# PSPIKE: A Parallel Hybrid Sparse Linear System Solver*

Murat Manguoglu[1], Ahmed H. Sameh[1], and Olaf Schenk[2]

[1] Department of Computer Science, Purdue University, West Lafayette IN 47907
[2] Computer Science Department, University of Basel, Klingelbergstrasse 50, CH-4056 Basel

**Abstract.** The availability of large-scale computing platforms comprised of tens of thousands of multicore processors motivates the need for the next generation of highly scalable sparse linear system solvers. These solvers must optimize parallel performance, processor (serial) performance, as well as memory requirements, while being robust across broad classes of applications and systems. In this paper, we present a new parallel solver that combines the desirable characteristics of direct methods (robustness) and effective iterative solvers (low computational cost), while alleviating their drawbacks (memory requirements, lack of robustness). Our proposed hybrid solver is based on the general sparse solver PARDISO, and the "Spike" family of hybrid solvers. The resulting algorithm, called PSPIKE, is as robust as direct solvers, more reliable than classical preconditioned Krylov subspace methods, and much more scalable than direct sparse solvers. We support our performance and parallel scalability claims using detailed experimental studies and comparison with direct solvers, as well as classical preconditioned Krylov methods.

**Key words:** Hybrid Solvers, Direct Solvers, Krylov Subspace Methods, Sparse Linear Systems.

## 1 Introduction

The emergence of extreme-scale parallel platforms, along with increasing number of cores available in conventional processors pose significant challenges for algorithm and software development. Machines with tens of thousands of processors and beyond place tremendous constraints on the communication requirements of algorithms. This is largely due to the fact that the restricted memory scaling model (generally believed to be no more than linear in the number of processing cores) allows for limited problem scaling. This in turn limits the amount of overhead that can be amortized across useful work in the program.

Increase in the number of cores in a processing unit without a commensurate increase in memory bandwidth, on the other hand, exacerbates an already significant memory bottleneck. Sparse algebra kernels are well-known for their poor

* This work was supported by grants from NSF (NSF-CCF-0635169), DARPA/AFRL (FA8750-06-1-0233), and a gift from Intel and partially supported by the Swiss National Science Foundation under grant 200021 − 117745/1.

processor utilization (typically in the range of 10 - 20% of processor peak). This is a result of limited memory reuse, which renders data caching less effective. Attempts at threading these computations rely on concurrency to trade off memory bandwidth for latency. By allowing multiple outstanding reads, these computations hide high latency of access. However, this is predicated on the ability of the memory system to handle multiple requests concurrently. In view of these emerging hardware trends, it is necessary to develop algorithms and software that strike a more meaningful balance between memory accesses, communication, and computation. Specifically, an algorithm that performs more floating point operations at the expense of reduced memory accesses and communication is likely to yield better performance.

This paper addresses the problem of developing robust, efficient, and scalable sparse linear system solvers. Traditional approaches to linear solvers rely on direct or iterative methods. Direct methods are known to be robust – i.e., they are capable of handling relatively ill-conditioned linear systems. However, their memory requirements and operation counts are typically higher. Furthermore, their scaling characteristics on very large numbers of processors while maintaining robustness, remain unresolved. Iterative methods, on the other hand, have lower operation counts and memory requirements, but are not as robust.

In this paper, we present a new parallel solver that incorporates the desirable characteristics of direct and iterative solvers while alleviating their drawbacks. Specifically, it is robust in the face of ill-conditioning, has lower memory references, and is amenable to highly efficient parallel implementation.

We first discuss the basic Spike solver for banded systems and PSPIKE (also known as Spike-PARDISO) hybrid solver for general sparse linear systems. Next, we demonstrate the scalability and performance of our solver on up to 256 cores of a distributed memory platform for a variety of applications. Finally, we enhance the robustness of our method through symmetric and nonsymmetric weighted reordering. We compare our methods to commonly used incomplete LU (ILU) factorization-based preconditioners, and direct solvers to demonstrate superior performance characteristics of our solver. We would like to note that there has been several attempts to design hybrid algorithms in the past based on either classical LU factorization or the schur complement method. Our approach, however, is different and exhibits superior scalability.

The test platform for our experiments is a cluster of dual quad-core Intel Xeon E5462 nodes, with each core operating at 2.8 GHz. The nodes are interconnected via Infiniband.

## 2 The Basic Spike Algorithm

Let $Ax = f$ be a nonsymmetric diagonally dominant system of linear equations where $A$ is of order $n$ and bandwidth $2m + 1$. Unlike classical banded solvers such as those in LAPACK that are based on LU-factorization of $A$, the spike algorithm [1,2,3,4,5,6,7] is based on the factorization $A = D \times S$, where $D$ is a block-diagonal matrix and $S$ is the spike matrix shown in Figure 1 for three

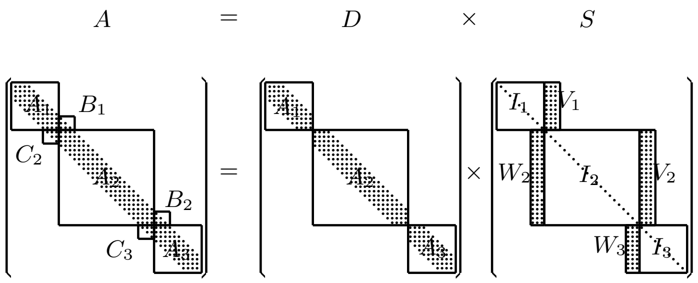

**Fig. 1.** Spike decomposition where $A = D * S, S = D^{-1}A$, $B_j, C_j \in \mathbb{R}^{m\times m}$

partitions. Note that the block diagonal matrices $A_1$, $A_2$, and $A_3$ are nonsingular by virtue of the diagonal dominance of $A$. For the example in Figure 1, the basic Spike algorithm consists of the following stages:

– **Stage 1:** Obtain the LU-factorization (without pivoting) of the diagonal blocks $A_j$ (i.e. $A_j = L_jU_j,\ j = 1, 2, 3$ )
– **Stage2:** Forming the spike matrix S and updating the right hand side
  (i) solve $L_1U_1[V_1, g_1] = [(\begin{smallmatrix} 0 \\ B_1 \end{smallmatrix}), f_1]$
  (ii) solve $L_2U_2[W_2, V_2, g_2] = [(\begin{smallmatrix} C_2 \\ 0 \end{smallmatrix}), (\begin{smallmatrix} 0 \\ B_2 \end{smallmatrix}), f_2]$
  (iii) solve $L_3U_3[W_3, g_3] = [(\begin{smallmatrix} C_3 \\ 0 \end{smallmatrix}), f_3]$
  $f_j$, $i \le j \le 3$, are the corresponding partitions of the right hand side $f$.
– **Stage 3:** Solving the reduced system,

$$\begin{bmatrix} I & V_1^{(b)} & 0 & 0 \\ W_2^{(t)} & I & 0 & V_2^{(t)} \\ W_2^{(b)} & 0 & I & V_2^{(b)} \\ 0 & 0 & W_3^{(t)} & I \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} = \begin{bmatrix} g_1^{(b)} \\ g_2^{(t)} \\ g_2^{(b)} \\ g_3^{(t)} \end{bmatrix} \tag{1}$$

where $(V_i^{(b)}, W_i^{(b)})$ and $(V_i^{(t)}, W_i^{(t)})$ are the bottom and top $m \times m$ blocks of $(V_i, W_i)$, respectively. Similarly, $g_i^{(b)}$ and $g_i^{(t)}$ are the bottom and top $m$ elements of each $g_i$. Once this smaller reduced system is solved the three partitions $x_i$, $i = 1, 2, 3$, of the solution $x$ are obtained as follows:

(i) $x_1 = g_1 - V_1z_2$
(ii) $x_2 = g_2 - W_2z_1 - V_2z_4$
(iii) $x_3 = g_3 - W_3z_3$

Clearly, the above basic scheme may be made more efficient if in stage 2 one can generate only the bottom and top $m \times m$ tips of the spikes $V_i$ and $W_i$, as well as the corresponding bottom and top tips of $g_i$. In this case, once the reduced system is solved, solving the system $Ax = f$ is reduced to solving the independent systems:

(i) $L_1U_1x_1 = f_1 - B_1z_2$
(ii) $L_2U_2x_2 = f_2 - C_2z_1 - B_2z_4$
(iii) $L_3U_3x_3 = f_3 - C_3z_3$

If the matrix $A$ is not diagonally dominant, we cannot guarantee that the diagonal blocks, $A_i$, are nonsingular. However, if we obtain the LU-factorization, without pivoting, of each block $A_i$ using diagonal boosting (perturbation), then

$$L_iU_i = (A_i + \delta A_i) \tag{2}$$

in which $||\delta A_i|| = \mathcal{O}(\epsilon||A_i||)$, where $\epsilon$ is the unit roundoff. In this case, we will need to solve $Ax = f$ using an outer iterative scheme with the preconditioner being the matrix $M$ that is identical to $A$, except that each diagonal block $A_i$ is replaced by $L_iU_i$ in Equation 2. These systems, of the form $My = r$, are solved using the Spike scheme outlined above.

## 3 The PSPIKE Scheme

PARDISO [8,9] is a state-of-the-art direct sparse linear system solver. Its performance and robustness have been demonstrated in the context of several applications. The hybrid PSPIKE scheme can be used for solving general sparse systems as follows. If the elements of the coefficient matrix naturally decay as one moves away from the main diagonal we do not need to reorder the linear system. Otherwise, the sparse matrix $A$ is reordered via a nonsymmetric reordering scheme, which ensures that none of the diagonal elements are zero, followed by a weighted reordering scheme that attempts to move as many of the largest elements as possible to within a narrow central band, $M$. This banded matrix, $M$ is used as a preconditioner for an outer iterative scheme, e.g., BiCGStab [10], for solving $Ax = f$. The major operations in each iteration are: (i) matrix-vector products, and (ii) solving systems of the form $My = r$.

Solving systems $My = r$ involving the preconditioner is accomplished using our proposed algorithm: PSPIKE as follows: the LU-factorization of each diagonal block partition $M_i$ (banded and sparse within the band) is obtained using PARDISO with supernodal pivoting and weighted graph matchings. The most recent version of PARDISO is also capable of obtaining the top and bottom tips of the left and right spikes $\hat{W}_i$ and $\hat{V}_i$, as well as the corresponding tips of the updated right hand side subvectors. Further, having the largest elements within the band, whether induced by the reordering or occurring naturally, allows us to approximate (or truncate) the resulting reduced system by its block diagonal, $\hat{S} = diag(\hat{S}_1, \hat{S}_2, ..., \hat{S}_p)$, where $p$ is the number of partitions and,

$$\hat{S}_j = \begin{bmatrix} I & \hat{V}_j^{(b)} \\ \hat{W}_{j+1}^{(t)} & I \end{bmatrix}. \tag{3}$$

This also enhances concurrency, especially when $M$ has a large bandwidth. For a more detailed discussion of the decay rate of spikes and the truncated Spike algorithm we refer the reader to [11]. In the following section, we will demonstrate the suitability of the PSPIKE solver for implementation on clusters consisting of several nodes in which each node is a multicore processor. While PARDISO is primarily suitable for single node platforms, PSPIKE is scalable across multiple nodes. We would also like to mention that our approach is suitable for MPI/OpenMP hybrid parallelism where each node can use the threading capability of PARDISO, as well as pure MPI parallelism where no threads are used. We give example of both programming paradigms in the following sections. The number of partitions in our implementation is the same as the number of MPI processes.

## 4 A Weighted Reordering Scheme

The decay of the absolute value of the matrix elements as one moves away from the main diagonal is essential for the convergence of the PSPIKE scheme. There are, however, linear systems which do not immediately possess this property. In fact the same system may or may not have decaying elements based on the reordering. We propose an additional layer of weighted symmetric/nonsymmetric reordering in order to enhance the robustness of the PSPIKE scheme for such systems. We will call this preprocessing method as the Weighted Reordering method(WR) in the rest of the paper.

Given a linear system $Ax = f$, we first apply a nonsymmetric row permutation as follows: $QAx = Qf$. Here, $Q$ is the row permutation matrix that either maximizes the number of nonzeros on the diagonal of $A$ [12] , or the permutation that maximizes the product of the absolute values of the diagonal entries [13]. The first algorithm is known as the scheme for the *maximum traversal search*. Both algorithms are implemented in the MC64 subroutine of the HSL[14] library.

Following the above nonsymmetric reordering and optional scaling, we apply the symmetric permutation $P$ as follows:

$$(PQAP^T)(Px) = (PQf). \tag{4}$$

We use HSL-MC73 [15] weighted spectral reordering to obtain the symmetric permutation $P$ that minimizes the bandwidth encapsulating a specified fraction of the total magnitude of nonzeros in the matrix. This permutation is determined from the symmetrized matrix $|A| + |A^T|$.

In the following sections the preconditioner we extract is treated as either dense or sparse within the band.

## 5 Banded Preconditioners (Dense within the Band)

In this section we study the convergence characteristics of various solvers on a uniprocessor. This set of problems consists of symmetric and nonsymmetric general sparse matrices from the University of Florida [16]. For each matrix we generated the corresponding right hand-side using a solution vector of all ones to ensure that $f \in span(A)$. The number $k$ represents the bandwidth of the preconditioner detected by our method. The linear systems involving the

**Table 1.** Properties of Test Matrices

| Matrix | k | Dimension(N) | Non-zeros(nnz) | Type |
|---|---|---|---|---|
| 1.FINAN512 | 50 | $74,752$ | $596,992$ | Financial Optimization |
| 2.FEM_3D_THERMAL1 | 50 | $17,880$ | $430,740$ | 3D Thermal FEM |
| 3.RAJAT31 | 30 | $4,690,002$ | $20,316,253$ | Circuit Simulation |
| 4.H2O | 50 | $67,024$ | $2,216,736$ | Quantum Chemistry |
| 5.APPU | 50 | $14,000$ | $1,853,104$ | NASA Benchmark |
| 6.BUNDLE1 | 50 | $10,581$ | $770,811$ | 3D Computer Vision |
| 7.RAJAT30 | 30 | $643,994$ | $6,175,244$ | Circuit Simulation |
| 8.DW8191 | 182 | $8,192$ | $41,746$ | Dielectric Waveguide |
| 9.DC1 | 50 | $116,835$ | $766,396$ | Circuit Simulation |
| 10.FP | 2 | $7,548$ | $834,222$ | Electromagnetics |
| 11.PRE2 | 30 | $659,033$ | $5,959,282$ | Harmonic Balance Method |
| 12.KKT_POWER | 30 | $2,063,494$ | $14,612,663$ | Nonlinear Optimization |
| 13.RAEFSKY4 | 50 | $19,779$ | $1,328,611$ | Structural Mechanics |
| 14.ASIC_680k | 3 | $682,862$ | $3,871,773$ | Circuit Simulation |
| 15.2D_54019_HIGHK | 2 | $54,019$ | $996,414$ | Device Simulation |

preconditioner are solved via LAPACK. After obtaining the reordered system via WR, we determine the half-bandwidth of the preconditioner ($k$) such that 99.99% of the total weight of the matrix is encapsulated within the band. If $n > 10,000$ and $n > 500,000$ we enforce upper limits of 50 and 30, respectively for $k$. For establishing a competitive baseline, we obtain an ILUT preconditioner via the maximum product traversal and scaling proposed by Benzi et al. [17] (ILUTI). Furthermore, we use PARDISO with the METIS [18] reordering and enable the nonsymmetric ordering option for indefinite systems. For ILUPACK [19], we use

**Table 2.** Total solve time for ILUTI, WR, ILUPACK, PARDISO and RCM methods

| | | *Banded* | | *LU based* | | |
|---|---|---|---|---|---|---|
| Matrix | Condest | WR | RCM | ILUTI(*,$10^{-1}$) | PARDISO | ILUPACK |
| 1 | $9.8 \times 10^{1}$ | 0.87 | 0.29 | 0.14 | 1.28 | 0.5 |
| 2 | $1.7 \times 10^{3}$ | 0.41 | 0.19 | 0.19 | 1.48 | 0.23 |
| 3 | $4.4 \times 10^{3}$ | 457.2 | F | 193.5 | 91.6 | F |
| 4 | $4.9 \times 10^{3}$ | 5.63 | 4.89 | 0.84 | 450.73 | 2.86 |
| 5 | $1.0 \times 10^{4}$ | 0.82 | 0.51 | 0.88 | 270.17 | 44.6 |
| 6 | $1.3 \times 10^{4}$ | 0.7 | 0.14 | 0.27 | 0.88 | 0.27 |
| 7 | $8.7 \times 10^{4}$ | 205.4 | F | 459.1 | 7.6 | F |
| 8 | $1.5 \times 10^{7}$ | 0.33 | 0.07 | F | 0.78 | F |
| 9 | $1.1 \times 10^{10}$ | 1.95 | 8.99 | 15.8 | 3.14 | 2.07 |
| 10 | $8.2 \times 10^{12}$ | 0.42 | 0.04 | F | 1.74 | 0.46 |
| 11 | $3.3 \times 10^{13}$ | 7.0 | F | F | 45.3 | F |
| 12 | $4.2 \times 10^{13}$ | F | F | F | 449.1 | F |
| 13 | $1.5 \times 10^{14}$ | 0.27 | 0.09 | 0.59 | 1.28 | F |
| 14 | $9.4 \times 10^{19}$ | 2.0 | F | 239.0 | 27.2 | F |
| 15 | $8.1 \times 10^{32}$ | 0.21 | 0.06 | 0.11 | 0.95 | 1.44 |

the PQ reordering option and a drop tolerance of $10^{-1}$ with a bound on the condition number parameter 50 (recommended in the user documentation for general problems). In addition, we enable options in ILUPACK to use matchings and scalings (similar to MC64). The BiCGStab iterations for solving systems are terminated when $||\hat{r}_k||_\infty/||\hat{r}_0||_\infty < 10^{-5}$.

In Table 2, we show the total solution time of each algorithm including the direct solver PARDISO. A failure indicated by F represent the method either failed during the factorization or iterative solution stages. We note that incomplete LU based methods (ILUTI and ILUPACK) fails in more cases than WR. While PARDISO is the most robust method for the set of problems, in terms of total solve time it is faster than WR only in two cases. Incomplete LU factorization based preconditioners are faster than banded preconditioners(WR and RCM) for problems that are well conditioned (condest $\leq 3.3 \times 10^4$).

## 6 PSPIKE for Solving General Sparse Linear Systems (Sparse within the Band)

We apply the PSPIKE scheme on four large general sparse systems (Table 3). Two of the matrices are obtained from the UF Sparse matrix collection, while the other two are from a nonlinear optimization package [20]. The timings we report

**Table 3.** Selection of Test Matrices

| Name | Application | unknowns | nonzeros | symmetry |
|---|---|---|---|---|
| MIPS50S1 | Optimization | 235, 592 | 1, 009, 736 | symmetric |
| MIPS80S1 | Optimization | 986, 552 | 4, 308, 416 | symmetric |
| G3_CIRCUIT | Circuit Simulation | 1, 585, 478 | 4, 623, 152 | symmetric |
| CAGE14 | DNA Model | 1, 505, 785 | 27, 130, 349 | general |

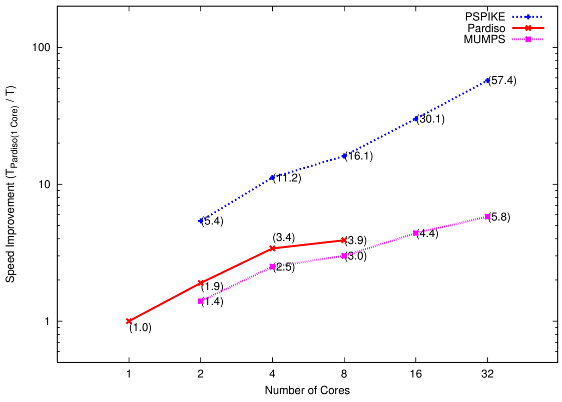

**Fig. 2.** MIPS50S1: The speed improvement compared to PARDISO using one core (70.4 seconds)

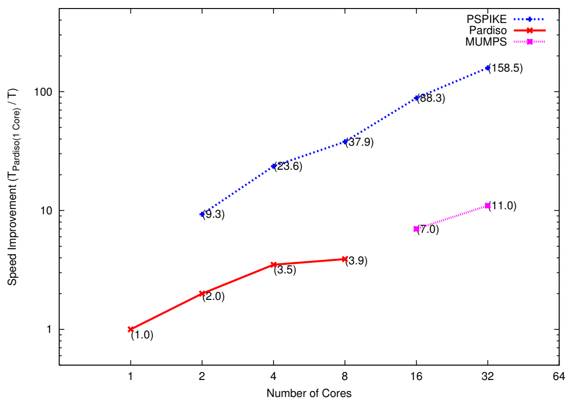

**Fig. 3.** MIPS80S1: The speed improvement compared to PARDISO using one core $(1,460.2$ seconds)

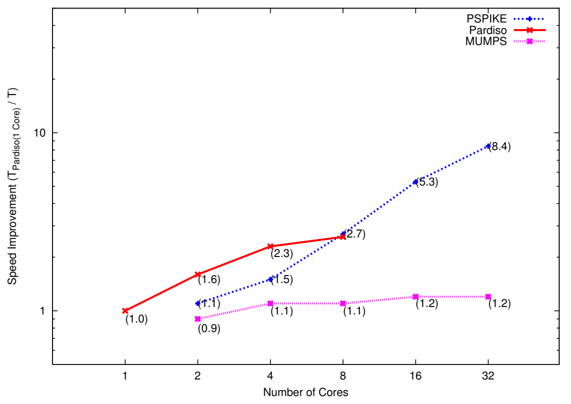

**Fig. 4.** G3_CIRCUIT: The speed improvement compared to PARDISO using one core (73.5 seconds)

are total times, including reordering, symbolic factorization, factorization, and solution. In these tests, we use METIS reordering for both MUMPS [21,22,23] and PARDISO. For PSPIKE, We use the same number of MPI processes as the number of cores and 1 thread per MPI process. We use BiCGStab with a banded preconditioner to solve these systems where the bandwidth of the preconditioner is 101. We apply HSL's MC64 reordering to maximize the absolute value of the products of elements on the main diagonal [13]. The BiCGStab iterations for solving systems are terminated when $||\hat{r}_k||_\infty/||\hat{r}_0||_\infty < 10^{-5}$. We note that the reason we use BiCGStab for the symmetric systems is that the coefficient matrix becomes nonsymmetric after using the nonsymmetric reordering (MC64). In Figures 2 and 3, we show the speed improvement realized by the PSPIKE scheme compared to the uniprocessor time spent in PARDISO for two nonlinear optimization problems. We note that the speed improvement of PSPIKE is

**Table 4.** Total Solve Time for CAGE14 using PSPIKE

| Number of Cores | 2 | 4 | 8 | 16 | 32 |
|---|---|---|---|---|---|
| Time(seconds) | 35.3 | 30.9 | 21.7 | 8.7 | 3.9 |

enhanced as the problem size increases. Furthermore, MUMPS runs out of memory for the larger problem if less than 32 cores (2 nodes) are used.

Figure 4 shows the speed improvement for the circuit simulation problem. Unlike the other two problems, MUMPS spends significant portion of the time in the reordering/symbolic factorization stage, which is not scalable. CAGE14 has the largest number of nonzeros per row among the four problems. As a result, both MUMPS and PARDISO run out of memory due to large fillin. The PSPIKE scheme, on the other hand, can solve this system. In Table 4, we present the total solve times for PSPIKE. The superlinear speed improvement we observe is due to the fact that as smaller blocks are factored via PARDISO, the fill-in improves much faster than the reduction in the matrix dimension.

## 7 PSPIKE for Solving Linear Systems Arising in a PDE-Constrained Optimization Problem (Sparse within the Band)

In this section we propose a block factorization based scheme for solving linear systems that arise in a PDE-constrained optimization problem [24]. In which the inner linear solves are accomplished by PSPIKE. Linear systems extracted from a nonlinear solve have the following block structure:

$$\begin{bmatrix} D & & B^T \\ & H & C^T \\ B & C & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} \tag{5}$$

Here, $D \in \mathbb{R}^{n \times n}$ is diagonal and $D_{ii} > 0$ for $i = 1, 2, ..., n$. Furthermore $H \in \mathbb{R}^{k \times k}$ is symmetric positive definite and $C \in \mathbb{R}^{k \times n}$ is dense with $k << n$. $B \in \mathbb{R}^{n \times n}$ is nonsymmetric banded and sparse within the band.

Premultiplying the above equation by $\begin{bmatrix} D^{-1} & & \\ & H^{-1} & \\ & & I \end{bmatrix}$ we get

$$\begin{bmatrix} I & & \tilde{B}^T \\ & I & \tilde{C}^T \\ B & C & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \hat{f}_1 \\ \hat{f}_2 \\ f_3 \end{bmatrix} \tag{6}$$

where $\tilde{B}^T = D^{-1}B^T$, $\tilde{C}^T = H^{-1}C^T$, $\hat{f}_1 = D^{-1}f_1$ and $\hat{f}_2 = H^{-1}f_2$. Rearranging the rows and columns, we have the system,

$$\begin{bmatrix} I & \tilde{B}^T & 0 \\ B & 0 & C \\ 0 & \tilde{C}^T & I \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \\ x_2 \end{bmatrix} = \begin{bmatrix} \hat{f}_1 \\ f_3 \\ \hat{f}_2. \end{bmatrix} \tag{7}$$

Observing that

$$\begin{bmatrix} I & \tilde{B}^T \\ B & 0 \end{bmatrix}^{-1} = \begin{bmatrix} 0 & B^{-1} \\ \tilde{B}^{-T} & -\tilde{B}^{-T}B^{-1} \end{bmatrix} \tag{8}$$

we can see that $x_2$ can be obtained by solving the small system,

$$(H + J^T DJ)x_2 = (f_2 - J^T f_1 + J^T Db) \tag{9}$$

in where $J$ and $b$ are obtained by solving $BJ = C$ and $Bb = f_3$, respectively. Consequently, $x_1$ and $x_2$ can be computed via $x_1 = b - Jx_2$ and $x_3 = B^{-T}(f_1 - Dx_1)$. The solution process requires solving linear systems with the coefficient matrices $B^T$ and $B$. We use BiCGStab with a banded preconditioner to solve these systems, in which the systems involving the preconditioner are solved via the PSPIKE scheme.

The BiCGStab iterations for solving systems involving $B^T$ and $B$ are terminated when $||\hat{r}_k||_\infty/||\hat{r}_0||_\infty < \epsilon_{in}$, where $\hat{r}_0$ and $\hat{r}_k$ correspond to the initial residual and the residual at $k^{th}$ iteration, respectively, and the systems involving the preconditioners are solved via the PSPIKE scheme.

We considered linear systems extracted from the first, tenth (middle), and twenty first (last) iterations. However, since the results are uniform across these three systems, we present in Figure 5 the results for the linear system of the tenth Newton iteration. Matrix dimensions are $n = 702, 907$, $k = 23$ and the bandwidth of the preconditioner is 21. Figure 5 illustrates the speed improvement for an MPI/OpenMP hybrid implementation (8 cores(threads) per MPI processes) of the PSPIKE scheme. We demonstrate the speed improvement of the PSPIKE scheme for two stopping criteria, $10^{-5}$ and $10^{-7}$. The corresponding final relative residuals are $\mathcal{O}(10^{-5})$ and $\mathcal{O}(10^{-7})$, respectively. For PARDISO and MUMPS the final relative residuals are $\mathcal{O}(10^{-12})$.

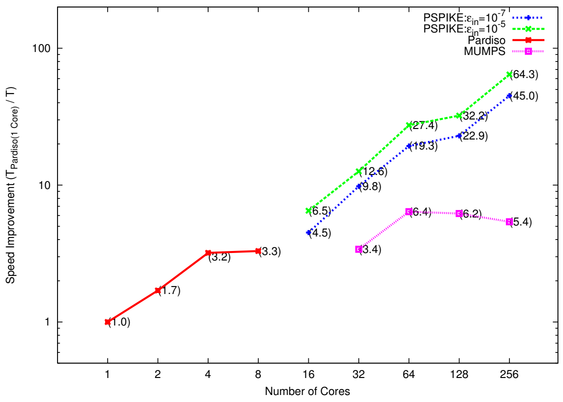

**Fig. 5.** The speed improvement compared to PARDISO using one core (1338 seconds)

## 8 Conclusions

We have demonstrated that our PSPIKE solver is highly efficient and scalable on multicore/message passing platforms. We also introduce the notion of weighted symmetric/nonsymmetric reordering to further enhance the robustness of the PSPIKE scheme. We demonstrate this enhanced robustness using a wide variety of test matrices.

**Acknowledgments.** We thank Ananth Grama and Mehmet Koyuturk for providing WR method and many useful comments, and Patrick Amestoy for recommendations that helped us in using the parallel direct solver MUMPS.

## References

1. Chen, S.C., Kuck, D.J., Sameh, A.H.: Practical parallel band triangular system solvers. ACM Transactions on Mathematical Software 4(3), 270–277 (1978)
2. Lawrie, D.H., Sameh, A.H.: The computation and communication complexity of a parallel banded system solver. ACM Trans. Math. Softw. 10(2), 185–195 (1984)
3. Berry, M.W., Sameh, A.: Multiprocessor schemes for solving block tridiagonal linear systems. The International Journal of Supercomputer Applications 1(3), 37–57 (1988)
4. Dongarra, J.J., Sameh, A.H.: On some parallel banded system solvers. Parallel Computing 1(3), 223–235 (1984)
5. Polizzi, E., Sameh, A.H.: A parallel hybrid banded system solver: the spike algorithm. Parallel Comput. 32(2), 177–194 (2006)
6. Polizzi, E., Sameh, A.H.: Spike: A parallel environment for solving banded linear systems. Computers & Fluids 36(1), 113–120 (2007)
7. Sameh, A.H., Sarin, V.: Hybrid parallel linear system solvers. Inter. J. of Comp. Fluid Dynamics 12, 213–223 (1999)
8. Schenk, O., Gärtner, K.: Solving unsymmetric sparse systems of linear equations with pardiso. Future Generation Computer Systems 20(3), 475–487 (2004) Selected numerical algorithms
9. Schenk, O., Gärtner, K.: On fast factorization pivoting methods for sparse symmetric indefinite systems. Electronic Transactions on Numerical Analysis 23, 158–179 (2006)
10. van der Vorst, H.A.: Bi-cgstab: a fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. SIAM J. Sci. Stat. Comput. 13(2), 631–644 (1992)
11. Mikkelsen, C.C.K., Manguoglu, M.: Analysis of the truncated spike algorithm. SIAM Journal on Matrix Analysis and Applications 30(4), 1500–1519 (2008)
12. Duff, I.S.: Algorithm 575: Permutations for a zero-free diagonal [f1]. ACM Trans. Math. Softw. 7(3), 387–390 (1981)
13. Duff, I.S., Koster, J.: The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. SIAM Journal on Matrix Analysis and Applications 20(4), 889–901 (1999)
14. HSL: A collection of Fortran codes for large-scale scientific computation (2004), `http://www.cse.scitech.ac.uk/nag/hsl/`

15. Hu, Y., Scott, J.: HSL_MC73: a fast multilevel Fiedler and profile reduction code. Technical Report RAL-TR-2003-036 (2003)
16. Davis, T.A.: University of Florida sparse matrix collection. NA Digest (1997)
17. Benzi, M., Haws, J.C., Tuma, M.: Preconditioning highly indefinite and nonsymmetric matrices. SIAM J. Sci. Comput. 22(4), 1333–1353 (2000)
18. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. 20(1), 359–392 (1998)
19. Bollhöfer, M., Saad, Y., Schenk, O.: ILUPACK Volume 2.1—Preconditioning Software Package (May 2006), `http://ilupack.tu-bs.de`
20. Wächter, A., Biegler, L.T.: On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. Mathematical Programming 106(1), 25–57 (2006)
21. Amestoy, P.R., Guermouche, A., L'Excellent, J.Y., Pralet, S.: Hybrid scheduling for the parallel solution of linear systems. Parallel Comput. 32(2), 136–156 (2006)
22. Amestoy, P.R., Duff, I.S., L'Excellent, J.Y., Koster, J.: A fully asynchronous multifrontal solver using distributed dynamic scheduling. SIAM J. Matrix Anal. Appl. 23(1), 15–41 (2001)
23. Amestoy, P.R., Duff, I.S.: Multifrontal parallel distributed symmetric and unsymmetric solvers. Comput. Methods Appl. Mech. Eng. 184, 501–520 (2000)
24. Schenk, O., Manguoglu, M., Sameh, A., Christian, M., Sathe, M.: Parallel scalable PDE-constrained optimization: antenna identification in hyperthermia cancer treatment planning. Computer Science - Research and Development 23(3), 177–183 (2009)