# Pre-exascale Architectures: OpenPOWER Performance and Usability Assessment for French Scientific Community

Gabriel Hautreux, Alfredo Buttari, Arnaud Beck, Victor Cameo, Dimitri Lecas, Dominique Aubert, Emeric Brun, Eric Boyer, Fausto Malvagi, Gabriel Staffelbach, et al.

# Pre-exascale Architectures: OpenPOWER Performance and Usability Assessment for French Scientific Community

Gabriel Hautreux[1], Alfredo Buttari[7], Arnaud Beck[11], Victor Cameo, Dimitri Lecas, Dominique Aubert[9], Emeric Brun[10], Eric Boyer[1], Fausto Malvagi[10], Gabriel Staffelbach, Isabelle d'Ast, Joeffrey Legaux[4], Ghislain Lartigue[5], Gilles Grasseau[11], Guillaume Latu, Juan Escobar, Julien Bigot[2], Julien Derouillat[2], Matthieu Haefele[2], Nicolas Renon[8], Philippe Parnaudeau, Philippe Wautelet, Pierre-Francois Lavallee, Pierre Kestener, Remi Lacroix, Stephane Requena[1], Anthony Scemama[3], Vincent Moureau[5], Jean-Matthieu Etancelin[6], and Yann Meurdesoif

[1] GENCI, Paris, France,
[2] Maison de la Simulation, CEA, CNRS, Univ. Paris-Sud, UVSQ, Université Paris-Saclay, 91191 Gif-sur-Yvette, France
[3] Laboratoire de Chimie et Physique Quantiques, Université de Toulouse, CNRS, UPS, France
[4] CERFACS, Toulouse FRANCE
[5] CORIA, CNRS UMR6614, Normandie Université, Saint-Etienne-du-Rouvray, France
[6] CReSTIC EA3804, ROMEO HPC Center, University of Reims Champagne-Ardenne, France
[7] IRIT, CNRS UMR5505, Université de Toulouse, France
[8] CALMIP, Université de Toulouse, Université Paul Sabatier, CNRS, UMS3667, France
[9] Observatoire Astronomique de Strasbourg, UMR 7550 Universite de Strasbourg - CNRS, Strasbourg, France
[10] Den-Service d´Études des Réacteurs et de Mathématiques Appliquées (SERMA), CEA, Université Paris-Saclay, 91191 Gif-sur-Yvette, France
[11] Leprince-Ringuet Laboratory (LLR), CNRS/IN2P3, Ecole polytechnique

**Abstract.** Exascale implies a major pre-requisite in terms of energy efficiency, as an improvement of an order of magnitude must be reached in order to stay within an acceptable envelope of 20 MW. To address this objective and to continue to sustain performance, HPC architectures have to become denser, embedding many-core processors (to several hundreds of computing cores) and/or become heterogeneous, that is, using graphic processors or FPGAs. These energy-saving constraints will also affect the underlying hardware architectures (e.g., memory and storage hierarchies, networks) as well as system software (runtime, resource managers, file systems, etc.) and programming models. While some of these architectures, such as hybrid machines, have existed for a number of years and occupy noticeable ranks in the TOP 500 list, they are still limited to a small number of scientific domains and, moreover, require significant

porting effort. However, recent developments of new paradigms (especially around OpenMP and OpenACC) make these architectures much more accessible to programmers. In order to make the most of these breakthrough upcoming technologies, GENCI and its partners have set up a technology watch group and lead collaborations with vendors, relying on HPC experts and early adopted HPC solutions. The two main objectives are providing guidance and prepare the scientific communities to challenges of exascale architectures.
The work performed on the OpenPOWER platform, one of the targeted platform for exascale, is described in this paper.

**Keywords:** OpenPOWER assessment, technological watch, OpenMP, OpenACC, benchmarks, usability, programmability

## 1 Introduction: Technological watch group environment

### 1.1 Partners and goals

The technological watch group, lead by GENCI, is gathering 20 experts from the French HPC community including: CEA, CNRS, Inria, Maison de la Simulation, Groupe Calcul and the national computing centers (CINES, IDRIS, TGCC) The main goal of this activity is to assess the usability of standard programming paradigms in order to port/optimize scientific applications and tools on multiple heterogeneous novel HPC platforms. As many efforts have been done during the past to port applications using standards like OpenMP, it has been defined as the main pre-requisite for our assessment.

On top of that, the collaboration initiated with IBM and NVidia within this project helped us to enable the application thanks to multiple workshops performed since the beginning of 2016.

### 1.2 Platform and environment available

One of the key platform assessed by GENCI and its partners is the Ouessant OpenPOWER system, hosted at IDRIS, Orsay, integrated by IBM with the following characteristics:

- 12 IBM System S822LC "Minsky" nodes with for each :
  - 2 IBM POWER8 10-core processors clocked at 4.2 GHZ and 128 GB of main memory;
  - 2 NVidia P100 GPUs per socket;
  - each socket is connected to the 2 GPUs via NVLink 1.0;
- all the nodes are federated through a Mellanox EDR Infiniband interconnect
- and access to a high bandwidth filesystem (IBM Spectrum Scale, formerly known as GPFS)

This platform is installed using a Linux distribution (RedHat7) and has a variety in its compilation/execution environment. PGI, IBM and LLVM compilers were used on the platform to assess the applications. The main difficulty is that, at the moment, those compilers all have pros and cons. As LLVM had (beginning of 2016) difficulties to compile Fortran applications for the POWER8 architecture, most of the users compiled using either PGI or IBM compilers.

The compilers are evolving very frequently (almost every other week). Hence, the results presented in this paper are a snapshot of the work performed at the end of March 2017.

### 1.3 Applications

To assess the platform, we first used the following portfolio of 15 representative "real" applications, running daily in production, with the involvement of their respective developers:

- **AVBP[1]:** a parallel CFD code that solves the three-dimensional compressible Navier-Stokes equations on unstructured and hybrid grids.
- **CMS-MEM[2]:** a Matrix Element Method for High Energy Physics (HEP)
- **Dynamico:** a new dynamical core for LMD-Z, the atmospheric general circulation model (GCM) part of IPSL-CM Earth System Model
- **EMMA[3]:** an adaptive mesh refinement cosmological simulation code with radiative transfer.
- **GPS[4]:** Gross Pitaevskii Simulator : modeling of Bose-Einstein Condensates, quantum turbulence, or ultracold quantum gases in optical lattices
- **Gysela:** models the electrostatic branch of the Ion Temperature Gradient turbulence in tokamak plasmas
- **Hydro[5]:** a mini-application which implements a simplified version of RAMSES, a code developed to study large scale structure and galaxy formation
- **Meso-NH[6]:** the non-hydrostatic mesoscale atmospheric model of the French research community
- **Metalwalls:** a French molecular dynamic application
- **PATMOS[7]:** a Monte Carlo transport application
- **QMC=Chem:** a Quantum Monte Carlo code applied to chemistry
- **qr_mumps:** a direct solver for sparse linear systems
- **RAMSES:** CFD applications for astrophysics
- **SPECFEM3D_GLOBE:** simulates global and regional (continental-scale) seismic wave propagation
- **YALES2[8]:** parallel CFD for two-phase combustion in complex geometries, solving 3D low-Mach number Navier-Stokes equations on unstructured grids.

This portfolio of applications has been chosen as it represents a wide range of domains and the close collaboration we have with the developers helped us to define significant (scientifically speaking) test cases.

### 1.4 Performance Indicators

The aime of this project is to provide guidance for the future HPC users of such architecture. First workshops have demonstrated that porting an application to the POWER8 architecture is completely straightforward (compile and run), at least since the end of 2016.

However, the OpenPOWER platform involving its 4 NVidia GPUs enables the nodes to provide such a huge computational capacity that the POWER8 processor alone can not be the target for an application.

Hence, the relevant results obtained on Ouessant aim to define:

- Baselines in terms of performance (time to solution) for a given field of application ported on a full Minsky node
- the GPU porting effort for different paradigms (CUDA, OpenMP, OpenACC, . . . )
- the maturity of the software stack for code offloading to GPUs

The results obtained will help the scientific communities and GENCI to define if OpenPOWER is a suitable architecture for their simulations on top of providing guidance for the upcoming HPC procurements for French national computing centers.

## 2 Work performed on each application

The contributions in this section are mainly provided by the users. Hence, some of the applications listed earlier are not described as the worked performed on the platform is not sufficient in order to get relevant results.

### 2.1 AVBP

AVBP is an explicit compressible code for fluid mechanics and reactive applications used by both research and industry groups[12]. It has a hybrid, parallel MPI + OpenMP implementation. The domain is partitioned over the MPI tasks. For each MPI task, the local domain is partitioned into groups of cells.

The main computational section of the code is constituted by an external loop over the groups of cells. The internal routine `scheme` includes 50% of the code with internal vectorized loops over the cells. The data implementation is structured in FORTRAN modules and globally shared over the routines. AVBP is based on a coarse grain OpenMP approach with the parallel static loop over the groups of cells. In order to remove memory bottlenecks, variables and arrays are declared with OpenMP Private clause and Threadprivate directive in the modules. The contributions of every thread are store independently in the arrays passed in parameter and indexed in the latest dimension over the groups, as illustrated below with the arrays `avis`.

[12] http://cerfacs.fr/logiciels-de-simulation-pour-la-mecanique-des-fluides/

```
!$OMP PARALLEL DO SHARED(nvert,avis,...) &
PRIVATE(kgroup,itype,ng1,...)
DO kgroup=1,ngroup
   call scheme(nglen, nvert(kgroup),kgroup,ng1,itype,...,&
      volc(ng1), factor(ng5), avis(kgroup),  .. avis2(kgroup),&
      avis4_tpf(kgroup),dw_spec_c_nv_buf(:,:,:,kgroup),&
      dw_fic_c_nv_buf(:,:,:,kgroup),...)
```

The first GPU implementation has been performed using OpenACC programing model; a switch to OpenMP 4.5 will be eventually studied in a second step. The choice of OpenACC over OpenMP 4.5 has been motivated by the existence of the Unified Memory feature, even though in the very first implementation of the code, the data transfers have been manually managed using explicit directives. The GPU implementation uses the same scheme as the current OpenMP implementation, with an offload of the entire `scheme` function to the GPU. The OpenMP directive controlling the main loop over the groups has been substituted by an OpenACC directive to generate OpenACC gang or CUDA block. The internal, vectorized loops are parallelized over OpenACC vectors and threads, to ensure a good GPU occupancy. Compared to the current, pure-CPU implementation, the following evolutions are required to manage the GPU offload:

- Data from modules used within the `scheme` routine has to be allocated on the GPU using the OpenACC `declare create` directive in the corresponding module, bound with enter and exit data directives, and copied or updated to/from the GPU using OpenACC `copy/update_device/update_host` directives before and after the main loop.
- Private variables and arrays in the threads with OpenMP Threadprivate directive was declared in the private clause of the OpenACC loop over the gang.
- All the routines and function potentially offloaded to the GPU have been declared with OpenACC `seq/vector` directives. These directives direct the compiler to generate both CPU and GPU paths.

### 2.2 EMMA

EMMA is an MPI + CUDA code which was already running on GPUs before the beginning of the project. The porting effort was very small, a simple compilation and then a run enabled to get first results on the platform. At the moment, around 50% of the code is running on GPUs. The goal is to reach 80% of the code by the end of 2017. However, first results on the platform are available for this application.

The CUDA kernel results are summarized in 1 while the full application is presented in 2.

We see a huge decrease in terms of performance for running the entire application, this is clearly explained as the full application has not been ported to CUDA.

**Table 1.** EMMA Single Kernel on OpenPOWER

| Node | SMT Mode | Processes/Node | Tasks/GPU | Time | Speed-up |
|---|---|---|---|---|---|
| P8 | 1 | 4 | 0 | 5.75 | 1.0 |
| P8+4P100 | 1 | 4 | 1 | 0.21 | 27.4 |

**Table 2.** EMMA Full Application on OpenPOWER

| Node | SMT Mode | Processes/Node | Tasks/GPU | Time | Speed-up |
|---|---|---|---|---|---|
| P8 | 1 | 4 | 0 | 14.6 | 1.0 |
| P8+4P100 | 1 | 4 | 1 | 5.8 | 2.5 |

### 2.3 GPS

GPS is an application developed in MPI and relying heavily on Fast Fourier Transforms.

Approximately 100% of the code used by the test case is offloaded using OpenACC.

The performance of this OpenACC implementation suffers from the absence of the GPU-Direct feature, which is set to be made available in 2018 with the next generation of systems based on POWER9. This leads to a significant performance penalty when using the Managed Memory feature in OpenACC.

In order to evaluate the expected performance gain from GPU acceleration once the GPU-Direct feature will be released, the code has been tested without MPI ; numerical results will not be valid, but the same amount of computation is performed. In this configuration (with no specific optimization), the execution is approximately twice faster when using 4 GPU versus 16 POWER8 cores.

### 2.4 GYSELA

The first target was to evaluate the potential of the performance boost which could be provided through GPU acceleration, and thus confirm that the OpenPOWER platform was a valid architecture for the GYSELA exploitation.

In the context of this preliminary validation phase, the work was conducted:

- On a subset of the full application: the 2D-Advection Kernel.
- Through a CUDA implementation, in order to bypass the potential current limitations of the compilation environment with respect to directive-based programming models (OpenACC or OpenMP).

The performance results achieved at the end of this first step are available in table 3.

These performance levels fully validate the capacity to benefit from a significant performance boost thanks to GPU acceleration. Based on this first outcome, the second phase started in Q2 2017, which will shift:

- From the single 2D-Advection Kernel to the whole application.
- From a CUDA-based implementation to an OpenACC-based implementation.

**Table 3.** GYSELA 2D-Advection Kernel Performance on OpenPOWER

| Node | Processes/Node | MCells/s | Speed-up |
|---|---|---|---|
| P8 | 1 | 9.0 | 1.0 |
| P8+1P100 | 1 | 56.9 | 6.3 |

### 2.5 Hydro

Hydro is a MPI + OpenMP code widely used by IDRIS in their workshops as a tutorial to learn MPI and OpenMP. This application is not involving deep optimization and tries to mimic what a common developer could implement in its application. The idea with Hydro is therefore to develop an OpenACC and an OpenMP implementation that could make the most of the P100 with a limited effort of development. This code could also become a good porting example for the community.

However, a CUDA implementation has been developed as a first step, in order to evaluate the performance gain the GPU acceleration can offer. The performance level achieved through the CUDA implementation will constitute a reference target for the directive-based implementations.

The following performance results have been obtained on a 8192x8192 grid (0.5 GB memory usage) using the CUDA implementation:

**Table 4.** HYDRO Performance on OpenPOWER

| Node | Time (s) | Speed-up |
|---|---|---|
| OpenPOWER, POWER8, SMT4 | 29 | 1.0 |
| OpenPOWER, POWER8 + 1P100 | 1.9 | 15.5 |

The development of an OpenACC implementation has already started. The performance of this implementation currently suffers from the absence of the GPU-Direct feature, which is set to be made available in 2018 with the next generation of systems based on POWER9.

### 2.6 Meso-NH

Meso-NH is a code developed in MPI and OpenACC.

A part of the code was already ported in OpenACC before the beginning of the project. There is an ongoing development and the team aims to port a huge part of the application before the end of 2017.

First results on one node are already available for a given kernel in 5.

For this particular kernel, we see a speedup of 5.6 for using 4 GPUs on top of the POWER8 processor. We also can see a speedup of 2.2 using 4 P100 compared to 2 K80.

Those results are pretty good, however if we run the full application, the speedups, in table 6 are not as good.

**Table 5.** Meso-NH Single Kernel Performance on OpenPOWER

| Node | Processes/Node | Tasks/GPU | Speedup |
|---|---|---|---|
| P8 | 16 | 0 | 1.0 |
| P8+2K80 | 16 | 8 | 2.6 |
| P8+4P100 | 16 | 4 | 5.6 |

**Table 6.** Meso-NH Whole Application Performance on OpenPOWER

| Node | Processes/Node | Tasks/GPU | Speedup |
|---|---|---|---|
| P8 | 16 | 0 | 1.0 |
| P8+2K80 | 16 | 8 | 1.3 |
| P8+4P100 | 16 | 4 | 1.5 |

At the moment, the developer have troubles to port a significant part of the application on GPU. The solver involved is not as easy to port as the one previously ported. A porting effort of 2 years is envisioned in order to obtain a speed-up of 5 on the full application.

### 2.7 Metalwalls

Metalwalls is a full MPI application for molecular dynamics. The code is written in Fortran 90 and has 20.000 lines of codes. The time loop (computational part) is 3.500 lines long and represents almost 100% of the time spent in the application. The strategy chosen by the development team was to port this application using OpenACC as it seemed, in their opinion, to be the most reliable technology available. 75% (of loop time) of the application is now ported to OpenACC, it took them 1 month to carry out this work. The estimated time for porting the full application is one other month.

The first results are in table 7.

**Table 7.** Metalwalls Performance on OpenPOWER

| Test Case | Nodes | SMT Mode | Tasks/Node | Tasks/GPU | Time | Speed-up |
|---|---|---|---|---|---|---|
| Small | 1 | 4 | 80 | 0 | 5.9 | 1.0 |
| Small | 1 | 1 | 4 | 1 | 2.0 | 3.0 |
| Large | 1 | 4 | 80 | 0 | 364.7 | 1.00 |
| Large | 1 | 1 | 1 | 1 | 96.7 | 3.8 |
| Large | 1 | 1 | 4 | 1 | 74.6 | 4.9 |

Those results, after only one month of work are pretty good. However the work now has to be focused on running multiple GPUs and porting the remaining part of the code(the speed-up for 1 GPU is 3.8, while 4GPUs is 4.9, which means that the scalability on multiple GPUs is not good at the moment).
However, as only a bit more than 75% of the code has been ported, that means

the theoretical speed-up they can achieve is around 4.0 (Amdahl's law). That's the reason why we can consider that porting this application on the OpenPOWER architecture is a success for them at the moment.

### 2.8 PATMOS

PATMOS is developed in C++11/14 with an hybrid parallelism based on MPI + OpenMP. A CUDA version of the application was also available in a prior release and the main work performed by the development team was to include those CUDA kernels into the main branch of the application. This work has been done during the project and now 5% of the code (in lines, but representing 75% of the CPU time) is available in CUDA. The target is to port another 5% in order to almost cover the whole application.

The results in table 8 and 9 are obtained using 1 MPI process per node, OpenMP threads are then used for the in-node parallelism. The results obtained on multiple nodes are almost as good as the ones obtained on the single node, which shows that the scalability of the application on this platform is very good.

**Table 8.** PATMOS single node result

| Test Case | Nodes | SMT Mode | Threads/Node | GPU used | Time | Speed-up |
|---|---|---|---|---|---|---|
| Small | 1 | 4 | 80 | 0 | 24.0 | 1.0 |
| Small | 1 | 4 | 80 | 4 | 6.0 | 4.0 |

**Table 9.** PATMOS multiple node result

| Test Case | Nodes | SMT Mode | Threads/Node | GPU used | Time | Speed-up |
|---|---|---|---|---|---|---|
| Large | 8 | 4 | 80 | 0 | 582.1 | 1.0 |
| Large | 8 | 4 | 80 | 4 | 152.2 | 3.8 |

Unfortunately, no test has been done using OpenMP for offloading to GPUs, but still we hope that we could run an OpenMP version on the platform in order to compare the performances we reach in CUDA and the performances we can reach using OpenMP. This part is ongoing and we expect to have good performances using OpenMP as well.

### 2.9 QMC=Chem

QMC=Chem is a quantum chemistry code which applies the quantum Monte Carlo (QMC) method to molecules to solve the Schrödinger equation.[9] Due to the embarrassingly parallel nature of the algorithm, its parallel scaling is almost ideal, and single-core optimization is crucial to improve the performance. Hence, QMC=Chem was specifically optimized for Intel Xeon processors, used in combination with the Intel Fortran compiler.

In this study, we have used QMC=Chem as a benchmark to test the performance obtained with the Power8 CPU combined with the XL compiler toolchain. We compare the results obtained on one node of the Ouessant cluster with those obtained on one node of the Occigen cluster, namely a dual-socket Intel Xeon CPU E5-2690v3 @ 2.6GHz (Haswell). We used two test cases, one small and one large, for which the hot spots are different kernels, both very representative of the usual production runs. For the two test cases, we have counted, with the Intel Software Development Emulator,[10] the total number of single precision (SP) and double precision (DP) floating point instructions using an SSE2 executable. This information combined with the elapsed time of the benchmarks allows us to give an estimate of the performance in GFlops/s. Using the ratios of single and double precision instructions (86% SP, 14% DP for the small case, and 4% SP, 96% DP for the large case), we can also give an estimate of the percent of the peak performance that was reached. The results are given in table 10.

**Table 10.** Single node performance in GFlops/s. Percent of the peak (mixed single and double precision) is given in parenthesis.

| Compiler | CPU | Number of | Small | | Large | |
|---|---|---|---|---|---|---|
| | | threads | GFlops/s | % Peak | GFlops/s | % Peak |
| GNU | Haswell | 1 | 6.8 | | 7.1 | |
| | | 24 | 134.5 | *(7.2%)* | 145.0 | *(13.9%)* |
| | | 48 | 158.5 | *(8.5%)* | 136.9 | *(13.2%)* |
| GNU | Power8 | 1 | 10.2 | | 2.1 | |
| | | 20 | 182.5 | *(15.5%)* | 39.3 | *( 5.9%)* |
| | | 40 | 230.1 | *(19.2%)* | 68.4 | *(10.2%)* |
| | | 160 | 258.2 | *(21.6%)* | 119.7 | *(17.9%)* |
| Intel | Haswell | 1 | 18.3 | | 10.9 | |
| | | 24 | 332.9 | *(17.9%)* | 219.1 | *(21.1%)* |
| | | 48 | 346.6 | *(18.7%)* | 183.4 | *(17.8%)* |
| IBM XL | Power8 | 1 | 12.1 | | 3.5 | |
| | | 20 | 218.3 | *(18.3%)* | 64.4 | *( 9.6%)* |
| | | 40 | 272.2 | *(22.8%)* | 96.4 | *(14.4%)* |
| | | 160 | 244.2 | *(20.4%)* | 103.0 | *(15.4%)* |

To reduce the bias due to the compiler we have first used the GNU Fortran compiler on both architectures. On the small benchmark, the performance is higher on the Power8 than on the Haswell, probably due to its larger L3 cache. On the large benchmark, the performance is higher on the Haswell node, and the multi-threading on the Power8 is really crucial to approach the performance of the Xeon. Then, we have run the benchmarks compiled with vendor compilers. Using the Intel compiler gives a ×2.1 acceleration on the Haswell node for the small test case, and a ×1.5 acceleration of the large test case. Such a large difference is due to the heavy use of Intel compiler directives in the hottest loops. Going from the GNU to the IBM XL compiler, only a ×1.05 acceleration is gained on the small test case, and the large test case becomes less efficient by a factor of ×0.86.

These results show that, without any particular tuning, QMC=Chem is able to reach 22.8% of the peak performance of the Power8 node on small cases, and 17.9% on large cases. This is a good start, and as there is no performance gain using the XL compiler, we expect that a substantial performance increase could be obtained with the XL compiler if some parts of the code are rewritten in a more Power8-friendly fashion.

### 2.10 qr_mumps

qr_mumps is a direct solver for sparse linear systems based on the multifrontal QR factorization. It currently supports single nodes with multiple cores and one GPU. The parallelization is achieved through the StarPU runtime system. The problem data is decomposed into blocks and the computations are arranged into tasks whose dependencies are expressed by a Directed Acyclic Graph. StarPU takes care of launching the execution of tasks when the related dependencies are satisfied and when computational units are available [11]. In qr_mumps a dynamic and hierarchical data partitioning is used in order to have a good mix of large grain tasks, which are executed on the GPU, and fine grain tasks which are executed on the CPU cores. Moreover, when a GPU is available, a dedicated scheduling policy is used which aims at maximizing the efficiency of tasks by assigning each to the unit which is best suited for its execution. These techniques allow for an effective use of all the computational resources available on the node [12].

The porting on the OpenPOWER platform was relatively easy and needed only minor code fixes due the strict compliance to the Fortran standards enforced by IBM compilers. The graph below reports the strong scalability of the qr_mumps solver on a number of problems from the University of Florida Sparse Matrix Collection.

On the largest problem (matrix TF18) a performance of 407 Gflop/s was achieved using 20 cores, which corresponds to a remarkable 73% of the peak, with a very good scalability.

Using one GPU with the 20 cores, the performance of 1.2 Tflop/s was reached (sse 11, meaning of speed-up of 3 between one P8 node and one P8 node + one P100.

**Table 11.** qr_mumps: Performance in Tflop/s

| Nodes | Processes/Node | GPUs | Performance | Speed-up |
|---|---|---|---|---|
| 1 | 80 | 0 | 0.4 | 1.0 |
| 1 | 80 | 1 | 1.2 | 3.0 |

The main problem at the moment is that the code does not use multiple GPUs on a single node as this functionality has not been implemented. qr_mumps will soon be integrated in a larger code which will enable this feature. The multi-GPU results are expected to be very good as well.

### 2.11 RAMSES

RAMSES-GPU is developed since 2009 in CUDA/C++ for astrophysics applications on regular grid. The code is 70k lines long (out of which about 16k are written in CUDA).

The main goal of the project was to make the code more portable. In order to achieve this goal, the developer decided to use Kokkos[13]. The first result using this paradigm is very interesting.

On average Pascal P100 is 2.8 to 4.0 faster than Kepler K80 (single GPU) with no special optimization, only using rebuild architecture flags in the CUDA implementation.
On top of that, Pascal P100 is about 10 times faster than Power8 (with 8 threads per core), still with CUDA. This result is illustrated in table 12.

**Table 12.** RAMSES cell-update per second on POWER8, K80 and P100

| Test Case | POWER8 | K80 | P100 | Speed-up P100 vs POWER8 |
|---|---|---|---|---|
| ”P1” | 6.7 | 32.8 | 83.5 | 12.5 |
| ”P2” | 2.1 | 4.7 | 19.5 | 9.3 |
| ”P3” | 0.7 | X | 7.5 | 10.7 |
| ”P4” | 0.27 | 0.83 | 2.7 | 10.0 |

On the Kokkos part, the 2nd-order MUSCL (2D / 3D) performance are 2% to 5% slower compared to hand-written CUDA kernels in RamsesGPU, which is an impressive result for a less intrusive implementation.

### 2.12 SPECFFEM3D_GLOBE

Specfem3D is widely known code developed in MPI, OpenMP and CUDA. The two test cases tested here are those available in the git repository (*test small bench very simple earth* and *test small bench more complex earth*).

The code is running well on one node, with a speed-up up to 27 using a GPU versus a single P8 core. The constraint with Specfem3D is that the test case defines a number of MPI processes (which corresponds to the mesh partitioning). Moreover, we tried using the NVIDIA MPS (Multi-Process Service) for the P100 which would enable to run multiple CUDA kernels sent by different MPI processes on the same GPU. This lead to huge slow-down caused by memory copy to the GPU. This constraint has to be addressed in order to improve performance.

**Table 13.** Specfem3D: test small bench more complex earth

| Nodes | Processes/Node | GPUs | Time | Speed-up |
|---|---|---|---|---|
| 1 | 24 | 0 | 707.27 | 1.00 |
| 1 | 24 | 2 | 52.66 | 13.43 |
| 1 | 24 | 4 | 37.83 | 18.69 |

The code is generated with the PGI compiler, CUDA gencode60 and IBM Spectrum MPI. Results for a single node workload are available in 13.

Results are good as we have a 18.69× speed-up between P8 node and P8 node + 4GPUs. However, as we had a 13.43× using only 2GPUs, we could have expected at least a speed-up of 20 for using the 4 of them. A deeper analysis on this point has to be done. On top of that, a scalability test on multiple nodes using a bigger test case has to be considered.

### 2.13 YALES2

The main flow solver of YALES2 relies on hybrid MPI+OpenMP parallelism and on object-oriented Fortran. The project of the development team was to assess the usability of the GPUs by porting one of the most time consuming kernel (the conjugate gradient algorithm for the solving of the Poisson equation) to CUDA and by running a realistic test case. This work consisted in i) changing the conjugate gradient iteration to exhibit data parallelism, ii) writing a generic C to Fortran interface so that CUDA can access to the data structures of the code, iii) integrating the kernel in the full application. A work is ongoing for porting all the kernels to GPUs, this task is starting and may take a few months/years before it is completed.

The performances for the simulation of the flow around a 3D cylinder with 3.9 million cells are given in table 14. The speed-up is measured only for the conjugate gradient (CG) step. Interestingly, running with a single process and a single GPU is faster than using 4 processes. The changes in the CG step, to exhibit data parallelism, slow down the code by approximately 20% but this is largely compensated by the speed-up of the GPU. Running with 1 GPU per process lead to a speed-up close to 4.

**Table 14.** Yales2 simulation of the flow around a 3D cylinder performances on OpenPOWER

| Node | Code | SMT Mode | Processes/Node | Tasks/GPU | Speed-up |
|---|---|---|---|---|---|
| P8 | standard | 1 | 4 | 0 | 1.00 |
| P8+1P100 | CUDA | 1 | 1 | 1 | 1.16 |
| P8+4P100 | CUDA | 1 | 4 | 1 | 3.63 |

## 3 Conclusion and Future Work

First of all, the POWER8 processor has to be defined as a very easy to use processor. We clearly have seen over the last year an improvement in the compilation environment which helped all of our users to port their application on the machine very easily. Unfortunately, for now, the goal that we want to achieve (i.e. porting all our applications to GPUs using OpenMP) is still a bit far from us. Indeed, the main difficulty for the users on this platform is to understand the GPU and how to adapt their application to it. While all of the users managed to port their application on the POWER8, only a few applications were run on the GPUs with a relevant level of performance. The platform can achieve very high performances while using CUDA kernels, the IBM XL compiler is mainly used for the code using CUDA kernels and provides a good level of performances. First results on OpenACC show that we can also get good performances using this paradigm but the time to achieve the performance is not in days, but in weeks (and even sometimes months or years). However, we have seen over the past year that the PGI compiler really adapted to the POWER8 architecture and can achieve very high performance. The first results obtained on the platform are available in figure 1.

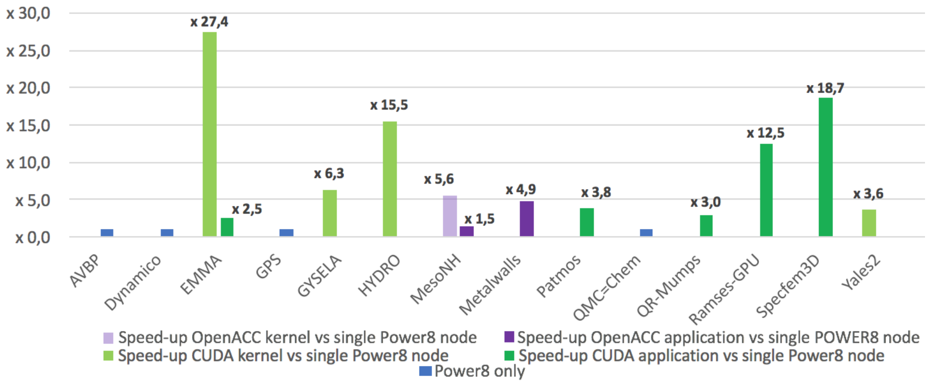

**Fig. 1.** First results obtained on Ouessant

OpenMP remains our main goal for legacy and portability of our applications, but we do have troubles at the moment using this paradigm on the platform. Indeed, the few tests we performed on the platform didn't give us good performances for GPU offloading, that's one of the reason why none of our users ported their application using OpenMP. On top of that, as the project started at the beginning of 2016, the OpenMP functionality were not available and that did not help the community to choose this paradigm. However, we still expect that this platform will manage to run OpenMP for GPUs and get the same level of performances than OpenACC. Those features should be implemented in the

IBM compiler by the end of the year. We also expect that modifying the code from OpenACC to write OpenMP kernel will not be too much time consuming.

The opening of the platform, since April 2017, to the full French scientific community will be a new opportunity for us to push for OpenMP and to continue working using this paradigm at the national level. On top of that, we will assess the scalability of the platform, using even larger test cases. The results obtained with PATMOS (having an almost perfect scalability up to 8 nodes) make us believe that this particular point should not be an issue.

Despite some porting troubles, the first results on the platform are very promising and we are looking forward to the next generation of OpenPOWER nodes.

Besides, we now have a focus on deep learning applications and the number of project that applies for the Ouessant platform using PowerAI is increasing. The first results are impressive and we are sure that this platform will help the artificial intelligence community to address new challenges.

## Acknowledgments

GENCI thanks all its partners within the project for their support as well as IBM and nVIDIA experts and all the developers that contributed to the work performed on this platform.

## References

1. Gourdain, N., Gicquel, L., Montagnac, M., Vermorel, O., Gazaix, M., Staffelbach, G., Garcia, M., Boussuge, J.F., Poinsot, T.: High performance parallel computing of flows in complex geometries - part 1: methods. Computational Science and Discovery **2**(November) (2009) 015003
2. Grasseau, G., Chamont, D., Beaudette, F., Bianchini, L., Davignon, O., Mastrolorenzo, L., Ochando, C., Paganini, P., Strebler, T.: Matrix element method for high performance computing platforms. Volume 664 of Journal of Physics: Conference Series., Bristol, Institute of Physics Publishing (2015) 092009–
3. Aubert, D., Deparis, N., Ocvirk, P.: EMMA: an adaptive mesh refinement cosmological simulation code with radiative transfer. **454** (November 2015) 1012–1037
4. P. Parnaudeau, A. Suzuki, J.M.S.E.: Gps: An efficient & spectrally accurate code for computing gross-pitaevskii equation. ISC-2015, Research Posters Session, July 12 - 16, 2015, Germany.
5. Pierre-Francois Lavallée, Guillaume Colin de Verdière, P.W.D.L.J.M.D.: Porting and optimizing hydro to new platforms and programming paradigms - lessons learnt
6. http://mesonh.aero.obs-mip.fr/mesonh53/MesoNHReferences
7. Brun, E., Chauveau, S., Malvagi, F.: Patmos: A prototype monte carlo transport code to test high performance architectures. In: M&C 2017 - International Conference on Mathematics & Computational Methods Applied to Nuclear Science & Engineering, April 16-20, Jeju, Korea (2017)
8. Moureau, V., Domingo, P., Vervisch, L.: Design of a massively parallel cfd code for complex geometries. Comptes Rendus Mcanique **339**(2) (2011) 141 – 148
9. Scemama, A., Caffarel, M., Oseret, E., Jalby, W.: Quantum Monte Carlo for large chemical systems: Implementing efficient strategies for petascale platforms and beyond. J. Comput. Chem. **34**(11) (jan 2013) 938–951
10. : Intel software development emulator. https://software.intel.com/en-us/articles/intel-software-development-emulator Accessed: 2017-04-26.
11. Agullo, E., Buttari, A., Guermouche, A., Lopez, F.: Implementing multifrontal sparse solvers for multicore architectures with sequential task flow runtime systems. ACM Trans. Math. Softw. **43**(2) (August 2016) 13:1–13:22
12. Agullo, E., Buttari, A., Guermouche, A., Lopez, F.: Task-based multifrontal QR solver for GPU-accelerated multicore architectures. In: HiPC, IEEE Computer Society (2015) 54–63
13. https://github.com/kokkos