Optimizing computing and energy performances on GPU clusters: experimentation on a PDE solver

S. Contassot-Vivier S. Vialle and T. Jost

***Abstract*—We present an experimental comparison between a synchronous and an asynchronous version of a same PDE solver on a GPU-cluster. In the context of our experiments, the GPU-cluster can be heterogeneous (different CPUs and GPUs). The comparison is done both on performance and energetic aspects.**

I. Motivations and objectives

Distributed architectures, like PC clusters, are a current and extensible solution to implement and to execute large and distributed algorithms and applications. However, modern PC clusters cumulate several computing architectures in each node. A PC cluster node has several CPU cores, each core supplies SSE units (small vector computing units sharing the CPU memory), and it is easy to install one or several GPU cards in each node (large vector computing units with their own memory). So, different kinds of computing *kernels* can be developed to achieve computations on each node. CPU cores, SSE units and GPUs have different computing and energy performances, and the optimal solution depends on the particular hardware features, on the algorithm and on the data size.

According to the algorithm used and to the chosen implementation, communications and computations of the distributed application can overlap or can be serialized. Overlapping communications and computations is a strategy that is not adapted to every parallel algorithm nor to every hardware, but it is a well-known strategy that can sometimes lead to serious performance improvements.

Moreover, although a bit more restrictive conditions apply on *asynchronous parallel algorithms*, a wide family of scientific problems support them. Asynchronous schemes present the great advantage on their synchronous counterparts to perform an implicit overlapping of communications by computations, leading to a better robustness to the interconnection network performances fluctuations and, in some contexts, to better performances [1].

So, some problems can be solved on current distributed architectures using different *computing kernels* (to exploit the different available computing hardware), with *synchronous* or *asynchronous* management of the distributed computations, and with *overlapped* or *serialized* computations and communications. These different solutions lead to various computing and energy performances according to the hardware, the cluster size and the data size. The optimal solution can change with these parameters, and applications users should not have to deal with these parallel computing issues.

LORIA, University Henri Poincaré, Nancy, France
IMS SUPELEC group, and AlGorille INRIA project team, France
AlGorille INRIA project team, France

Our long term objective is to develop auto-adaptive multi-algorithms and multi-kernels applications, in order to achieve optimal runs according to a user defined criterion (minimize the execution time, the energy consumption, or minimize the energy delay product...). However, the development of this kind of auto-adaptive solutions remains a challenge. The first step of our approach is to develop and experiment different versions of some classical HPC applications. Then, we will attempt to identify pertinent benchmarks, performance models and generic optimization rules. They will be the foundations of an auto-adaptive strategy for multi-algorithms and multi-kernels applications. The SPRAT framework [5] investigates this approach to dynamically choose CPU or GPU kernels at runtime, but considers only one computing node. We aim at being able to dynamically choose between CPU or GPU kernels and between *synchronous* or *asynchronous* distributed algorithms, according to the nodes used in an heterogeneous CPU+GPU cluster.

This article focuses on the development and experiment of a PDE solver on a heterogeneous GPU cluster, using synchronous or asynchronous distributed algorithms. We have already shown in [6] that the use of GPUs for the inner linear solver provides substantial gains. In this paper, we aim at finding the best communication scheme (sync or async) and implementation solution (CPU or GPU) according to a given context of GPU cluster (number of machines and homogeneity or heterogeneity). Both computing performances and energy consumption have been measured and analyzed in function of the cluster size and the cluster heterogeneity. Finally, different optimal solutions have been identified in this multi-parameter space: GPU cluster always appears more efficient up to 16 nodes and probably up to 33 nodes (see section A), but more experiments are required to validate this hypothesis. Moreover, depending on the respective numbers of fast nodes and slow nodes used, the most efficient solution will be either synchronous or asynchronous (see section B).

Section II introduces the synchronous and asynchronous algorithms of our distributed PDE solver, then sections III and IV introduce the heterogeneous GPU cluster we used and the experimental performance we measured. Section V summarizes our results and suggests the next steps of this research project.

II. Distributed PDE solver algorithm

Our benchmark application performs the resolution of PDEs using the multisplitting-Newton algorithm and an efficient linear solver. It is applied to a 3D transport model, described in [3], which simulates chemical species in shallow waters. To achieve this, the PDE system representing the model is linearized, discretized and its Jacobian matrix is computed (on the CPU). The Euler equations are used to approximate the derivatives. Since the size of the simulation domain can be huge, the domain is distributed among several nodes of a cluster. Each node solves a part of the resulting linear system and sends the relevant updated data to the nodes that need them. The general scheme is as follows:

- Rewriting of the problem under a fixed point problem formulation:
  $x = T(x), x \in \mathbb{R}$ where $T(x) = x - F'(x)^{-1}F(x)$

$\Rightarrow$ We get $F' \times \Delta X = -F$ with F' a sparse matrix

- F is distributed over the available nodes
- Each node computes a different part of ΔX using the Newton algorithm over its sub-domain
- F is updated with the entire X vector
- X is itself updated via messages exchanges between the nodes

In this process, most of the time is spent in the linear solver required for the computation of ΔX. So, it was implemented on GPU, using the biconjugate gradient algorithm. This algorithm was chosen because it performs well on non-symmetric matrices (on both convergence time and numerical accuracy), it has a low memory footprint, and it is relatively easy to implement.

A. GPU implementation of the linear solver

As GPUs have currently a limited amount of memory, the data representation is a crucial factor which requires very special care. Thus, our sparse matrices are stored in a compact way. Moreover, the memory accesses are treated carefully. To get coalesced memory accesses, our data structures are padded so that every line of a matrix starts on a multiple of 16 elements. When coalesced reads cannot be achieved in a vector, 1D texture cache is used to hide latencies as much as possible. We also use shared memory as a cache memory whenever it is possible in order to avoid costly slower reads to the device global memory. The different kernels used in the solver are divided to reuse as much data as possible at each call, hence minimizing transfers between the global memory and the registers.

B. Synchronous and asynchronous aspects

The asynchronism is inserted in the process depicted above at the level of the data exchanges of the X vector between the inner iterations performed within each time-step of the simulation. One synchronization is still required between each time step, as illustrated in Fig. 1. At this moment, the Jacobian matrix is locally updated for the computation of the next time-step.

The communications management is a bit more complex than in the synchronous version as it must enable sending and receiving operations at any time during the algorithm. Although the use of non-blocking communications seems appropriate, it is not sufficient, especially concerning receptions. This is why it is necessary to use several threads. The principle is to use separated threads

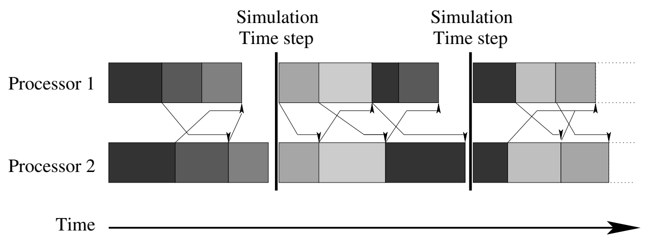


Fig. 1. Asynchronous iterations inside each time step of the computation

to perform the communications, while the computations are continuously done in the main thread without any interruption, until convergence detection. In our version, we used non-blocking sendings in the main thread and an additional thread to manage the receptions. It must be noted that in order to be as reactive as possible, some communications may be initiated by the receiving thread (for example to send back the local state of the unit).

Subsequently to the multi-threading, the use of mutex is necessary to protect the accesses to data and some variables in order to avoid concurrency and potentially incoherent modifications.

Another difficulty brought by the asynchronism comes from the global convergence detection. Some specific mechanisms must replace the simple global reduction of local states of the units to ensure the validity of the detection [2]. The most general scheme may be too expensive in some simple contexts such as local clusters. So, when some information about the system are available (for example bounded communication delay), it is often more pertinent to use a simplified mechanism whose efficiency is better and whose validity is still ensured in that context. Although both general and simplified schemes have been developed for this study, the performances presented in the following section are related to the simplified scheme which gave the best results.

III. Testbed introduction

The platform used to conduct our experiments is a set of two clusters hosted by SUPELEC in Metz. The first one is composed of 15 machines with Intel Core2 Duo CPUs running at 2.66GHz, 4GB of RAM and one Nvidia GeForce 8800GT GPU with 512MB per machine. The operating system is Linux Fedora with CUDA 2.3. The second cluster is composed of 17 machines with Intel Nehalem CPUs (4 cores + hyperthreading) running at 2.67GHz, 6GB RAM and one Nvidia GeForce GTX 285 with 1GB per machine. The OS is the same as the previous cluster. As the 8800GTs do not support double precision arithmetic, our program has been compiled with the `sm_11` flag for all the experiments.

Concerning the interconnection network, both clusters use a Gigabit Ethernet network. Moreover, they are connected to each other and can be used as a single heterogeneous cluster via the OAR management system.

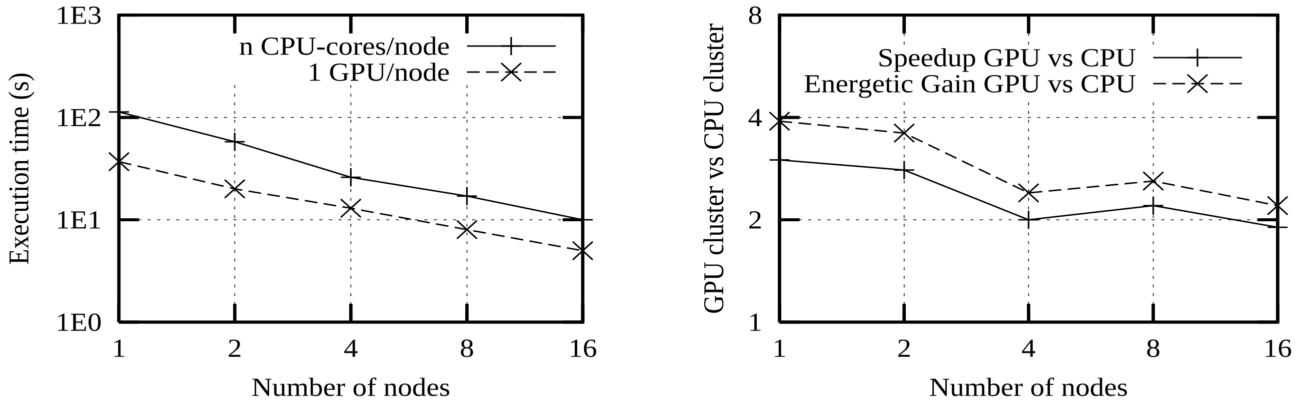


Fig. 2. Execution time of our PDE solver benchmark (synchronous version) on the multicore CPU cluster and on the GPU cluster (left), and speedup and energetic gain of the GPU cluster compared to the multicore CPU cluster (right)

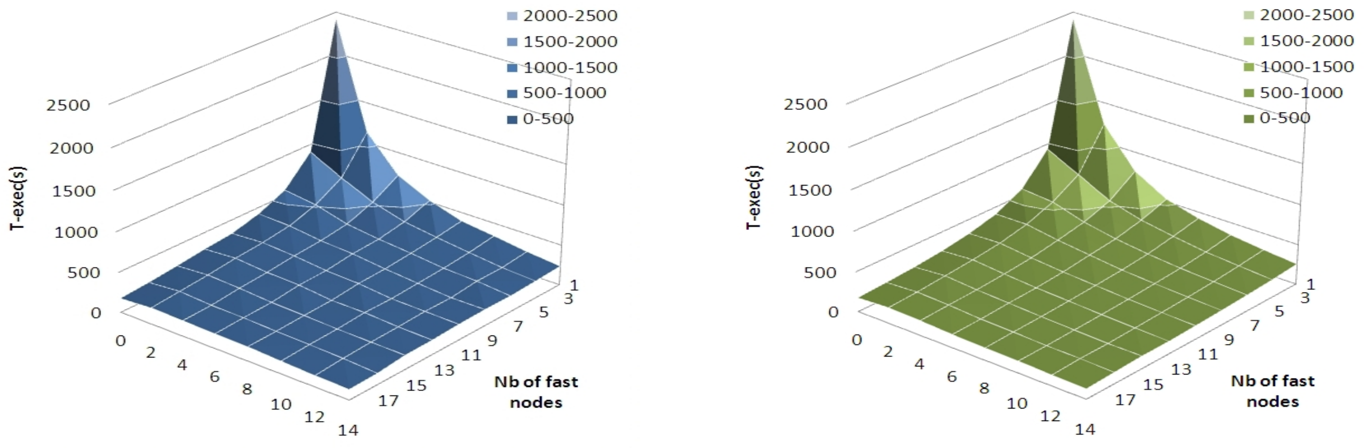


Fig. 3. Execution time of our PDE solver on a 100 × 100 × 100 problem, on the heterogeneous GPU cluster, with synchronous (left) and asynchronous (right) schemes

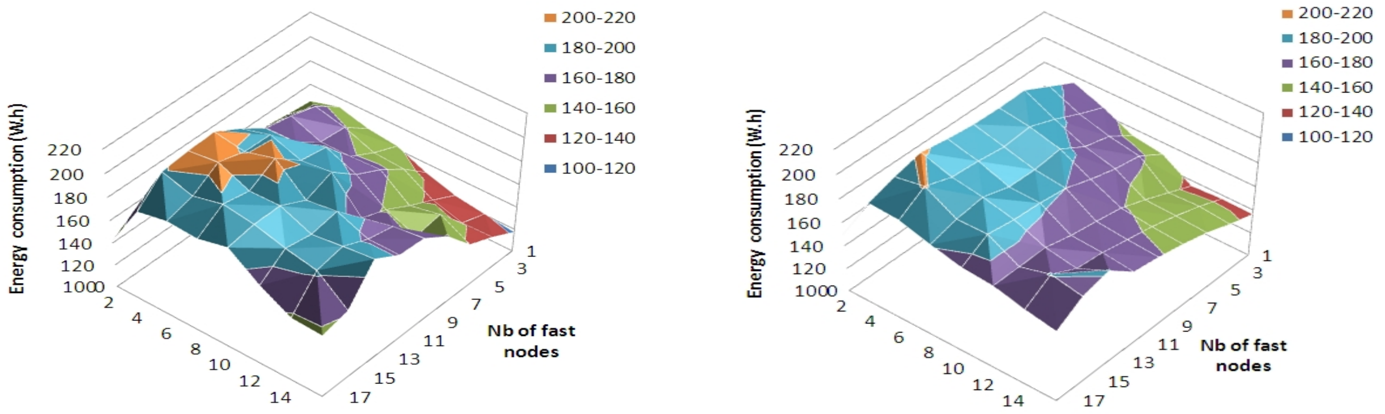


Fig. 4. Energy consumption of our PDE solver on a 100 × 100 × 100 problem, on the heterogeneous GPU cluster, with synchronous (left) and asynchronous (right) schemes

IV. Experiments

A. *GPU cluster* vs *CPU cluster*

Figure 2 (left) shows the execution times of our PDE solver benchmark (in synchronous mode) using either the multicore CPUs of the cluster (all the CPU cores on each node) or using the GPUs of the cluster (one CPU core and one GPU per node). We used only the most recent nodes of our cluster, composed of Intel Nehalem CPUs and Nvidia GeForce GTX 285 GPUs, appeared on the market approximately at the same date. Our benchmark runs faster with the GPUs, scaling up to 16 nodes, and consumes less energy (not shown on Fig. 2).

However, Fig. 2 (right) shows the performance and energetic gains of the GPU cluster *vs* the multicore CPU cluster. It can be seen that substantial gains are achieved with the use of GPUs instead of CPUs. But, a slight decrease appears when the number of processors increases. This is due to the fact that computation times are smaller on GPUs whereas the inter-node communication times remain unchanged and an additional overhead is induced by the data exchanges between GPUs and CPUs. Thus, the ratio

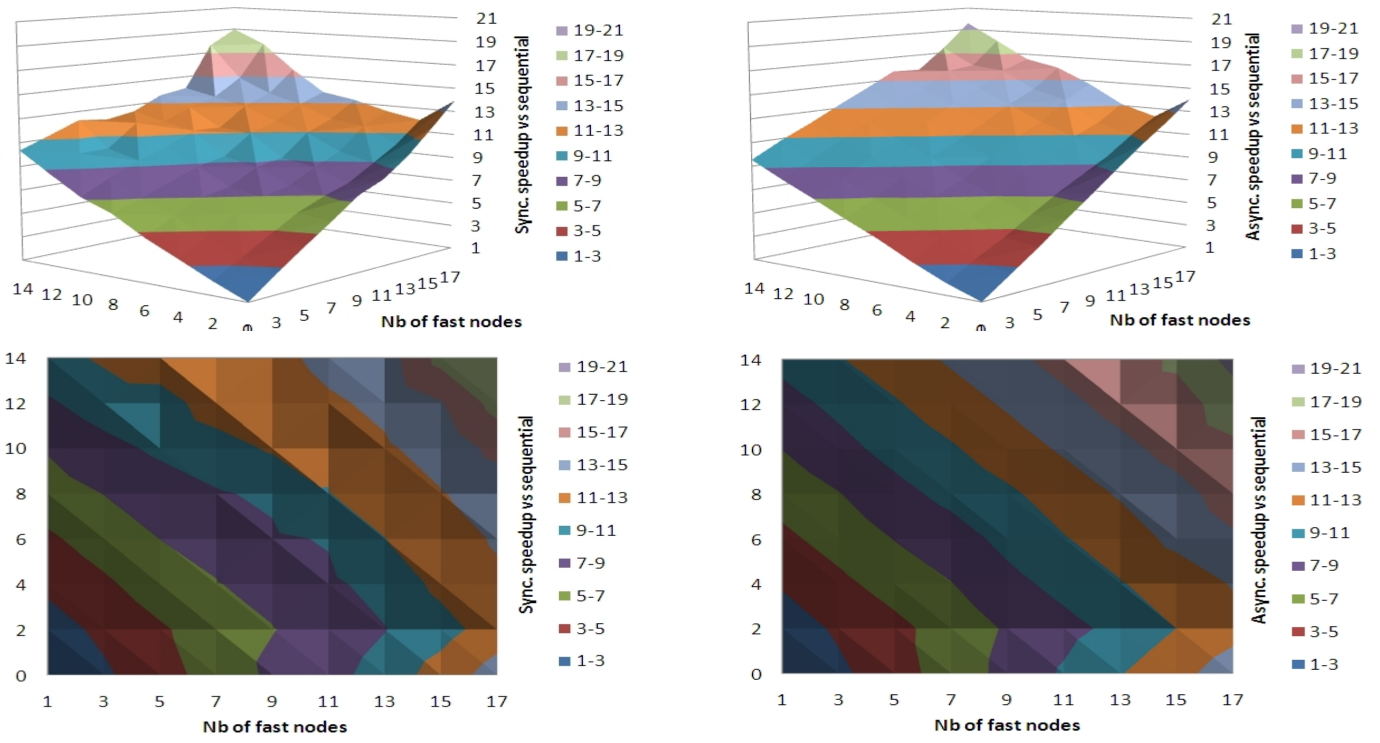


Fig. 5. Speedup of our PDE solver on a $100\times100\times100$ problem, on the heterogeneous GPU cluster, with synchronous (left) and asynchronous (right) schemes, compared to the sequential version

of communications over computations is larger on the GPU cluster. This results in a regular decrease of the speedup and energetic gain of the GPU cluster compared to the CPU cluster: GPU cluster supremacy decreases when the number of nodes increases.

B. *Synchronous* vs *asynchronous run on heterogeneous GPU cluster*

The first aspect addressed in our experiments is the evolution of the execution times according to the number of machines taken from the two available GPU clusters. As can be seen in Fig. 3, both surfaces are quite similar at first sight. However, there are some differences which are emphasized by the speedup distribution according to the sequential version, presented in Fig. 5. There clearly appears that the asynchronous version provides a more regular evolution of the speedup than the synchronous one. This comes from the fact that the asynchronous algorithm is more robust to the degradations of the communications performances. Such degradations appear when the number of processors increases, implying a larger number of messages transiting over the interconnection network and then a more important congestion. Thus, as in the previous comparison between GPU and CPU versions, the asynchronism puts back the performance decrease due to slower communications in the context of a heterogeneous GPU cluster.

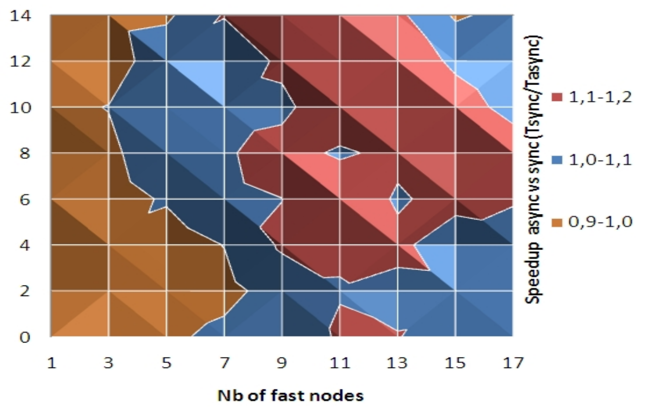
In order to precisely identify the contexts of use in which the asynchronism brings that robustness, we have plotted in Fig. 6 (left), the speedup of the asynchronous GPU algorithm according to its synchronous counterpart.

First of all, it can be seen that asynchronism does not always brings a gain. This is not actually a surprise because when the ratio of communications time over computations time is negligible, the impact of communications over the overall performances is small. So, on one hand the implicit overlapping of communications by computations performed in the asynchronous version provides a very small gain. On the other hand, the asynchronous version generally requires more iterations, and thus more computations, to reach the convergence of the system. Finally, the computation time of the extra iterations done in the asynchronous version is larger in this context than the gain obtained on the communications side. That context is clearly visible on the left part of the speedup surface, corresponding to a large pool of slow processors and just a few fast processors.

As soon as the communication-times to computation-times ratio becomes sensible, which is the case either when adding processors or taking faster ones, the gain provided by the asynchronism over the communications becomes more important than the iterations overhead, and the asynchronous version becomes faster. Unfortunately, it can be observed on Fig. 6 (left) that the separation between those two contexts is not strictly regular and studying the relative speedup map would be necessary in order to deduce a general rule to apply on this kind of PDE solver.

C. *Energetic aspects*

Concerning the energetic aspect, the first point concerns the gains obtained by the use of GPUs in place of CPUs, given in Fig. 2 (right). It can be seen that those gains



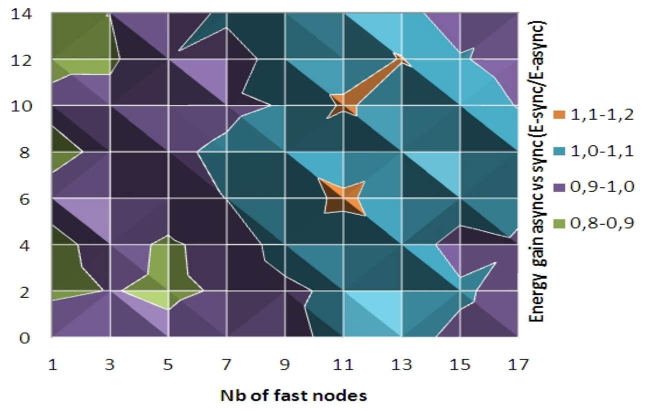


Fig. 6. Speedup (left) and energy gain (right) of asynchronous version *vs* synchronous version, on our heterogeneous GPU cluster

closely follow those of the speedup, with a nearly constant factor. That similarity of the two curves is quite obvious as the energy spent directly depends on the time of use. However, the vertical offset between the curves is a bit more surprising. This comes from the fact that the GPU and the CPU have different energy consumptions and computational powers at full load. Although the current GPUs have generally a larger consumption than the CPUs, they also have a larger computational power, and that last ratio is greater than the first one ($\frac{E_{GPU}}{E_{CPU}} < \frac{C_{GPU}}{C_{CPU}}$) at full load. Moreover, the total amount of energy used by one node is not fully spent in computations. In fact, two parts can be identified: the one actually used for the *computations* and another one for the *system* (minimal energy at idle time). So, the relative ratio of the *system* part over the *computation* part is lower when using a GPU than when using a CPU. Those two factors explain why the GPU version obtains a better energetic efficiency than the CPU one.

In order to get the complete energetic behavior of the couple algorithmic scheme - cluster, we have measured the energy consumption in function of the number of nodes and the algorithmic scheme used. The results are presented in Fig. 4. The first interesting point is that the energy consumption does not follow the performance behavior. This is explained by the performance speedup which follows a sub-linear trajectory when adding nodes in the system (see Fig. 5). Thus, multiplying the number of nodes by two does not reduce the computation time by two and the global energetic efficiency of the cluster decreases when the number of nodes increases. The second interesting point is the comparison between the synchronous and asynchronous energetic surfaces. As for the performances, asynchronism tends to be more robust as the surface is smoother and globally lower. However, here again there is no simple separation between the synchronous and asynchronous gains, as illustrated in Fig. 6 (right).

So, as for the performances, the study of such energetic gain maps will be necessary to design an optimization strategy for this kind of computing problem.

V. Conclusion and perspectives

A complete experimental study of a parallel PDE solver on a heterogeneous GPU cluster has been presented. The results show that GPUs are interesting both in terms of performance and energy consumption when the number of processors is not too high. Also, our experiments have pointed out that asynchronous algorithmic tends to bring a better scalability in such heterogeneous contexts of multi-level parallel systems, on the energetic side as well as on the performance one.

Finally, that study has also pointed out that the optimal choice of algorithmic scheme and hardware to use is not simple and requires a deeper study of the performance and energetic maps. Those results are a first step towards the design of performance and energetic models of parallel iterative algorithms on GPU clusters. In order to help this task of optimal choice, we also plan to study the *Energy Delay Product* [4] that allows to track a compromise between computational and energy efficiency.

References

[1] J. Bahi, S. Contassot-Vivier, and R. Couturier. Evaluation of the asynchronous iterative algorithms in the context of distant heterogeneous clusters. *Parallel Computing*, 31(5):439–461, 2005.

[2] J. Bahi, S. Contassot-Vivier, and R. Couturier. An efficient and robust decentralized algorithm for detecting the global convergence in asynchronous iterative algorithms. In *8th International Meeting on High Performance Computing for Computational Science, VECPAR'08*, pages 251–264, Toulouse, June 2008.

[3] J. Bahi, R. Couturier, K. Mazouzi, and M. Salomon. Synchronous and asynchronous solution of a 3D transport model in a grid computing environment. *Applied Mathematical Modelling*, 30(7):616–628, 2006.

[4] R. Gonzalez and M. Horowitz. Energy dissipation in general pupose microprocessors. *IEEE Journal of solid-state circuits*, 31(9), September 1996.

[5] K. Sato H. Takiza and H. Kobayashi. SPRAT: Runtime processor selection for energy-aware computing. In *Cluster Computing*, 2008.

[6] T. Jost, S. Contassot-Vivier, and S. Vialle. An efficient multi-algorithms sparse linear solver for GPUs. In *EuroGPU mini-symposium of the International Conference on Parallel Computing, ParCo'2009*, pages 546–553, Lyon, September 2009.