Development of General Purpose Numerical Software Infrastructure for Large Scale Scientific Computing

Project Representative
Akira Nishida Research Institute for Information Technology, Kyushu University

Author
Akira Nishida Research Institute for Information Technology, Kyushu University

The Scalable Software Infrastructure Project was initiated as a national project in Japan for the purpose of constructing a scalable parallel software infrastructure for scientific computing. The project covered three areas: iterative solvers for linear systems, fast integral transforms, and their portable implementation.

Modular programming was adopted to enable users to write their codes by combining elementary mathematical operations. Implemented algorithms were selected from the viewpoint of scalability on massively parallel computing environments. Since the first release in September 2005, the codes have been used by thousands of research projects around the world.

Keywords: high performance computing, parallel algorithms, modular programming

1. Overview

Construction of a software infrastructure for highly parallel computing environments requires precisely prediction of future hardware technologies, and design of scalable and portable software for these technologies.

The Scalable Software Infrastructure for Scientific Computing (SSI). Project was initiated in November 2002, as a national project in Japan, for the purpose of constructing a scalable software infrastructure [1], [2], [3]. Based on the policies, we have used various types of parallel computers, and carefully designed our libraries on them, to maintain portability and usability. The architectures included shared-memory parallel computers, distributed-memory parallel computers, and vector supercomputers. Since 2006, the SSI project has been selected for a joint research with the Earth Simulator Center to port our libraries on massively parallel vector computing environments. Since 2003, we have signed a contract with the IBM Watson Research Center on the joint study of library implementation on massively parallel environments with tens of thousands of processors. The results of the SSI project will be evaluated on larger computers in the near future.

In the SSI project, we have studied object-oriented implementation of libraries, autotuning mechanisms, and languages for the implemented libraries. The results were applied to a modular iterative solver library Lis and a fast Fourier transform library FFTSS. The libraries were written in C and equipped with Fortran interfaces. We have also developed a simple interface for library collections SILC, with an extension to scripting language.

2. Lis: a Library of Iterative Solvers for Linear Sytems

In the fields such as fluid dynamics and structural analysis, we must solve large-scale systems of linear equations with sparse matrices to compute high resolution numerical solutions of partial differential equations. We have developed Lis, a library of iterative solvers and preconditioners with various sparse matrix storage formats. Supported solvers, preconditioners, and matrix storage formats are listed in Table. 1-4. We present an example of the program using Lis in Fig. 1.

There are a variety of portable software packages that are applicable to the iterative solver of sparse linear systems. SPARSKIT is a toolkit for sparse matrix computations written in Fortran. PETSc is a C library for the numerical solution of partial differential equations and related problems, which is to be used in application programs written in C, C++, and Fortran. PETSc includes parallel implementations of iterative solvers and preconditioners based on MPI. Aztec is another library of parallel iterative solvers and preconditioners written in C. The library is fully parallelized using MPI. From the viewpoint of functionality, our library and all three of the libraries mentioned above support different sets of matrix storage formats, iterative solvers, and preconditioners. In addition, our library is parallelized using OpenMP with the first-touch policy and takes the multicore architecture into consideration. Many feedbacks from the users have been applied to Lis, and Lis has been tested on various platforms from small personal computers with Linux, Macintosh, and Windows operating systems to massively parallel computers, such as NEC SX, IBM Blue Gene, and Cray XT series. Major tested platforms and target options are listed in Table. 5-6. The code of Lis has attained the vectorization

Table 1 Solvers for linear equations.

1.0.x	CG	Added in 1.1.x	CR
	BiCG		BiCR
	CGS		CRS
	BiCGSTAB		BiCRSTAB
	BiCGSTAB(l)		GPBiCR
	GPBiCG		BiCRSafe
	Orthomin(m)		FGMRES(m)
	GMRES(m)		IDR(s)
	TFQMR		MINRES
	Jacobi		
	Gauss-Seidel		
	SOR		

Table 2 Solvers for eigenproblems.

Added in 1.2.0	Power Iteration
	Inverse Iteration
	Approximate Inverse Iteration
	Conjugate Gradient
	Lanczos Iteration
	Subspace Iteration
	Conjugate Residual

Table 3 Preconditioners.

1.0.x	Jacobi	Added in 1.1.0	Crout ILU
	ILU(k)		ILUT
	SSOR		Additive Schwarz
	Hybrid		User defined preconditioner
	I+S		
	SA-AMG		
	SAINV		

Table 4 Matrix storage formats.

Point	Compressed Row Storage
	Compressed Column Storage
	Modified Compressed Sparse Row
	Diagonal
	Ellpack-Itpack generalized diagonal
	Jagged Diagonal Storage
	Dense
	Coordinate
Block	Block Sparse Row
	Block Sparse Column
	Variable Block Row

Table 5 Major tested platforms.

C compilers	OS
Intel C/C++ Compiler 7.0, 8.0, 9.1, 10.1, 11.1, Intel C++ Composer XE	Linux Windows
IBM XL C/C++ V7.0, 9.0	AIX Linux
Sun WorkShop 6, Sun ONE Studio 7, Sun Studio 11, 12	Solaris
PGI C++ 6.0, 7.1, 10.5	Linux
gcc 3.3, 4.3	Linux Mac OS X Windows
Microsoft Visual C++ 2008, 2010	Windows
Fortran compilers (optional)	**OS**
Intel Fortran Compiler 8.1, 9.1, 10.1, 11.1, Intel Fortran Composer XE	Linux Windows
IBM XL Fortran V9.1, 11.1	AIX Linux
Sun WorkShop 6, Sun ONE Studio 7, Sun Studio 11, 12	Solaris
PGI Fortran 6.0, 7.1, 10.5	Linux
g77 3.3 gfortran 4.3, 4.4 g95 0.91	Linux Mac OS X Windows

Table 6 Major target options.

<target>	Configure scripts
cray_xt3	./configure CC=cc FC=ftn CFLAGS="-O3 -B -fastsse -tp k8-64" FCFLAGS="-O3 -fastsse -tp k8-64 -Mpreprocess" FCLDFLAGS="-Mnomain" ac_cv_sizeof_void_p=8 cross_compiling=yes --enable-mpi ax_f77_mangling="lower case, no underscore, extra underscore"
fujitsu_pq	./configure CC=fcc FC=frt ac_cv_sizeof_void_p=8 CFLAGS="-O3 -Kfast,ocl,preex" FFLAGS="-O3 -Kfast,ocl,preex -Cpp" FCFLAGS="-O3 -Kfast,ocl,preex -Cpp -Am" ax_f77_mangling="lower case, underscore, no extra underscore"
hitachi	./configure CC=cc FC=f90 FCLDFLAGS="-lf90s" ac_cv_sizeof_void_p=8 CFLAGS="-Os -noparallel" FCFLAGS="-Oss -noparallel" ax_f77_mangling="lower case, underscore, no extra underscore"
ibm_bgl	./configure CC=blrts_xlc FC=blrts_xlf90 CFLAGS="-O3 -qarch=440d -qtune=440 -qstrict -I/bgl/BlueLight/ppcfloor/bglsys/include" FFFLAGS="-O3 -qarch=440d -qtune=440 -qsuffix=cpp=F -qfixed=72 -w -I/bgl/BlueLight/ppcfloor/bglsys/include" FCFLAGS="-O3 -qarch=440d -qtune=440 -qsuffix=cpp=F90 -w -I/bgl/BlueLight/ppcfloor/bglsys/include" ac_cv_sizeof_void_p=4 cross_compiling=yes --enable-mpi ax_f77_mangling="lower case, no underscore, no extra underscore"
nec_es	./configure CC=esmpic++ FC=esmpif90 AR=esar RANLIB=true ac_cv_sizeof_void_p=8 ax_vector_machine=yes cross_compiling=yes --enable-mpi --enable-omp ax_f77_mangling="lower case, no underscore, extra underscore"
nec_sx9_cross	./configure CC=sxmpic++ FC=sxmpif90 AR=sxar RANLIB=true ac_cv_sizeof_void_p=8 ax_vector_machine=yes cross_compiling=yes ax_f77_mangling="lower case, no underscore, extra underscore" 5

```
LIS_MATRIX          A;
LIS_VECTOR          b,x;
LIS_SOLVER          solver;
int                 iter;
double              times,itimes,ptimes;

lis_initialize(argc, argv);
lis_matrix_create(LIS_COMM_WORLD,&A);
lis_vector_create(LIS_COMM_WORLD,&b);
lis_vector_create(LIS_COMM_WORLD,&x);
lis_solver_create(&solver);
lis_input(A,b,x,argv[1]);
lis_vector_set_all(1.0,b);
lis_solver_set_optionC(solver);
lis_solve(A,b,x,solver);
lis_solver_get_iters(solver,&iter);
lis_solver_get_times(solver,&times, &itimes,&ptimes);
printf("iter = %d time = %e (p=%e i=%e)¥n",iter,times, ptimes, itimes);
lis_finalize();
```

Fig. 1 Example of the C program using Lis.

ratio of 99.1% and the parallelization ratio of 99.99%. We show a comparison of the MPI version of Lis and PETSc in Fig. 2, for solving a three-dimensional Poisson equationon an SGI Altix 3700 with 32 processors, processors, which suggests the practicality of our library.

In recent years, multilevel algorithms for large-scale linear equations, such as the algebraic multigrid (AMG), have been investigated by many researchers. In most cases, multigrid methods show linear scalability, and the number of iteration counts is O(n) for a problem of size n. The algebraic multigrid method is based on a principle similar to the geometric multigrid, which utilizes the spatial information on physical problems, but this method differs from the geometric multigrid by considering the coefficient as a vertex-edge incidence matrix. In addition, by using the information on the elements and their relations, this method generates coarser level matrices without higher frequency errors. The complexity of the algebraic multigrid is equivalent to the geometric multigrid and can be applied to irregular or anisotropic problems. We proposed an efficient parallel implementation of the algebraic multigrid preconditioned conjugate gradient method based on the smoothed aggregation (SA-AMGCG) and found that the proposed implementation provides the best performance as the problem size becomes larger [38]. Currently, the algebraic multigrid is the most effective algorithm for the general-purpose preconditioning, and its scalability is also remarkable. We have implemented the algebraic multigrid in Lis and have tested the algebraic multigrid in massively parallel environments. We presented weak scaling results for a twodimensional Poisson equation of dimension 49 million on 1,024 nodes of a Blue Gene system in Fig. 3.

The convergence of the Krylov subspace methods are much influenced by the rouding errors. Higher precision operations are effective for the improvement of convergence, however the arithmetic operations are costly. We implemented the quadruple precision operations using the double-double precision for both the systems of linear equations and the eigenvalue problems, and accelerated them by using Intel's SSE2 SIMD instructions. To improve the performance, we also applied techniques such as loop unrolling. The computation time of our implementation is only 3.5 times as much as Lis' double precision, and five times faster than Intel Fortan's REAL*16. Furthermore, we proposed the DQ-SWITCH algorithm, which efficiently switches the double precision iterations to the quadruple precision to reduce

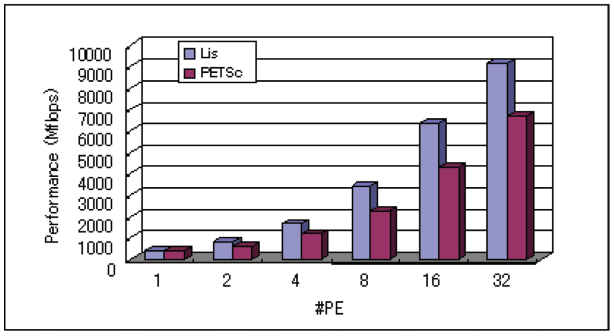

Fig. 2 Comparison of the MPI version of Lis and PETSc. Matrix size is 1,000,000 and number of nonzero entries is 26,207,180.

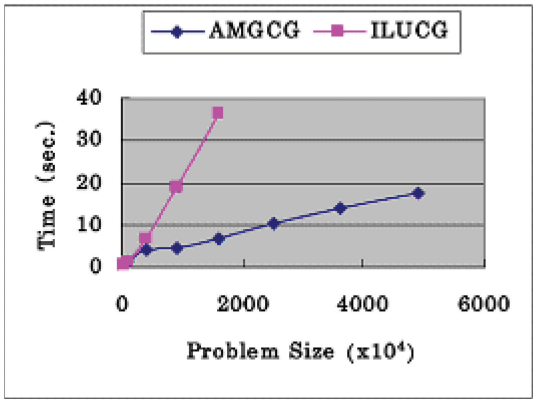

Fig. 3 Comparison of AMGCG and ILUCG.

the computation time. The idea of the SIMD accelerated double-double precision operations was incorporated into Japan's next generation 10 petaflops supercomputer project by RIKEN.

In structural analysis and materials science such as solid-state physics and quantum chemistry, efficient algorithms for large-scale eigenproblems for large-scale simulations are indispensable. There are several methods to compute eigenvalues of large-scale sparse matrices. We implemented major algorithms based on the Krylov subspace, from the viewpoint of scalability in parallel environments. The eigenproblems can be solved combined with appropriate preconditioners, including the algebraic multigrid.

The performance of iterative solvers is affected by the data structure of given matrices, the methodology of their parallelization, and the hierarchy of computer architectures. We have studied the validity of the performance optimization of iterative solvers by benchmarking the MFLOPS performance of matrix vector product kernels on the given computing environment. Figure 4 shows the performance of a kernel spmvtest1, derived from a discretized 1-dimensional Poisson equation, for size from up to 1,280,000 on a single node of SX-9 at JAMSTEC, and Fig. 5-7 show the performance for size up to 40,960,000 on three scalar clusters with DDR Infiniband interconnect at Kyushu University. While the scalar architecture based machines show performance degradation after they reach their peak performance with the data size of 500kB to 2MB per core, vector architecture shows gradual performance increase until it reaches about 8-9GFLOPS per core (with the diagonal (DIA) format in this case), and keep it as the data size grows. It suggests that we should use as many cores with large caches as possible when using a scalar architecture for such problems.

To date, we have counted more than three thousand projects around the world. It is just the first step for us to achieve more flexibility in scalable scientific computing, but we hope our efforts reduce some barriers towards upcoming exascale scientific computing environments in the near future.

References

[1] A. Nishida, "SSI: Overview of simulation software infrastructure for large scale scientific applications (in Japanese)", IPSJ, Tech. Rep. 2004-HPC-098, 2004.

[2] A. Nishida, "Experience in Developing an Open Source Scalable Software Infrastructure in Japan", Lecture Notes in Computer Science, vol. 6017, pp. 87-98, 2010.

[3] A. Nishida, R. Suda, H. Hasegawa, K. Nakajima, D. Takahashi, H. Kotakemori, T. Kajiyama, A. Nukada, A. Fujii, Y. Hourai, S. L. Zhang, K. Abe, S. Itoh, and T. Sogabe, The Scalable Software Infrastructure for Scientific Computing Project, Kyushu University, 2009, http://www.ssisc.org/.

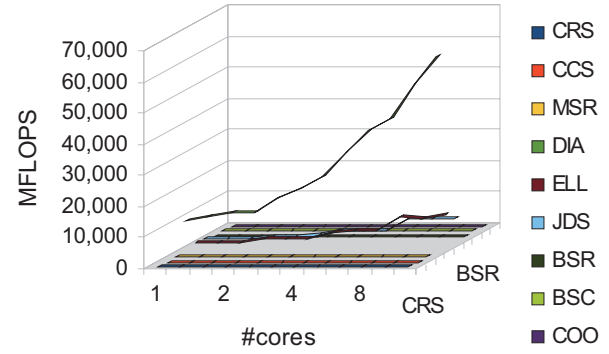

Fig. 4 Performance of spmvtest1 for size from 40,000 to 1,280,000 on a single node of the Earth Simulator 2.

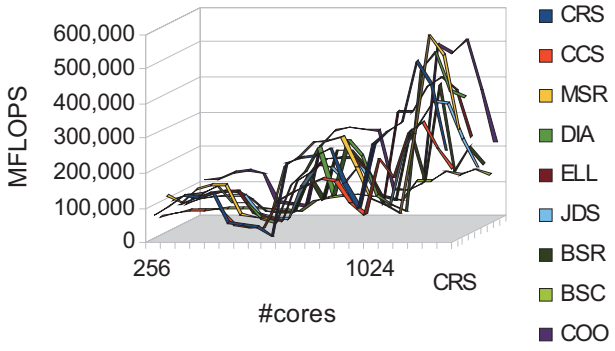

Fig. 5 Performance of spmvtest1 for size from 320,000 to 40,960,000 on the Fujitsu PRIMEGY RX200S3 Cluster at Kyushu University.

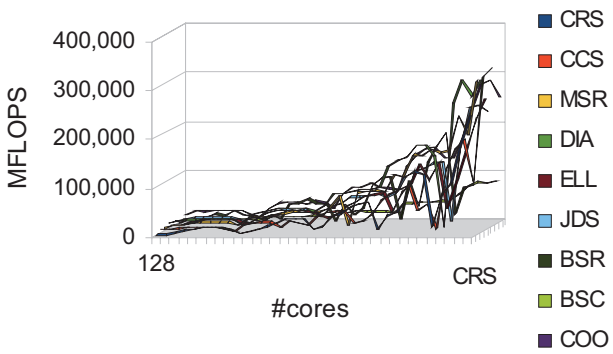

Fig. 6 Performance of spmvtest1 for size from 160,000 to 40,960,000 on the Fujitsu PRIMEQUEST Cluster at Kyushu University.

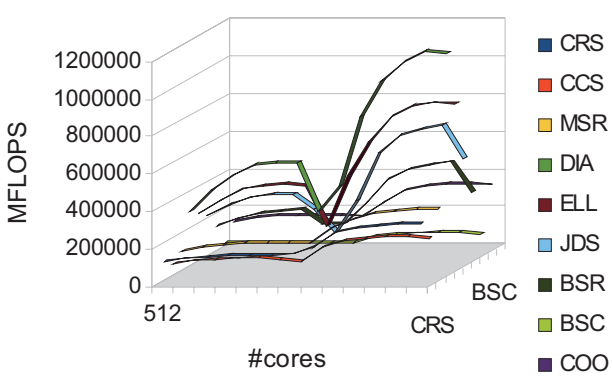

Fig. 7 Performance of spmvtest1 for size from 1,280,000 to 40,960,000 on the Hitachi SR16000 Cluster at Kyushu University.

大規模科学計算向け汎用数値ソフトウェア基盤の開発

プロジェクト責任者
西田　晃　九州大学　情報基盤研究開発センター
著者
西田　晃　九州大学　情報基盤研究開発センター

本プロジェクトでは、従来それぞれの分野において別個に進められてきた並列アルゴリズムや実装に関する知見をもとに、大規模化が予想される今後の計算環境に対応したスケーラブルなソフトウェア基盤を整備することを目的として、反復解法、高速関数変換、及びその効果的な計算機上への実装手法を中心に、平成 14 年度より科学技術振興機構戦略的創造研究推進事業の一環として、多様な計算機環境を想定した開発を行っている。モジュール化されたインタフェースを採用し、複雑な機能を持つライブラリを容易に構築できるようにするとともに、スケーラビリティの観点から並列化に適したアルゴリズムを開発、実装し、高並列な環境での使用に耐えるライブラリを実現している。本研究の成果はネットワークを通じて広く一般に配布し、フィードバックをもとにより汎用性の高いソフトウェアとしていく方針を採っており、平成 17 年 9 月よりソースコードを無償公開するとともに、ユーザの要望を反映した更新を適宜行なっている。平成 18 年度からは、地球シミュレータセンター共同プロジェクトの一環として、高並列なベクトル計算機環境への最適化を実施し、その成果をライブラリとして公開し、多くのユーザに利用されている。本年度は小規模利用環境への移植を中心に行うとともに、4 倍精度演算を用いた固有値解法ライブラリを実装し、その有効性を実証した。

キーワード:ハイパフォーマンスコンピューティング, 並列アルゴリズム, モジュラープログラミング