Mixed Mode Programming for Sparse Linear Algebra

Alfredo Buttari[1] * and Salvatore Filippone[1] **

TR-2003-10 June 2003
Università di Roma – Tor Vergata
Dept. of Computer Science

1 Introduction

Sparse linear algebra algorithms, and especially algorithms for the iterative solution of sparse lineary systems, lie at the heart of many scientific computing applications, ranging from computational fluid dynamics to structural engineering and electromagnetic analysis. Their efficient implementation is thus a very important challenge for the numerical software community.

The current trend in high performance computing architectures is to move towards clusters of shared memory multiprocessor machines; the advances in the hardware and system performance however have yet to be matched by corresponding advances in programming paradigms.

In this paper we report on our experiments in implementing a hybrid programming model for sparse linear algebra computations. We have taken an existing library interface based on MPI [3] and reimplemented its kernels by using OpenMP parallelization for intra-node computations. We comment on the major points related to the parallelization of the various algorithms, and on the viability of the various operating environments, with respect to the computing, communication and compilation systems.

Our aim in this work is to provide an interface for the convenient implementation of iterative methods for sparse linear systems on clusters of shared memory computers; such computing platforms include most currently available supercomputers such as the IBM SP machines, as well as networks of commodity workstations based on the Intel processor architecture.

The field of sparse linear algebra has seen recently the emergence of a new standard proposed by Duff et al [1], which is a substantial update of the previous effort documented in [2]; our library is based on the same internals, and we are currently working at making the user-level interface compatible with the new standard.

2 General Overview

The implementation of iterative linear system solvers as envisaged in our library, comprises the following main areas:

* alfredo.buttari@uniroma2.it
** salvatore.filippone@uniroma2.it

– Sparse matrix by dense matrix product;
– Sparse triangular systems factorization and solution;
– Vector/dense matrix operations (sums, dot products etc. . .)
– Data exchange and update;
– Data structure preprocessing and initialization.

The hybrid programming model we have implemented entails a top-level data distribution among computational nodes that is identical to the approach described in [3]; in each computational node we then apply the OpenMP parallel directives to obtain the shared memory parallelization. In general we may note the following:

Matrix-vector products: These operations are naturally parallelized with the insertion of `PARALLEL DO` directives at the level of the main loop, which is usually a loop on the matrix rows (details depending on the actual sparse matrix storage format).

Sparse triangular systems: Operations with sparse triangular systems are at the heart of ILU preconditioning and its variants. A parallelization at the level of the single triangualr system solve would have been prohibitively expensive, given the cost of synchronization on current computing platforms. We thus decided to partition the matrix to be factored and to consider only the diagonal blocks, in analogy to what happens at the MPI level: a computational run having two tasks with one thread each or a run with one task with two threads would apply exactly the same preconditioner.

Vector and dense matrix operations: These operations are naturally parallelized at the OpenMP level because the moslty consist of simple loops, possibly with the use of `REDUCTION` clauses. However the efficiency of parallelization is very much dependent on the combination of hardware, operating system, compiler and problem size; we were forced to experiment to find out appropriate thresholds to be included in the code.

Data exchange: Our interpretation of the hybrid paradigm is that we perform message-passing with only one thread per process; indeed thread safety is too dependent on the MPI implementation, and is not currently available on MPICH (arguably the most popular implementation).

Data structure preprocessing: These routines are currently much less advanced, and will need more work to be implemented in a way that preserves both thread safety and performance.

3 Experimental Results

We tested our approach on the following platforms:

1. Linux workstation cluster, using RedHat Linux 8.0 with kernel 2.4.18, the Portland Group Fortran compiler 4.0, MPICH 1.2.4 on a Fast Ethernet connection,

2. Linux workstation cluster, using RedHaty Linux 8.0 with kernel 2.4.18, the Intel Fortran compiler for Linux 7.1, MPICH 1.2.4 on a Fast Ethernet connection,

We are also getting access to an IBM SP2 with AIX 4.3.3, XLF 7.1 and POE 3.1.0 and to another LInux cluster with Gigabit Ethernet connectivity, but our performance experiments are not complete at this time.

Time with Intel compiler				
NP	$AXPBY$	$NRMI$	$MATVECT$	$PREC$
1Ta*1Th	1.04 e-2	8.90 e-2	0.167	2.15 e-3
2Ta*1Th	6.77 e-3	4.77 e-2	9.66 e-2	1.97 e-3
1Ta*2Th	1.06 e-2	5.69 e-2	0.105	1.35 e-3
2Ta*2Th	7.05 e-3	3.21 e-2	6.54 e-2	6.94 e-4

Table 1. Kernels with $180K$ matrix (3M nonzeros) and vector dimension

Time with Portland compiler				
NP	$AXPBY$	$NRMI$	$MATVECT$	$PREC$
1Ta*1Th	1.52 e-2	0.105	0.197	2.45 e-3
2Ta*1Th	9.25 e-3	5.29 e-2	0.11	1.20 e-3
1Ta*2Th	1.05 e-2	6.73 e-2	0.122	1.87 e-3
2Ta*2Th	8.27 e-3	3.66 e-2	7.54 e-2	9.41 e-4

Table 2. Kernels with $180K$ matrix (3M nonzeros) and vector dimension

Time with Intel compiler				
NP	$AXPBY$	$NRMI$	$MATVECT$	$PREC$
1Ta*1Th	9.13 e-4	1.50 e-2	2.55 e-2	2.59 e-4
2Ta*1Th	1.19 e-3	9.06 e-3	1.56 e-2	1.50 e-4
1Ta*2Th	1.27 e-3	1.16 e-2	1.72 e-2	2.23 e-4
2Ta*2Th	1.14 e-3	6.71 e-3	1.17 e-2	1.16 e-4

Table 3. Kernels with $17K$ matrix (550K nonzeroes) and vector dimension

In the preceding tables we have reported algebraic kernels execution times (tables 3, 1,4,2) obtained with different threads/tasks configurations using two different domain sizes and solver times (table 5) for a $180KX180K$ sparse linear

	Time with Portland compiler			
NP	$AXPBY$	$NRMI$	$MATVECT$	$PREC$
1Ta*1Th	9.82 e-4	1.54 e-2	2.82 e-2	2.38 e-4
2Ta*1Th	1.59 e-3	9.62 e-3	1.69 e-2	1.19 e-4
1Ta*2Th	9.69 e-4	1.05 e-2	1.79 e-2	2.18 e-4
2Ta*2Th	1.7 e-3	7.43 e-3	1.30 e-2	1.14 e-4

Table 4. Kernels with $17K$ matrix (550K nonzeroes) and vector dimension

	Solver Time with Intel compiler		
NP	Tot time	time/itx	N itx
1Ta*1Th	26.7	0.92	29
2Ta*1Th	19.8	0.51	39
1Ta*2Th	25.2	0.64	39
2Ta*2Th	15.0	0.37	40

Table 5. Bi-CGStab solver with $180K$ matrix dimension with 3294221 nnzeroes

system with $3M$ nonzeroes. A somewhat surprising observation is that at leaset in this configuration the parallelization through MPICH is competitive even when the underlying communication layer is Fast Ethernet, which is not exactly a leading edge technology. While it is certainly true that further tuning work is needed to guarantee portable performance across OpenMP implementation for our kernels, we believe that the efficiency of the current thread model in Linux is less than satisfactory for fine grained computations. In this respect the work currently underway in the kernel and scheduled for release 2.6, in conjunction with glibc 2.3, promises to improve the situation by a large amount, provided that the compilers are ported to the new configuration.

4 Conclusions

We have presented our experience in implementing sparse linear algebra computations in a hybrid MPI/OpenMP programming model. The results we got show some promise, even though in many cases the tools are not yet robust enough for the casual/general user.

In the near future we plan to complete our implementation, to experiment with different partitioning/reordering strategies [4] and to apply the hybrid programming paradigm to engineering applications in the field of computational fluid dynamics.

References

1. Iain S. Duff, Michael H. Heroux, and Roldan Pozo. An overview of the sparse basic linear algebra subprograms: The new standard from the BLAS technical forum.

ACM Transactions on Mathematical Software, 28(2):239–267, June 2002.
2. I.S. Duff, M. Marrone, G. Radicati, and C. Vittoli. Level 3 basic linear algebra subprograms for sparse matrices: a user level interface. *ACM Trans. Math. Softw.*, 23(3):379–401, September 1997.
3. Salvatore Filippone and Michele Colajanni. PSBLAS: A library for parallel linear algebra computation on sparse matrices. *ACM Transactions on Mathematical Software*, 26(4):527–550, December 2000.
4. Leonid Oliker, Xiaoye Li, Parry Husbands, and Rupak Biswas. Effects of ordering strategies and programming paradigms on sparse matrix computations. *SIAM Review*, 44(3):373–393, September 2002.