# Impact of Asynchronism on GPU Accelerated Parallel Iterative Computations

Sylvain Contassot-Vivier, Thomas Jost, Stéphane Vialle

# 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
Sylvain.Contassotvivier@loria.fr
[2] AlGorille INRIA project team, France. Thomas.Jost@loria.fr
[3] SUPELEC - UMI 2598, France. Stephane.Vialle@supelec.fr

**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 (with different CPUs and GPUs).

**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 significant gain in performances. In that context, one of our previous works was focused on the use of clusters of GPUs for solving PDEs [19]. The underlying scheme is a two-stage iterative algorithm in which the inner linear computations are performed on the GPUs [18]. Important gains were obtained both in performance and energy consumption. Since the beginning of parallelism, several works related to asynchronism in parallel iterative algorithms (see for example [7, 10, 2]) have shown that this algorithmic scheme could be a very interesting alternative to classical synchronous schemes in some parallel contexts. Although a bit more restrictive conditions apply on *asynchronous parallel algorithms* [5], a wide family of scientific problems support them. Moreover, contexts in which this algorithmic scheme is advantageous compared to the synchronous one have also been identified. As asynchronism allows an efficient and implicit overlapping of communications by computations, it is especially well suited to contexts where there is a significant ratio of communication time relatively 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.

In fact, our long term objective is to develop auto-adaptive multi-algorithms and multi-kernels applications in order to achieve optimal executions according to a user defined criterion such as the execution time, the energy consumption, or the energy-delay product [15]. 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 a cluster with heterogeneous CPUs and GPUs.

The test application used for our experiments in this study is the classical advection-diffusion-reaction problem in a 3D environment and with two chemical species (see for example [17]). Two series of experiments have been performed, one with a homogeneous cluster and another one with a heterogeneous cluster with two couples of CPU-GPU. Both computing performances and energy consumption have been measured and analyzed in function of the cluster size and the cluster heterogeneity.

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 noticeably degrade performances in large or hierarchical systems, even for local systems (*i.e.* physically close nodes connected through a fast local network).

Since the very first works on asynchronous iterations [9, 20, 4], the interest of those schemes has increased in the last few decades [5, 8, 1, 13, 14]. Although they cannot be used for all problems, they are efficiently usable for a large part of them. In scientific computing, asynchronism can be expressed only in iterative algorithms. We recall that iterative methods perform successive approximations toward the solution of a problem (notion of convergence) whereas direct methods give the exact solution within a fixed number of operations. Although iterative methods are generally slower than direct ones, they are often the only known way to solve some problems. Moreover, they generally present the advantage of being 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 parallel 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 [2]. 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 [12] which allows for a rather important level of asynchronism [21]. 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 of each simulation time step. The Euler equations are used to approximate the derivatives. Since the size of the simulation domain can be huge, the domain is split and homogeneously 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 algorithmic scheme of the method is as follows:

- Initialization:
  - Rewriting of the problem under a fixed point problem formulation: $x = T(x), x \in \mathbb{R}^n$ 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 homogeneously distributed over the computing units
- Iterative process, repeated for each time step of the simulation:
  - Each unit computes a different part of $\Delta x$ 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 $\Delta x$
  - The non-local elements of $x$ come from the other units using messages exchanges
  - $F$ is updated by using the entire vector $x$

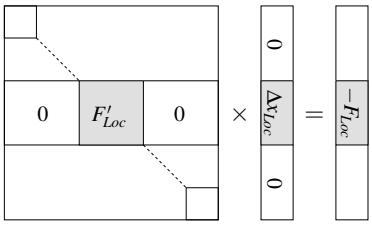


**Fig. 1.** Local computations associated with the sub-domain of one unit.

### 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 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 [11]. 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. At very early stages of development, we also tried to use the Bi-CGSTAB algorithm [22] and local preconditioners (Jacobi and SSOR), but this provided very little or no gain in terms of computing time and numerical accuracy, so we decided to keep the first, simpler solution.

**GPU implementation** Several aspects are critical in a GPU: the regularity of the computations, 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 [16] in BLAS [6], 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 as well as the matrix itself since the transpose is just a re-ordering of the diagonals. That last feature is essential as it is required in the biconjugate gradient method.

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 [18].

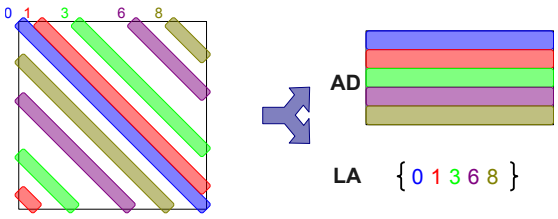


**Fig. 2.** Compact and regular sparse matrix representation.

### 2.3 Asynchronous aspects

In the asynchronous version, computing the state of the system (*i.e.* the concentration of the two chemical species across the space) at a given time of evolution (the EDP is time-dependent) is performed asynchronously. This typically involves solving several linear systems on each node, with some communications between each of these inner iterations. However, once this has been done, one synchronization is still required before beginning the next simulation time step, as illustrated in Fig. 3.

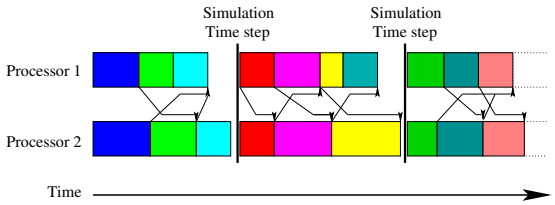


**Fig. 3.** Asynchronous iterations inside each time step of the computation.

In practice, 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 receives. 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 receives. It must be noted that in order to be as reactive as possible, some communications related to the control of the algorithm (the global convergence detection) may be initiated directly by the

receiving thread (for example to send back the local state of the unit) without requiring any process or response from the main thread.

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 convergence detection. To ensure the validity of the convergence detection, the simple global reduction of local states of the units must be replaced by some specific mechanisms. We have proposed a decentralized version of such a detection in [3]. 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 of convergence detection have been developed for this study, the performances presented in the following section are related to the simplified scheme which gave the best performances.

## 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 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. In all the experiments, our program has been compiled with the `sm_11` flag to be compatible with both kinds of GPUs, and using OpenMPI 1.4.2 for message passing. 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 initial 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 presented in Table 1 and Table 3. The problem size indicated in the left column corresponds to the number of spatial elements in the 3D domain. As we have two chemical species, for a volume of $50^3$ elements, the global linear system is a square matrix with $2 \times 50^3$ 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.

The results obtained in that context are interesting but not as good as could be expected. The decrease of the gain (last column in the tables) when the

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

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

problem size increases is quite natural as the ratio of communications relatively 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 relatively fast 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 through the Gigabit Ethernet network 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.

9 Machines of the newer cluster

| Pb size | Sync | Async | Speed up Async/Sync | Gain (%) |
|---|---|---|---|---|
| 50×50×50 | 39.68 | 25.81 | 1.54 | 34.95 |
| 100×100×100 | 249.63 | 200.25 | 1.25 | 19.78 |
| 150×150×150 | 714.85 | 635.78 | 1.12 | 11.06 |
| 200×200×200 | 1599.01 | 1617.28 | 0.99 | -1.14 |

14 Machines of the newer cluster

| Pb size | Sync | Async | Speed up Async/Sync | Gain (%) |
|---|---|---|---|---|
| 50×50×50 | 20.95 | 17.83 | 1.17 | 14.89 |
| 100×100×100 | 182.85 | 132.35 | 1.38 | 27.62 |
| 150×150×150 | 486.69 | 442.16 | 1.10 | 9.15 |
| 200×200×200 | 1101.29 | 1029.61 | 1.07 | 6.51 |

**Table 2.** Execution times (in seconds) with 9 and 14 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, which is not surprising according to the previous considerations.

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

| Pb size | Sync | Async | Speed up Async/Sync | Gain (%) |
|---|---|---|---|---|
| 100×100×100 | 53.21 | 52.01 | 1.02 | 2.25 |
| 150×150×150 | 155.13 | 164.05 | 0.94 | -5.75 |
| 200×200×200 | 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 perceptible unbalance between them. Moreover, it seems that the overhead of the asynchronism, due to the additional 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 fully support double precision real numbers. Thus, as previously mentioned, 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.

As can be seen in the first two series of experiments, there are some fluctuations in the gains with the homogeneous cluster and rather deceiving results with the heterogeneous cluster, which denote a complex behavior of this kind of algorithm according to the context of use. Those observations imply additional experiments to identify the frontier of gain between synchronism and asynchronism in function of the number of processors and the problem size. Such experiments are presented below.

The first aspect addressed in our additional experiments is the evolution of the execution times according to the number of machines taken from the two available GPU clusters for a fixed problem size. As can be seen in Fig. 4, 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, 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 the speedup of the asynchronous GPU algorithm according to its synchronous counterpart.

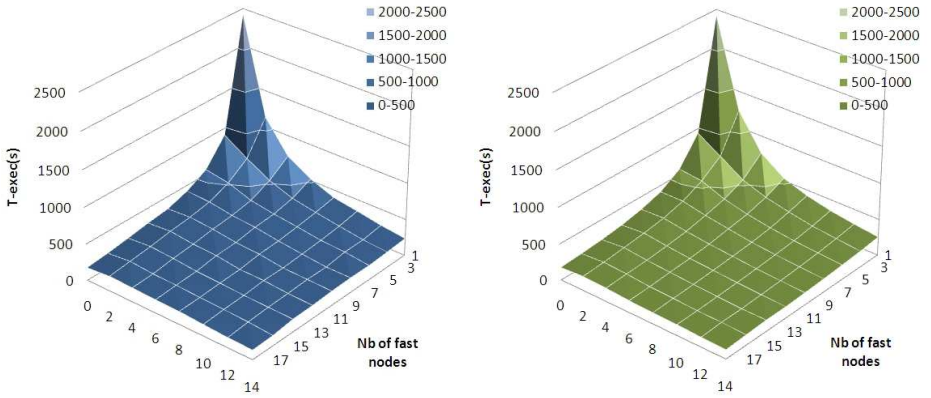


**Fig. 4.** Execution time of our PDE solver on a $100 \times 100 \times 100$ problem, with the heterogeneous GPU cluster, with synchronous (left) and asynchronous (right) schemes.

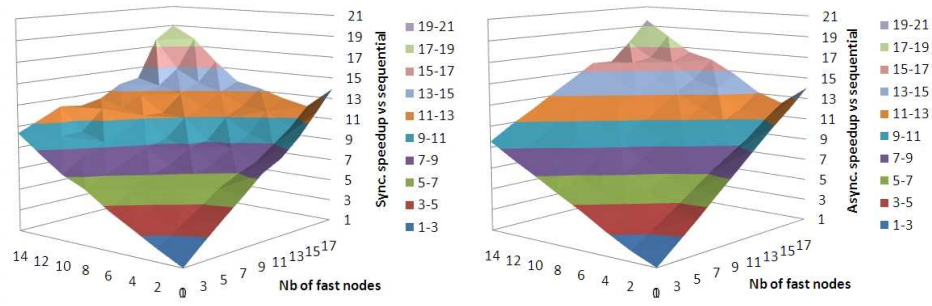


**Fig. 5.** Speedup of our PDE solver on a $100 \times 100 \times 100$ problem, with the heterogeneous GPU cluster, with synchronous (left) and asynchronous (right) schemes, compared to the sequential version.

First of all, we have the confirmation that asynchronism does not always bring a gain. As already mentioned, this comes from that fact that 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. Hence, in some contexts the computation time of the extra iterations done in the asynchronous version is larger than the gain obtained on the communications side. Such contexts are 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 significant, 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. In those cases, the gains obtained are quite significant as they can exceed 20% of the total execution time (see Tables 1 and 2). Unfortunately, it can be observed in the example of Fig. 6 that the separation between those two contexts is not strictly regular and studying the relative speedup map will be necessary in order to achieve an automatic selection of the most efficient operating mode of this kind of PDE solver in every context of use.

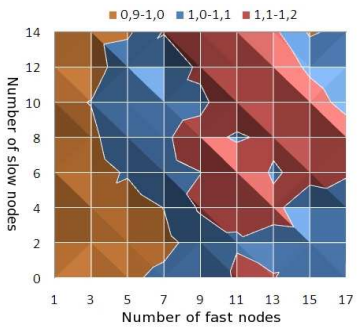


**Fig. 6.** Speedup of async. *vs* sync. version with the heterogeneous GPU cluster on a $100^3$ problem.

## 4 Conclusion and perspectives

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 that gains are not systematic. Some interesting gains ($\geq$ 20%) can be observed in some contexts and our experiments have pointed out that asynchronism tends to bring a better scalability in such heterogeneous contexts of multi-level parallel systems. However, the frontier between the two algorithmic schemes is not simple, implying that the optimal choice of algorithmic scheme and hardware to use in combination requires a finer model of performance.

As far as we know, that study is among the very firsts of its kind and it shows that this subject requires further works. The obtained results are quite encouraging and motivate us to design a performance model of parallel iterative algorithms on GPU clusters. That model should be based on the different activities (CPU and/or GPU computing, communications,...) during the application execution. An obvious perspective is the auto-tuning by the precise identification of the areas in which one of the operating modes (synchronous or asynchronous) is better suited than the other one 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.

## Acknowledgments

Authors wish to thank Région Lorraine for its support, and Patrick Mercier for his continuous technical management of the GPU clusters.

## References

1. D. Amitai, A. Averbuch, M. Israeli, and S. Itzikowitz. Implicit-explicit parallel asynchronous solver for PDEs. *SIAM J. Sci. Comput.*, 19:1366–1404, 1998.

2. 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.
3. 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.
4. G. M. Baudet. Asynchronous iterative methods for multiprocessors. *J. ACM*, 25:226–244, 1978.
5. D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, Englewood Cliffs NJ, 1989.
6. Basic linear algebra subprograms. `http://www.netlib.org/blas/`.
7. A. Bojanczyk. Optimal asynchronous newton method for the solution of nonlinear equations. *J. ACM*, 31:792–803, 1984.
8. R. Bru, V. Migallon, J. Penadés, and D. B. Szyld. Parallel synchronous and asynchronous two-stage multisplitting methods. *ETNA*, 3:24–38, 1995.
9. D. Chazan and W. Miranker. Chaotic relaxation. *Linear Algebra Appl.*, 2:199–222, 1969.
10. M. Cosnard and P. Fraignaud. Analysis of asynchronous polynomial root finding methods on a distributed memory multicomputer. *IEEE Trans. on Parallel and Distributed Systems*, 5(6), Juin 1994.
11. R. Fletcher. Conjugate gradient methods for indefinite systems. In G. Watson, editor, *Numerical Analysis*, volume 506 of *Lecture Notes in Mathematics*, pages 73–89. Springer Berlin / Heidelberg, 1976. 10.1007/BFb0080116.
12. A. Frommer and G. Mayer. On the theory and practice of multisplitting mehods in parallel computation. *Computing*, 49:63–74, 1992.
13. A. Frommer and D. B. Szyld. Asynchronous iterations with flexible communication for linear systems. *Calculateurs Parallèles, Réseaux et Systèmes Répartis*, 10:421–429, 1998.
14. A. Frommer and D. B. Szyld. On asynchronous iterations. *J. Comput. and Appl. Math.*, 123:201–216, 2000.
15. R. Gonzalez and M. Horowitz. Energy dissipation in general pupose microprocessors. *IEEE Journal of solid-state circuits*, 31(9), September 1996.
16. M. A. Heroux. A proposal for a sparse blas toolkit. SPARKER working note #2, Cray research, Inc, 1992.
17. W. Hundsdorfer and J. G. Verwer. *Numerical Solution of Time-Dependent Advection-Diffusion-Reaction Equations*, volume 33 of *Springer Series in Computational Mathematics*. Springer, 1st ed. 2003.
18. T. Jost, S. Contassot-Vivier, and S. Vialle. An efficient multi-algorithm sparse linear solver for GPUs. In *Parallel Computing : From Multicores and GPU's to Petascale*, volume 19 of *Advances in Parallel Computing*, pages 546–553. IOS Press, 2010.
19. 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.
20. J.-C. Miellou. Algorithmes de relaxation chaotique Ã  retards. *R.A.I.R.O.*, R-1:55–82, 1975.
21. D. B. Szyld and J. Xu. Convergence of partially asynchronous block quasi-newton methods for nonlinear systems of equations. *J. Comp. and Appl. math.*, 103:307–321, 1999.
22. H. A. van der Vorst. Bi-cgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 13(2):631–644, 1992.