Scientific Programming 19 (2011) 13–25
DOI 10.3233/SPR-2011-0316
IOS Press

Performance models for the Spike banded linear system solver

Murat Manguoglu [a], Faisal Saied [b], Ahmed Sameh [b] and Ananth Grama [b]
[a] *Department of Computer Engineering, Middle East Technical University, Ankara, Turkey*
E-mail: manguoglu@ceng.metu.edu.tr
[b] *Department of Computer Science, Purdue University, West Lafayette, IN, USA*

Abstract. With availability of large-scale parallel platforms comprised of tens-of-thousands of processors and beyond, there is significant impetus for the development of scalable parallel sparse linear system solvers and preconditioners. An integral part of this design process is the development of performance models capable of predicting performance and providing accurate cost models for the solvers and preconditioners. There has been some work in the past on characterizing performance of the iterative solvers themselves. In this paper, we investigate the problem of characterizing performance and scalability of banded preconditioners. Recent work has demonstrated the superior convergence properties and robustness of banded preconditioners, compared to state-of-the-art ILU family of preconditioners as well as algebraic multigrid preconditioners. Furthermore, when used in conjunction with efficient banded solvers, banded preconditioners are capable of significantly faster time-to-solution. Our banded solver, the Truncated Spike algorithm is specifically designed for parallel performance and tolerance to deep memory hierarchies. Its regular structure is also highly amenable to accurate performance characterization. Using these characteristics, we derive the following results in this paper: (i) we develop parallel formulations of the Truncated Spike solver, (ii) we develop a highly accurate pseudo-analytical parallel performance model for our solver, (iii) we show excellent predication capabilities of our model – based on which we argue the high scalability of our solver. Our pseudo-analytical performance model is based on analytical performance characterization of each phase of our solver. These analytical models are then parameterized using actual runtime information on target platforms. An important consequence of our performance models is that they reveal underlying performance bottlenecks in both serial and parallel formulations. All of our results are validated on diverse heterogeneous multi-clusters – platforms for which performance prediction is particularly challenging. Finally, we provide predict the scalability of the Spike algorithm using up to 65,536 cores with our model. In this paper we extend the results presented in the Ninth International Symposium on Parallel and Distributed Computing.

Keywords: Parallel, scalability, performance, prediction, model, banded

1. Introduction and motivation

With the availability of petascale computing platforms, the long-held vision of highly scalable parallel platforms is now being realized. While the challenges associated with the development of algorithms and software that can utilize these platforms have been articulated, the task of programming parallel ensembles with up to 100K processing cores must now be accomplished in the context of complex scientific and engineering applications. This extreme task requires intricate algorithm and software design, coupled with accurate analytical modeling. Programs that "appear" to scale to moderate configurations of 10K cores provide no guarantees of performance when the number of cores is scaled by an order of magnitude. Empirical performance extrapolation is fraught with significant errors and inaccuracies. In this context, accurate prediction of scalability and performance extrapolation before comprehensive implementation are essential components of algorithm and software development.

With respect to kernel algorithms, sparse linear system solvers are among the most ubiquitous and computationally expensive kernels in scientific and engineering applications. The irregular structure of these computations, both in terms of memory accesses, as well as associated FLOP counts, makes their performance notoriously difficult to characterize with meaningful accuracy. Furthermore, these kernels often yield low processor utilization, in the range of 10–15% of peak. While there exist several parallel libraries of sparse linear system solvers with demonstrated scalability to thousands of processors, there exist few results

on analytical extrapolation of performance to petascale platforms. The need for powerful preconditioning techniques for iterative solvers poses fundamental considerations for parallel scalability of the solvers.

Motivated by need for extreme scalability and the deep memory hierarchies of current platforms, we have developed the next generation of hybrid solvers – the Spike family of algorithms [3,5,10,22,29–32]. The Spike solver toolkit is specifically designed for banded systems (potentially sparse within the band). By using a block-diagonal/Spike decomposition, it is inherently parallel. By trading off FLOPS for memory references, it delivers much better processor performance as well. This combination results in an overall parallel performance improvement of up to an order of magnitude compared to state-of-the-art solvers in ScaLapack [4]. Other alternatives are based on approximate inverse preconditioners (e.g., see [15,18]).

Spike has also been extended to the solution of general sparse linear systems (e.g., see [24,26,27,37]). This method relies on the extraction of a narrow band from a general sparse matrix, for use as a preconditioner. In contrast to existing envelope reduction techniques, our method brings the heavy elements in the matrix closer to the diagonal – thus resulting in an effective preconditioner. The resulting banded preconditioned system is solved using the parallel Spike algorithm, wrapped inside an iterative solver. We show the resulting iterative solver is significantly better in terms of robustness and time-to-solution compared to a wide range of Incomplete LU-factorization (ILU) preconditioners and parallel algebraic multigrid preconditioners for broad classes of matrices. In addition to preconditioning, iterative schemes require sparse matrix–vector multiplication and often inner products.

An important aspect of Spike is the use of dense kernels within its banded solve. In addition to yielding excellent performance through use of optimized BLAS kernels, this also render the algorithm amenable to analytical characterization. Specifically, the algorithm can be broken down into a sequence of steps, the performance of each can be concisely captured by associated asymptotic expressions. The interleaved communication in parallel formulation of Spike can be parameterized for target architectures, potentially resulting in accurate parallel performance models.

Based on these promising results, in this paper, we aim to demonstrate the scalability characteristics of the Spike solver toolkit. Specifically, we rely on an analytical characterization of various steps associated with parallel Spike solve. Using this analytical characterization, we build a highly accurate performance model. We show that the performance model yields excellent predictive results across wide ranges of machine configurations and problem parameters. We demonstrate our results on complex heterogeneous architectures (clusters with SMP nodes), where underlying network latencies and bandwidths vary significantly depending on machine configurations. Even for such challenging platforms, with small training sets, we are able to build highly accurate models of parallel performance.

Performance evaluation of parallel systems (a parallel system is defined as a combination of a parallel platform and a program executing on the platform) is a challenging task. The performance of serial programs can be expressed in terms of the input size, with the expectation that this quantification holds good on all serial platforms. Similar quantification of parallel programs must include, in addition to problem size, the number of processors, and various communication parameters of the underlying platform. What results is a quantification of performance that is specific to the machine, the problem instance, and in some cases, the number of processors (particularly for heterogeneous platforms). Changing any of these render the quantification less accurate. Simply stated, the observed performance of a program on a problem instance, using a given number of processors, says little about the performance on larger problem instances or numbers of processors [17]. With these motivating challenges, researchers have investigated both analytical and empirical models for performance evaluation.

One of the major drawbacks of analytical modeling is its reliance on asymptotes – namely that in many cases, the results correspond to arbitrarily large problem instances or machine configurations. This, combined with the difficulty in tight asymptotic analyses has motivated research on empirical models for performance analysis. The major drawback of an empirical model is that it provides only limited ability to extrapolate performance to systems that cannot be sampled (machines that do not exist). Furthermore, the highly non-linear nature of performance profiles does not lent itself easily to sampled interpolation. One of the major applications of empirical modeling is in the development of adaptive libraries [9,21,39].

Motivated by the drawbacks of analytical and empirical approaches, researchers have attempted integrating the two approaches preserving associated benefits. A pseudo-empirical (or semi-empirical) approach [6,7] contains both analytical and empirical components. The key difference between these efforts and our results here is the application and platform. Specifically,

the irregular nature of our problem and the potentially heterogeneous nature of our underlying platform pose significant technical challenges. Unlike purely empirical performance models that suffer from extrapolation/sampling errors or purely analytical models that only provide loose asymptotes on performance, our model combines the benefits of both into a comprehensive pseudo-analytical model. In addition to being accurate, this model also clearly identifies performance bottlenecks. For example, where asymptotes associated with memory access dominate computation, we can infer that the computation is memory bound. Similarly, where parameters associated with message latencies dominate those associated with bandwidth terms, we can conclude that message latencies dominate performance; conversely that aggregating multiple messages may result in further improvements in performance. Such insights are critical for improving performance on ultra-scale parallel systems.

We present the following specific results in this paper: (i) development of a highly scalable parallel formulation of the Truncated Spike solver, (ii) derivation of asymptotic performance estimates of various steps in Truncated Spike, (iii) parameterization of asymptotic performance models on target platforms, (iv) validation of the accuracy of parameterized performance models through extrapolation in machine configurations as well as problem size and (v) using performance models to argue superior scalability characteristics of the parallel Spike solver to petascale architectures.

In the rest of the paper we provide the following: (i) applications of banded preconditioners for the solution of general sparse linear systems of equations in Section 2; (ii) an overview of the Truncated Spike solver and its various steps in Section 3 together with results that illustrate its superior scalability and its ability to realize low relative residuals compared to banded solvers in ScaLapack; (iii) performance modeling of the Truncated Spike solver together with an analytical characterization of its various steps in Section 4; and (iv) detailed description of our methodology for training the analytical model in Section 5. Finally, we validate our model and draw inferences from it in Section 6, and conclude with a summary of our contributions and outline of ongoing research in Section 7.

2. Banded preconditioners and solvers

Our recent work has established preconditioners based on banded approximations of matrices as effective high-performance alternatives to existing ILU-based methods. Banded approximations to matrices are derived by bringing heavy elements close to the diagonal using a weighted spectral ordering. Unlike classical envelope reduction algorithms, such as Reverse Cuthill–McKee [8], weighted spectral reordering aims at reordering the matrix so that the larger elements are placed closer to the main diagonal. Weighted spectral reordering is achieved by using the second smallest eigenvalue and the corresponding eigenvector, namely the Fiedler vector [11] of the Laplacian matrix corresponding to the original matrix. After computing the Fiedler vector sorting it (ascending or descending order) produces the permutation that reorders the matrix so that large elements are closer to the main diagonal and extracted central dominant band is a much more effective preconditioner. MC73 algorithm [20] in Harwell Subroutine Library is a multilevel scheme for computing the Fiedler vector. We have recently developed a parallel algorithm [25] that achieves significant speed improvement over MC73 for computing the Fiedler vector. The new algorithm is based on the Trace Minimization method [33,34].

We have demonstrated that: (i) these banded preconditioners are capable of better convergence characteristics and robustness than other iterative solvers; and (ii) when used with an efficient (approximate) banded solver, they result in considerably shorter time-to-solution especially for problems with a larger condition number. We summarize these results in Table 1 for variety of linear systems (Table 2) and refer readers to [26] for a comprehensive description of the method.

Simulation of microelectromechanical systems (MEMS) is one of the areas where solution of large linear systems of equations is required. PRISM project at Purdue University provided us with a linear system of 11,333,520 unknowns. The coefficient matrix is symmetric but not positive definite. The relative residual required by the outer non-linear solver is 10^{-2}. We used an Intel cluster that consists of X5560 processors on each node and Infiniband interconnect between the nodes.

Multigrid methods [12] has recently gained considerable attention (see, for example, [2,14,38,41]). Geometric multigrid methods are often problem specific and is not applicable for irregular problems such as problems that are using unstructured grids. Algebraic multigrid (AMG) method has been developed as a general solver that does not rely on the formulation of the underlying problem. Multigrid methods typically consists of the following steps: smoothing, restriction and prolongation. They can be either used as a solver or as a solver for the systems involving a preconditioner.

To establish the basis of our comparison we use two well-known algebraic multigrid preconditioners,

namely Trilinos-ML [13] and Hyper (BoomerAMG) [19] with PETSc. We used a banded preconditioner with bandwidth 11.

BoomerAMG is developed in Lawrence Livermore National Laboratory. We use the default settings for BoomerAMG. In Fig. 1 we show the total solve time for both Spike-based banded preconditioned BiCGStab [40] and BoomerAMG preconditioned BiCGStab. Scalability of BoomerAMG suffers from excessive time spent in preconditioner factorization time after 64 cores.

Trilinos-ML, developed in Sandia National Laboratory, is another algebraic multigrid preconditioner. For Trilinos-ML we experimented with various smoothers for the coarsest level grid, namely Chebyshev, Jacobi and Gauss–Seidel. Chebyshev smoother is the fastest and the most scalable therefore we show the relative improvement over Trilinos-ML preconditioner with Chebyshev smoother in Fig. 2. Our banded preconditioner is faster than Trilinos-ML if more than 32 cores is used.

Our Spike-based banded preconditioner is general and works with a broad class of iterative solvers. The associated Spike solver is also shown to scale well with increasing number of processors. When used with a scalable solver, the solver-banded preconditioner combination holds potential for significant performance gains. In this paper, we comprehensively analyze the scalability of the Spike-based preconditioner. We analytically quantify the cost of each phase of the Spike preconditioner. We parametrize our analytical model and demonstrate its ability to accurately predict performance on increasing number of processors. We then use this validated parametrized analytical model to show that our precon-

Table 1

Total solve time (in seconds) for banded preconditioner, ILUTI, ILUPACK preconditioned BiCGStab and PARDISO on a uniprocessor ($\|r_k\|_\infty/\|r_0\|_\infty \leqslant 10^{-5}$), F means the method has failed

Matrix	Banded	LU based ILUTI(*, 10^{-1})	PARDISO	ILUPACK
1	0.87	0.14	1.28	0.5
2	0.41	0.19	1.48	0.23
3	457.2	193.5	91.6	F2
4	5.63	0.84	450.73	2.86
5	0.82	0.88	270.17	44.6
6	0.7	0.27	0.88	0.27
7	77.3	26.2	1358.7	330
8	205.4	459.1	7.6	F
9	0.33	F	0.78	F
10	1.95	15.8	3.14	2.07
11	F	F	1.07	F
12	0.42	F	1.74	0.46
13	7.0	F	45.3	F
14	F	F	449.1	F
15	0.27	0.59	1.28	F
16	2.0	239.0	27.2	F
17	0.21	0.11	0.95	1.44

Table 2

Properties of test matrices, k is the semi-bandwidth of the preconditioner

Number	Name	Condest	k	Dimension (N)	Non-zeros (nnz)	Type
1	FINAN512	9.8×10^1	50	74,752	596,992	Financial optimization
2	FEM_3D_THERMAL1	1.7×10^3	50	17,880	430,740	3D thermal FEM
3	RAJAT31	4.4×10^3	30	4,690,002	20,316,253	Circuit simulation
4	H2O	4.9×10^3	50	67,024	2,216,736	Quantum chemistry
5	APPU	1.0×10^4	50	14,000	1,853,104	NASA benchmark
6	BUNDLE1	1.3×10^4	50	10,581	770,811	3D computer vision
7	MIPS80S-1	3.3×10^4	30	986,552	7,630,280	Nonlinear optimization
8	RAJAT30	8.7×10^4	30	643,994	6,175,244	Circuit simulation
9	DW8192	1.5×10^7	182	8192	41,746	Dielectric waveguide
10	DC1	1.1×10^{10}	50	116,835	766,396	Circuit simulation
11	MSC23052	1.4×10^{12}	50	23,052	1,154,814	Structural mechanics
12	FP	8.2×10^{12}	2	7548	834,222	Electromagnetics
13	PRE2	3.3×10^{13}	30	659,033	5,959,282	Harmonic balance method
14	KKT_POWER	4.2×10^{13}	30	2,063,494	14,612,663	Nonlinear optimization (KKT)
15	RAEFSKY4	1.5×10^{14}	50	19,779	1,328,611	Structural mechanics
16	ASIC_680k	9.4×10^{19}	3	682,862	3,871,773	Circuit simulation
17	2D_54019_HIGHK	8.1×10^{32}	2	54,019	996,414	Device simulation

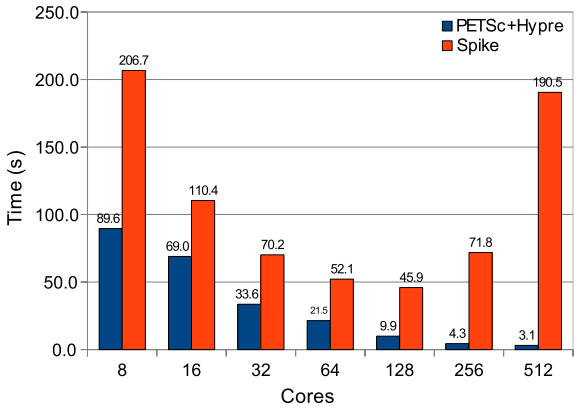

Fig. 1. Total solve time for banded Spike and Hypre preconditioner. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2011-0316.)

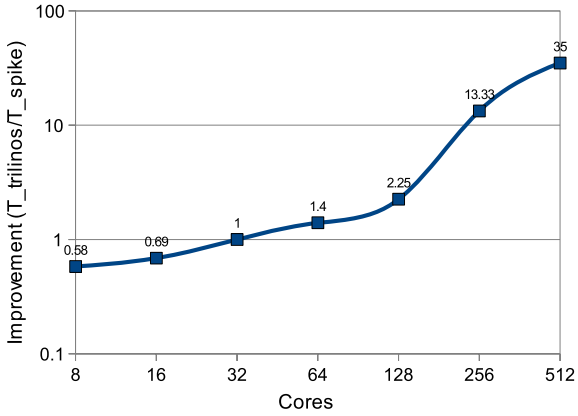

Fig. 2. Speed improvement of banded Spike preconditioner over Trilinos-ML. (Colors are visible in the online version of the article; http:// dx.doi.org/10.3233/SPR-2011-0316.)

ditioner is capable of excellent scalability. Note that the overall scalability is also impacted by the outer solver. For example, all Krylov subspace methods (such as CG, BiCGStab, . . . , etc.) require computation of inner products(MPI_ALLREDUCE). Preconditioned Richardson iterations and Chebyshev iterations [28], on the other hand, does not require computation of inner products. In Fig. 3, we illustrate the impact of global reduction on overall scalability for a medium size test problem.

3. The Spike linear system solver

The goals of this paper are two fold: to demonstrate the parallel efficiency and scalability of the Spike banded linear system solver/preconditioner, and to develop a predictive performance model in support of observed experimental scalability. We initiate this discussion with a brief overview of the Spike solver algorithm.

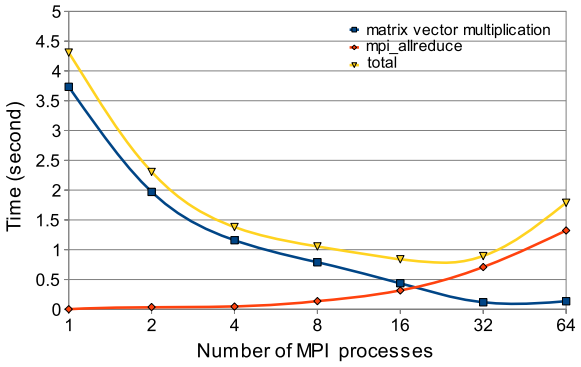

Fig. 3. Comparison of sparse matrix vector multiplication, global reductions (MPI_ALLREDUCE) and total time for solving a medium size problem using preconditioned conjugate gradient (PCG). (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2011-0316.)

Consider a method for solving the linear system $AX = F$, where A is a narrow banded $N \times N$ matrix and F corresponds to an $N \times s$ matrix of multiple right-hand side vectors. Please note that the Spike algorithm admits matrices that are sparse within the band – generalized banded form (see Fig. 4). Furthermore, the method can be generalized to arbitrary sparse matrices, using Spike as a solver for the preconditioner (a banded approximation to the matrix) [26]. LU-factorization of each diagonal block, A_j, can be computed via Lapack [1] and Pardiso [35,36] for banded and generalized banded case, respectively. For our description here, we assume that the bandwidth b is much less than the system order N. Consider a partitioning of the banded linear system into a block tridiagonal form as shown in Fig. 5, with $p = 3$ partitions. Each banded diagonal block A_j ($j = 1, \ldots, p$), is of order n_j (or roughly of order N/p), and the coupling matrices B_j ($j = 1, \ldots, p-1$), and C_j ($j = 2, \ldots, p$), are of order $m \ll n_j$.

In order to illustrate the basic Spike solver, we assume that each A_j is non-singular. We obtain the factorization $A = DS$, where D is a block-diagonal matrix consisting only of the diagonal blocks A_j and the Spike matrix S is shown in Fig. 5. For a given partition j, we call V_j ($j = 1, \ldots, p-1$) and W_j ($j = 2, \ldots, p$), respectively, the right and the left spikes each of order $n_j \times m$.

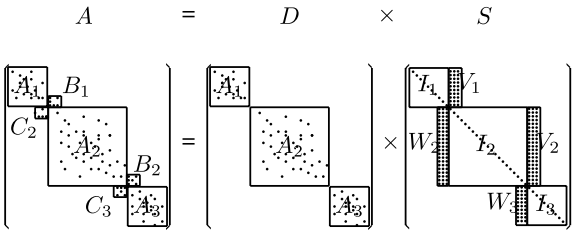

Fig. 4. Decomposition where $A = D * S, S = D^{-1}A$, $B_j, C_j \in \mathbb{R}^{m\times m}$ and A is generalized banded.

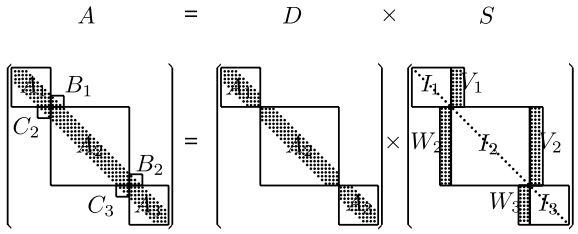

Fig. 5. Decomposition where $A = D * S, S = D^{-1}A$, $B_j, C_j \in \mathbb{R}^{m\times m}$ and A is banded.

Solving the system $AX = F$ thus reduces to the following two steps:

$$\text{(a) solve } DG = F, \tag{1}$$

$$\text{(b) solve } SX = G. \tag{2}$$

The inherently parallel nature of the Spike algorithm becomes apparent from these two steps. Unlike traditional LU-factorization schemes, the computationally expensive step (step (a) above) is perfectly parallel in Spike. Step (b) above corresponds to solving a system $SX = G$, which can be reduced to a system of much smaller size:

$$\hat{S}\hat{X} = \hat{G}. \tag{3}$$

This reduced system consists of the m rows of S immediately above and below each partition. It is then possible to extract from Eq. (2) the independent reduced linear system (3), which involves only the top and bottom elements of V_j, W_j, i.e., $V_j^{(b)}$, $V_j^{(t)}$, $W_j^{(b)}$ and $W_j^{(t)}$, as well as the corresponding parts of X and G. This reduced system is block tridiagonal with $(p-1)$ diagonal blocks, the kth of which is given by:

$$\begin{bmatrix} I_m & V_k^{(b)} \\ W_{k+1}^{(t)} & I_m \end{bmatrix}$$

and the corresponding off-diagonal blocks are given by:

$$\begin{bmatrix} W_k^{(b)} & 0 \\ 0 & 0 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 0 & 0 \\ 0 & V_{k+1}^{(t)} \end{bmatrix}.$$

Once the solution $\hat{X}$ of the reduced system (3) is obtained, the rest of the solution X is retrieved with perfect parallelism.

The Spike algorithm, which builds on existing dense matrix primitives (BLAS3), its own library of banded matrix primitives, as well as existing Lapack blocked matrix algorithms (see Fig. 6), consumes more arithmetic operations than the classical banded LU-factorization based methods (for example, those used in Lapack, and it parallel counterpart Scalapack). However, Spike significantly reduces memory references and interprocessor communications resulting in higher performance than that realized by ScaLapack. For example, Fig. 7 illustrates the superior scalability of Spike compared to ScaLapack for a diagonally dominant banded system on an Intel Clovertown. Note that ScaLapack is actually slower than Lapack on two cores

Level	Description
3	SPIKE
2	LAPACK blocked algorithms
1	Primitives for banded matrices (developed for SPIKE)
0	BLAS3 (matrix-matrix primitives)

Fig. 6. Hierarchy of computational modules in Spike algorithm. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2011-0316.)

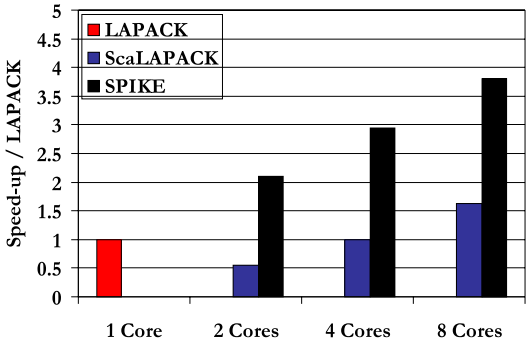

Fig. 7. Comparison of Spike algorithm to ScaLapack on two 4 core Intel Clovertown processor. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2011-0316.)

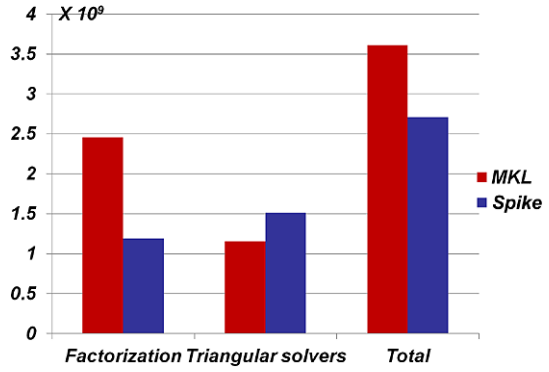

Fig. 8. Off-chip data accessed (in bytes) for solving a banded system with Spike and MKL-ScaLapack. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2011-0316.)

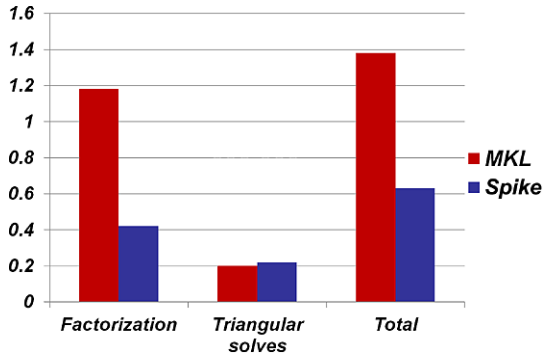

Fig. 9. Solution time (in seconds) for a banded system with Spike and MKL-ScaLapack. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2011-0316.)

filename	kl	ku	N	~cond
1- misc/CYLSHELL/s3dkq4m2.csr	614	614	90449	N/A
2- misc/CYLSHELL/s3dkt3m2.csr	614	614	90449	N/A
3- SPARSKIT/FIDAP/fidap035.csr	244	247	19716	4.33E+12
4- SPARSKIT/DRIVCAV/e40r0000.csr	451	451	17281	2.19E+08
5- SPARSKIT/DRIVCAV/e40r5000.csr	451	451	17281	2.20E+10
6-Harwell-Boeing/bcsstruc3/bcsstk25.csr	292	292	15439	1.28E+13
7-Harwell-Boeing/bcsstruc2/bcsstk18.csr	1243	1243	11948	6.49E+11
8-Harwell-Boeing/bcsstruc2/bcsstk17.csr	521	521	10974	1.95E+10

Fig. 10. Set of banded matrices from the University of Florida matrix collection with large condition number, where ku and kl are upper and lower bandwidths, respectively, and N is system size. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2011-0316.)

Matrices	MKL (1 Core)	MKL (2 Cores)	SPIKE (2 Cores)
1	9E-03	3E-02	2E-05
2	9E-03	3E-02	3E-05
3	5E-06	4E-06	5E-06
4	8E-09	3E-08	2E-10
5	1E-08	2E-08	2E-09
6	1E-07	3E-07	4E-08
7	2E-10	1E-10	6E-11
8	2E-09	4E-09	1E-09

Fig. 11. Final relative residuals after solving systems with Spike and MKL-ScaLapack. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2011-0316.)

and matches the speed of Lapack on 4 cores. In contrast, Spike becomes twice as fast as Lapack on 2 cores with reasonable speed improvements as the number of cores increases. Further study [23] illustrates the reason for the scalability of Spike by monitoring the off-chip data accessed by ScaLapack (in Intel's Math Kernel Library (MKL)) and Spike during their respective factorization and solve stages (see Fig. 8). Note that Spike requires rougly half the size of data accessed off-chip during the factorization stage. This results, in turn, to Spike requiring less than half the time required by ScaLapack to solve a banded system of order 0.6 million with bandwidth 99, see Fig. 9. Since the Truncated Spike algorithm is often used with an outer iterative scheme (usually BiCGstab) – requiring very few outer iterations – for non-diagonally dominant banded systems, we have compared both the relative residuals and the time achieved by Spike in comparison to both Lapack and ScaLapack on 2 cores of an Intel Clovertown platform for solving 8 banded systems (sparse within the band) obtained from the University of Florida matrix collection, see Fig. 10. These systems, however, were treated as dense within the band. Figure 11 illustrates the superior relative residuals obtained by our hybrid Spike scheme compared to Lapack and ScaLapack.

Recent work on generalizing Spike for solving banded systems that are sparse within the band, as well as its use as a solver for banded preconditioners of iterative schemes for handling general sparse systems, has shown excellent performance characteristics, e.g., see [26]. In particular, we have tested the use of the Spike algorithm for solving narrow banded preconditioners (bandwidth less than or equal to 51) extracted from 17 general sparse systems (orders ranging from 7000 to 2 million, and the number of non-zero elements ranging from 42,000 to 20 million) chosen from the University of Florida collection to represent 12 different applications (see Table 2). In all successful cases, BiCGstab + Spike as solver for banded pre-

conditioners achieved higher parallel scalability than BiCGstab + ILUPACK (approximate LU-factorization preconditioners). Table 1 illustrates the robustness of our scheme, two convergence failures compared to eight failures for BiCGstab + ILUPACK.

Furthermore, Spike admits a variety of algorithms in each of its solver steps – effectively presenting a polyalgorithm capable of being tuned to diverse architecture for further performance optimization. In this paper we deal with diagonally dominant linear systems. For such systems, since spikes decay as one moves away from the main diagonal, a truncated variation of the algorithm is used. An extensive error analysis for the Truncated Spike algorithm is given in [29].

4. Performance modeling and cost analysis of Spike

We identify various steps in the Spike solver and analytically quantify the number of operations in each step. We also identify various communication steps in parallel implementation and associate cost models with these steps. We then train these models on target hardware platforms to derive comprehensive performance models. We validate these models by extrapolating performance (well) beyond training sets and verifying it experimentally. Finally, we use our validated performance model to argue the excellent scaling characteristics of the Spike solver.

To model computational steps, we count number of memory references as well as number of operations (typically floating point operations). To model communication steps, we model latency, link bandwidth, as well as global congestion. It is important to note that different terms may manifest themselves to varying degrees on different hardware platforms. Fitting these terms to experimental data automatically detects these relationships, as we shall demonstrate. Furthermore, fitting to experimental data also reveals whether specific steps are memory/computation bound, latency/bandwidth bound, etc. These are important features that directly guide algorithm development and code optimization.

4.1. Cost analysis of the Truncated Spike steps

We identify the following steps in the solution of linear systems using the Spike algorithm. In each step, we also give the Lapack routine that can be used to accomplish the operation. Building the Spike solver out of optimized dense kernels significantly enhances performance by optimizing for deep memory hierarchies. We assume in our performance models that the upper and lower bandwidths of the matrix A are identical, i.e., $b = b_{\mathrm{l}} = b_{\mathrm{u}}$. We assume that the dimension of matrix A is N, and that the dimension of each partition, n, is given by N/p. Recall that p is the number of partitions used to cast A into its block tridiagonal form. We also assume that the number of partitions is identical to the number of processors used to solve the linear system.

(1) Compute the LU-factorization of blocks of A (using call to subroutine DDBTRF)

- $L_j U_j \leftarrow A_j$ for $j = 1, 2, 3, 4$.

This step involves nb^2 computations and nb memory references. We model this cost as $\alpha_1 nb^2 + \beta_1 nb$.

(2) Compute spikes (using call to subroutine DTBTRS):

- Solve for V_j: $L_j U_j V_j = [0 \ \ldots \ 0 \ B_j^{\mathrm{T}}]^{\mathrm{T}}$ for $j = 1, 2, 3$.
- Solve for W_j: $L_j U_j W_j = [C_j^{\mathrm{T}} \ 0 \ldots \ 0]^{\mathrm{T}}$ for $j = 2, 3, 4$.

This step involves nb^2 computations and nb memory references. We model this cost as $\alpha_2 nb^2 + \beta_2 nb$.

(3) Communicate spike tips $W_j^{(\mathrm{t})}$ (processor j sends data to processor $j-1$, for $j = 2, 3, 4$). $W_j^{(\mathrm{t})}$ is the top $b \times b$ part of W_j. The data sent per process is b^2. Since each process communicates this amount of data, we model this by the expression, $\alpha_3 b^2(p-1) + \beta_3 b^2 + \gamma_3$. Here, the first term models congestion on the network since $p-1$ processes communicate b^2 data items at the same time, the second term models link bandwidth, and the third term models communication latency. Note that the congestion term may dominate on shared address space platforms.

(4) Factorize the reduced system (using call to subroutine DGETRF)

$$\begin{pmatrix} I & V_j^{(\mathrm{b})} \\ W_j^{(\mathrm{t})} & I \end{pmatrix} \tag{4}$$

for $j = 2, 3, 4$. Since this corresponds to a dense factorization of matrices of size $b \times b$, we model the cost using the expression $\alpha_4 b^3 + \beta_4 b^2$. The first term corresponds to the number of operations and the second step, the number of memory references.

Table 3
Truncated Spike algorithm cost model

Stage	Cost
1. Factorize the diagonal blocks	$\alpha_1 nb^2 + \beta_1 nb$
2. Compute spikes	$\alpha_2 nb^2 + \beta_2 nb$
3. Communicate tips of spikes	$\alpha_3 b^2(p-1) + \beta_3 b^2 + \gamma_3$
4. Factorize the reduced system	$\alpha_4 b^3 + \beta_4 b^2$
5. Modify the right-hand side	$\alpha_5 nb$
6. Communicate tips of MRHS	$\alpha_6 b(p-1) + \beta_6 b + \gamma_6$
7. Solve the reduced system	$\alpha_7 b^2$
8. Communicate solution of the reduced system	$\alpha_8 b(p-1) + \beta_8 b + \gamma_8$
9. Retrieve the solution	$\alpha_9 nb + \beta_9 n$

(5) Modify the right-hand side by solving (using call to subroutine DTBTRS): $L_j U_j g_j = f_j$ ($j = 1, 2, 3, 4$). This step requires nb operations and nb memory references. We model this by the expression $\alpha_5 nb$.

(6) Communicate the modified right-hand side tips $g_j^{(t)}$ (processor j sends data to processor $j-1$ for $j = 2, 3, 4$). The amount of data communicated by each process is b. We model this communication cost by the expression $\alpha_6 b(p-1) + \beta_6 b + \gamma_6$. As before, the first term corresponds to congestion, since $p-1$ processes communicate b data words at the same time, the second term to link bandwidth, and the third term to communication latency.

(7) Solve the reduced system

$$\begin{pmatrix} I & V_j^{(b)} \\ W_{j+1}^{(t)} & I \end{pmatrix} \begin{pmatrix} x_j^{(b)} \\ x_{j+1}^{(t)} \end{pmatrix} = \begin{pmatrix} g_j^{(b)} \\ g_{j+1}^{(t)} \end{pmatrix} \tag{5}$$

for $j = 2, 3, 4$. This corresponds to triangular solves of systems of order b. The number of operations and memory references are both b^2. The cost is therefore modeled as $\alpha_7 b^2$.

(8) Communicate the reduced system solution $g_{j+1}^{(t)}$ (processor j sends data to processor $j+1$ for $j = 1, 2, 3$). As in Step 6, the amount of data communicated by each process is b words, resulting in a total cost of $\alpha_8 b(p-1) + \beta_8 b + \gamma_8$.

(9) Retrieve x_j ($j = 1, 2, 3, 4$) (using call to subroutine DGEMV) $x_j = f_j - V_j x_{j+1}^{(t)} - W_j x_{j-1}^{(b)}$ ($V_4 = 0$ and $W_1 = 0$). This step involves nb operations associated with the matrix–vector product and n operations for vector addition. The number of memory references in these two steps is also given by nb and n, respectively. The associated cost is modeled as $\alpha_9 nb + \beta_9 n$.

The costs for various Spike steps are summarized in Table 3. We note that Stages 3, 6 and 8 are communication stages, while Stages 1, 2, 4, 5, 7 and 9 are computation only.

5. Parameterizing Spike performance models to target platforms

We train our performance model for Spike on two different target platforms – the Sun Constellation Linux Cluster at the University of Texas (RANGER) and an Intel Xeon Cluster (IXC). RANGER has 3936 nodes, with each node containing four 2.3 GHz AMD Opteron Quad-Core 64-bit processors with a 1.0 GHz Hypertransport system Bus. Each node in the system has 32 GB of memory. Nodes are interconnected using an Infiniband network. IXC has 256 nodes, with each node containing two four-core Xeon E5462 (2.8 GHz, 12 MB L2 cache) processors with 16 GB RAM per node and 1600 MHz front side bus. This cluster also uses an Infiniband interconnect. We have also trained our models to more monolithic machines such as the IBM SP. However, the heterogeneous interconnects of RANGER and IXC (on-chip interconnect, intranode network, and Infiniband) provide more interesting challenges from the point-of-view of performance modeling.

With respect to linear systems, we use diagonally dominant systems of dimension 5,000,000 and 10,000,000 with 4.0 on the main diagonal and -0.01 on off diagonals with a right-hand side vector of all ones. We use Goto blas [16] and PGI 7.1 compiler on RANGER, while on IXC we use Intel Fortran Compiler 10.1 and Intel MKL 9.1. On RANGER, we train the model using 16, 32, 64 processors (1, 2 and 4 nodes, respectively) for bandwidths $b = 15, 25$ and 35 and matrix dimension 5,000,000. On IXC, we train the model using 16, 32 and 64 processors (4, 8 and 16 nodes, respectively) for bandwidths $b = 10, 20$ and 30 and matrix dimension 10,000,000. We select the bandwidths and matrix dimensions in such a way that the performance of the serial subroutine calls saturates (note that at small matrix sizes, the performance of the serial components of the algorithm may themselves vary significantly). We time the individual stages of the algorithm via wall clock time on each MPI process independently and use the maximum of all calls as the

time consumed in that stage. Using a least squares approximation, we fit the models from Table 3 to determine various constants on each platform.

Those constants, namely $\alpha_i, \beta_i, \gamma_i$ for $i = 1, \ldots, 9$, for the RANGER and IXC platforms are given in Tables 4 and 5, respectively. If a parameter is smaller than 10^{-12}, we replace it with $\sim$0 in the tables.

We note that Step 1 (factorization of diagonal blocks) is primarily memory bound on the IXC, the communication step in Stage 3 has little global contention on both platforms, and the factorization in Step 4 is primarily processor bound on both platforms. We also note that Steps 3, 6 and 8 are communication steps. Since both platforms have Infiniband interconnects, we observe similar communication parameters.

6. Experimental validation of model

To validate the accuracy of the model, we predict performance of the solver under two scenarios – scaling number of processors for a fixed problem size and scaling problem size with number of processors. In each case, we enumerate predicted and observed runtime for the two platforms. As before, we generate systems of desired sizes.

Table 4
Spike cost model parameters on RANGER

Stage	α_i	β_i	γ_i
1	8.62×10^{-10}	2.61×10^{-8}	–
2	3.96×10^{-8}	3.66×10^{-7}	–
3	$\sim$0	1.21×10^{-6}	9.48×10^{-4}
4	$\sim$0	7.43×10^{-6}	–
5	2.85×10^{-8}	–	–
6	4.31×10^{-9}	4.10×10^{-7}	3.59×10^{-5}
7	1.19×10^{-7}	–	–
8	2.57×10^{-7}	$\sim$0	6.42×10^{-4}
9	8.86×10^{-8}	9.51×10^{-9}	–

Table 5
Spike cost model parameters on IXC

Stage	α_i	β_i	γ_i
1	$\sim$0	3.07×10^{-8}	–
2	2.70×10^{-8}	1.46×10^{-7}	–
3	$\sim$0	3.93×10^{-9}	2.55×10^{-5}
4	2.94×10^{-8}	1.30×10^{-8}	–
5	1.79×10^{-8}	–	–
6	3.16×10^{-9}	1.19×10^{-7}	2.39×10^{-5}
7	1.69×10^{-7}	–	–
8	1.51×10^{-7}	$\sim$0	1.24×10^{-4}
9	1.05×10^{-8}	2.55×10^{-9}	–

On the RANGER platform, we generate two systems of 5M and 10M unknowns. We solve these systems on 128, 246, 512 and 1024 processors. Lack of availability of larger chunks of the machine kept us from scaling number of processors further. We note the following two aspects of our validation setup that are particularly noteworthy:

- Performance prediction is more difficult for smaller problem sizes and correspondingly lower efficiencies. For scalable systems such as ours, larger problem instances generally lead to good efficiencies, and correspondingly easier prediction since speedup approaches linearity. For this reason, our choice of problem sizes poses significant challenges for performance prediction.
- Our training set is restricted to 64 processors, or four nodes. The intra-node interconnect is significantly different from the Infiniband interconnect across nodes. It is easy to see that our training set is fairly limited considering the three levels of interconnect that must be accurately modeled.

With this choice of machine and problem sizes, we present the predicted and measured runtime along with prediction error (all in seconds) on the RANGER platform in Table 6. In most cases, the error in prediction is less than 0.05 s. In some exceptional cases, the error is as high as 0.2 s. In our opinion, this is excellent prediction accuracy, considering the heterogeneous nature of the interconnect, the limited training set and small problem sizes (per processor). In fact, variations in serial performance alone can account for much of this prediction error.

On the IXC platform, we also use two different problem sizes. In this case, we use matrices of dimension 10M and 30M. The choice of larger problems is motivated by significant variations in serial runtime of the Lapack subroutines used in various steps of Spike. Since serial performance prediction of Lapack routines is beyond the scope of this paper, we operate in a range where serial performance is relatively stable. We verify our model on 128, 246 and 512 processors, enumerating measured and predicted parallel times, along with prediction error (also in seconds) in Table 7. As before, we note that in all cases, the prediction error is less than 0.2 s.

Results from Tables 6 and 7 clearly demonstrate excellent accuracy of our performance models. Us-

Table 6
Model verification for RANGER (in seconds)

N	b	p	Measured	Model	Error
5,000,000	35	128	2.78	2.64	0.13
5,000,000	25	128	1.55	1.49	0.06
5,000,000	15	128	0.70	0.66	0.04
5,000,000	35	256	1.49	1.33	0.16
5,000,000	25	256	0.79	0.75	0.04
5,000,000	15	256	0.35	0.33	0.02
5,000,000	35	512	0.67	0.67	0.00
5,000,000	25	512	0.38	0.38	0.00
5,000,000	15	512	0.20	0.17	0.03
5,000,000	35	1024	0.37	0.35	0.02
5,000,000	25	1024	0.21	0.20	0.01
5,000,000	15	1024	0.10	0.09	0.01
10,000,000	35	128	5.36	5.27	0.09
10,000,000	25	128	3.03	2.98	0.06
10,000,000	15	128	1.36	1.31	0.05
10,000,000	35	256	2.78	2.64	0.13
10,000,000	25	256	1.55	1.49	0.06
10,000,000	15	256	0.68	0.66	0.02
10,000,000	35	512	1.51	1.33	0.18
10,000,000	25	512	0.79	0.75	0.04
10,000,000	15	512	0.35	0.33	0.02
10,000,000	35	1024	0.69	0.68	0.01
10,000,000	25	1024	0.45	0.38	0.07
10,000,000	15	1024	0.19	0.17	0.02

ing the models, it is possible to perform more detailed analytical studies of scalability. A simple observation is that the asymptotic times associated with the various computation steps in Spike clearly dominate the communication steps. For this reason, as problem sizes are increased with increase in number of processors, we can maintain constant efficiencies (or linear speedups) – thus validating the scalability of our Spike solver. These conclusions are also supported by our experimental results in Tables 6 and 7.

Using our model we can predict the performance and the scalability of the Spike algorithm for much larger number of cores than we physically have or have access to. We provide the prediction for the scalability of Spike for solving a large system with 1,000,000,000 unknowns and semi-bandwidth 30 using upto 65,536 cores for RANGER and IXC platforms (see Fig. 12). The Spike algorithm scaled well upto 65,536 cores on both platforms.

Table 7
Model verification for IXC (in seconds)

N	b	p	Measured	Model	Error
10,000,000	30	128	2.45	2.38	0.07
10,000,000	20	128	1.26	1.16	0.09
10,000,000	10	128	0.36	0.37	0.02
10,000,000	30	256	1.22	1.19	0.03
10,000,000	20	256	0.62	0.58	0.04
10,000,000	10	256	0.14	0.19	0.05
10,000,000	30	512	0.59	0.60	0.01
10,000,000	20	512	0.23	0.29	0.07
10,000,000	10	512	0.06	0.09	0.03
30,000,000	30	128	7.29	7.13	0.15
30,000,000	20	128	3.71	3.49	0.21
30,000,000	10	128	1.06	1.11	0.06
30,000,000	30	256	3.67	3.57	0.10
30,000,000	20	256	1.89	1.75	0.15
30,000,000	10	256	0.55	0.56	0.00
30,000,000	30	512	1.88	1.79	0.10
30,000,000	20	512	0.99	0.87	0.12
30,000,000	10	512	0.30	0.28	0.02

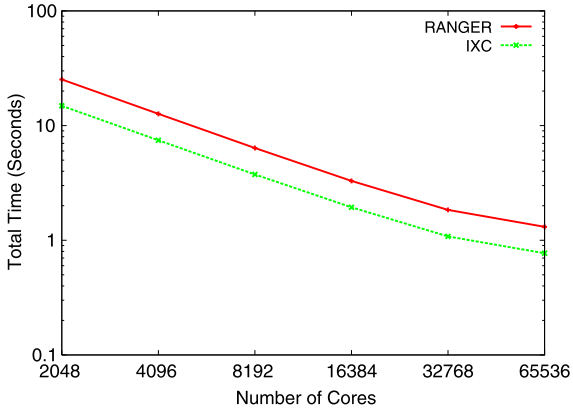

Fig. 12. Predicted total solve time for Spike. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2011-0316.)

7. Concluding remarks

In this paper, we presented a highly efficient parallel linear solver, Spike, along with a comprehensive performance model. We trained this model on two different platforms and used the model to extrapolate performance to larger machine configurations and problem instances. We verified the predicted performance through experiments, and demonstrated that our models are highly accurate. We also argue from this

performance model that the computational costs associated with the algorithm asymptotically dominate communication overheads, leading to highly scalable parallel formulations. Coupled with the superior performance of Spike even on small number of processors compared to state of the art banded solvers in ScaLapack, we argue that Spike is capable of excellent performance on large-scale parallel platforms. An important consequence of our performance models is that they reveal underlying performance bottlenecks in both serial and parallel formulations. Specifically, they identify computation steps that are memory or processor bound, and communication steps that are dominated by latency, link bandwidth, or global contention. These bottlenecks drive future algorithm design and code optimization.

The performance models in this paper targeted one instance of the Spike polyalgorithm. Other variants of the Spike solver are more difficult to characterize than the Truncated Spike algorithm considered here. Furthermore, the use of Spike for solving general sparse linear systems poses significant challenges for accurate performance modeling. These are avenues for continuing efforts on detailed performance characterization of the Spike solver.

Acknowledgements

This work was partially supported by grants from the NSF (NSF-CCF-0635169), the Department of Energy (DE-FC52-08NA28617), the PRACE project funded in part by the EU's 7th Framework Programme (FP7/2007-2013) under grant agreement No. RI-261557 and by a gift from the Intel Corporation.

We would like to thank you Professor Zhiyuan Li for running tests on TACC RANGER platform. We also thank Sanjay Marthur and Gazi Yildirim for providing the PRISM linear system.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney and D. Sorensen, LAPACK: A portable linear algebra library for high-performance computers, Knoxville, Technical Report, 1990, available at: citeseer.ist.psu.edu/anderson90lapack.html.

[2] D.N. Arnold, R.S. Falk and R. Winther, Multigrid in h (div) and h (curl), *Numerische Mathematik* **85** (2000), 197–217, available at: http://dx.doi.org/10.1007/PL00005386.

[3] M.W. Berry and A. Sameh, Multiprocessor schemes for solving block tridiagonal linear systems, *The International Journal of Supercomputer Applications* **1**(3) (1978), 37–57.

[4] L.S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker and R.C. Whaley, ScaLAPACK: a linear algebra library for message-passing computers, in: *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*, Minneapolis, MN, 1997, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997, 15 pp. (electronic), available at: citeseer.ist.psu.edu/article/blackford97scalapack.html.

[5] S.C. Chen, D.J. Kuck and A.H. Sameh, Practical parallel band triangular system solvers, *ACM Transactions on Mathematical Software* **4**(3) (1978), 270–277.

[6] C.-Y. Chou, H.-Y. Chang, S.-T. Wang and C.-H. Wu, A semi-empirical model for maximal linpack performance predictions, in: *CCGRID*, 2006, pp. 343–348.

[7] J. Cuenca, L.-P. García, D. Giménez, J. González and A. Vidal, Empirical modelling of parallel linear algebra routines, in: *5th International Conference, PPAM 2003*, Lecture Notes in Computer Science, Vol. 3019, 2004, pp. 169–174.

[8] E. Cuthill and J. McKee, Reducing the bandwidth of sparse symmetric matrices, in: *Proceedings of the 1969 24th National Conference*, ACM, New York, NY, USA, 1969, pp. 157–172.

[9] J. Dongarra and V. Eijkhout, Self-adapting numerical software for next generation applications, *International Journal of High Performance Computing and Applications* **17** (2002), 02–07.

[10] J.J. Dongarra and A.H. Sameh, On some parallel banded system solvers, *Parallel Computing* **1**(3) (1978), 223–235.

[11] M. Fiedler, Algebraic connectivity of graphs, *Czechoslovak Mathematical Journal* **23**(98) (1973), 298–305.

[12] R.P. Fedorenko, A relaxation method for solving elliptic difference equations, *USSR Computational Mathematics and Mathematical Physics* **1** (1961), 1092.

[13] M. Gee, C. Siefert, J. Hu, R. Tuminaro and M. Sala, ML 5.0 smoothed aggregation user's guide, Sandia National Laboratories, Technical Report SAND2006-2649, 2006.

[14] U. Ghia, K.N. Ghia and C.T. Shin, High-Re solutions for incompressible flow using the Navier–Stokes equations and a multigrid method, *Journal of Computational Physics* **48**(3) (1982), 387–411, available at: http://www.sciencedirect.com/science/article/B6WHY-4DD1X97-1KG/2/67432b085c087cd844fc1195aac5cc72.

[15] K.M. Giannoutakis and G.A. Gravvanis, A performance study of normalized explicit finite element approximate inverse preconditioning on uniprocessor and multicomputer systems, *Engineering Computations* **23**(3) (2006), 192–217.

[16] K. Goto and R. Van De Geijn, High-performance implementation of the level-3 BLAS, *ACM Transactions on Mathematical Software* **35**(1) (1978), 1–14.

[17] A. Grama, A. Gupta, V. Kumar and G. Karypis, *Introduction to Parallel Computing*, Addison-Wesley, Boston, MA, USA, 2003.

[18] G.A. Gravvanis, On the solution of boundary value problems by using fast generalized approximate inverse banded matrix techniques, *The Journal of Supercomputing* **25**(2) (2003), 119–129.

[19] V.E. Henson and U.M. Yang, Boomeramg: A parallel algebraic multigrid solver and preconditioner, *Applied Numerical Mathematics* **41**(1) (1978), 155–177.

[20] Y. Hu and J. Scott, HSL_MC73: a fast multilevel Fiedler and profile reduction code, Technical Report RAL-TR-2003-036, 2003.

[21] D.F. Kvasnicka and C.W. Ueberhuber, Developing architecture adaptive algorithms using simulation with MISS-PVM for performance prediction, in: *ICS'97: Proceedings of the 11th International Conference on Supercomputing*, ACM, New York, NY, USA, 1997, pp. 333–339.

[22] D.H. Lawrie and A.H. Sameh, The computation and communication complexity of a parallel banded system solver, *ACM Transactions on Mathematical Software* **10**(2) (1978), 185–195.

[23] L. Liu, Z. Li and A.H. Sameh, Analyzing memory access intensity in parallel programs on multicore, in: *Proceedings of the 22nd Annual International Conference on Supercomputing, ICS'08*, ACM, New York, NY, USA, 2008, pp. 359–367, available at: http://doi.acm.org/10.1145/1375527.1375579.

[24] M. Manguoglu, A parallel hybrid sparse linear system solver, in: *Computational Electromagnetics International Workshop, CEM 2009*, July 2009, pp. 38–43.

[25] M. Manguoglu, A highly efficient parallel algorithm for computing the Fiedler vector, ArXiv e-prints, 2010.

[26] M. Manguoglu, M. Koyuturk, A. Grama and A.H. Sameh, Weighted matrix ordering and parallel banded preconditioners for iterative linear system solvers, *SIAM Journal on Scientific Computing* **32**(3) (1978), 1201–1216.

[27] M. Manguoglu, A. Sameh and O. Schenk, A parallel hybrid sparse linear system solver, in: *Proceedings of EURO-PAR09*, Lecture Notes in Computer Science, Vol. 5704, 2009, pp. 797–808.

[28] T.A. Manteuffel, The Tchebychev iteration for nonsymmetric linear systems, *Numerische Mathematik* **28** (1977), 307–327, available at: http://dx.doi.org/10.1007/BF01389971.

[29] C.C.K. Mikkelsen and M. Manguoglu, Analysis of the Truncated Spike algorithm, *SIAM Journal on Matrix Analysis and Applications* **30**(4) (2008), 1500–1519, available at: http://link.aip.org/link/?SML/30/1500/1.

[30] E. Polizzi and A.H. Sameh, A parallel hybrid banded system solver: the spike algorithm, *Parallel Computing* **32**(2) (1978), 177–194.

[31] E. Polizzi and A.H. Sameh, Spike: A parallel environment for solving banded linear systems, *Computers and Fluids* **36**(1) (1978), 113–120.

[32] A.H. Sameh and D.J. Kuck, On stable parallel linear system solvers, *Journal of Computational and Applied Mathematics* **25**(1) (1978), 81–91.

[33] A. Sameh and Z. Tong, The trace minimization method for the symmetric generalized eigenvalue problem, *Journal of Computational and Applied Mathematics* **123**(1,2) (2000), 155–175.

[34] A.H. Sameh and J.A. Wisniewski, A trace minimization algorithm for the generalized eigenvalue problem, *SIAM Journal on Numerical Analysis* **19**(6) (1982), pp. 1243–1259, available at: http://link.aip.org/link/?SNA/19/1243/1.

[35] O. Schenk and K. Gärtner, Solving unsymmetric sparse systems of linear equations with PARDISO, *Future Generation Computer Systems* **20**(3) (1978), 475–487, available at: http://www.sciencedirect.com/science/article/B6V06-49NXY7J-F/2/e8260e5d8f19639019cddea4776c024c.

[36] O. Schenk and K. Gärtner, On fast factorization pivoting methods for sparse symmetric indefinite systems, *Electronic Transactions on Numerical Analysis* **23** (2006), 158–179.

[37] O. Schenk, M. Manguoglu, A. Sameh, M. Christian and M. Sathe, Parallel scalable PDE-constrained optimization: antenna identification in hyperthermia cancer treatment planning, *Computer Science Research and Development* **23**(3,4) (2009), 177–183.

[38] K. Stüben, A review of algebraic multigrid, *Journal of Computational and Applied Mathematics* **128** (2001), 281–309, available at: http://portal.acm.org/citation.cfm?id=373538.373547.

[39] N. Thomas, G. Tanase, O. Tkachyshyn, J. Perdue, N.M. Amato and L. Rauchwerger, A framework for adaptive algorithm selection in STAPL, in: *PPoPP'05: Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ACM, New York, NY, USA, 2005, pp. 277–288.

[40] H.A. van der Vorst, Bi-CGSTAB: a fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems, *SIAM Journal on Scientific and Statistical Computing* **13**(2) (1978), 631–644.

[41] P. Wesseling, Introduction to multigrid methods, Technical report, 1995.