D3.3 Porting, benchmarking and optimization – 2nd progress report Version 1.0

Document Information

Contract Number	288777
Project Website	www.montblanc-project.eu
Contractual Deadline	M24
Dissemination Level	RE
Nature	Report
Coordinator	Nico Sanna (CINECA)
Contributors	All Mont-Blanc Partners
Reviewer	N. Puzovic (BSC), T. Palfer-Sollier (BULL)
Keywords	Kernels, OmpSs, ARM

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.

Change Log

Version	Description of Change
v0.1	Initial Draft released to the European Commission
v0.2	First proposed template to the partners of the Consortium
v0.3	First proposed draft after contributions from the partners of the Consortium
v0.4	Version reviewed by the internal reviewers
v0.5	Version modified after internal reviewers suggestions
V1.0	Final version for the European Commission

Table of Contents

Executive Summary

This report refers to the activities planned in WP3 under Task 3.2 and 3.3.

After completion of D3.1 we identified two subsets of application kernels: small-size and medium-size kernels. After the status update of T3.2 given in D3.2, in the following we describe the WP3 final porting activities by reporting the detailed status of advancement with respect to D3.2. As in D3.2, after the porting on ARM, we focused over two major issues affecting the results on the kernels' development: (i) the porting to OmpSs; (ii) the porting to OpenCL. As already reported in D3.2, some preliminary benchmarking has been carried out on the available Mont-Blanc prototypes but a full optimization of the most promising kernels (related to T3.3) will be made when the final system will be released.

Activities expected in T3.2 can be considered as concluded with most of the kernels preliminarily integrated into the full application from WP4 even if some porting activities will continue in P3. In particular, *(i)* the small size kernels development activities have been concluded and will continue with the three kernels integrated into the full application they refer to; *(ii)* all the medium-size kernels were integrated into the corresponding full application and passed OmpSs compilation; (iii) medium-size kernels porting over OmpSs is almost completed while OpenCL porting is still in progress.

1 Introduction

This report refers to the activities planned in WP3 under Task 3.2

T3.2. Porting of the kernels (m6:m24)
The aim of this task is to provide a ported version of the source code for each kernel selected in Task 3.1 and run them on the target platform. Critical aspects of the porting will be pointed out for each kernel and overcome by means of specific actions. Recovery strategies will be identified and applied if necessary. This activity will also provide assistance to Task 4.1 in enabling the full applications containing one or more of the selected kernels to the platform.

and also related to the next Task 3.3

T3.3. Profiling, benchmarking and optimization (m6:m36)
This task will profile the kernels to determine the kernel behaviour in detail which helps to select those kernels that can optimally fit on the target hardware. The profiles will be a good basis for porting the routine since they allow to detect the most time consuming parts of the code and to optimize them, exploiting at best the underlying architecture. This task will be mainly involved in optimizing the kernels at a low architectural level. (…)

while the activities that were planned for period M18-M24 for T3.4 [Efficiency, performance and productivity evaluation (m18:m36)] have been shifted to P3 of the project plan when the final Mont-Blanc prototype will be available.

In the table below we summarize the status of advancement of the porting in WP3:

WP4 Application	WP3 Kernel	ARM	Task Identified	Taskyfied (OmpSs)	OmpSs + CUDA	OmpSs + OpenCL
EUTERPE	PCG Kernel					
QuantumESPRESSO	DGEMM/FFT					
BigDFT	Magicfilter					
SPECFM3D	Several routines					
COSMO	Himeno/OPCODE					
MP2C	Thermostat					
RAMSES	Hydro					

LEGEND: Green: task completed; Orange: task in final stage; White: to be carried out in P3.

2 Small Size Kernels

In the first progress report[1] of WP3, we identified three small-size kernel belonging to the EUTERPE[2] and QuantumESPRESSO[3] applications:

Diagonalization of complex Hermitian matrices
Computation of 1D-2D Fast Fourier Transforms
Solution of sparse linear systems

We recall that "small size kernels" are of paramount importance within Mont-Blanc project because:

- They are representative of the entire set of applications.
- Their key features can stress the hardware (including communication network) and software stack of the Tibidabo prototype.

Although different, it can be shown that all the three small-size kernels depend on well defined and well identified low level (small) code structures. Thus, a common approach based on a top-down analysis of their sources has been used to reveal the most time-consuming part of codes and to identify the key software components to be ported over OmpSs[4].

The small-size kernel for QuantumESPRESSO
Not surprisingly, these basic, albeit time-consuming, code sections rely at the very end on two well known functions: the BLAS (Basic Linear Algebra Subprograms) *dgemm()* function for linear algebra operations and 1D FFT for Fast Fourier Transforms. In fact, as reported in D3.2[5], we have already identified the most computational demanding low-level routines of the (Communication-Aware Conjugate Gradient) CA-CG[6] PETSc (Portable, Extensible Toolkit for Scientific Computation) library to be, among others, *MatMult()* and *VecTDot()* both of which rely on the linear algebra operations. Then, keeping in mind the final integrations of these kernels into the corresponding full application, we referred to the literature on QuantumESPRESSO by identifying a paper where a porting and comprehensive performance analysis of GEMM and FFT functions have been carried out on hybrid HPC architectures based on GPU[7]. According with this work, when using PWscf of QuantumESPRESSO the GEMM and FFT (3D) operations account for more than 70% of the total computing time (CPU) when running the DEISA AUSURF112 test-case. This study, among others, is a further confirmation that we correctly identified in D3.1 the small-size kernel eligible to be ported to OmpSs but is also introducing a pathway for the migration of the kernels into the full application. In fact, in the paper the authors embedded GEMM operations into the phiGEMM library and ported the 3D FFT calls in the *vloc_psi()* code section to the nVIDIA CUFFT library. This has been accomplished by creating a proper set of stub functions in order to integrate FORTRAN calls with the C calling interface of the CUBLAS and CUFFT libraries. This methodology has permitted us to clearly pinpoint in the sources the reference to the CUDA kernels and to immediately identify the code sections eligible to be taskified with OmpSs.

As an example, we report here the pseudo-code listing within the QuantumESPRESSO FFT driver we have identified for the porting over OmpSs:

```
# begin OpenMP region
   do i=1,nsl in parallel
      call 1D-FFT along z (f[offset])
   end do
# end OpenMP region

   call fw_scatter( . . . )

# begin OpenMP region
   do i=1,nzl in parallel
      do j=1,Nx
         if ( dofft[j] ) then
            call 1D-FFT along y (f[offset])
      end do
      call 1D-FFT along x (f[offset]) Ny - times
   end do
# end OpenMP region
```

The pseudo-code show the forward transformation (the backward is analogous) where *nsl* indicates the number of z sticks to be transformed and *nzl* indicates the number of planes on a processors after a parallel redistribution.
In analogy with most of the code ported in WP3/WP4 the OpenMP directive can be, in such a suitable cases, almost straightforwardly ported to OmpSs by inserting the proper clauses in the parallel regions

```
!$OMP TARGET DEVICE(OPENCL/CUDA/..) NDRANGE(.., ..,..) FILE(..)
 COPY_DEPS
!$OMP TASK IN(..,..,..) OUT(..,..,..)

    call 1D-FFT along z (f[offset])
```

At present, the zone of the sources considered eligible to be ported on OmpSs have been identified and we are currently porting the driver routines for GEMM and FFT operations. We expect to have this porting concluded by the release of the next ARM+GPU based prototype in BSC (Pedraforca) where to test the OmpSs interface with the CUDA kernel already distributed with QE. Soon after, we will start the translation of the CUDA kernels so to have the final version of the application ready for the general availability of the final Mont-Blanc prototype.

The small-size kernel for EUTERPE

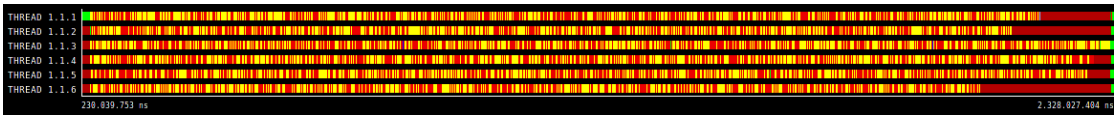
To solve the field equation in EUTERPE code, we developed a hybrid solver using openMP + MPI. The new solver is based on conjugate gradient method and is coded in Fortran language. In particular, the solver coded implements a Jacobi Preconditioned Conjugate Gradient (PCG).

The solver is parallelized using parallel loops (including the sparse matrix-vector multiplication and dot product operations), so the porting to OmpSs is almost direct (after identifying the dependences of the variables).

The results obtained show that the speedup of this solver with respect to a sequential execution is near linear.

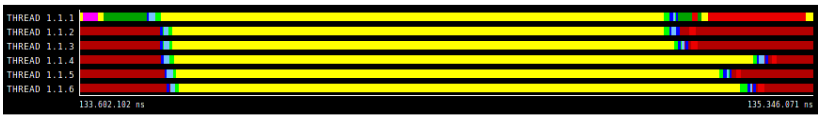
On the other side, due to the features of EUTERPE code, the solution of sparse linear system was not the most time-consuming kernel in the code, so other two kernels mentioned in the deliverable 3.1 have been developed: ***ccassign*** (charge/current calculation on the grid) and ***particle_push***. This two routines contains the small kernels listed in D3.1, because they were too small to take advantage on the GPUs, so the aim was to include the caller routines to include more code in the kernel.

All these kernels have been ported to OmpSs using parallel loops. Different techniques have been evaluated to guarantee a consistent access to shared variables. Some traces have been obtained from tests and the analysed results show the best way to proceed is defining a private copy of the shared variable per thread (Fig.1)



(a) Critical sections

(b) Atomic clauses



(c) Privated copies of a shared variable

Fig. 1. Traces of an execution of the *ccassign* kernel. The first attempt using critical sections (a) to ensure that only one thread is able to read/write the same variable at any time suffers a huge overhead due to the synchronization cost (red colour). The next attempt using of atomic clauses (b) eliminates the synchronization overhead. Finally, the uses of a private copy of the variable per each thread (c) gives us the best time by reducing the running time of each thread (yellow colour).

The porting to OpenCL is in progress. This step is taking more time because the code has to be translated from Fortran to C language, and this implies changes in the structures like n-dimension arrays to improve the access to them and coding the data movement between host and device. Some parts of the code have been tested and others are partially finished but they are still prototypes. So, we can not predict accurately what will be the performance of the integrated GPU, but we hope it will be more efficient.

Next planned steps in OmpSs are taskification of the parallel loops to get tasks. Basically, the parallel loops will be blocked to generate tasks. Moreover, once OpenCL code is finished, the kernels will be translated to CUDA. We expect this task will take less time, because the code in C will be similar.

3 Medium Size Kernels

A second class of significant code sections identified after the work in T3.1 refer to 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. Complementary to those indicated as *small-size*, *medium-size* kernels have been selected as best suited to test the performance of real applications (on real dataset) and to fit better a task based paradigm.

3.1 The preconditioning kernel of BigDFT

BigDFT [8] 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.

BigDFT OpenCL pipeline fully covers the preconditioning kernel. This kernel, and the sub-kernel that compose it, have been tested on several OpenCL platforms. Those platforms are nVidia, AMD (CPU and GPU) and Intel (CPU).

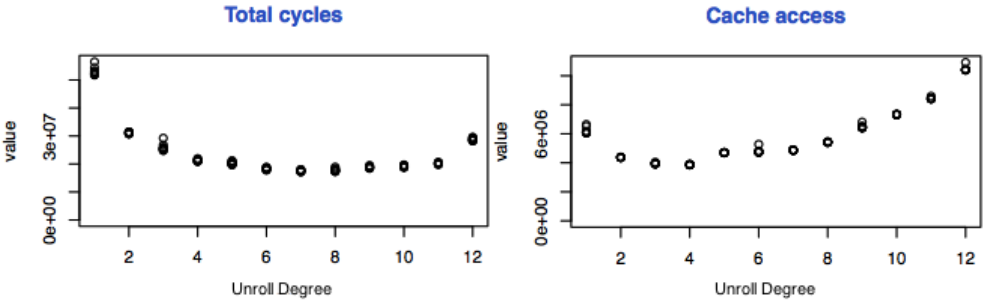
In order to validate the portability of BigDFT on the future Mont-Blanc prototype, we ordered and set-up an Exynos 5 Dual development platform (ARNDALE-5250, Samsung dual cortex A15). It is using the Linaro linux distribution. The version used is based on a 3.1 kernel. The compiler used is **gcc-4.6**, and the "**–O2**" compilation flag is used. Experiments are carried out at 1.7 GHz.

The board was compared to an Intel x5550 processor and a Snowball board (ST-Ericsson Nova A9500 dual cortex A9 at 1GHz). For energy evaluation the TDP of the x5550 was used (95W, best case for Intel), the total board power consumption for the Snowball (2.5W worst case) and the estimated power consumption of the development board when not using the GPU (5W). Here are some preliminary results:

	Snwoball (2 cores)	Intel x5550 (4 cores)	Exynos 5 Dual (1 core)	Exynos 5 Dual (2 cores)
BigDFT (SiH4) time (s)	420.4	18.1	159.0	97.4
BigDFT (SiH4) energy (J)	1050	1720	759	487

These results are of course imprecise, but nonetheless they reveal that the cortex A15 architecture is much more efficient at running BigDFT than was the cortex A9 one. The OpenCL version of the application could not be tested as we do not have yet access to the OpenCL driver from ARM.

As the porting of BigDFT (and of its preconditioning kernel) is complete in term of functionalities, two objectives remain for now in the context of WP3. The first is the porting of the kernel to OmpSs; BigDFT now builds when using the OmpSs FORTRAN compiler so it is something that can be undertaken. Nonetheless execution problems remain. The second is the optimization of the kernel.
The preconditioning kernel is mainly using 3D convolutions as base building blocks. Those blocks have to be optimized individually to fit the cortex A15 architecture. In order to achieve this goal we are working on building a code tuner with automatic benchmarking capabilities, in order to generate the most suitable versions of our 3D convolutions. This tool can generate C, FORTRAN and OpenCL, thus allowing us to express and optimize our convolutions for both the CPU and the OpenCL GPU code path.
The framework is improving and a broader coverage of BigDFT is implemented. Here are two graphs representing the time to perform a **magicfilter** convolution as well as the number of cache accesses necessary to do so, depending on the unrolling degree of the most outer loop. The experiments were run on a Snowball platform.



As the figure shows, selecting the correct unrolling degree (7 in this case) can yield impressive performance gains.
The convolutions used to build the preconditioning kernel are being expressed inside our framework, and we are working toward a complete coverage of this kernel. The framework could also be used to insert OmpSs directives (it is already used to insert OpenMP directives) but it is too soon to decide if this granularity is appropriate. Further tracing and analysis efforts using the target architecture (or the development board) are required before a final decision are made.

3.2 The reduced, non-production version of SPECFEM3D

As stated in D3.2 the SPECFEM3D[9] kernel has been already ported to OmpSs and the main routines identified with CUDA. Nonetheless, the porting from CUDA to OpenCL and the corresponding profiling activities on the prototype architecture have been delayed. As such, the porting and evaluation of this kernel is at present still work in progress we expect to be concluded in P3 and the corresponding activities reported onto the next deliverable D3.5.

3.3 The reduced, non-production version of COSMO

As reported in D3.2 we recall here that the OPCODE test-bed was the COSMO version chosen for the initial porting over the Mont-Blanc prototypes and this task was successfully completed at M18 on the Tibidabo cluster. In order to proceed with the OmpSs+OpenCL, porting a more detailed profiling (which can be obtained from the analysis of YUTIMING output) was carried out on ARM and Intel environment. The profiling output clearly shown that the Dynamical core (DYN) and Physics part (PHY) of OPCODE are both responsible for most of the computing time for a given simulation (at least with the "artificial" input configuration).
Thus, as reported in the following, we focused our porting activities on these two kernels letting the complete code analysis to further studies.

Himeno Porting to OmpSs
Since OPCODE has a quite large number of source lines and routines, at first we had to study a prototype code able to simulate the DYN and/or the PHY part of the package thus to rely on a more manageable code size for the porting activity to OmpSs+OpenCL. The idea was that after the porting of these templates we could easily move the code or its structure to the final version of OPCODE. This approach is quite customary in any porting/development activity in computer programming but a key factor affecting the final quality of the porting (performance, numerical accuracy and efficiency) is of course the choice of the template code. In particular, since the PHY part is already well structured in term of parallel task via OpenACC (see below and D3.2) we focused our study on the DYN part of OPCODE and decided to branch its porting activity including in task T3.3 the Himeno benchmark[10]. Himeno is a well-known benchmark available in literature able to take measurements while following the major loops in solving the Poisson's equation solution using the Jacobi iteration method and has also a consolidated reputation as test-bed of hybrid parallel architecture[11].
Among other things, the reasons why we chose Himeno as template for the DYN part of OPCODE can be summarized below:

- The structure of Himeno Benchmark is really quite similar to the structure of the solver routine we can find in OPCODE Dynamical core. Furthermore, the algorithm of the Himeno Solver can be almost straightforwardly mapped onto one of the algorithm in OPCODE Dynamical core (i.e., the source code of *fast_wave* routine).
- It results easier to change the structure of Himeno Benchmark with respect to the original OPCODE source. To this end, working on Himeno could in principle greatly simplify the porting to OmpSs (taskification) while permitting a more detailed control at source level in order to guarantee the numerical correctness of results, and thus, we expect the final porting to OpenCL to be better focused on the most time-consuming kernels.
- The performance evaluation of the ported OmpSs+OpenCL version of the Himeno benchmark is expected to be much more effective with respect to the original OPCODE. In fact, measuring FLOPS, code efficiency and numerical accuracy is certainly a more complicated goal to reach using the Dynamical Core and/or the Physics core of OPCODE.

The Himeno benchmark selected was the standard Fortran90 serial implementation available for downloading from the official web site. As a further advantage, using the Fortran version of Himeno we were able to test the new environment which is related to the (experimental) use of Mercurium and Fortran. In fact, in collaboration with WP5 since M18 we tested various version of Mercurium thus producing useful feedbacks to the compiler developers' team. We would emphasize that thanks to this fruitful collaboration planned in the Mont-Blanc project, as of M24 all problems related to the porting of the Himeno Benchmark to OmpSs+OpenCL were successfully solved.
By going into the details, the first step of the porting to OmpSs was related to the taskification of the main solver of the Himeno Benchmark (*jacobi* routine).
In order to guarantee the correctness of the solver, the *jacobi* routine was splitted into *update* and *swap* routine.
In this preliminary step we skipped the computation of the GOSA reduction variable. We recall that GOSA is related to the computation of some discrete quantity on the 3D grid and that it can be used to check the quality of results at the end of iterative scheme. For development reasons, the analysis of GOSA was postponed but successfully carried out in a further step (see next section). So, at this stage of the code development, we had a new version of Himeno which was successfully taskified.

Among other things, we would highlight how by using OmpSs the user can access during the development stage to a sort of "incremental parallelization. This is very useful because we can test, for example, a Fortran version of the code assuming that only a "taskification" occur without any other OpenCL or CUDA code development.
For example we can define a *reduc_smp* subroutine in this way:

```
!$OMP TARGET DEVICE(SMP)
!$OMP TASK IN(U) OUT(RES)
 SUBROUTINE REDUC_SMP(N, U, RES)
 INTEGER :: N
 REAL :: U(N,N), RES(2)
 END SUBROUTINE REDUC_SMP
```

and then we can use it together with some OpenCL or CUDA kernel which was previously validated.

Himeno Porting to OpenCL
After the OmpSs porting of the Himeno Benchmark we were easily able, thanks to the taskification of the code, to write the OpenCL kernels related to initialization of Himeno matrices (*init* routine), *update* and *swap*. We decided at this stage not to focus on kernel optimization assuming that the Mont-Blanc target (SoC) architecture could be very different from the machine we have used to develop and run (the Eurora prototype at CINECA). For this reason, we used a very "basic" approach where only global memory was selected for all kernels while shared memory implementations was made only to provide different templates for further optimizations.

Himeno version	MFLOPs	Time(s)
Fortran original	5462	20.6
OmpSs+OpenCL	39783	2.8

Table 1: 512x512x256 test case, 100 iterations, GOSA reduction disabled.

Results were obtained on Eurora prototype installed at CINECA. Tesla K20s was used for OmpSs+OpenCL runs.

Himeno (with GOSA reduction) Porting to OpenCL

GOSA reduction is a critical step in Himeno benchmark and is related to the correctness of results by comparing the results' accuracy of the serial and parallel implementation. In the benchmark it is assumed to compute some quantity which is related to a "residual" during a given iteration of original Jacobi routine.
The OpenCL implementation of GOSA reduction was carried out with WP5 (BSC) support and we would emphasize that, at present, code efficiency it is certainly not at its best. In fact, by implementing this parallel operations with OmpSs (that is, having GOSA reduction implemented in the Himeno Benchmark) resulted in adding a lot of coding machinery letting the code more "constrained" with respect to the original version of Himeno. Although we could have to solve similar issues in implementing the Himeno structure into the Dynamical core routines of OPCODE, at present we do not expect any particular issue for this activity in T3.4.
As an example, we report in Table 2 some preliminary results of calculations carried out on the Eurora machine when running the testbed on CPU only (Fortran original) and on CPU+GPU (OmpSs+OpenCL). The resulting performance show a speedup of ca. 5 when passing from the CPU to the GPU run, without loss of accuracy of the scalar GOSA residual.

Himeno version	MFLOPs	Time(s)	GOSA
Fortran original	5453	20.5	7.7E-04
OmpSs+OpenCL	27140	4.12	7.8E-04

Table 2: 512x512x256 test case, 100 iterations, GOSA reduction enabled.

Next steps and follow-up

As of M24, the Himeno benchmark structure is going to be moved to the DYN part of OPCODE as well as the OpenACC directives of PHY thus performance evaluation activities as medium-size kernel will be accomplished within WP4 into the integrated COSMO application. As a consequence, in P3 the reference medium-size kernels in WP3 for COSMO OPCODE will be the Himeno benchmark for DYN and the most promising OmpSs kernels of PHY. In P3, both code sections development and benchmarking will be tailored in strict correlation with the Mont-Blanc prototype availability in this timeframe of the project so to perform at best the code optimization and deployment on the final hardware and software environment.

3.4 The thermostat routine of MP2C

Multi-Particle Collision Dynamics MP2C[12,13] operates on discrete thermal particles, where random rotations in velocity space describe inter-particle collisions. In order to maintain a temperature or to introduce desired temperature gradients, thermostats can be applied. In order to conserve the statistical properties of the fluid, a thermostat is applied which operates locally, introducing thermal fluctuations which correspond to a given temperature, while maintaining the proper statistical velocity distribution function.

The basic principle of the thermostat is that energy fluctuations for a small number of degrees of freedom are considered, corresponding to the number of particles within collision cells. The distribution function for the kinetic energy on the cell level is considered as a generator function for random variates, in order to preserve the statistical properties on the cell level. Random variates obeying this function (which contains exponential and Gamma-functions) can be generated by an acceptance-rejection method. In a next step, the velocities of particles within a given cell are scaled to the fluctuating energy of particles within the cell. This procedure guarantees conservation of momentum as well as thermal statistical properties of the fluid.

Since operations can be performed locally on a cell level, no explicit communication is necessary for local values. Only for the final calculation of the system velocity distribution function as well as the system temperature, a global operation is required, which, however is a control value and not directly required for the operation of the algorithm.

In order to improve the overall speed of the thermostat routine, it is considered to port some parts of the kernel to GPUs. Since the calculation of modified velocities is cell based no data dependencies exist between neighbored cells and (if needed) the data can be partitioned to be calculated efficiently on a GPU.

Based on results obtained with the OpenACC concept, parts could be detected in the kernel, which are most suitable to be executed within the OMPSs task model. First results were obtained for including OMPSs tasks into the kernel, which will form the basis for a further inclusion of direct GPU support. While including the OmpSs tasks, special attention has given to the calculation of random numbers needed by the thermostat. Two possibilities were considered: (i) a serial pre-calculation of a big amount of random numbers which could be distributed to different tasks; (ii) defining a unique seed value on each thread allowing to compute independent random numbers within the threads. The second approach was preferred as the first one induces a serial work component in the code, which will ultimately lead to a saturation of parallel efficiency of the code including tasks.

It is planned to finish the inclusion of the OmpSs into the kernel and then to advance to the use of GPU (OpenCL) kernels under the assumption that the workload on a GPU will be sufficient to get an overall efficiency gain, which can be expected for parts in the thermostat routine (kernel), where a large number of similar operations are performed on particles. Examples for such regions would include those parts of the thermostat, where, e.g., cell center-of-mass velocities are calculated or where normalizations are performed.

3.5 The "Hydro" kernel

RAMSES[14] is an astrophysics code developed by Romain Teyssier at CEA (Saclay, France) to study large-scale structure and galaxy formation.
The code is written in FORTRAN with extensive use of MPI library and is well known in the Computational Fluid Dynamics community and has proven to be scalable to tens of thousands of processes.

The HYDRO package used into Mont Blanc by WP3 is a simplified version of the RAMSES code since:

- The space domain is rectangular two-dimensional splitting with a regular cartesian mesh (there is no Adaptive Mesh Refinement);
- HYDRO solves compressible Euler equations of hydrodynamics;
- HYDRO is based on finite volume numerical method using a second order Godunov scheme for Euler equations;
- A Riemann solver computes numerical flux at the interface of two neighbouring computational cells;
- HYDRO code has about 1500 lines.

HYDRO is neither a small kernel nor big software. It is much more like a mini-app which uses common algorithms representative of the ones found in HPC. This code has been developed by CNRS/IDRIS and CEA in the field of the PRACE1IP project[15], it provides a very multi-paradigm code for being used as benchmark for heterogeneous many core platforms.

Hence, it seems to be a good candidate to fulfil the goals of WP3 and more generally by the project:

- study and assess classical parallelization techniques and more advance programming frameworks related to accelerators or ARM based machines;
- compare the performances and parallel scalability of a wide variety of architectures, including the low power ones coming from Mont Blanc and based on scalar nodes or hybrid (scalar/gpus) nodes.

In order to cope with the wide varieties of approaches that we want to assess, two main branches of the code have been developed:

1. The initial FORTRAN branch including OpenMP, MPI, hybrid MPI/OpenMP approaches,
2. A C branch to mainly facilitate the porting of the application on GPU platforms including CUDA, OpenCL, HMPP programming frameworks and novel HPC languages including UPC and CoArray Fortran.

During the Mont Blanc project, an OmpSs version is planned to be written also.

The code is written in Fortran90 and it includes 5 files:

- `main.f90`
- `module_hydro_principal.f90`: implementation of algorithm and of the computation of the new time step
- `module_hydro_utils.f90`: implementation of Riemann solver
- `module_hydro_IO.f90`: implementation of read and write routines
- `module_hydro_commun.f90`: parameters and constants

Porting of Hydro to OpenCL

Following the porting of the C99 version of the code in CUDA, a shift to an OpenCL implementation was a quite straightforward process. All the analysis was done, the constraints understood. It was just a problem of moving the code to yet another language with its idiosyncrasies.
The case of OpenCL can be decomposed into two sub problems:

1. porting the kernel to the OpenCL language and
2. calling the kernels plus operating the data movements in an efficient way.

Moving a kernel from CUDA to OpenCL can almost be done automatically. With a set of clever macros and helper functions in order to hide unnecessary complexity, one can easily produce a single source code which can be used either with CUDA or OpenCL. It tooks us a couple of hours to convert all the kernels to pure OpenCL.
The real difficulty is to call the kernels. While CUDA has been nicely integrated into the C language, OpenCL relies on an API which is rather verbose and cumbersome to use. To ease the burden of OpenCL programming, we developed a set of comfort functions allowing for an almost CUDA looking implementation (the `ocltools.h` and `ocltools.h` files).

Moving to OpenCL proved itself to be a rather easy task. Most of the troubles we faced were due to immature implementations, OpenCL being a rather fragile environment for most vendors.

Performance

End of 2012: in order to compare the different versions we use a 9000x9000 domain with no VTK output to avoid the I/O time which introduces timings variations according to the load of the machine (a single workstation based on Intel Westmere-EP with an nVIDIA C2050 GPU attached).

We compute 10 iterations and report the total elapsed time for a single CPU or a single GPU.

9000x9000	C99 2D B=1	C99 2D B=20	CUDA 2D B=1	CUDA 2D B=20	HMPP CUDA	OpenCL
Westmere-EP	701.724s	729.978s	N/A	N/A	N/A	N/A
C2050	N/A	N/A	117.936s	55.150s	175.746s	269.865s
Speedup	1	0.96	5.95	12.72	3.99	2.60

B=1 means 1 line (column) at a time, B=20 means a band of 20 lines (columns) at a time. B=20 was chosen because at higher values the speedup was not much improved. On the other hand, it shows that 2D coding could have a negative effect on the CPU version (here true blocking should be experimented).
This example shows that the GPU implementation using CUDA is between 6 to 13 times more faster than the scalar version while the OpenCL version is less performing with a 2.6x increase in performance (on B=1).

More recently, new tests were performed with a muscled SandyBridge-EP workstation using 2 leading edge target architectures:

- nVIDIA K20c board based on the Kepler architecture with a peak performance of 1170 GFlops and a memory bandwidth of 208 GB/s.
- Intel Xeon Phi SE10P with a peak performance of 1070 GFlops and a memory bandwidth of 352 GB/s.

The dataset used is a 2D 4091x4091 grid with a 1000 blocking size in order to fit into the memory of the different accelerators.

Target	Timing (s)
Intel SNB-EP 1 core	780.8
Intel SNB-EP 16 cores	109.7
nVIDIA K20c CUDA	52.37
nVIDIA K20c OpenCL	42.09
Xeon Phi SE10P (240 threads)	147.5

Such new results obtained during the summer 2013 are showing now a better performance of the OpenCL version over the CUDA version of hydro. This is mainly due to the improvement of the OpenCL driver which was in 2012 quite unstable and not mature for performance.

Next Steps
During the last year of Mont Blanc the C version of the code will be ported to OmpSs.

4 Conclusions and next steps

The status of advancement in porting the small and medium size kernels is, as a whole, fairly good with small deviation with respect to the Mont-Blanc WP3 workplan. This aspect confirms the quality of the application development workplan in the Mont-Blanc proposal as well as of the activities carried out in the first 18 months of WP3 (in strict correlation with the other WPs).

Among other things, we would highlight as the availability of the Tibidabo prototype has permitted to have all the WP3 codes ported over the ARM 32 bits environment. This will positively affect the next steps toward a full porting of the kernels (and of the related applications) on the final Mont-Blanc prototype by making almost straightforward the porting over the 64 bits ARM SDK and OS. In this respect, the preliminary field test of the initial ARM prototype has shed light on the possible guidelines for code optimization over the 64 bits final environment.

On the other hands, the beta release of the OmpSs SDK for FORTRAN codes has greatly simplified the initial porting of the kernels under the “task paradigm” of programming with some of them now preliminarily running on (prototype) development systems. Taking into account that most of the kernels were already ported over OpenACC (if not relying directly on CUDA/OpenCL code sections) we expect with their porting through OmpSs to greatly gain the maximum performance benefits of the Exynos ARM-GPU hybrid architecture.

Activities expected in T3.2 can be considered as concluded with most of the kernels preliminarily integrated into the full application of WP4 even if some porting activities will continue in P3. In particular, *(i)* the small size kernels development activities have been concluded and will continue with the three kernels integrated into the full application they refer to; *(ii)* all the medium-size kernels were integrated into the corresponding full application and passed OmpSs compilation; (iii) medium-size kernels porting over OmpSs is almost completed while OpenCL porting is still in progress.

References

[1] Project ICT-288777 Mont-Blanc: *D3.1 Kernel Selection Criteria and Assessment Report*. June 2012.

[2] X. Sàez, A. Soba, E. Sànchez, R. Kleiber, F. Castejòn, and J. M. Cela. Improvements of the particle-in-cell code EUTERPE for petascaling machines. Computer Physics Communications, 182:2047-2051, September 2011.

[3] Paolo Giannozzi, Stefano Baroni, Nicola Bonini, Matteo Calandra, Roberto Car, Carlo Cavazzoni, Davide Ceresoli, Guido L Chiarotti, Matteo Cococcioni, Ismaila Dabo, Andrea Dal Corso, Stefano de Gironcoli, Stefano Fabris, Guido Fratesi, Ralph Gebauer, Uwe Gerstmann, Christos Gougoussis, Anton Kokalj, Michele Lazzeri, Layla Martin-Samos, Nicola Marzari, Francesco Mauri, Riccardo Mazzarello, Stefano Paolini, Alfredo Pasquarello, Lorenzo Paulatto, Carlo Sbraccia, Sandro Scandolo, Gabriele Sclauzero, Ari P Seitsonen, Alexander Smogunov, Paolo Umari, and Renata MWentzcovitch. QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials. Journal of Physics: Condensed Matter, 21(39):395502 (19pp), 2009.

[4] A. Duran, E. Ayguadé, R. Badia, J. Labarta, L. Martinell, X. Martorell, and J. Planas. OmpSs: A proposal for programming heterogeneous multi-core architecture. Parallel Processing Letters, 21:173-193, 2011.

[5] Project ICT-288777 Mont-Blanc: *D3.2 Porting of kernels - progress report*. April 2013.

[6] http://www.cs.berkeley.edu/~mhoemmen/pubs/thesis.pdf

[7] F. Spiga and I. Girotto, phiGEMM: a CPU-GPU library for porting Quantum ESPRESSO on hybrid systems, 20th Euromicro International Conference On Parallel Distributed and Network-based Processing (IEEE 2012), DOI: 10.1109/PDP.2012.72.

[8] L. Genovese, B. Videau, M. Ospici, T. Deutsch, S. Goedecker, and J.-F. Mhaut. Daubechies wavelets for high performance electronic structure calculations: The BigDFT project. Comptes Rendus Mécanique, 339:149-164, 2011. http://inac.cea.fr/L_Sim/BigDFT/.

[9] SPECFEM3D. http://www.geodynamics.org/cig/software/specfem3d-globe.

[10] Written by Prof. R. Himeno, Director of RIKEN HPCC center (JP), http://accc.riken.jp/2444.htm.

[11] E.g., see, E. H. Phillips and M. Fatica, IPDPS-2010 Atlanta (GA, USA). DOI: http://dx.doi.org/10.1109/IPDPS.2010.5470394

[12] J. Freche, W. Frings, and G. Sutmann. High throughput parallel-I/O using SION-lib for mesoscopic particles dynamics simulations on massively parallel computers. Advances in Parallel Computing, 19:371-378, 2010.

[13] G. Sutmann and W. Frings. Extending scalability of MP²C to more than 250k compute cores. Technical report, Forschungszentrum J•ulich, 2010.

[14] R. Osmont, D. Pomar□ede, V. Gautard, R. Teyssier, and B. Thooris. Monitoring and control of the RAMSES simulation program. In Proceedings of the CCP2007 Conference on Computational Physics, Brussels, Belgium, 2007.

[15] PRACE - Partnership for advanced computing in Europe, 2010.