Oil & Gas Science and Technology – Rev. IFP Energies nouvelles, Vol. 69 (2014), No. 4, pp. 753-766
Copyright © 2014, IFP Energies nouvelles
DOI: 10.2516/ogst/2013184

Dossier

Geosciences Numerical Methods
Modélisation numérique en géosciences

Survey on Efficient Linear Solvers for Porous Media Flow Models on Recent Hardware Architectures

Ani Anciaux-Sedrakian[1]*, Peter Gottschling[2,3], Jean-Marc Gratien[1] and Thomas Guignon[1]

[1] *IFP Energies nouvelles, 1-4 avenue de Bois-Préau, 92852 Rueil-Malmaison Cedex - France*
[2] *SimuNova, Helmholtzstr. 10, 01069 Dresden - Germany*
[3] *Institute Scientific Computing, TU Dresden, 01062 Dresden - Germany*
e-mail: ani.anciaux-sedrakian@ifpen.fr - peter.gottschling@simunova.com - jean-marc.gratien@ifpen.fr - thomas.guignon@ifpen.fr

* Corresponding author

Résumé — Revue des algorithmes de solveurs linéaires utilisés en simulation de réservoir, efficaces sur les architectures matérielles modernes — Depuis quelques années, en calculs haute performance les constructeurs ont recours de plus en plus à des architectures basées sur des unités de calculs multicœurs éventuellement accélérées avec des cartes de type GPGPU (*General Purpose Processing on Graphics Processing Units*). L'interêt de telles architectures offrant un grand nombre d'unités de calcul pourrait être grand pour le domaine de la simulation d'écoulements multiphasiques en milieu poreux, utilisée par exemple dans les applications de type séquestration géologique du CO_2 ou simulateur de récupération avancée de pétrole dans des réservoirs. Il faut néanmoins vérifier si les algorithmes des logiciels actuels sont adaptés pour être efficaces avec ces nouvelles technologies.
La résolution de grands systèmes linéaires creux constitue souvent la partie la plus coûteuse des simulateurs d'écoulement en milieu poreux. En effet, ces systèmes sont souvent mal conditionnés dû au caractère souvent très hétérogène et anisotrope des données géologiques. Les solveurs linéaires constituent pour ces raisons un point crucial pour les performances de ces simulateurs. Dans cet article, nous proposons un panorama des différents algorithmes de solveurs linéaires et de préconditionneurs utilisés dans nos applications. Nous analysons leur efficacité numérique et leur performance en fonction de différentes configurations matérielles. Nous proposons une nouvelle approche, basée sur la programmation hybride, performante sur des architectures hétérogènes à base de processeurs multicœurs ou d'accélérateurs de type GPGPU. Cette approche est validée dans l'implémentation d'un BiCGStab préconditionné avec des algorithmes de type ILU(0), BSSOR, préconditionneur polynomial ou CPR-AMG. Des tests de performances ont alors été effectués sur differents cas d'études d'écoulement en milieu poreux, utilisant des maillages de grande taille.

Abstract* — *Survey on Efficient Linear Solvers for Porous Media Flow Models on Recent Hardware Architectures — *In the past few years, High Performance Computing (HPC) technologies led to General Purpose Processing on Graphics Processing Units (GPGPU) and many-core architectures. These emerging technologies offer massive processing units and are interesting for porous media flow simulators may used for CO_2 geological sequestration or Enhanced Oil Recovery (EOR) simulation. However the crucial point is "are current algorithms and software able to use these new technologies efficiently?"*
The resolution of large sparse linear systems, almost ill-conditioned, constitutes the most CPU-consuming part of such simulators. This paper proposes a survey on various solver and preconditioner

algorithms, analyzes their efficiency and performance regarding these distinct architectures. Furthermore it proposes a novel approach based on a hybrid programming model for both GPU and many-core clusters. The proposed optimization techniques are validated through a Krylov subspace solver; BiCGStab and some preconditioners like ILU0 on GPU, multi-core and many-core architectures, on various large real study cases in EOR simulation.

INTRODUCTION

In basin modeling or reservoir simulations, multiphase porous media flow models with highly heterogeneous data and complex geometries, lead to solve complex non-linear Partial Differential Equations (PDE) systems. These PDE are discretized with a cell-centered finite volume scheme in space and a fully implicit scheme in time, leading to a non-linear system which is solved with an iterative Newton solver. At each Newton step, the system is linearized, and the generated linear system is solved with a Bi-Conjugated Gradient Stabilized (BiCGStab) [1] or Generalized Minimal RESidual (GMRES) [1] algorithm, well suited for large, sparse and unstructured systems. This resolution phase constitutes the most expensive part of the simulation, representing nearly 60% to 80% of the total simulation time. The efficiency of the linear solvers is therefore a key point of the simulator's performance. As the cost of these iterative algorithms depends directly on the number of iterations required for convergence, the choice of a robust parallel preconditioner and appropriate solver options is important. Indeed these appropriate options have a strong impact on the cumulative number of iterations for the whole simulation.

Furthermore, to continue increasing the performance of architecture following the Moore's law; reducing the energy consumption, hardware developers have to design heterogeneous architectures based on multi-core CPU enhanced with GPU accelerators. Indeed, the multi-GPU architectures are economically more interesting than the multi-core ones.

However, these architectures introduce new challenges in terms of algorithms and system software design; algorithms need to be adapted or sometimes completely rewritten according to the targeted architecture. Moreover, the bottleneck of each algorithm can vary for different hardware configurations. For instance, with GPU accelerator (heterogeneous multi-GPU) the main issue is to optimize the data transfer between the host memory and the remote local memory of the GPU while in the case of multi-core architecture, the major problem is to manage the concurrent access of the cores to the memory under the constraint of limited memory bandwidth. The heterogeneous multi-GPU architecture combines both constraints; low latency for data transfer between the CPU and the accelerators (GPU or many-core cards) and concurrent access of the cores to the memory.

In this paper, we focus on different types of preconditioners, on their implementation with different libraries (Hypre, Petsc, PMTL4, IFPSolver, MGCSolver) and we study their performance regarding different hardware architectures. We show how the choice of appropriate solver options and parallel preconditioners is tightly coupled to the hardware configuration. In the first section, we present some hardware and software considerations in the context of GPUs. The second section is a review of the main preconditioner algorithms commonly used with Krylov solvers. The third section describes the porting of preconditioned BiCGStab to GPU. Results are gathered and analyzed in the forth section, while section five concludes this work.

1 HARDWARE CONTEXT

In the exascale computing roadmap, using accelerators (GPUs and many-core) is a key point. These kinds of architectures offer a very high peak of floating-point operations per second, a very high peak memory bandwidth, for very interesting ratios of Flop per watt and Flop per euro.

In the past twenty years, GPU have evolved from fixed-function processors to massively parallel floating point engines. Hence, the idea of General-Purpose Computation on Graphic Processing Units (GPGPU) emerged to take advantage of the processing power of GPU for non-graphical tasks. General purpose programming tools like BrookGPU [2], CUDA [3], or OpenCL [4], ease GPU use for all programmers. However to fully benefit from the GPU, computational power still requires that algorithms exhibit a high degree of parallelism and regular data structures. This stems from the specific GPU hardware architecture: hundred basic computation cores (also called Stream Processor; SP) handle arithmetic and load/store operations without any flow control. The latter is handled by an additional unit called sequencer. This leads to a programming model called Data Parallel [5, 6]. Such programming and execution model is similar to those used in the 80's and 90's on massively parallel supercomputers [5, 6]. Nvidia GPU

provide a massively multithreaded execution model where threads are identical instruction flows working with different data.

Here, we focus on the two most widely available and used Nvidia GPU architectures at the time of writing this manuscript, namely: FERMI and Kepler. The FERMI architecture provides 448 SP for C/M2070 and up to 512 SP for M2090 models, grouped into 14 SM (16 for M2090) each containing 32 SP. With the Kepler architecture, NVIDIA provides the SMX (Streaming Multiprocessor eXtreme) architecture. Each SMX contains 192 cores, each capable of completing a single-precision floating-point multiple and adds (fused, to avoid truncating the intermediate result) in every clock cycle. In the K20 product, 13 of these SMX units are combined to give a total of 2 496 cores, giving a peak performance of 3.52 TFlops in single or 1.17 TFlops in double precision.

On the software side, CUDA and OpenCL have emerged as the leaders of general purposes software tools. While CUDA only provides support for Nvidia GPU, OpenCL is more generic and support Nvidia, AMD GPU and also multi-core CPU. In this work, we use the CUDA programming environment. Moreover, we consider that the GPU and the CPU process disjoint parts of the code. In this context, the GPU role is to decrease the computation time by assisting the CPU for the dedicated parts, even when taking into account the data transfer penalty.

2 PRECONDITIONERS OVERVIEW

In multiphase porous media flow models, the discretization and the linearization of the PDE lead to large linear systems to compute the pressure or/and the temperature which are elliptic/parabolic unknowns and the saturations or/and the compositions which are hyperbolic unknowns. These linear systems, often ill-conditioned due to the heterogeneous and anisotropic geological data, are classically solved with iterative Krylov subspaces methods such as GMRES, CGS, BiCGStab.

Efficient preconditioners are necessary to improve the linear systems' condition numbers so that the used iterative solvers can converge with a reasonable number of iterations.

For a given sparse linear system, there is often no single preconditioner which yields the best performance on all computer architectures. The efficiency of preconditioners depends numerically on the problem size, on the heterogeneity, the discontinuities and the anisotropies of the problem. But the efficiency of their implementation depends on the target hardware architecture. Some algorithms may be numerically efficient and help to reduce the number of iterations considerably, but they may be difficult to parallelize and not efficient in terms of flops on certain hardware configurations. Even more, to have performance on hardware configurations like GPGPU, it is preferable to have algorithms offering a high degree of parallelism at a fine-grain level, while with multi-core configuration, to avoid memory access concurrency problems, it is preferable to have algorithms offering a coarse-grain level degree of parallelism. Therefore, the compromise between the numerical efficiency of a preconditioner and the performance of its algorithm's implementation for each hardware configuration must be studied. In this paper, we focus on BiCGStab linear solver (used for both symmetric and non-symmetric systems) preconditioned by Neumann Polynomial, BSSOR, ILU, AMG and CPR-AMG [7]. The strong and the weak points of these preconditioners are discussed below. In our study, we use the before-mentioned preconditioners implemented in Hypre [8], PETSc [9], PMTL4 [10], MGCSolver [11] and IFPSolver [12] libraries described in Section 4.3.

2.1 Neumann Polynomial

The Neumann Polynomial preconditioner is the simplest polynomial preconditioner. Let ω be the largest eigenvalue of matrix A and $P_m(A)$ a polynomial of degree m then $M^{-1} = P_m(A)$ is defined as:

$$P_m(A) = \omega(I + (I - \omega A) + ((I - \omega A)^2 + ... + (I - \omega A)^m)$$

This preconditioner – based on matrix-vector products – is very well parallelisable on every kind of hardware configurations. Nevertheless, it is not numerically robust because it does not reduce significantly the condition number of the matrix. For ill-conditioned systems provided by reservoir simulations, it is often not efficient enough numerically to ensure the Krylov solver convergence. On multi-core architectures, this preconditioner is usually not competitive with other preconditioners. However, as its construction is not expensive and it can be parallelized easily at a fine-grain level, the interest to it has increased again few years ago because it is easy to implement for GPGPU, its algorithm being well suited for the GPGPU's data parallel programming model. Thus, the fact that it requires many more iterations to converge than most of other preconditioners is compensated by the performance of efficient matrix-vector product implementations.

The behaviour of this preconditioner and its handling on the different architectures is discussed in the numerical experimentation Section 4.

2.2 BSSOR Preconditioner

The Block Symmetric Successive Over-Relaxation (BSSOR) preconditioner consists of applying one or several iterations of the block Symmetric Successive Over Relaxation (SSOR) method well described in [1].

Let us consider the linear system $A\mathbf{x} = \mathbf{b}$ where A is a matrix, $\mathbf{x}$ and $\mathbf{b}$ vectors. Be L, D, and U the lower, diagonal, and the upper parts of A respectively so that $A = L + D + U$. The backward Gauß-Seidel iteration computes $\mathbf{x}^{BGS}$ from $\mathbf{x}^k$ by solving the upper triangular system:

$$L\mathbf{x}^k + D\mathbf{x}^{BGS} + U\mathbf{x}^{BGS} = \mathbf{b}$$

with a backward substitution algorithm.

The forward iteration calculates $\mathbf{x}^{FGS}$ from $\mathbf{x}^k$ by solving the lower triangular system:

$$L\mathbf{x}^{FGS} + D\mathbf{x}^{FGS} + U\mathbf{x}^k = \mathbf{b}$$

with a forward substitution algorithm.

The symmetric version of the method applies successively the backward then the forward iteration as follows:

$$L\mathbf{x}^k + D\mathbf{x}^{BGS} + U\mathbf{x}^{BGS} = \mathbf{b}$$

$$L\mathbf{x}^{SGS} + D\mathbf{x}^{SGS} + U\mathbf{x}^{BGS} = \mathbf{b}$$

The relaxation step consists in computing the iterate $k+1$ as the weighted average of the iterate k and the symmetric Gauß-Seidel solution $\mathbf{x}^{SGS}$ with a relaxation factor ω: $\mathbf{x}^{k+1} = \omega\mathbf{x}^k + (1-\omega)\mathbf{x}^{SGS}$

Although backward or forward substitution algorithms are sequential algorithms, the SSOR algorithm is interesting in its block version because it can be parallelized at a fine-grain level by the means of a block-coloring renumbering procedure (graph-coloring approach). This consists in introducing the dual graph of the matrix and partitioning the vertices into independent blocks of different colors.

Let $G(A)$ be the dual graph of a matrix $A = (a_{i,j})$. The vertices of $G(A)$ represent the rows (resp. columns) of A. The edges (i,j) connecting the vertices i and j represent the non-zero elements $a_{i,j} \neq 0$ of A. The coloring algorithms assign to each vertex a color that is different from all its neighbours' color. Thereby, the number of colors is kept as small as possible by heuristics (finding a coloring with minimal colors is an NP-complete problem [13]). We can easily notice then, that in the backward or forward substitution algorithms, operations on rows of the same color are completely independent as they depend only on vector components of colors different from the current color.

2.3 Incomplete LU Factorization: ILU

The Incomplete LU factorization preconditioners – well described in [38] – are very common preconditioners in reservoir or basin simulators. They apply an incomplete LU factorization of the matrix A to compute a sparse lower triangular matrix L and sparse upper triangular one U such that $A \approx LU$. The factorization is incomplete in the sense that the non-zero LU factors not belonging to the original matrix pattern – the so called fill-in entries – may be dropped. In the zero degree incomplete version, ILU(0), all these entries are dropped. In the more accurate but more expensive version ILU(k,τ), they are dropped regarding two parameters: a maximal fill-in level factor k and a drop-in threshold τ to filter small entries [1, 37]. The preconditioning operation solves $LU.\mathbf{x} = y$ with a backward substitution followed by a forward substitution [1].

This preconditioner, well known to be efficient for standard cases (*i.e.* not ill-conditioned and moderate problem size), is not naturally parallel, since its algorithm is recursive. This algorithm can be parallelized at coarse or fine-grain level with various graph renumbering techniques. In this paper, we study three different strategies of parallelization: the first one is a graph-coloring approach allowing for a high level of parallelism at fine-grain level, and two others for coarse-grain parallelism, the block Jacobi approach and the domain distribution technique, based on 1D, 2D or 3D partitions generated *via* the libraries like Metis [14] or IFPPartitioner [15].

The graph coloring technique, generally used to improve the level of parallelism at a fine-grain level, is efficient in iterative solution methods for example in the SPMD context. However, in the ILU(0) context, this renumbering technique, modifying the shape of the original matrix graph, has an influence on the rate of dropped fill-in entries during the incomplete factorization. This well-known renumbering effect may decrease the numerical efficiency of the preconditioner, as it may increase the number of iterations to achieve convergence compared to the factorization with the natural ordering. This strategy is adopted in MCGSolver to take advantage of GPU accelerators that require fine-grain parallelism.

Several tests on different cases showed that the block Jacobi ILU(0) preconditioner may suffer from numerical unstability since the links between each domain are not taken into account. To achieve a fully parallel version of the ILU factorization, that takes into account all links between adjacent domains while preserving the parallel performance, the IFPSolver renumbers cells and groups them in interior ones – those not connected to other

sub-domains – and interface ones – connected to other sub-domain – which are handled in a Schur renumbering style. The interface cells have been sub-grouped into three kinds of interface cells:

- the upstream cells, only connected to domains with a higher rank than the current domain;
- the mixed cells, connected to both domains with rank higher and lower than the current domain;
- the downstream cells, only connected to domains with rank lower than the current domain.

This renumbering technique allowed us to improve the degree of parallelism at coarse-grain level. The computations on interior cells are independent and may be executed in parallel. We obtained a second level of parallelism on interface cells, computing upstream cells then downstream cells independently and optimizing the communication needed to handle the dependencies between sub domains. The whole mechanism of this optimization, applied in IFPSolver, is discussed in detail in [16].

2.4 Algebraic MultiGrid

Algebraic MultiGrid (AMG) [35, 37] is a robust preconditioner for elliptic problems. It is appreciated for its extensibility qualities with M-matrix systems: the number of iterations required to converge only depends minimally on the problem size and can be entirely size-independent. This is an important feature for massive parallel applications. However, its construction, much more expensive than traditional ILU preconditioner often suffers of scalability on large number of processors.

The main idea of Algebraic MultiGrid is to remove smooth error components on the coarser grid. This is done by solving the residual equation on a coarse grid and then interpolating the error correction onto the finest grid. The coarsening scheme is the major and crucial part of the AMG preconditioner for both the sequential and the parallel version.

AMG is based on the coarsening heuristics. The traditional coarsening scheme proposed by Ruge and Stüben (RS) [19], tends to work well for two spatial dimensions (2D) problems. Several works, concerning coarsening algorithms, are done in order to adapt them to 3D problems. The following coarsening algorithms, well-known for three-dimensional (3D) problems, are implemented for the parallel environment. The first one, Parallel Modified Independent Set (PMIS) algorithm [20], is a modification of an existing parallel algorithm; CLJP. In this algorithm, the coarse-grid result is independent of the number of processors and the distribution of the points across processors. The second one, the Hybrid Modified Independent Set (HMIS) algorithm [20], is obtained by combining the PMIS algorithm with a one-pass RS scheme.

A depth study in [20] is done concerning the effect of different coarsening schemes on the multi-core architecture, using Hypre BoomerAMG. In our situation, PMIS and HMIS Boomer AMG coarsening algorithms give almost the same result for both total number of solver iterations and total solver time.

2.5 CPR-AMG

The CPR-AMG is a two-stage preconditioner which is introduced by Lacroix *et al.* [7]. This preconditioner, based on the CPR[1] accelerations, extracts and solves pressure subsystems. Then, residuals associated with the computed solution are corrected by an additional preconditioning step on the whole system.

CPR-AMG combines one V cycle of AMG (we can use one of the BoomerAMG from the HYPRE library, SAMG, or at some point PMTL4 AMG) preconditioner on a pressure equation together with an ILU(0) or BSSOR preconditioner on the full system.

A basic presentation of this two-stage preconditioner could be defined as follows: let $A\mathbf{x} = \mathbf{b}$ be the linear system to solve, where A represents the full system with A_{pp}; pressure block coefficients, A_{ss}; non-pressure block coefficients (saturation block[2] or saturation and concentration block[3]) and A_{ps}, A_{sp}; the coupling coefficients:

$$\mathrm{A} = \begin{pmatrix} A_{pp} & A_{ps} \\ A_{sp} & A_{ss} \end{pmatrix}$$

$$\mathbf{x} = \begin{pmatrix} x_p \\ x_s \end{pmatrix} \text{ and } \mathbf{b} = \begin{pmatrix} b_p \\ b_s \end{pmatrix}$$

- apply ILU(0) (or BSSOR) on the full system; $ILU0(A)\mathbf{x}^{(1)} = \mathbf{b}$
- compute the new residual; $r_p = b_p - A_{pp}x_p^{(1)} - A_{ps}x_s^{(1)}$
- apply AMG-Vcycle on the residual pressure equation; $AMG(A_{pp})x_p^{(2)} = r_p$
- correction of the pressure unknowns; $x_p = x_p^{(1)} + x_p^{(2)}$

This preconditioner is the most expensive preconditioner which permits to decrease the number of iterations

[1] CPR intends to decompose the pressure part (elliptic unknowns) and approximately solve it. The pressure solution is used to constraint the residual on the full system, thus achieving a more robust and flexible overall solution strategy.
[2] In black-oil context.
[3] In compositional context.

to the convergence considerably. In contrast to the Neumann Polynomial, CPR-AMG is expensive to build and apply, but it offers a very fast solution. Thanks to CPR accelerator and to AMG, large-scale and complex reservoir problems, in fully implicit formulation, are solved efficiently.

3 BI-CGSTAB ACCELERATION WITH GPU IMPLEMENTATION

This section examines the advantages and the difficulties encountered by deploying GPU technology to perform sparse linear algebra computations. We study first the Sparse Matrix-Vector product (SpMV) which is the base Kernel, common among all the preconditioners discussed before and two preconditioner based on SpMV: ILU(0), BSSOR and Polynomial. We focus on SpMV and preconditioners because they consume most of the compute time in BiCGStab iterations. Thus, the effort is most important in these components while the vector operations are efficiently enough realized by Nvidia's CUBLAS library.

3.1 SpMV

Sparse Matrix Vector product (SpMV) on GPU has been widely studied since introduction of General-Purpose GPU (GPGPU) computation [21-23]. Although SpMV exhibits simple and massive parallelism: in $\mathbf{y} = A.\mathbf{x}$ each y_i computation is independent from each other whereas each y_i is a dot product that also exhibits parallelism; the before-mentioned investigations always show the importance of choosing the right sparse format for the right sparse structure due to the need of massively fine-grain and regular parallelism for efficient GPU use. Thus unstructured sparse formats like CSR or COO do not suit well for GPU computation. Nvidia now provides in its CUDA toolkit SpMV for different sparse formats and this should be considered as a reference.

Although Nvidia provides efficient SpMV for Ellpack and Hybrid format (mix of Ellpack and COO), these SpMV are still designed for generic purpose and do not use linear specific systems that come from the reservoir simulator. To use these specifics, we propose an SpMV that explores the sparse block structure of the matrix A where each non-null element is a small 3×3 or 2×2 block, depending on the underlaying Black-oil simulation. By using blocks, we reduce the indirection cost in SpMV: within the sparse matrix A, a column index represents 2 or 3 contiguous elements of $\mathbf{x}$ so that the effort loading column indices is reduced by a factor of 2 or 3 (*Fig. 1*). Furthermore, the data reuse is better

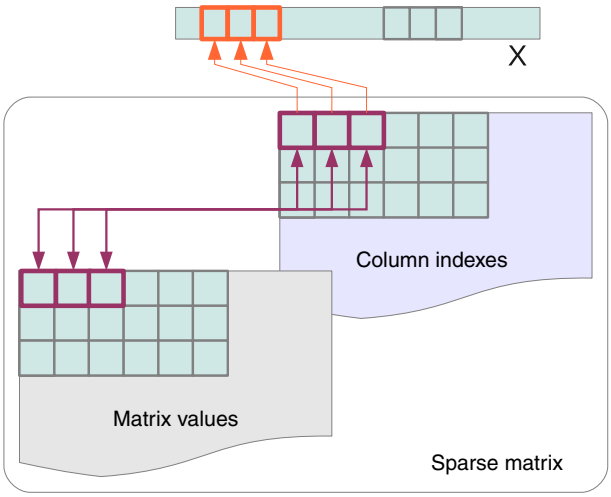


Figure 1

SpMV GPU: using block structure.

in block matrices and the access patterns are more regular. We also use a reordering of A that groups lines with the same width (number of non-null blocks). Thus, instead of computing $\mathbf{y} = A.\mathbf{x}$, our SpMV computes $\mathbf{y} = P.A.P^{-1}.\mathbf{x}$ with the permutation matrix P.

3.2 Preconditioners

We develop an ILU(0), a BSSOR and a polynomial preconditioner for GPU. These three preconditioners exhibit different levels of parallelism: while the polynomial

preconditioner exhibits a high level of parallelism because it uses the same SpMV as in the Krylov iteration, ILU(0) is based on triangular solve with the incomplete factors L and U. The parallelism of the sparse triangular strongly depends on matrix ordering and if we consider "natural" ordering coming from the reservoir simulator (i,j,k grid traversal), the degree of parallelism in the triangular solve is at most the size of the largest sparse row. Such parallelism is insufficient for GPU use.

To go beyond this limitation, a well-known solution is to reorder the matrix according to the node coloring of the associated adjacency graph. Graph coloring consists of assigning a color to each node such that no adjacent node has the same color while trying to minimize the number of colors, *i.e.* to maximize the number of nodes with the same color. A simple greedy algorithm that tries to minimize the number of colors locally does not necessarily minimize the number of colors globally but provides sufficient results in our context because this algorithm:

- gives two equally sized sets with the large majority of nodes;
- gives additional sets with the remaining nodes while the total number of sets is less than or equal to the graph depth plus one.

However, a draw back is that the number of nodes in additional sets can be very small compared to the first two sets. Also, this greedy algorithm finds the classical red black coloring with graphs coming from structured 2D or 3D simulations.

Once the graph is colored, we reorder the matrix by grouping equations (nodes) of the same color. Figure 2 shows the matrix profile for this reordering: the matrix has c diagonal blocks (D_i) where c is the number of colors and other sparse coefficients lie in upper U_i and lower L_i blocks.

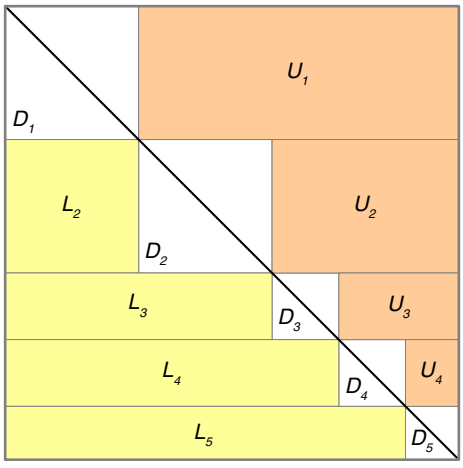


Figure 2

Matrix reodering for GPU triangular solve.

With this structure, the triangular solution process $L.\mathbf{x} = \mathbf{y}$ or $D.\mathbf{x} = \mathbf{y}$ is decomposed into series of diagonal solver steps where the right hand side is built using the solutions x_i from the previous steps:

$$
\begin{aligned}
D_1.x_1 &= y_1 \\
D_2.x_2 &= y_2 - L_2.x_1 \\
D_3.x_3 &= y_3 - L_3.\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\
&\vdots \\
D_c.x_c &= y_c - L_c.\begin{bmatrix} x_1 \\ \dots \\ x_{c-1} \end{bmatrix}
\end{aligned}
$$

Each solver step is fully parallel because every vector component x_i can be computed independently from another. In practice, we use an SpMV for $z_i = L_i.\begin{bmatrix} x_1 \\ \dots \\ x_{i-1} \end{bmatrix}$ and a diagonal solve for $D_i.x_i = y_i - z_i$. Then the GPU computation of each x_i is efficient if each D_i has a large number of rows.

The GPU version of Algebraic MultiGrid (AMG) preconditioner is under study. Different works [24] and implementations of this preconditioner are already available in different libraries like PETSc [25], NVAMG (GPU implementation) [26] and Trilions [27] (many-core implementation). The integration of these libraries in our simulator is under study, in order to compare their performance with that of the MCGSolver.

4 NUMERICAL EXPERIMENTATION

High-performance computing has always been used in the petroleum industry for the simulation at fine scale of large full-field reservoirs. The used parallel reservoir simulator [28] in this paper is a thermal multi-purpose simulator permitting steam injection. It is the new generation IFP Energies nouvelles research reservoir simulator, based on Arcane [29]. In this paper, the strong and the weak points of the before-mentioned preconditioners are reviewed using this simulator.

4.1 Platform Description

The used platform – ener110[(4)] – is composed of 378 nodes. Each node contains 2 Sandy Bridge octo-cores

E7-2670 CPUs with a tact rate of 2.6 GHz. The 14 nodes of ener110 have two K20 GPU cards based on the Kepler architecture [30]. The nodes are interconnected with QDR infiniband links (40 Gbit/s) and offer 110 Teraflops.

4.2 Study Case Description

The results presented in this paper were obtained on three different variants of the Spe10 model, on the tenth SPE comparative solution project model [31] and on a real thermal test case. Spe10 is a incompressible water-oil, black-oil, two-phase flow problem. It is built on a Cartesian regular geometry with no top structure or faults[5] and simulates a 3D waterflood in a geostatistical model. In this model one water injection well at the center of the reservoir and four production wells at the four corners of reservoir are defined and specified with bottom hole pressure.

- case 1: the original Spe10 model contains more than one million active cells (1 122 000 cells; $61 \times 221 \times 86$), with heterogenous porosity and permeability simulating only 30 days;
- case 2: the original Spe10 model with a homogeneous porosity of 0.2 and a permeability of 100 simulating only 30 days;
- case 3: fine-scale geological model of Spe10 with 17 952 000 cells ($241 \times 881 \times 86$) that describes a homogeneous porosity of 0.2 and a permeability of 100 simulating 2 000 days;
- case 4: a thermal model which contains 26 700 cells ($6 \times 50 \times 89$), one producer well and one steam injector well (the steam injector is positioned above the producer), with heterogenous porosity and permeability simulating 800 days.

4.3 Solver Packages Overview

There exist various software packages for parallel computers which offer different preconditioners. This section presents a brief description of some of these libraries used in this paper.

4.3.1 IFPSolvers-MCGSolver

IFPSolvers (IFPS) and MCGSolver (MCGS) are software packages developed by IFPEN to provide linear solver algorithms for its industrial simulators in reservoir simulation [12], basin simulation [32], or in engine combustion simulation [33]. The provided algorithms aim for efficiency on IFPEN customer hardware configurations like Windows 64 platform, Linux cluster with multi-core nodes partially accelerated with GPGPU. These packages provide some Krylov solvers (BiCG-Stab, GMRES, CG), some parallel preconditioners like ILU(0), CPR-AMG, BSSOR, PolyN (Neumann polynomial preconditioner), some specific domain partitioner and matrix graph renumbering algorithms. The packages are interfaced with external libraries like Hypre [8] or SAMG [34].

4.3.2 PMTL4

The Parallel Matrix Template Library v4 (PMTL4) [10, 35] provides linear algebra operations on distributed data as a C^{++} template library. Available data types are distributed vector and sparse and dense matrix types as well as abstractions to conveniently handle distribution and migration. On top of the usual matrix and vector operations, PMTL4 contains a comprehensive set of iterative Krylov subspace solvers including the beforementioned CGS, GMRES, and BiCGStab. Due to the generic design of the library, all solvers could be directly used from the sequential MTL4 version without re-implementation. Parallelism is introduced here just by re-compilation: if the vectors and matrices have distributed types then the overloading mechanism selects the according parallel operations. The library establishes its own domain-specific language embedded in C^{++} in order to provide an intuitive, textbook-like interface. However, this user-friendliness does not harm the performance. To the contrary, advanced meta-programming techniques not only avoid run-time overhead but even enable new forms of performance tuning [36]. The future development will include more specialized solutions for applications like the one in this paper. At the same time, those components will be designed in a flexible and combinable manner for reusing them productively in further applications.

The current ILU implementation is a block version of ILU(0), *i.e.* the preconditioner is independently applied on the local unknowns of each subdomain ignoring all entries on interior boundaries. PMTL4 will also provide an interface to Euclid in the near future – the hooks within Euclid are already prepared by David Hysom.

4.3.3 PETSc

The Portable, Extensible Toolkit for Scientific computation (PETSc) is a parallel open-source library (suite of data structures and routines) which permits large-scale application codes on parallel (and serial) architectures to solve their PDE systems. PETSc uses the MPI

[5] The reason for using a simple geometry is to provide maximum flexibility in the selection of upscaled grids which is exploited in this paper.

standard for all message-passing communication. PETSc has an efficient implementation of Block Jacobi with ILU (0,1 or 2) subdomain solves. In this paper, we use PETSc version 3.1.

4.3.4 HYPRE

HYPRE is a library for solving large, sparse linear systems of equations on massively parallel computers. HYPRE provides a high-performance parallel Algebraic MultiGrid preconditioner: Boomer AMG, for both structured and unstructured problems. It uses MPI for all message-passing communication. In this paper, we use version 2.0.

4.4 Detailed Results

In this section, we analyze the performance of different preconditioners on the recent architectures: homogenous many-core distributed architectures and heterogenous hybrid ones. We compare our solutions proposed in IFPSolver and MCGSolver with PMTL4 and the well-known and widely used library PETSc. The performed simulations, using all preconditioners of the presented libraries, are performed without any accuracy loss. Regardless of the chosen preconditioner, the simulation performs for each test case the same number of time steps and the same cumulative number of Newton steps. The tables present for an entire simulation the total cumulative number of solver iterations (*Tab. 1-4*) and total simulation time (*Tab. 5*) or total solver time (*Tab. 6-8*) for each used cores number.

4.4.1 Distributed Architecture

In the first step, we analyze the multi-core architecture with large numbers of cores. We choose a large but homogeneous model: case 3. The experimental results[6] are shown in Tables 1 and 5. It can be seen that CPR-AMG offers a robust solution obtained in less time, using less cores. Figure 3 demonstrates that for this homogenous model, Block-Jacobi technique, used in PETSc and PMTL4, offers better scalability, since the links between domains are not crucial and the boundaries communication are avoided. This characteristic can be also seen in Tables 2 and 6.

For ill conditioned matrices – which is the case 1 and the case 4 situation – the proposed ILU(0) by IFPSolver and CPR-AMG offer a better solutions since the inner boundaries are not ignored. The experimentation results shown in Tables 3, 7, 9, 10 confirm that. We demonstrate in Figure 3 and Figure 4 that CPR-AMG offers a fast solution but it is not scalable.

[6] PETSc results are not coherent for 1 024 cores.

TABLE 1
Case 3: cumulative number of solver iteration during whole simulation using classical parallelization (MPI)

Cores number	128	256	512	1 024
IFPS CPR-AMG	3 925	3 936	4 673	3 297
IFPS ILU(0)	45 7936	469 786	547 684	456 077
PMTL4 ILU(0)	601 321	499 305	539 657	656 813
PETSC ILU(0)	635 181	645 551	609 245	***
PETSC ILU(1)	387 824	414 799	402 307	***
PETSC ILU(2)	333 447	338 046	338 905	***

TABLE 2
Case 2: cumulative number of solver iteration during whole simulation using classical parallelization (MPI)

Cores number	16	32	64	128
IFPS CPR-AMG	121	122	123	126
IFPS ILU(0)	6 236	6 995	7 029	7 747
PMTL4 ILU(0)	4 399	4 384	4 612	4 594
PETSc ILU(0)	4 653	4 764	5 023	5 047
PETSc ILU(1)	2 435	2 526	2 820	2 931
PETSc ILU(2)	2 134	2 319	2 629	2 725

TABLE 3
Case 1: cumulative number of solver iteration during whole simulation using classical parallelization (MPI)

Cores number	16	32	64	128
IFPS CPR-AMG	1 298	1 259	1 181	1 428
IFPS ILU(0)	7 309	7 789	7 843	7 981
PMTL4 ILU(0)	7 540	8 307	8 282	8 632
PETSc ILU(0)	8 183	8 949	8 887	9 325
PETSc ILU(1)	5 433	6 589	6 372	7 112
PETSc ILU(2)	5 055	6 176	6 205	6 980

4.4.2 Heterogenous Architecture

In this section, we analyze the performance gain obtained thanks to GPU accelerators. Since GPU

TABLE 4
Case 1: cumulative number of solver iteration during whole simulation using hybrid parallelization

Cores number	1c	8c(mpi)	1c/1GPU
IFPS CPR-AMG	1 685	1 235	***
IFPS ILU(0)	7 749	7 192	***
PMTL4 ILU(0)	6 892	7 463	***
PETSc ILU(0)	7 749	8 088	***
PETSc ILU(1)	4 520	2 410	***
PETSc ILU(2)	3 809	4 641	***
MCGS color ILU(0)	12 393	***	12 662
MCGS BSSOR	22 362	***	21 898
MCGS PolyN	12 301	12 160	12 309

TABLE 5
Case 3: total simulation time (s) using classical parallelization (MPI)

Cores number	128	256	512	1 024
IFPS CPR-AMG	4 346	2 506	2 360	2 477
IFPS ILU(0)	31 354	17 637	12 124	7 025
PMTL4 ILU(0)	38 346	15 946	9 373	6 048
PETSC ILU(0)	44 584	25 134	1 0749	***
PETSC ILU(1)	32 892	18 863	1 5447	***
PETSC ILU(2)	37 081	19 528	9 811	***

TABLE 6
Case 2: total solver time (s) using classical parallelization (MPI)

Cores number	16	32	64	128
IFPS CPR-AMG	84.6	38.2	21.6	21.3
IFPS ILU(0)	177.8	100.0	51.4	29.4
PMTL4 ILU(0)	167.8	86.2	48.6	26.6
PETSc ILU(0)	153.0	105.9	53.1	29.3
PETSc ILU(1)	103.2	59.8	38.4	23.9
PETSc ILU(2)	126.5	68.3	47.8	27.2

TABLE 7
Case 1: total solver time (s) using classical parallelization (MPI)

Cores number	16	32	64	128
IFPS CPR-AMG	151.6	79.7	41.9	49.4
IFPS ILU(0)	194.0	108.5	57.5	29.8
PMTL4 ILU(0)	223.4	117.8	63.5	36.3
PETSc ILU(0)	269.5	206.2	86.0	51.9
PETSc ILU(1)	233.4	138.0	87.4	51.4
PETSc ILU(2)	267.9	169.9	82.3	71.8

TABLE 8
Case 1: performance results (total solver time) of hybrid parallelization (PolyN 8 cores use threads instead of mpi)

Cores number	1c	8c(mpi)	1c/1GPU
IFPS CPR-AMG	1 038	280	***
IFPS ILU(0)	2157	374	***
PMTL4 ILU(0)	1 796	294	***
PETSc ILU(0)	2 159	500	***
PETSc ILU(1)	1 586	205	***
PETSc ILU(2)	1 793	493	***
MCGS color ILU (0)	2 144	***	297
MCGS BSSOR	8 495	***	1 129
MCGS PolyN	2 786	1 035	367

TABLE 9
Case 4: cumulative number of solver iteration during whole simulation using classical parallelization (MPI)

Cores number	1	4	8	16
IFPS CPR-AMG	20 898	22 759	24 413	25 052
IFPS ILU(0)	278 413	296 243	308 324	310 157
PMTL4 ILU(0)	155 643	205 361	217 348	239 380
PETSc ILU(0)	204 686	248 023	263 654	285 464
PETSc ILU(1)	133 978	229 705	258 613	283 661
PETSc ILU(2)	114 628	217 700	257 875	282 339

TABLE 10
Case 4: total solver time (s) using classical parallelization (MPI)

Cores number	1	4	8	16
IFPS CPR-AMG	858.4	274.1	184.6	152.2
IFPS ILU(0)	1 489.3	363.3	225.7	129.1
PMTL4 ILU(0)	4 842.3	1 444.9	791.3	480.0
PETSc ILU(0)	3 348	945.7	574.9	531.1
PETSc ILU(1)	4 139.4	1 333.6	771.8	663.2
PETSc ILU(2)	6 053.8	2 028	1 171.9	755.1

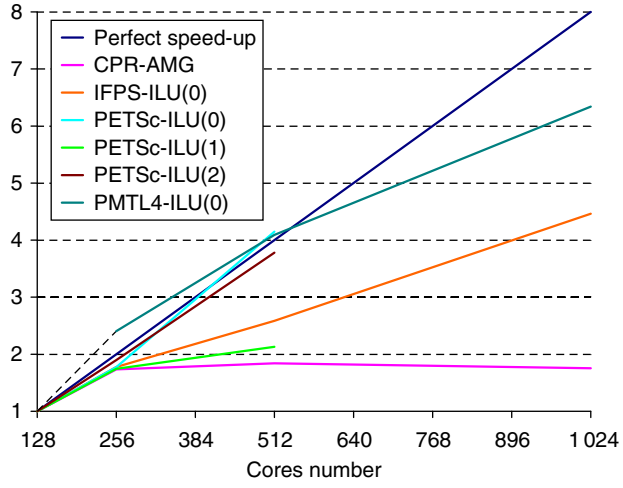


Figure 3
Case 3: homogeneous spe10, speed-up.

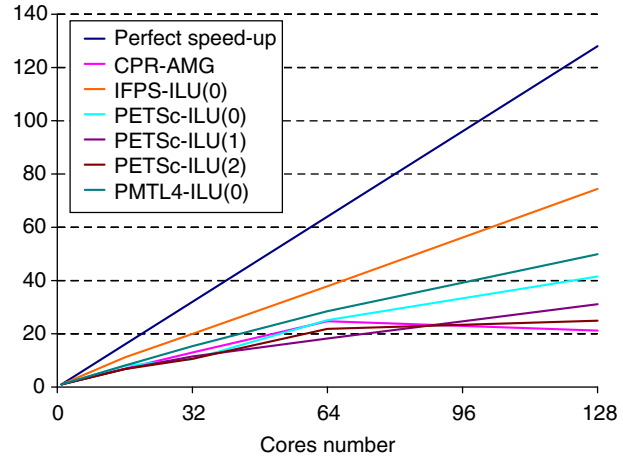


Figure 4
Case 1: original spe10, speed-up.

preconditioner are based on SpMV, we first present synthetic SpMV performaces on CPU and GPU. These days, the cost of one CPU and one GPU is quite similar. Therefore the following question makes sense: "is it interesting, in term of performance, to have a GPU accelerator instead of an additional CPU?"

SpMV performances

In Figure 5, we compare the performance of SpMV developed at IFPEN and Nvidia's cusparse SpMV for CSR, BSR, Ellpack and Hybrid formats running on Fermi C0270 and Kepler K20 boards against a CPU SpMV running on one Xeon Sandy-Bridge processor using between one and eight cores. The CPU SpMV implementations are block-CSR algorithms developed for the MCGSolver and the block-CSR from the Intel MKL library. We use matrices ranging from 2.4×10^4 to 2×10^6 equations extracted from IFPEN reservoir simulator. These results show the situations:

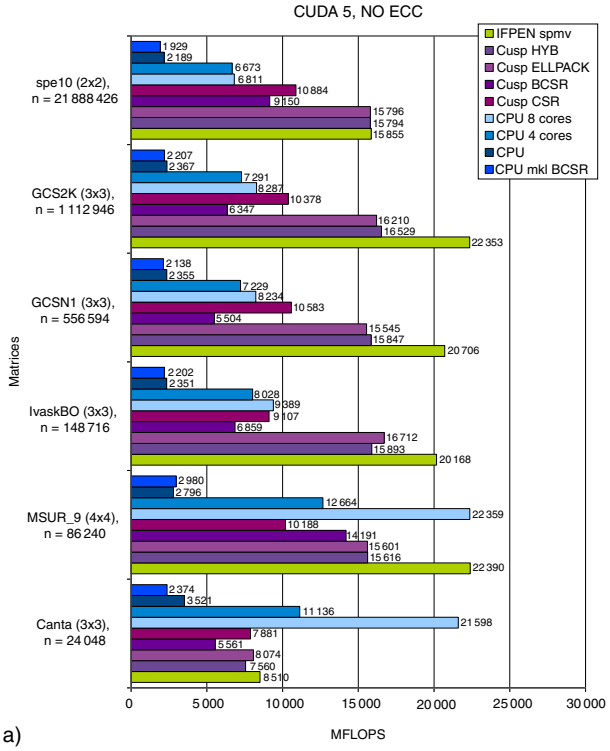
- where the matrix is small enough (Canta, 2.4×10^4 eq.) to fit into L3 or L2 processor caches. Then, the CPU performance is with 4 and 8 threads and is better than the best GPU version. Also, multi-core CPU SpMV offers good scaling with speed-up of 3.2 and 6.1;
- where the matrices are too large to fit into CPU caches. Then all GPU SpMV – except CSR – offer a significant performance boost over CPUs, even with 4 and 8 cores. Acceleration compared to 8 cores varies from 1.9 to 3 for the K20 GPU.

Also block CSR from either MKL or Cusparse does not improve performance against CSR except for Cusparse BSR when block size is 2×2 or 4×4 on K20 but never competes with Cusparse Ellpack or Hybrid format except for MSUR_9 matrix.

Reservoir simulator results

We present simulation results for Case 1 and Case 4 which show two different behaviours for GPU solver acceleration.

For Case 1, performance results presented in Tables 8 and 4 highlight the obtained benefit by using the hybrid CPU + GPU implementation of ILU(0) compared to ILU(0) using 8 cores of a CPU. This table shows that for the ill conditioned Spe10 system, the hybrid CPU + GPU offers a better performance, even if the number of solver iterations is more important, due to the graph-coloring, than the traditional version implemented in IFPSolver, PETSc and PMTL4. Note that CPU + GPU solver times include all data setup and transfer to/from GPU. For polynomial preconditioner, we have



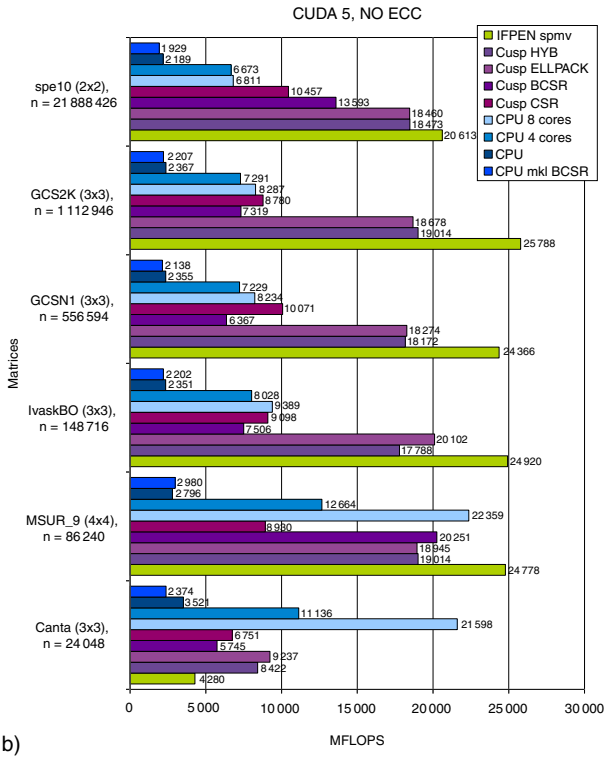


Figure 5

SpMV on a) C2070 and b) K20 compared to Intel Sandy-Bridge XEON E5-2680@2.7 Ghz.

TABLE 11

Case 4: performance results (total solver time) of hybrid parallelization (PolyN 8 cores use threads instead of mpi)

Cores number	1c	8c(mpi)	1c/1GPU
IFPS CPR-AMG	858	185	***
IFPS ILU(0)	1 489	226	***
PMTL4 ILU(0)	4 842	791	***
PETSc ILU(0)	3 348	575	***
PETSc ILU(1)	4 139	772	***
PETSc ILU(2)	6 054	1 172	***
MCGS color ILU(0)	3 376	***	738
MCGS BSSOR	6 685	***	1 370
MCGS PolyN	4 207	878	788

TABLE 12

Case 4: cumulative number of solver iteration during whole simulation using hybrid parallelization

Cores number	1c	8c(mpi)	1c/1GPU
IFPS CPR-AMG	20 898	24 413	***
IFPS ILU(0)	278 413	308 324	***
PMTL4 ILU(0)	155 643	217 348	***
PETSc ILU(0)	204 686	263 654	***
PETSc ILU(1)	133 978	258 613	***
PETSc ILU(2)	114 628	257 875	***
MCGS color ILU(0)	362 468	***	361 221
MCGS BSSOR	383 595	***	381 337
MCGS PolyN	387 253	388 092	384 983

a convergence rate equivalent to GPU color ILU(0) but with a higher algorithmic iteration cost: the polynomial degree is 2 which means 2 SpMV per preconditioner call. This cost is at least twice the cost of an ILU(0) solve; anyway GPU polynomial preconditioner is competitive against all one core CPU preconditioners and close to some eight cores CPU preconditioners. On the other side GPU BSSOR has higher iteration cost and also slower convergence rate thus provides small improvement against one core CPU preconditioner and no improvement against eight cores one. It is also interesting to note that CPU + GPU solver/preconditioner have an acceleration factor close to 7.5 against their CPU counterpart.

For Case 4, as shown in Table 7 and 8, CPU + GPU solvers offer an acceleration between 4.6 and 5.3 against their one core CPU counterpart but do not compete with best eight cores MPI solvers (IFPS CPR-AMG and IFPS ILU(0)). This is due to the relatively small case size ($\approx 10^4$ solver unknows) where the parallel CPU solvers gain efficiency as data set fits in processor caches but also to a worse numerical behaviour of GPU preconditioners. As an exemple for IFPS ILU(0)/ MCGS color ILU (0), we see a 1.6 increase factor for number of iteration in case1 while the same factor is 2.3 in Case 4. For GPU polynomial and BSSOR preconditioner the observed behaviour is close to the one for Case 1: GPU polynomial is close to color ILU(0) and GPU BSSOR provides no perfomance improvement.

CONCLUSION AND FUTURE WORKS

The purpose of our article was to evaluate the performance of the linear solvers used in large parallel reservoir simulators on new hardware configurations. We have focused our study on two kinds of configurations: large memory distributed architectures with a large number of cores and heterogeneous nodes with multi-core CPUs accelerated with GPGPU cards. We evaluated the scalability of a BiCGStab solver with different preconditioners of the IFPS, PETSc and PMTL4 solver packages, with some runs on 128 up to 1 024 cores, on a 17 950 000 grid block study test case. For this kind of architecture, CPR-AMG is the most efficient preconditioner mainly due to its extensibility qualities even if its construction suffers of strong scalability problems. The preconditioners of the ILU family, well parallelized by means of renumbering techniques, have a good scalability behaviour although they numerically suffer of lack of extensibility.

With heterogeneous multi-core nodes accelerated with GPGPUs, we have tested new solvers and preconditioners, which are based on SpMV, taking advantage of GPGPU accelerators. These algorithms turn to be competitive with respect to the best CPU ones using one core and in some cases against eight cores even with worse convergence rates. Also with good performance of GPU SpMV, we expect that SPAI preconditioners [38] will provide better convergence rates with the same level of performance for solver iteration. Our results demonstrate that using GPGPUs becomes now a real alternative to multi-core nodes for solving ill-conditioned linear systems with Krylov solvers in reservoir simulation. Once we proved the efficiency of the GPU implementation, the next step will be to compare our solution with other libraries. Different implementations are available in different libraries like PETSc [25] and Trilions [27] (many-core implementation). The integration of these libraries in our simulator is under study.

To handle Petascale and later Exascale architectures, we need to improve the scalability of the CPR-AMG preconditioner beyond 1 024 cores up to at least 10 000 cores in order to have both weak and strong scalability. Within nodes, the use of accelerators in heterogeneous architecture becomes a heavy trend, for energy and economical raisons. Right now, the preconditioners designed for GPGPU become competitive with respect to the best CPU one. Nevertheless, to really take advantage of the computation power offered by accelerators, we need to design new algorithms which are numerically more efficient to reduce the number of iterations, as for preconditioners like CPR-AMG. As this last one is not adapted for fine grain size parallelization, we plan to consider other new multi-level methods, based on domain decomposition algorithms which are promising as they present both numerical robustness and are potentially more easier to accelerate with GPGPU.

REFERENCES

1 Saad Y. (2000) *Iterative Methods for Sparse Linear Systems*, second edition, SIAM, pp. 144-227.

2 Buck I., Foley T., Horn D., Sugerman J., Fatahalian K., Houston M., Hanrahan P. (2004) *ACM Transactions on graphics* **23**, 777-786.

3 NVIDIA Corp. http://www.nvidia.com/object/cuda_home_new.html.

4 http://www.khronos.org/opencl/.

5 Patterson D.A., Hennessy J.L. (2011) The Hardware/Software Interface, in *Computer Organization and Design*, E.D. Morgan Kaufmann Publishers Inc., Elsevier, fourth edition, Chap. 7 and Appendix A.

6 Culler D.E., Pal Singh J., Gupta A. (1997) A Hardware/Software Approach, in *Parallel Computer Architecture*, E.D. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 44-47.

7 Lacroix S., Vassilevski Yu, Wheeler J., Wheeler M.F. (2003) Iterative solution methods for modeling multiphase flow in porous media fully implicitly, *SIAM Journal* **25**, 3, 905-926.

8 Chow E., Cleary A.J., Falgout R.D. (1998) Design of the hypre Preconditioner Library, in proceding of the *SIAM Workshop on Object Oriented Methods for Inter-operable Scientific and Engineering Computing*.

9 Balay S., Brown J., Buschelman K., Gropp W.D., Kaushik D., Knepley M.G., McInnes L.C., Smith B.F., Zhang H. (2012) *PETSc Web page*, http://www.mcs.anl.gov/petsc.

10 Gottschling P. *et al.* (2011) PMTL4 web page, http://www.simunova.com.

11 Gratien J.-M., Guignon T., Hacene M. (2011) Solveurs linéaires sur GPU pour la simulation d'écoulement en milieux poreux, in *SMAI Mini symposia, Nouvelles tendances en méthodes numériques et calcul scientifique pour la simulation des écoulements polyphasiques*, Guiel, France, 23-27 May.

12 Gratien J.-M., Guignon T., Magras J.-F., Quandalle P., Ricois O. (2006) How to Improve the Scalability of an Industrial Parallel Reservoir Simulator, *Parallel and Distributed Computing and Systems*, *PDCS*, 13-15 Nov., Dalles, Tx, 513-098

13 Karp R.M. (1972) Reducibility Among Combinatorial Problems, in *Complexity of Computer Computations*, R.E. Miller, J.W. Thatcher (eds), Plenum, New York, pp. 85-103.

14 Karypis G., Kumar V. (1995) *METIS - Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0*, Technical Report.

15 Gratien J.-M., Guignon T., Magras J.-F., Quandalle P., Ricois O. (2007) Scalability and load balancing problems in parallel reservoir simulation, in Proceeding of *SPE-Reservoir Simulation Symposium*, Houston, 26-28 Fev.

16 Gratien J.-M., Magras J.-F., Quandalle P., Ricois O. (2004) Introducing a new generation of reservoir simulation software, in Proceedings of *ECMOR, European Conference on the Mathematics of Oil Recovery*, Cannes, 30 Aug.-2 Sept.

17 Trottenberg U., Oosterlee C., Schüller A. (2001) *Multigrid Methods*, Academic Press.

18 Briggs W.L., Henson V.E., McCormick S.F. (2000) *A Multigrid Tutorial*, second edition, SIAM, pp. 137-159.

19 Brezina M., Cleary A.J., Falgout R.D., Henson V.E., Jones J.E., Manteuffel T.A., McCormick S.F., Ruge J.W. (2000) Algebraic Multigrid Based on Element Interpolation (AMGe), *SIAM Journal on Scientific Computing* **22**, 5, 1570-1592.

20 De Sterck H., Yang U.M., Heys J. (2006) Reducing complexity parallel in algebraic multigrid preconditioners, in *Proceeding of SIMAX* **27**, 1019-1039.

21 Bolz J., Farmer I., Grinspun E., Schröder P. (2003) Sparse matrix solvers on the GPU: conjugate gradients and multigrid, *ACM Trans. Graph.* **22**, 3, 917-924.

22 Buatois L., Caumon G., Lévy B. (2007) Concurrent Number Cruncher: An Efficient Sparse Linear Solver on the GPU, in *High Performance Computing and Communications*, Houstan, 26-28 Sept, *Lecture Notes in Computer Science* **4782**, 358-371.

23 Bell N., Garland M. (2008) *Efficient Sparse Matrix-Vector Multiplication on CUDA*, NVIDIA Technical Report.

24 Emans M., Liebmann M., Basara B. (2012) Steps towards GPU Accelerated Aggregation AMG, *11th International Symposium on Parallel and Distribuled Computing – ISPDC 2012*, Munich, 25-29 June, pp. 79-86.

25 Minden V., Smith B.F., Knepley M.G. (2010) Preliminary implementation of PETSc using GPUs, Proceedings of the *International Workshop of GPU Solutions to Multiscale Problems in Science and Engineering*, *GMP-SMP 2010*, Harbin, 26-28 July.

26 NVAMG Reference Manual (2013).

27 Heroux M.A., Bartlett R.A., Howle V.E., Hoekstra R.J., Hu J.J., Kolda T.G., Lehoucq R.B., Long K.R., Pawlowski R.P., Phipps E.T., Salinger A.G., Thornquist H.K., Tuminaro R.S., Willenbring J.M., Williams A., Stanley K.S. (2005) An overview of the Trilinos project, *ACM Trans. Math. Softw.* **31**, 397-423.

28 Ricois O. (2011) *Vision Général du simulateur ARCEOR*, IFPEN Technical Report number 62096.

29 Grospellier G., Lelandais B. (2009) The Arcane development framework, Proceedings of the *8th workshop on Parallel/High-Performance Object-Oriented Scientific Computing*, *POOSC '09*, 978-1-60558-547-5, Genova, Italy, pp. 4:1-4:11.

30 NVIDIA's Next Generation CUDATM Compute Architecture: Kepler TM GK110, *The Fastest, Most Efficient HPC Architecture Ever Built*, White paper http://www.nvidia.fr/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf.

31 Christie M.A., Blunt M.J. (2001) Tenth SPE Comparative Solution Project: A Comparison of Upscaling Techniques, in *SPE Reservoir Simulation Symposium*, Houston, Tx, 11-14 Fev.

32 Willien F., Chetvchenko I., Masson R., Quandalle P., Agelas L., Requena S. (2009) AMG preconditioning for sedimentary basin simulations in Temis calculator, *Marine and Petroleum Geology Journal* **26**, 4, 519-524.

33 Bohbot J., Gillet N., Benkenida A. (2009) IFP-C3D: An unstructured parallel solver for reactive compressible gas flow with spray, *Oil Gas Sci. Technol.* **64**, 309-336.

34 Stüben K. (2001) *Algebraic Multigrid: An Introduction for Positive Definite Problems with Applications*, Academic Press, pp. 413-532.

35 Gottschling P., Hoefler T. (2012) Productive Parallel Linear Algebra Programming with Unstructured Topology Adaption, *International Symposium on Cluster, Cloud and Grid Computing 2012*, Ottawa, Canada, May, ACM/IEEE.

36 Gottschling P., Steinhardt C. (2010) Meta-Tuning in MTL4, *ICNAAM 2010: International Conference of Numerical Analysis and Applied Mathematics*, **1281**, 778-782, Sept., 10.1061.3498599, Proc. *ICNAAM '10 – Symposium Automated Computing*.

37 Hysom D., Pothen A. (2001) A scalable parallel algorithm for incomplete factor preconditioning, *SIAM Journal on Scientific Computing* **22**, 6, 2194-2215.

38 Barnard S.T., Bernardo L.M., Simon H.D. (1999) An MPI implementation of the SPAI preconditioner on the t3E, *Int. J. High Perf. Comput. Appl. J.* **13**, 107-128.

Manuscript accepted in July 2013
Published online in January 2014

Copyright © 2014 IFP Energies nouvelles

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than IFP Energies nouvelles must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee: request permission from Information Mission, IFP Energies nouvelles, fax. + 33 1 47 52 70 96, or revueogst@ifpen.fr.