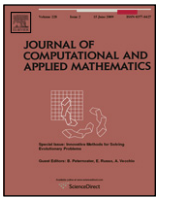
Journal of Computational and Applied Mathematics 236 (2011) 319–325

Contents lists available at SciVerse ScienceDirect

Journal of Computational and Applied Mathematics

journal homepage: www.elsevier.com/locate/cam



A domain-decomposing parallel sparse linear system solver

Murat Manguoglu

Computer Engineering, Middle East Technical University, Ankara, 06800, Turkey

ARTICLE INFO

Keywords:
Sparse linear systems
Parallel solvers
Direct solvers
Iterative solvers

ABSTRACT

The solution of large sparse linear systems is often the most time-consuming part of many science and engineering applications. Computational fluid dynamics, circuit simulation, power network analysis, and material science are just a few examples of the application areas in which large sparse linear systems need to be solved effectively. In this paper, we introduce a new parallel hybrid sparse linear system solver for distributed memory architectures that contains both direct and iterative components. We show that by using our solver one can alleviate the drawbacks of direct and iterative solvers, achieving better scalability than with direct solvers and more robustness than with classical preconditioned iterative solvers. Comparisons to well-known direct and iterative solvers on a parallel architecture are provided.

1. Introduction

Many applications in science and engineering give rise to large sparse linear systems of equations. Some of these systems arise in the discretization of partial differential equations (PDEs) modeling various physical phenomena, such as in computational fluid dynamics, semiconductor device simulations, and material science. Large and sparse linear systems also arise in applications that are not governed by PDEs (e.g. power system networks, circuit simulation, and graph problems).

Numerical simulation processes often consist of many layers of computational loops (e.g. see Fig. 1). It is a well-known fact that the cost of the solution process is almost always governed by the solution of linear systems, especially for large-scale problems.

The emergence of multicore architectures and highly scalable platforms motivates the development of novel algorithms and techniques that emphasize concurrency and are tolerant of deep memory hierarchies, as opposed to minimizing raw FLOP counts. While direct solvers are reliable, they are often memory intensive for large problems, and offer limited scalability. Iterative solvers, on the other hand, are more efficient but, in the absence of robust preconditioners, they lack reliability.

In this paper, we introduce a parallel sparse linear system solver that is a hybrid, and hence combines both direct and iterative methods. We note that we are using the term "hybrid" to emphasize that our solver is using both direct and iterative techniques. We advocate that by using our solver in hybrid mode one can alleviate the drawbacks of direct and iterative solvers, i.e. achieving more scalability than with a direct solver and more robustness than with a classical preconditioned iterative solver.

The rest of this paper is organized as follows. In Section 2, we discuss the background and related work. In Section 3, we give a description of the new algorithm and a simple example to demonstrate the details of the implementation. In Section 4, we present several numerical experiments. Finally, we conclude the paper with discussions in Section 5.

E-mail addresses: manguoglu@ceng.metu.edu.tr, murat.manguoglu@gmail.com.

0377-0427/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.cam.2011.07.017

Loop: Time Integration
  Loop: Nonlinear Iteration
    Loop: Linear Systems
      On parallel computing platforms; multicore to petascale architectures
    End ϵ
  End η
End Δt

Fig. 1. Target computational loop.

2. Background and related work

Considerable effort has been spent on algebraic parallel sparse linear system solvers. Sparse linear system solvers are traditionally divided into two groups: (i) direct solvers and (ii) iterative solvers. In the first group, some examples are MUMPS [1–3], Pardiso [4,5], and SuperLU [6].

Iterative solvers mainly consist of classical preconditioned Krylov subspace methods, and preconditioned Richardson iterations. Unlike direct sparse system solvers, iterative methods (with classical blackbox preconditioners) are not as robust. This is true even with the most recent advances in creating LU-based preconditioners [7–9]. Approximate inverse preconditioners [10–14] are known to be more favorable for parallelism.

The Spike algorithm [15–21] is a parallel solver for banded systems that combines direct and iterative methods, and it is one of the first examples of a hybrid linear system solver. More recently, in [22–24], the Spike algorithm was used for solving banded systems involving the preconditioner that is obtained after reordering the coefficient matrix with weights for sparse linear systems.

3. Domain-decomposing parallel solver

We introduce a new parallel hybrid sparse linear system solver called the Domain Decomposition Parallel Solver (DDPS), which can be used for solving sparse linear systems of equations: $Ax = f$. In [25], we presented an algorithm that used incomplete LU factorization for the diagonal block and its application on fluid structure interaction problems. In this paper, we introduce the DDPS, which uses the direct solver Pardiso within each block, and we extend the results to general sparse systems from a variety of application areas.

We are motivated to create the DDPS since many applications use domain decomposition to distribute the work among the processors and because of the lack of reliability of black box preconditioned Krylov subspace methods and the lack of scalability of direct solvers. METIS [26,27] is often used to partition the domain (and hence to partition the matrices). The DDPS is similar to the Spike algorithm, but unlike Spike it does not assume a banded structure for the coefficient matrix A. Given a general sparse linear system $Ax = f$, we partition $A \in R^{n\times n}$ into p block rows $A = [A_1, A_2, \ldots, A_p]^T$. Let

$$A = \mathcal{D} + R, \tag{1}$$

where $\mathcal{D}$ consists of the p block diagonals of A,

$$\mathcal{D} = \begin{pmatrix} A_{11} & & & \\ & A_{22} & & \\ & & \ddots & \\ & & & A_{pp} \end{pmatrix}, \tag{2}$$

and R consists of the remaining elements (i.e. $R = A - \mathcal{D}$). Let $\tilde{L}_i$ and $\tilde{U}_i$ be incomplete LU factorizations of A_{ii}, where $i = 1, 2, \ldots, p$. We define

$$\tilde{\mathcal{D}} = \begin{pmatrix} \tilde{A}_{11} & & & \\ & \tilde{A}_{22} & & \\ & & \ddots & \\ & & & \tilde{A}_{pp} \end{pmatrix}, \tag{3}$$

in which $\tilde{A}_{ii} = \tilde{L}_i\tilde{U}_i$.

The DDPS algorithm is shown in Fig. 2. We assume that the system $Ax = f$ is the one after METIS reordering.

Stages 1–5 are considered as a preprocessing phase in which the right-hand side is not required. After preprocessing, we solve the system via a Krylov subspace method and using a preconditioner. The major operations in a Krylov subspace method are: (i) matrix vector multiplications, (ii) inner products, and (iii) preconditioning operations in the form of $Pz = y$

Data: $Ax = f$ and a partitioning information
Result: x
1. $\mathcal{D} + R \longleftarrow A$ for the given partitioning information;
2. $\tilde{L}_i\tilde{U}_i \longleftarrow A_{ii}$ (approximate or exact) for $i = 1, 2, ..., p$;
3. $\tilde{R} \longleftarrow R$ (by dropping some elements) ;
4. $G \longleftarrow \tilde{\mathcal{D}}^{-1}\tilde{R}$;
5. identify nonzero columns of G and store their indices in array c ;
6. Solve $Ax = f$ via a Krylov subspace method with a preconditioner $P = \tilde{\mathcal{D}} + \tilde{R}$ and stopping tolerance ϵ_{out})

Fig. 2. The DDPS algorithm.

solve $Pz = y$
  $(\tilde{\mathcal{D}}^{-1}Pz = \tilde{\mathcal{D}}^{-1}y \Rightarrow (I + G)z = g)$;
  6.1 $g \longleftarrow \tilde{\mathcal{D}}^{-1}y$;
  6.2 $\hat{G} \longleftarrow (I(c, c) + G(c, c))$; $\hat{z} \longleftarrow z(c)$; $\hat{g} \longleftarrow g(c)$;
  6.3 solve the smaller independent system: $\hat{G}\hat{z} = \hat{g}$ (directly or iteratively with stopping tolerance ϵ_{in}) ;
  6.4 $z(c) \longleftarrow \hat{z}$;
  6.5 $z \longleftarrow g - Gz$;
end

Fig. 3. Preconditioning operation: $Pz = y$.

(for some y). Only the details of the preconditioning operations for the DDPS are given in Fig. 3. In this paper, we use a variation of preconditioned BiCGStab [28] as the outer iterative solver.

Each stage, with the exception of stage 6.3, can be executed with perfect parallelism, requiring no interprocessor communications. In stage 6.3, the solution of the smaller system $\hat{G}\hat{z} = \hat{g}$ is the only part of the algorithm that requires communication. The size of $\hat{G}$ is problem dependent, and it is expected to have an influence on the overall scalability of the algorithm. In this paper, the smaller reduced system $\hat{G}\hat{z} = \hat{g}$ is solved iteratively via BiCGStab without preconditioning. The size of $\hat{G}$ is determined by the number of nonzero columns in G. We employ several techniques to reduce the number of nonzero columns in $\hat{G}$.

- We use METIS reordering to reduce the total communication volume, hence reducing the size of $\hat{G}$ by reducing the number of elements in R. (We note that METIS works on undirected graphs; therefore, we apply METIS on $(|A| + |A^T|)/2$.)
- It is required that the diagonal blocks, A_{ii}, are nonsingular. In that case, however, in addition to METIS, applying HSL MC64 reordering and/or a diagonal perturbation can be considered.
- We use the following dropping strategy. Given a tolerance $\delta \in [0, 1]$, if, for any column k in R_i, $\|R(:, k)_i\|_\infty \leq \delta \times \max_j \|R(:, j)_i\|_\infty (i = 1, 2, \ldots, p)$, we do not consider that column when forming $\hat{G}$. Here, R_i is the block row partition of R (i.e. $R = [R_1, R_2, \ldots, R_p]^T$). We call this dropping strategy *a priori dropping*. Another possibility is to drop elements after computing G *a posteriori dropping*. In this paper, however, we only consider *a priori dropping*.

Notice that dropping elements from R in stage 3 to reduce the size of $\hat{G}$ results in an approximation of the solution. Furthermore, we can use approximate LU factorization of the diagonal blocks in stage 2 and solve $\hat{G}\hat{x} = \hat{g}$ iteratively in stage 6.3. Therefore, we place an outer iterative layer (e.g. BiCGStab) in which we use the above algorithm as a solver for systems involving the preconditioner $P = \tilde{\mathcal{D}} + \tilde{R}$, where $\tilde{R}$ consists only of the columns that are not dropped. We stop the outer iterations when the relative residual at the kth iteration $\|r_k\|_\infty / \|r_0\|_\infty \leq \epsilon_{out}$.

The DDPS is a direct solver if (i) nothing is dropped from R, (ii) exact LU factorization of A_{ii} is computed, and (iii) $\hat{G}\hat{z} = \hat{g}$ is solved exactly. In the case of using the DDPS as a direct solver, an outer iterative scheme may not be required, but it is recommended. In this paper we use the direct solver Pardiso for computing LU factorization of the diagonal blocks.

The choices we make in stages 2, 3, and 6.3 result in a solver that can be as robust as a direct solver or as scalable as an iterative solver, or anything in between. Notice that the outer iterative layer also benefits from our partitioning strategy, as METIS reduces the total communication volume in parallel sparse matrix vector multiplications.

We note that $\hat{G}$ consists of dense columns within each partition which we store as a two-dimensional array in memory, and as a result matrix vector multiplications can be done via level 2 BLAS [29,30] (or level 3 in case of multiple right-hand sides).

In order to illustrate the steps of the basic DDPS algorithm (without any approximations) we provide the following system, $Ax = f$, with nine unknowns,

$$\begin{pmatrix} \mathbf{0.2} & \mathbf{1.0} & \mathbf{-1} & 0 & 0.01 & 0 & 0 & 0 & -0.01 \\ \mathbf{0.01} & \mathbf{0.3} & \mathbf{0} & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{-0.1} & \mathbf{0} & \mathbf{0.4} & 0 & 0.3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{0.3} & \mathbf{0.6} & \mathbf{2} & 0 & 0 & 0 \\ 0 & -0.2 & 0 & \mathbf{0} & \mathbf{0.4} & \mathbf{0} & 0 & 0 & 1.1 \\ 0 & 0 & 0 & \mathbf{-0.2} & \mathbf{0.1} & \mathbf{0.5} & 0 & 0 & 0 \\ 1.2 & 0 & 0 & 0 & 0 & 0 & \mathbf{0.4} & \mathbf{0.02} & \mathbf{3.0} \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{2.0} & \mathbf{0.5} & \mathbf{0} \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{0} & \mathbf{0.1} & \mathbf{0.6} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}. \tag{4}$$

Block diagonal matrix D is indicated in bold for three partitions, where each partition is of size 3. After premultiplying both sides with D^{-1} from the left, we obtain the modified system, $(I+G)x = g$ (we do not need to form D^{-1} explicitly to compute $D^{-1}R$):

$$\begin{pmatrix} \mathbf{1} & \mathbf{0} & 0 & 0 & \mathbf{-9.12} & 0 & 0 & 0 & \mathbf{0.12} \\ \mathbf{0} & \mathbf{1} & 0 & 0 & \mathbf{0.304} & 0 & 0 & 0 & \mathbf{0.004} \\ 0 & 0 & 1 & 0 & -1.53 & 0 & 0 & 0 & 0.03 \\ 0 & 0.0909 & 0 & 1 & 0 & 0 & 0 & 0 & -0.5 \\ \mathbf{0} & \mathbf{-0.5} & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{2.75} \\ 0 & 0.1364 & 0 & 0 & 0 & 1 & 0 & 0 & -0.75 \\ 0.5172 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ -2.069 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \mathbf{0.3448} & \mathbf{0} & 0 & 0 & \mathbf{0} & 0 & 0 & 0 & \mathbf{1} \end{pmatrix} \begin{pmatrix} \mathbf{x_1} \\ \mathbf{x_2} \\ x_3 \\ x_4 \\ \mathbf{x_5} \\ x_6 \\ x_7 \\ x_8 \\ \mathbf{x_9} \end{pmatrix} = \begin{pmatrix} \mathbf{-2} \\ \mathbf{3.4} \\ 2 \\ -3.1818 \\ \mathbf{2.5} \\ 0.2273 \\ -1.3103 \\ 7.2414 \\ \mathbf{0.4598} \end{pmatrix}. \tag{5}$$

We note that unknowns 1, 2, 5, and 9 form a smaller independent reduced system (indicated in bold):

$$\begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{-9.12} & \mathbf{0.12} \\ \mathbf{0} & \mathbf{1} & \mathbf{0.304} & \mathbf{-0.004} \\ \mathbf{0} & \mathbf{-0.5} & \mathbf{1} & \mathbf{2.75} \\ \mathbf{0.3448} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{pmatrix} \begin{pmatrix} \mathbf{x_1} \\ \mathbf{x_2} \\ \mathbf{x_5} \\ \mathbf{x_9} \end{pmatrix} = \begin{pmatrix} \mathbf{-2} \\ \mathbf{3.4} \\ \mathbf{2.5} \\ \mathbf{0.4598} \end{pmatrix}, \tag{6}$$

which has the solution $[x_1, x_2, x_5, x_9]^T = [-3.2389, 3.4413, -0.1151, 1.5766]^T$. Finally, we can retrieve the solution of the system via

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{pmatrix} = \begin{pmatrix} -2 \\ 3.4 \\ 2 \\ -3.1818 \\ 2.5 \\ 0.2273 \\ -1.3103 \\ 7.2414 \\ 0.4598 \end{pmatrix} - \begin{pmatrix} 0 & 0 & -9.12 & 0.12 \\ 0 & 0 & 0.304 & 0.004 \\ 0 & 0 & -1.53 & 0.03 \\ 0 & 0.0909 & 0 & -0.5 \\ 0 & -0.5 & 0 & 2.75 \\ 0 & 0.1364 & 0 & -0.75 \\ 0.5172 & 0 & 0 & 0 \\ -2.069 & 0 & 0 & 0 \\ 0.3448 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_5 \\ x_9 \end{pmatrix}, \tag{7}$$

and obtain $x = [-3.2389, 3.4413, 1.7766, -2.7063, -0.1151, 0.9405, 0.365, 0.5402, 1.5766]^T$.

4. Numerical experiments

The set of problems is obtained from the University of Florida Sparse Matrix Collection [31]. We choose the largest nonsymmetric matrix from each application domain. The list of the matrices and their properties are given in Table 1. For each matrix, we generate the corresponding right-hand side using a solution vector of all ones to ensure that $f \in span(A)$. All numerical experiments were performed on an Intel Xeon (X5560@2.8 GHz) cluster with Infiband interconnection and 16 GB memory per node. The number of MPI processes is equal to the number of cores used and is also equal to the number of partitions for the DDPS.

For the iterative solvers, the outer iterations are terminated when the number of iterations reaches 1, 000 or the relative residual meets the stopping criterion ($\|f - Ax\|_\infty / \|f\|_\infty \leq 10^{-5}$). Failures of the solvers are indicated by F1 or F2 when the solver runs out of memory or the final relative residual is larger than 10^{-5}, respectively. We limit the maximum number of iterations to 100 and the stopping tolerance to $\epsilon_{in} = 10^{-4}$ for the inner iterations of the DDPS.

We use ILUPACK with the following parameters. Reorderings: weighted matching and AMD [32], droptol: 10^{-1}, estimate for the condition numbers of the factors: 50, and an elbow space of 10. These are recommended by the user guide for general

Table 1
Linear systems from the University of Florida Sparse Matrix Collection. *n*, *nnz*, and *dd* stand for matrix size, number of nonzeros, and the degree of diagonal dominance, respectively.

System	*n*	*nnz*	*dd*	*Problem domain*
ATMOSMODL	1,489,752	10,319,760	0	Computational fluid dynamics
HVDC2	189,860	1,339,638	0	Power network
LANGUAGE	399,130	1,216,334	6.2×10^{-4}	Directed weighted graph
OHNE2	181,343	6,869,939	1.4×10^{-11}	Semiconductor device simulation
RAJAT31	4,690,002	20,316,253	0	Circuit simulation
THERMOMECH_DK	204,316	2,846,228	0.32	Thermal
TMT_UNSYM	917,825	4,584,801	1	Electromagnetic
TORSO3	259,156	4,429,042	9.9×10^{-2}	2D/3D problem
XENON2	157,464	3,866,688	8.2×10^{-2}	Material science

Table 2
Total solve times (in seconds) for MUMPS, Pardiso, the DDPS, and ILUPACK.

MPI processes	MUMPS					Pardiso	DDPS				ILUPACK
	1	2	4	8	16	1	2	4	8	16	1
ATMOSMODL	**F1**	**F1**	**F1**	**F1**	171.3	1291.0	391.6	781.0	149.1	100.7	13.6
HVDC2	1.5	1.6	1.4	1.5	1.9	2.0	1.4	1.6	6.9	**F2**	**F2**
LANGUAGE	504.6	273.9	**F2**	**F2**	**F2**.	1191.3	124.7	15.2	6.4	2.0	3.4
OHNE2	42.5	27.3	19.3	13.2	8.5	43.9	21.2	9.4	**F2**	**F2**	**F2**
RAJAT31	78.3	67.6	59.3	54.1	53.7	57.9	258.7	150.5	106.2	45.1	**F2**
THERMO	2.9	2.3	2.1	2.1	2.8	3.0	2.0	10.5	20.5	11.2	6.8
TMT_UNSYM	14.5	12.1	10.3	9.8	9.7	10.8	170.7	140.2	99.0	77.8	**F2**
TORSO3	40.2	26.3	18.2	12.4	9.6	49.4	20.6	10.0	4.0	2.1	2.2
XENON2	13.5	8.2	6.1	4.5	4.2	14.7	14.9	7.6	3.9	2.9	**F2**

Table 3
Speedup of the DDPS compared to Pardiso.

MPI processes	Pardiso	DDPS			
	1	2	4	8	16
ATMOSMODL	1.0	3.3	1.7	8.7	12.8
HVDC2	1.0	1.4	1.2	0.3	**F2**
LANGUAGE	1.0	9.6	78.2	185.7	609.4
OHNE2	1.0	2.1	4.8	**F2**	**F2**
RAJAT31	1.0	0.2	0.4	0.6	1.3
THERMO	1.0	1.5	0.3	0.2	0.3
TMT_UNSYM	1.0	0.1	0.1	0.1	0.1
TORSO3	1.0	2.4	4.9	12.2	23.6
XENON2	1.0	1.0	1.9	3.7	5.1

sparse linear systems. ILUPACK uses GMRES(30) with a variation of an incomplete LU factorization based preconditioner. MUMPS and Pardiso have been used with their default parameters and using METIS reordering.

In Table 2, we present the total solve time for MUMPS, Pardiso, the DDPS ($\delta = 0.9$), and ILUPACK. For five systems out of nine, the DDPS is faster than MUMPS (for 16 MPI processors). In addition, the DDPS is more robust than ILUPACK and almost as robust as the MUMPS direct solver: using 16 partitions the DDPS fails only in two cases while ILUPACK and MUMPS fail in 5 cases and 1 case, respectively. The DDPS never runs out of memory while MUMPS runs out of memory for one of the problems unless more than 8 partitions are used.

The speedup with respect to the Pardiso solver using a single core is given in Table 3. We note that two problems achieve superlinear speed improvement due to cache effects.

In Table 4, the number of outer BiCGStab iterations for the DDPS is provided as one increases the number of partitions. With the exception of two cases, namely hvdc2 and thermomech_dk, the number of iterations depends weakly (less than linearly) on the number of partitions (or MPI processes).

The average number of inner BiCGStab iterations is given in Table 5. Since we make sure that the reduced system size is small via various techniques described earlier, the number of iterations is relatively small for all systems, with weak dependence on the number of processes.

In Table 6, we show the effect of varying the drop tolerance, δ, while the number of partitions is fixed at 16. A small δ results in a variation of the DDPS that is more like a direct solver. Although this causes the number of iterations to decrease, it also increases the memory requirement, and the solver runs out of memory. For small δ, memory problem appears in two cases, namely rajat31 and atmosmodl. In five cases, the number of outer iterations decreases as we decrease δ. In the remaining two cases, the DDPS failed, even though δ was set to be a small number.

Table 4
Number of outer BiCGStab iterations for the DDPS and ILUPACK.

MPI processes	DDPS				ILUPACK
	2	4	8	16	1
ATMOSMODL	18	18	21.5	21.5	26
HVDC2	0.5	12.5	260	**F2**	**F2**
LANGUAGE	5	7	6	6	4
OHNE2	0.5	0.5	**F2**	**F2**	**F2**
RAJAT31	71.5	86.5	106.5	99	**F2**
THERMO	84.5	248.5	752.5	856	31
TMT_UNSYM	89.5	192	212	294	**F2**
TORSO3	8.5	10	8.5	8.5	5
XENON2	53	63	67	90.5	**F2**

Table 5
Average number of inner BiCGStab iterations for the DDPS.

MPI processes	2	4	8	16
ATMOSMODL	0.5	3.28	0.5	4.74
HVDC2	0.5	3.12	14.93	**F2**
LANGUAGE	0.5	0.5	0.92	1
OHNE2	3.5	3.5	**F2**	**F2**
RAJAT31	3.54	3.16	7.18	4.95
THERMO	1	1	2.93	3.7
TMT_UNSYM	13.2	3.32	12.14	15.32
TORSO3	4.32	2.9	3.35	4.53
XENON2	1	1	1	4.7

Table 6
Number of outer BiCGStab iterations for the DDPS for 16 MPI processes.

δ	0.99	0.9	0.6	0.3	0.1	1.0E−5
ATMOSMODL	23.5	21.5	19	**F1**	**F1**	**F1**
HVDC2	**F2**	**F2**	**F2**	**F2**	**F2**	8
LANGUAGE	7	6	6	4	2.5	1
OHNE2	**F2**	**F2**	**F2**	**F2**	**F2**	**F2**
RAJAT31	99	99	99	**F1**	**F1**	**F1**
THERMOMECH_DK	645.5	856	**F2**	**F2**	**F2**	414
TMT_UNSYM	294	294	231	**F2**	**F2**	**F2**
TORSO3	8.5	8.5	8.5	6.5	4	1
XENON2	99	90.5	105.5	**F2**	**F2**	1

5. Conclusion

We have introduced a new hybrid sparse linear system solver called the DDPS. We have shown that our new sparse linear system solver is often faster than direct solvers and more robust than classical preconditioned Krylov subspace methods. The DDPS is flexible, as it can be used in a variety of configurations. Depending on the solver for the diagonal blocks, a new variation of the algorithm will arise. The choice we make for solving the inner reduced system further increases the number of possibilities. Although we have used METIS to show the application of the algorithm on general sparse systems, the DDPS algorithm is ideally suited for problems in which the matrices are already distributed via domain decomposition to minimize interprocessor communication.

Acknowledgments

The author would like to thank Ahmed Sameh, Ananth Grama, David Kuck, Eric Cox, Faisal Saied, Henry Gabb, Kenji Takizawa, and Tayfun Tezduyar for the numerous discussions and for their support. This work has been partially supported by the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no: RI-261557 and METU BAP-08-11-2011-128 grant.

References

[1] P.R. Amestoy, A. Guermouche, J.-Y. L'Excellent, S. Pralet, Hybrid scheduling for the parallel solution of linear systems, Parallel Computing 32 (2) (2006) 136–156.
[2] P.R. Amestoy, I.S. Duff, J.-Y. L'Excellent, J. Koster, A fully asynchronous multifrontal solver using distributed dynamic scheduling, SIAM Journal on Matrix Analysis and Applications 23 (1) (2001) 15–41.

[3] P.R. Amestoy, I.S. Duff, Multifrontal parallel distributed symmetric and unsymmetric solvers, Computer Methods in Applied Mechanics and Engineering 184 (2000) 501–520.
[4] O. Schenk, K. Gärtner, Solving unsymmetric sparse systems of linear equations with PARDISO, Future Generation Computer Systems 20 (3) (2004) 475–487.
[5] O. Schenk, K. Gärtner, On fast factorization pivoting methods for sparse symmetric indefinite systems, Electronic Transactions on Numerical Analysis 23 (2006) 158–179.
[6] X.S. Li, J.W. Demmel, SuperLU-DIST: a scalable distributed-memory sparse direct solver for unsymmetric linear systems, Association of Computing Machinery. Transactions on Mathematical Software 29 (2) (2003) 110–140.
[7] M. Benzi, D.B. Szyld, A. van Duin, Orderings for incomplete factorization preconditioning of nonsymmetric problems, SIAM Journal on Scientific Computing 20 (5) (1999) 1652–1670.
[8] M. Benzi, J.C. Haws, M. Tůma, Preconditioning highly indefinite and nonsymmetric matrices, SIAM Journal on Scientific Computing 22 (4) (2000) 1333–1353.
[9] M. Bollhöfer, Y. Saad, O. Schenk, ILUPACK, Vol. 2.1, Preconditioning Software Package, http://ilupack.tubs.de (May 2006).
[10] G. Gravvanis, P. Matskanidis, K. Giannoutakis, E. Lipitakis, Finite element approximate inverse preconditioning using posix threads on multicore systems, in: Proceedings of the International Multiconference on Computer Science and Information Technology, 5, 2010, pp. 297–302.
[11] G. Gravvanis, High Performance Inverse Preconditioning, Archives of Computational Methods in Engineering 16 (1) (2009) 77–108.
[12] G. 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.
[13] G. Gravvanis, Explicit preconditioned generalized domain decomposition methods, International Journal of Applied Mathematics 4 (1) (2000) 57–72.
[14] M. Benzi, C. Meyer, M. Tůma, et al., A sparse approximate inverse preconditioner for the conjugate gradient method, SIAM Journal on Scientific Computing 17 (5) (1996) 1135–1149.
[15] A.H. Sameh, D.J. Kuck, On stable parallel linear system solvers, Journal of the ACM 25 (1) (1978) 81–91.
[16] D.J.K.S.C. Chen, A.H. Sameh, Practical parallel band triangular system solvers, ACM Transactions on Mathematical Software 4 (3) (1978) 270–277.
[17] D.H. Lawrie, A.H. Sameh, The computation and communication complexity of a parallel banded system solver, Association of Computing Machinery. Transactions on Mathematical Software 10 (2) (1984) 185–195.
[18] M.W. Berry, A. Sameh, Multiprocessor schemes for solving block tridiagonal linear systems, The International Journal of Supercomputer Applications 1 (3) (1988) 37–57.
[19] J.J. Dongarra, A.H. Sameh, On some parallel banded system solvers, Parallel Computing 1 (3) (1984) 223–235.
[20] E. Polizzi, A.H. Sameh, A parallel hybrid banded system solver: the spike algorithm, Parallel Computing 32 (2) (2006) 177–194.
[21] E. Polizzi, A.H. Sameh, Spike: a parallel environment for solving banded linear systems, Computers & Fluids 36 (1) (2007) 113–120.
[22] M. Manguoglu, M. Koyutürk, A.H. Sameh, A. Grama, Weighted matrix ordering and parallel banded preconditioners for iterative linear system solvers, SIAM Journal on Scientific Computing 32 (3) (2010) 1201–1216.
[23] M. Manguoglu, A. Sameh, O. Schenk, A parallel hybrid sparse linear system solver, in: LNCS – Proceedings of EURO-PAR09 5704, 2009, pp. 797–808.
[24] O. Schenk, M. Manguoglu, A. Sameh, M. Christian, M. Sathe, Parallel scalable PDE-constrained optimization: antenna identification in hyperthermia cancer treatment planning, Computer Science Research and Development 23 (2009) 177–183.
[25] M. Manguoglu, K. Takizawa, A. Sameh, T. Tezduyar, Nested and parallel sparse algorithms for arterial fluid mechanics computations with boundary layer mesh refinement, International Journal for Numerical Methods in Fluids 65 (2011) 135–149. doi:10.1002/fld.2415.
[26] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM Journal on Scientific Computing 20 (1998) 359–392.
[27] G. Karypis, V. Kumar, Parallel multilevel *k*-way partitioning scheme for irregular graphs, SIAM Journal on Scientific Computing 41 (1999) 278–300.
[28] R. Barrett, M. Berry, T.F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H.V. der Vorst, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd ed., SIAM, Philadelphia, PA, 1994.
[29] C.L. Lawson, R.J. Hanson, D.R. Kincaid, F.T. Krogh, Basic linear algebra subprograms for fortran usage, Association of Computing Machinery. Transactions on Mathematical Software 5 (1979) 308–323. doi:10.1145/355841.355847. URL: http://doi.acm.org/10.1145/355841.355847.
[30] J.J. Dongarra, J. DuCroz, S. Hammarling, I.S. Duff, A set of level 3 basic linear algebra subprograms, Association of Computing Machinery. Transactions on Mathematical Software 16 (1990) 1–17. doi:10.1145/77626.79170. URL: http://doi.acm.org/10.1145/77626.79170.
[31] T.A. Davis, University of Florida sparse matrix collection, NA Digest, 1997.
[32] P.R. Amestoy, T.A. Davis, I.S. Duff, An approximate minimum degree ordering algorithm, SIAM Journal on Matrix Analysis and Applications 17 (4) (1996) 886–905.