D3.1– Kernel Selection Criteria and Assessment Report
Version 1.0

Document Information

<table>
<thead>
<tr>
<th>Contract Number</th>
<th>288777</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Website</td>
<td><a href="http://www.montblanc-project.eu">www.montblanc-project.eu</a></td>
</tr>
<tr>
<td>Contractual Deadline</td>
<td>M8</td>
</tr>
<tr>
<td>Dissemination Level</td>
<td>RE</td>
</tr>
<tr>
<td>Nature</td>
<td>Report</td>
</tr>
<tr>
<td>Coordinator</td>
<td>Massimiliano Culpo (CINECA)</td>
</tr>
<tr>
<td>Contributors</td>
<td>All Mont-Blanc Partners</td>
</tr>
<tr>
<td>Reviewers</td>
<td>Jean François Lemerre (BULL), Sylvie Lesmanne (BULL), Taner Akgun (BSC)</td>
</tr>
<tr>
<td>Keywords</td>
<td>Kernels, OmpSs</td>
</tr>
</tbody>
</table>

Notices: The research leading to these results has received funding from the European Community’s Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 288777.

©2011 Mont-Blanc Consortium Partners. All rights reserved.
Change Log

<table>
<thead>
<tr>
<th>Version</th>
<th>Description of Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>v0.1</td>
<td>Initial draft released to the European Commission</td>
</tr>
<tr>
<td>v0.2</td>
<td>1st proposed draft (r7 in the SVN repository)</td>
</tr>
<tr>
<td>v0.3</td>
<td>1st contributions from the partners of the consortium (r35 in the SVN repository)</td>
</tr>
<tr>
<td>v0.4</td>
<td>Version reviewed by the internal reviewer (r41 in the SVN repository)</td>
</tr>
<tr>
<td>v1.0</td>
<td>Version to send to the EC</td>
</tr>
</tbody>
</table>
Contents

1 Introduction  4

2 Assessment criteria  4

3 WP4 applications: prospective kernels  5
   3.1 EUTERPE  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  5
   3.2 SPECFEM3D  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  7
   3.3 MP2C  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  8
   3.4 BigDFT  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  11
   3.5 Quantum ESPRESSO  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  14
   3.6 PEPC  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  17
   3.7 COSMO  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  20
   3.8 BQCD  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  24

4 The “Hydro” kernel  25

5 Selection of the final set of kernels  27

6 Conclusion and next steps  29
1 Introduction

The following document reports on the selection of the performance-critical kernels to be ported to the OmpSs [5] programming model during the course of the project. This work, started within WP3-T3.1 and now continuing in WP3-T3.2 and WP3-T3.3, pursues on the one hand an increased performance and portability of the kernels themselves due to the shift of paradigm from a serial or thread-oriented model to a task-based model supported by an efficient run-time scheduler. On the other hand it should devise a set of best-practices to provide WP4 colleagues with helpful guidelines when porting full applications.

The document is organized as follows. In Section 2 the assessment criteria used for the selection of the final set of kernels will be briefly described. Then, in Section 3, most of the applications considered in WP4 will be analyzed, their profiling will be discussed and the potential candidates for the final set of kernels will be proposed. In Section 4 Hydro, a mini-application that is not part of WP4 but will be included in the final set of kernels, will be presented in detail. The decision to include this kernel into the final set is a consequence of its extreme portability and efficiency. In fact, this code has been benchmarked on many platforms and with many different paradigms, and will thus prove to be an excellent mean of comparison against the state-of-the-art technology for both the OmpSs programming model and the prototype ARM cluster. In Section 5 the final set of kernels will be presented. Finally, in Section 6 the conclusion stemming from this first piece of work will be drawn and perspectives on the next steps to be undertaken will be given.

2 Assessment criteria

Within the scope of this document a kernel is considered to be:

A portion of a program that is well-defined from the algorithmic standpoint and presents a clear pattern of communication and/or computation.

The main idea behind this definition is to use kernels as a mean to capture common requirements for classes of programs, avoiding as much as possible the unnecessary complications in the data structures that are related to any particular implementation. Providing neat and lightweight vehicles to express algorithms, kernels will indeed be of help when trying to devise best practices related either to performance-tuning on novel architectures or to programming model shifts.

As the main objectives of WP3 are the analysis of the programmability of the overall system and the evaluation of the impact of the OmpSs programming model within the compute-node, an ensemble of assessment guidelines has to be defined to be used for the selection of the final set of kernels. More specifically, it has been decided that the main criteria for the evaluation of a kernel should be based on:
1. the number of operations performed within the kernel,
2. its capability to represent a wide class of HPC-applications,
3. the load it imposes on the underlying computation/communication infrastructure.

The first criterion has been employed to ensure that kernels will be large enough to be efficiently taskified with OmpSs, avoiding any unnecessary performance penalty due to the run-time scheduling overhead. The second criterion addresses the impact that the work to be performed in T3.2 and T3.3 will have, as it tends to maximize the number of codes that could possibly benefit from the best practices learned when porting and optimizing the kernels. Finally the last criterion excludes from the list of possible kernels the ones that are e.g. I/O intensive, focusing instead on the ones that require a fair amount of overlap between computations within a node and communications among nodes.

3 WP4 applications: prospective kernels

This section collects the preliminary profilings of the applications considered in WP4. Based on the criteria presented in Section 2 possible kernels stemming from each application are proposed.

3.1 EUTERPE

EUTERPE [14] is a particle-in-cell (PIC) gyrokinetic code for 3D global non-linear simulations of fusion plasma instabilities, in particular in tokamaks and stellarators. The code was created at Centre de Recherches en Physique des Plasmas (CRPP), and subsequently has been further developed at Max Planck Institut für Plasmaphysik (IPP). It is written in Fortran (32k lines) with a few C preprocessor directives. The code uses the freely available libraries FFTW (Fastest Fourier Transform in the West) and PETSc (Portable, Extensible Toolkit for Scientific Computation). It is parallelized using MPI. In addition, OpenMP parallelization has been introduced to EUTERPE v2.61 at Barcelona Supercomputing Center (BSC).

Profiling results A performance analysis of EUTERPE has been carried out on MareNostrum at Barcelona Supercomputing Center (BSC), Spain. The time breakdown of the most important parts of the code for two test cases is given below. The small case uses

---

1 Though WP3 will not take into account I/O intensive kernels, work in this direction will be performed by WP3 within the course of the project.
2 Three applications that are part of WP4 (YALES2, SMMP and ProFASI) were not ported or profiled in due time and therefore will not be part of this report.
3 MareNostrum consists of 2560 JS21 blade computing nodes, where every blade has two dual-core IBM 64-bit PowerPC 970MP processors at 2.3 GHz running on Linux, with 8 GB of RAM and 36 GB local disk storage.

---
Figure (1): Trace of \textit{Euterpe} for one time step, explicitly showing the three most time-consuming sections of the code discussed in the text.

32 cores and a $32 \times 32 \times 16$ grid. The large case is for 256 cores and a $1024 \times 1024 \times 64$ grid. As can be seen from the list, the three most important sections of the code account for over 64% of the total execution time. Their overall weight increases as the problem size is increased. A sample trace showing the three sections is given in Figure 1.

<table>
<thead>
<tr>
<th>Description</th>
<th>Small test</th>
<th>Large test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Particle push</td>
<td>40%</td>
<td>20%</td>
</tr>
<tr>
<td>Charge/current calculation on the grid</td>
<td>&gt; 20%</td>
<td>&gt; 40%</td>
</tr>
<tr>
<td>Field solver</td>
<td>4%</td>
<td>&gt; 20%</td>
</tr>
<tr>
<td>The sum of the three above</td>
<td>&gt; 64%</td>
<td>&gt; 80%</td>
</tr>
</tbody>
</table>

**Prospective kernels** A number of small and frequent operations in the most time-consuming sections of the code corresponding to the particle push, the charge/current calculation on the grid, and the field solver have been identified as potential candidate kernels:

- getfield: gets the potential and electric/magnetic field vector
- cache.sort: sorts the marker arrays for better cache reuse
- bcpart: imposes boundary conditions
- equil.grad: calculates gradients
- equil.xy2sc: coordinate transformations
- equil.n: gets species density from linear interpolation
- equil.t: gets species temperature from linear interpolation
- equil.absb: gets the magnitude of the magnetic field
- equil.phieq: specifies derivative of external electric potential
Figure (2): Profiling of the MPI version of SPECFEM3D when executed with 4 processes. Green color corresponds to the process_element_with_Deville function, purple corresponds to the assemble_MPI_StarSs function and red to the remaining functions which have been inlined.

Some of these kernels (in particular related to matrix manipulations, coordinate transformations, and gradient calculations) may be generalized and could be potentially useful in other applications.

3.2 SPECFEM3D

SPECFEM3D [10] is an application that models seismic waves propagation in complex 3D geological models using the spectral element method (SEM). This approach, combining finite element and pseudo-spectral methods, allows the formulation of the seismic waves equations with a greater accuracy and flexibility if compared to more traditional methodologies. The original code has been developed in C during the course of the TEXT project [3] and has been parallelized with MPI and StarSs.

Profiling Results An MPI only binary has been generated by compiling and linking the code with the MPI compiler/linker, excluding StarSs support. This binary has been executed on MareNostrum using the following configuration values for a 4 processors run:

- \text{NEX}_\text{XI} = 64
- \text{NEX}_\text{ETA} = 64
- \text{NPROC}_\text{XI} = 2
- \text{NPROC}_\text{ETA} = 2

SPECFEM3D mainly consists of an iterative code that updates a global matrix. Although its sources are divided in different functions, the profile in Figure 2 shows only the two of them that are annotated within the traces (the others having been inlined at compile-time). These two representative functions are:

- \text{process_element_with_Deville}: using, in average, 55% of the total time,
Figure (3): Average time spent for each thread inside SPECFEM3D routines when executed with 4 MPI processes.

<table>
<thead>
<tr>
<th>Thread</th>
<th>Time (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 1.1.1</td>
<td>1,382.58</td>
</tr>
<tr>
<td>Thread 1.2.1</td>
<td>1,185.08</td>
</tr>
<tr>
<td>Thread 1.3.1</td>
<td>1,231.08</td>
</tr>
<tr>
<td>Thread 1.4.1</td>
<td>1,153.42</td>
</tr>
</tbody>
</table>

Total: 4,952.14 us
Average: 1,238.03 us
Maximum: 1,382.58 us
Minimum: 1,153.42 us
StdDev: 87.90 us
Avg/Max: 0.90

Figure (4): Paraver visualization showing the MPI calls used by each thread and the communication lines between them.

- **assemble_MPI_StarSs**: spending another 27% of the total time.

The average time spent by each thread in each function is reported in Figure 3, showing a clear imbalance within the MPI functions, due to a slower execution of the routines in the first node. This behavior should be evaluated more thoroughly. The communication pattern appears in Figure 4 showing that each iteration communicates with the previous and following iterations.

**Prospective kernels** The application itself is already a kernel from the main Fortran version [1].

**3.3 MP2C**

MP2C [6, 16] (Massively Parallel Multi-Particle Collision) is a scalable parallel program which couples Multi-Particle Collision Dynamics (MPC) with Molecular Dynamics.

---

4Even the function is called `assemble_MPI_StarSs` it simply uses MPI without StarSs.
Figure (5): Scaling of a small MP2C run from 1 to 8 processes using 17280 MPC and 1331 MD particles performing 1000 time steps.

(MD). MPC is a particle-based method to model hydrodynamic effects on a mesoscopic scale. It is a local algorithm, which performs random collisions between particles on a cell level, requiring only the knowledge of particle properties from spatially local neighboring particles. The collision algorithm is energy and momentum conserving and therefore reproduces hydrodynamic effects on larger spatial scales. Coupling this method to standard schemes allows to take into account e.g. hydrodynamic interactions between solvated molecules. The algorithm consists basically of two steps: (i) a streaming step, where particles are propagated ballistically in space according to their actual velocity and the applied time step; (ii) a collision or momentum transfer step, where collisions between multiple particles are performed. Different variants of MPC vary in the way the momentum exchange between particles within a collision cell is performed. Collision cells are organized in a grid which is stochastically shifted over the system in every step.

The current version of MP2C implements an MPC variant which is called Stochastic Rotation Dynamics (SRD). In this method, the collision step is performed such that the velocity relative to the center-of-mass velocity inside a locally defined collision cell is rotated by a given angle $\alpha$ around a randomly chosen axis.

In addition to the hydrodynamics, MP2C is able to simulate short-range interactions based on Molecular Dynamics (MD). It is possible to simulate different kinds of potentials and bonds between particles. The MD part can be used as a stand-alone version or can be coupled to the MPC part. This coupling allows to study the dynamics of e.g. polymer chains including hydrodynamic interactions, or driven particle laden flows.

Profiling Results In order to investigate the scaling behavior of MP2C we used two different test cases. The first is a small system, composed of $N_{MD} = 11^3$ MD particles with initial positions on a regular grid, dissolved in an MPC fluid with $N_{MPC} = 1.7 \times 10^4$ particles. This small example was chosen as a test for low computational load, when using up to 8 processes on the cluster. As a second example a configuration consisting of $N_{MD} = 2.5 \times 10^4$ MD particles and about $2.5 \times 10^7$ MPC particles in a box of dimensions
Figure (6): Scaling of a larger MP2C run from 1 to \( \frac{1}{64} \) processes using \( 2.5 \cdot 10^7 \) MPC and \( 25 \,000 \) MD particles performing 1000 time steps.

75 × 75 × 450 length units was chosen, which resembles more a production run of MP2C. It can be seen from Figure 5 that the small example scales badly due to the low computational load per process. Nevertheless, tests on other machines exhibit a better scalability suggesting a larger overhead in the communication between processes on Tibidabo. Although the larger example in Figure 6 shows a much better scaling, it can be anticipated that also this behavior will start to saturate for larger number of processes. 

A time breakdown of MP2C code on Tibidabo with 25000 MD particles and approximately \( 2.5 \times 10^7 \) MPC particles showed the following results:

- **MD+MPC total:** \( 2.09778 \times 10^2 \) secs = 100.00000%
- **MPC total:** \( 1.04494 \times 10^2 \) secs = 100.00000% [49.81181%]
- **particle move:** \( 1.09483 \times 10^1 \) secs = 10.18824%
- **setup of cells:** \( 1.19476 \times 10^1 \) secs = 11.22586%
- **list construction:** 8.64066E+00 secs = 8.09277%
- **com-vel of cell:** 1.18389E-01 secs = 0.00272%
- **rel. vel. in cell:** 4.83804E+00 secs = 4.41274%
- **random rotat. axis:** 1.1385E+00 secs = 1.06062%
- **random rotation:** 1.01517E+01 secs = 9.44071%
- **system values:** 6.91000E+00 secs = 5.27714%
- **thermostat:** 7.43435E+00 secs = 43.89513%
- **parallel cell (fwd):** 7.43435E-01 secs = 0.45247%
- **parallel cell (bwd):** 8.43002E-01 secs = 0.65644%
- **parallel exchange:** 7.64234E+00 secs = 7.31189%
- **MD total:** \( 1.82146 \times 10^1 \) secs = 100.00000% [8.65369%]

---

\(^5\)Results for this test case are only available with 8 and 16 processes on Tibidabo. For reasons still under investigation, the program stopped when running with 32 processes.
As it can be seen, the majority of the runtime is spent within the MPC part of the code, especially in the thermostat routine. This routine is important for controlling a desired temperature when performing non-equilibrium simulations and spends most part of the time sorting particles into a fixed cell structure and calculating local thermal energies or non-gaussian distributed random variates.

Prospective kernels Since the thermostat routine consists of several different tasks that can be performed in parallel, it is definitely a suitable candidate for kernel extraction. Within the routine the particles have to be sorted into the cell structure and the kinetic energy has to be calculated. Depending on the number of degrees of freedom as well as on the local kinetic energy value, a random number from a $\chi^2$-distribution is calculated according to an acceptance-rejection method.

3.4 BigDFT

BigDFT [7] is an ab-initio simulation software based on the Daubechies wavelets family. The software computes the electronic orbital occupations and energies. It is written mainly in FORTRAN (121k lines) with part in C/C++ (20k lines) and it is parallelized using MPI, OpenMP and an OpenCL support. Several execution modes are available, depending on problem under investigation. Cases can be periodic in various dimensions, and use K-points to increase the accuracy along periodic dimensions. Test cases can also be isolated. Those different modes show various computing behaviors. In the context of this study we used a periodic test case.

BigDFT has been profiled using a standard ZnO super-lattice containing 8 atoms. As stated this test case, representative of a lot of systems studied using BigDFT, is periodic in every direction and contains 36 orbitals (which limits its parallelism to 36 MPI processes). In order to gather profiling results, the code has been instrumented using Extrae and PAPI. Extrae traces are analyzed with Paraver.

Profiling Results The scalability of BigDFT on the specified test case is shown in Figure 7. As it can be seen the performance is poor, with a speedup of 16 for 36 MPI processes. The second profiling result is shown in Figure 8. It shows the time spent within different functions of BigDFT (bottom) and the time spent communicating (top).
It is clear from the graphs that global communications are sometimes delayed for a fixed amount of time. These important delays explain the poor scalability of the code. Below the time spent in the different functions of *BigDFT* is reported:

- Preconditioning: 27%
- Wave-Function Transposition (communication, all to all v): 21%
- Projectors Applications: 16%
- Density Computation: 8%
- Kinetic Energy: 8%
- ...

**Prospective kernels**  Figure 8 reveals that most of the time is spent in the *Preconditioning* subroutine, which is composed of a lot of different operations representative of *BigDFT* wavelet approach. This kernel is thus a good candidate for the selection in WP3.1. Nonetheless, it does not contain communications, so we propose a selection of this kernel plus the two *Wave-Function Transposition(s)* and the Cholesky kernel in-between, as shown in Figure 8.

Another kernel that can be of interest is a small kernel, frequently used by bigger routines, called *magicfilter*. Its pseudo code is shown in Figure 9. This kernel can be rather sensitive to optimization as seen in Figure 10, where we unrolled the outer loop by a factor varying from 1 to 12. The impact on performance is quite important, providing an improvement by a factor of 2.5 compared to the non-unrolled version. Several other optimizations could be applied to this simple case, and parallelization attempts may be done using OmpSs as soon as a parallel for implementation will be

![Figure (7): Scaling of BigDFT from 1 to 36 cores using a ZnO 8 atoms test case.](image)
Figure (8): Traces of BigDFT gathered using Extrae and displayed using Paraver. The top graph represents communications (in orange) while the bottom graph represents the time spent in various functions. A proposed kernel is composed of the preconditioning and its associated communications (all to all v) and Cholesky.
double filt[16] = {F0, F1, ..., F15};
void magicfilter(int n, int ndat, double *in, double *out) {
    double temp;
    for (j=0; j<ndat; j++) {
        for (i=0; i<n; i++) {
            temp = 0;
            for (k=0; k<16; k++) {
                temp += in[((i-7+k)%n) + j*n] * filt[k];
            }
            out[j + i*ndat] = temp;
        }
    }
}

Figure (9): Pseudo C code for the magicfilter of BigDFT. It merges a convolution and a transposition.

provided. The work done on this kernel can be extended for the most part to the other parts of Preconditioning.

3.5 Quantum ESPRESSO

Quantum ESPRESSO [8] is an integrated suite of computer codes based on density functional theory, plane waves, pseudo-potentials (separable, norm-conserving and ultrasoft) and projector-augmented waves. The acronym ESPRESSO stands for opEn Source Package for Research in Electronic Structure, Simulation, and Optimization. It is freely available under the terms of the GNU General Public License (GPL). It builds upon newly restructured codes that have been developed and tested by some of the original authors of novel electronic-structure algorithms and applied in the last twenty years by hundreds of material modeling groups. In order to profile Quantum ESPRESSO we have taken advantage of the internal self-profiling capability of the code. The software, when built with MPI support, uses calls to MPI_WTIME to compute the time spent in the most important subroutines and return useful information on the scalability.

Profiling results Quantum ESPRESSO contains two main computational kernels, PWscf (pw.x) and CPV (cp.x), with two different work-flows and work-loads. For the study on Tibidabo we used CPV which implements Car-Parrinello molecular-dynamics algorithms. The experience on other machines has always shown that for this kind of kernel most of the time is spent in performing FFT, DGEMM and Hermitian matrix diagonalization operations. For the sake of completeness we should report that on Tibidabo Quantum ESPRESSO is linked against FFTW 2.5, and uses BLAS and PBLAS versions distributed along with the code. In Figure 11 we show the time spent in the seven highest-level subroutines of Quantum ESPRESSO. A brief description of the most important operations performed in each subroutine is given below:
Figure (10): Influence of loop unrolling factor on BigDFT magicfilter, when the outer loop is unrolled with steps varying from 1 to 12.
Figure (11): Profiling of cp.z varying the number of tasks on Tibidabo.

init data initialization
rhoofr loop over 3D FFTs
vofrho few 3D FFTs that are larger than those in rhoofr and dforce
dforce loop on 3D FFTs and dot products
calbec DGEMM on two set of vectors
ortho parallel DGEMM and matrix diagonalization
update parallel DGEMM

Analizing the results we can see that subroutines involving parallel linear algebra scale reasonably well up to 8 nodes, whereas subroutine involving 3D-FFTs start to saturate between 4 and 8 processors. The 3D-FFT is computed in parallel distributing the z axis, i.e. each processor takes a subset of the total number of planes. These well known distribution schemes imply the presence of a global ALLTOALL operation. The wall-clock time of the communication and of the computation is shown in Figure 12 varying the number of tasks. It can be seen that the on-core computations of the 3D-FFTs scale almost perfectly, the small deviations from linearity being possibly due to the fact that the low number of planes per processor decrease the numerical efficiency of the FFT.

Prospective kernels The profiling revealed that the main bottlenecks for Quantum ESPRESSO on Tibidabo are given by 3D-FFT and DGEMM operations. While it will surely be worth trying to increase the performance of these tests using auto-tuned libraries (like ATLAS) or libraries specifically designed for ARM processors, another viable alternative could be the exploitation of the OmpSs runtime system to build a multi-
3.1 Kernel Selection Criteria and Assessment Report

3.6 PEPC

The Pretty Efficient Parallel Coulomb solver (PEPC [9]) is a public tree code that has been developed at Jülich Supercomputing Centre since the early 2000s. The tree code is a non-recursive version of the Barnes-Hut algorithm, using a level-by-level approach to both tree construction and traversals. The parallel version is a hybrid MPI/pthreads implementation of the Warren-Salmon ‘Hashed Oct-Tree’ scheme, including several variations of the tree traversal routine - the most challenging component in terms of scalability.

The oct-tree method was originally introduced in the mid 1980s to speed up astrophysical N-body simulations with long range interactions. The speed up is achieved by reducing the computational effort in the force calculation from the usual $O(N^2)$ operations needed for brute-force summation to a more amenable $O(N \log N)$. It is well suited to dynamic, nonlinear problems and can be combined with multiple-timestep integrators. As such it is part of the Scalable Fast Coulomb Solver (ScaFaCoS) project working on a unified parallel library for various methods to solve electrostatic (and gravitational) problems in large particle simulations.

Profiling Results A straightforward profiling of PEPC is not possible with most tools because of the hybrid MPI/pthreads implementation (due to better overall performance there no longer is a pure-MPI version, the minimum number of threads is two). To achieve good performance, inlining functions is also vital which again makes meaningful profiles harder since the inlined functions can only be resolved if instrumented by hand. However, the number of calls to some of those functions is so large that the extra overhead would distort the results.
We performed our tests on JuRoPA with a small demo application of PEPC called pepc-mini. Previous tests on Tibidabo revealed no qualitative differences between the ARM and Intel architectures apart from the generally different overall execution speed. The scaling was similar on both machines and in view of the scaling capabilities of PEPC the tests were performed with a very low number of MPI ranks and five threads (four ‘worker’ threads plus one oversubscribed communication thread). With our prior knowledge of the most compute intensive regions we tried two approaches to obtain more qualitative results: a hand instrumented analysis with Extrae and a profile generated with Vampir. While the first approach is very coarse grained (due to the number of instrumented functions) it can show the impact of the built-in load-balancing (Figure 13) and reveals the communication scheme of PEPC (Figure 14). A computed profile of the time spent in the user functions on rank 0 is shown in Table 1. With this granularity we see that:
Table (1): Profile of first MPI rank for **PEPC** as computed from the coarse grained trace, Figure 13.

- 65% is spent in the worker thread during tree traversal (threads 2-5). Here the tree walk for single particles and the necessary force calculation takes place. In addition request lists for communication are filled.

- 20% is spent in the communication thread which requests data based on the request list and answers other nodes’ requests (thread 1).

We stress that the communication in the important regions is point-to-point and non-deterministic (Figure 14), as it depends on the tree structure and the load-balancing, both changing at every time step. As a consequence, the number of calls to the computationally intensive functions and their ordering is also not determined.

The second approach of obtaining timing information makes use of the profiling mode of **Vampir**. This disables inlining of functions hence producing extra overhead. It also does not provide a call-graph making it ambiguous to identify compute intensive regions since some functions are called from different parts of the code and cannot be assigned clearly or may be counted multiple times. The use of pthreads adds even more complexity to that. The important functions of the profile are (total runtime was approx. 172s):

```plaintext
<table>
<thead>
<tr>
<th>excl.</th>
<th>incl.</th>
<th>excl. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>time</td>
<td>calls / call name</td>
</tr>
<tr>
<td>47.392s</td>
<td>124.786s</td>
<td>17774.65 2.666ms module_walk_MOD_walk_single_particle</td>
</tr>
<tr>
<td>15.233s</td>
<td>15.233s</td>
<td>128485360.25 0.118us module_spacefilling_MOD_level_from_key</td>
</tr>
<tr>
<td>13.719s</td>
<td>20.187s</td>
<td>64235603.70 0.213us module_spacefilling_MOD_is_ancestor_of_particle</td>
</tr>
<tr>
<td>13.086s</td>
<td>19.888s</td>
<td>64358470.33 0.203us module_htable_MOD_key2addr</td>
</tr>
<tr>
<td>12.190s</td>
<td>0.00ns</td>
<td>8.80 1.385s pthreads_exithread</td>
</tr>
<tr>
<td>11.361s</td>
<td>19.343s</td>
<td>51941316.15 0.218us module_interaction_specific_MOD_calc_force_per_interaction</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>5.801us</td>
<td>6.962us</td>
<td>2.20 2.637us module_interaction_specific_MOD_calc_force_per_particle</td>
</tr>
<tr>
<td>5.638us</td>
<td>34.336s</td>
<td>2.20 2.563us module_walk_MOD_tree_walk</td>
</tr>
<tr>
<td>5.405us</td>
<td>1.724ms</td>
<td>2.20 2.457us module_pepc_MOD_pepc_timber_tree</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
```
As mentioned, the inclusive time is not reliable (e.g. module_walk_MOD_tree_walk results in calls to module_walk_MOD_walk_single_particle which is not fully included in the inclusive time). Adding up some of the longer functions along the call-graph of pepc-mini (Figure 15) but not removing any double counting, we find mock inclusive relative timings of some of the intensive parts to be:

- 68%* walk_single_particle, traverses the tree and calculates interaction
- 17%* is_ancestor_of_particle, checks relation of particles based on their key
- 16% key2addr, calculates mapping of hash keys to addresses
- 13%* run_communication_loop, fulfils local and remote requests for particles
- 9% level_from_key, calculates tree level from the particle key

The values with * indicate unclear assignment of function calls. Note that not all functions in the call graph have been followed and included in the above relative timings. Comparing those results with the approximations from our first approach seems consistent however. We again stress that some of those functions will be inlined during standard execution. Their number of calls points to the overhead incurred. Also observe the time per call of those functions: although they may be responsible for a large fraction of the computation, algorithmically they are very short.

**Prospective kernels** The difficulties in profiling PEPC already show that identifying simple kernels that also lend themselves to taskification for OmpSs is not straightforward. The mentioned non-deterministic execution of a tree code inhibit generating a set of input parameters to isolated functions. The tree structure impacts the runtime behaviour and building up the tree is likewise an important part of the code. On the other hand, PEPC has a modular structure, Figure 16, with frontend and interaction parts that are specific to the physics application while the tree algorithm is common to all of those. From that point of view, the PEPC library containing the core tree code may be considered a kernel.

Lacking simple input parameters, the demo version pepc-mini is the best candidate to be used in this work-package: it comprises of a short physics setup that results in a particle tree with all its necessary structures. This route has already been followed for the TEXT project during which the taskification based on SmpSs has been performed.

### 3.7 COSMO

The COntsortium for Small-scale MOdeling (COSMO [11]) was formed in October 1998 with the objective to create a meso-to-micro scale prediction and simulation system. To meet the computational requirement of the model, the program has been coded in Standard Fortran 90 and parallelized using the MPI library for message passing on distributed memory machines.
Figure (15): A zoomed-in crop of the call-graph. The different colours identify: red are functions singled out from the profile, green is the approximate inclusive fraction of execution time, blue is the exclusive fraction of the execution time along with time per call.
Figure (16): Structure of the Treecode Framework. Currently, PEPC supports three interaction-specific modules and several respective frontends for different applications ranging from plasma physics through vortex-particle methods to smooth particle hydrodynamics. Due to well-defined interfaces, symbolised by the grey blocks, the implementation of further interaction models as well as additional frontends is straightforward.

Profiling results  COSMO has been profiled using as a benchmark 100 time-steps of a small size real production run (lm7_a_m18h). The scaling behavior of the application is shown in Figure 17 (all the profiling data have been obtained using TAU [15] compiler instrumentation). The sub-optimal performance stems from the static division of the domain implemented within the application. In fact the wall-clock time spent by different processes in most of the routines implementing physical models (computations + related communications) largely depends on the properties of the domain being analyzed. This can be seen properly from Figure 18, showing the breakdown of the mean exclusive-time and the corresponding standard deviation in the case of a run with 60 processes. As it can be noticed there is a relevant deviation from the mean exclusive-time in both the time spent inside communication routines and the time spent inside computational routines (e.g. hydci_pp). Further analysis show that the deviation does not sensibly decrease incrementing the number of processes, as reported in Figure 19 for MPI_Sendrecv and hydci_pp routines.

Prospective kernels  The version of COSMO used to perform these benchmarks (COSMO_RAPS) is in itself a kernel, as it contains only the computational part of the real application without all the data structures that are related to the weather forecast production chain. As the results shown in the previous paragraph suggest, the main bottlenecks for future improvements could be...
Figure (17): Scaling for the first 100 time steps of the *lm7_a_m18b* benchmark test from 2 to 60 processes on CINECA PLX [4] computer.

Figure (18): Breakdown of the mean exclusive-time for the case of 60 processes and corresponding standard deviation.
Figure (19): Mean percentage of the wall-clock time spent inside MPI_Sendrecv and hydci_pp routines plus corresponding error-bar.

tleneck of this application is related to the unbalanced load that is statically assigned to each process at startup. In this sense the implementation of a multi-threaded task-based layer may help improving the load-balancing and thus the performance of the overall application. The addition of OmpSs directives inside performance-critical routines is thus envisaged as a possible mean to overcome the current scalability limitations.

3.8 BQCD

BQCD [12] is a hybrid Monte-Carlo code that simulates Quantum Chromodynamics with dynamical standard Wilson fermions. The code is parallelized by a domain decomposition technique implemented with four different approaches (multi-threaded OpenMP, pure MPI, hybrid MPI/OpenMP and shmem). The computations are defined on a four-dimensional regular grid with periodic boundary conditions.

Profiling Results  A strong scaling study up to 128 cores with a lattice of size $16^4$ on Tibidabo (Figure 20) shows that the hybrid MPI/OpenMP implementation outperforms the pure MPI one. The scaling behavior of both versions appears to be linear. A detailed comparison of the trace files for both cases is still on-going. A first preliminary analysis of the pure MPI implementation is shown in Figure 21 (representing the time spent in different MPI routines) and Figure 22 (showing in red group communication routines).
Figure (20): Strong scaling for the 16\textsuperscript{4} lattice on Tibidabo using different communication setups.

Prospective kernels The previous results showed that the main bottleneck of this code corresponds to the entire conjugate gradient solver, which can then be considered the candidate kernel for BQCD.

4 The “Hydro” kernel

Hydro is a simplified version of an astrophysics code, RAMSES [13], developed to study large-scale structure and galaxy formation. To maintain the size of its sources within a range suitable for a kernel application, several constraints have been imposed, the more

Figure (21): MPI call profile for the pure MPI version of BQCD
important of which are:

1. the discretization of the space domain with a structured Cartesian mesh,
2. the simulation of only the compressible Euler equations of hydrodynamics,
3. the use of a second-order Godunov scheme to discretize the Euler equations,
4. the use of a Riemann solver to compute numerical fluxes at the interfaces.

Within the framework of PRACE-1IP [2] Hydro has been further ported to a variety of programming paradigms and architectures that could be considered competitors of both the OmpSs programming model and the prototype ARM clusters being developed in MontBlanc. Due to all these characteristics this mini-application proved to be the best benchmark candidate to study, assess and compare the performance of the new technologies proposed by the project.

**Profiling results** To cope with a wide variety of approaches the sources of Hydro naturally evolved in two different branches. The first one is written in Fortran 90 and provides implementation of “classical” parallelization techniques (multi-threaded OpenMP, MPI and hybrid MPI/OpenMP). The second one has been developed in C and is mainly used to port the application on accelerators (using e.g. CUDA, OpenCL) and on novel HPC languages such as UPC.

**Serial application** Figure 23 shows the time breakdown of a sequential run of the first of the two branches, obtained on Tibidabo with the tool “gprof”. The data set used for this profiling is a square domain of size 250 × 250, as a larger data set would not fit the memory of a single node. It could be noticed that approximately 80% of the wall-clock time is spent inside four routines, that constitute thus the computational core of this benchmark. The same benchmark run performed on an IBM Power6 processor is shown in Figure 24, exhibiting a much lower relative weight of the **trace** and **slope**
Figure (23): Sequential profiling of Hydro performed on a 250 × 250 test case on Tividabo.

Figure (24): Sequential profiling of Hydro performed on a 250 × 250 test case on an IBM Power6 cluster.

routines. This can be easily explained by the fact that these routines contain a lot of intrinsic function calls for which an extremely optimized version is still missing in the ARM part of the GNU Compiler Collections.

Parallel application The scalability of a pure MPI version of Hydro is shown in Figure 25 in terms of speed-up against the serial execution. The performance seems to be fair up to 16 cores where it starts to degrade mainly due to the size of the dataset that imposes a great imbalance between communications and computations.

5 Selection of the final set of kernels

A first analysis of the results presented in Section 3 seems to suggest that kernels may be roughly classified into two different categories. On the one hand there are many codes that spend most of the time into small-size functions, performing widely known algebraic operations. On the other hand there are applications that expose a more complex algorithmic structure, and for which it is possible to identify medium-size kernels made up of different routines that are related to each other according to some specific pattern.
It can be argued that kernels of the first kind (small-size kernels) allow for fast code rewriting and, due to the larger amount of scientific literature available, will possibly permit a better comprehension of the benchmark results. Medium-size kernels on the other hand are more suited to test the performance of real applications and fit better to a task based paradigm. These complementary characteristics lead us to the decision of having representatives of both types in the final set of kernels.

Small-size kernels Within this category we decided to select:

1. the diagonalization of complex Hermitian matrices,
2. the computation of Fast Fourier Transforms,
3. the solution of sparse linear systems.

The first of these three items stems from Quantum ESPRESSO (Section 3.5), and results to be important for many material science applications. The other two kernels are more related to EUTERPE (Section 3.1) and are also widely used in scientific codes. Notice that the experimentations to be performed on these kernels are limited to the efficient use of the OmpSs paradigm, and thus are complementary to the low-level optimizations that will be performed by WP5 during the tuning of serial run-time libraries.

Medium-size kernels The medium-size kernels that will be ported to OmpSs are:

1. the preconditioning kernel (with related communications) of BigDFT,
2. the reduced, non-production version of SPECFEM3D,
3. the reduced, non-production version of \textsc{COSMO},

4. the \textit{thermostat} routine of \textsc{MP2C},

5. the “Hydro” kernel.

Notice that all the elements in the previous list come from different fields of application (material science for \textsc{BigDFT}, weather prediction for \textsc{COSMO}, etc.) and thus give an idea of the wide-spectrum of algorithms whose porting and optimization will be studied during the course of MontBlanc.

\section{Conclusion and next steps}

In this document we have reported on the selection of the performance-critical kernels to be used in the forthcoming part of the project. The assessment criteria that guided this choice have been discussed in Section 2. In Section 3 preliminary profilings of WP4 applications have been shown and prospective kernels have been proposed for each. In Section 4 we motivated the need for including the “Hydro” benchmark into the list of selected kernels, highlighting its features and unique characteristics. Finally in Section 5 we presented the final list and argued on the choice of dividing them into two different categories.

In the forthcoming part of the project this set of kernels will be ported to the OmpSs programming model within the scope of T3.2. A particular care will be taken to locate critical aspects, related to either the ARM platform or the shift of paradigm, and provide means to overcome them. This activity should aim at providing a set of best-practices to help WP4 in porting full applications. A thorough profiling of the ported kernels will then be performed in T3.3, and based on that a subset to be optimized at a low architectural level will be chosen.

\section*{References}


