Using Mixed Precision for Sparse Matrix Computations to Enhance the Performance while Achieving 64-bit Accuracy

ALFREDO BUTTARI
ENS Lyon
JACK DONGARRA
University of Tennessee Knoxville
Oak Ridge National Laboratory
Univeristy of Manchester
JAKUB KURZAK
University of Tennessee Knoxville
PIOTR LUSZCZEK
The MathWorks
and
STANIMIR TOMOV
University of Tennessee Knoxville

By using a combination of 32-bit and 64-bit floating point arithmetic, the performance of many sparse linear algebra algorithms can be significantly enhanced while maintaining the 64-bit accuracy of the resulting solution. These ideas can be applied to sparse multifrontal and supernodal direct techniques and sparse iterative techniques such as Krylov subspace methods. The approach presented here can apply not only to conventional processors but also to exotic technologies such as Field Programmable Gate Arrays (FPGA), Graphical Processing Units (GPU), and the Cell BE processor.

This work was supported in part by the National Science Foundation and Department of Energy.
Authors' address: University of Tennessee, Knoxville, TN 37996; email:dongarra@cs.utk.edu.

© 2008 ACM 0098-3500/2008/07-ART17 $5.00 DOI: 10.1145/1377596.1377597. http://doi.acm.org/10.1145/1377596.1377597.

Categories and Subject Descriptors: B.2.4 [**Arithmetic and Logic Structures**]: High-Speed Arithmetic; F.2.1 [**Analysis of Algorithms and Problem Complexity**]: Numerical Algorithms and Problems—*Computations on matrices*; G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra—*Error analysis*; *linear systems*; G.4 [**Mathematical Software**]: —*Algorithms design and analysis*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Linear systems, iterative refinement, floating point, precision

ACM Reference Format:
Buttari, A., Dongarra, J., Kurzak, J., Luszczek, P., and Tomov, S. 2008. Using mixed precision for sparse matrix computations to enhance the performance while achieving 64-bit accuracy. ACM Trans. Math. Softw. 34, 4, Article 17 (July 2008), 22 pages. DOI = 10.1145/1377596.1377597. http://doi.acm.org/10.1145/1377596.1377597.

1. INTRODUCTION

Benchmarking analysis and architectural descriptions reveal that on many commodity processors the performance of 32-bit floating point arithmetic (single precision computations) may be significantly higher than 64-bit floating point arithmetic (double precision computations). This is due to a number of factors. First, many processors have vector instructions such as the SSE2 instruction set on the Intel IA-32 and IA-64 and AMD Opteron family of processors or the AltiVec unit on the IBM PowerPC architecture. In the SSE2 case, a vector unit can complete four single precision operations every clock cycle but can complete only two in double precision. (For the AltiVec, single precision can complete eight floating point operations per cycle as opposed to four floating point operations in double precision.) Another reason lies in the fact that single precision data can be moved at a faster rate through the memory hierarchy as a result of a reduced amount of data to be transferred. Finally, the fact that single precision data occupies less memory than double precision data means that more single precision values can be held in cache than the double precision counterpart, which results in a lower rate of cache misses (the same reasoning can be applied to translation look-aside buffers (TLBs)). A combination of these factors can lead to significant enhancements in performance for sparse matrix computations as we will show in the following sections. A remarkable example is the IBM Cell BE processor where the peak performance using single precision floating point arithmetic is more than an order of magnitude higher than that of double precision (204.8GFlop/s vs 14.6GFlop/s for the 3.2GHz version of the chip). The performance of some linear algebra operations can be improved based on the consideration that the most computationally expensive tasks can be performed by exploiting single precision operations and only resorting to double precision at critical stages while attempting to provide the full double precision accuracy. This technique is supported by the well-known theory of iterative refinement [Demmel 1997; Higham 2002], which has been successfully applied to the solution of dense linear systems [Langou et al. 2006]. This work is an extension of the work by Langou et al. [2006] for the case of sparse linear systems, covering both direct and iterative solvers.

Algorithm 1 PCG (b, x_o, E_{tol}, ...)

```
1: $r_0 = b - Ax_0$
2: for $i = 1, 2, \ldots$ do
3:     $z_{i-1} = Mr_{i-1}$
4:     $\rho_{i-1} = r_{i-1}^T z_{i-1}$
5:     if $i = 1$ then
6:         $d_1 = z_0$
7:     else
8:         $\beta = \rho_{i-1}/\rho_{i-2}$
9:         $d_i = z_{i-1} + \beta d_{i-1}$
10:    end if
11:    $q_i = Ad_i$
12:    $\alpha = \rho_{i-1}/{d_i}^T q_i$
13:    $x_i = x_{i-1} + \alpha_i d_i$
14:    $r_i = r_{i-1} - \alpha_i q_i$
15:    check convergence and exit if done
16: end for
```

2. SPARSE DIRECT AND ITERATIVE SOLVERS

Most sparse direct methods for solving linear systems of equations are variants of either multifrontal [Duff and Reid 1983] or supernodal [Ashcraft et al. 1987] factorization approaches. There are a number of freely available packages that implement these methods. We have chosen for our tests the software package MUMPS [Amestoy et al. 2000; 2001; 2006] as the representative of the multifrontal approach and SuperLU [Li and Demmel 2003; Demmel et al. 1999a; 1999b; Li 1996] for the supernodal approach. Our main reason for selecting these two software packages is that they are implemented in both single and double precision, which is not the case for other freely available solvers such as UMFPACK [Duff 1997; Davis 1999; 2004].

Fill-ins, and the associated memory requirements, are inherent for direct sparse methods. And although there are various reordering techniques designed to minimize the amount of these fill-ins, for problems of increasing size, there is a point where the memory requirements become prohibitively high and direct sparse methods are no longer feasible. Iterative methods are a remedy because only a few working vectors and the primary data are required [Barrett et al. 1994; Saad 2003].

Two popular iterative solvers on which we will illustrate the techniques addressed in this article are the Conjugate Gradient (CG) method (for symmetric and positive definite matrices) and the Generalized Minimal Residual (GMRES) method for nonsymmetric matrices [Saad and Schultz 1986]. The preconditioned versions of the two algorithms are given correspondingly in Algorithms 1 and 2 with the descriptions that follow the standard notation [Barrett et al. 1994; Saad 2003].

The preconditioners, denoted in both cases as M, are operators intended to improve the robustness and the efficiency of the iterative algorithms. In particular, we will use *left* preconditioning, where instead of

$$Ax = b,$$

Algorithm 2 GMRES (b, x_o, E_{tol}, m, ...)

```
1: for $i = 0, 1, ...$ do
2:     $r = b - Ax_i$
3:     $\beta = h_{1,0} = ||r||_2$
4:     check convergence and exit if done
5:     for $k = 1, 2, ..., m$ do
6:         $v_k = r \,/\, h_{k,k-1}$
7:         $r = A\ M\ v_k$
8:         for $j = 1$ to $k$ do
9:             $h_{j,k} = r^T v_j$
10:            $r = r - h_{j,k}\ v_j$
11:        end for
12:        $h_{k+1,k} = ||r||_2$
13:        if $h_{k+1,k}$ is small enough then break
14:    end for
15:    // Define $V_k = [v_1, \ldots, v_k]$, $H_k = \{h_{i,j}\}_{1 \le i \le k+1, 1 \le j \le k}$
16:    Find $w_k$, a $k$-dim column vector, that minimizes $||b - A(x_i + M\ V_k\ w_k)||_2$
17:    // note: or equivalently, find $w_k$ that minimizes $||\beta e_1 - H_k\ w_k||_2$
18:    $x_{i+1} = x_i + M\ V_k\ w_k$
19: end for
```

we solve $MAx = Mb$, and *right* preconditioning, where the problem is transformed to $AMu = b$, $x = Mu$. Intuitively, to serve its purpose, M needs to be easy to compute, apply, and store, as well as to approximate A^{-1}.

The basic idea of our approach is to use faster but lower precision computations whenever possible. As we show in the rest of the article, this idea can be used to design the preconditioner M that has the two requirements mentioned previously. And since our basic idea can be exploited (in iterative solvers) through proper preconditioning, the applicability of the approach is far-reaching and not limited to either the preconditioners or the solvers, each of which are used to demonstrate our idea.

3. MIXED-PRECISION ITERATIVE REFINEMENT

The iterative refinement technique is a well-known method that has been extensively studied and applied in the past. A fully detailed description of this method can be found elsewhere [Demmel 1997; Higham 2002; Stewart 2001; Wilkinson 1965; Björck 1990]. The iterative refinement approach has been used in the past to improve the accuracy of linear systems' solutions and it is shown in Algorithm 3.

Once the system is solved at step 1, the solution can be refined through an iterative procedure where, at each iteration, the residual is computed based on the solution at the previous iteration (step 4), a correction is computed as in step 5, and finally this correction is applied as in step 6. While the common usage of iterative refinement [Higham 2002; Anderson et al 1999] consists of performing all the arithmetic operations with the same precision (either single or double), we have investigated the application of mixed-precision iterative refinement where the most expensive steps, 1 and 5, are performed in single precision, and steps 4 and 6 are performed in double precision. Work

Algorithm 3 The iterative refinement method for the solution of linear systems

```
1: $x_0 \leftarrow A^{-1}b$
2: $k = 1$
3: **for** $k = 1, 2, \ldots$ **do**
4:     $r_k \leftarrow b - Ax_{k-1}$
5:     $z_k \leftarrow A^{-1}r_k$
6:     $x_k \leftarrow x_{k-1} + z_k$
7:     $k \leftarrow k + 1$
8:     check convergence and exit if done
9: **end for**
```

by others [Strzodka and Göddeke 2006a, 2006b; Göddeke et al. 2005] that is somehow related to this main idea does not use exactly our approach. The error analysis for the mixed-precision iterative refinement, explained in Moler [1967], Forsythe and Moler [1967], Golub and Loan [1989], shows that by using this approach, it is possible to achieve the same accuracy as if the system was solved in full double precision arithmetic provided that the matrix is not too badly conditioned. From a performance point of view, the potential of this method lies in the fact that the most computationally expensive steps, 1 and 5, can be performed very fast in single precision arithmetic, while the only tasks that require double precision accuracy are steps 4 and 6 whose cost can be considered much lower.

We will refer in the following to single precision (SP) as 32-bit and to double precision (DP) as 64-bit floating point arithmetic and also lower and higher precision arithmetic will be correspondingly associated with SP and DP.

3.1 Mixed-Precision Iterative Refinement for Sparse Direct Solvers

Using the MUMPS package for solving systems of linear equations can be described in three distinct steps.

(1) System Analysis. In this phase the system sparsity structure is analyzed in order to estimate the fill-in which provides an estimate of the memory requirement that will be allocated in the following steps. Also, pivoting is performed based on the structure of $A + A^T$, ignoring numerical values. Only integer operations are performed at this step.
(2) Matrix Factorization. In this phase, the $PQAQ^T = LU$ factorization is performed, where P is a row permutation matrix and Q is the reordering matrix from step 1. This is the computationally most expensive step of the system solution.
(3) System Solution. The system is solved in three steps: $Ly = PQb$, $Uz = y$, and $x = Q^Tz$.

Once steps 1 and 2 are performed, each iteration of the refinement loop needs only to perform the system solution (i.e., step 3) whose cost can be up to two orders of magnitude lower than the cost of the system factorization. The implementation of the mixed-precision iterative refinement method with the MUMPS package is shown in Algorithm 4.

Algorithm 4 Mixed-precision Iterative Refinement with the MUMPS package

```
1:  system analysis
2:  $LU \leftarrow PQAQ^T$                  (SP)
3:  solve $Ly = PQb$                    (SP)
4:  solve $Uz = y$                      (SP)
5:  $x_0 = Q^T z$                       (SP)
6:  for $k = 1, 2, \ldots$ do:
7:      $r_k \leftarrow b - Ax_{k-1}$       (DP)
8:      solve $Ly = PQr_k$              (SP)
9:      solve $Uz = y$                  (SP)
10:     $z_k = Q^T z$                   (SP)
11:     $x_k \leftarrow x_{k-1} + z_k$      (DP)
12:     check convergence and exit if done
13: end for
```

At the end of each line of the algorithm, we indicate the precision used to perform the corresponding operation. Based on backward stability analysis, we consider that the solution x is of double precision quality when

$$\|b - Ax\|_2 \leq \|x\|_2 \cdot \|A\|_{fro} \cdot \epsilon_d,$$

where $\|\cdot\|_{fro}$ is the Frobenius norm and ϵ_d is the system precision for 64-bit arithmetics. This provides a stopping criterion. If some maximum number of iterations is reached, then the algorithm should signal failure to converge. All the control parameters for the MUMPS solver have been set to their default values, which means that the matrix scaling and permuting and pivoting order strategies are determined at runtime based on the matrix properties.

3.2 Mixed-Precision Iterative Refinement for Sparse Iterative Solvers

The general framework of mixed-precision iterative refinement given at the beginning of this section can be easily extended to sparse iterative solvers. Indeed, the mixed-precision iterative refinement can be interpreted as a preconditioned Richardson iteration with the preconditioner computed and applied (during the iterations) in single precision [Turner and Walker 1992]. This interpretation can be further extended from Richardson to other preconditioned iterative methods. And, in general, as long as the iterative method at hand is backward stable and converges, one can apply similar reasoning as in Langou et al. [2006] to show that the solution obtained would be accurate in higher precision. The feasibility of using mixed precision in computing and/or applying a preconditioner depends first on whether there is a potential to introduce speedups in the computation and second on how the method's robustness would change.

A simple example of a mixed precision preconditioner is when the storage of data used in the preconditioner is in single precision. The potential benefit here is that accessing SP data is faster (*vs* DP data) due to less memory traffic. The success of this approach regarding speed depends on what percentage of the overall computation is spent on the preconditioner (and in particular accessing preconditioner SP data). For example, a simple diagonal preconditioner may benefit little from it, while a domain decomposition-based [Quarteroni and Valli 1999] block diagonal preconditioner or a multigrid V-cycle [Hackbusch 1985] may benefit significantly. Also, multigrid-based solvers may benefit both in speed (as the bulk of the computation is in their V/W-cycles) and memory requirements. An example of successful application of this type of approach in CFD [Gropp et al. 2000; 2001] was done in a PETSc solver [Balay et al. 2001], which was accelerated with a Schwartz preconditioner using block-incomplete factorizations over the separate subdomains that are stored in single precision. Regarding robustness, there are various algorithmic issues to consider, including ways to automatically determine limitations of the approach at runtime. This brings up another possible idea to explore, which is the use of lower precision arithmetic for only parts of the preconditioner. Examples here may come from adaptive methods that automatically locate the singularities of the sought solution, and hence the corresponding parts of the matrix responsible for resolving them. This information may be used in combination with the solver and preconditioner (e.g., hierarchical multigrid) to achieve both speedup and robustness of the method.

Another interesting example is when not only the higher precision storage, but also the higher precision arithmetic are replaced with lower precision. This is the case that would allow one to apply the technique not only to conventional processors, but also to FPGAs, GPUs, the Cell BE processor, etc.

The focus of the current work is to enable the efficient use of lower precision arithmetic to sparse iterative methods in general when no preconditioner, or when just a simple and computationally inexpensive (relative to the rest of the computation) preconditioner, is available. The idea of accomplishing this is to use the preconditioned version of the iterative method at hand and also replace the preconditioner M by an iterative method but one implemented in reduced precision arithmetic. Thus, by controlling the accuracy of this iterative inner solver, more computations can be done in reduced precision and less work is needed in the full precision arithmetic.

The robustness of variations of this *nesting* of iterative methods, also known in the literature as *inner-outer* iteration, has been studied before, both theoretically and computationally [Golub and Ye 2000; Saad 1991; Simoncini and Szyld 2002b; Axelsson and Vassilevski 1991; Notay 2000; Vuik 1995; van den Eshof et al. 2003]. The general appeal of these methods is that computational speedup is possible when the inner solver uses an approximation to the original matrix that is also faster to apply. Moreover, even if no faster matrix-vector product is available, speedup can often be observed due to improved convergence (e.g., see restarted GMRES *vs* GMRES-FGMRES [Simoncini and Szyld 2002b] and Section 4.3). To our knowledge, using mixed-precision for performance enhancement has not been done in the framework suggested in this article. In the sections that follow, we show a way to do it for CG and GMRES.

3.2.1 *CG-Based Inner-Outer Iteration Methods.* We suggest the PCG-PCG inner-outer Algorithm 5. It is given as a modification to the reference PCG Algorithm 1, and therefore only the lines that change are written out (i.e., line 3).

Algorithm 5 PCG_PCG (b , x_o, E_{tol}, . . .)

```
    . . .
3:  PCG_single ( $r_{i-1}$, $z_{i-1}$, $NumIters$, . . . )
    . . .
```

Algorithm 6 PCG_single (b , x, $NumIters$, . . .)

```
1:   $r_0 = b$ ; $x_0 = 0$
2:   for $i = 1$ to $NumIters$ do
         . . .
15:      [check SP convergence and exit if done]
16:  end for
```

The inner PCG, denoted by PCG_single, is in SP and is described by Algorithm 7 again as a modification of the reference PCG. The preconditioner available for the reference PCG is used in SP in the inner PCG. Note that our initial guess in the inner loop is always taken to be 0, and we perform a fixed number of iterations (step 15 is in brackets since practically we want to avoid exiting due to a small residual unless it is on the order of the machine's single precision). The number of inner iterations is fixed and depends on the particular problem at hand. We take it to be the number of iterations it takes `PCG_single` to do a fixed (e.g., 0.3) relative reduction for the initial residual r_0. Work on criteria to compute the (variable) number of inner iterations guaranteeing convergence can be found in Simoncini and Szyld [2002a].

If a preconditioner is not available, we can similarly define a CG-PCG algorithm where the inner loop is just a CG in SP. Furthermore, other iterative solvers can be used for the inner loop as long as they result in symmetric and positive definite (SPD) operators. For example, stationary methods like Jacobi, Gauss-Seidel (combination of one backward and one forward to result in SPD operator), and SSOR can be used. Note that with these methods, a constant number of iterations and initial guess 0 result in a constant preconditioner. The use of a Krylov space method in the inner iteration, as in the currently considered algorithm, results in a nonconstant preconditioner. Although there is convergence theory for these cases [Simoncini and Szyld 2002b], how to set the stopping criteria still remains to be resolved as do variations in the algorithms, etc. [Golub and Ye 2000; Notay 2000] in order to obtain optimal results. For example, Golub and Ye [2000] consider the inexact PCG (β is taken as $\frac{(r_{i-1}-r_{i-2})\cdot z_{i-1}}{r_{i-2}\cdot z_{i-2}}$), which allows certain local orthogonality relations to be preserved from the standard PCG, which provides grounds for theoretically studied aspects of the algorithm. We tried this approach as well and, although our numerical results were similar to Golub and Ye [2000], overall the algorithm described here gave better results. In general, nonconstant preconditioning deteriorates the CG convergence, often resulting in convergence that is characteristic of the Steepest Descent algorithm. Still, shifting the computational load to the inner PCG reduces this effect and gives convergence that is comparable to the convergence of a reference PCG algorithm. We note that nonconstant preconditioning can be better accommodated in GMRES (see the

Algorithm 7 GMRES_FGMRES (b, x_o, E_{tol}, m_{in}, m_{out}, ...)

```
    ...
 5: for $k = 1$ to $m_{out}$ do
        ...
 7:     GMRES_single( $r_k$, $z_k$, 1, $m_{in}$, ... )
        $r = Az_k$
        ...
15:     // Define $Z_k = [z_1, \ldots, z_k]$, $H_k = \{h_{i,j}\}_{1 \le i \le k+1, 1 \le j \le k}$
16:     Find $w_k$, a $k$-dim column vector, that minimizes $||b - A(x_i + Z_k w_k)||_2$
        ...
18:     $x_{i+1} = x_i + Z_k w_k$
19: end for
```

Algorithm 8 GMRES_single (b, x, *NumIters*, m, ...)

```
 1: $x_0 = 0$
    for $i = 1$ to $NumIters$ do
        ...
 4:     [check SP convergence and exit if done]
        ...
```

next section). See also Simoncini and Szyld [2005] for a way to interpret and theoretically study the effects of nonconstant preconditioning.

3.2.2 *GMRES-Based Inner-Outer Iteration Methods.* For our outer loop, we take the flexible GMRES (FGMRES [Saad 1991, 2003]) which is a minor modification to Algorithm 2 meant to accommodate nonconstant preconditioners. The price is m additional storage vectors. Algorithm 7 gives our inner-outer GMRES-FGMRES (the additional vectors are introduced at line 7: $z_k = Mv_k$ where M is replaced with the `GMRES_single` solver in Algorithm 8).

As with PCG-PCG, the algorithm is given as a modification to the reference GMRES(m) algorithm from Algorithm 2 and, therefore, only the lines that change are written out. The inner GMRES_single is in SP. The preconditioner available for the reference GMRES is used in SP in the inner GMRES_single. Note that again our initial guess in the inner loop is always taken to be 0, and we perform a fixed number of cycles/restarts (in this case, just 1; step 4 is in brackets since we want to avoid exiting due to a small residual unless it is of the order of the machine's single precision).

The potential benefits of FGMRES compared to GMRES are becoming better understood [Simoncini and Szyld 2002b]. Numerical experiments, as we also show, confirm cases of improvements in speed, robustness, and sometime memory requirements for these methods. For example, we show a maximum speedup of close to $12\times$ on a problem of size $602,091$ (see Section 4). The memory requirements for the method are the matrix in compressed row storage (CRS) format [Barrett et al. 1994], the nonzero matrix coefficients in SP, twice the outer restart size number of vectors in DP, and inner restart size number of vectors in SP.

Table I. Properties of a Subset of the Tested Matrices (The condition number estimates were computed on the Opteron 246 architecture by means of MUMPS subroutines.)

Matrix No.	Matrix name	Size	Nonzeros	Cond. num. est.
1	G64	7000	82918	$O(10^4)$
2	Si10H16	17077	875923	$O(10^3)$
3	c-71	76638	859554	$O(10)$
4	cage11	39082	559722	$O(1)$
5	dawson5	51537	1010777	$O(10^4)$
6	nasasrb	54870	2677324	$O(10^7)$
7	poisson3Db	85623	2374949	$O(10^3)$
8	rma10	46835	2374001	$O(10)$
9	s3rmt3m1	5489	112505	$O(10^9)$
10	wang4	26068	177196	$O(10^3)$

Table II. Properties of the Matrices Used With the Iterative Sparse Solvers

Level	Size	Nonzeros	Cond. num. est.
1	11,142	442,225	$O(10^3)$
2	25,980	1,061,542	$O(10^4)$
3	79,275	3,374,736	$O(10^4)$
4	230,793	9,991,028	$O(10^5)$
5	602,091	26,411,323	$O(10^5)$

The Generalized Conjugate Residuals (GCR) method [Vuik 1995; van der Vorst and Vuik 1994] is comparable to the FGMRES and can replace it successfully as the outer iterative solver.

4. NUMERICAL EXPERIMENTS

4.1 The Test Collection for Mixed-Precision Sparse-Direct and Iterative Solvers

We tested our implementation of a mixed-precision sparse-direct solver on a test suite of 41 matrices taken from the University of Florida's Sparse Matrix Collection (http://cise.ufl.edu/research/sparse/matrices/). The matrices were selected randomly from the collection since there is no information available about their condition number. A smaller subset of ten matrices (described in Table I) will be discussed in this document for readability reasons. The matrices in this smaller subset were chosen in order to provide examples of all the significant features observed on the test suite. The results for all of the 41 matrices in the test suite can be found in Buttari et al. [2006].

For the iterative sparse methods, we used a set of five matrices of increasing size. More precisely, the matrices were produced by an adaptive finite element method discretization of a 3D linear elasticity problem of the form

$$\mu \,\triangle u + (\lambda + \mu)\nabla \, Div \; u = f,$$

where u is displacement, f is a given force, and μ and λ are constants [Gurtin 1981]. The discretization is on a tetrahedral mesh, using piecewise linear finite elements (see Table II).

Table III. Hardware Characteristics of the Architectures Used to Measure the Experimental Results

	Clock freq.	Vector Units	Memory
AMD Opteron 246	2 GHz	SSE, SSE2 3DNow!	2 GB
Sun UltraSparc-IIe	502 MHz	none	512 MB
Intel PIII Copp.	900 MHz	SSE, MMX	512 MB
PowerPC 970	2.5 GHz	AltiVec	2 GB
Intel Woodcrest	3 GHz	SSE, SSE2 MMX	4 GB
Intel XEON	2.4 GHz	SSE, SSE2 MMX	2 GB
Intel Centrino Duo	2.5 GHz	SSE, SSE2 MMX	4 GB

Table IV. Software Characteristics of the Architectures Used to Measure the Experimental Results

	Compiler	Compiler flags	BLAS
AMD Opteron 246	Intel v9.1	-O3	Goto
Sun UltraSparc-IIe	Sun v9.0	-fast -xchip=ultra2e -xarch=v8plusa	Sunperf
Intel PIII Copp.	Intel v9.0	-O3	Goto
PowerPC 970	IBM v8.1	-O3 -qalign=4k	Goto
Intel Woodcrest	Intel v9.1	-O3	Goto
Intel XEON	Intel v8.0	-O3	Goto
Intel Centrino Duo	Intel v9.0	-O3	Goto

4.2 Performance Characteristics of the Tested Hardware Platforms

The implementation of the mixed-precision algorithm for sparse direct methods presented in Section 3.1 has been tested on the architectures reported, along with their main characteristics, in Tables III and IV. All the tests were done with sequential code, thus only one execution unit was used even on those processors that present multiple cores. All of these architectures have vector units except the Sun UltraSparc-IIe one; this architecture has been included for the purpose of showing that even in the case where the same number of single and double precision operations can be completed in one clock cycle, significant benefits can still be achieved thanks to the reduced memory traffic and higher cache hit rate provided by single precision arithmetic.

The implementation of the mixed-precision algorithms for sparse iterative solvers described in Section 3.2 was only tested on the Intel Woodcrest architecture. The application for the numerical tests on the mixed-precision iterative method for sparse direct solvers was coded in Fortran 90, and the application

Table V. Performance Comparison Between Single and Double Precision Arithmetic for Matrix-Matrix and Matrix-Vector Product Operations on Square Matrices

	Size	SGEMM/ DGEMM	Size	SGEMV/ DGEMV
AMD Opteron 246	3000	2.00	5000	1.70
Sun UltraSparc-IIe	3000	1.64	5000	1.66
Intel PIII Copp.	3000	2.03	5000	2.09
PowerPC 970	3000	2.04	5000	1.44
Intel Woodcrest	3000	1.81	5000	2.18
Intel XEON	3000	2.04	5000	1.82
Intel Centrino Duo	3000	2.71	5000	2.21

Table VI. Performance Comparison Between Single and Double Precision Arithmetic on a Fixed Number of Iterations of Conjugate Gradient (100 iterations) and Generalized Minimal RESidual (2 cycles/restarts of GMRES(20)) Methods Both With and Without Diagonal Scaling Preconditioner. The runs were performed on Intel Woodcrest (3 GHz with a 1333 MHz front side bus.)

	SCG/DCG		SGMRES/DGMRES	
Size	no prec.	prec.	no prec.	prec.
11,142	2.24	2.11	2.04	1.98
25,980	1.49	1.50	1.52	1.51
79,275	1.57	1.50	1.58	1.50
230,793	1.73	1.72	1.74	1.69
602,091	1.50	1.50	1.67	1.63

for the numerical tests on the inner-outer iteration method for sparse iterative solvers was coded in C.

Table V shows the difference in performance between the single and double precision implementation for the two dense BLAS operations matrix-matrix product (_GEMM) and matrix-vector product (_GEMV). They are the two principal computational kernels of sparse direct solvers: sparse data structures get rearranged to fit the storage requirements of these kernels and thus benefit from their high performance rates (as opposed to the performance of direct operation on sparse data structures). In particular, column three (column five) reports the ratio between the performance of SGEMM (SGEMV) and DGEMM (DGEMV). The BLAS libraries used are capable of exploiting the vector units where available, and thus the speedups shown in Table V are due to a combination of higher number of floating point operations completed at each clock cycle, reduced memory traffic on the bus, and higher cache hit rate.

Table VI shows the difference in performance for the single and double precision implementation of the two sparse iterative solvers, Conjugate Gradient and Generalized Minimal Residual. Columns two and three report the ratio between the performance of single and double precision CG for a fixed number (100) of iterations in both preconditioned and unpreconditioned cases. Columns four and five report the same information for the GMRES(20) method where the number of cycles/restarts has been fixed to two. Since the sparse matrix kernels involved in these computations were not vectorized, the speedup shown in Table VI is exclusively due to reduced data traffic on the bus and a higher cache hit rate.

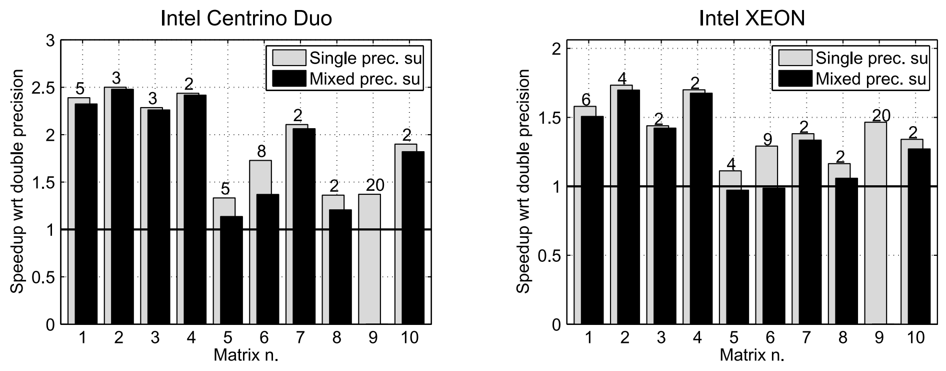


Fig. 1. Experimental results for the MUMPS solvers. The light-shaded bars report the ratio between the performance of the single precision solver and the double precision one. The dark-shaded bars report the ratio between the mixed-precision solver and the double precision one. The absence of the dark bar for a matrix means that convergence was not achieved within the maximum number of iterations (20). The number of iterations required to converge is given by the number above the bars. Left: Intel Centrino Duo. Right: Intel XEON.

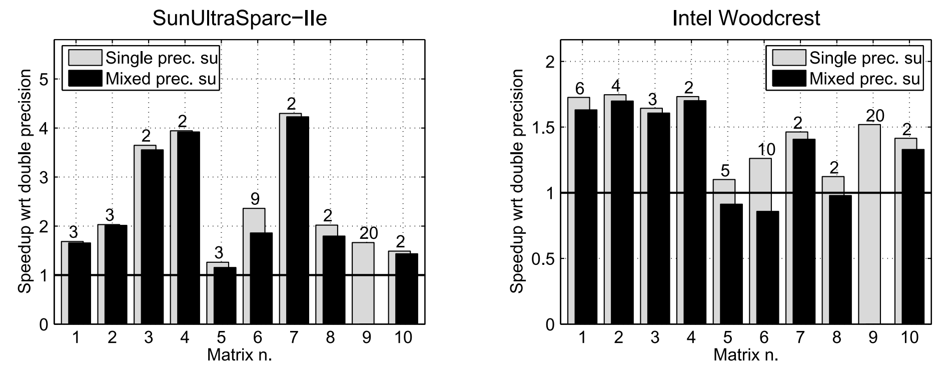
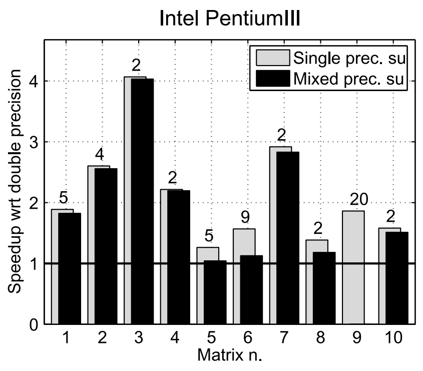


Fig. 2. Experimental results for the MUMPS solvers. Left: Sun UltraSparc-IIe. Right: Intel Woodcrest.

4.3 Experimental results

Figures 1, 2, 3, 4 show that the MUMPS single precision solver is always faster than the equivalent double precision one (i.e., the light bars are always above the thick horizontal line that corresponds to one). This is mainly due to both reduced data movement and better exploitation of vector arithmetic (via SSE2 or AltiVec where present) since multifrontal methods have the ability to do matrix-matrix products.

The results presented also show that mixed-precision iterative refinement is capable of providing considerable speedups with respect to the full double precision solver, while providing the same (in many cases better) accuracy. To run these experiments, a convergence criterion different from the one discussed in Section 3.1 was used. To make the comparison fair, in fact, the iterative refinement was stopped whenever the residual norm was the same as that



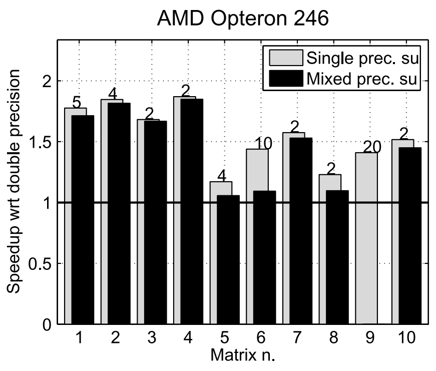


Fig. 3. Experimental results for the MUMPS solvers. Left: Intel Pentium III Coppermine. Right: AMD Opteron 246.

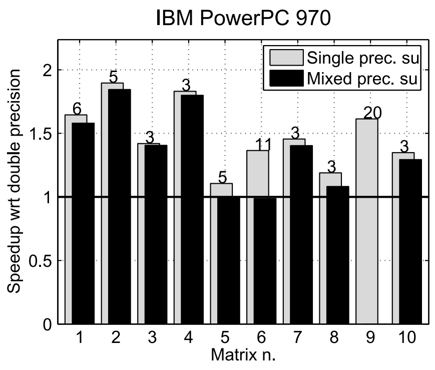


Fig. 4. Experimental results for the MUMPS solvers. PowerPC 970.

computed for the double precision solver. In addition, the iterative refinement was stopped if convergence was not achieved within a maximum number of iterations equal to 20; a failure is reported when this limit is reached.

The performance of the mixed precision solver is usually very close to that of the single precision solver mainly because the cost of each iteration is often negligible compared to the cost of the matrix factorization. It is important to note that, in some cases, the speedups reach very high values (more than 4.0 faster for the Poisson3Db matrix on the Sun UltraSparc-IIe architecture). This is due to the fact that the memory requirements are too high to accomodate the fill-in generated in the factorization phase as the main memory available on the system is lower for this machine. It forces the virtual memory system to swap pages to disk resulting in a considerable loss of performance. Since double precision data is twice as large as single precision, disk swapping may affect only the double precision solver and not the single precision one. It can be noted that disk swapping issues did not affect the results measured on those machines that are equipped with more memory, while it is usual on the Intel Pentium III and the Sun UltraSparc-IIe architectures that only have 512MB of memory. Finally, the data presented in Figures 1- 4 show that, for some cases,

Table VII. Accuracy of the Double Precision Solver and Mixed-precision Iterative Solver on the Intel Woodcrest Architecture. (The last column reports number of iterations performed by the mixed-precision method to achieve the reported accuracy.)

Matrix No.	Matrix name	DP residual	MP residual	# it.
1	G64	1.61e-10	2.00e-11	6
2	Si10H16	3.03e-12	1.93e-14	4
3	c-71	1.00e-14	9.78e-15	3
4	cage11	3.94e-16	1.04e-16	2
5	dawson5	5.88e-11	4.40e-12	5
6	nasasrb	1.08e-10	8.12e-11	10
7	poisson3Db	1.56e-14	1.04e-14	2
8	rma10	1.03e-13	6.85e-14	2
9	s3rmt3m1	3.19e-8	5.76e+2	20
10	wang4	9.65e-15	6.13e-15	2

the mixed-precision iterative refinement solver does not provide a speedup. This can be mainly attributed to three causes (or any combination of them).

(1) The difference in performance between the single and the double precision solver is too small. In this case, even a few iterations of the refinement phase will compensate for the small speedup. This is, for example, the case of the dawson5 matrix on the PowerPC 970 architecture.

(2) The number of iterations to convergence is too high. The number of iterations to convergence is directly related to the matrix condition number [Langou et al. 2006]. The case of the nasasrb matrix on the PowerPC 970 architecture show that even if the single precision solver is almost 1.4× faster, the mixed precision solver is slower than the double precision one because of the high number of iterations (11) needed to achieve the same accuracy. If the condition number is too high, the method may not converge at all as in the case of the s3rmt3m1 matrix. If convergence is not achieved within the maximum number of iterations (20 in our experiments), the dark bar is not reported in the figures meaning that the speedup of the mixed precision iterative refinement method over the double precision one can be considered equal to zero since a wrong result is produced.

(3) The cost of each iteration is high compared to the performance difference between the double precision solver and the single precision one. In this case, even a few iterations can eliminate the benefits of performing the system factorization in single precision. As an example, take the case of the rma10 matrix on the Intel Woodcrest architecture; two iteration steps on this matrix took 0.1 seconds, which is almost the same time needed to perform six iteration steps on the G64 matrix.

It is worth noting that, apart from the cases where the method does not converge, whenever the method results in a slowdown, the loss is on average only 7%.

Table VII shows the residual of the solutions computed with the double precision solver and the mixed-precision iterative solver for sparse direct methods on the Intel Woodcrest architecture. Note that for the matrices used and, in general, for well-conditioned matrices, the mixed-precision iterative method is

Table VIII. Time to Solution of SuperLU in Single and Double Precision Solvers for the Selected Sparse Matrices on Intel Woodcrest with Reference and Optimized BLAS

	Reference BLAS			Goto BLAS		
Name	t_{DP}	t_{SP}	t_{DP}/t_{SP}	t_{DP}	t_{SP}	t_{DP}/t_{SP}
G64	102.76	80.54	1.27	101.27	85.23	1.18
Si10H16	392.71	308.83	1.27	343.88	345.18	0.99
c-71	1326.36	984.31	1.34	1266.28	1076.13	1.17
cage11	1274.90	945.81	1.34	1143.23	1047.29	1.09
dawson5	6.94	5.86	1.18	6.19	5.70	1.08
nasasrb	33.02	27.90	1.18	30.87	30.48	1.01
poisson3Db	553.19	404.26	1.36	515.00	450.44	1.14
rma10	3.10	2.74	1.13	2.97	2.52	1.17
s3rmt3m1	0.32	0.28	1.14	0.35	0.32	1.09
wang4	21.77	16.99	1.28	19.32	18.96	1.01

capable of delivering the same or better accuracy than the double precision solver. The same was also observed for the mixed-precision sparse iterative solvers. In the case of matrix s3rmt3m1, convergence is not achieved within the maximum number of iterations on any platform due to the high condition number (see Table I); the residual of the mixed-precision iterative refinement solver is much higher (and very high in general) than that of the double precision one in this case.

Table VIII shows the timings of the sequential version of SuperLU on selected matrices from our test collection for single and double precision solvers. Both reference and Goto BLAS timings are shown. The sequential version of SuperLU implements matrix-vector multiply (_GEMV) as its computational kernel. This explains the rather modest gains (if any) in the performance of the single precision solver over the double precision one: only up to 30%. The table also reveals that when optimized BLAS are used, the single precision is slower than double for some matrices, an artifact of small sizes of dense matrices passed to BLAS and the level of optimization of the BLAS for this particular architecture. The results are similar for other tested architectures, which leads to a conclusion that there is not enough benefit in using our mixed precision approach for this version of SuperLU.

The largest performance gain of only 30% for SuperLU is not supported by the data from Table V. The explanation for this is twofold: the calls to the _GEMV kernel routine involve very small matrices and the fact that speeding up the computational part of the factorization doesn't affect the symbolical part of it, that is, the operations on sparse data structures (the size and amount of work on these data structures is always the same regardless of the precision chosen for the matrix data). The consequence of the former is much higher performance sensitivity to memory latency, function call overhead, and slow down due to clean-up code. The latter is also true for MUMPS but is offset as the multifrontal factorization progresses: the frontal matrix sizes keep growing, which in turn results in performance gains comparable to those from Table V.

Next, we present our results on the mixed-precision iterative sparse solvers from Section 3.2. All the results are from runs on Intel Woodcrest (3GHz with a 1333MHz frontside bus).

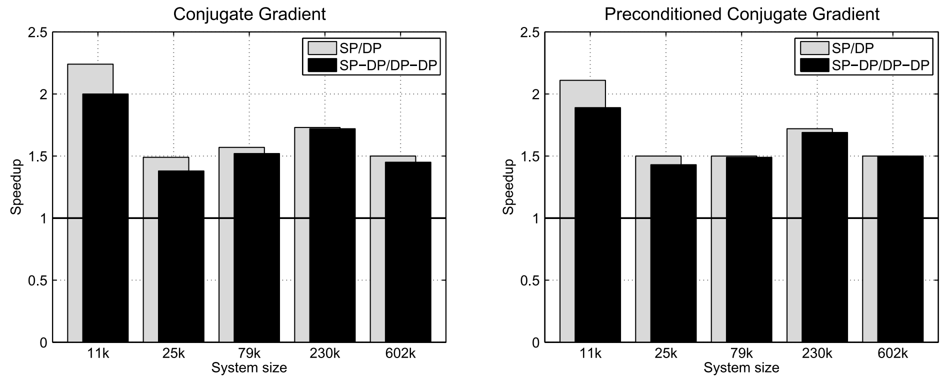
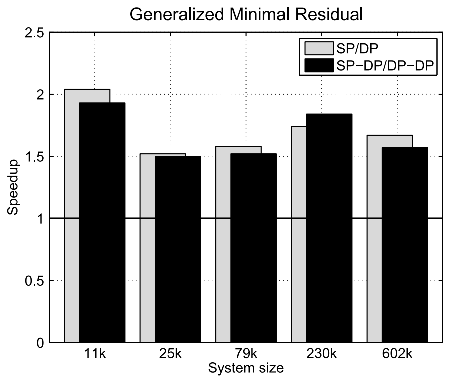


Fig. 5. Left: Speedup using SP vs DP CG (light bars) and the SP-DP *vs* DP-DP CG-PCG (dark). Right: Similar graph comparison but for the PCG algorithm (see also Section 3.2.1). The computations were on a Intel Woodcrest (3GHz with a 1333MHz frontside bus).

It is not known how to choose the restart size m to get optimal results even for the reference GMRES(m). Assuming, for example, that the bigger m the better does not guarantee better execution time, and sometimes the convergence can get even worse [Embree 2003]. An alternative worth further exploration is to use a truncated version of GMRES [Saad and Wu 1996]. Another interesting approach is self adaptivity [Demmel et al. 2005]. Here, to do a fair comparison, we ran it for m = 25, 50 (PETSc's default [Balay et al. 2001]), 100, 150, 200, and 300, and chose the best execution time. Experiments show that the mixed-precision method suggested is stable in regard to changing the restart values in the inner and outer loops. The experiments presented are for inner and outer m = 20. Note that this choice also results in smaller memory requirements than GMRES, with $m \approx 70$ and higher (for most of the runs GMRES(100), and was best among the previous choices for m) since the overhead in terms of DP vectors is $20+20$ (outer GMRES) +10 (20 SP vectors in the inner loop) +20 (matrix coefficients in SP; there are approximately 40 nonzeros per row, see Table II). In all the cases presented, we had the number of inner cycles/restarts set to one.

In Figure 5, we give the speedups for using mixed SP-DP vs DP-DP CG (dark bars). Namely, on the left, we have the results for CG-PCG and, on the right, for PCG-PCG with diagonal preconditioner in the inner loop PCG. Similarly, in Figure 6, we give the results for GMRES-FGMRES (on the left) and PGMRES-FGMRES (on the right). Also, we compare the speedups of using SP vs DP for just the reference CG and PCG (correspondingly left and right in Figure 5, light bars) and SP vs DP for the reference GMRES and PGMRES (light bars in Figure 6). Note that in a sense the speedups in the light bars should represent the maximum that could be achieved by using the mixed precision algorithms. The fact that we get close to this maximum performance shows that we have successfully shifted the load from DP to SP arithmetic (with overall computation having less than 5% in DP arithmetic). The reason that the performance speedup for SP-DP vs DP-DP GMRES-FGMRES in



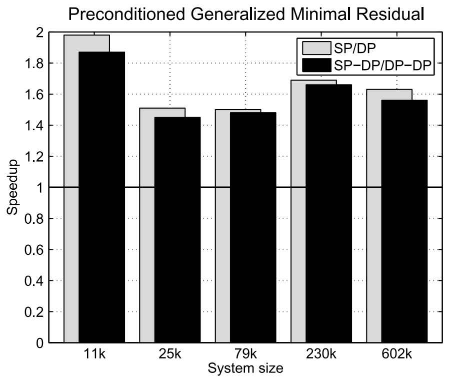
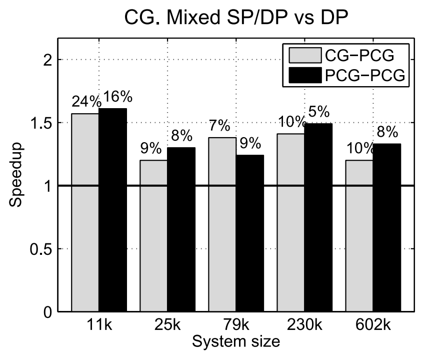


Fig. 6. Left: Speedup of using SP vs DP GMRES (light bars) and the SP-DP vs DP-DP GMRES-FGMRES (dark). Right: Similar graph comparison but for the PGMRES algorithm (see also Section 3.2.2. The computations were on a Intel Woodcrest (3GHz with a 1333MHz frontside bus).



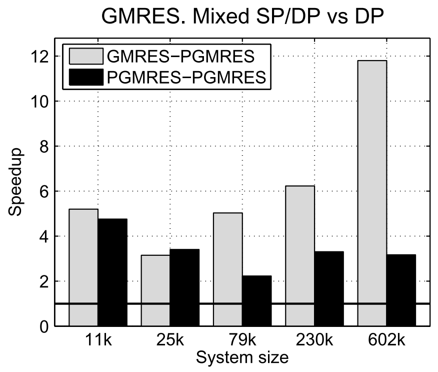


Fig. 7. Left: Speedup of mixed SP-DP CG-PCG vs DP CG (light bars) and SP-DP PCG-PCD vs DP CG (dark) with diagonal preconditioner. The numbers on top of the bars indicate the percentage overhead measured in numbers of iterations. Right: Similar graph comparison but for the GMRES based algorithms. The computations were on a Intel Woodcrest (3GHz with a 1333MHz bus).

Figure 6 (left, 4th matrix) is higher than the speedup of SP GMRES vs DP GMRES is that the SP-DP GMRES-FGMRES did one less outer cycle until convergence than the DP-DP GMRES-FGMRES.

Results comparing the SP-DP methods with the reference DP methods are shown in Figure 7 (left is a comparison for CG, right is for GMRES). The numbers on top of the bars on the left graph indicate the overhead as the number of iterations that it took for the mixed precision method to converge versus the reference DP method (e.g., overhead of 10% indicates 10% more iterations were performed in the mixed SP-DP vs the DP method). Even with the overhead, we see a performance speedup of at least 20% over the tested matrices. For the GMRES-based mixed precision methods, we see a significant improvement based on a reduced number of iterations and the effect of the SP speedup (from 45 to 100% as indicated in Figure 6). For example, the speedup factor of 12 for the biggest problem is due to speedup factors of approximately 7.5 from improved convergence and 1.6 from effects associated with the introduced SP storage and arithmetic.

Finally, we note the speedup for direct and iterative methods and its effect on performance. The speed up of moving to single precision for _GEMM-calling code (MUMPS) was approaching two and thus guaranteed success of our mixed-precision iterative refinement just as it did for the dense matrix operations. Not so for the _GEMV-calling code (SuperLU) for which the speedup did not exceed 30%, and thus no performance improvement was expected. However, for most of the iterative methods, the speedup was around 50%, and still we claim our approach to be successful. Inherently, the reason for speedup is the same for both settings (SuperLU and the iterative methods): the reduced memory bus traffic and possible superlinear effects when data fits in cache while being stored in single precision. But for the SuperLU case, there is the direct method overhead: the maintenance of evolving sparse data structures, which is done in fixed-point arithmetic so it does not benefit from using single precision floating-point arithmetic, and hence yields the overall performance gains insufficient for our iterative refinement approach.

5. FUTURE WORK

We are considering a number of extensions and new directions for our work. The most broad category is the parallel setting. MUMPS is a parallel code, but it was used in a sequential setting in this study. Similarly, SuperLU has a parallel version that differs from the sequential counterpart in a very important way; it uses the matrix-matrix multiply kernel (_GEMM). This would give a better context for comparing multifrontal and supernodal approaches since they use the same underlying computational library. The only caveat is the lack of a single precision version of the parallel SuperLU solver. Another aspect brought by the latter solver is using static pivoting. While it vastly improves numerical stability of parallel SuperLU, it also improves the convergence of the iterative refinement that follows. This should result in less iterations and shorter solve time.

Using PETSc and its parallel framework for (among others) iterative methods could give us an opportunity to investigate our approach for a wider range of iterative methods and preconditioning scenarios. First though, we would have to overcome a technical obstacle of combining two versions of PETSc (one using single and one using double precision) in a single executable.

We have performed preliminary experiments on actual IBM Cell BE hardware (as opposed to the simulator, which does not accurately account for memory system effects, a crucial component of sparse methods) with sparse matrix operations and are encouraged by the results to port our techniques in full. This would allow us to study their behavior with a much larger gap in the performance of the two precisions.

Our algorithms and their descriptions focus solely on two precisions, single and double. We see them, however, in a broader context of higher and lower precision where, for example, a GPU performs computationally-intensive operations in its native 16-bit arithmetic, and consequently the solution is refined using 128-bit arithmetic emulated in software (if necessary). As mentioned before, the limiting factor is conditioning of the system matrix. In fact, an estimate (up to the order of magnitude) of the condition number (often available from previous runs or the physical problem properties) may become an input parameter to an adaptive algorithm [Dongarra and Eijkhout 2002] that attempts to utilize the fastest hardware available if its limited precision can guarantee convergence.

Also, the methods for sparse eigenvalue problems that result in Lanczos and Arnoldi algorithms are amenable to our techniques, and we would like to study their theoretical and practical challenges.

REFERENCES

AMESTOY, P. R., DUFF, I. S., AND L'EXCELLENT, J.-Y. 2000. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Meth. Appl. Mech. Eng. 184*, 501–520.

AMESTOY, P. R., DUFF, I. S., L'EXCELLENT, J.-Y., AND KOSTER, J. 2001. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM J. Matrix Anal. Appl. 23*, 15–41.

AMESTOY, P. R., GUERMOUCHE, A., L'EXCELLENT, J.-Y., AND PRALET, S. 2006. Hybrid scheduling for the parallel solution of linear systems. *Paral. Comput. 32*, 136–156.

ANDERSON, E., BAI, Z., BISCHOF, C., BLACKFORD, S. L., DEMMEL, J. W., DONGARRA, J. J., CROZ, J. D., GREENBAUM, A., HAMMARLING, S., MCKENNEY, A., AND SORENSEN, D. C. 1999. *LAPACK User's Guide* 3rd Ed. Society for Industrial and Applied Mathematics, Philadelphia, PA.

ASHCRAFT, C., GRIMES, R., LEWIS, J., PEYTON, B. W., AND SIMON, H. 1987. Progress in sparse matrix methods in large sparse linear systems on vector supercomputers. *Intern. J. of Supercomput. Appl. 1*, 10–30.

AXELSSON, O. AND VASSILEVSKI, P. S. 1991. A black box generalized conjugate gradient solver with inner iterations and variable-step preconditioning. *SIAM J. Matrix Anal. Appl. 12,* 4, 625–644.

BALAY, S., BUSCHELMAN, K., GROPP, W. D., KAUSHIK, D., KNEPLEY, M. G., MCINNES, L. C., SMITH, B. F., AND ZHANG, H. 2001. PETSc Web page. http://www.mcs.anl.gov/petsc.

BARRETT, R., BERRY, M., CHAN, T. F., DEMMEL, J., DONATO, J. M., DONGARRA, J., EIJKHOUT, V., POZO, R., ROMINE, C., AND DER VORST, H. V. 1994. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods.* Society for Industrial and Applied Mathematics, Philadelphia, PA. http://www.netlib.org/templates/Templates.html.

BJÖRCK, A. 1990. Iterative refinement and reliable computing. In *Reliable Numerical Computation*, M. G. Cox and S. Hammarling, Eds. Oxford University Press, Oxford, UK, 249–266.

BUTTARI, A., DONGARRA, J., KURZAK, J., LUSZCZEK, P., AND TOMOV, S. 2006. Computations to enhance the performance while achieving the 64-bit accuracy. Tech. rep. UT-CS-06-584, University of Tennessee Knoxville. LAPACK Working Note 180.

DAVIS, T. A. 1999. A combined unifrontal/multifrontal method for unsymmetric sparse matrices. *ACM Trans. Math. Softw. 25*, 1–19.

DAVIS, T. A. 2004. A column pre-ordering strategy for the unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw. 30*, 196–199.

DEMMEL, J., DONGARRA, J., EIJKHOUT, V., FUENTES, E., PETITET, A., VUDUC, R., WHALEY, R. C., AND YELICK, K. 2005. Self-adapting linear algebra algorithms and software. *Proc. IEEE 93,* 2. http://www.spiral.net/ieee-special-issue/overview.html.

DEMMEL, J. W. 1997. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, Philadelphia, PA.

DEMMEL, J. W., EISENSTAT, S. C., GILBERT, J. R., LI, X. S., AND LIU, J. W. H. 1999a. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Anal. Appl. 20,* 3, 720–755.

DEMMEL, J. W., GILBERT, J. R., AND LI, X. S. 1999b. A asynchronous parallel supernodal algorithm for sparse gaussian elimination. *SIAM J. Matrix Anal. Appl. 20,* 3, 915–952.

DONGARRA, J. J. AND EIJKHOUT, V. 2002. Self-adapting numerical software for next generation applications. Tech. rep. ICL-UT-02-07, Innovative Computing Lab, University of Tennessee, Lapack Working Note 157. http://icl.cs.utk.edu/iclprojects/pages/sans.html.

DUFF, I. S. AND REID, J. K. 1983. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Trans. Math. Softw. 9,* 3, 302–325.

DUFF, T. A. D. I. S. 1997. An unsymmetric-pattern multifrontal method for sparse LU factorization. *SIAM J. Matrix Anal. Appl. 18*, 140–158.

EMBREE, M. 2003. The tortoise and the hare restart gmres. *SIAM Rev. 45*, 259–266.

FORSYTHE, G. E. AND MOLER, C. B. 1967. *Computer Solution of Linear Algebraic Systems*. Prentice-Hall, Englewood Cliffs, NJ.

GÖDDEKE, D., STRZODKA, R., AND TUREK, S. 2005. Accelerating double precision FEM simulations with GPUs. In *Simulationstechnique 18th Symposium in Erlangen*. F. Hülsemann, M. Kowarschik, and U. Rüde, Eds. Vol. Frontiers in Simulation. SCS Publishing House e.V., 139–144.

GOLUB, G. H. AND LOAN, C. F. V. 1989. *Matrix Computations* 2nd Ed. Johns Hopkins University Press, Baltimore, MD.

GOLUB, G. H. AND YE, Q. 2000. Inexact preconditioned conjugate gradient method with inner-outer iteration. *SIAM J. Scie. Comput. 21,* 4, 1305–1320.

GROPP, W. D., KAUSHIK, D. K., KEYES, D. E., AND SMITH, B. F. 2000. Latency, bandwidth, and concurrent issue limitations in high-performance CFD. Tech. rep. ANL/MCS-P850-1000, Argonne National Laboratory.

GROPP, W. D., KAUSHIK, D. K., KEYES, D. E., AND SMITH, B. F. 2001. High-performance parallel implicit CFD. *Parall. Comput. 27,* 4, 337–362.

GURTIN, M. E. 1981. *An Introduction to Continuum Mechanics*. Academic Press, New York, NY.

HACKBUSCH, W. 1985. *Multigrid Methods and Applications*. Springer Series in Computational Mathematics, Vol. 4, Springer-Verlag, Berlin, Germany.

HIGHAM, N. J. 2002. *Accuracy and Stability of Numerical Algorithms* 2nd Ed. Society for Industrial and Applied Mathematics, Philadelphia, PA.

LANGOU, J., LANGOU, J., LUSZCZEK, P., KURZAK, J., BUTTARI, A., AND DONGARRA, J. 2006. Exploiting the performance of 32-bit floating point arithmetic in obtaining 64 bit accuracy. In *Proceedings of the ACM/IEEE Conference on High Performance Networking and Computing (SC'06)*. Tampa, FL. http://icl.cs.utk.edu/iter-ref.

LI, X. S. 1996. SuperLU software, Ph.D. thesis, Computer Science Department, University of California at Berkeley. http://www.nersc.gov/ xiaoye/SuperLU/.

LI, X. S. AND DEMMEL, J. W. 2003. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Math. Softw. 29*, 110–140.

MOLER, C. B. 1967. Iterative refinement in floating point. *J. ACM 14,* 2, 316–321.

NOTAY, Y. 2000. Flexible conjugate gradients. *SIAM J. Scie. Comput. 22*, 1444–1460.

QUARTERONI, A. AND VALLI, A. 1999. *Domain Decomposition Methods for Partial Differential Equations*. Oxford University Press, Cambridge, UK.

SAAD, Y. 1991. A flexible inner-outer preconditioned GMRES algorithm. Tech. rep. 91-279, Department of Computer Science and Egineering, University of Minnesota, Minneapolis, MN.

SAAD, Y. 2003. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA.

SAAD, Y. AND SCHULTZ, M. H. 1986. GMRES: A generalized minimal residual method for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 856–869.

SAAD, Y. AND WU, K. 1996. DQGMRES: a direct quasi-minimal residual algorithm based on incomplete orthogonalization. *Num. Linear Algeb. Appl. 3,* 4, 329–343.

SIMONCINI, V. AND SZYLD, D. 2002a. Theory of inexact Krylov subspace methods and applications to scientific computing. Tech. rep. 02-4-12, Department of Mathematics, Temple University.

SIMONCINI, V. AND SZYLD, D. B. 2002b. Flexible inner-outer Krylov subspace methods. *SIAM J. Numer. Anal. 40,* 6, 2219–2239.

SIMONCINI, V. AND SZYLD, D. B. 2005. The effect of non-optimal bases on the convergence of Krylov subspace methods. *Numer. Math. 100,* 4, 711–733.

STEWART, G. W. 2001. *Matrix algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA.

STRZODKA, R. AND GÖDDEKE, D. 2006a. Mixed precision methods for convergent iterative schemes. EDGE'06, 23.-24. Chapel Hill, NC.

STRZODKA, R. AND GÖDDEKE, D. 2006b. Pipelined mixed precision algorithms on FPGAs for fast and accurate PDE solvers from low precision components. In *IEEE Proceedings on Field–Programmable Custom Computing Machines (FCCM'06)*. IEEE Computer Society Press. To appear.

TURNER, K. AND WALKER, H. F. 1992. Efficient high accuracy solutions with gmres(m). *SIAM J. Sci. Stat. Comput. 13,* 3, 815–825.

VAN DEN ESHOF, J., SLEIJPEN, G. L. G., AND VAN GIJZEN, M. B. 2003. Relaxation strategies for nested Krylov methods. Technical report TR/PA/03/27, CERFACS, Toulouse, France.

VAN DER VORST, H. A. AND VUIK, C. 1994. GMRESR: a family of nested GMRES methods. *Num. Linear Algeb. Appl. 1,* 4, 369–386.

VUIK, C. 1995. New insights in gmres-like methods with variable preconditioners. *J. Comput. Appl. Math. 61,* 2, 189–204.

WILKINSON, J. H. 1965. *The Algebraic Eigenvalue Problem*. Oxford University Press, Oxford, UK.

Received November 2006; revised April 2007, June 2007; accepted July 2007