories of Europe Limited (FLE) has found that the Hypre library's hybrid sparse linear system solvers/preconditioners, which can be used by Fluidity-ICOM through the PETSc interface, are competitive with the pure MPI implementation on Intel's Woodcrest processors (4-way nodes). There is also an ongoing project on hybridizing PETSc with MPI and OpenMP. Therefore, in order to run a complete simulation using mixed mode parallelism, sparse matrix assembly is now the most important component remaining to be parallelised using OpenMP.

**Performance Improvement: Matrix assembly kernel**

While improving I/O is not a direct objective of this work plan, significant benefit can be expected from using mixed-mode parallelism. For example, only one process per node will be involved in I/O (contrast with the pure MPI case where potentially 24 processes per node could be performing I/O on Phase 2b of HECToR), which will significantly reduce the number of metadata operations on the file system at large process counts. In addition, the total size of the mesh halo increases with number of partitions (i.e. number of processes). It can be shown empirically that the size of the vertex halo in a linear tetrahedral mesh grows as $O(P^{1.5})$, where $P$ is the number of partitions. Therefore, the use of hybrid OpenMP/MPI will decrease the total memory footprint per compute node, the total volume of data to write to disk, and the total number of metadata operations given Fluidity's files-per-process I/O strategy.

For a given simulation, a number of different matrices need to be assembled, e.g. continuous and discontinuous finite element formulations for velocity, pressure and tracer fields for the Navier-Stokes equations and Stokes flow. Each of these have to be individually parallelised using OpenMP. Parallelism can be realised through well-established graph colouring techniques, where the graph defines the data dependencies in the matrix assembly. This approach removes data contention, so called critical sections in OpenMP, allowing very efficient parallelisation.

The procedure is as follows:

1. Calculate the dual graph mesh from the input mesh. This is only done once.
2. Create the sparsity pattern from the dual graph mesh for the different finite-element type schemes of the different terms in the equations. For instance, the upwind scheme for the advection term and the Compact Discontinuous Galerkin scheme for the diffusion term. This will produce the element dependency graph.
3. Using the greedy colouring algorithm, colour the dual graph according to the sparsity of the adjacency matrix of the graph. If the vertices are ordered according to their degrees, the resulting greedy colouring uses at most one more than the graph's maximum degree. (This heuristic is sometimes called the Welsh–Powell algorithm.)
4. Add a loop over colours outside the OpenMP *parallel* region. Within the *parallel* region there are no data dependencies in the calculation, which would hinder efficient execution.

One of the key performance considerations for achieving performance on ccNUMA nodes is memory bandwidth. In order to optimize memory bandwidth, we will employ the following methods to ensure good performance:

- First touch initialisation and other methods to ensure that page faults are satisfied by the memory bank directly connected to the CPU that raises the page fault;
- Thread pinning to ensure that individual threads are bound to the same core throughout the computation.

Typically all of the above must be implemented in order to minimise data traffic between CPU's and non-local memory banks. With the above procedure, we expect that using pure Open-MP will give better performance than pure MPI within one node. With threaded PETSc, we are expecting improved scalability for Fluidity-ICOM.

# 4 Material Science

## 4.1 ABINIT

ABINIT [37] is a package, delivered under the GNU General Public Licence (GPL), 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. 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.

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. In recent years, a development of wave functions utilising 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*". 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.

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 work in progress.

As for the performance analysis phase [7], this “performance improvements exploration” phase is divided in three sections: 1-*ground-state calculations using plane waves*, 2-*ground-state calculations using wavelets*, 3-*Excited states calculations*.

### *4.1.1 Performance improvement: ground-state calculations using plane waves*

During the performance analysis phase [7], we identified the critical parts of the code:

- LOBPCG algorithm and diagonalisation/orthogonalisation of wave functions: this routine solves the eigenvalue problem by minimisation using LOBPCG algorithm
- Hamiltonian application: this routine applies Hamiltonian H (and overlap matrix S) to the wave-functions, divided in local operator, non-local operator and communications.
- Forces: this routine computes forces on atoms.

Several other sections have been investigated, but proved to be negligible compared to the previous sections. When looking at the code performances increasing the number of CPU cores, it clearly appears that the main obstacle to scalability is the application of *LOBPCG* algorithm. Only this part will be investigated here.

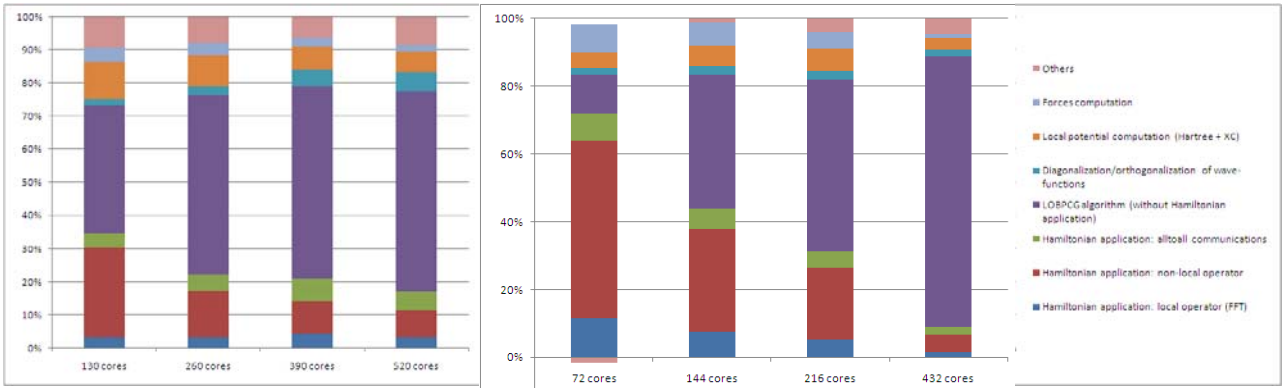


**Figure 12: Repartition of time in ABINIT routines: on the left: varying the number of plane-wave CPU cores; on the right: varying the number of band CPU cores.**

#### 4.1.1.1 Eigenvalue problem (LOBPCG) improvment: Prototype code improving load balancing; Prototype code using openMP parallelization.

The Locally Optimal Block Preconditioned Conjugate Gradient Method (LOBPCG) is an algorithm for finding the smallest eigenvalues and the corresponding eigenvectors of a symmetric positive definite generalized eigenvalue problem.

As shown in the Figure 12, the time spent in the LOBPCG algorithm become predominant; a first (but deep) analysis of the corresponding code section demonstrated that this is essentially due to: 1- an unbalanced load, 2-communications in orthogonalizations/diagonalizations.

In order to improve the performances, both the load balancing must be improved and the communication overhead must be reduced.

The envisaged prototype codes are the following:

*1- Unbalanced load correction*

The unbalanced load is due to both “band” and “plane wave” distribution. In some disadvantageous cases, some cores can have a load 1.75 times larger than others. Plane wave vectors are incorrectly distributed among processors. Modifying their repartition, according to the physical system, can change this. Only a minor modification at the level of the distribution routine is required, but this has to be done with subtlety. The expected improvement depends on the treated physical system but can reach 50% on some disadvantageous cases.

**A prototype code applying the non-local Hamiltonian using a better band/plane wave distribution will be tested.**

*2- Communications in diagonalizations/orthogonalizations*

The analysis shows that the implicated communications mainly are reductions in the orthogonalization and eigenvalue solvers. As there is no hope in decreasing the size of the matrices, the most promising evolution seems to be the use of shared memory parallelism… and *openMP* is the most natural choice. This choice is reinforced when looking at the architecture of some of the *PRACE* nodes. The TGCC-CURIE computer has a “large-nodes” partition with possibly 128 cores on each node.

**A prototype code of the orthogonalization routine using *openMP* will be tested.**

#### 4.1.1.2 Use of Graphics Processing Units improvement: Prototype code using GPU activation thresholds

Use of Graphic Processing Units (GPU) will be available in the 6.12 version of ABINIT; this implementation is in beta stage. It uses NVIDIA *CUDA* library and is still evolving. Some additional "free of use" packages have been also linked: *NVidia cuBlas*, *NVidia cuFFT*, *MAGMA*.

The GPU implementation is fully compatible with all MPI levels of parallelisation except at the plane wave level. Performance tests show that it is only efficient when the computational load is large enough, i.e. when big physical systems are treated (large simulation cells and/or heavy materials). It is also demonstrated that the efficiency of the GPU code strongly depends on the performance ratio between CPU and GPU.

In order to improve the performance, an envisaged modification is the (automatic) activation of the use of GPU code sections when thresholds are reached. This concerns the *FFT* and the *LaPACK* calls. For these two different parts, the thresholds are necessarily different. This implementation will be based on the automatic determination of thresholds according to the CPU/GPU architecture (only CPU/GPU ratio is important, taking transfers into account). This could be made by calling small FFT or *LaPACK* routines at start simultaneously on CPU and GPU and comparing the obtained performances. Then, according to the physical system size, the code could decide to use CPU or GPU version of the algorithms.

**A prototype code using an automatic process to determine whether GPU has to be used for *FFT* or *LaPACK* will be tested.**

### *4.1.2 Performance improvement: ground-state calculations using wavelets (BigDFT)*

*BigDFT* uses three level of optimization:

- *MPI* over orbitals for coarse parallelization,
- *OpenMP* for each orbital for fine optimization,
- *OpenCL* to accelerate the code by means of GPUs.

As in the plane wave version of ABINIT, we would first improve the *OpenMP* optimization of *BigDFT* especially for the exchange-correlation potential calculation (*LibXC*). This task is straightforward and should give some significant performance imporvements for small simulated systems. This will represent a gain between 10 and 20% depending on the number of threads.

**A prototype code calculating the exchange-correlation potential with openMP directives will be tested.**

The second task would be the improvement of the Poisson solver used both by *BigDFT* and ABINIT. The Poisson solver is based on *FFT* (Fast Fourier Transform) in order to calculate long range solution of the Poisson equation. The second task would be the use of *OpenCL* to calculate *FFT* on GPUs in order to accelerate the Poisson solver. This will represent a small gain for large systems but a substantial gain for small systems. The main interest is to have an *OpenCL* version of *FFT* is to accelerate also the exchange-correlation calculations in the specific case of *hybrid functionals*. In this case, the gain will be substantial for large systems. The calculation of the exchange-correlation for hybrid functionals represents 90% of the time. Using GPUs, we will reduce the execution time by a factor of 6. This will give a gain of 4 for the whole code.

We point out that this speedup could be also realised by means of *OpenMP* directives. The advantage of using GPU is important if we can use both *OpenMP* and GPU for different operations in the code. This will be realised in future developments.

This *OpenCL* FFT could be also used in future developments for the excited states calculation which use a large number of FFTs.

**A prototype code will be tested with *OpenCL FFT* for all long range convolutions.**

#### *4.1.3 Performance improvement: excited states calculations*

*GW* calculations are very CPU and memory demanding due to the large number of empty bands that are usually needed to converge the *quasiparticle* corrections. For this reason, the *GW* code of ABINIT distributes the full set of bands at the beginning of the run such that each Processing Unit can compute locally its own partial contribution without having to exchange data with the other MPI nodes involved in the run. Most of the computation is distributed among the node and a few collective MPI communications are required to gather the final results.

The preliminary tests performed at the Barcelona Supercomputing Center have revealed that both the computation of the polarizability and of the *self-energy* matrix elements scale well up to 512 processors. The degradation of the total speedup observed for large number of processors is mainly due to:

- The reading of the orbitals from the KSS file
- The matrix inversion performed during the computation of the screening.
- An unbalanced distribution of the work during the calculation of the exchange part of the self-energy.

The envisaged prototype codes are the following:

1- Reading of the orbitals

In order to achieve better scaling in the I/O part, we plan to replace the plain Fortran-IO implementation with a new version based on MPI-IO.

**A prototype code using collective MPI-IO routines to read the orbitals will be tested.**

2- Matrix inversion

The matrix inversion represents a serious bottleneck for large-scale applications since the CPU time quickly increases with the number of atoms.

**A prototype code using *ScalaPACK* routines to to perform the inversion will be tested.**

3- Unbalanced distribution

The unbalanced distribution of the work in the calculation of the exchange part of the self-energy can be avoided by resorting to a different distribution of the orbitals such that each node can participate in the computation.

**A prototype code employing a different MPI distribution of the orbitals will be implemented and tested.**

Another point worth stressing is that the present implementation is best suited for crystalline compounds where symmetries can be used to reduce the number of independent matrix elements that have to be computed explicitly. Isolated or disordered systems can be treated by means of the supercell technique but, in this case, the implementation is not optimal as the

non-scalable arrays whose size increases with the number of atoms will dominate the memory requirements.

To overcome this limitation we plan to use a hybrid MPI-OpenMP implementation in which MPI is used at a coarse level in order to distribute the most memory-demanding arrays while *OpenMP* is employed for the fine grain parallelization.

**A prototype code employing *OpenMP* to parallelize the most CPU intensive parts (FFT, matrix algebra, loops,…) will be implemented and tested.**

## 4.2 Quantum ESPRESSO

Quantum ESPRESSO [38] is a distribution of software for atomistic simulations based on electronic structure using density-functional theory with a plane-wave basis set and pseudo potential. Main packages are PWscf (PW), a self-consistent electronic structure solver, and CP, a variable-cell Car Parrinello molecular dynamics package.

The flowchart of the PW code is shown in the figure below. This is the typical flowchart of a density-functional calculation: in the self-consistency cycle, once the wave functions are calculated from the input, the diagonalization of the Hamiltonian matrix is performed for each k-point. As we showed in D8.1.2, this is certainly the most expensive part of the self-consistency cycle. Then the charge density is calculated and from that new potentials are generated.

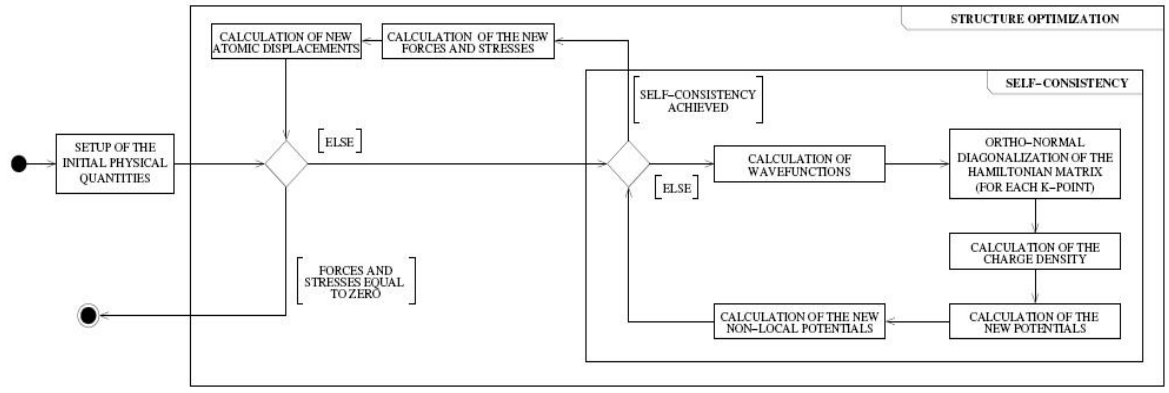


**Figure 13: Schematic UML activity diagram of PWscf code.**

The computational cost of the diagonalization of the hamiltonian matrix is strictly related to the physical properties of the system under investigation. PW implements two different methods to achieve this task: a Davidson method and the Conjugate Gradient method. Within this project we aim to focus on the Davidson method since it is widely used also in other Quantum chemistry codes.

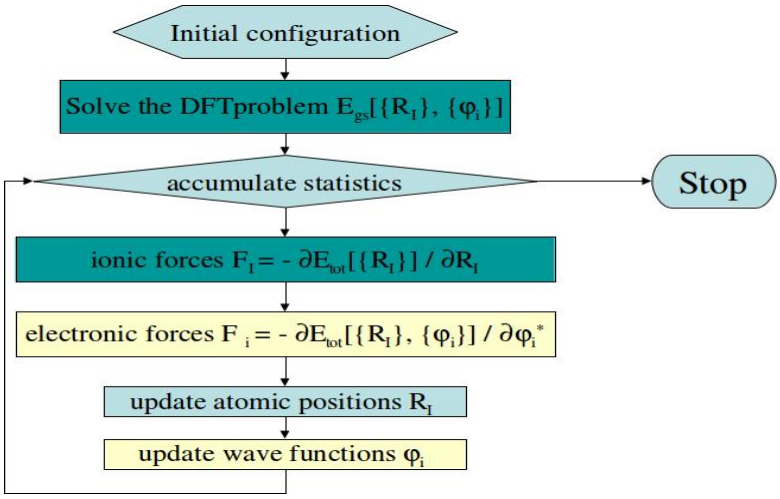


**Figure 14: Flow diagram of the CP code implemented in the Quantum ESPRESSO suite.**

The CP code (schema above) calculates the molecular dynamics of a system following the Car Parrinello schema. After the solution of the density-functional problem, for each time step along the trajectory, ionic and electronic forces should be calculated in order to evolve the dynamics of the system. In this case, as we showed in the previous deliverable, a relevant part of the computation effort is due to the evaluation of forces. Here, a significant issue is related to the computation of the 3DFFT. The second main bottleneck, in terms of scalability is given, as for PW, by linear algebra (including a matrix diagonalization) involved in the wave functions orthogonalization process (not displayed in the figure, but computed in each step).

### *4.2.1 Performance Analysis Resume*

Potential bottlenecks in both PW and CP are strictly related to some physical quantities of the system that affect both array and matrix dimensions (see table below).

| Id | Brief Description |
|---|---|
| N_w | Number of plane waves used in wavefunction expansion * |
| N_g | Number of G-vectors (used in potential/charge density expansion)** |
| N_i | Dimensions of the FFT grid. Where i = 1,2,3... |
| N_a | Number of atoms in the unit cell or supercell |
| N_b | Number of (Kohn-Sham) states (bands) used in the calculation. |
| N_p | Number of projectors in non-local pseudopotentials (summed over the cell) |
| N_k | Number of k-points in the irreducible Brillouin Zone |
| * | N_w ~ V (size of the unit cell) |
| ** | N_g ~ N_1 x N_2 x N_3 - N_g ~ *a* N_w<br>with *a*=8 for a norm-conserving pseudopotential and *a*=20 for a ultrasoft. |

The main common computational engines of those codes are modularized such that high performance across different architectures is achieved by the systematic use of standardized mathematical libraries such as BLAS, LAPACK, ScaLAPACK and FFTW (see table below):

| Problem | Current Implementation |
|---|---|
| Calculation of density, n(r) | FFT + Linear algebra (matrix-matrix multiplications, matrix size ~ N_w x N_a*N_p ) |
| Calculation of potential $V(r)=V_{xc}[n(r)]+V_H[n(r)]$ | FFT + operations on a real-space grid |
| Iterative diagonalization (PW); electronic force calculation (CP), H\psi products | FFT + linear algebra (matrix-matrix multiplications, matrix size ~ N_w x N_b ) |
| Subspace diagonalization (PW); iterative orthonormalization of Kohn-Sham states (CP) | Diagonalization of matrices + matrix-matrix multiplication, matrix size ~ N_b x N_b |

Although the performance of the key engines (PWscf and CP) was demonstrated to run and scale efficiently on massively parallel computers up to thousands of processors, the previous document [7] underlined how both the diagonalization of the Hamiltonian matrix and FFT computation become main bottlenecks when increasing the number of distributed nodes.

### *4.2.2 The state of the art*

While the current G-space data distribution is better exploited during subspace diagonalization (PWscf) or iterative orthonormalization (CP), it does not represent the best solution while performing a number of 3DFFTs across distributed G-vectors. A band-index parallelization would certainly design a schema performing better on massive parallel architecture. Pre and post data reordering does not reduce the amount of communication but it would allow performing each single 3DFFT locally. This solution has never been implemented in Quantum ESPRESSO but it gains strength when considering a huge increase of compute density into heterogeneous computing nodes, as this is one of the recognized faithful road-maps toward the next generation of supercomputers [31].

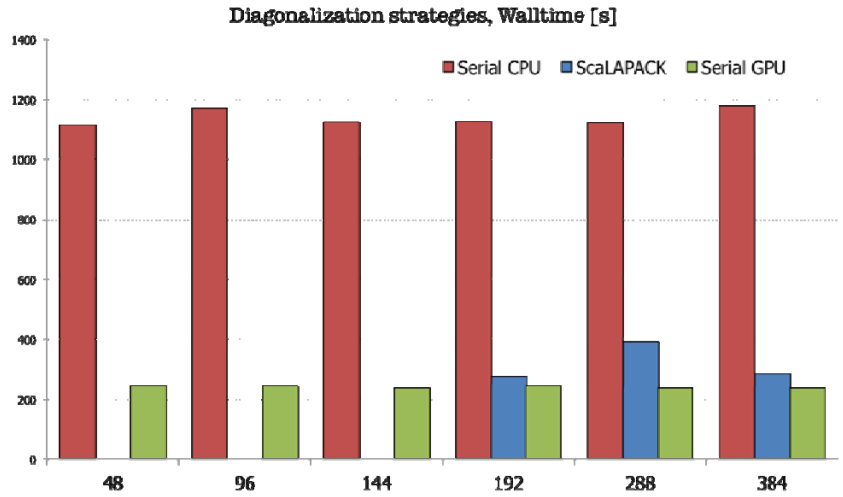


**Figure 15: Compute time spent to solve eigenvalue problem comparing single node libraries such as LAPACK (red), MAGMA (green-hybrid) with ScaLAPACK (blue-pure MPI). On the x-axis and y-axis the number of cores used for PWscf calculation and the time in seconds are shown.**

Subspace diagonalization (PWscf) or iterative orthonormalization (CP) require linear algebra operations (eigenvalue/eigenvector solvers and matrix-matrix multiplications) on square N_b × N_b matrices, where N_b is the number of electron states (or a small multiple of it). By

default this algorithm is performed using the LAPACK library but to exploit architectures with a small amount of memory on each node (for large systems N_b ~ several thousands), ScaLAPACK parallel diagonalization routines are available to perform this operation on distributed data.

While the scalability limitations of the ScaLAPACK library are well know this schema usually demonstrated better performance than the use of LAPACK within a single CPU node. This perspective certainly changes if we consider the use of MAGMA/PLASMA which allows us to take full advantage of hybrid accelerated nodes (see Figure 15). These preliminary numbers certainly require a deeper analysis but already show how the use of accelerators can be considered an option to improve performance. The substitution of the Davidson algorithm with an indirect subspace iteration method or the use of a conjugate gradient approach are considered two other possible options to overcome this problem. In particular, the latter method is already implemented within the CP module. However this would return to what is reported on the previous paragraph: the need to improve the efficiency of the 3DFFT calculations.

The most recent version of PWscf has been ported to NVIDIA GPUs. This version takes advantage of the phiGEMM developed at ICHEC. The phiGEMM library extends the NVIDIA version to support matrix-matrix multiplication on single-precision, double-precision and complex matrices in both transpose and conjugate forms along with other features described later. In [32], it is reported how the use of phiGEMM library can have an outstanding impact on the scf calculation.

## 4.3 Yambo

Yambo [36] 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.

During the performance analysis phase a very clear bottleneck was identified: the matrix inversion step, as can be seen in Fig. 58 of D8.1.2. The use of ScaLAPACK does not seem to bring any scaling improvements, although some runtimes are smaller than when using standard LAPACK (Fig. 59 of D8.1.2). The matrix inversion procedure used in the code, with either LAPACK or ScaLAPACK, is based on LU factorization (LAPACK routines zgetri and zgetrf). In order to improve the performance of this step of yambo, a new method is therefore needed. Several alternatives can be considered:

1. Use of GPUs, i.e., use of GPU versions of zgetrf and zgetri. This implies that the entire matrix is sent to the GPU, which might pose a problem for large matrices (a fairly typical 9000x9000 complex matrix occupies 1.2 GB).
2. Use of MAGMA for the LU factorization. This approach has the same drawbacks as the previous one.
3. Replace LU-factorization based Gaussian elimination by Gauss-Jordan elimination, that is essentially a refinement/operation reordering of the former.

The proposed course of action is to start by implementing proposal 2. above, as this requires minimal coding and proceed to develop and implement a multicore and/or GPU aware version of Gauss-Jordan elimination.

## 4.4 Siesta

As the performance analysis in [7] clearly showed, by far the most important kernel to work on is the diagonalisation. Currently Siesta [39] uses routines provided by the ScaLAPACK library. It suffers from two disadvantages:

1. It is meant for dense matrices, so the amount of memory needed is unnecessarily high.
2. It scales badly for large numbers of processors (see Figure 16).

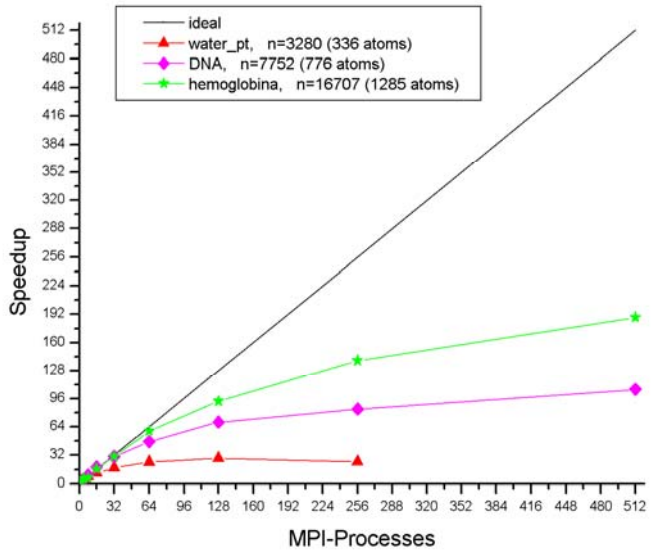


**Figure 16: Scaling of the ScaLAPACK GEV solver for different matrix sizes**

Thus an algorithm exploiting sparseness and featuring good scaling is needed. The latter can be achieved by using parallelism on different levels. A method providing this is the Sakurai-Sugiura algorithm. The most costly part of this method is the solution of a set of linear equations, based on the Hamiltonian and overlap matrices. These systems are independent, thus they can be solved very efficiently in parallel. First tests showed that for current problems the number of linear systems can reach several hundreds. As Figure 17 shows, the efficiency of ScaLAPACK running on hundreds of processes has already dropped dramatically.

Moreover, for each of these systems a parallel solver can be used. Many libraries for this purpose already exist, based on several technologies:

1. MPI offers the possibility to use an arbitrary number of processes
2. Using OpenMP for example a whole node can be used for solving one system, taking advantage of shared memory
3. Also GPU-based solvers can be used. Tests with CUSP [43] showed promising performance. Due to the parallelization on other levels this method provides a good opportunity to use many GPUs in parallel efficiently.

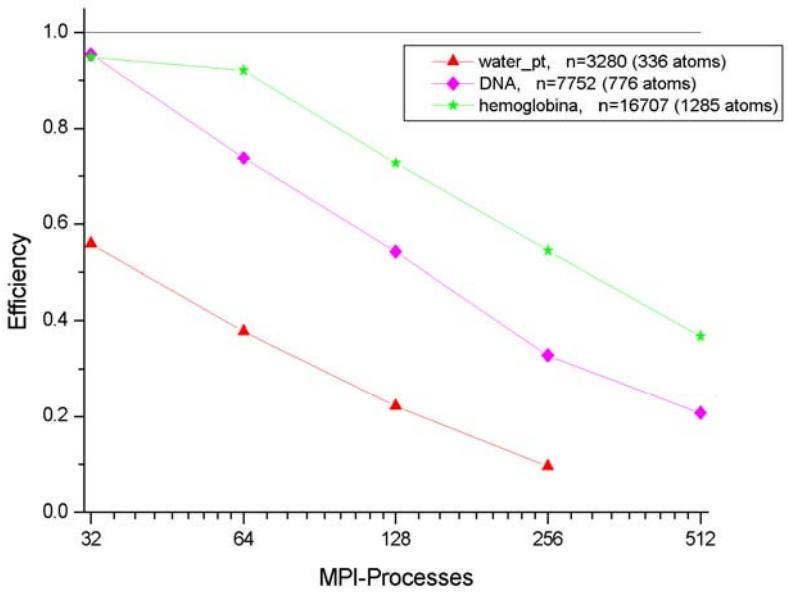


**Figure 17: Efficiency of the ScaLAPACK GEV solver for different matrix sizes**

Already a simple, by far not optimised prototype implementation of this algorithm reaches the solution time of ScaLAPACK. A better performance for larger problems solved on many processes is expected for the Sakurai-Sugiura method due to its superior efficiency, which depends mainly on the linear solver used.

## 4.5 Octopus

Octopus [35] is a real-space, real-time computer code to simulate the dynamics of electrons and nuclei under the influence of external time-dependent fields in the framework of Time-Dependent Density Functional Theory (TDDFT). The real-time propagation of the TDDFT equations frees the code from the limitations imposed by perturbation theory and enables it to model systems under very strong external fields.

During the performance analysis phase, two different bottlenecks were identified:

1. In ground state mode, parallelization in states requires a considerable amount of communication due to the orthogonalization and subspace diagonalization procedures that mix different states.
2. 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. One of its components, the Hartree potential, is obtained solving Poisson's equation. However, the Poisson solver does not scale with the number of processors, becoming the most time consuming step of the simulation for large numbers of processors.

### *4.5.1 Performance improvement: State-parallelization ScaLAPACK Kernel*

Dealing with different states in different nodes implies a large amount of communication between nodes due to the need of orthogonalizing the different states and also due to the use of subspace diagonalization in the eigensolver, mixing states located in different nodes. If one wants to do parallel linear algebra, basically the only general tool that one can use (in FORTRAN) is ScaLAPACK. As a first attempt at tackling this problem, one should fully implement the use of ScaLAPACK in this part of the code, test it and eventually refine the implementation. The expected performance gain would only be visible when dealing with large systems in large numbers of processors.

However, ScaLAPACK is complicated, hard to use, and in the end it is not well integrated with MPI, which causes problems to attach it to any large code. There is an excellent solution to this, PETSC, but this is not really a library, but a framework. Therefore, it is hard to use in a large project. The ideal solution would be a real library as flexible as the lower levels PETSc, as complete as ScaLAPACK, but easy to use. But as a first step, one might not need to fully replace ScaLAPACK. The main hassle is BLACS that was designed to be an interface for different communication libraries but that in fact is very difficult to use with MPI. For example, it is very difficult to use a subset of the processors in ScaLAPACK, since BLACS expects all processes to call some function. A simple solution would be a very thin reimplementation of BLACS on top of MPI with topologies (BLACS assumes a 2D topology).

The implementation of ScaLAPACK was already started by the code developers, but it is still very crude and was not benchmarked nor fine-tuned. We plan to finish this implementation and refine it according to the results of benchmarks. If the performance gain is insufficient we will follow the path of reimplementing BLACS.

### *4.5.2 Performance improvement: Poisson Solver Kernel*

There are at least two different roads to address the Poisson solver problem: solving Poisson's equation in reciprocal space using a truly parallel FFT library and using a fast multipole method. The first solution is the most promising and the easiest to implement. There is a drawback: using, e.g., Juelich's PFFT library, the real-space grid has to be partitioned in a way that renders it extremely inefficient for the stencils used to compute numerical derivatives in other parts of the code. This means that two different partitioning schemes will have to coexist and a grid re-partitioning step will have to be taken after each call to PFFT. This is a communication-intensive step that can degrade the performance to the point of rending PFFT impracticable. Thus, this is the real bottleneck that needs to be addressed.

We expect the fast multipole method (FMM) to be efficient only for very large numbers of processors/grid points. The PFFT method, on the other hand, should be the method of choice for intermediate numbers of processors/grid points. Thus, both methods should be implemented and the code should advise the user on the best method for the problem at hand. The implementation of FMM is straightforward, and has already been started by the code developers. We will collaborate with them on testing and improving this implementation. PFFT, however, is a different story. We are already working on its implementation and we started by re-writing all the routines that deal explicitly with the partitions in order to allow for different partitions to coexist. Next, we will address the re-partitioning in order to avoid the expensive "all to all" communications that are used in the current version of the code. This will imply keeping track of the two partitions to which each point in the grid belongs – one partition for PFFT and a different partition for the other parts of the code – and communicate only with the nodes that will handle the partitions that intersect the partition handled by the current node. A further refinement will be taking in account the need for the PFFT partition when defining the standard partitions. This would reduce the number of partition intersections and thus the communication overhead.

# 5 Particle Physics

Based on the performance analysis given in D8.1.2 [7], we continue our discussion here of the kernels which we focus on improving. We describe those kernels and what they do and give a description of our plan to improve their performance and initial rough estimates of the improvement gain expected based on the performance analysis carried out in the previous deliverable.

## 5.1 Improving the tmLQCD package

In this part we work on the tmLQCD package. Improving this code includes: improving the overlap between scheduling of communications and computations in the Dirac operator, implementing a hybrid MPI/OpenMP parallelism approach and implementing deflated iterative solvers for multiple right-hand sides.

### *5.1.1 Improving the Dirac operator*

Application of the Dirac operator or Dirac matrix to a lattice vector is a fundamental operation in lattice QCD codes. This matrix involves nearest neighbor terms that requires communication between processes which is usually implemented using MPI. In addition, it has loops that go over all lattice sites. Optimization of this operation is very important for the overall performance of the code. For Twisted-Mass Lattice QCD, the matrix elements of the application of the Dirac operator on a lattice vector is given by:

$$\phi(x) = (m_0 + 4 + i\mu_q\gamma_5)\psi(x) - \frac{1}{2}\sum_{\mu=1}^{4}[U_\mu(x)(1+\gamma_\mu)\psi(x+a\hat{\mu}) + U_\mu^h(x-a\hat{\mu})(1-\gamma_\mu)\psi(x-a\hat{\mu})]$$

where $\psi(x)$ is the input source vector at lattice site *x*. At each lattice site *x*, $\psi(x)$ is actually a 12 component vector corresponding to the three quark colors and four Dirac spin indices. $\gamma_\mu, \gamma_5$are constant 4 by 4 matrices acting on the spin degrees of freedom and *U(x)* is a 3 by 3 complex matrix which gives the value of the gauge field link at site *x* and they are elements of the SU(3) group. $U^h$ denotes the Hermitian conjugate of *U*. Note that *U* acts only on the color degrees of freedom. The lattice spacing is $a$ and $\hat{\mu}$ is a unit vector in the $\mu$ direction. The parameters $m_0$ and $\mu_q$ are the two mass parameters for the Twisted-Mass action.

To compute $\phi(x)$ we have two parts: the first part is just a scaling of $\psi(x)$ by a constant number and doesn't require any communication between neighboring sites. The other two terms involve the source vector at neighboring sites $x + a\hat{\mu}$ and $x - a\hat{\mu}$ which requires the communication of boundary sites between processes. This second part is implemented in a user function called *Hopping_Matrix(ϕ,ψ)*. We have seen in our performance analysis that about this Hopping_Matrix function accounts for about 80% of the total time spent in user-defined functions. Because of this, we focus our improvement effort on the optimization of this function. In Figure 18, we show an example profile with a many node test for $L = 48, T = 96$ lattice.

Before discussing the optimization of the *Hopping_Matrix* function, the following important points are implemented in the tmLQCD code:

- In many situations, the Dirac operator needs to be applied to a lattice vector O(1000) times without changing the gauge field *U*. This is the case for iterative solvers such as the Conjugate Gradient algorithm. In such cases, only the boundary sites of the lattice vector need to be exchanged between applications of the Dirac operator.
- In spinor space, the operators $1 \pm \gamma_\mu$ are projection operators which project a 4 component spinor into two component spinors. So, $(1 \pm \gamma_\mu)\psi(x)$ has two independent

spinor combinations. Using this fact, half of the size of the boundary sites of a lattice vector need to be exchanged.

- In order to reduce cache misses, an additional copy of the gauge field is stored such that and looping is done over the input vector . This way, the gauge field is accessed in the same order as needed by the Dirac operator.

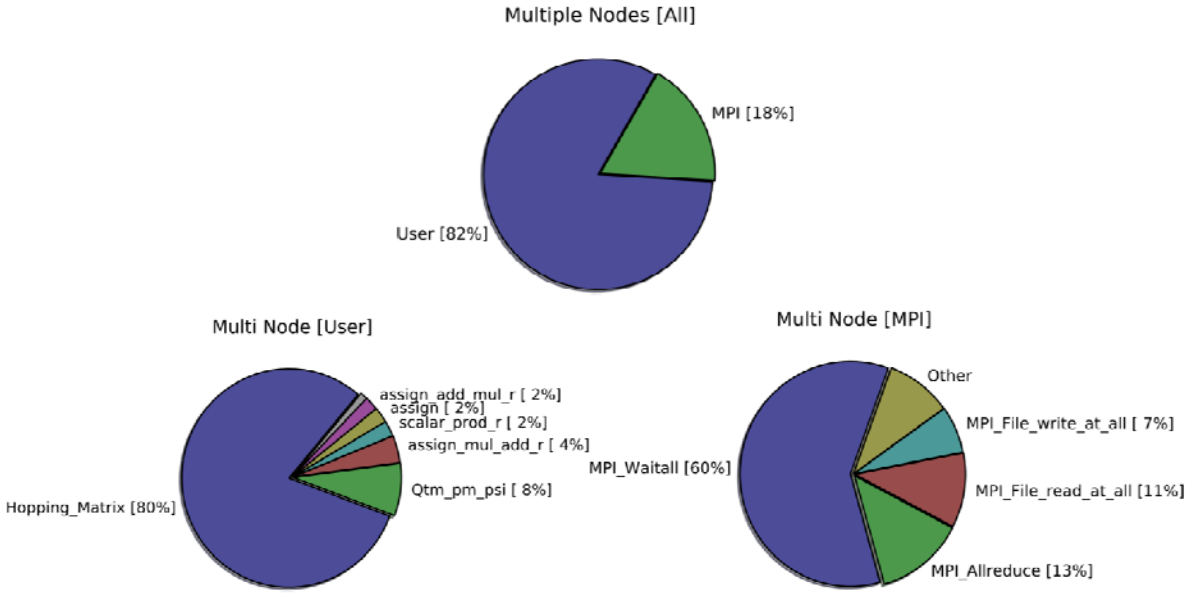


**Figure 18: Profiling of the twisted mass inverter code on 24 nodes. The chart in the centre shows User and MPI functions with respect to the total time. The left chart is a break-down of the User functions (percentages are with respect to the total time spent in User functions) and the right chart is a break-down of the MPI functions (percentages are with respect to the total time spent in MPI functions)**

### 5.1.1.1 Optimization of the communication/computation schedule:

The profiling results showed that the call to the blocking *MPI_Waitall* consumes a large fraction of the total time spent in MPI functions (about 60% for this multi-node case). This is related to the way computation and communication is scheduled inside the *Hopping_Matrix*. Currently, this is implemented as follows:

- compute for all sites *x*.
- compute for all sites *x*.

  where are projectors from a 4 component spinor into a 2 component spinor. The half-spinor fields are computed locally on each node and no communication is needed because of the existence of the gauge field copy.

- Exchange boundaries of and . This part is done by calling the user function *xchange_halffield*. Note that even though communications are done using the non-blocking *MPI_Isend* and *MPI_IRecv*, in effect the communication of the boundaries is blocking, because no computations are done before the associated *MPI_Waitall* is posted. The function *xchange_halffield* exchanges the boundaries as follows:

  for each direction :

  1. Send boundaries to the neighbor and receive boundaries from neighbor.
  2. Send boundaries to the neighbor and receive boundaries from neighbor.
  3. Wait for all 16 communication processes to finish (MPI_Waitall).

- The result is then computed as

  $\phi(x) = (m_0 + 4 + i\mu_q\gamma_5)\psi(x) - \frac{1}{2}\sum_{\mu=1}^{4}[P_{\mu}^{2-4}\phi^{L}(x + a\hat{\mu}, \mu) + P_{-\mu}^{2-4}\phi^{R}(x - a\hat{\mu}, \mu)]$for all sites x, Where $P_{\pm\mu}^{2-4}$ are the reverse projectors from half-spinor to full-spinor. This step requires only computation and no communication.

The above schedule is not optimal and we suggest to improve it by overlapping communication with computation. There is more than one way to achieve this. One example is as follows:

- Compute $\phi(x) = (m_0 + 4 + i\mu_q\gamma_5)\psi(x)$ for all sites *x*.
- For $\mu = 0,1,2,3$, send boundaries to the $+\mu$ neighbor and receive boundaries from $-\mu$ neighbor.
- For $\mu = 0,1,2,3$:
  1. Call MPI_wait for the send to the $+\mu$and receive from the $-\mu$ requests for this $\mu$.
  2. Compute $\phi(x) = \phi(x) - \frac{1}{2}P_{-\mu}^{2-4}\phi^{R}(x - a\hat{\mu}, \mu)$ for all sites x.
  3. Send boundaries to the $-\mu$ neighbor and receive boundaries from the $+\mu$ neighbor.
- For $\mu = 0,1,2,3$:
  1. Call MPI_Wait for the send to the $-\mu$ and receive from the $+\mu$ neighbors.
  2. $\phi(x) = \phi(x) - \frac{1}{2}P_{+\mu}^{2-4}\phi^{L}(x + a\hat{\mu}, \mu)$ for all sites x.

This schedule has the potential of reducing the wait time for the communication to complete and is expected to reduce the MPI time used by the *Hopping_Matrix*.

#### 5.1.1.2 Using a hybrid MPI/OpenMP parallelism:

Apart from modifying the communication schedule, we will investigate benefits of including thread-level parallelism for on-node communications. This can potentially have a double benefit to the code:

- The memory footprint of the kernel will be lower because less boundary buffers shall need to be allocated. Currently the code treats every process as having a single NUMA domain, and so every core requires temporary buffers for MPI communications. A shared memory approach for the intra-node communications, using OpenMP, will alleviate the memory requirements of the kernel, especially as the number of cores sharing the same memory space increases in future architectures.
- Although this depends on the MPI implementation, thread-level parallelism for the intra-node communications could save on the overhead required to set-up such data transfers via MPI. Certain MPI implementations use buffering of communications, and some even allocate these buffers with every communication call. Therefore parallelizing intra-node communications with OpenMP could benefit the total run-time of the code.

### *5.1.2 Algorithmic Improvements*

For completeness, we include here three algorithmic improvements which are being

investigated:

- **Deflated Iterative Methods:** Standard iterative solvers such as Conjugate Gradient or BiCGStab become slow as the quark mass approaches zero. Deflated iterative methods such as GMRES-DR, Eig-CG, or Domain-Decomposition methods have not been sufficiently tested for the case of the Twisted-Mass action. We plan on performing an evaluation of these methods and implement those that are not currently implemented in this package. Initial testing of the Eig-CG solver showed a speedup in the solution time of a factor of 2 on a L=24, T=48 Lattice.
- **Using Poisson brackets to tune Hybrid Monte Carlo integrators:** The standard algorithm to generate lattice configurations with dynamical quarks is Hybrid Monte Carlo (HMC). The most time consuming part of the HMC method is the molecular dynamics (MD) step, where we evolve the system using some approximate integrator. In general, the integrator contains free parameters, which can be tuned such that the acceptance rate is maximized, while keeping the step size as large as possible. The best way to do this is using Poisson bracket measurements. Depending on how well tuned the default integration scheme is, the optimization of the integrator allows to decrease the number of the inversions of the fermionic matrix needed by HMC. New integrator schemes can be tested (such as force-gradient integrators), hopefully providing further improvements.
- **Parallel Landau and Coulomb gauge fixing:** On the lattice, Landau and Coulomb gauge fixing is performed using a local optimization method, such as Steepest Descent. Such methods suffer from critical slowing down: the number of iterations needed to solve the problem increase with $V^z$, where V is the size of the problem (in our case, the lattice volume). For gauge fixing, naïve procedures have an associated critical exponent z around 2. A Fourier-accelerated method has been proposed (with z~0), but it requires the use of Fast Fourier Transforms (FFT), whose parallelization is far from being trivial. The standard and well-known FFTW package (www.fftw.org) provides parallel routines which only parallelize along one dimension. We therefore want to explore other possible FFT packages allowing further parallelization, such that we can consider the use of more processors.

# 6 Conclusions and next steps

In this deliverable, the performance modelling approach was adopted in order to characterize the performance of a number of numerical kernels. These had been identified in community codes as the most critical to effectively utilise novel HPC architectures. In the current deliverable, we described the main algorithmic features of these kernels and the possible solutions towards the next generation of HPC systems, emphasising those aspects that have to be subject of the final performance modelling stage. This final stage, which will be reported in deliverable D8.1.4 “Plan for Community Code Refactoring”, will allow a quantitative estimation of the possible performance improvements on coming HPC systems and will define the approaches to take in the refactoring effort.

In some cases, as for Astrophysics, the expected improvements affect specific codes, being designed on their peculiar algorithmic features. The impact is therefore vertical on single applications, and the corresponding user community can have a significant immediate advantage. However, the acquired experience can then be spread to similar algorithms and numerical approaches adopted by other software, extending its impact to a much broader community.

For other codes, as in the case of Material Science, features and needs common to various codes have emerged, and the refactoring work on corresponding kernels can be shared and exploited by different applications. For example, linear algebra solvers, in particular for the calculation of eigenvalues and eigenvectors, are critical for most of the codes selected in this domain. Therefore, the experimentation and adoption of specialized libraries, tuned on innovative architectures, replacing the traditional solvers, can be highly profitable for all these applications. On the other hand, such a wide and heterogeneous spectrum of applications represents for these libraries a valuable test-bed for their further consolidation and for the tuning of their performances.

Furthermore, a valuable achievement is the dramatic improvement in know-how, resulting from experiencing new HPC solutions, both hardware and software, on “real” codes (and not simplified testing tools) in “production” conditions (as opposed to artificial benchmark cases).

Finally, we stress how the current performance modelling is successful thanks only to a consolidated synergy between the scientific community code developers, with their knowledge of codes and algorithms, and the HPC experts, with their competencies in HPC systems and software, parallel programming and algorithms optimisation.

At the end of the performance modelling phase, all the information necessary to define a precise work plan for code refactoring will be available. The specification of the targets, the partners’ roles, the timeline and the milestones for each code will be the subject of the next step of WP8, to be accomplished by M6. A further, but not less relevant, goal for the next phase is represented by the specification of a testing and validation procedure, necessary to verify the correctness and quality of the results produced by the re-implemented kernels (fast algorithms producing wrong results are obviously useless…). This is a crucial step to certify the quality of the work to the scientific communities, and to permit them to introduce and accept the new kernels into the official production distributions. This represents the final main objective of WP8.