Available online at www.sciencedirect.com

SCIENCE DIRECT®

Ocean Modelling 10 (2005) 185–191

Ocean Modelling

www.elsevier.com/locate/ocemod

FoSSI: the family of simplified solver interfaces for the rapid development of parallel numerical atmosphere and ocean models

Stephan Frickenhaus [a,b,*], Wolfgang Hiller [b], Meike Best [b,1]

[a] *Competence Center of High Performance Computing, BremHLR, Bremen, Germany*
[b] *Alfred-Wegener-Institute for Polar and Marine Research, Computing Center, Columbusstrasse 27568, Bremerhaven, Germany*

Received 28 November 2003; received in revised form 1 June 2004; accepted 14 June 2004
Available online 2 November 2004

Abstract

The portable software FoSSI is introduced that—in combination with additional free solver software packages—allows for an efficient and scalable parallel solution of large sparse linear equations systems arising in finite element model codes. FoSSI is intended to support rapid model code development, completely hiding the complexity of the underlying solver packages. In particular, the model developer need not be an expert in parallelization and is yet free to switch between different solver packages by simple modifications of the interface call.

FoSSI offers an efficient and easy, yet flexible interface to several parallel solvers, most of them available on the web, such as PETSC, AZTEC, MUMPS, PILUT and HYPRE. FoSSI makes use of the concept of handles for vectors, matrices, preconditioners and solvers, that is frequently used in solver libraries. Hence, FoSSI allows for a flexible treatment of several linear equations systems and associated preconditioners at the same time, even in parallel on separate MPI-communicators. The second special feature in FoSSI is the task specifier, being a combination of keywords, each configuring a certain phase in the solver setup. This enables the user to control a solver over one unique subroutine. Furthermore, FoSSI has rather similar features for all solvers, making a fast solver intercomparison or exchange an easy task. FoSSI is a community

* Corresponding author. Address: Alfred-Wegener-Institute for Polar and Marine Research, Computing Center, Columbusstrasse 27568, Bremerhaven, Germany. Tel.: +49 471 4831 1179; fax: +49 471 4831 1590.
E-mail address: sfrickenhaus@awi-bremerhaven.de (S. Frickenhaus).
1 Funded by the German Ministry for Education and Research (DEKLIM-PLASMA).

1463-5003/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.ocemod.2004.06.004

software, proven in an adaptive 2D-atmosphere model and a 3D-primitive equation ocean model, both formulated in finite elements.

The present paper discusses perspectives of an OpenMP-implementation of parallel iterative solvers based on domain decomposition methods. This approach to OpenMP solvers is rather attractive, as the code for domain-local operations of factorization, preconditioning and matrix–vector product can be readily taken from a sequential implementation that is also suitable to be used in an MPI-variant. Code development in this direction is in an advanced state under the name ScOPES: the Scalable Open Parallel sparse linear Equations Solver.

Keywords: Sparse parallel solvers; Iterative solvers; Finite element; User interface

1. Introduction

Modularized software is the basis for efficient and portable code development and is a prerequisite for rapid software development of co-operating working-groups for example in the field of earth system modeling. The resulting inter-operability of software components nowadays plays an important role also in scientific computing. This is due to the fact that inter-disciplinary research has become a major paradigm. However, software infrastructure for efficient co-operation are frequently missing. Furthermore, creation of long-term re-usable and thus portable software requires knowledge about the life-time and capabilities of the used software components. In the field of scientific research another serious problem may arise. During research successful algorithms turn to be no longer applicable with growing problem complexity. As an example one may consider the development of a finite-element-model that in the initial phase uses linear elements and later needs higher-order elements, e.g., for stability reasons. In such a case it may turn out that the initially used solver libraries cannot be used in the final code version. It may then be not feasible to switch to another solver within the remaining project time. The solution to this problem is to use an almost omnipotent solver that integrates almost all solvers present on the web. Such a solver package can be found for example within the SLES module of PETSC (Balay et al., 2002), the Portable Extendible Toolkit for Scientific Computing. However, it still remains an open problem how developers of the model code may test various combinations of solvers and preconditioners offered by the underlying solver library within a reasonable time and without deep knowlegde of the spectrum of applicable methods. Thus there is a strong demand for a simple-to-use programming-interface for the end user that is programmed and maintained by an expert, who is able to rapidly extend the interface on user-demands. FoSSI, the Family of Simplified Solver Interfaces, is such a collection of user-interfaces to different parallel sparse linear solvers, grown up on user demands. The interfacing is greatly simplified for linear systems with matrix-representations in compressed sparse row (CSR) format. The supported MPI-parallel (Gropp et al., 1994) sparse iterative solvers (Saad, 1996; van der Vorst, 2003) are PETSC, PILUT (Karypis and Kumar, 1997), AZTEC (Tuminaro et al., 1999), HYPRE (Falgout and Yang, 2002), and the parallel direct solver MUMPS (Amestoy et al., 2000). The interfaces are able to keep several matrices and preconditioners within each solver library, allowing to reuse distributed matrix values and factorizations for consecutive solves. Each solver is equipped with a single subroutine for solver configuration and invocation. Different tasks within the interface, e.g., distribution of matrix values, incomplete factorization, solution and cleanup, can be combined within a single task-identifier. The parallelism is completely hidden behind the specification of an MPI-communicator, allowing to use various solvers at the same runtime, or, alternatively, allowing parallel solution of many different linear equation systems synchronously. FoSSI can be downloaded from the web under

http://www.awi-bremerhaven.de/InfoCenter/IT/WorkingGroups/SciComp/FoSSI.html. Version 1.2 is tested on the following list of platforms (Type/OS/MPI/Compilers):

SGI/Irix6.5/MPT/MIPS, IBM-Power4/AIX5.1L/POE/XLF,XLC, IA32-PC/RedHat 9/LAM-MPI/Intel, SUNFire/Solaris9/ClusterTools/Forte-7.

2. FoSSI—overview

FoSSI emerged from a user interface developed for the PILUT-solver (Karypis and Kumar, 1997) within the FENA project, which is the precursor of the Finite Element Ocean Model FEOM (Danilov et al., 2004). It has been found that the usage of such a powerful user interface makes code development very efficient. Nevertheless, PILUT in its MPI-2 variant turned out not to be a good choice on some compute platforms , e.g., the IBM-Regatta. Therefore, and for convenience (user acceptance), the interface principles have been overtaken for other solver libraries, such as PETSC, AZTEC and HYPRE. All three of these offer the user a wide area of accessible data structures and configuration options. Especially the library internal storage of distributed matrices, vectors and preconditioners and their reference by handles can be exploited within the user interfaces by attributing to each parallel linear equations system a FoSSI-handle, which is simply a user selected number out of a limited range. The second principle of the mentioned libraries is to separate the setup of matrices, vectors, preconditioners, the configuration of the solver, the invocation of the solver and the deallocation of solver internal data structures. This feature has been made transparent to the user within the FoSSI-interfaces by means of the task specifier concept. A task specifier is a sum of elementary task specifiers, each being a configuration or action option for the parallel solver. For example, in the PETSC-interface a parallel matrix structure is generated, filled with symmetrically scaled values by specifying the task `PET_STRUCT + PET_MVALS + PET_SYM_SCAL`. A more detailed preconditioner configuration is given by the task specifier `PET_PCASM + PET_ASMB + PET_ICC + PET_OVL_2`, which sets up an additive Schwarz-preconditioner with overlap 2 in a symmetric implementation (known in PETSC as `PETSC_ASM_BASIC`), and an incomplete Cholesky factorization on the subdomains. This setup is specified together with the solve specifier, e.g., `PET_SOLVE + PET_CG` for a conjugate gradient iterative solver.

For a good solver selection and tuning, the user needs detailed performance (timing) information. At least, the number of applied iterations and the times for factorization and solution should be given back to the user or be printed on output. FoSSI gives some more information back to the user, such as the elapsed time for setup of a parallel matrix, for gathering the solution, for the full call, or even the cumulated time for all calls. The specifier `PET_REPORT` orders performance information in text form before the code returns from the FoSSI-interface routine. The main limitation of the current state of FoSSI is its required input format of matrices and vectors. FoSSI uses global matrix and vector data. However, the interface needs only values on rows that are actually used by the MPI-task the row is attributed to (according to the user-given partitioning). In this way, the user does not have to take care about distributed data formats. This limitation will be overcome in the implementation of a self-contained solver library equipped with a FoSSI similar user interface. This solver will be independent of other solver packages (see last section).

3. FoSSI used in atmosphere and ocean modeling

FoSSI is currently used in two projects: first, within the Finite Element Ocean Model FEOM (Danilov et al., 2004), which is a 3D primitive equation ocean model on a static mesh of columns of tetraeders, and second, within the PLASMA model (Läuter et al., 2003), which is a 2D-adaptive atmosphere model based on the shallow water equations on the sphere, formulated in finite elements with a semi-Lagrangian discretization of time. Both codes currently work with upto approxiomately 0.5 million degrees of freedom. The setup times for the matrix and vector distribution through FoSSI is negligible within both models (below 0.5 s for four CPUs, perfectly scaling; data not shown). It has been found within the adaptive code, that the overhead to re-distribute data based on new global matrix structures due to re-meshing is also negligible. Detailed performance data is not given in the present paper as the underlying numerical cores of the model codes are described in other papers. Accordingly, we do not report on performance measurements of the employed solver packages. However, it is worth mentioning that the PETSC-solver has demonstrated a superior scalability and flexibility with respect to the spectrum of preconditioners. The HYPRE algebraic multigrid preconditioners used for the solution of 3D tracer advection equation in the FEOM code showed comparable performance.

4. An OpenMP implementation performance outlook

For some model codes, based on OpenMP-parallelization (see Chandra et al., 2000), a solver based on the same parallelization technique is needed, as the coupling of a multi-threaded (OpenMP) code to an MPI-solver, waiting in the background, may become rather costful. Although it is possible to communicate data between threads and MPI-tasks in a safe way (Rakowsky et al., 2002), an optimal scheduling of the threads and/or MPI-tasks may require each of them to run on a separate processor. In particular, a "spinning" MPI-Task is awakening faster, but does not allow for a fast context switch to a corresponding OpenMP-thread running on the same processor. The consequence is, that on some compute platforms, for a scalable code the number of required processors is the number of MPI-tasks in the solver plus the number of threads in the rest of the code. Such a processor setup obviously is not easily load-balanced, as the processors running the multi-threaded code have to wait for the solution of the linear equations system, thus wasting compute time.

Generally, an OpenMP (multi-threaded) parallel iterative solver can be programmed on the basis of a sequential (single CPU) code by distributing parallel work within OpenMP parallel do-loops. Here, the problem of memory affinity arises: for larger SMP computers memory is distributed over several CPU-boards of the system, being remotely accessible at an increased latency time, typically an extra 200–300 ns per cache line. Thus, requests for data from a remote board generate overhead in terms of idle CPU time. On some systems this can be overcome by a so called “first touch memory allocation policy”, meaning, that physical memory addresses are generated at the first time memory is accessed (e.g., initiallized), rather than when it is allocated. This allows the operating system to place pages of memory near the accessing CPU, e.g., on the same system board. As memory access on cache-based systems is organized in the cache hierarchy by reference to pages, the memory of a data array may be stored physically on more than one system board. Obviously, to reach performance, the programmer must organize memory accesses in his code in such a way that memory affinity is preserved throughout the runtime of the program code. This means, that parallel memory access patterns must be repeated, i.e., access to an array of data must be parallelized in the same way throughout the code, such that each thread deals most of the time with processor near, i.e., affine memory.

Iterative sparse solvers are based on three time intensive operations: preparation of a preconditioner (in most cases a variant of incomplete factorization), the matrix–vector product and the preconditioning step (application of a preconditioner to approximately solve a sub-problem within the iterative algorithm). The matrix–vector product can be easily OpenMP-parallelized through a parallel outer (row-) loop. If the matrix is ordered with near diagonal structure, it makes sense to store vectors and matrix values in a processor affine memory, such that only a minor fraction of matrix–vector values is accessed from remote memory. In contrast, the more robust preconditioners based on incomplete factorizations of higher level, have an immanent sequential part in their preparation as well as in their application. One standard way to overcome this problem is to treat the problem with a domain decomposition technique (see Chan and Mathew, 1994), separating work into independent subdomains, each treated by its own processor. Within OpenMP, regarding the memory affinity, this is naturally done by introducing a thread-index into shared (global) data structures. Technically spoken, a static global data array is always shared between the threads, and its pages physical locations may be gouverned by memory affinity. To preserve memory affinity in a transparent way, each subdomain may be given a leading index to the global arrays used for communication, e.g., a buffer vector `Buff[i]` becomes `Buff[t][i]`, where `t` is the logical thread number (obtained from `omp_get_thread_num()`). Within this approach, it is straight forward to code the basis of OpenMP-parallel solvers from MPI-parallel solvers. The MPI-send and receive operations can simply be replaced by memory copy operations between buffers. Furthermore, the code for factorization and preconditioner application can be taken over from an MPI-implementation without changes. In the next version of FoSSI, an efficient and robust parallel solver will be provided to the community that is independent of the parallelization paradigm, namely, the user may chose the MPI- or the OpenMP-version with the same set of features and rather similar performance characteristics.

To demonstrate the efficiency of the approach discussed above, performance measurements are presented in Table 1, allowing to compare the overhead of communication of the MPI- and the OpenMP version of code. It is seen that the vector update (also known as vector gather) in the domain decomposition method is more efficient in OpenMP than in MPI (columns 2 vs. 3). This is due to the overhead of the MPI-implementation over the direct memory-to-memory-copy in the OpenMP-code, although the MPI communication is forced to work over shared memory. In the matrix–vector product, the MPI version of code shows more overhead than expected from the pure update timings. Obviously, the matrix–vector product of the domain-decomposed code

Table 1
Communication times in seconds for 1000 vector updates/matrix–vector products on IBM-p690 (1.3 GHz); the matrix is from the 2D-Poisson problem

#CPUs	MPI-update	OMP-update	MPI-mvu	OMP-mvu	mv omp-for
2	0.06	0.06	23.9	20.5	20.6
4	0.09	0.05	12.47	10.4	10.4
8	0.11	0.03	6.26	6.1	5.7
16	0.13	0.06	3.51	3.5	3.3
30	0.15	0.2	2.32	3.1	4.1

mvu means domain decomposed matrix–vector product with vector update. mv omp-for is the row-loop-parallel OpenMP version of the global sparse matrix–vector product.

performs similarly fast as the parallel loop code. On a p690, four 8-CPU subsystems (Multi-Chip-Modules) are coupled by a communication bus, thus the communication overhead is expected to increase when more than eight CPUs are used. It is noteworthy that we did not analyse how processes and threads were distributed over the 32 CPUs. To achieve a reliable memory affinity it is necessary to bind processes/threads to CPUs.

5. Conclusion

We introduced a convenient way to access various parallel solvers for finite element modeling codes of geophysical flow, even for the unexperienced user. The Family of Simplified Solver Interfaces provides a flexible and highly performing parallel tool. Further development in the field of robust iterative solvers parallelized in OpenMP and MPI are in progress and necessary, in particular, for real domain decomposed model codes, i.e., where global matrices and vectors are not available. The emerging solvers will also take into consideration system specific optimization options such as processor binding for memory affinity and higher optimization techniques for vector architectures. In contrast to the existing solvers with FoSSI-interfaces, these new solvers will allow for a matrix and vector distribution in such a way that the user may program the code around the solver in a completely domain decomposed way with only processor local data structures. This is achieved by handing over the communication routine for vector updates to the user. It is planned to make these solvers accessible to the scientific community for non-commercial use in source form as a portable maintained stand-alone software package under the acronym ScOPES.

As presented in this paper, FoSSI may serve as a starting point to rapid parallelization of model codes strongly preserving the performance characteristics of the underlying solver packages. Furthermore, FoSSI can be read as an exemplary code for modularization techniques that make code development much easier. The reader is strongly encouraged to make free use of the concepts and the know-how gathered within the freely available source code of FoSSI.

Acknowledgement

The development of FoSSI in the context of PLASMA is funded by the German Ministry for Education and Research within the German climate research programme DEKLIM. The development of ScOPES is partly funded within the Competence Center for High Performance Computing Bremen (BremHLR) by the Ministry for Education and Science Bremen. The authors are very grateful for the contribution of test problems by the FEOM-developers Genady Kivman and Sergej Danilov as well as by the PLASMA co-developers Natalja Rakowsky and Matthias Laüter.

References

Amestoy, P.R., Duff, I.S., L'Excellent, J.-Y., 2000. Multifrontal parallel distributed symmetric and unsymmetric solvers. Comput. Methods Appl. Mech. Eng. 184, 501–520, Available from <http://www.enseeiht.fr/apo/MUMPS/>.

Balay, S., Gropp, W., Curfman-Mclnnes, L., Smith, B., 2002. Petsc users manual. Technical Report ANL-95/11. Revision 2.1.3. Available from <http://www-fp.mcs.anl.gov/petsc/>.

Chandra, R., Dagum, L., Kohr, D., 2000. Parallel Programming in OpenMP. Morgan Kaufmann Publishers.

Chan, T.F., Mathew, T.P., 1994. Domain decomposition algorithms. Acta Numer., 61–143.

Danilov, S., Kivman, G., Schröter, J., 2004. A finite element ocean model: principles and evaluation. Ocean Modell. 6, 125–150.

Falgout, R.D., Yang, U.M., 2002. HYPRE: a library of high performance pre-conditioners, Part III. Comput. Sci. 2002.

Gropp, W., Lusk, E., Skjellum, A., 1994. Using MPI. MIT Press, Cambrige.

Karypis, G., Kumar, V., 1997. Parallel threshold-based ILU Factorization. In: Proceedings of 9th Supercomputing Conference, San Diego. ACMSIGARCH, pp. 1–24. Available from <http://www.supercomp.org/sc97/proceedings/>.

Läuter, M., Handorf, D., Dethloff, K., Frickenhaus, S., Rakowsky, N., Hiller,W., 2003. An adaptive Lagrange–Galerkin shallow-water model on the sphere. In: Heinze, T., Lanser, D., Layton, A.T. (Eds.), Proceedings of the Workshop on Current Development in Shallow Water Models on the Sphere, 10–14 March 2003. Munich University of Technology, Munich, Germany.

Rakowsky, N., Frickenhaus, S., Hiller, W., Läuter, M., Handorf, D., Dethloff, K., 2002. A self-adaptive finite element model of the atmosphere. In: Zwieff-hofer, W., Kreitz, N. (Eds.), Proceedings of the 10th ECMWF Workshop on the Use of High Performance Computing in Meteorology: Realizing Ter-aComputing. World Scientific.

Saad, Y., 1996. Iterative Methods for Sparse Linear Systems. PWS Publishing Company.

Tuminaro, R.S., Heroux, M., Hutchinson, S.A., Shadid, J.N., 1999. Official aztec users guide: version 2.1. Available from <http://www.cs.sandia.gov/CRF/pspapers/Aztec_ug_2.1.ps>.

van der Vorst, H., 2003. Iterative Methods for Large Linear Systems. Cambridge University Press.