Mixed Precision Iterative Refinement Techniques for the Solution of Dense Linear Systems

Alfredo Buttari[1], Jack Dongarra[1,3,4], Julie Langou[1], Julien Langou[2], Piotr Luszczek[1], and Jakub Kurzak[1]

[1]*Department of Electrical Engineering and Computer Science, University Tennessee, Knoxville, Tennessee*
[2]*Department of Mathematical Sciences, University of Colorado at Denver and Health Sciences Center, Denver, Colorado*
[3]*Oak Ridge National Laboratory, Oak Ridge, Tennessee*
[4]*University of Manchester*

September 9, 2007

Abstract

By using a combination of 32-bit and 64-bit floating point arithmetic, the performance of many dense and sparse linear algebra algorithms can be significantly enhanced while maintaining the 64-bit accuracy of the resulting solution. 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. Results on modern processor architectures and the Cell BE are presented.

Introduction

In numerical computing, there is a fundamental performance advantage in using the single precision, floating point data format over the double precision one. Due to more compact representation, twice the number of single precision data elements can be stored at each level of the memory hierarchy including the register file, the set of caches, and the main memory. By the same token, handling single precision values consumes less bandwidth between different memory levels and decreases the number of cache and TLB misses. However, the data movement aspect affects mostly memory-intensive, bandwidth-bound applications, and historically has not drawn much attention to mixed precision algorithms.

In the past, the situation looked differently for computationally intensive workloads, where the load was on the floating point processing units rather than the memory subsystem, and so the single precision data motion advantages were for the most part irrelevant. With the focus on double precision in the scientific computing, double

precision execution units were fully pipelined and capable of completing at least one operation per clock cycle. In fact, in many high performance processor designs single precision units were eliminated in favor of emulating single precision operations using double precision circuitry. At the same time, a high degree of instruction level parallelism was being achieved by introducing more functional units and relatively complex speculation mechanisms, which did not necessarily guarantee full utilization of the hardware resources.

That situation began to change with the widespread adoption of short vector, Single Instruction Multiple Data (SIMD) processor extensions, which started appearing in the mid 90's. An example of such extensions are the Intel MultiMedia eXtensions (MMX) that were mostly meant to improve processor performance in Digital Signal Processing (DSP) applications, graphics and computer games. Short vector, SIMD instructions are a relatively cheap way of exploiting data level parallelism by applying the same operation to a vector of elements at once. It eliminates the hardware design complexity associated with the bookkeeping involved in speculative execution. It also gives better guarantees for practically achievable performance than does runtime speculation, provided that enough data parallelism exists in the computation. Most importantly, short vector, SIMD processing provides the opportunity to benefit from replacing the double precision arithmetic with the single precision one. Since the goal is to process the entire vector in a single operation, the computational throughput doubles while the data storage space halves.

Most processor architectures available today have been augmented, at some point, in their design evolution with short vector, SIMD extensions. Examples include Streaming SIMD Extensions (SSE) for the AMD and the Intel line of processors; PowerPC's Velocity Engine, AltiVec, and VMX; SPARC's Visual Instruction Set (VIS); Alpha's Motion Video Instructions (MVI); PA-RISC's Multimedia Acceleration eXtensions (MAX); MIPS-3D Application Specific Extensions (ASP) and Digital Media Extensions (MDMX) and ARM's NEON feature. The different architectures exhibit large differences in their capabilities. The vector size is either 64 bits or, more commonly, 128 bits. The register file size ranges from just a few to as many as 256 registers. Some extensions only support integer types while others operate on single precision, floating point numbers, and yet others process double precision values.

Today, the Synergistic Processing Element (SPE) of the CELL processor can probably be considered the state of the art in short vector, SIMD processing. Possessing 128-byte long registers and a fully pipelined fused, multiply-add instruction, it is capable of completing as many as eight single precision, floating point operations each clock cycle. When combined with the size of the register file of 128 registers, it is capable of delivering close to peak performance on many common computationally intensive workloads.

Table 1 shows the difference in peak performance between single precision (SP) and double precision (DP) of four modern processor architectures; also, on the last column is reported the ratio between the time needed to solve a dense linear system in double and single precision by means of the LAPACK `DGESV` and `SGESV` respectively. Following the recent trend in chip design, all of the presented processors are multi-core architectures. However, to avoid introducing the complexity of thread-level parallelization to the discussion, we will mainly look at the performance of individual

Table 1: Floating point performance characteristics of *individual cores* of modern, multi-core processor architectures. `DGESV` and `SGESV` are the LAPACK subroutines for dense system solution in double precision and single precision respectively.

Architecture	**Clock [GHz]**	**DP Peak [Gflop/s]**	**SP Peak [Gflop/s]**	time(`DGESV`)/ time(`SGESV`)
AMD Opteron 246	2.0	4	8	1.96
IBM PowerPC 970	2.5	10	20	1.87
Intel Xeon 5100	3.0	12	24	1.84
STI Cell BE	3.2	1.8[1]	25.6	11.37

cores throughout the chapter. The goal here is to focus on instruction-level parallelism exploited by short vector SIMD'zation.

Although short vector, SIMD processors have been around for over a decade, the concept of using those extensions to utilize the advantages of single precision performance in scientific computing did not come to fruition until recently, due to the fact that most scientific computing problems require double precision accuracy. It turns out, however, that for many problems in numerical computing, it is possible to exploit the speed of single precision operations and resort to double precision calculations at few stages of the algorithm to achieve full double precision accuracy of the result. The techniques described here are fairly general and can be applied to a wide range of problems in linear algebra, such as solving linear systems of equations, least square problems, singular value and eigenvalue problems. Here we are going to focus on solving dense linear systems of equations, both non-symmetric and symmetric positive definite, using direct methods. An analogous approach for the solution of sparse systems, both with direct and Krylov iterative methods, is presented in [1].

1 Direct Methods for Solving Dense Systems

1.1 Algorithm

Iterative refinement is a well known method for improving the solution of a linear system of equations of the form $Ax = b$ [2]. The standard approach to the solution of dense linear systems is to use the LU factorization by means of Gaussian elimination. First, the coefficient matrix A is factorized into the product of a lower triangular matrix L and an upper triangular matrix U using LU decomposition. Commonly, partial row pivoting is used to improve numerical stability resulting in the factorization $PA = LU$, where P is the row permutation matrix. The solution for the system is obtained by first solving $Ly = Pb$ (*forward substitution*) and then solving $Ux = y$ (*back substitution*). Due to the round-off error, the computed solution x carries a numerical error magnified by the condition number of the coefficient matrix A. In order to improve the computed solution, an iterative refinement process is applied, which produces a correction to the computed solution at each iteration, which then yields the basic iterative refinement

[1] The DP unit is not fully pipelined, and has a 7 cycle latency.

algorithm (Algorithm 1). As Demmel points out [3], the non-linearity of the round-off error makes the iterative refinement process equivalent to the Newton's method applied to the function $f(x) = b - Ax$. Provided that the system is not too ill-conditioned, the algorithm produces a solution correct to the working precision. Iterative refinement is a fairly well understood concept and was analyzed by Wilkinson [4], Moler [5] and Stewart [2].

Algorithm 1 The iterative refinement method for the solution of linear systems

1: $x_0 \leftarrow A^{-1}b$
2: $k = 1$
3: **repeat**
4: $\quad r_k \leftarrow b - Ax_{k-1}$
5: $\quad z_k \leftarrow A^{-1}r_k$
6: $\quad x_k \leftarrow x_{k-1} + z_k$
7: $\quad k \leftarrow k + 1$
8: **until** convergence

The algorithm can be modified to use a mixed precision approach. The factorization $PA = LU$ and the solution of the triangular systems $Ly = Pb$ and $Ux = y$ are computed using single precision arithmetic. The residual calculation and the update of the solution are computed using double precision arithmetic and the original double precision coefficients. The most computationally expensive operations, including the factorization of the coefficient matrix A and the forward and backward substitution, are performed using single precision arithmetic and take advantage of its higher speed. The only operations that must be executed in double precision are the residual calculation and the update of the solution. It can be observed, that all operations of $O(n^3)$ computational complexity are handled in single precision, and all operations performed in double precision are of at most $O(n^2)$ complexity. The coefficient matrix A is converted to single precision for the LU factorization and the resulting factors are also stored in single precision. At the same time, the original matrix in double precision must be preserved for the residual calculation. The mixed precision, iterative refinement algorithm is outlined in Algorithm 2; the (32) subscript means that the data is stored in 32-bit format (i.e., single precision) and the absence of any subscript means that the data is stored in 64-bit format (i.e., double precision). Implementation of the algorithm is provided in the LAPACK package by the routine DSGESV.

Higham [6] gives error bounds for the single and double precision, iterative refinement algorithm when the entire algorithm is implemented with the same precision (single or double, respectively). He also gives error bounds in single precision arithmetic, with refinement performed in double precision arithmetic [6]. The error analysis in double precision, for our mixed precision algorithm (Algorithm 2), is given in Appendix A.

The same technique can be applied to the case of symmetric, positive definite problems. Here, Cholesky factorization (LAPACK's SPOTRF routine) can be used in place of LU factorization (SGETRF), and a symmetric back solve routine (SPOTRS) can be used in place of the routine for the general (non-symmetric) case (SGETRS). Also,

Algorithm 2 Solution of a linear system of equations using mixed precision, iterative refinement. (SGETRF and SGETRS are names of LAPACK routines).

$A_{(32)}, b_{(32)} \leftarrow A, b$
$L_{(32)}, U_{(32)}, P_{(32)} \leftarrow \text{SGETRF}(A_{(32)})$
$x^{(1)}_{(32)} \leftarrow \text{SGETRS}(L_{(32)}, U_{(32)}, P_{(32)}, b_{(32)})$
$x^{(1)} \leftarrow x^{(1)}_{(32)}$
$i \leftarrow 0$
repeat
  $i \leftarrow i + 1$
  $r^{(i)} \leftarrow b - Ax^{(i)}$
  $r^{(i)}_{(32)} \leftarrow r^{(i)}$
  $z^{(i)}_{(32)} \leftarrow \text{SGETRS}(L_{(32)}, U_{(32)}, P_{(32)}, r^{(i)}_{(32)})$
  $z^{(i)} \leftarrow z^{(i)}_{(32)}$
  $x^{(i+1)} \leftarrow x^{(i)} + z^{(i)}$
until $x^{(i)}$ is *accurate enough*

the matrix-vector product Ax can be implemented by the BLAS' DSYMV routine, or DSYMM for multiple right hand sides, instead of the DGEMV and DGEMM routines for the non-symmetric case. The mixed precision algorithm for the symmetric, positive definite case is presented by Algorithm 2. Implementation of the algorithm is provided in the LAPACK package by the routine DSPOSV.

Algorithm 3 Solution of a symmetric positive definite system of linear equations using mixed precision, iterative refinement. (SPOTRF and SPOTRS are names of LAPACK routines).

$A_{(32)}, b_{(32)} \leftarrow A, b$
$L_{(32)}, L^T_{(32)} \leftarrow \text{SPOTRF}(A_{(32)})$
$x^{(1)}_{(32)} \leftarrow \text{SPOTRS}(L_{(32)}, L^T_{(32)}, b_{(32)})$
$x^{(1)} \leftarrow x^{(1)}_{(32)}$
$i \leftarrow 0$
repeat
  $i \leftarrow i + 1$
  $r^{(i)} \leftarrow b - Ax^{(i)}$
  $r^{(i)}_{(32)} \leftarrow r^{(i)}$
  $z^{(i)}_{(32)} \leftarrow \text{SPOTRS}(L_{(32)}, L^T_{(32)}, r^{(i)}_{(32)})$
  $z^{(i)} \leftarrow z^{(i)}_{(32)}$
  $x^{(i+1)} \leftarrow x^{(i)} + z^{(i)}$
until $x^{(i)}$ is *accurate enough*

1.2 Experimental Results and Discussion

To collect performance results for the Xeon, Opteron and PowePC architectures, the LAPACK iterative refinement routines DSGESV and DSPOSV were used, for the non-symmetric and symmetric cases, respectively. The routines implement classic, blocked versions of the matrix factorizations and rely on the layer of Basic Linear Algebra Subroutines (BLAS) for architecture specific optimizations to deliver performance close to the peak. As mentioned before, in order to simplify the discussion and leave out the aspect of parallelization, we have decided to look at the performance of individual cores on the multi-core architectures.

Figures 1-8 show the performance of the single-core serial implementations of Algorithm 2 and Algorithm 3 on the architectures in Table 2.

Table 2: Hardware and software details of the systems used for performance experiments.

Architecture	**Clock [GHz]**	Memory **[MB]**	BLAS	Compiler
AMD Opteron 246	2.0	2048	Goto-1.13	Intel-9.1
IBM PowerPC 970	2.5	2048	Goto-1.13	IBM-8.1
Intel Xeon 5100	3.0	4096	Goto-1.13	Intel-9.1
STI Cell BE	3.2	512	–	Cell SDK-1.1

These figures show that the mixed precision, iterative refinement method can run very close to the speed of the full single precision solver while delivering the same accuracy as the full double precision one. On the AMD Opteron, Intel Woodcrest and IBM PowerPC architectures (see Figures 1- 6), the mixed precision, iterative solver can provide a speedup of up to 1.8 for the unsymmetric solver and 1.5 for the symmetric one, if the problem size is big enough. For small problem sizes, in fact, the cost of even a few iterative refinement iterations is high compared to the cost of the factorization and thus, the mixed precision, iterative solver is less efficient than the full double precision one.

For the Cell processor (see Figures 7 and 8), parallel implementations of Algorithms 2 and 3 have been produced in order to exploit the full computational power of the processor. Due to the large difference between the single precision and double precision floating point units (see Table 1), the mixed precision solver performs up to $7\times$ and $11\times$ faster than the double precision peak in the unsymmetric and symmetric, positive definite cases respectively. Implementation details for this case can be found in [7, 8].

2 Conclusions

The algorithms presented 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

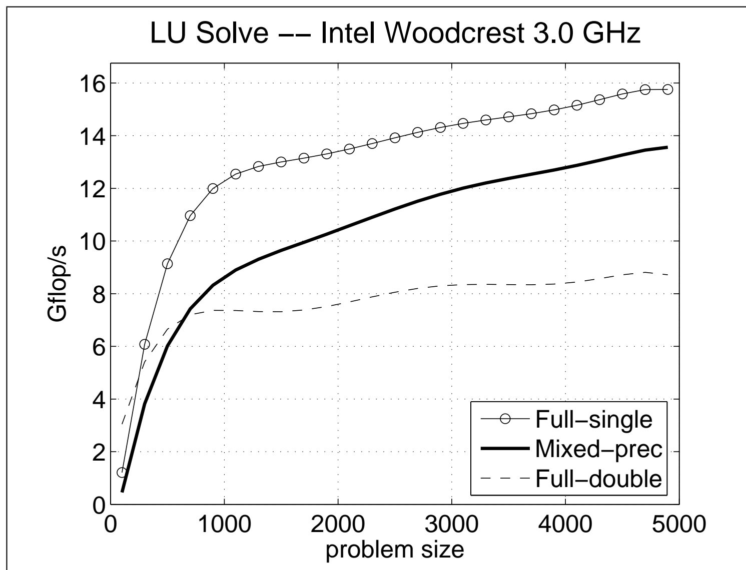

Figure 1: Performance of mixed precision, iterative refinement for unsymmetric problems on Intel Woodcrest.

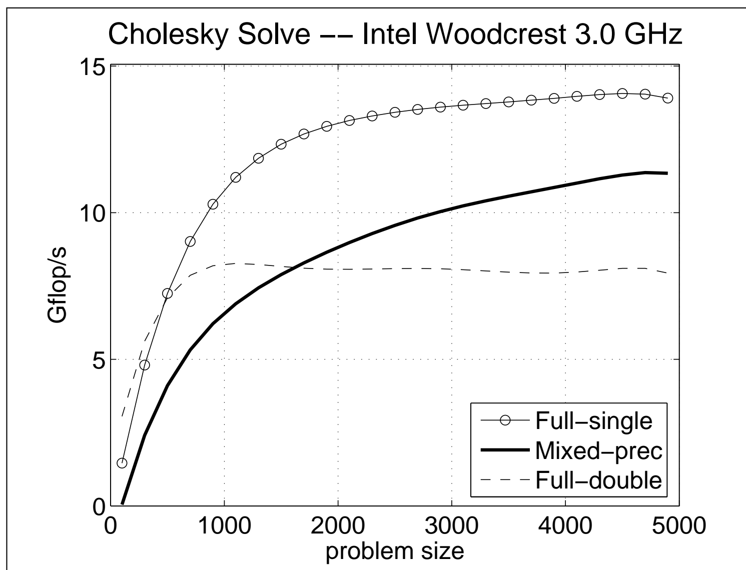

Figure 2: Performance of mixed precision, iterative refinement for symmetric, positive definite problems on Intel Woodcrest.

(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

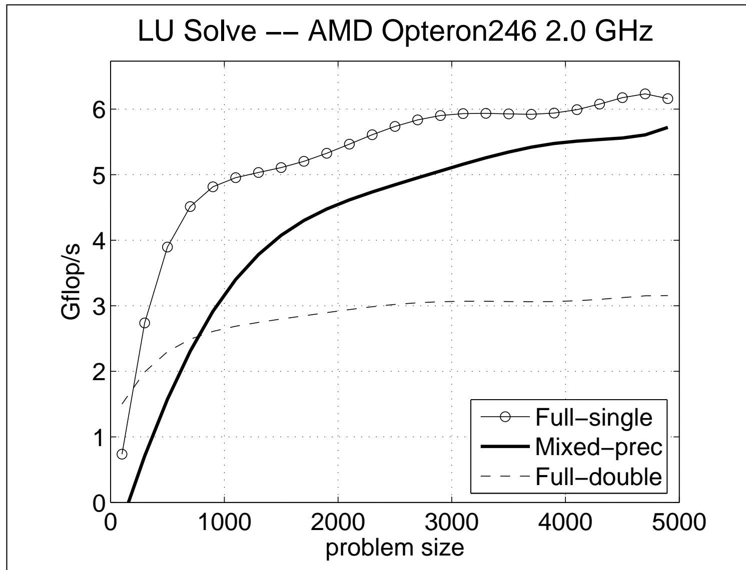

Figure 3: Performance of mixed precision, iterative refinement for unsymmetric problems on AMD Opteron246.

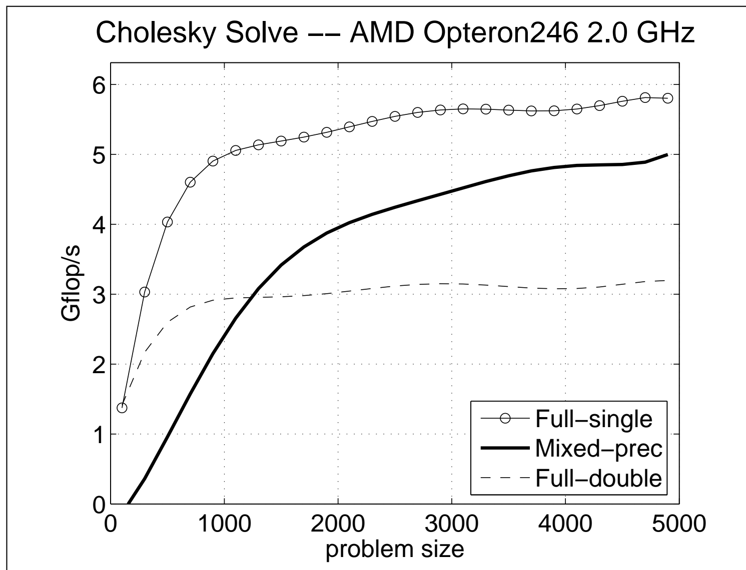

Figure 4: Performance of mixed precision, iterative refinement for symmetric, positive definite problems on AMD Opteron246.

(often available from previous runs or the physical problem properties) may become an input parameter to an adaptive algorithm that attempts to utilize the fastest hardware

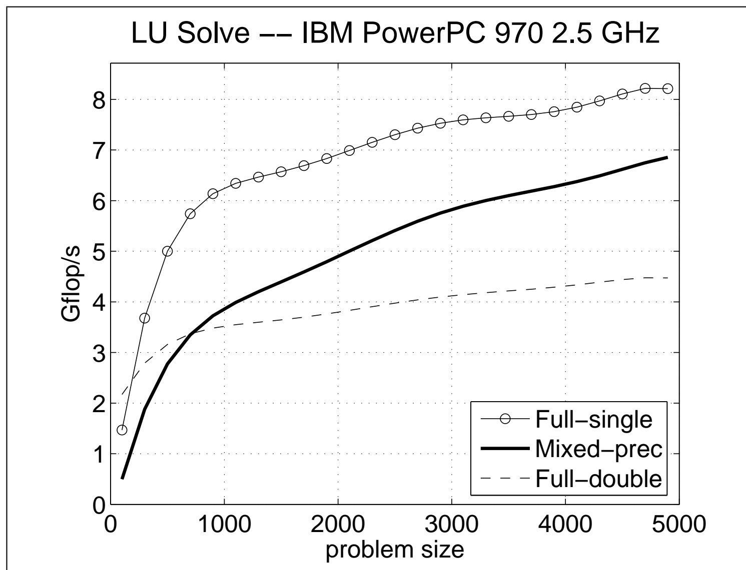

Figure 5: Performance of mixed precision, iterative refinement for unsymmetric problems on IBM PowerPC 970.

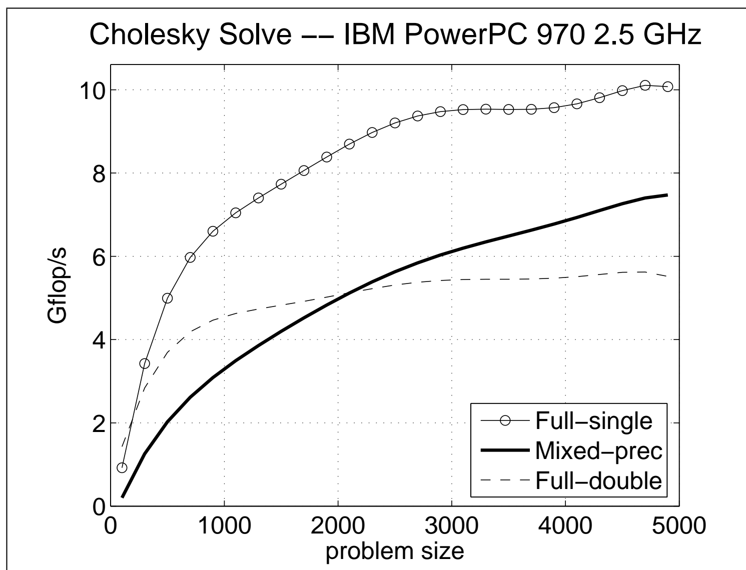

Figure 6: Performance of mixed precision, iterative refinement for symmetric, positive definite problems on IBM PowerPC 970.

available, if its limited precision can guarantee convergence. Also, the methods for sparse eigenvalue problems that result in Lanczos and Arnoldi algorithms are amenable

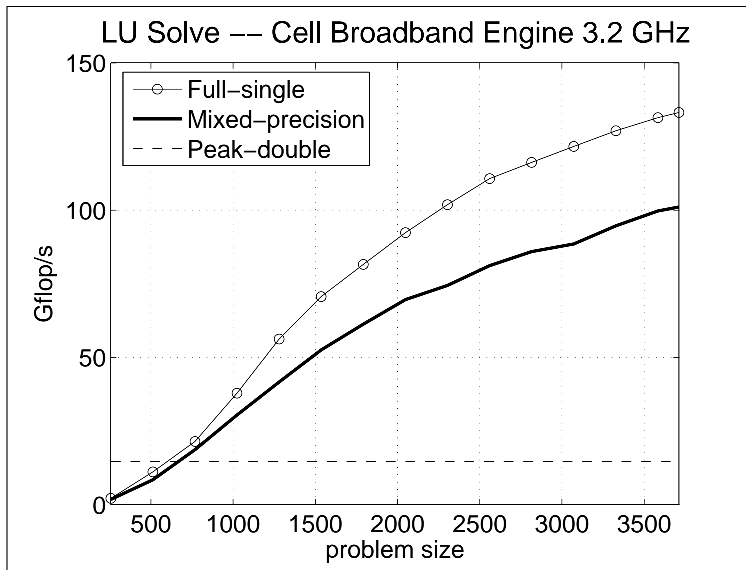

Figure 7: Performance of mixed precision, iterative refinement for unsymmetric problems on CELL Broadband Engine.

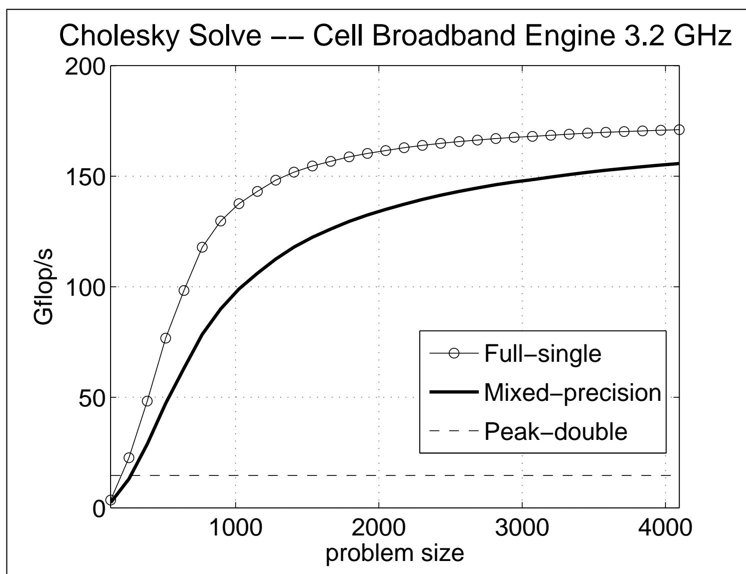

Figure 8: Performance of mixed precision, iterative refinement for symmetric, positive definite problems on CELL Broadband Engine.

to our techniques, and we would like to study their theoretical and practical challenges.

It should be noted that this process can be applied whenever a Newton or "Newton-

like" method is used. That is whenever we are computing a correction to the solution as in $x_{i+1} = x_i - f(x_i)/f'(x_i)$ or $(x_{i+1} - x_i) = -f(x_i)/f'(x_i)$ this approach can be used. We see solving optimization problems as a natural fit.

References

[1] Alfredo Buttari, Jack Dongarra, Jakub Kurzak, Piotr Luszczek, and Stanimire Tomov. Using mixed precision for sparse matrix computations to enhance the performance while achieving 64-bit accuracy. *ACM Transactions on Mathematical Software*, 1, to appear December 2008.

[2] G. W. Stewart. *Introduction to Matrix Computations*. Academic Press, 1973.

[3] J. W. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997.

[4] J. H. Wilkinson. *Rounding Errors in Algebraic Processes*. Prentice-Hall, 1963.

[5] C. B. Moler. Iterative refinement in floating point. *J. ACM*, 14(2):316–321, 1967.

[6] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 1996.

[7] J. Kurzak and J. J. Dongarra. Implementation of mixed precision in solving systems of linear equations on the CELL processor. *Concurrency Computat. Pract. Exper.* to appear.

[8] J. Kurzak and J. J. Dongarra. Mixed precision dense linear system solver based on cholesky factorization for the CELL processor. *Concurrency Computat. Pract. Exper.* in preparation.

[9] G. W. Stewart. *Matrix algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

A Algorithm and Floating-Point Arithmetic Relations

For the following analisys the iterative refinement algorithm

1: Initialize x_1
   for $k = 1, 2, ...$ **do**:
2:     $r_k = b - Ax_k$ (ε_d)
3:     solve $Ad_k = r_k$ (ε_s)
4:     solve $x_{k+1} = x_k + d_k$ (ε_d)
   end for

performed in floating-point arithmetic is assumed, where the residual r_k (step 2) and the new approximate solution x_{k+1} (step 4) are computed using double precision (ε_d) arithmetic, and the correction vector d_k (step 3) is computed using single precision (ε_s) arithmetic.

Step 3 is performed using a backward stable algorithm (for example Gaussian elimination with partial pivoting, the GMRES method...).

Backward stability implies that there exists H_k such that

$$(A + H_k)d_k = r_k \quad where \quad ||H_k|| \leq \phi(n)\varepsilon_s||A||, \tag{1}$$

where $\phi(n)$ is a reasonably small function of n. In other words, Equation (1) states that the computed solution d_k is an exact solution for an approximated problem.

Steps 2 and 4 are performed in double precision arithmetic and, thus, the classical error bounds hold:

$$r_k = fl(b - Ax) \equiv b - Ax_k + e_k \quad where \quad ||e_k|| \leq \varphi_1(n)\varepsilon_d(||A|| \cdot ||x_k|| + ||b||), \tag{2}$$

$$x_{k+1} = fl(x_k + d_k) \equiv x_k + d_k + f_k \quad where \quad ||f_k|| \leq \varphi_2(n)\varepsilon_d(||x_k|| + ||d_k||). \tag{3}$$

A.1 Results and Interpretation

Using Equations (1) (2) and (3), we will prove in Sectiion A.2 that for any k

$$||x - x_{k+1}|| \leq \alpha_F||x - x_k|| + \beta_F||x||, \tag{4}$$

where α_F and β_F are defined as

$$\alpha_F = \frac{\phi(n)\kappa(A)\varepsilon_s}{1 - \phi(n)\kappa(A)\varepsilon_s} + 2\varphi_1(n)\kappa(A)\varepsilon_d + \varphi_2(n)\varepsilon_d + 2(1 + \varphi_1(n)\varepsilon_d)\varphi_2(n)\kappa(A)\varepsilon_d \tag{5}$$

$$\beta_F = 4\varphi_1(n)\kappa(A)\varepsilon_d + \varphi_2(n)\varepsilon_d + 4(1 + \varphi_1(n)\varepsilon_d)\varphi_2(n)\kappa(A)\varepsilon_d. \tag{6}$$

Note that α_F and β_F are of the form

$$\alpha_F = \psi_F(n)\kappa(A)\varepsilon_s \quad and \quad \beta_F = \rho_F(n)\kappa(A)\varepsilon_d. \tag{7}$$

For Equation (4) to hold, the matrix A is required to be not too ill-conditioned with respect to the single-precision (ε_s) used; specifically the assumption in Equation (8) is made:

$$(\rho_F(n)\kappa(A)\varepsilon_s)(1 - \psi_F(n)\kappa(A)\varepsilon_s)^{-1} < 1 \tag{8}$$

Assuming $\alpha_F < 1$, Equation (9) follows

$$||x - x_{k+1}|| \leq \alpha_F^k||x - x_1|| + \beta_F\frac{1 - \alpha_F^k}{1 - \alpha_F}||x||, \tag{9}$$

and so x_k converges to $\tilde{x} \equiv \lim_{k\to+\infty} x_k$ where

$$\lim_{k\to+\infty} ||x - x_k|| = ||x - \tilde{x}|| \leq \beta_F(1 - \alpha_F)^{-1}||x|| = \frac{\rho_F(n)\kappa(A)\varepsilon_d}{1 - \psi_F(n)\kappa(A)\varepsilon_s}||x||.$$

The term α_F is the rate of convergence and depends on the condition number of the matrix A, $\kappa(A)$, and the single precision used, ε_s. The term β_F is the limiting accuracy of the method and depends on the double precision accuracy used, ε_d.

Regarding the backward error analysis, Section A.3 contains the proof for the following relation:

$$\frac{||b - Ax_{k+1}||}{||A|| \cdot ||x_{k+1}||} \leq \alpha_B \cdot \frac{||b - Ax_k||}{||A|| \cdot ||x_k||} + \beta_B, \tag{10}$$

where

$$\alpha_B = \frac{\phi(n)\kappa(A)\gamma\varepsilon_s}{1 - \phi(n)\kappa(A)\varepsilon_s} + 2\varphi_1(n)\gamma\varepsilon_d, \tag{11}$$

$$\beta_B = (4\varphi_1(n)\gamma + \varphi_2(n)(1 + 2\gamma)(1 - \varphi_2(n)\varepsilon_d)^{-1})\varepsilon_d. \tag{12}$$

Note that α_B and β_B are of the form

$$\alpha_B = \psi_B(n)\kappa(A)\varepsilon_s \quad and \quad \beta_B = \rho_B(n)\varepsilon_d. \tag{13}$$

For Equation (10) to hold it is necessary to assume that the matrix A is not too ill-conditioned with respect to the single precision ε_s arithmetic used; namely, the following assumptions must hold:

$$\psi_F(n)\kappa(A)\varepsilon_s + (\rho_F\kappa(A)\varepsilon_s)(1 - \psi_F(n)\kappa(A)\varepsilon_s)^{-1} < 1 \quad and \tag{14}$$

$$(\rho_B(n)\varepsilon_d)(1 - \psi_B(n)\kappa(A)\varepsilon_s)^{-1} < 1. \tag{15}$$

The term α_B is the speed of convergence and depends on the condition number of the matrix A, $\kappa(A)$ and the single precision used, ε_s. the term β_B is the limiting accuracy of the method and depends on the double precision used, ε_d.

At convergence the following condition holds

$$\lim_{k \to +\infty} \frac{||b - Ax_k||}{||A|| \cdot ||x_k||} = \beta_B(1 - \alpha_B)^{-1} = \frac{\rho_B(n)}{1 - \psi_B(n)\kappa(A)\varepsilon_s}\varepsilon_d \tag{16}$$

which states that the solver is normwise backward stable.

A.2 Forward Error Analysis

From [9] it is possible to prove that, if $\phi(n)\kappa(A)\varepsilon_s < 1/2$, then $(A + H_k)$ is nonsingular and

$$(A + H_k)^{-1} = (I + F_k)A^{-1} \quad where \quad ||F_k|| \leq \frac{\phi(n)\kappa(A)\varepsilon_s}{1 - \phi(n)\kappa(A)\varepsilon_s} < 1. \tag{17}$$

From Equations (1) and (3) it comes

$$x - x_{k+1} = x - x_k - (A + H_k)^{-1}r_k - f_k,$$

and then using Equations (2) and (16)

$$\begin{aligned} x - x_{k+1} &= x - x_k - (I + F_k)A^{-1}(b - Ax_k + e_k) - f_k \\ &= x - x_k - (I + F_k)(x - x_k + A^{-1}e_k) - f_k \\ &= -F_k(x - x_k) - (I + F_k)A^{-1}e_k - f_k. \end{aligned}$$

Taking the norms of both sides of the last equation and using the fact that $||F_k|| < 1$, see Equation (17), we get

$$||x - x_{k+1}|| \leq ||F_k|| \cdot ||x - x_k|| + 2 \cdot ||A^{-1}|| \cdot ||e_k|| + ||f_k||.$$

Using Equations (2) and (3)

$$||x-x_{k+1}|| \leq ||F_k|| \cdot ||x-x_k|| + 2\varphi_1(n)\varepsilon_d ||A^{-1}|| \cdot (||A|| \cdot ||x_k|| + ||b||) + \varphi_2(n)\varepsilon_d(||x_k|| + ||d_k||). \tag{18}$$

Equations (19), (20) and (21) contain a bound for the quantities $||x_k||$, $||A|| \cdot ||x_k|| + ||b||$ and $||d_k||$ by the quantities $||x - x_k||$ and $||x||$. Next step will be to inject these three bounds in Equation (18) which will yield the final result on forward error given in Equation (22).

Triangle inequality yields

$$||x_k|| \leq ||x - x_k|| + ||x||. \tag{19}$$

Then, using the fact that $Ax = b$,

$$||A|| \cdot ||x_k|| + ||b|| \leq ||A|| \cdot ||x - x_k|| + 2 \cdot ||A|| \cdot ||x||. \tag{20}$$

Finally, using Equations (1) and (4)

$$||d_k|| = ||(A + H_k)^{-1} r_k|| = ||(I + F_k)A^{-1} r_k|| \leq 2||A^{-1}|| \cdot ||r_k||.$$

Equation (2) yields

$$||r_k|| \leq ||b|| + ||A|| \cdot ||x_k|| + ||e_k|| \leq (1 + \varphi_1(n)\varepsilon_d) \cdot (||A|| \cdot ||x_k|| + ||b||)$$

which, using Equation (20), can be transformed as

$$||d_k|| \leq 2 \cdot (1 + \varphi_1(n)\varepsilon_d) \cdot \kappa(A) \cdot (||x - x_k|| + 2 \cdot ||x||). \tag{21}$$

Injecting Equations (19), (20) and (21) in Equation (18) yields

$$\begin{aligned} ||x - x_{k+1}|| \quad \leq \quad & \Bigg[\tfrac{\phi(n)\kappa(A)\varepsilon_s}{1-\phi(n)\kappa(A)\varepsilon_s} + 2\varphi_1(n)\kappa(A)\varepsilon_d \\ & +2(1 + \varphi_1(n)\varepsilon_d)\varphi_2(n)\kappa(A)\varepsilon_d \Bigg] \cdot ||x - x_k|| \\ & +(4\varphi_1(n)\kappa(A)\varepsilon_d + \varphi_2(n)\varepsilon_d \\ & +4(1 + \varphi_1(n)\varepsilon_d)\varphi_2(n)\kappa(A)\varepsilon_d) \cdot ||x||. \end{aligned} \tag{22}$$

If α_F and β_F are defined as

$$\alpha_F = \frac{\phi(n)\kappa(A)\varepsilon_s}{1 - \phi(n)\kappa(A)\varepsilon_s} + 2\varphi_1(n)\kappa(A)\varepsilon_d + \varphi_2(n)\varepsilon_d + 2(1 + \varphi_1(n)\varepsilon_d)\varphi_2(n)\kappa(A)\varepsilon_d, \tag{23}$$

$$\beta_F = 4\varphi_1(n)\kappa(A)\varepsilon_d + \varphi_2(n)\varepsilon_d + 4(1 + \varphi_1(n)\varepsilon_d)\varphi_2(n)\kappa(A)\varepsilon_d, \tag{24}$$

then

$$||x - x_{k+1}|| \leq \alpha_F ||x - x_k|| + \beta_F ||x||,$$

where $\alpha_F = \psi(n)\kappa(A)\varepsilon_s$ and $\beta_F = \rho(n)\kappa(A)\varepsilon_d$.

A.2.1 Bound on $||x_k||$ and $||d_k||$ in terms of $||x_{k+1}||$

Assuming, without loss of generality, that $x_1 = 0$, from Equation (9) the following inequalities can be derived

$$||x_k|| \leq \left(1 + \alpha_F^{k-1} + \beta_F \frac{1-\alpha_F^{k-1}}{1-\alpha_F}\right) \cdot ||x||,$$

$$||x|| \leq \left(1 - \alpha_F^{k} - \beta_F \frac{1-\alpha_F^{k}}{1-\alpha_F}\right)^{-1} \cdot ||x_{k+1}||.$$

From the assumption that $\alpha_F + \frac{\beta_F}{1-\alpha_F} < 1$ the following inequality holds

$$||x_k|| \leq \frac{\left(1 + \alpha_F^{k-1} + \beta_F \frac{1-\alpha_F^{k-1}}{1-\alpha_F}\right)}{\left(1 - \alpha_F^{k} - \beta_F \frac{1-\alpha_F^{k}}{1-\alpha_F}\right)} \cdot ||x_{k+1}||$$

and, by defining

$$\gamma_k \equiv \frac{\left(1 + \alpha_F^{k-1} + \beta_F \frac{1-\alpha_F^{k-1}}{1-\alpha_F}\right)}{\left(1 - \alpha_F^{k} - \beta_F \frac{1-\alpha_F^{k}}{1-\alpha_F}\right)} \leq \gamma$$

the following formula is obtained

$$||x_k|| \leq \gamma \cdot ||x_{k+1}||. \tag{25}$$

Equation (3) yields

$$||d_k|| = ||x_{k+1} - x_k - f_k|| \leq ||x_{k+1}|| + (1 + \varphi_2(n)\varepsilon_d)||x_k|| + \varphi_2(n)\varepsilon_d||d_k||,$$

which, in combination with Equation (25) gives

$$||d_k|| \leq (1 - \varphi_2(n)\varepsilon_d)^{-1}(1 + \gamma + \varphi_2(n)\gamma\varepsilon_d)||x_{k+1}||. \tag{26}$$

Note that the assumption

$$\alpha_F + \frac{\beta_F}{1-\alpha_F} < 1 \tag{27}$$

was made in this section.

A.3 Backward Error Analysis

From [9] it is possible to prove that, if $\phi(n)\kappa(A)\varepsilon_s < 1/2$, then $(A + H_k)$ is nonsingular and

$$(A + H_k)^{-1} = A^{-1}(I + G_k) \quad where \quad ||G_k|| \leq \frac{\phi(n)\kappa(A)\varepsilon_s}{1 - \phi(n)\kappa(A)\varepsilon_s} < 1. \tag{28}$$

Equations (1) and (3) yield

$$x - x_{k+1} = x - x + k - (A + H_k)^{-1} r_k - f_k,$$

and, then, using Equations (2) and (28)

$$x - x_{k+1} = x - x_k - A^{-1}(I + G_k)(b - Ax_k + e_k) - f_k.$$

Finally, multiplying both sides by A on the left

$$b - Ax_{k+1} = -G_k(b - Ax_k) - (I + G_k)e_k - Af_k.$$

Taking the norm of both sides and using the fact that $||G_k|| < 1$ gives

$$||b - Ax_{k+1}|| \leq ||G_k|| \cdot ||b - Ax_k|| + 2 \cdot ||e_k|| + ||A|| \cdot ||f_k||.$$

Using Equations (2) and (3) gives

$$\begin{aligned} ||b - Ax_{k+1}|| \quad \leq \quad & ||G_k|| \cdot ||b - Ax_k|| + (2\varphi_1(n) + \varphi_2(n))\varepsilon_d \cdot ||A|| \cdot ||x_k|| \\ & + 2\varphi_1(n)\varepsilon_d \cdot ||b|| + \varphi_2(n)\varepsilon_d \cdot ||A|| \cdot ||d_k||. \end{aligned}$$

Assuming Equation (27) holds, Equations (25) and (26) can be used and, based on the fact that $||b|| = ||b - Ax_k|| + ||A|| \cdot ||x_k||$

$$\begin{aligned} ||b - Ax_{k+1}|| \quad \leq \quad & (||G_k|| + 2\varphi_1(n)\varepsilon_d) \cdot ||b - Ax_k|| \\ & + (4\varphi_1(n)\gamma + \varphi_2(n)\gamma\varphi_2(n)(1 - \varphi_2(n)\varepsilon_d)^{-1}(1 + \gamma + \varphi_2(n)\gamma\varepsilon_d))\varepsilon_d \cdot ||A|| \cdot ||x_{k+1}||. \end{aligned}$$

Finally

$$\frac{||b - Ax_{k+1}||}{||A|| \cdot ||x_{k+1}||} \leq \alpha_B \cdot \frac{||b - Ax_k||}{||A|| \cdot ||x_k||} + \beta_B,$$

where

$$\alpha_B = \frac{\phi(n)\kappa(A)\gamma\varepsilon_s}{1 - \phi(n)\kappa(A)\varepsilon_s} + 2\varphi_1(n)\gamma\varepsilon_d,$$

$$\beta_B = (4\varphi_1(n)\gamma + \varphi_2(n)(1 + 2\gamma)(1 - \varphi_2(n)\varepsilon_d)^{-1})\varepsilon_d.$$