C. R. Mecanique 339 (2011) 96–103

Contents lists available at ScienceDirect

Comptes Rendus Mecanique

www.sciencedirect.com

High Performance Computing / Le Calcul Intensif

Parallel hierarchical hybrid linear solvers for emerging computing platforms

Un solveur hybride linéaire parallèle hiérarchique pour les plateformes de calcul émergentes

Emmanuel Agullo, Luc Giraud, Abdou Guermouche, Jean Roman [*]

HiePACS Project – INRIA Bordeaux Sud-Ouest, Joint INRIA–CERFACS Lab. on High Performance Computing, PRES de Bordeaux, 351, Cours de la Libération, 33405 Talence cedex, France

A R T I C L E I N F O

Article history:
Available online 3 January 2011

Keywords:
Computer science
Hierarchical platform
Parallel architecture

Mots-clés :
Informatique, algorithmique
Plateforme hiérarchique
Architecture parallèle

A B S T R A C T

The design of the extreme-scale platforms that are expected to become available in the forthcoming decade will represent a convergence of technological trends and the boundary conditions imposed by over half a century of algorithm and application software development. These platforms will be hierarchical because they provide coarse grain parallelism between nodes and fine grain parallelism within each node. They are also expected to be very heterogeneous since multi-core chips and accelerators have completely different architectures and potentials. It is clear that such a degree of complexity will embody radical changes that will render obsolete the current software infrastructure for large-scale scientific applications. In this paper, we illustrate a hierarchical algorithmic approach for the implementation of an efficient parallel sparse linear solver that combines direct and iterative methods. Such a hybrid approach exploits the advantages of both numerical techniques and enables the use of several levels and grains of parallelism. This combination express different levels of parallelism and permits an optimal trade-off between numerical and parallel efficiency. Consequently, such a numerical technique appears as a promising candidate for intensive simulations on future many-core parallel platforms.

R É S U M É

La conception des plateformes d’échelle extrême qui devraient être disponibles dans la décade à venir représenteront la convergence de tendances technologiques et définiront le cadre imposé au développement des futurs algorithmes et applications scientifiques. Ces plateformes seront hiérarchiques puisqu’elles disposeront d’un parallélisme à grain moyen entre les nœuds de calcul et d’un autre à grain fin à l’intérieur des unités de calcul. Elles seront également probablement très hétérogènes puisque leurs composantes multi-cœurs et cartes accélératrices auront des architectures et des caractéristiques très différentes. Il est clair qu’un tel degré de complexité nécessitera des changements radicaux qui rendront obsolètes les infracstrcutures logicielles actuelles pour le calcul des grandes applications. Dans cet article, nous illustrons une approche algorihmique hiérarchique pour la mise en œuvre d’un solveur linéaire parallèle qui combine des méthodes itératives et directes. Une telle approche hybride exploite les avantages des deux méthodes et permet l’utilisation

[*] Corresponding author.
E-mail address: jean.roman@inria.fr (J. Roman).

1631-0721/$ – see front matter © 2010 Académie des sciences. Published by Elsevier Masson SAS. All rights reserved.
doi:10.1016/j.crme.2010.11.005

de plusieurs niveaux de parallélisme de grains différents. Cette combinaison autorise un compromis optimal entre la performance parallèle et numérique de l'algorithme. En conséquence, une telle approche numérique apparaît comme une candidate prometteuse pour le calcul intensif sur les futures machines multi-cœurs.

1. The parallel computing platforms: landscape and trends

The design of the extreme-scale platforms that are expected to become available in the forthcoming decade will represent a convergence of technological trends and the boundary conditions imposed by over half a century of algorithm and application software development. Although the precise details of these new designs are not yet known, the main trend is that future platforms are likely to be composed of a large number of accelerator-based multi-core nodes. These platforms are hierarchical because they provide coarse grain parallelism between nodes and fine grain parallelism within each node. They are also very heterogeneous since multi-core chips and accelerators have completely different architectures and potentials. It is clear that such a degree of complexity will embody radical changes as compared to the today's systems and that these changes will render obsolete the current software infrastructure for large-scale scientific applications.

Such evolutions have been observed in the past at a lower scale. Dense linear algebra libraries have always been in the vanguard of High Performance Computing (HPC) trends. Therefore, one way to motivate our plan to design new numerical methods for solving large sparse linear systems of equations on large-scale emerging machines is to first point to the example of the evolution of dense linear algebra numerical methods and software in the last decades.

In the early seventies, the need to provide a standard API for applications relying on dense linear algebra kernels led to the release of the LINPACK library. The factorizations were performed as a sequence of column factorizations followed by updates of the trailing submatrix that performed extremely well on the HPC computers at that time that were vector machines. When the complexity of hardware architectures grew in the eighties, with the intensive use of cache memory, the update corresponding to a single column factorization was no longer efficient. Instead, column blocks (so-called panels) were factorized all at once leading to a much more efficient update of the trailing submatrix thanks to a better data locality of the data reused in cache. Furthermore, the model of a monolithic library embedding all dense linear algebra routines became unsustainable. These routines were indeed split into two subsets associated with two different layers of abstraction. The Basic Linear Algebra Subroutines (BLAS) were simple enough routines (such as the BLAS-3 matrix multiplication) so that they could be highly tuned by vendors at a limited cost on each platform they delivered. The second set of routines were higher level routines (such as factorizations) were fully redesigned to rely on the BLAS in order to achieve high performance. In the nineties, the emergence of distributed memory clusters as mainstream platforms led the software community to design new parallel algorithms to cope with the new characteristics of the computers. To achieve an efficient load balancing, the matrix was split in submatrices that were processed by different processors according to a two-dimensional cyclic grid. A new communication layer, the Basic Linear Algebra Communication Subroutines (BLACS), enabled to design a portable library for distributed memory machines, so-called Scalable Lapack (ScaLAPACK).

The new paradigm of platforms composed of a large number of accelerator-based multi-core nodes is probably a hardware revolution of broader impact than all the above evolutions that the HPC software community has to face. Again, the dense linear algebra community pioneered to tackle this challenge. Indeed, new algorithms providing a finer granularity of the tasks, enabling more dynamicity and ensuring further locality. For instance, so-called tile algorithms [1,2] and communication-avoiding algorithms [3] strongly improved the performance of dense factorizations. To ensure performance portability that is required to accommodate with the high pace currently followed by the evolution of parallel architectures, they designed libraries that are further modular. In particular, a new layer, the runtime system, is used to permit to do dynamically what can hardly be done statically anymore [4,5].

In the rest of the paper, we now address the problem of solving a large sparse linear system, which is often one of the most important computational step of large-scale numerical simulations. We propose a method that seems to be a secure path for achieving high performance on the forthcoming peta/exascale computers. Our numerical method is hierarchical (two coarse levels of parallelism, but possibly more at finer grain) and relies on existing building blocks that can take advantage of complex nodes (as in the methodology described above in the case of dense linear algebra).

2. A methodological approach: a case study in sparse linear algebra

2.1. Introduction

Solving large sparse linear systems $Ax = b$, where A is a given matrix, b is a given vector, and x is an unknown vector to be computed, appears often in the inner-most loops of intensive simulation codes. It is consequently the most time-consuming computation in many large-scale computer simulations in science and engineering, such as computational incompressible fluid dynamics, structural analysis, wave propagation, and design of new materials in nanoscience, to name a few. Over the past decade or so, several teams have been developing innovative numerical algorithms to exploit advanced high performance, large-scale parallel computers to solve these equations efficiently. There are two basic approaches for solving linear systems of equations: direct methods and iterative methods. Those two large classes of methods have somehow opposite features with respect to their numerical and parallel implementation efficiencies.

Direct methods are based on the Gaussian elimination that is probably among the oldest method for solving linear systems. Tremendous effort has been devoted to the design of sparse Gaussian eliminations that efficiently exploit the sparsity of the matrices. These methods indeed aim at exhibiting dense submatrices that can then be processed with computational intensive standard dense linear algebra kernels such as BLAS, LAPACK and ScaLAPACK. Sparse direct solvers have been for years the methods of choice for solving linear systems of equations because of their reliable numerical behavior [6]. There are on-going efforts in further improving existing parallel packages. However, it is admitted that such approaches are intrinsically not scalable in terms of computational complexity or memory for large problems such as those arising from the discretization of large three-dimensional partial differential equations (PDEs). Furthermore, the linear systems involved in the numerical simulation of complex phenomena result from some modeling and discretization which contains some uncertainties and approximation errors. Consequently, the highly accurate but costly solution provided by stable Gaussian eliminations might not be mandatory.

Iterative methods, on the other hand, generate sequences of approximations to the solution either through fixed point schemes or via search in Krylov subspaces [7]. The best known representatives of these latter numerical techniques are the conjugate gradient [8] and the GMRES [9] methods. These methods have the advantage that the memory requirements are small. Also, they tend to be easier to parallelize than direct methods. However, the main problem with this class of methods is the rate of convergence, which depends on the properties of the matrix. Very effective techniques have been designed for classes of problems such as the multigrid methods [10], that are well suited for specific problem such as those arising from the discretization of elliptic PDEs. The essence of the multigrid approaches is to hierarchically reduce the size of the problem via coarsening mechanisms and solving recursively a correction equation on a sequence of embedded problems. This efficient numerical mechanism based on coarsening techniques leads to a reduce amount of computation when it goes sequentially down the hierarchy which limits the possible parallelism on large scale platforms. Some attempts have been made to develop additive variants to express reasonable amount of parallelism by enabling independent treatment between the different levels of the hierarchy of problems. However, these schemes have encountered a limited success on a few academic problems and their generalization to general linear systems remains an open question. Another trend to preserve a good trade-off between numerical efficiency and large amount of parallelism consists in the so-called aggressive coarsening that attempts to limit the number of levels in the hierarchy. We refer to [11, Chapter 10] and the references therein for a recent survey. One way to improve the convergence rate of Krylov subspace solver is through preconditioning and fixed point iteration schemes are often used as preconditioner. In many computational science areas, highly accurate solutions are not required as long as the quality of the computed solution can be assessed against measurements or data uncertainties. In such a framework, the iterative schemes play a central role as they might be stopped as soon as an accurate enough solution is found. In our work, we consider stopping criteria based on the backward error analysis [12–14].

Our approach to high-performance, scalable solution of large sparse linear systems in parallel scientific computing is to combine direct and iterative methods. Such a hybrid approach exploits the advantages of both direct and iterative methods. The iterative component allows us to use a small amount of memory and provides a natural way for parallelization. The direct part provides its favorable numerical properties. Furthermore, this combination enables us to exploit naturally several levels of parallelism that logically match the hardware feature of emerging many-core platforms as stated in Section 1. In particular, we can use parallel multi-threaded sparse direct solvers within the many-core nodes of the machine and message passing among the nodes to implement the gluing parallel iterative scheme.

The general underlying ideas are not new. They have been used to design domain decomposition techniques for the numerical solution of PDEs [15–17]. Domain decomposition refers to the splitting of the computational domain into subdomains with or without overlap. The splitting strategies are generally governed by various constraints/objectives but the main one is to enhance parallelism. The numerical properties of the PDEs to be solved are usually extensively exploited at the continuous or discrete levels to design the numerical algorithms. Consequently, the resulting specialized technique will only work for the class of linear systems associated with the targeted PDEs. In our work, we develop domain decomposition techniques for general unstructured linear systems. More precisely, we consider numerical techniques based on a non-overlapping decomposition of the graph associated with the sparse matrices. The vertex separator, constructed using graph partitioning [18,19], defines the interface variables that will be solved iteratively using a Schur complement approach, while the variables associated with the interior sub-graphs will be handled by a sparse direct solver. Although the Schur complement system is usually more tractable than the original problem by an iterative technique, preconditioning treatment is still required. For that purpose, we developed parallel preconditioners and designed a hierarchical parallel implementation. Linear systems with a few tens of millions unknowns have been solved on a few thousand of processors using our designed software prototypes. We designed novel algorithms that greatly improve our solvers scalability, especially in exploiting the emerging advances of petascale many-core computers.

2.2. Parallel hierarchical implementation

In this section, methods based on non-overlapping regions are described. Such domain decomposition algorithms are often referred to as substructuring schemes. This terminology comes from the structural mechanics discipline where non-overlapping ideas were first developed. The structural analysis finite element community has been heavily involved in the design and development of these techniques. Not only is their definition fairly natural in a finite element framework but their implementation can preserve data structures and concepts already present in large engineering software packages.

Let us now further describe this technique and let $\mathcal{A}x = b$ be the linear problem. For the sake of simplicity, we assume that $\mathcal{A}$ is symmetric in pattern and we denote $G = \{V, E\}$ the adjacency graph associated with $\mathcal{A}$. In this graph, each vertex is associated with a row or column of the matrix $\mathcal{A}$ and it exists an edge between the vertices i and j if the entry $a_{i,j}$ is non-zero.

We assume that the graph G is partitioned into N non-overlapping sub-graphs $G_1, \ldots, G_N$ with boundaries $\Gamma_1, \ldots, \Gamma_N$. The governing idea behind substructuring or Schur complement methods is to split the unknowns in two subsets. This induces the following block reordered linear system:

$$\begin{pmatrix} \mathcal{A}_{II} & \mathcal{A}_{I\Gamma} \\ \mathcal{A}_{\Gamma I} & \mathcal{A}_{\Gamma\Gamma} \end{pmatrix} \begin{pmatrix} x_I \\ x_\Gamma \end{pmatrix} = \begin{pmatrix} b_I \\ b_\Gamma \end{pmatrix} \tag{1}$$

where x_Γ contains all unknowns associated with sub-graph interfaces and x_I contains the remaining unknowns associated with sub-graph interiors. Because the interior points are only connected to either interior points in the same sub-graph or with points on the boundary of the sub-graphs, the matrix $\mathcal{A}_{II}$ has a block diagonal structure, where each diagonal block corresponds to one sub-graph. Eliminating x_I from the second block row of Eq. (1) leads to the reduced system

$$\mathcal{S}x_\Gamma = f \tag{2}$$

where

$$\mathcal{S} = \mathcal{A}_{\Gamma\Gamma} - \mathcal{A}_{\Gamma I}\mathcal{A}_{II}^{-1}\mathcal{A}_{I\Gamma} \quad \text{and} \quad f = b_\Gamma - \mathcal{A}_{\Gamma I}\mathcal{A}_{II}^{-1}b_I \tag{3}$$

The matrix $\mathcal{S}$ is referred to as the *Schur complement matrix*. This reformulation leads to a general strategy for solving (1). Specifically, an iterative method can be applied to (2). Once x_Γ is known, x_I can be computed with one additional solve on the sub-graph interiors.

Let Γ denote the entire interface defined by $\Gamma = \bigcup \Gamma_i$; we notice that if two sub-graphs G_i and G_j share an interface then $\Gamma_i \cap \Gamma_j \neq \emptyset$. As interior unknowns are no longer considered, new restriction operators must be defined as follows. Let $\mathcal{R}_{\Gamma_i} : \Gamma \to \Gamma_i$ be the canonical point-wise restriction which maps full vectors defined on Γ into vectors defined on Γ_i. Thus, in the case of many sub-graphs, the fully assembled global Schur $\mathcal{S}$ is obtained by summing the contributions over the sub-graphs. The global Schur complement matrix (3) can be written as the sum of elementary matrices

$$\mathcal{S} = \sum_{i=1}^{N} \mathcal{R}_{\Gamma_i}^T \mathcal{S}_i \mathcal{R}_{\Gamma_i} \tag{4}$$

where

$$\mathcal{S}_i = \mathcal{A}_{\Gamma_i\Gamma_i} - \mathcal{A}_{\Gamma_i\mathcal{I}_i}\mathcal{A}_{\mathcal{I}_i\mathcal{I}_i}^{-1}\mathcal{A}_{\mathcal{I}_i\Gamma_i} \tag{5}$$

is a local Schur complement associated with G_i. It can be defined in terms of sub-matrices from the local matrix $\mathcal{A}_i$ defined by

$$\mathcal{A}_i = \begin{pmatrix} \mathcal{A}_{\mathcal{I}_i\mathcal{I}_i} & \mathcal{A}_{\mathcal{I}_i\Gamma_i} \\ A_{\Gamma_i\mathcal{I}_i} & A_{\Gamma_i\Gamma_i} \end{pmatrix} \tag{6}$$

While the Schur complement system is significantly better conditioned than the original matrix $\mathcal{A}$, it is important to consider further preconditioning when employing a Krylov method. It is well known, for example, that $\kappa(A) = \mathcal{O}(h^{-2})$ when $\mathcal{A}$ corresponds to a standard discretization (e.g. piecewise linear finite elements) of the Laplace operator on a mesh with spacing h between the grid points. Using two non-overlapping sub-graphs effectively reduces the condition number of the Schur complement matrix to $\kappa(S) = \mathcal{O}(h^{-1})$. While improved, preconditioning can significantly lower this condition number further.

We introduce the general form of the preconditioner considered in this work. The preconditioner presented below was originally proposed in [20] in two dimensions and successfully applied to large three-dimensional problems and real life applications in [21,22]. To describe this preconditioner we define the local assembled Schur complement, $\bar{\mathcal{S}}_i = \mathcal{R}_{\Gamma_i}S\mathcal{R}_{\Gamma_i}^T$, that corresponds to the restriction of the Schur complement to the interface Γ_i. This local assembled preconditioner can be built from the local Schur complements $\mathcal{S}_i$ by assembling their diagonal blocks.

With these notations the preconditioner reads

$$M_d = \sum_{i=1}^{N} \mathcal{R}_{\Gamma_i}^T \bar{\mathcal{S}}_i^{-1} \mathcal{R}_{\Gamma_i} \tag{7}$$

If we considered a planar graph partitioned into horizontal strips (1D decomposition), the resulting Schur complement matrix has a block tridiagonal structure as depicted in (8)

$$\begin{pmatrix} \ddots & & & & \\ & S_{k,k} & S_{k,k+1} & & \\ & S_{k+1,k} & S_{k+1,k+1} & S_{k+1,k+2} & \\ & & S_{k+1,k+2} & S_{k+2,k+2} & \\ & & & & \ddots \end{pmatrix} \tag{8}$$

For that particular structure of $\mathcal{S}$ the submatrices in boxes correspond to the $\bar{\mathcal{S}}_i$. Such diagonal blocks, that overlap, are similar to the classical block overlap of the Schwarz method when writing in a matrix form for 1D decomposition. Similar ideas have been developed in a pure algebraic context in earlier papers [23,24] for the solution of general sparse linear systems. Because of this link, the preconditioner defined by (7) is referred to as algebraic additive Schwarz for the Schur complement. One advantage of using the assembled local Schur complements instead of the local Schur complements (like in the Neumann–Neumann [25,26]) is that in the SPD case the assembled Schur complements cannot be singular (as $\mathcal{S}$ is SPD [20]).

The original idea of non-overlapping domain decomposition method consists into subdividing the graph into sub-graphs that are individually mapped to one processor. With this data distribution, each processor P_i can concurrently partially factorize it to compute its local Schur complement $\mathcal{S}_i$. This is the first computational phase that is performed concurrently and independently by all the processors. The second step corresponds to the construction of the preconditioner. Each processor communicates with its neighbors (in the graph partitioning sense) to assemble its local Schur complement $\bar{\mathcal{S}}_i$ and performs its factorization. This step only requires a few point-to-point communications. Finally, the last step is the iterative solution of the interface problem (2). For that purpose, parallel matrix-vector product involving $\mathcal{S}$, the preconditioner $M_\star$ and dot-product calculation must be performed. For the matrix-vector product each processor P_i performs its local matrix-vector product involving its local Schur complement and communicates with its neighbors to assemble the computed vector to comply with Eq. (4). Because the preconditioner (7) has a similar form as the Schur complement (4), its parallel application to a vector is implemented similarly. Finally, the dot products are performed as follows: each processor P_i performs its local dot-product and a global reduction is used to assemble the result. In this way, the hybrid implementation can be summarized by the above main three phases.

Classical parallel implementations (*1-level parallel*) of such a hybrid scheme assign one sub-graph per processor. We believe that applying only this paradigm to very large applications has some drawbacks and limitations. For many applications, increasing the number of sub-graphs leads to increasing the number of iterations to converge. If no efficient numerical mechanism, such as coarse space correction for elliptic problems [27,15], is available the convergence rate might be significantly deteriorated and the solution process becomes ineffective. In order to alleviate the numerical growth of the iterations, when the number of sub-graphs is increased to feed each processor, we might keep the number of sub-graphs small while handling each sub-graph by more than one processor introducing *2-levels* of parallelism. Such an implementation enables us to exploit the natural parallelism in the computation on the sub-graphs but also parallelism within the treatment of each sub-graph. A natural implementation that complies with the hardware features of forthcoming many-core cluster exploits message passing to handle the parallelism between the sub-graphs and multi-threading on a many-core node for the calculation associated to each sub-graph. This latter computation mainly requires to used a multi-threaded sparse direct solver such as [28,29]. The other numerical kernel can be implemented using multi-threaded BLAS or LAPACK such as those available in PLASMA [30] or in MAGMA [31] for computation on heterogeneous computing nodes composed by CPU and GPU. This *multi-level* parallelism capability is more particularly well suited to the hierarchical structure of peta/exaflop computers and is the main point of our algorithmic hierarchical approach.

2.3. Numerical and parallel performance

The construction of the proposed local preconditioners can be computationally expensive because the dense matrices $\mathcal{S}_i$ should be factorized. We intend to reduce the storage and the computational cost to form and apply the preconditioner by using sparse approximation of $\bar{\mathcal{S}}_i$ in M_d following the strategy described by (9). The approximation $\hat{\mathcal{S}}_i$ can be constructed by dropping the elements of $\bar{S}_i$ that are smaller than a given threshold. More precisely, the following symmetric dropping formula can be applied:

$$\hat{s}_{\ell j} = \begin{cases} 0, & \text{if } |\bar{s}_{\ell j}| \leqslant \xi(|\bar{s}_{\ell\ell}| + |\bar{s}_{jj}|) \\ \bar{s}_{\ell j}, & \text{otherwise} \end{cases} \tag{9}$$

where $\bar{s}_{\ell j}$ denotes the entries of $\bar{\mathcal{S}}_i$. The resulting preconditioner based on these sparse approximations reads

$$M_{sp} = \sum_{i=1}^{N} \mathcal{R}_{\Gamma_i}^T \hat{\mathcal{S}}_i^{-1} \mathcal{R}_{\Gamma_i}$$

We notice that such a dropping strategy preserves the symmetry in the symmetric case but it requires to first assemble $\bar{S}_i$ before sparsifying it.

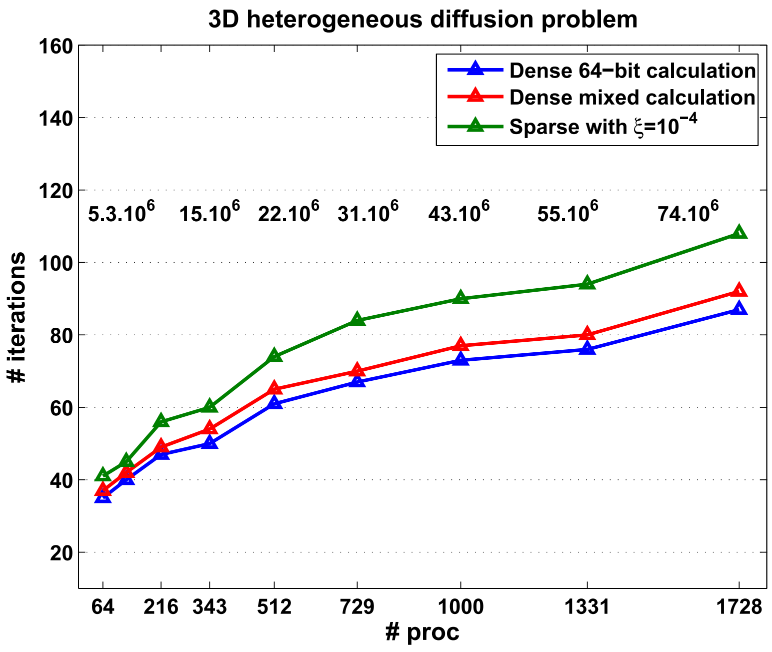

Fig. 1. Number of preconditioned conjugate gradient iterations when the global problem size varies from 2.7 million (27 cores) up to 75 million (1728 cores).

Motivated by accuracy reasons, many large-scale scientific applications and industrial numerical simulation codes are fully implemented in 64-bit floating-point arithmetic. On the other hand, many recent processor architectures exhibit 32-bit computational power that is significantly higher than that for 64-bit. We might legitimately ask whether all the calculation should be performed in 64-bit or if some pieces could be carried out in 32-bit. This leads to the design of mixed-precision algorithms. Particular care is necessary when choosing the part to be computed in 32-bit arithmetic so that the introduced rounding error or the accumulation of these rounding errors does not produce a meaningless solution. We propose to take advantage of the 32-bit speed and memory benefit and build some part of the code in 32-bit arithmetic. Our goal is to use costly 64-bit arithmetic only where necessary to preserve accuracy. We consider here a simple approach of performing all the steps of the Krylov subspace methods except the preconditioning in 64-bit. In this respect, it is important to note that the preconditioner only attempts to approximate the inverse of the matrix S so introducing a slight perturbation by performing this step in low precision might not affect dramatically the convergence rate of the iterative scheme. In our mixed-precision implementation only the preconditioned residual is computed using 32-bit arithmetic. The consequence of this strategy [32,33] is that the Gaussian elimination (factorization) of the local assembled Schur complement (used as preconditioner), and the forward and the backward substitutions to compute the preconditioned residual, are performed in 32-bit while the rest of the algorithm is implemented in 64-bit. When the local assembled Schur complement is dense, cutting the size of this matrix in half has a considerable effect in terms of memory space. Another benefit is in the total amount of communication that is required to assemble the preconditioner. As for the memory required to store the preconditioner, the size of the exchanged messages is also half that for 64-bit. Consequently, if the network latency is neglected, the overall time to build the preconditioner for the 32-bit implementation should be half that for the 64-bit implementation.

In Fig. 1 (Fig. 2) we first report on the weak scalability of our solver in term of iteration count (resp. parallel elapsed time). We solve linear systems resulting from the discretization of a 3D heterogeneous diffusion problem define in a cube; the size of the sub-graph (i.e., number of vertices/unknowns) is kept constant when the number of sub-graphs is varied. In these experiments we solve a 2.7 million unknown problem on 27 cores all the way up to a 75 million problem on 1728 cores. For those experiments, the two-level parallel implementation is not used.

None of the preconditioners implements any coarse spatial component to account for the global coupling of the elliptic PDEs, hence they do not scale perfectly when the number of sub-graphs is increased. However, the scalability is not that bad and clearly much better than that observed on two-dimensional examples [20]. The number of iterations is multiplied by about two to three when going from 27 to 1728 cores (i.e., multiplying by about 64 the number of processors).

From a computational view point the behavior is slightly different as the global cost of the computation is not completely governed by the iterative part. The initialization phase, i.e., factorization of the local problem and setup of the preconditioner, involve direct numerical kernels whose cost is almost constant for all the runs. Consequently the global elapsed time is only multiplied by 1.5 while the size of the problem solved is multiplied by 64. An ideally scalable solver would have a constant elapsed time. Those experiments illustrate the numerical robustness of our hybrid solver that can be further enhanced by exploiting multiple levels of parallelism.

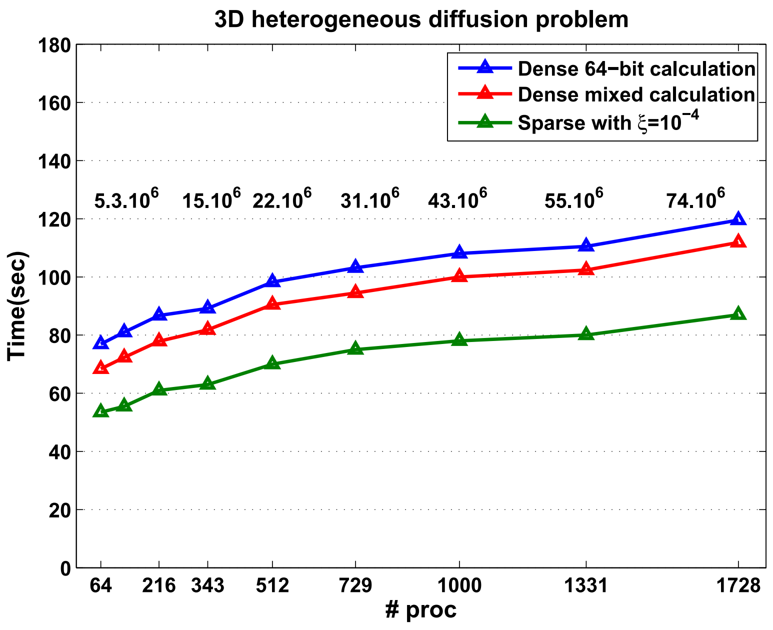

Fig. 2. Parallel elapsed time when the global problem size varies from 2.7 million (27 cores) up to 75 million (1728 cores).

Table 1
Numerical performance and advantage of the *2-level parallel* method compared to the standard *1-level parallel* method for a 3D structural mechanics problem.

# total processors	Algo	# sub-graphs	# processors/sub-graph	# iter	Iterative loop time
Fuselage 1 million elements with 6.5×10^6 *dof*					
16 processors	*1-level parallel*	16	1	147	77.9
	2-level parallel	8	2	98	51.4
32 processors	*1-level parallel*	32	1	176	58.1
	2-level parallel	16	2	147	44.8
	2-level parallel	8	4	98	32.5
64 processors	*1-level parallel*	64	1	226	54.2
	2-level parallel	32	2	176	40.1
	2-level parallel	16	4	147	31.3
	2-level parallel	8	8	98	27.4

In order to illustrate the benefit of using the *2-level parallel* implementation we consider a linear system arising from a structural mechanics simulation. It corresponds to the discretization of a section of the fuselage of an aircraft where unstructured Midlinn shell elements are used. The linear systems is of size 6.5 millions. We display in Table 1, the measured performance of the *2-level parallel* implementation on an 8-way multi-core IBM SMP cluster. For a fixed number of cores we vary the number of sub-graphs and the number of cores allocated to each sub-graph. It can be seen that the lower the number of sub-graphs, the faster the convergence and the smaller the solution time.

The numerical attractive features of the *2-level parallel* approach is that increasing the number of processors to speedup the solution of large linear systems does not imply increasing the number of iterations to converge as it is often the case with the *1-level parallel* approach. This strategy is very attractive in the case where we have a system with many right hand side to solve. In this case it is better to keep the number of iterations as small as possible.

The parallel implementations of the numerical kernels involved in the iterative loop is efficient enough to speedup the solution time. When we have 32 processors, standard (*1-level parallel*) implementation partitions the global graph into 32 sub-graphs; it requires 176 iterations to convergence and get the solution in 58.1 seconds. With the *2-level parallel* implementation, either 16 or 8 sub-graphs can be used. The 16 sub-graph partition requires 147 iterations performed in 44.8 seconds and the 8 sub-graph calculation needs 98 iterations performed in 32.5 seconds. This example illustrates the advantage of the *2-level parallel* implementation from a numerical viewpoint.

3. Concluding remarks

In this paper, we illustrate a hierarchical algorithmic approach for the implementation of an efficient parallel hybrid sparse solver. The main goal of the design of *2-levels* parallelism algorithm is the efficient use of parallel modern computers in offering an optimal trade-off between numerical and parallel efficiency. The combination of the two levels of parallelism enables an optimal usage of the computing resources while preserving attractive numerical performance. Consequently, such a numerical technique appears as a promising candidate for intensive simulations on future many-core parallel platforms.

References

[1] A. Buttari, J. Langou, J. Kurzak, J.J. Dongarra, A class of parallel tiled linear algebra algorithms for multicore architectures, Parallel Comput. Syst. Appl. 35 (2009) 38–53.

[2] E. Chan, E.S. Quintana-Orti, G. Gregorio Quintana-Orti, R. van de Geijn, Supermatrix out-of-order scheduling of matrix operations for SMP and multicore architectures, in: Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures SPAA'07, June 2007, pp. 116–125.

[3] J. Demmel, L. Grigori, M. Hoemmen, J. Langou, Communication-optimal parallel and sequential QR and LU factorizations, LAPACK Working Note 204, August 2008.

[4] E. Agullo, C. Augonnet, J. Dongarra, H. Ltaief, R. Namyst, S. Thibault, S. Tomov, Faster, cheaper, better – a hybridization methodology to develop linear algebra software for GPUs, Technical Report 230, LAPACK Working Note, September 2010.

[5] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, H. Haidar, T. Herault, J. Kurzak, J. Langou, P. Lemarinier, H. Ltaief, P. Luszczek, A. YarKhan, J. Dongarra, Distributed-memory task execution and dependence tracking within DAGuE and the DPLASMA project, Technical Report 232, LAPACK Working Note, September 2010.

[6] N.J. Higham, Accuracy and Stability of Numerical Algorithms, second ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002.

[7] Y. Saad, Iterative Methods for Sparse Linear Systems, second ed., SIAM, Philadelphia, 2003.

[8] M.R. Hestenes, E. Stiefel, Methods of conjugate gradients for solving linear system, J. Res. Nat. Bur. Stds. B 49 (1952) 409–436.

[9] Y. Saad, M.H. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comp. 7 (1986) 856–869.

[10] W. Hackbusch, Multigrid Methods and Applications, Springer-Verlag, Berlin, 1985.

[11] M.A. Heroux, P. Raghavan, H.D. Simon, Parallel Processing for Scientific Computing, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2006.

[12] R. Barrett, M. Berry, T.F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. Van der Vorst, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, second ed., SIAM, Philadelphia, PA, 1994.

[13] J. Drkošová, M. Rozložník, Z. Strakoš, A. Greenbaum, Numerical stability of the GMRES method, BIT 35 (1995) 309–330.

[14] A. Greenbaum, Iterative Methods for Solving Linear Systems, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997.

[15] B.F. Smith, P. Bjørstad, W. Gropp, Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations, first ed., Cambridge University Press, New York, 1996.

[16] A. Quarteroni, A. Valli, Domain Decomposition Methods for Partial Differential Equations, Numerical Mathematics and Scientific Computation, Oxford Science Publications, Oxford, 1999.

[17] T. Mathew, Domain Decomposition Methods for the Numerical Solution of Partial Differential Equations, Springer Lecture Notes in Computational Science and Engineering, Springer, 2008.

[18] G. Karypis, V. Kumar, MeTiS – Unstructured graph partitioning and sparse matrix ordering system, version 2.0, University of Minnesota, June 1995.

[19] C. Chevalier, F. Pellegrini, PT-SCOTCH: a tool for efficient parallel graph ordering, Parallel Comput. 34 (6–8) (2008).

[20] L.M. Carvalho, L. Giraud, G. Meurant, Local preconditioners for two-level non-overlapping domain decomposition methods, Numer. Linear Algebra Appl. 8 (4) (2001) 207–227.

[21] L. Giraud, A. Haidar, L.T. Watson, Parallel scalability study of hybrid preconditioners in three dimensions, Parallel Comput. 34 (2008) 363–379.

[22] A. Haidar, On the parallel scalability of hybrid solvers for large 3D problems, PhD dissertation, INPT, June 2008. TH/PA/08/57.

[23] X.-C. Cai, Y. Saad, Overlapping domain decomposition algorithms for general sparse matrices, Numer. Linear Algebra Appl. 3 (1996) 221–237.

[24] G. Radicati, Y. Robert, Parallel conjugate gradient-like algorithms for solving nonsymmetric linear systems on a vector multiprocessor, Parallel Comput. 11 (1989) 223–239.

[25] J.-F. Bourgat, R. Glowinski, P. Le Tallec, M. Vidrascu, Variational formulation and algorithm for trace operator in domain decomposition calculations, in: Tony Chan, Roland Glowinski, Jacques Périaux, Olof Widlund (Eds.), Domain Decomposition Methods, SIAM, Philadelphia, PA, 1989, pp. 3–16.

[26] Y.-H. De Roeck, P. Le Tallec, Analysis and test of a local domain decomposition preconditioner, in: Roland Glowinski, Yuri Kuznetsov, Gérard Meurant, Jacques Périaux, Olof Widlund (Eds.), Fourth International Symposium on Domain Decomposition Methods for Partial Differential Equations, SIAM, Philadelphia, PA, 1991, pp. 112–128.

[27] P.E. Bjørstad, O.B. Widlund, Iterative methods for the solution of elliptic problems on regions partitioned into substructures, SIAM J. Numer. Anal. 23 (6) (1986) 1093–1120.

[28] P. Hénon, P. Ramet, J. Roman, PaStiX: A high-performance parallel direct solver for sparse symmetric definite systems, Parallel Comput. 28 (2) (January 2002) 301–321.

[29] J.W. Demmel, J.R. Gilbert, X.S. Li, An asynchronous parallel supernodal algorithm for sparse Gaussian elimination, SIAM J. Matrix Anal. Appl. 20 (4) (1999) 915–952.

[30] PLASMA users' guide, parallel linear algebra software for multicore architectures, version 2.0, http://icl.cs.utk.edu/plasma, November 2009.

[31] S. Tomov, R. Nath, P. Du, J. Dongarra, MAGMA, version 0.2, user guide, http://icl.cs.utk.edu/magma, November 2009.

[32] J. Kurzak, J. Dongarra, Implementation of the mixed-precision high performance LINPACK benchmark on the CELL processor, Technical Report LAPACK Working Note #177 UT-CS-06-580, University of Tennessee Computer Science, September 2006.

[33] J. Langou, J. Langou, P. Luszczek, J. Kurzak, A. Buttari, J. Dongarra, Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy, Technical Report LAPACK Working Note #175 UT-CS-06-574, University of Tennessee Computer Science, April 2006.