Mixed-Precision Preconditioners in Parallel Domain Decomposition Solvers

Luc Giraud[1], Azzam Haidar[2], and Layne T. Watson[3]

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

1 Introduction

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 for 64-bit. One recent and significant example is the IBM CELL multiprocessor that is projected to have a peak performance near 256 Gflops in 32-bit and "only" 26 GFlops in 64-bit computation. 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. For the solution of linear systems, mixed-precision algorithms (single/double, double/quadruple) have been studied in dense and sparse linear algebra mainly in the framework of direct methods (see [5, 4, 8, 9]). For such approaches, the factorization is performed in low precision, and, for not too ill-conditioned matrices, a few steps of iterative refinement in high precision arithmetic is enough to recover a solution to full 64-bit accuracy (see [4]). For nonlinear systems, though, mixed-precision arithmetic is the essence of algorithms such as inexact Newton.

For linear iterative methods, we might wonder if such mixed-precision algorithms can be designed. The most natural way, in Krylov subspace methods, is to implement all but the preconditioning steps 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 this paper, we investigate the use of mixed-precision preconditioners in parallel 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.

The paper is organized as follows. In Section 2 we motivate using 32-bit rather than 64-bit from a speed perspective. Section 3 is devoted to a brief exposition of

the non-overlapping domain decomposition technique we consider for the parallel numerical experiments discussed in Section 4.

2 Mixed-Precision Algorithms

Counter to the 64-bit RISC trend, for some recent architectures, a 64-bit operation is more expensive than a 32-bit one. In particular, those that possess a SSE (streaming SIMD extension) execution unit can perform either two 64-bit instruction or four 32-bit instructions in the same time. This class of chip includes for instance the IBM PowerPC, the Power MAC G5, the AMD Opteron, the CELL, and the Intel Pentium. Table 1 reports the performance of basic dense kernels involved in numerical linear algebra: the _GEMV BLAS-2 matrix-vector product and the _POTRF/_POTRS LAPACK Cholesky factorization and backward/forward substitution. It can be seen that 32-bit calculation generally outperforms 64-bit. For a more exhaustive set of experiments on various computing platforms, refer to [8, 9]. The source of time reduction is not only the processing units that perform more operations per clock-cycle, but also a better usage of the complex memory hierarchy that provides ultra-fast memory transactions by reducing the stream of data block traffic across the internal bus and bringing larger blocks of computing data into the cache. This provides a speed up of two in 32-bit compared to 64-bit computation for BLAS-3 operations in most LAPACK routines.

Table 1. Elapsed time (sec) to perform BLAS-2 and LAPACK routines on various platforms when the size m of the matrices is varied.

CRAY XD1 AMD Opteron processor

n	DGEMV	SGEMV	Ratio	DPOTRF	SPOTRF	Ratio	DPOTRS	SPOTRS	Ratio
2000	0.012	0.005	2.18	0.823	0.462	1.78	0.010	0.004	2.22
7000	0.121	0.056	2.16	29.41	16.04	1.83	0.116	0.056	2.07

MAC Power PC G5 processor VMX/AltiVec extensions

n	DGEMV	SGEMV	Ratio	DPOTRF	SPOTRF	Ratio	DPOTRS	SPOTRS	Ratio
2000	0.028	0.008	3.51	0.828	0.453	1.82	0.032	0.022	1.45
7000	0.354	0.122	2.90	23.71	13.27	1.78	0.372	0.355	1.05

Another advantage of 32-bit floating point arithmetic is that data storage is reduced by half, providing an increase in data throughput. Similarly, in a distributed memory environment, the message sizes are halved.

3 Exploiting 32-bit Calculation in Domain Decomposition

Consider the following second order self-adjoint elliptic problem in the unit cube $\Omega = (0,1)^3 \subset \mathbb{R}^3$:

$$\begin{cases} -\nabla(a(x,y,z)\nabla u) = f(x,y) \text{ in } \Omega, \\ u = 0 \quad \text{on } \partial\Omega, \end{cases} \tag{1}$$

where $a(x,y,z) \in \mathbb{R}^3$ is a positive definite symmetric matrix function. We assume that the domain Ω is partitioned into N non-overlapping subdomains Ω_1,..., Ω_N and boundary $\Gamma = \cup\, \Gamma_i$, where $\Gamma_i = \partial\Omega_i \backslash \partial\Omega$. We discretize (1) by using a finite element method resulting in a symmetric positive definite linear system, $A_h u_h = f_h$. Let I denote the union of the interior points in the subdomains, and let B denote the interface points separating the subdomains. Then grouping the unknowns corresponding to I in the vector u_I and the unknowns corresponding to B in the vector u_B, we obtain the following reordering of the fine grid problem:

$$\begin{pmatrix} A_{II} & A_{IB} \\ A_{IB}^T & A_{BB} \end{pmatrix} \begin{pmatrix} u_I \\ u_B \end{pmatrix} = \begin{pmatrix} f_I \\ f_B \end{pmatrix}. \tag{2}$$

Eliminating u_I in the second block row leads to the following reduced equation for u_B:

$$S u_B = f_B - A_{IB}^T A_{II}^{-1} f_I \;\text{ with }\; S = A_{BB} - A_{IB}^T A_{II}^{-1} A_{IB}, \tag{3}$$

where S is referred to as the Schur complement matrix that is symmetric positive definite if A_h is symmetric positive definite. Let $\mathcal{R}_{\Gamma_i} : \Gamma \to \Gamma_i$ be the canonical point-wise restriction that maps full vectors defined on Γ into vectors defined on Γ_i. The Schur complement matrix (3) can be written as the sum of elementary matrices, $S = \sum_{i=1}^{N} \mathcal{R}_{\Gamma_i}^T S^{(i)} \mathcal{R}_{\Gamma_i}$, where $S^{(i)} = A_{\Gamma_i\Gamma_i} - A_{\Gamma_i I_i} A_{I_i I_i}^{-1} A_{I_i \Gamma_i}$ is a local Schur complement. We define the assembled local Schur complement, $\bar{S}^{(i)} = \mathcal{R}_{\Gamma_i} S \mathcal{R}_{\Gamma_i}^T$, that corresponds to the restriction of the Schur complement to the interface Γ_i. The assembled local Schur complement can be computed in a parallel environment from the local Schur complement via a few neighbor to neighbor communications. We define the additive Schwarz preconditioner by $M_{AS} = \sum_{i=1}^{N} \mathcal{R}_{\Gamma_i}^T \left(\bar{S}^{(i)}\right)^{-1} \mathcal{R}_{\Gamma_i}$, (see [3]).

We propose to take advantage of the 32-bit speed and memory benefit and build some part of the code in 32-bit. Our goal is to use costly 64-bit arithmetic only where necessary to preserve accuracy. We consider here the simple approach of performing all the steps of a Krylov subspace method except the preconditioning in 64-bit [10]. In this respect, it is important to note that the preconditioner only attempts to approximate the inverse of the matrix S. Since our matrices are symmetric positive definite, the Krylov subspace method of choice is conjugate gradient (CG). In our mixed-precision implementation only the preconditioned residual is computed in 32-bit. The import of this strategy 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.

4 Numerical Results

The target computer is the Terascale computer system X located at Virginia Tech's. The system Xserve is a 1,100 dual G5 processor nodes ran at 2.3GHz; each node has 4GB of main memory. The G5 cluster operates at 20.24 64-bit Teraflops peak, and the networking consists both of standard Ethernet for "non-computational" tasks, and special Mellanox Cougar InfiniBand 4x HCA networks for high-bandwidth and low-latency communications. 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 MVAPICH (MPI for InfiniBand on VAPI Layer). The code is written in Fortran 90 and compiled with the IBM compiler.

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 (see [1, 2]); that offers the possibility to compute the Schur complement matrices $S^{(i)}$ at an affordable memory and computational cost thanks to its 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 factorized using the dense linear LAPACK kernel, to construct the additive Schwarz preconditioners. 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 this section we compare the performance of a fully 64-bit with a mixed-precision implementation. For all the parallel experiments, we solve either the two-dimensional or the three-dimensional elliptic PDE defined respectively in the unit square or cube, using a uniform domain decomposition into equal sized squares or cubes. Since the goal is to study the numerical efficiency of the preconditioner, we only perform scaled experiments where the matrix size for the subdomains is kept constant (i.e., constant $\frac{H}{h}$ where H is the diameter of the subdomains, and h is the mesh size) when the number of subdomains is increased. In the table, we refer to the fully 64-bit and mixed-precision experiments as M_d and M_m, respectively.

In order to illustrate the effect on the convergence rate, we report in Table 2 the number of conjugate gradient iterations to reduce the scaled residual $\frac{\|r_k\|}{\|b\|}$ below 10^{-8}, where b is the right-hand side of the Schur complement system. We consider the smooth and not too ill-conditioned problems associated with the Poisson equation and a heterogeneous diffusion problem with coefficient jumps from 1 to 10^3 in various places of the unit square or cube. This latter example gives rise to more ill-conditioned linear systems to solve. Finally, while keeping constant the size of the subdomains, we vary their numbers and consider two different subdomain sizes.

In Table 2, we report result observed on the two dimensional model. Because only local preconditioner are considered, it can be seen that the number of iterations grows with the number of subdomains. In terms of iterations, it can be seen that for the two different problems M_m behaves closely to M_d. With this choice of the

mixed-precision, it can be expected a reduction of the global elapsed time as described below. For the three dimensional case, the behavior of the preconditioners is depicted in Table 2. The first observation, that we do not further develop, is that the preconditioner, which does not implement any coarse space component to account for the global coupling of the PDEs, does not scale too badly when the number of subdomains is increased. Its scalability with respect to the size of the subdomains is also acceptable as only a slight increase is observed when we go from subdomains with about 15,000 degrees of freedom (dof) to subdomains with about 43,000 dof. On the accuracy effect of the mixed-precision usage, it can be observed once again that it only moderately increases the number of iterations, and the increase does not depend much either on the number of subdomains or on the size of the subdomains. As expected, the growth is also slightly larger on the ill-conditioned heterogeneous problem as for the Poisson problem.

Table 2. Number of conjugate gradient iterations when the number of subdomains and the subdomain grid is varied: $2D$ and $3D$ case.

2D experiments		*Poisson Problem*				*Discontinuous Problem*			
subdomain grid		25	64	144	256	25	64	144	256
35×35	M_d	15	24	33	40	23	33	55	61
	M_m	15	26	33	41	23	34	55	62
1000×1000	M_d	20	34	45	55	37	47	76	91
	M_m	21	35	47	57	39	48	78	93

3D experiments		*Poisson Problem*				*Discontinuous Problem*			
subdomain grid		27	64	125	216	27	64	125	216
$25 \times 25 \times 25$	M_d	17	24	26	31	23	33	36	44
	M_m	19	26	28	33	24	34	39	45
$35 \times 35 \times 35$	M_d	19	26	30	33	25	35	40	47
	M_m	21	29	30	35	25	37	42	49

In Table 3 we report on three-dimensional numerical experiments related to the construction of the preconditioner for different problem sizes, varying the number of processors from 27 up to 216 (i.e., varying the decomposition of the cube from $3\times3\times3$ up to $6\times6\times6$). We depict the preconditioner setup time for both M_d and M_m. The row entitled "init" corresponds to the calculation of the local Schur complement using the MUMPS package. The construction of the local Schur complements that are involved in the matrix-vector product in the conjugate gradient algorithm is performed in both cases in 64-bit arithmetic, so the cost is the same for the two variants. The "setup precond" row is the time required to assemble and factorize, using LAPACK, the assembled local Schur complement. As might be expected, these results show that the 32-bit preconditioner setup time is significantly smaller than that for 64-bit. Note that the 32-bit arithmetic cut in half the time for assembling the local Schur matrix, due to halving the amount of communication. Also the $\mathcal{O}(n^3)$ floating-point operations of the LL^T factorization $[S/D]POTRF$ are about a factor of 1.8 faster in 32-bit.

Table 3. Parallel performance for various steps of the preconditioned conjugate gradient implementations ($35 \times 35 \times 35$ subdomain grid).

# proc	27		64		125		216	
	M_d	M_m	M_d	M_m	M_d	M_m	M_d	M_m
init	26.8	26.8	26.8	26.8	26.8	26.8	26.8	26.8
setup precond	21.2	12.3	21.2	12.3	21.3	12.3	21.4	12.4
time per iter	0.73	0.68	0.73	0.69	0.76	0.71	0.76	0.72
total	66.3	56.1	73.5	64.6	78.5	68.9	83.9	74.5
# iter	25	25	35	37	40	42	47	49

From a memory viewpoint, using M_m saves 150 MB of memory per processor for the example with about 43,000 dof per subdomain. From a computational perspective, memory and CPU time, the saving is clear. It can be seen that the time per iteration is almost constant and does not depend much on the number of processors for both preconditioners. In terms of the overall computing time, the row entitled "total" in Table 3 displays the overall elapsed time to solve the heterogeneous diffusion problem with 43,000 dof per subdomain when the number of domains is varied. These results show that on the most difficult problem the time saved by the use of mixed-precision arithmetic still compensates for a slight increase in the number of iterations, and that M_m outperforms M_d.

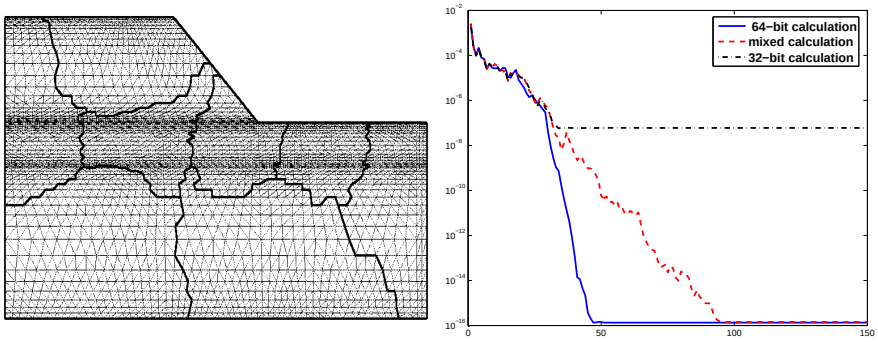

Fig. 1. Convergence history of $\|r_k\|/\|b\|$ (right) on a 15 000 dof problem in mixed finite element device modeling simulation on a mosfet (left).

In Figure 1, we report the convergence history of PCG using entire 64/32-bits and mixed arithmetic calculation on an unstructured 2D problem arising from mixed finite element discretization in device modeling simulations (150 000 dof and 16 subdomains) [7]. This real life example exhibits similar numerical behavior as the ones observed on the academic examples of Table 2. Namely, the pure 32-bit calculation has a limiting accuracy much larger than the mixed and the full 64-bit computation.

5 Concluding Remarks

In a linear iterative parallel domain decomposition solver, the use of 32-bit arithmetic was limited to the preconditioning step. The main advantage of using mixed-precision is that it reduces the data storage, the computational time, and the communication overhead while only marginally degrading the convergence rate without preventing to reach similar accuracy as full 64-bit calculation. This work is just a first study of mixed arithmetic implementation, and other variants will be considered in future work. While the current work is purely experimental, some theoretical studies deserve to be undertaken following possibly some techniques presented in [11]. Finally, we mention that the one-level preconditioner presented here can be considered in a two-level scheme, we refer to [6] for more details on that aspect.

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 J. Matrix Anal. Appl.*, 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 Comput.*, 32(2):136–156, 2006.

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

[4] J. Demmel, Y. Hida, W. Kahan, S.X. Li, S. Mukherjee, and E.J. Riedy. Error bounds from extra precise iterative refinement. Technical Report UCB/CSD-04-1344, LBNL-56965, University of California in Berkeley, 2006. Short version appeared in ACM Trans. Math. Software, vol. 32, no. 2, pp 325-351, June 2006.

[5] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, S. X. Li, and J. W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Anal. Appl.*, 20(3):720–755, 1999.

[6] L. Giraud, A. Haidar, and L.T. Watson. Parallel scalability study of three dimensional additive Schwarz preconditioners in non-overlapping domain decomposition. Technical Report TR/PA/07/05, CERFACS, Toulouse, France, 2007.

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

[8] 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.

[9] 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.

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

[11] G. Meurant. *The Lanczos and Conjugate Gradient Algorithms: From Theory to Finite Precision Computations.* Software, Environments, and Tools 19. SIAM, 2006.