Block-Based Approach to Solving Linear Systems

Sunil R. Tiyyagura and Uwe Küster

High Performance Computing Center Stuttgart,
University of Stuttgart,
Nobelstrasse 19, 70569 Stuttgart, Germany
sunil,kuester@hlrs.de

Abstract. This paper addresses the efficiency issues in solving large sparse linear systems parallely on scalar and vector architectures. Linear systems arise in numerous applications that need to solve PDEs on complex domains. The major time consuming part of large scale implicit Finite Element (FE) or Finite Volume (FV) simulation is solving the assembled global system of equations. First, the performance of widely used public domain solvers which target performance on scalar machines is analyzed on a typical vector machine. Then, a newly developed parallel sparse iterative solver (Block-based Linear Iterative Solver – BLIS) targeting performance on both scalar and vector systems is introduced and the time needed for solving linear systems is compared on different architectures. Finally, the reasons behind the scaling behaviour of parallel iterative solvers is analysed.

Keywords: Sparse linear algebra, Scalability, Block algorithms, Indirect memory addressing.

1 Introduction

The most powerful supercomputers today are an agglomeration of thousands of scalar processors connected with innovative interconnects. The major concern with this developing trend is the scalability of communication intensive applications which need global synchronization at many points, for example a conjugate gradient solver. Vector architectures seem to be a better alternative for such applications by providing a cluster of very powerful SMP (Symmetric Multiprocessing) nodes capable of processing a huge amount of computation and thereby reducing the overhead for synchronization. The rapidly increasing gap between sustained and peak performance of the high performance computing architectures [1] is another concern facing computational scientists today. The sustained computation to communication ratio and the computation to memory bandwidth ratio are much better for vector architectures when compared to clusters of commodity processors [2]. This emphasizes the need to look towards vector computing as a future alternative for certain class of applications.

In this paper we focus on the performance of parallel linear iterative solver on both scalar and vector machines. The work described here was done on the basis

Y. Shi et al. (Eds.): ICCS 2007, Part I, LNCS 4487, pp. 128–135, 2007.

of the research finite element program *Computer Aided Research Analysis Tool* (CCARAT), that is jointly developed and maintained at the Institute of Structural Mechanics of the University of Stuttgart and the Chair of Computational Mechanics at the Technical University of Munich. The research code CCARAT is a multipurpose finite element program covering a wide range of applications in computational mechanics, like e.g. multi-field and multi-scale problems, structural and fluid dynamics, shape and topology optimization, material modeling and finite element technology. The code is parallelized using MPI and runs on a variety of platforms.

The major time consuming portions of a finite element simulation are calculating the local element contributions to the globally assembled matrix and solving the assembled global system of equations. As much as 90% of the time in a very large scale simulation can be spent in the linear solver, specially if the problem to be solved is ill-conditioned. While the time taken in element calculation scales linearly with the size of the problem, often the time in the sparse solver does not. Major reason being the kind of preconditioning needed for a successful solution. In Sect. 2 of this paper, the performance of public domain solvers is analysed on vector architecture. In Sect. 3, a newly developed parallel iterative solver (Block-based Linear Iterative solver – BLIS) targeting performance on both the architectures is introduced. Sect. 4 discusses performance and scaling results of BLIS.

2 Public Domain Solvers

Most public domain solvers like AZTEC [3], PETSc, Trilinos [4], etc. do not perform on vector architecture as well as they do on superscalar architectures. The main reason being their design considerations that primarily target performance on superscalar architectures thereby neglecting the following performance critical features of vector systems.

2.1 Average Vector Length

This is an important metric that has a huge effect on performance. In sparse linear algebra, the matrix object is sparse where as the vectors are still dense. So, any operations involving only vectors, like the dot product, result in a good average vector length as the innermost vectorized loop runs over long vectors. The problem is with operations involving the sparse matrix object like the matrix vector product (MVP) which is a key kernel of Krylov subspace methods. The data structure used to store the sparse matrix plays an important role in the performance of such operators. Naturally resulting row based data structures used in the above mentioned solvers result in low average vector length which is further problem dependent. This leads to partially empty pipeline processing and hence hinders performance of a critical operator on vector machines.

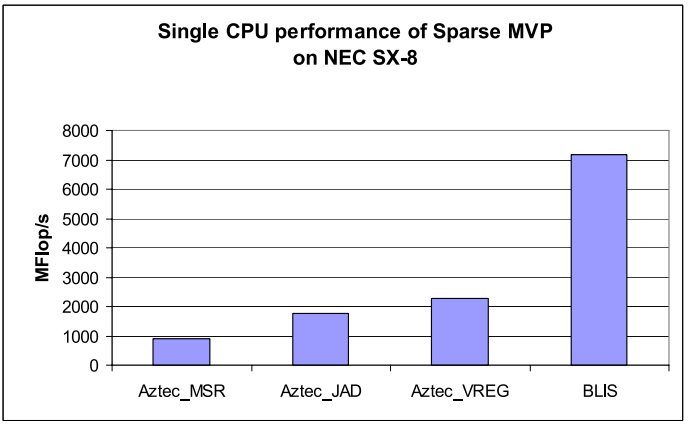

Fig. 1. Single CPU performance of Sparse MVP on NEC SX-8

A well known solution to this problem is to use pseudo diagonal data structure to store the sparse matrix [5]. We tried to introduce similar functionality into AZTEC. Figure 1 shows the single CPU performance in Sparse MVP on the NEC SX-8 with the native row based (Modified Sparse Row - MSR) and the introduced pseudo diagonal based (JAgged Diagonal - JAD) matrix storage formats. Using vector registers to reduce memory operations for loading and storing the result vector further improves the performance of JAD based sparse MVP to 2.2 GFlop/s. Further optimizations result in a maximum performance of 20% vector peak (which is 16 GFlop/s) for sparse MVP on NEC SX-8 [6].

2.2 Indirect Memory Addressing

The performance of sparse MVP on vector as well as on superscalar architectures is not limited by memory bandwidth, but by latencies. Due to the sparse storage, the vector to be multiplied in a sparse MVP is accessed randomly (non-strided access). Hence, the performance of this operation completely depends on the implementation and performance of the "vector gather" assembly instruction on vector machines. Though the memory bandwidth and byte/flop ratios of a vector architecture are in general far more superior than any superscalar architecture, superscalar architectures have the advantage of cache re-usage for this operation. But, the cost of accessing the main memory is so high on superscalar machines that without special optimizations [7], sparse MVP performs at around 5% peak.

It is interesting in this context to look into architectures which combine the advantages of vector pipelining with memory caching, like the CRAY X1. Sparse MVP with pseudo diagonal format on this machine performs at about 14% peak [8]. This is comparable to the performance achieved for point-based JAD algorithm on the NEC SX-8. An upper bound of around 30% peak is estimated for most sparse matrix computations on the CRAY X1.

3 Block-Based Linear Iterative Solver (BLIS)

In the sparse MVP kernel discussed so far, the major hurdle to performance is not memory bandwidth but the latencies involved due to indirect memory addressing. Block based computations exploit the fact that many FE problems typically have more than one physical variable to be solved per grid point. Thus, small blocks can be formed by grouping the equations at each grid point. Operating on such dense blocks considerably reduces the amount of indirect addressing required for sparse MVP [6]. This improves the performance of the kernel dramatically on vector machines [9] and also remarkably on superscalar architectures [10,11]. BLIS uses this approach primarily to reduce the penalty incurred due to indirect memory access.

3.1 Available Functionality

Presently, BLIS is working with finite element applications that have 4 unknowns to be solved per grid point. JAD sparse storage format is used to store the dense blocks. This assures sufficient average vector length for operations done using the sparse matrix object (Preconditioning, Sparse MVP). The single CPU performance of sparse MVP, Fig. 1, with a matrix consisting of 4x4 dense blocks is around 7.2 GFlop/s (about 45% vector peak) on the NEC SX-8.

BLIS is based on MPI and includes well known Krylov subspace methods such as BiCGSTAB and GMRES. Block scaling, block Jacobi and block ILU(0) on subdomains are the available matrix preconditioner methods. Exchange of halos in sparse MVP can be done using MPI non-blocking or MPI persistent communication.

3.2 Future Work

BLIS presently works with applications that have 4 unknowns to be evaluated per grid point. This will further be extended to handle any number of unknowns. Blocking functionality will be provided in the solver in order to free the users from preparing blocked matrices. This makes adaptation of the library to an application easy. Scheduling communication via coloring and reducing global synchronization at different places in Krylov subspace algorithms has to be extensively looked into for further improving scaling of the solver [12].

4 Performance

This section explains the performance of BLIS on both scalar and vector architectures. The machines tested are a cluster of NEC SX-8 SMPs and a cluster of Intel 3.2 GHz Xeon EM64T processors. The network interconnect available on NEC SX-8 is a proprietary multi-stage crossbar called IXS and on the Xeon cluster a Gigabit ethernet. Vendor tuned MPI library is used on the SX-8 and Voltaire MPI library on the Xeon cluster to run the following example.

Table 1. Different discretizations of the introduced numerical example

Discretization	No.of elements	No. of nodes	No. of unknowns
1	33750	37760	151040
2	81200	88347	353388
3	157500	168584	674336
4	270000	285820	1143280
5	538612	563589	2254356
6	911250	946680	3786720

4.1 Numerical Example

In this example the laminar, unsteady 3-dimensional flow around a cylinder with a square cross-section is examined. The setup was introduced as a benchmark example by the DFG Priority Research Program "Flow Simulation on High Performance Computers" to compare different solution approaches of the Navier-Stokes equations[13]. The fluid is assumed to be incompressible Newtonian with a kinematic viscosity $\nu = 10^{-3}\ m^2/s$ and a density of $\rho = 1.0\ kg/m^3$. The rigid cylinder (cross-section: 0.1 m x 0.1m) is placed in a 2.5 m long channel with a square cross-section of 0.41 m by 0.41 m. On one side a parabolic inflow condition with the mean velocity $u_m = 2.25\ m/s$ is applied. No-slip boundary conditions are assumed on the four sides of the channel and on the cylinder.

4.2 BLIS Scaling on Vector Machine

The scaling behaviour of the solver on NEC SX-8 was tested for the above mentioned numerical example using stabilized 3D hexahedral fluid elements in CCARAT. Table 1 lists all the six discretizations of the example used.

Figure 3 plots strong scaling of BLIS upto 128 processors using BiCGSTAB algorithm along with block Jacobi preconditioning on the NEC SX-8. All problems were run for 5 time steps where each non-linear time step needs about 3-5 newton iterations for convergence. The number of iterations needed for convergence in BLIS for each newton step varies largely between 200-2000 depending on the problem size (number of equations). The plots show the drop in sustained floating point performance of BLIS from above 6 GFlop/s to 3 GFlop/s

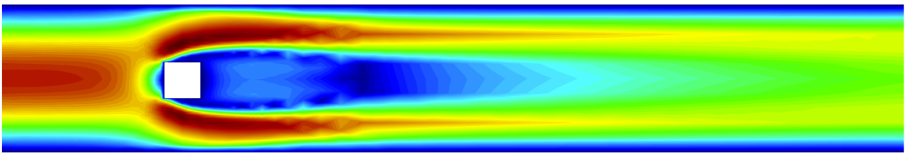

Fig. 2. Velocity in the midplane of the channel

depending on the number of processors used for each problem size. The right plot of Fig. 3 explains the reason for this drop in performance in terms of drop in computation to communication ratio in BLIS. It has to be noted that major part of the communication with the increase in processor count is spent in MPI global reduction call which needs global syncronization.

Figure 4 plots the same data for increasing problem size per CPU. Each curve represents performance using particular number of CPUs with varying problem size. Here the effect of global synchronization on scaling can be clearly seen. As the processor count increases, the performance curves climb slowly till the performance saturates. This behaviour can be directly attributed to the time spent in communication which is clear from the right plot. These plots are hence important as they accentuate the problem with Krylov subspace algorithms where large problem sizes are needed to sustain high performance on large processor counts. This is a drawback for certain class of applications where the demand for HPC (High Performance Computing) is due to the largely transient nature of the problem. For instance, in some Fluid-Structure interaction examples, the problem size is not too large but thousands of time steps are necessary to simulate the transient effects.

4.3 Performance Comparison on Scalar and Vector Machines

Here, we compare the performance of the linear solver on both scalar and vector machines. A relatively smaller discretization (353K unknowns) was used. Elapsed time (seconds) in linear iterative solver for 5 time steps is listed on the y-axis of the plot in Fig. 5. Fastest solution method (BiCGSTAB) and preconditioning (ILU on the Xeon cluster and block Jacobi on NEC SX-8) was used on the respective machines.

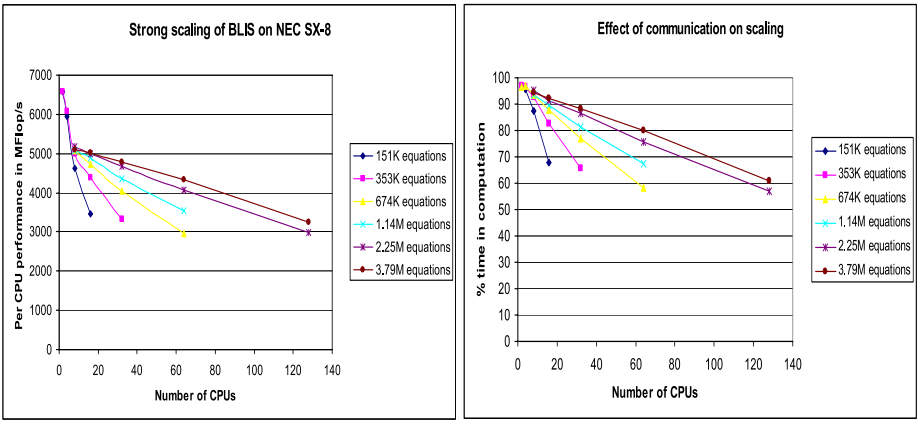

Fig. 3. Strong scaling of BLIS (left) and computation to communication ratio in BLIS on NEC SX-8 (right)

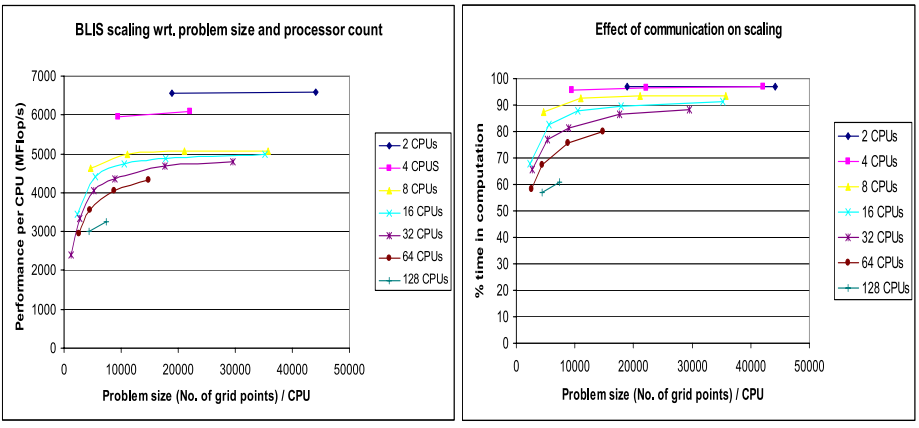

Fig. 4. Scaling of BLIS wrt. problem size on NEC SX-8 (left) Computation to communication ratio in BLIS on NEC SX-8 (right)

The performance of BLIS is compared to AZTEC on both the machines. BLIS is about 20% faster than AZTEC on the Xeon cluster due to better convergence and block based approach. BLIS is about 10 times faster than AZTEC on the NEC SX-8 for lower processor counts when the problem size per processor provides sufficient average vector length. This is the main reason behind developing a new linear solver. The solution of the linear system is about 4-5 times faster on the NEC SX-8 than on the Xeon cluster for lower processor counts. As the number of processors increases, the performance of BLIS on NEC SX-8 drops due to decrease in computation to communication ratio and also drop in average vector length. Contrarily, this is an advantage (better cache utilization) on the Xeon cluster due to a smaller problem size on each processor. This can be noticed from the super linear speedup on the Xeon cluster.

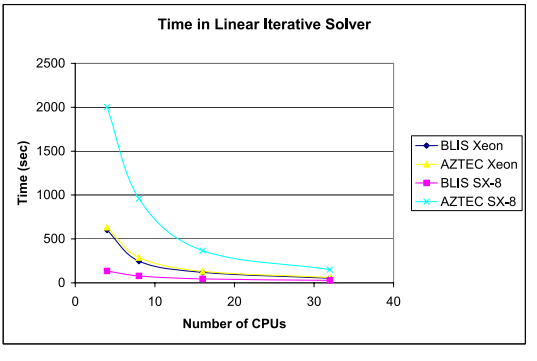

Fig. 5. Elapsed time (seconds) in linear solver

5 Summary

The reasons behind the dismal performance of most of the public domain sparse iterative solvers on vector machines were analyzed. We then introduced the Block-based Linear Iterative Solver (BLIS) which is currently under development targeting performance on all architectures. Results show an order of magnitude performance improvement over other public domain libraries on the tested vector system. A moderate performance improvement is also seen on the scalar machines.

References

1. Oliker, L., Canning, A., Carter, J., Shalf, J., Ethier, S.: Scientific computations on modern parallel vector systems. In: Proceedings of the ACM/IEEE Supercomputing Conference (SC 2004), Pittsburgh, USA (2004)
2. Rabenseifner, R., Tiyyagura, S.R., Müller, M.: Network bandwidth measurements and ratio analysis with the hpc challenge benchmark suite (hpcc). In Martino, B.D., Mueller, D.K., Dongarra, J., eds.: Proceedings of the 12th European PVM/MPI Users' Group Meeting (EURO PVM/MPI 2005). LNCS 3666, Sorrento, Italy, Springer (2005) 368–378
3. Tuminaro, R.S., Heroux, M., Hutchinson, S.A., Shadid, J.N.: Aztec user's guide: Version 2.1. Technical Report SAND99-8801J, Sandia National Laboratories (1999)
4. Heroux, M.A., Willenbring, J.M.: Trilinos users guide. Technical Report SAND2003-2952, Sandia National Laboratories (2003)
5. Saad, Y.: Iterative Methods for Sparse Linear Systems, Second Edition. SIAM, Philadelphia, PA (2003)
6. Tiyyagura, S.R., Küster, U., Borowski, S.: Performance improvement of sparse matrix vector product on vector machines. In Alexandrov, V., van Albada, D., Sloot, P., Dongarra, J., eds.: Proceedings of the Sixth International Conference on Computational Science (ICCS 2006). LNCS 3991, Reading, UK, Springer (2006)
7. Im, E.J., Yelick, K.A., Vuduc, R.: Sparsity: An optimization framework for sparse matrix kernels. International Journal of High Performance Computing Applications **(1)18** (2004) 135–158
8. Agarwal, P., et al.: ORNL Cray X1 evaluation status report. Technical Report LBNL-55302, Lawrence Berkeley National Laboratory (May 1, 2004)
9. Nakajima, K.: Parallel iterative solvers of geofem with selective blocking preconditioning for nonlinear contact problems on the earth simulator. GeoFEM 2003-005, RIST/Tokyo (2003)
10. Jones, M.T., Plassmann, P.E.: Blocksolve95 users manual: Scalable library software for the parallel solution of sparse linear systems. Technical Report ANL-95/48, Argonne National Laboratory (1995)
11. Tuminaro, R.S., Shadid, J.N., Hutchinson, S.A.: Parallel sparse matrix vector multiply software for matrices with data locality. Concurrency: Practice and Experience **(3)10** (1998) 229–247
12. Demmel, J., Heath, M., van der Vorst, H.: Parallel numerical linear algebra. Acta Numerica **2** (1993) 53–62
13. Schäfer, M., Turek, S.: Benchmark Computations of Laminar Flow Around a Cylinder. Notes on Numerical Fluid Mechanics **52** (1996) 547–566