Impact of asynchronism on GPU accelerated parallel iterative computations

Sylvain Contassot-Vivier*[1,2], Thomas Jost†[2], and Stéphane Vialle‡[2,3]

[1]*Loria, University Henri Poincaré, Nancy, France*
[2]*AlGorille INRIA project team, France*
[3]*IMS Group, SUPELEC, France*

Abstract We study the impact of asynchronism on parallel iterative algorithms in the particular context of local clusters of workstations including GPUs. The application test is a classical PDE problem of advection-diffusion-reaction in 3D. We propose an asynchronous version of a previously developed PDE solver using GPUs for the inner computations. The algorithm is tested with two kinds of clusters, a homogeneous one and a heterogeneous one.

Keywords Parallelism, GPGPU, Asynchronism, Scientific computing

1 Introduction

Scientific computing generally involves a huge amount of computations to obtain accurate results on representative data sets in reasonable time. This is why it is important to take as much advantage as possible of any new device which can be used in the parallel systems and bring a sensible gain in performances. In that context, one of our previous works was focused on the use of clusters of GPUs for solving PDEs [7]. The underlying scheme is a two-stage iterative algorithm in which the inner linear computations are performed on the GPUs [6]. Important gains were obtained both in performance and energy consumption. In the meantime, we also showed in our works related to asynchronism in parallel iterative algorithms [5, 1] that this algorithmic scheme could be very interesting in some specific combinations of parallel system and algorithm. Moreover, we also identified the context in which this algorithmic scheme is advantageous compared to the synchronous one. As asynchronism allows an efficient and implicit overlapping of communications by computations, it is especially well suited to contexts where there is a sensible ratio of communication time according to the computation time. This is for example the case in large local clusters or grids where communications through the system are expensive compared to local accesses.

Our motivation for conducting the study presented in this paper comes from the fact that a local cluster of GPUs represents a similar context of costly communications according to computations. Indeed, the cost of data transfers between the GPU memory and the CPU memory inside each machine is added to the classical cost of local communications between the machines. So, we propose in this work to study the interest of using asynchronism in our PDE solver in that specific context.

The test application used for our experiments is the classical advection-diffusion-reaction problem in a 3D environment and two chemical species (see [3] for further details). Two series of experiments have been performed, one with a heterogeneous cluster with two couples of CPU-GPU, and another one with a homogeneous cluster.

The following section presents the algorithmic scheme of our iterative PDE solver together with the implementation sketch of the asynchronous version. Then, the experiments are presented and the results are discussed in Section 3.

2 Asynchronous PDE Solver

It is quite obvious that over the last few years, the classical algorithmic schemes used to exploit parallel systems have shown their limit. As the most recent systems are more and more complex and often include multiple levels of parallelism with very heterogeneous communication links between those levels, one of the major drawbacks of the previous schemes has become their synchronous nature. Indeed, synchronizations may sensibly degrade performances in large or hierarchical systems, even for local systems.

For a few years now, asynchronous algorithmic schemes have emerged, and although they cannot be used for all problems, they are efficiently usable for a majority of them. In scientific computing, it can be expressed only in iterative algorithms. Although those methods are generally slower than direct ones, they are often the only known way to solve some problems and they are also less memory consuming.

The asynchronous feature consists in suppressing any idle time induced by the waiting for the dependency data to be exchanged between the computing units of the par-

*Email: Sylvain.Contassotvivier@loria.fr
†Email: Thomas.Jost@loria.fr
‡Email: Stephane.Vialle@supelec.fr

allel system. Hence, each unit performs the successive iterations on its local data with the dependency data versions it owns at the current time. The main advantage of this scheme is to allow an efficient and implicit overlapping of communications by computations. On the other hand, the major drawbacks of asynchronous iterations are: a more complex behavior which requires a specific convergence study, and a larger number of iterations to reach convergence. However, the convergence conditions in asynchronous iterations are verified for numerous problems and, in many computing contexts, the time overhead induced by the additional iterations is largely compensated by the gain in the communications [1, 4]. In fact, as partly mentioned in the introduction, as soon as the frequency of communications relatively to computations is high enough and the communication costs are larger than local accesses, an asynchronous version may provide better performances than a synchronous version.

2.1 Multisplitting-Newton algorithm

There are several methods to solve PDE problems, each of them including different degrees of synchronism/asynchronism. The method used in this study is the multisplitting-Newton which allows for a rather important level of asynchronism. In that context, we use a finite difference method to solve the PDE system. Hence, the system is linearized, a regular discretization of the spatial domain is used and the Jacobian matrix of the system is computed at the beginning. The Euler equations are used to approximate the derivatives. The algorithmic scheme of the method is as follows:

- Rewriting of the problem under a fixed point problem formulation:
  $x = T(x), x \in \mathbb{R}$ where $T(x) = x - F'(x)^{-1}F(x)$ and F' is the Jacobian
- We get $F' \times \Delta X = -F$ with F' a sparse matrix (in most cases)
- F' and F are distributed over the computing units
- Each unit computes a different part of ΔX using the quasi-Newton algorithm over its sub-domain as can be seen in Fig. 1
- The local elements of X are directly updated with the local part of ΔX
- The non-local elements of X come from the other units using messages exchanges
- F is updated by using the entire vector X

2.2 Inner linear solver

The method described above is a two-stage algorithm in which a linear solver is needed in the inner stage. In fact, most of the time of the algorithm is spent in that linear solver. This is why we chose to use the most powerful elements of the parallel system on that part. Thus, the linear computations have been placed on the GPUs. Due to

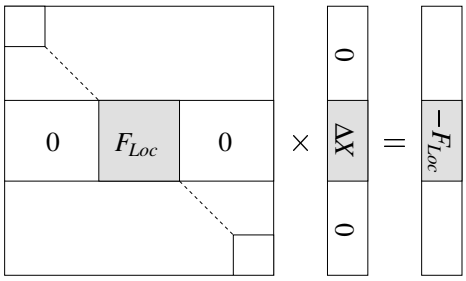

Figure 1: Local computations associated to the sub-domain of one unit

their regularity, those treatments are very well suited to the SIMD architecture of the GPU. Hence, on each computing unit, the linear computations required to solve the partial system are performed on the local GPU while all the algorithmic control, non-linear computations and data exchanges between the units are done on the CPU.

The linear solver has been implemented both on CPU and GPU, using the biconjugate gradient algorithm (see [6] for further details). This linear solver 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.

2.2.1 GPU implementation

Several aspects are critical in a GPU: the regularity of the computations and the memory which is of limited amount and the way the data are accessed. In order to reduce the memory consumption of our sparse matrix, we have used a compact representation, depicted in Fig. 2, similar to the DIA (diagonal) format in BLAS, but with several additional advantages. The first one is the regularity of the structure which allows us to do coalesced memory accesses most of the time. The second one is that it provides an efficient access to the transpose of the matrix, which is required in the biconjugate gradient method.

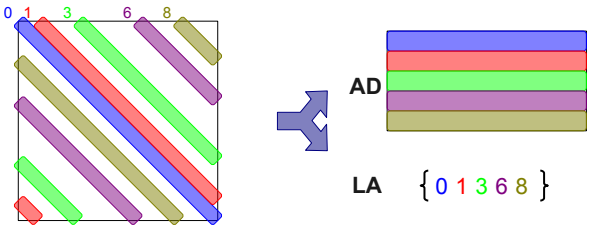

Figure 2: Compact and regular sparse matrix representation

In order to be as efficient as possible, the shared memory has been used as a cache memory whenever it was possible in order to avoid the slower accesses to the global memory of the GPU. 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. To get full details on those kernels, the reader should refer to [6].

2.3 Asynchronous aspects

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 data to the other units that need them. In the asynchronous version, this is that part which is performed asynchronously. One synchronization is still required between each time step of the simulation, as illustrated in Fig. 3.

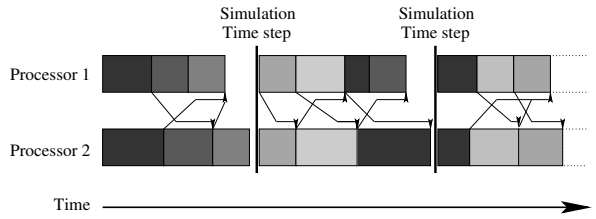

Figure 3: Asynchronous iterations inside each time step of the computation

At the practical level, the main differences with the synchronous version lie in the suppression of some barriers and in the way the communications between the units are managed. Concerning the first aspect, all the barriers between the inner iterations inside each time step of the simulation are suppressed. The only remaining synchronization is the one between each time step as pointed out above.

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 a multi-threaded programming is required. The principle is to use separated threads 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 sends 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 detection of the convergence. 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.

3 Experimental results

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. The second cluster is composed of 17 machines with Intel Nehalem CPUs (4 cores + hyper-threading) 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. 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.

In that hardware context, two series of experiments seemed particularly interesting to us. The first one consists in running our application for several problem sizes on one of the homogeneous clusters. We chose the most recent one, with the Nehalem CPUs and GTX 285 GPUs. The second series of experiments is similar to the first one except that instead of using only one cluster, we used the two clusters to obtain a heterogeneous system with 32 nodes.

The results are respectively presented in Table 1 and Table 3. The problem size indicated in the left column corresponds to the number of elements for each dimension of the spatial domain (3D). Thus, for a size of 50, there are 50^3 elements, and as we have two chemical species, the global linear system is a square matrix with 2×50^3 lines and columns. Fortunately, the local nature of dependencies in the advection-diffusion-reaction problem implies that only 9 diagonals in that matrix are non-zero.

Pb size	Sync	Async	Speed up	Gain (%)
50	16.52	14.85	1.11	10.10
100	144.52	106.09	1.36	26.59
150	392.79	347.40	1.13	11.55
200	901.18	866.31	1.04	3.87
250	1732.60	1674.30	1.03	3.36

Table 1: Execution times (in seconds) with the homogeneous cluster (17 machines).

The results obtained in that context are interesting but not as good as could be expected. The decrease of the gain when the problem size increases is quite natural as the ratio of communications according to the computations decreases and the impact of synchronizations becomes less preponderant over the overall performances. However, the rather limited maximal gain is a bit deceiving. In fact, it can be explained, at least partially, by the very high performance network used in the cluster, the rather small amount of data exchanged between the nodes and the homogeneity of the nodes and loads. In such a context, it is clear

that the synchronous communications are not so expensive compared to the extra iterations required by the asynchronous version. Also, it can be deduced that although the GPU $\leftrightarrow$ CPU data transfers play a role in the overall performances, their impact on our PDE solver is less important than one could have thought at first glance.

Two additional experiments have been done with the same cluster but with less processors in order to observe the behavior of our PDE solver when the number of processors varies. The results are provided in Table 2.

14 Machines of the newer cluster				
Pb size	Sync	Async	Speed up	Gain (%)
50	20.95	17.83	1.17	14.89
100	182.85	132.35	1.38	27.62
150	486.69	442.16	1.10	9.15
200	1101.29	1029.61	1.07	6.51
9 Machines of the newer cluster				
Pb size	Sync	Async	Speed up	Gain (%)
50	39.68	25.81	1.54	34.95
100	249.63	200.25	1.25	19.78
150	714.85	635.78	1.12	11.06
200	1599.01	1617.28	0.99	-1.14

Table 2: Execution times (in seconds) with 14 and 9 homogeneous machines.

Those results confirm the general trend of gain decrease when the problem size increases. It can also be observed that for smaller clusters, the limit of gain brought by asynchronism is reached sooner. And finally, there are some fluctuations in the gains which denote a complex behavior of this kind of algorithm according to the context of use. This would require a deeper study to identify the frontier of gain between synchronism and asynchronism in function of the number of processors and the problem size.

Concerning the second context of use, the heterogeneous cluster, the results presented in Table 3 are very surprising.

Pb size	Sync	Async	Speed up	Gain (%)
100	53.21	52.01	1.02	2.25
150	155.13	164.05	0.94	-5.75
200	322.11	395.11	0.81	-22.66

Table 3: Execution times (in seconds) with the heterogeneous cluster (15 + 17 machines).

In fact, the heterogeneity of the machines should imply different computation speeds and the synchronizations should induce a global slow down imposed by the slowest machine. Nevertheless, the results tend to show that the difference in the powers of the machines is not large enough to induce a sufficiently sensible unbalancing between them. Moreover, it clearly appears that the overhead of the asynchronism in terms of iterations is rapidly more important than the gain in the communications, leading to a loss in performances.

Also, another point that may explain the degraded performances of the asynchronous version in the heterogeneous cluster is that the GPU cards used in the older cluster do not support double precision real numbers. Thus, the program is compiled to use only single precision numbers, which divides the data size by a factor two and then also the communications volumes, reducing even more the impact of the communications on the overall execution times.

4 Conclusion

Two versions of a PDE solver algorithm have been implemented and tested on two clusters of GPUs. The conclusion that can be drawn concerning the interest of asynchronism in such a context of parallel system for that kind of application is mitigated. Some gains can be observed but they are rather limited. Moreover, the asynchronous version is not always better than the synchronous one, denoting a combination application-system not completely suited to that kind of algorithm.

However, as far as we know, that study is among the very firsts of its kind and it shows that this subject requires further works. For example, an interesting topic would be to precisely identify the areas in which one of the operating modes (sync or async) is better suited than the other to a given context of number of processors and problem size. In addition, using load-balancing in that context should also improve performances of both versions.

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] J. M. Bahi, S. Contassot-Vivier, and R. Couturier. Asynchronism for iterative algorithms in a global computing environment. In *The 16th Annual International Symposium on High Performance Computing Systems and Applications (HPCS'2002)*, pages 90–97, Moncton, Canada, June 2002.

[5] J. M. Bahi, S. Contassot-Vivier, and R. Couturier. *Parallel Iterative Algorithms: from sequential to grid computing*. Numerical Analysis & Scientific Computing Series. Chapman & Hall/CRC, 2007.

[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*, Lyon, Sept. 2009.

[7] T. Jost, S. Contassot-Vivier, and S. Vialle. On the interest of clusters of gpus. In *Grid'5000 Spring School 2010*, Lille, France, Apr. 2010.