Parallel scalability study of three dimensional additive Schwarz preconditioners in non-overlapping domain decomposition

L. Giraud* A. Haidar† L. T. Watson‡

16th January 2007

Abstract

In this paper we study the parallel scalability of variants of additive Schwarz preconditioners for three dimensional non-overlapping domain decomposition methods. To alleviate the computational cost, both in terms of memory and floating-point complexity, we investigate variants based on a sparse approximation or on mixed 32- and 64-bit calculation. The robustness of the preconditioners is illustrated on a set of linear systems arising from the finite element discretization of elliptic PDEs through extensive parallel experiments on up to 1000 processors. Their efficiency from a numerical and parallel performance view point are studied.

1 Introduction

In recent years, there has been important development of domain decomposition algorithms for numerically solving partial differential equations. Nowadays some preconditioners for Krylov methods possess optimal convergence rates for given classes of elliptic problems. These optimality or quasi-optimality properties are often achieved thanks to the use of two-level preconditioners that are composed of local and global terms acting either in an additive or in a multiplicative way. A concise and nice overview of domain decomposition techniques can be found in [6]. A series of useful books are also available [18, 21, 22] where the interested reader can find detailed presentations of these numerical techniques and a complete bibliography.

In this paper we study local components for non-overlapping domain decomposition applied to the parallel solution of large three dimensional elliptic PDE problems. In Section 2, we describe a set of parallel local preconditioners that are the main focus of this paper. In order to alleviate the computational cost of constructing these new local preconditioners, that require the explicit computation of the local Schur complement, we propose cheaper alternatives based on a sparse approximation or on mixed 32- and 64-bit calculation. This latter strategy is mainly motivated by the observation that many recent processor architectures exhibit 32-bit computational power that is significantly higher than that for 64-bit. We show experimental results that demonstrate their numerical efficiency. The parallel distributed implementation of these techniques is described in Section 3 and an extensive parallel scalability study on large numbers of processors is commented on in Section 4. These numerical experiments are performed on two types of partial differential equations, heterogeneous and/or anisotropic problems, in three dimensional domains using up to a thousand processors. Finally, in Section 5 we show a few results to illustrate how these local preconditioners can be combined with a coarse spatial correction to improve the numerical scalability, and make a few concluding remarks.

*ENSEEIHT-IRIT, 2 Rue Camichel 31071 Toulouse Cedex, France. giraud@n7.fr
†CERFACS, 42 Avenue G. Coriolis, 31057 Toulouse Cedex, France. haidar@cerfacs.fr
‡Departments of Computer Science and Mathematics, Virginia Polytechnic Institute & State University, Blacksburg, Virginia, USA. ltw@cs.vt.edu

2 Algebraic additive Schwarz preconditioner and its variants

We consider the second order self-adjoint three dimensional elliptic problem

$$\left\{ \begin{array}{rcl} -\mathrm{div}(K.\nabla u) &=& f \quad \mathrm{in} \quad \Omega, \\ u &=& 0 \quad \mathrm{on} \quad \partial\Omega, \end{array} \right. \tag{1}$$

where K is a strictly positive definite bounded symmetric tensor on the open convex bounded computational domain Ω.

We assume that the domain Ω is decomposed into N non-overlapping open subdomains Ω_1, ..., Ω_N with boundaries $\partial\Omega_1$, ..., $\partial\Omega_N$. We assume that a mesh is given that is a refinement of the subdomain partitioning. We discretize (1) by linear finite elements resulting in a symmetric and positive definite linear system

$$Au = f.$$

Let Γ be the set of all the indices of the mesh points that belong to the interfaces between the subdomains. Grouping the unknowns for the mesh points corresponding to Γ in the vector u_Γ and the ones corresponding to the unknowns in the interior I of the subdomains in u_I, boundaries, we get the reordered problem:

$$\begin{pmatrix} A_{II} & A_{I\Gamma} \\ A_{I\Gamma}^T & A_{\Gamma\Gamma} \end{pmatrix} \quad \begin{pmatrix} u_I \\ u_\Gamma \end{pmatrix} \quad = \quad \begin{pmatrix} f_I \\ f_\Gamma \end{pmatrix} \ . \tag{2}$$

Eliminating u_I from the second block row of (2) leads to the following reduced equation for u_Γ:

$$S u_\Gamma = f_\Gamma - A_{I\Gamma}^T A_{II}^{-1} f_I \ , \tag{3}$$

where

$$S = A_{\Gamma\Gamma} - A_{I\Gamma}^T A_{II}^{-1} A_{I\Gamma} \tag{4}$$

is the Schur complement of the matrix A_{II} in A. The matrix S inherits from A the symmetric positive definiteness property. Therefore we use preconditioned conjugate gradient iterations [10] for solving (3).

For the sake of simplicity, we describe the basis of our local preconditioner in two dimensions as its generalization to three dimensions is straightforward. In Figure 1, we depict an internal

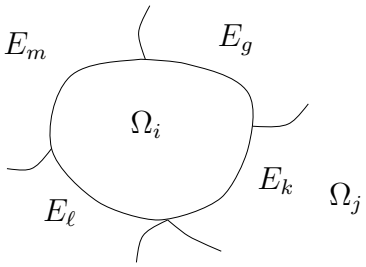


Figure 1: An internal subdomain.

subdomain Ω_i with its edge interfaces E_m, E_g, E_k, and E_ℓ that define $\Gamma_i = \partial\Omega_i \backslash \partial\Omega$. Let $R_{\Gamma_i} : \Gamma \rightarrow \Gamma_i$ be the canonical pointwise restriction that maps full vectors defined on Γ into vectors defined on Γ_i, and let $R_{\Gamma_i}^T : \Gamma_i \rightarrow \Gamma$ be its transpose. For a stiffness matrix A arising from a finite element discretization, the Schur complement matrix (4) can also be written

$$S = \sum_{i=1}^{N} R_{\Gamma_i}^T S^{(i)} R_{\Gamma_i},$$

where

$$S^{(i)} = A_{\Gamma_i}^{(i)} - A_{i\Gamma_i}^T A_{ii}^{-1} A_{i\Gamma_i} \tag{5}$$

is referred to as the local Schur complement associated with the subdomain Ω_i. $S^{(i)}$ involves submatrices from the local stiffness matrix $A^{(i)}$, defined by

$$A^{(i)} = \begin{pmatrix} A_{ii} & A_{i\Gamma_i} \\ A_{i\Gamma_i}^T & A_{\Gamma_i}^{(i)} \end{pmatrix}. \tag{6}$$

The matrix $A^{(i)}$ corresponds to the discretization of Equation (1) on the subdomain Ω_i with Neumann boundary condition on Γ_i and A_{ii} corresponds to the discretization of Equation (1) on the subdomain Ω_i with homogeneous Dirichlet boundary conditions on Γ_i. The local Schur complement matrix, associated with the subdomain Ω_i depicted in Figure 1, is dense and has the following 4×4 block structure:

$$S^{(i)} = \begin{pmatrix} S_{mm}^{(i)} & S_{mg} & S_{mk} & S_{m\ell} \\ S_{gm} & S_{gg}^{(i)} & S_{gk} & S_{g\ell} \\ S_{km} & S_{kg} & S_{kk}^{(i)} & S_{k\ell} \\ S_{\ell m} & S_{\ell g} & S_{\ell k} & S_{\ell\ell}^{(i)} \end{pmatrix}, \tag{7}$$

where each block accounts for the interactions between the degrees of freedom of the edges of the interface $\partial\Omega_i$.

While the Schur complement system is significantly better conditioned than the original matrix 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 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 subdomains 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. The preconditioner presented below was originally proposed in [4] in two dimensions and successfully applied to large two dimensional semiconductor device modeling in [9]. To describe this preconditioner we define the local assembled Schur complement, $\bar{S}^{(i)} = R_{\Gamma_i} S R_{\Gamma_i}^T$, that corresponds to the restriction of the Schur complement to the interface Γ_i. This local assembled preconditioner can be build from the local Schur complements $S^{(i)}$ by assembling their diagonal blocks thanks to a few neighbor to neighbor communications. For instance, the diagonal blocks of the complete matrix S associated with the edge interface E_k, depicted in Figure 1, is $S_{kk} = S_{kk}^{(i)} + S_{kk}^{(j)}$. Assembling each diagonal block of the local Schur complement matrices, we obtain the local assembled Schur complement, that is:

$$\bar{S}^{(i)} = \begin{pmatrix} S_{mm} & S_{mg} & S_{mk} & S_{m\ell} \\ S_{gm} & S_{gg} & S_{gk} & S_{g\ell} \\ S_{km} & S_{kg} & S_{kk} & S_{k\ell} \\ S_{\ell m} & S_{\ell g} & S_{\ell k} & S_{\ell\ell} \end{pmatrix}.$$

With these notations the algebraic additive Schwarz preconditioner reads

$$M_d = \sum_{i=1}^{N} R_{\Gamma_i}^T \left(\bar{S}^{(i)} \right)^{-1} R_{\Gamma_i}. \tag{8}$$

2.1 Sparse Algebraic Additive Schwarz preconditioner

In three dimensional problems the size of the dense local Schur matrices can be large, consequently it is computationally expensive to factorize and solve linear systems with them. One possible alternative to get a cheaper preconditioner is to consider a sparse approximation for $\bar{S}^{(i)}$ in (8),

which may result in a saving of memory to store the preconditioner and saving of computation to factorize and apply it. This approximation $\hat{S}^{(i)}$ 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:

$$\hat{s}_{ij} = \begin{cases} 0, & \text{if } |s_{ij}| \leq \xi(|s_{ii}| + |s_{jj}|), \\ s_{ij}, & \text{otherwise.} \end{cases} \tag{9}$$

The resulting preconditioner reads

$$M_{sp} = \sum_{i=1}^{N} R_{\Gamma_i}^T \left(\hat{S}^{(i)}\right)^{-1} R_{\Gamma_i}.$$

2.2 Single precision Additive Schwarz preconditioner

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. However, the switch from 64-bit operations into 32-bit operations increases rounding error. Thus we have to be careful when choosing 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 a Krylov subspace method except the preconditioning in 64-bit [14]. 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 in 32-bit. The import of this strategy [13, 12] is that the Gaussian elimination (factorization) of the local assembled Schur complement (used as preconditioner), and the forward and the back substitutions to compute the preconditioned residual, are performed in 32-bit while the rest of the algorithm is implemented in 64-bit.

Since 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. These improvements are illustrated by detailed numerical experiments with the mixed-precision implementation reported in Section 4.

To summarize, we have four variants of the additive Schwarz preconditioner that use various ways to represent the assembled local Schur complement:

$M_{\mathrm{d}-64}$ that uses dense 64-bit matrices;

$M_{\mathrm{d}-\mathrm{mix}}$ that uses dense 32-bit matrices,

$M_{\mathrm{sp}-64}$ that uses sparsified 64-bit matrices;

$M_{\mathrm{sp}-\mathrm{mix}}$ that uses sparsified 32-bit matrices.

3 Parallel implementation

In a parallel distributed memory environment, the domain decomposition strategy is followed to assign each local PDE problem (subdomain) to one processor that works independently of other processors and exchange data using message passing. In what follows we start by taking a brief look at our parallel implementation that relies on a unique feature of the multifrontal sparse direct solver MUMPS [1, 2]; that offers the possibility to compute the Schur complements matrices $S^{(i)}$ at an affordable memory and computational cost thanks to its multifrontal approach [7]. Basically, the Schur complement feature of MUMPS can be viewed as an incomplete factorization, where the factorization of the root ($A_{\Gamma\Gamma}$), associated with the interface indices, is disabled. Consequently this feature fully benefits from the general overall efficiency of the multifrontal approach. Those local Schur complement matrices computed explicitly on each processor are then assembled using neighbor to neighbor communication, which is independent of the number of processors. Then they are either factorized using a dense linear LAPACK kernel [3], to construct the dense additive Schwarz preconditioners denoted by $M_{\mathrm{d}-64}$ or $M_{\mathrm{d}-\mathrm{mix}}$, or first sparsified and then factorized again using MUMPS for the sparsified alternatives. We refer to this technique as a sparsified additive Schwarz preconditioner and denote it as $M_{\mathrm{sp}-64}$ and $M_{\mathrm{sp}-\mathrm{mix}}$. Finally, we note that the solution of this reduced linear system associated with the Schur complement is typically performed by a distributed preconditioned conjugate gradient solver. In modern parallel numerical libraries [8], the Krylov solvers are implemented using the reverse communication mechanism such that only three external parallel computational kernels have to be implemented.

The first is the dot product calculation, which is the only operation that require an overall communication using the MPI_ALLREDUCE routine.

The second is the matrix-vector product involving the local Schur complement. We notice that with the local Schur complement being explicitly computed, the matrix-vector product is performed using dense kernels from BLAS. Such a calculation makes the iterations cheaper than the standard approach involving backward/forward substitutions with the sparse Cholesky factors associated with the Dirichlet problems. This computational step is followed by a few neighbor to neighbor communications to assemble the resulting vector defined on the interface.

The third is the preconditioning step, which requires a backward/forward substitution of the factorized local assembled, possibly sparsified, Schur matrix $\bar{S}^{(i)}$, followed by some neighbor to neighbor communication to sum all the contributions from the interface degrees of freedom.

4 Parallel scalability studies

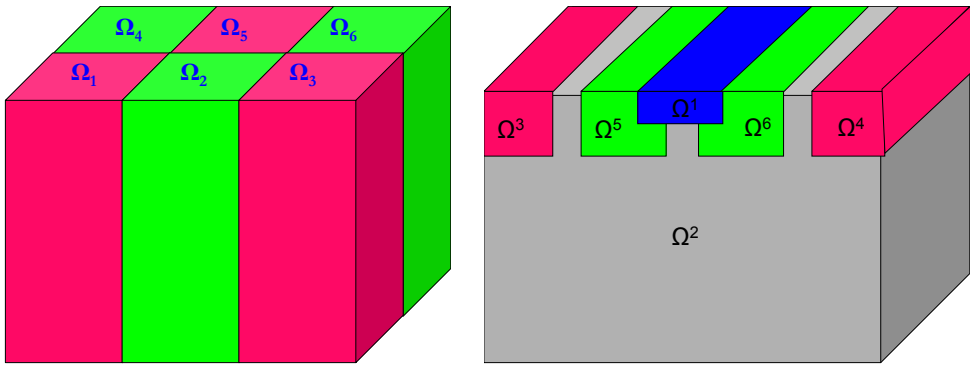
In this part of the paper, we first describe in Section 4.1 the computational framework considered for our parallel numerical experiments. We investigate in Section 4.2 the numerical behaviors of the sparsified and mixed arithmetic variants and compare them with the classical dense 64-bit additive Schwarz preconditioner. Section 4.3 is the core of the parallel study where we first illustrate through classical speedup experiments the advantage of increasing the number of processors for solving a problem of a prescribed size; then we study the numerical scalability of the preconditioners by conducing scaled speedup experiments where the problem size is increased linearly with the number of processors.

4.1 Computational framework

4.1.1 Target parallel platform

Although many runs have been performed on various parallel platforms, we only report in this paper on experiments conduced on the System X computer installed at Virginia Tech. This parallel

distributed computer is a 1100 dual node Apple Xserve G5 cluster machine based on 2.3 GHz PowerPC 970FX processors with a 12.25 Tflops peak performance. This computer has a distributed memory architecture, where each node has 4 GBytes ECC DDR400 (PC3200) of RAM. Thus, data sharing among processors is performed using the message passing library MVAPICH. The interconnection networks between processors are 10Gbps InfiniBand with 66 SilverStorms 9xx0 family switches and Gigabit Ethernet with 6 Cisco Systems 240-port 4506 switches.



(a) Pattern 1 (b) Pattern 2

Figure 2: variable coefficient domains.

4.1.2 Model problems

To investigate the robustness and the scalability of the preconditioners we consider various model problems by considering the diffusion coefficient matrix K in (1) as diagonal with piecewise constant function entries defined in the unit cube as depicted in Figure 2. The diagonal entries $a(x,y,z)$, $b(x,y,z)$, $c(x,y,z) \in \mathbb{R}^3$ of K are bounded positive functions on Ω enabling us to define heterogeneous and/or anisotropic problems. To vary the difficulties we consider both discontinuous and anisotropic PDEs where constant diffusion coefficients are defined either along vertical beams (Pattern 1 type problems) or horizontal beams (Pattern 2 type problems). This latter pattern corresponds to MOSFET problems arising in device modeling simulation. For the sake of completeness we also consider the classical Poisson problem where all the coefficient functions a, b and c are identically one. More precisely we define the following set of problems:

Problem 1: Poisson where $a(\cdot) = b(\cdot) = c(\cdot) = 1$.

Problem 2: heterogeneous diffusion problem based on Pattern 1;

$$a(\cdot) = b(\cdot) = c(\cdot) = \begin{cases} 1 & \text{in} \quad \Omega^1 \cup \Omega^3 \cup \Omega^5, \\ 10^3 & \text{in} \quad \Omega^2 \cup \Omega^4 \cup \Omega^6. \end{cases}$$

Problem 3: heterogeneous and anisotropic diffusion problem based on Pattern 1; $a(\cdot) = 1$ and

$$b(\cdot) = c(\cdot) = \begin{cases} 1 & \text{in} \quad \Omega^1 \cup \Omega^3 \cup \Omega^5, \\ 10^3 & \text{in} \quad \Omega^2 \cup \Omega^4 \cup \Omega^6. \end{cases}$$

Problem 4: heterogeneous and anisotropic diffusion problem based on Pattern 2; $a(\cdot) = 1$ and

$$b(\cdot) = c(\cdot) = \begin{cases} 1 & \text{in} \Omega^1, \\ 10^3 & \text{in} \Omega^2, \\ 10^{-3} & \text{in} \Omega^3 \cup \Omega^4 \cup \Omega^5 \cup \Omega^6. \end{cases}$$

4.2 Numerical behavior of the preconditioners

In this section we investigate the numerical behavior of the sparsified and mixed arithmetic preconditioners and compare them to the classical $M_{\mathrm{d}-64}$. To this end we consider the convergence history of $\frac{\|r_k\|}{\|b\|}$ along the iterations, where b denotes the right-hand side of the Schur complement system to be solved and r_k the residual at the kth iteration. We notice that this scaled residual can be viewed as a norm-wise backward error where perturbations are only considered on the right-hand side [11].

4.2.1 Sparsified preconditioners

The attractive feature of $M_{\mathrm{sp}-64}$ compared to $M_{\mathrm{d}-64}$ is that it enables us to reduce both the memory requirement to store the preconditioner and the computational cost to construct it (dense versus sparse factorization). However, the counterpart of this computing resource saving could be a deterioration of the preconditioner quality that would slow down the convergence of CG. In order to study the effect of the sparsification of the preconditioners on the convergence rate we display in Figure 3 the convergence history for various choices of the dropping parameter ξ involved in the definition of $M_{\mathrm{sp}-64}$ in (9). The reported experiments correspond to the classical Poisson problem (Figure 3 (a)) and to Problem 2 (Figure 3 (b)). For the various choices of this parameter the memory spaces required by the preconditioners on each processor are given in Table 1.

ξ	0	10^{-5}	10^{-4}	10^{-3}	10^{-2}
Memory	367_{MB}	29.3_{MB}	7.3_{MB}	1.4_{MB}	0.4_{MB}
Percentage	100%	8%	2%	0.4%	0.1%

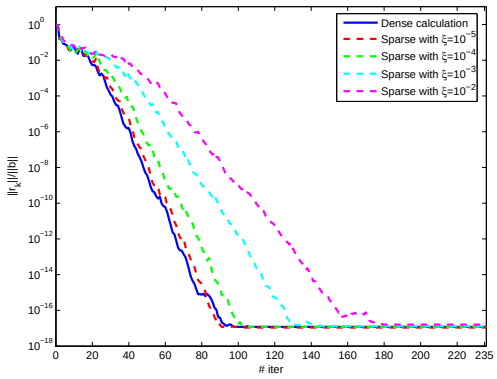
Table 1: Amount of memory in $M_{\mathrm{sp}-64}$ vs. $M_{\mathrm{d}-64}$ for various choices of the dropping parameter.

The trends that can be observed on these particular choices of problems (underlying PDEs: Poisson and Problem 2; domain partition: $350 \times 350 \times 350$ mesh partitioned into 1000 subdomains) have been observed on many other examples. That is, for small values of the dropping parameter the convergence is marginally affected while the memory saving is already significant; for larger values of the dropping parameter a lot of resources are saved in the construction of the preconditioner but the convergence becomes very poor. A reasonable trade-off between computing resource savings and convergence rate is generally for a choice of the dropping parameter equal to 10^{-4} that enables us to retain around 2% of the entries of the local Schur complement. This value for the dropping threshold is used in the rest of this paper to define $M_{\mathrm{sp}-64}$ and $M_{\mathrm{sp-mix}}$.

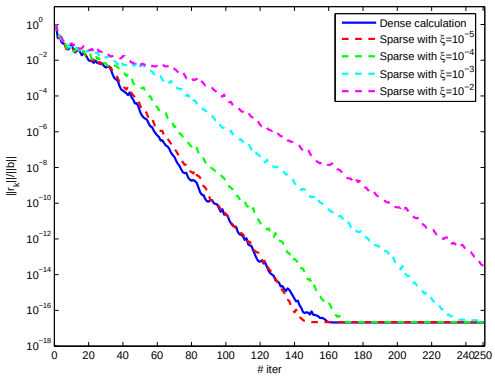
4.2.2 Mixed arithmetic preconditioners

A distinctive framework feature of this work is the use of mixed-precision preconditioners in domain decomposition, where the 32-bit calculations are expected to significantly reduce not only the elapsed time of a simulation but also the memory required to implement the preconditioner. In that respect all but the preconditioning step are implemented in high precision. The preconditioner is expected to approximatively solve the original problem, so introducing a slight perturbation by performing this step in low precision might not affect dramatically the convergence rate of the iterative scheme.

In order to compare the convergence rate of a fully 32-bit, a fully 64-bit, and a mixed-precision implementation, we depict in Figure 4 the convergence history for the three implementations of M_d. The problems that are solved correspond to Schur complement systems arising from the discretization on a $350 \times 350 \times 350$ grid (around 43 million degrees of freedom) decomposed and mapped onto 1000 processors. We display in Figure 4 (a) the convergence history for the Poisson problem, while Figure 4 (b) corresponds to Problem 2. It can be observed that for these not too ill



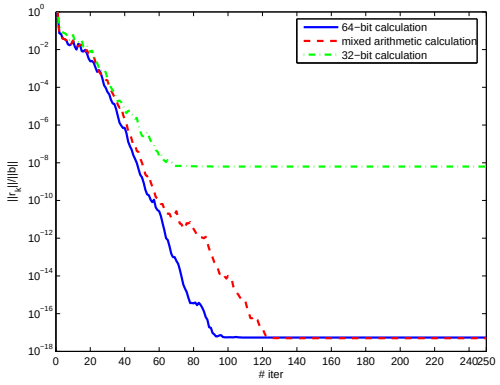
(a) Poisson problem.



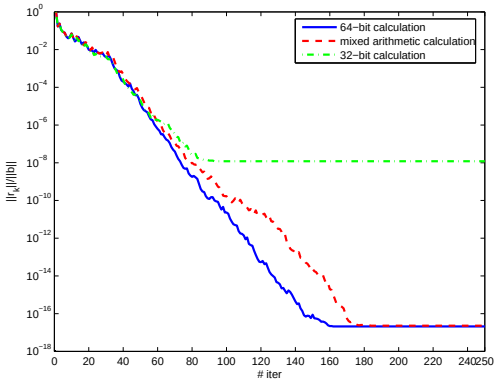
(b) Problem 2.

Figure 3: Convergence history for a $350 \times 350 \times 350$ mesh mapped onto 1000 processors for various dropping thresholds.

conditioned problems, the 32-bit calculation of the preconditioning step does not delay too much the convergence of CG. Down to the accuracy of about 32-bit machine precision, the three curves have very similar paths. As expected, the 32-bit implementation of CG reaches a limiting accuracy at the level of the single precision machine epsilon, while the full 64-bit and the mixed arithmetic implementations both attained an accuracy at the level of 64-bit machine precision. We should point out that the mixed strategy can only be considered for problems where the preconditioner is not too ill-conditioned (respectively, the initial problem is not too ill-conditioned) so that it is not singular in 32-bit arithmetic.



(a) Poisson problem.



(b) Problem 2.

Figure 4: M_d convergence history for a $350 \times 350 \times 350$ mesh mapped onto 1000 processors.

4.3 Parallel performance

For all the experiments reported in the sequel the stopping criterion for the linear solver is defined by

$$\frac{\|r_k\|}{\|b\|} \leq 10^{-8},$$

where r_k is the residual computed by CG and b the right-hand side of the Schur complement system; the initial guess is always the zero vector. We first consider experiments where the size of the initial linear system (i.e., mesh size) is kept constant when the number of processors is varied. Such iso-problem size experiments mainly emphasize the interest of parallel computation in reducing the elapsed time to solve a problem of a prescribed size. We then perform scaled experiments where the problem size is varied linearly with the number of processors. Such iso-granularity experiments illustrate the ability of parallel computation in performing large simulation (fully exploiting the local memory of the distributed platform) in ideally a constant elapsed time. For all the experiments, each subdomain is allocated to one processor.

4.3.1 Iso-problem size experiments

In these experiments we consider both the Poisson problem and a heterogeneous anisotropic problem (Problem 4) discretized on a $211 \times 211 \times 211$ grid. The number of processors is varied from 216 to 1000 and Tables 2 and 3 display the corresponding parallel elapsed time, memory requirements, and number of iterations. The number of iterations carried out is given in the row headed "# iter". The size of each of the local Schur complement matrices is given in Mbytes in the row entitled "Memory". We also display the wall-clock time taken in carrying out the computation of the Schur complement, the construction and the factorization of the preconditioner ("Setup"), and the time taken by one CG iteration ("Time/iter"). The total time to solve the problem is given by "Total". Finally we give the speedup computed using the elapsed time on 216 processors as reference.

We note that, for a fixed size problem, increasing the number of processors means decreasing the local sub-problem size. This leads to smaller local Schur complements but the global Schur system becomes larger, which contributes to an increase in the number of iterations. We observe that the growth in the number of iterations is not significant while the reduction in the data storage is very important. The amount of data managed by each processor is smaller, providing a speedup for the BLAS-2 operations and BLAS-3 operations in the direct solvers. This produces a significant drop in the setup time and in the time per iteration. Thus one can easily conclude that the growth in the number of iterations was offset by decreasing the direct solver execution time and by reducing the amount of data. We notice that superlinear speedups (i.e., speedup larger than the increase in processor number) are observed for all these experiments. This is mainly due to the nonlinear complexity of the dense and sparse direct solvers that induces a superlinear reduction in the setup time.

	# subdomains ≡ # processors				
	216	343	512	729	1000
Memory (MB)	413	223	126	77	54
# iter	33	35	37	40	43
Time/iter	0.86	0.48	0.29	0.21	0.13
Setup	67.13	26.68	12.90	6.85	4.42
Total	95.45	43.63	23.84	15.47	10.15
Speed-up	1	2.18	4.00	6.17	9.40

Table 2: Performance on the Poisson problem discretized on a $211 \times 211 \times 211$ mesh when the number of processors is varied.

4.3.2 Iso-granularity experiments

In this section we study the numerical efficiency of the preconditioners. We perform scaled experiments where either

- the size of the subdomains is kept constant (i.e., $\frac{H}{h}$ constant where H is the diameter of the subdomains and h the mesh size) when the number of subdomains is increased;
- or the number of processors is kept fixed while increasing the size of the underlying subdomain mesh (i.e., $\frac{H}{h}$ varies).

Numerical scalability. In order to illustrate the effect on the convergence rate, we report in the tables below (Tables 4–6) the number of preconditioned conjugate gradient iterations for the four considered preconditioners on various model problems and various local problem sizes. Recall that the threshold used to construct the sparse preconditioners $M_{\text{sp}-64}$ and $M_{\text{sp}-\text{mix}}$ is 10^{-4}, which enables us to retain around 2% of the entries of the local Schur complement. In these tables reading across a row shows the behavior with fixed subdomain size when the number of the processors goes from 27 up to 1000 while the overall problem size increases; for every column the number of processors (subdomains) is kept constant while refining the mesh size within each subdomain. In optimal situations, numerical scalability would mean that the convergence rate would not depend on the number of subdomains; this would lead to constant computing time when the overall size of the problem and the number of processors increase proportionally.

Table 4 is devoted to experiments on the Poisson problem, Table 5 to Problem 2, and Table 6 reports results on the heterogeneous and anisotropic problems. We can first observe that the problems with both heterogeneity and anisotropy are the most difficult to solve and that the Poisson problem is the easiest.

For all the problems, the dependency of the convergence rate on the mesh size can be significant. This behavior is similar for the four preconditioners. When we go from subdomains with about 8,000 degrees of freedom (dof) to subdomains with about 43,000 dof, the number of iterations can increase by over 25%. Notice that with such an increase in the subdomain size, the overall system size is multiplied by more than five; on 1000 processors the global system size varies from eight million dof up to about 43 million dof.

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 subdomains is increased. However, the scalability is not that bad and clearly much better than that observed on two dimensional examples [4]. The number of iterations is multiplied by about two to 3.5 when going from 27 to 1000 processors (i.e., multiplying by about 40 the number of processors). The trend of the growth is similar for all the problems and is comparable for all the variants of the preconditioners on a given problem. Notice that with 43,000 dof per subdomain the size of the linear systems that are solved varies from 1.1 million on 27 processors up to 43 million on 1000 processors.

Computing resource behavior. In the sequel, we report experiments with fixed subdomain size of about 43,000 degrees of freedom while increasing the number of processors from 125 to

	# subdomains ≡ # processors				
	216	343	512	729	1000
Memory (MB)	413	223	126	77	54
# iter	155	184	210	237	246
Time/iter	0.88	0.51	0.28	0.18	0.13
Setup	68.73	26.60	12.81	6.80	4.58
Total	205.40	121.72	72.15	51.22	38.45
Speed-up	1	1.69	2.84	4.01	5.34

Table 3: Performance on Problem 4 discretized on a $211 \times 211 \times 211$ mesh when the number of processors is varied.

		# subdomains ≡ # processors							
subdomain grid size		27	64	125	216	343	512	729	1000
$20 \times 20 \times 20$	$M_{\mathrm{d-64}}$	16	23	25	29	32	35	39	42
	$M_{\mathrm{d-mix}}$	18	24	26	31	34	38	41	46
	$M_{\mathrm{sp-64}}$	16	23	26	31	34	39	43	46
	$M_{\mathrm{sp-mix}}$	18	25	27	34	37	41	45	49
$25 \times 25 \times 25$	$M_{\mathrm{d-64}}$	17	24	26	31	33	37	40	43
	$M_{\mathrm{d-mix}}$	19	26	28	33	36	40	44	47
	$M_{\mathrm{sp-64}}$	17	25	28	34	37	42	45	49
	$M_{\mathrm{sp-mix}}$	19	26	29	36	41	44	48	53
$30 \times 30 \times 30$	$M_{\mathrm{d-64}}$	18	25	27	32	34	39	42	45
	$M_{\mathrm{d-mix}}$	20	27	29	34	38	41	48	49
	$M_{\mathrm{sp-64}}$	18	26	29	36	40	44	48	52
	$M_{\mathrm{sp-mix}}$	19	28	31	39	42	46	52	57
$35 \times 35 \times 35$	$M_{\mathrm{d-64}}$	19	26	30	33	35	40	44	47
	$M_{\mathrm{d-mix}}$	21	29	30	35	39	42	46	50
	$M_{\mathrm{sp-64}}$	19	28	30	38	46	46	50	56
	$M_{\mathrm{sp-mix}}$	21	30	33	41	44	49	54	59

Table 4: Number of preconditioned conjugate gradient iterations for the Poisson problem when the number of subdomains and the subdomain mesh size is varied.

		# subdomains ≡ # processors							
subdomain grid size		27	64	125	216	343	512	729	1000
$20 \times 20 \times 20$	$M_{\mathrm{d-64}}$	22	32	34	41	45	55	60	67
	$M_{\mathrm{d-mix}}$	23	33	37	44	48	58	63	70
	$M_{\mathrm{sp-64}}$	23	34	39	47	49	62	70	76
	$M_{\mathrm{sp-mix}}$	24	35	40	48	52	64	70	79
$25 \times 25 \times 25$	$M_{\mathrm{d-64}}$	23	33	36	44	47	58	64	69
	$M_{\mathrm{d-mix}}$	24	34	39	45	50	60	67	72
	$M_{\mathrm{sp-64}}$	25	34	41	50	53	67	74	82
	$M_{\mathrm{sp-mix}}$	26	36	43	51	57	69	78	84
$30 \times 30 \times 30$	$M_{\mathrm{d-64}}$	24	34	35	46	49	61	65	71
	$M_{\mathrm{d-mix}}$	25	35	38	47	52	64	69	74
	$M_{\mathrm{sp-64}}$	27	37	41	53	57	74	77	85
	$M_{\mathrm{sp-mix}}$	28	39	44	57	61	76	82	92
$35 \times 35 \times 35$	$M_{\mathrm{d-64}}$	25	35	40	47	50	61	67	73
	$M_{\mathrm{d-mix}}$	25	37	42	49	54	65	70	77
	$M_{\mathrm{sp-64}}$	28	41	45	56	60	74	84	90
	$M_{\mathrm{sp-mix}}$	29	43	49	59	64	80	88	96

Table 5: Number of preconditioned conjugate gradient iterations for Problem 2 when the number of subdomains and the subdomain mesh size is varied.

		# subdomains ≡ # processors							
subdomain grid size		27	64	125	216	343	512	729	1000
Problem 3									
$20 \times 20 \times 20$	$M_{\mathrm{d-64}}$	39	56	67	87	90	104	123	132
	$M_{\mathrm{d-mix}}$	45	58	69	91	94	108	126	135
	$M_{\mathrm{sp-64}}$	39	57	69	90	92	106	126	134
	$M_{\mathrm{sp-mix}}$	42	59	71	93	96	111	129	139
$25 \times 25 \times 25$	$M_{\mathrm{d-64}}$	43	57	70	91	93	106	125	138
	$M_{\mathrm{d-mix}}$	48	61	73	94	97	111	131	142
	$M_{\mathrm{sp-64}}$	44	60	73	94	97	112	131	143
	$M_{\mathrm{sp-mix}}$	45	63	76	98	101	116	135	148
$30 \times 30 \times 30$	$M_{\mathrm{d-64}}$	44	60	71	93	95	109	129	138
	$M_{\mathrm{d-mix}}$	50	63	74	96	99	114	136	143
	$M_{\mathrm{sp-64}}$	45	63	75	99	100	118	139	145
	$M_{\mathrm{sp-mix}}$	47	65	78	103	104	121	140	151
$35 \times 35 \times 35$	$M_{\mathrm{d-64}}$	44	62	72	94	96	111	131	137
	$M_{\mathrm{d-mix}}$	52	65	76	97	101	115	136	142
	$M_{\mathrm{sp-64}}$	46	66	77	102	105	120	141	149
	$M_{\mathrm{sp-mix}}$	49	69	80	106	108	126	145	155
Problem 4									
$20 \times 20 \times 20$	$M_{\mathrm{d-64}}$	49	69	81	110	127	152	156	174
	$M_{\mathrm{d-mix}}$	51	71	85	116	132	158	160	179
	$M_{\mathrm{sp-64}}$	50	69	84	111	131	154	159	177
	$M_{\mathrm{sp-mix}}$	52	72	87	116	132	157	163	181
$25 \times 25 \times 25$	$M_{\mathrm{d-64}}$	52	72	85	114	129	154	162	178
	$M_{\mathrm{d-mix}}$	55	76	89	119	134	162	171	183
	$M_{\mathrm{sp-64}}$	53	74	89	116	136	158	168	184
	$M_{\mathrm{sp-mix}}$	56	77	92	121	138	166	174	188
$30 \times 30 \times 30$	$M_{\mathrm{d-64}}$	54	75	88	118	132	158	167	180
	$M_{\mathrm{d-mix}}$	56	79	91	122	140	163	175	186
	$M_{\mathrm{sp-64}}$	55	77	92	121	146	164	173	189
	$M_{\mathrm{sp-mix}}$	58	81	96	125	143	172	180	194
$35 \times 35 \times 35$	$M_{\mathrm{d-64}}$	55	77	89	120	133	158	169	183
	$M_{\mathrm{d-mix}}$	57	80	92	126	141	166	178	188
	$M_{\mathrm{sp-64}}$	58	81	96	124	148	167	177	195
	$M_{\mathrm{sp-mix}}$	60	84	99	129	147	175	187	200

Table 6: Number of preconditioned conjugate gradient iterations for the heterogeneous and anisotropic problems when the number of subdomains and the subdomain mesh size is varied.

1000. We look at the scaled experiments from a parallel elapsed time perspective considering the overall elapsed time to solve the problems as well as the elapsed times for each individual step of the solution process. These steps are initialization, preconditioner setup, and the iterative loop.

- The initialization phase, referred to as initialization, is shared by all the implementations. It corresponds to the time for factorizing the matrix associated with the local Dirichlet problem and constructing the local Schur complement using the MUMPS package. This phase also includes the final solution for the internal dof, once the interface problem has been solved (i.e., solution of the local Dirichlet problem). The computational cost of this initialization phase is displayed in Table 7.
- The setup time to build the preconditioner includes the communication to assemble the local Schur complement through neighbor to neighbor exchanges and the factorization of the dense or sparsified resulting matrix.
- The iterative loop is the CG iterations.

The initialization times, displayed in Table 7, are independent of the number of subdomains and only depend on their size. We can again observe the nonlinear cost of the direct solver with respect to the problem size. This nonlinear behavior was the main origin of the superlinear speedups observed in the iso-problem size experiments in Section 4.3.1.

Subdomain grid size	$20 \times 20 \times 20$	$25 \times 25 \times 25$	$30 \times 30 \times 30$	$35 \times 35 \times 35$
Time	1.3	4.2	11.2	26.8

Table 7: Initialization time (sec).

The preconditioner setup time is the time required to build the preconditioner, which is the time for assembling the local assembled Schur matrix, using neighbor to neighbor communication, and factorizing this local assembled Schur matrix using LAPACK for $M_{\mathrm{d}-64}$ and $M_{\mathrm{d}-\mathrm{mix}}$, or first sparsifying and then factorizing using MUMPS for $M_{\mathrm{sp}-64}$ and $M_{\mathrm{sp}-\mathrm{mix}}$. This time is reported in Table 8. We should mention that the assembly time does not depend much on the number of processors (because the maximum communication is performed among 27 neighbors for the internal subdomains), but rather on whether 64-bit or 32-bit calculation is used. Using a mixed approach enables a reduction by a factor around 1.7 for the dense variants and 1.3 for the sparse ones. Larger savings are observed when dense and sparse variants are compared, the latter being about three times faster.

Subdomain grid size	$M_{\mathrm{d}-64}$	$M_{\mathrm{d}-\mathrm{mix}}$	$M_{\mathrm{sp}-64}$	$M_{\mathrm{sp}-\mathrm{mix}}$
$20 \times 20 \times 20$	0.93	0.56	0.50	0.23
$25 \times 25 \times 25$	3.05	1.85	1.64	1.15
$30 \times 30 \times 30$	8.73	4.82	3.51	3.01
$35 \times 35 \times 35$	21.39	12.36	6.22	4.92

Table 8: Preconditioner setup time (sec).

In Table 9 we illustrate the elapsed time scalability of the parallel preconditioned conjugate gradient iteration. In that table we only give times for 43,000 dof subdomains. The first observation is that the parallel implementation of the preconditioned conjugate gradient method scales almost perfectly as the time per iteration is nearly constant and does not depend much on the number of processors (i.e, 0.76 seconds on 125 processors and 0.82 on 1000 processors for $M_{\mathrm{d}-64}$). The main reason for this scalable behavior is the efficiency of the global reduction involved in the dot product calculation that does not depend much on the number of processors; all the other communications

are neighbor to neighbor and their costs do not depend on the number of processors. Furthermore, the relative cost of the reduction is negligible compared to the other steps of the algorithm. It can be observed that 32-bit arithmetic does not reduce much the time per iteration for both $M_{\mathrm{d-mix}}$ and $M_{\mathrm{sp-mix}}$ in comparison with the 64-bit ones $M_{\mathrm{d-64}}$ and $M_{\mathrm{sp-64}}$, respectively. This is due to the fact that the ratio of the 64-bit to the 32-bit forward/backward substitution time is only about 1.13 (compared to almost two for the factorization involved in the setup of the preconditioner phase). This marginalizes the impact of the 32-bit calculation in the preconditioning step and makes the time per iteration for both full 64-bit and mixed arithmetic very similar. Finally, it is clear that the sparsified variants are of great interest as they cut in half the time per iteration compared to their dense counterparts. Applying the sparsified preconditioners is almost eight times faster than using the dense ones.

A comparison of the overall solution times is given in Table 10 for the standard Poisson problem, in Table 11 for Problem 2, and in Table 12 for the two heterogeneous and anisotropic problems. The block row "Total" is the parallel elapsed time for the complete solution of the linear system. It corresponds to the sum of the times for all the steps of the algorithm, which are the initialization, the setup of the preconditioner, and the iterative loop. We notice that the row "Total" permits us to evaluate the parallel scalability of the complete methods (i.e., combined numerical and parallel behavior); the time should remain constant for perfectly scalable algorithms. It can be seen that the growth in the solution time is rather moderate, when the number of processors grows from 125 (about 5.3 million unknowns) to 1000 (about 43 million unknowns). Although the methods do not scale well numerically, their parallel elapsed time performances scale reasonably well. The ratio of the total elapsed time expended for running on 1000 processors to the time on 125 processors is about 1.22 for $M_{\mathrm{d-64}}$ and around 1.28 for the other three variants for the Poisson problem. These ratios are larger for the more difficult problems as the number of iterations grows.

For the Poisson problem represented in Table 10, we observe that the most expensive kernels are the initialization and the preconditioner setup. Thus for the two mixed arithmetic algorithms $M_{\mathrm{d-mix}}$ and $M_{\mathrm{sp-mix}}$ the slight increase in the number of iterations that introduces a slight increase in the elapsed time for the iterative loop is swiftly covered by the vast reduction in the preconditioner setup time, especially for the dense mixed preconditioner $M_{\mathrm{d-mix}}$. Therefore, the mixed arithmetic algorithms outperform the 64-bit ones in terms of overall computing time. By looking at the sparsified variants we observe a considerable reduction in the time per iteration and in the preconditioner setup time induced by the use of the sparse alternatives. Since the use of these variants only introduces a few extra iterations compared to their dense counterparts, this time reduction per iteration is directly reflected in a significant reduction of the total time.

The performances on Problem 2 are displayed in Table 11. The results show that the most time consuming part is the iterative loop. We see that the time saved in the preconditioner setup by the use of mixed-precision arithmetic still compensates for a slight increase in the number of iterations. Consequently on the heterogeneous diffusion problem the mixed-precision algorithm outperforms the 64-bit one. If we now look at the sparsified variant, the tremendous reductions in both the time per iteration (two times faster than the dense one) and the preconditioner setup time (three times faster than the dense one) offset the gap in the number of iterations. Consequently, the sparse alternatives clearly outperform the dense ones. Similar comments can be made for the performances on Problems 3 and 4 as shown in Table 12. In all the experiments the sparsified

# processors	125	216	343	512	729	1000
$M_{\mathrm{d-64}}$	0.76	0.76	0.77	0.78	0.79	0.82
$M_{\mathrm{d-mix}}$	0.73	0.75	0.75	0.76	0.77	0.80
$M_{\mathrm{sp-64}}$	0.40	0.41	0.41	0.42	0.42	0.44
$M_{\mathrm{sp-mix}}$	0.39	0.39	0.40	0.41	0.42	0.43

Table 9: Parallel elapsed time for one iteration of the preconditioned conjugate gradient (sec).

versions outperform their dense counterparts and the mixed sparse variant often gives the fastest scheme.

We now examine the four variants from a memory requirement perspective. For that, we depict in Table 13 the amount of memory required on each processor of the parallel platform. We report in each column of Table 13 the size in megabytes for a preconditioner when the size of the subdomains is increased. For the sparse variants, we give in parenthesis the percentage of retained entries. These figures indicate that both the mixed arithmetic approach and the sparse variant reduce significantly the memory usage. A feature of the sparse variants is that they reduce the memory usage dramatically. The mixed precision strategy cut in half the required data storage, which has a considerable effect in terms of computing system operation and execution time, and also cut in half the time for assembling the local Schur matrix, due to halving the total neighbor to neighbor subdomain communication.

5 Concluding remarks and future work

Although these preconditioners are local, consequently not numerically scalable, they exhibit a fairly good parallel time scalability as the relative cost of the setup partially hides the moderate increase in the number of iterations. A possible remedy to overcome this lack of numerical scalability is to introduce a coarse grid component. To illustrate the ability of our preconditioners to

	Total solution time					
# processors	125	216	343	512	729	1000
$M_{\mathrm{d}-64}$	71.0	73.3	75.1	79.4	83.0	86.7
$M_{\mathrm{d}-\mathrm{mix}}$	61.1	65.4	68.4	71.1	74.6	79.2
$M_{\mathrm{sp}-64}$	45.0	48.6	51.9	52.3	54.0	57.7
$M_{\mathrm{sp}-\mathrm{mix}}$	44.6	47.7	49.3	51.8	54.4	57.1
	Time in the iterative loop					
# processors	125	216	343	512	729	1000
$M_{\mathrm{d}-64}$	22.8	25.1	26.9	31.2	34.8	38.5
$M_{\mathrm{d}-\mathrm{mix}}$	21.9	26.2	29.2	31.9	35.4	40.0
$M_{\mathrm{sp}-64}$	12.0	15.6	18.9	19.3	21.0	24.6
$M_{\mathrm{sp}-\mathrm{mix}}$	12.9	16.0	17.6	20.1	22.7	25.4

Table 10: Parallel elapsed time for the solution of the Poisson problem (sec).

	Total solution time					
# processors	125	216	343	512	729	1000
$M_{\mathrm{d}-64}$	78.6	83.9	86.7	95.8	101.1	108.0
$M_{\mathrm{d}-\mathrm{mix}}$	69.8	75.9	79.7	88.6	93.1	100.8
$M_{\mathrm{sp}-64}$	51.0	56.0	57.6	64.1	68.3	72.6
$M_{\mathrm{sp}-\mathrm{mix}}$	50.8	54.7	57.3	64.5	68.7	73.0
	Time in the iterative loop					
# processors	125	216	343	512	729	1000
$M_{\mathrm{d}-64}$	30.4	35.7	38.5	47.6	52.9	59.9
$M_{\mathrm{d}-\mathrm{mix}}$	30.7	36.8	40.5	49.4	53.9	61.6
$M_{\mathrm{sp}-64}$	18.0	23.0	24.6	31.1	35.3	39.6
$M_{\mathrm{sp}-\mathrm{mix}}$	19.1	23.0	25.6	32.8	37.0	41.3

Table 11: Parallel elapsed time for the solution of Problem 2 (sec).

Problem 3						
	Total solution time					
# processors	125	216	343	512	729	1000
$M_{\mathrm{d-64}}$	102.9	119.6	122.1	134.8	151.7	160.5
$M_{\mathrm{d-mix}}$	94.6	111.9	114.9	126.6	143.9	152.8
$M_{\mathrm{sp-64}}$	63.8	74.8	76.1	83.4	92.2	98.6
$M_{\mathrm{sp-mix}}$	62.9	73.1	74.9	83.4	92.6	98.4
	Time in the iterative loop					
# processors	125	216	343	512	729	1000
$M_{\mathrm{d-64}}$	54.7	71.4	73.9	86.6	103.5	112.3
$M_{\mathrm{d-mix}}$	55.5	72.8	75.8	87.4	104.7	113.6
$M_{\mathrm{sp-64}}$	30.8	41.8	43.0	50.4	59.2	65.6
$M_{\mathrm{sp-mix}}$	31.2	41.3	43.2	51.7	60.9	66.7
Problem 4						
	Total solution time					
# processors	125	216	343	512	729	1000
$M_{\mathrm{d-64}}$	115.8	139.4	150.6	171.4	181.7	198.2
$M_{\mathrm{d-mix}}$	106.3	133.7	144.9	165.3	176.2	189.6
$M_{\mathrm{sp-64}}$	71.4	83.9	93.7	103.2	107.4	118.8
$M_{\mathrm{sp-mix}}$	70.3	82.0	90.5	103.5	110.3	117.7
	Time in the iterative loop					
# processors	125	216	343	512	729	1000
$M_{\mathrm{d-64}}$	67.6	91.2	102.4	123.2	133.5	150.1
$M_{\mathrm{d-mix}}$	67.2	94.5	105.8	126.2	137.1	150.4
$M_{\mathrm{sp-64}}$	38.4	50.8	60.7	70.1	74.3	85.8
$M_{\mathrm{sp-mix}}$	38.6	50.3	58.8	71.8	78.5	86.0

Table 12: Parallel elapsed time to solve the heterogeneous and anisotropic problems (sec).

Subdomain grid size	$M_{\mathrm{d-64}}$	$M_{\mathrm{d-mix}}$	$M_{\mathrm{sp-64}}$	$M_{\mathrm{sp-mix}}$
$20 \times 20 \times 20$	35.8_{MB}	17.9_{MB}	1.8_{MB} (5%)	0.9_{MB} (5%)
$25 \times 25 \times 25$	91.2_{MB}	45.6_{MB}	2.7_{MB} (3%)	1.3_{MB} (3%)
$30 \times 30 \times 30$	194.4_{MB}	97.2_{MB}	3.8_{MB} (2%)	1.6_{MB} (2%)
$35 \times 35 \times 35$	367.2_{MB}	183.6_{MB}	7.3_{MB} (2%)	3.6_{MB} (2%)

Table 13: Local data storage for the four preconditioners.

act efficiently as the local component of a two-level scheme, we consider a simple two-level preconditioner obtained by adding an additional term to them. This term is $R_0^T A_0^{-1} R_0$, where R_0 is a projection onto a coarse space V_0 and $A_0 = R_0 A R_0^T$. Many definitions for the coarse space and the corresponding R_0 operator have been considered (and are still being developed). For our experiments, the coarse spatial component extracts one degree of freedom per subdomain as described in [5]. Its performance on Problem 2 is displayed in Table 14 and can be compared with those in Tables 9 and 11. We observe that the coarse grid correction significantly ameliorates the growth in the number of iterations when the number of subdomains is increased—on 1000 processors, almost half the number of iterations are saved. This saving of iterations does not directly translate into time savings. Each iteration becomes marginally more expensive, but the dominating part is clearly spent in the setup.

	Total solution time					
# processors	125	216	343	512	729	1000
$M_{\mathrm{d}-64}$	78.1	85.2	83.8	89.1	91.4	94.3
$M_{\mathrm{sp}-64}$	47.7	52.8	51.5	54.2	56.0	57.5
	Time for coarse setup					
# processors	125	216	343	512	729	1000
$M_{\mathrm{d}-64}$	2.04	2.05	2.20	2.38	2.76	3.68
$M_{\mathrm{sp}-64}$	2.04	2.05	2.20	2.38	2.76	3.68
	Time in the iterative loop					
# processors	125	216	343	512	729	1000
$M_{\mathrm{d}-64}$	29.9	37.0	35.6	41.0	43.2	46.1
$M_{\mathrm{sp}-64}$	14.7	19.8	18.5	21.2	22.9	24.4
	# iterations					
# processors	125	216	343	512	729	1000
$M_{\mathrm{d}-64}$	34	42	40	45	47	48
$M_{\mathrm{sp}-64}$	35	46	42	47	51	52
	Time per iteration					
# processors	125	216	343	512	729	1000
$M_{\mathrm{d}-64}$	0.88	0.88	0.89	0.91	0.92	0.96
$M_{\mathrm{sp}-64}$	0.42	0.43	0.44	0.45	0.45	0.47

Table 14: Performance of a parallel two-level preconditioner on Problem 2 using a $35 \times 35 \times 35$ subdomain mesh.

This paper describes four local preconditioners, that can be viewed as additive Schwarz preconditioners for the Schur complement in non-overlapping domain decomposition, for the solution of large three dimensional elliptic problems. These four variants are based on dense or sparse local assembled Schur complements that are computed either in 32- or 64-bit arithmetic. On most of the numerical examples, $M_{\mathrm{sp}-\mathrm{mix}}$ gives the fastest solution time and consumes the least memory. It appears to be the best trade-off from both a numerical and a computational cost perspective. Such mixed arithmetic algorithms deserve more study, especially if the trend initiated by the IBM CELL multiprocessor is followed. This processor is projected to have a peak performance near 256 Gflops in single precision and "only" 26 GFlops in double precision.

References

[1] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and*

Applications, 23(1):15–41, 2001.

[2] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2):136–156, 2006.

[3] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorenson. *LAPACK Users' Guide, 2nd Edition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1995.

[4] L. M. Carvalho, L. Giraud, and G. Meurant. Local preconditioners for two-level non-overlapping domain decomposition methods. *Numerical Linear Algebra with Applications*, 8(4):207–227, 2001.

[5] L. M. Carvalho, L. Giraud, and P. Le Tallec. Algebraic two-level preconditioners for the schur complement method. *SIAM Journal on Scientific Computing*, 22(6):1987–2005, 2001.

[6] T. F. Chan and T. P. Mathew. Domain decomposition algorithms. In *Acta Numerica 1994*, pages 61–143. Cambridge University Press, 1994.

[7] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear systems. *ACM Transactions on Mathematical Software*, 9:302–325, 1983.

[8] V. Frayssé and L. Giraud. A set of conjugate gradient routines for real and complex arithmetics. Technical Report TR/PA/00/47, CERFACS, Toulouse, France, 2000. Public domain software availabe on www.cerfacs.fr/algor/Softs.

[9] L. Giraud, A. Marrocco, and J.-C. Rioual. Iterative versus direct parallel substructuring methods in semiconductor device modelling. *Numerical Linear Algebra with Applications*, 12(1):33–53, 2005.

[10] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear system. *J. Res. Nat. Bur. Stds.*, B49:409–436, 1952.

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

[12] J. Kurzak and 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.

[13] J. Langou, J. Langou, P. Luszczek, J. Kurzak, A. Buttari, and 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.

[14] S. Lanteri. Private communication, 2006.

[15] G. Mateescu, C. J. Ribbens, and L. T. Watson. A domain decomposition preconditioner for Hermite collocation problems. *Numer. Methods Partial Differential Equations*, 19:135–151, 2003.

[16] W. D. McQuain, C. J. Ribbens, L. T. Watson, and R. C. Melville. Preconditioned iterative methods for sparse linear algebra problems arising in circuit simulation. *Comput. Math. Appl.*, 27:25–45, 1994.

[17] G. G. Pitts, C. J. Ribbens, and L. T. Watson. Domain decomposition and high order finite differences for elliptic PDEs. In R. Sincovec, D. Keyes, M. Leuze, L. Petzold, and D. Reed, editors, *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pages 727–731. SIAM, 1993.

[18] A. Quarteroni and A. Valli. *Domain decomposition methods for partial differential equations*. Numerical mathematics and scientific computation. Oxford science publications, Oxford, 1999.

[19] C. J. Ribbens, G. G. Pitts, and L. T. Watson. Parallel ELLPACK for shared memory multiprocessors. *Comput. Systems Engrg.*, 4:531–540, 1993.

[20] C. J. Ribbens, L. T. Watson, and C. deSa. Toward parallel mathematical software for elliptic partial differential equations. *TOMS*, 19:457–473, 1993.

[21] B. F. Smith, P. E. Bjørstad, and W. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996.

[22] A. Toselli and O. Widlund. *Domain Decomposition methods-Algorithms and Theory*. Springer, 2005.