Implementation and performance assessment of a parallel solver for sparse linear systems of equations and rules for optimal solution

T. Grytsenko & A. Peratta
Wessex Institute of Technology, UK

Abstract

Many computational algorithms in science and engineering give rise to large sparse linear systems of equations which need to be solved as efficiently as possible. As the size of the problems of interest increases, it becomes necessary to consider exploiting multiprocessors to solve these systems. This paper reports on the implementation of a parallel solver for sparse linear systems of equations and proposes simple formulas for predicting the speedup in terms of the size of the linear system and number of processors in the cluster. The iterative solvers considered in this paper are i – Conjugate Gradient Squared Method (CGS), ii – Generalised Minimal Residual Method (GMRES) and iii – the Transpose Free Quasi-Minimal Residual Method (TFQMR) from the Aztec library implemented with the MPI interface and Parallel Knoppix based cluster.

Keywords: parallel solver, iterative solver, sparse matrix, cluster, MPI, Aztec, Parallel Knoppix.

1 Introduction

Many computational algorithms in science and engineering give rise to large sparse linear systems of equations (LSES) which need to be solved as efficiently as possible. Most modern iterative methods for solving sparse LSES have as their key computational step the computation

$$Ax = b \tag{1}$$

WIT Transactions on Information and Communication Technologies, Vol 38, © 2007 WIT Press
www.witpress.com, ISSN 1743-3517 (on-line)
doi:10.2495/DATA070061

where A is a sparse matrix, and x and b are input and output vectors, respectively. For large-scale computing, the calculation of (1) is time consuming when computed without paying particular attention to data structure and computer architecture. However, the elapsed computational time can be greatly reduced on a parallel computer by partitioning the original problem into blocks and then distributing them over the processors and performing the computation in parallel. This paper reports on architecture and implementation of parallel solver for sparse LSES based on Linux cluster and proposes the rules for optimal solution of LSES by means of parallel solver.

The next section gives an overview of the tools employed. In Section 3, architecture of developed software, configuration of cluster and its performance are described. Section 4 provides test results for parallel solver based on this cluster. Section 5 proposes the rules for optimal solution of LSES by means of developed parallel solver. Finally, Section 6 provides the conclusions and further research remarks.

2 Used techniques and tools

This section provides a brief introduction of the four techniques and tools used in developed software: Mpich, Aztec and Parallel Knoppix.

2.1 Mpich library

MPICH is an implementation of the Message Passing Interface (MPI) [4]. The goals of MPICH are to provide an MPI implementation for different platforms such as clusters and parallel processors. MPI was taken as a basis for parallel calculations because of its portability and high performance [3,4]. MPI is widely used in parallel scientific libraries including applied libraries for solving LSES. In this study MPICH v1.2.7 has been used.

2.2 Aztec library

Aztec [5] is a library that provides algorithms for the solution of large sparse linear systems arising in scientific and engineering applications. Aztec includes a number of Krylov iterative methods [6] such as conjugate gradient (CGS), generalized minimum residual (GMRES), biconjugate gradient (BiCG), stabilized biconjugate gradient (BiCGSTAB), transpose-free quasi-minimal residual (TFQMR) to solve LSES. It is portable to most parallel platforms since it uses MPI library to perform data communication across network or between processors in parallel computer. In this study Aztec v2.1 has been used.

2.3 Operative system

As an Operative system (OS) Parallel Knoppix [1,2] has been used. This is a cluster OS where nodes of the cluster can be configured almost automatically. The Parallel Knoppix is pre-configured to support from 2 to 254 nodes, but it can

easily be modified to allow for larger clusters. The last version of Parallel Knoppix gives a possibility to plug new computational nodes on-the-fly so that developed parallel solver becomes extremely scalable. In this study Parallel Knoppix 2.6.16.16 has been used.

3 Software and hardware architecture

3.1 Software architecture

The parallel solver is represented as a 3-layer software (see Figure 1), where both OS and hardware are on the first layer, MPI interface is on the second one and the solver itself including Aztec library represents the last one. As a hardware platform any parallel computer with distributed memory (either cluster or parallel processor) can be used. The proposed software architecture provides good portability and scalability. The parallel solver, written in standard C++, uses only the STL so that it is a platform independent tool. The developed software provides interfaces for third-party utilities such as Metis [7] and Mondriaan [7,8].

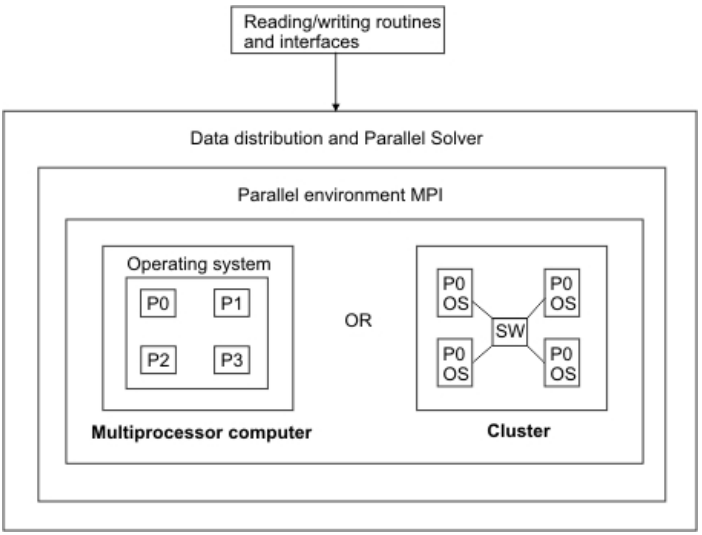

Figure 1: Software tool architecture.

3.2 Data distribution model and assembling the results

Any parallel solver needs data to be partitioned over the processors or nodes in a computational environment. A partitioner should distribute the vectors x and b and the nonzero entries of A (1) so that each block contains almost the same number of entries and is as independent as possible from other blocks so that, when the blocks are distributed, communication between them is low. In this

study two different approaches have been used in order to distribute an initial matrix over the parallel processes in a system. The first one is based on linear partitioning and operates in the following way:

Processor 0 is assigned the first $\left[\frac{N+N_{PR}-1}{N_{PR}}\right]$ rows

Processor 1 is assigned the next $\left[\frac{N+N_{PR}-2}{N_{PR}}\right]$ rows, and so on.

where N is the total number of rows in matrix A (eq. 1) and N_{PR} is the total number of processors in a system. In this data distribution model (see Figure 2) the main Process 0 broadcasts the number of rows in matrix A to all other existing processes, then each process sends an answer signal as a vector update[] which consists of indices of rows in the initial matrix A that have to be distributed to this certain process. Then, Process 0 sends the A matrix entries within the corresponding row limits to each process and clears those elements in its local memory so that at the end of distribution stage each process consists of only its own elements.

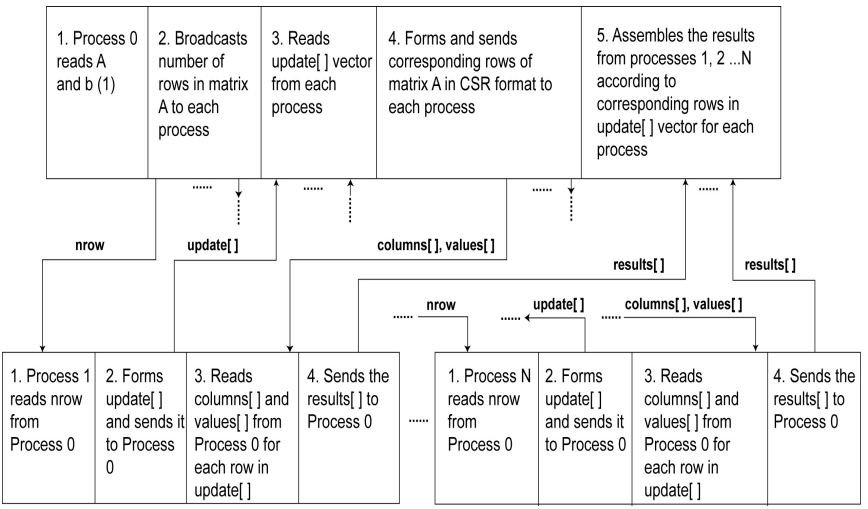

Figure 2: Data distribution and results assembling model for linear partitioning.

If there is a necessity to use an external partitioner rather than the linear one it is possible to employ the second approach. In this approach (see Figure 3) Process 0 broadcasts the number of rows N in matrix A to all other existing processes exactly as in the previous approach. Then, Process 0 generates indices of rows to be updated by each process and sends this information to all existing

processes in the system. Afterwards, Process 0 sends the entries of corresponding rows to each process and clears those elements in its local memory so that at the end of distribution each process consists of only its own elements. Second approach is more centralised and has clearly expressed the main Process 0 which does not only send a content of rows but also generates the rows to be updated by every process. This technique is developed to make an interface to third-party partitioners such as Metis and Mondrian. The present paper does not analyse using of these partitioners.

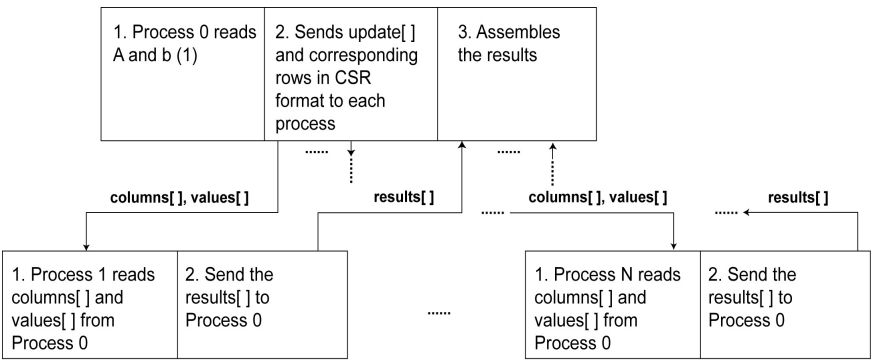

Figure 3: Data distribution and results assembling model for external partitioners.

3.3 Hardware platform description

In this study cluster that consists of two dual-processor computers has been employed. Table 1 shows the corresponding specifications. The computers are connected to each other by 100 Mbits/sec switch.

The computational performance achieved in this cluster is represented in Table 2.

Table 1: Configuration of the cluster.

Component	Model	Quantity
Motherboard	Intel Pentium III FC-PGA Dual ZIF PGA370, Thunder LE S2510	2
Processor	P III 800 Mhz	4
RAM	4 DIMM x 512 Mb	2
Switch 100 Mbit/sec	Netgear Prosafe 16 port, model FS116	1

4 Numerical results

In order to test the developed parallel solving core a series of benchmarks has been prepared. All of the tests in a benchmark have the same sparsity ratio (2) but different number of rows/columns.

Table 2: Cluster performance.

Bandwidth, Mbit/sec	Number of processors	Performance, Gflops
100	1	0.66
	2	1.07
	4	1.5

$$SR = \frac{N_{NZ}}{N^2}, \tag{2}$$

where N is the number of rows and columns of the LSES and N_{NZ} is the number of non-zero elements of matrix A.

To estimate a quality of the results as well as performance of developed software and installed hardware platform, the CPU times in function of the number of rows/columns for single, 2 and 4 processors have been measured.

In this paper parallel versions of CGS, GMRES and TFQMR iterative methods have been studied. For this purpose a series of benchmarks has been prepared. The benchmarks have sparsity ratio about 0.001 and consist of five LSES with number of rows 648, 3000, 12288, 24000 and 41472. The test results are shown in Figure 4.

Figure 4 shows that (i) the solution time increases together with the number of processors for all solvers when system is small (N=648). This happens because the solver distributes initial data across the processes and distribution stage takes more time than solution stage. According to Figure 4, (ii) CGS-method is the fastest one for LSES of any dimension. For all three solvers, (iii) communication time increases together with the dimensionality of LSES. Number of processors also affects communication time: (iv) the higher number of processors the higher communication time. Figure 5 summarises the solution speedup and gives a general view on how duration of different calculation stages is distributed with respect to each other. As a result, (v) GMRES is most scalable method but it is, however, the slowest one. (vi) TFQMR is the least scalable method. (vii) CGS is the fastest one and has medium scalability. (viii) Preparation and communication stages take about one third of total time to solve linear system by CGS method, about one sixth by GMRES and one fourth by TFQMR.

5 Analysis of results

The relationship between CPU time T, N_{PR} and N has been established by means of non-linear least squares method with the following trial functions:

$$\ln T = A + BN_{PR}^{\alpha} + C(\ln N)^{\beta},$$

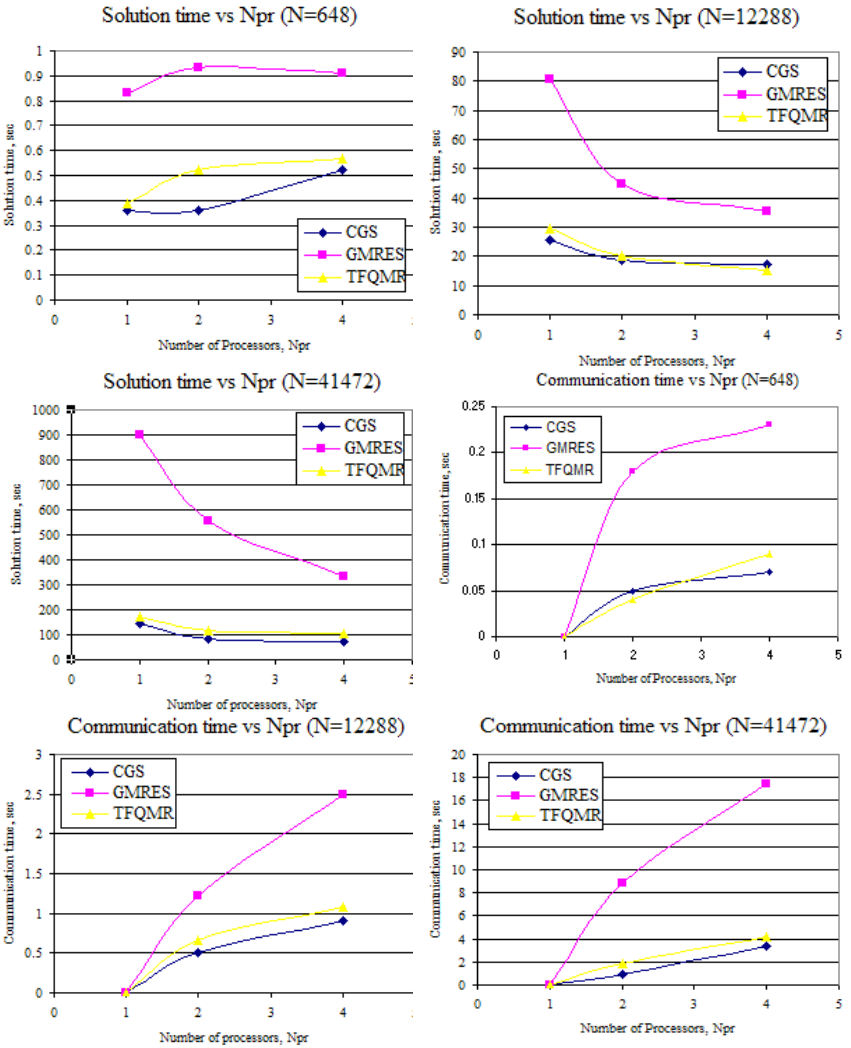

Figure 4: Execution and communication time of CGS-, GMRES- and TFQMR-solvers for LSES with 648, 12288 and 41472 rows depending on number of processors.

where A, B, C, α and β were sought so that to minimise the error between trial function and experimental data. The results for CGS, GMRES and TFQMR solvers are given by equations (3), (4) and (5) respectively (see Figure 6).

$$\ln T = -3.16 + \frac{0.73}{N_{PR}^2} + 0.065 * (\ln N)^2 \qquad (3)$$

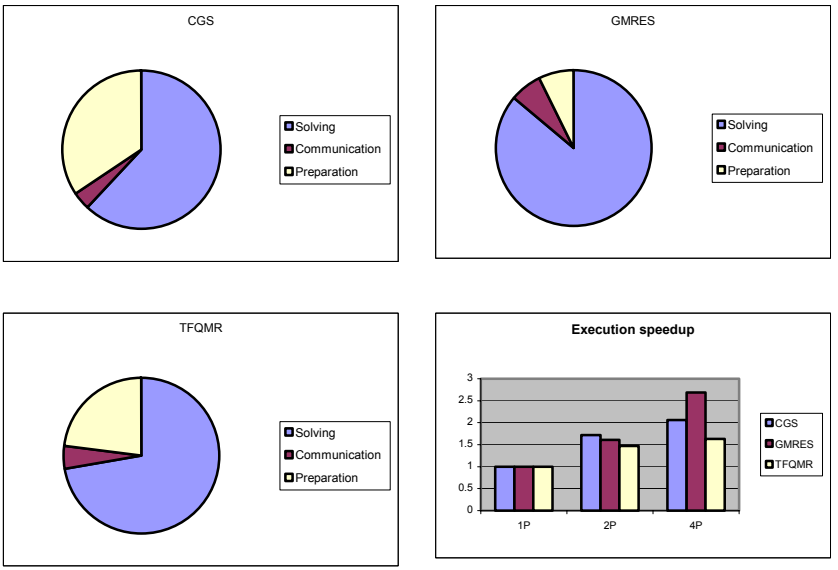

Figure 5: Relative life span of the three main stages and solution speedup for CGS-, GMRES- and TFQMR-method for LSES with 41472 rows on 4 processors.

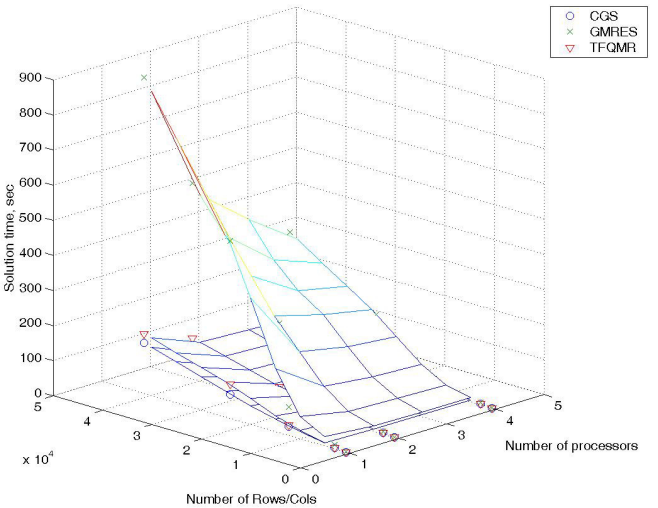

Figure 6: 3D-functions $T(N_{PR}, N)$ for CGS-, GMRES- and TFQMR-method.

$$\ln T = 8.97 - 0.74 * \ln N_{PR} - \frac{438.3}{\sqrt{N}} \quad (4)$$

$$\ln T = 17.46 + \frac{0.595}{N_{PR}^2} - \frac{137.1}{\ln N} \quad (5)$$

Using equations 3–5 it is possible to predict solution time for the clusters with more than 4 processors having the same architecture. The developed rules can be adopted for a control module that decides which one of the solvers has to be applied according to the number of processors available in the cluster and the number of rows/cols in the LSES.

The rules show that if the number of processors is less than 64, then CGS becomes the best option for the solution of LSES with low sparsity ratio (2). However, if the cluster consists of more than 64 processors, then GMRES shows the best solution time 41.6 seconds for LSES with N = 41472.

6 Conclusions

As a result of this work a small cluster for testing purposes has been set up and on its basis a parallel solver for linear systems of equations has been developed. Parallel versions of CGS, GMRES and TFQMR iterative methods for the solution of LSES have been compared to each other. As a conclusion, CGS is the fastest method but offers lowest scalability. TFQMR has medium scalability and performance. GMRES resulted optimum in terms of scalability and, according to equations (3-5), if the number of processors is more than 64 it shows the best performance. However, if the number of processors is less than 64, then CGS is the best option for the solution of LSES with low sparsity ratio (2).

One of the advantages of the developed set of tools is their scalability. The combination of tools proposed does not depend on any specific hardware platform and parallel solver can be used without any modifications on any number of computers up to 254 and any number of processors supported by current kernel of Knoppix OS.

In further work it would be useful to make a comparison and performance analysis of mentioned iterative solvers using network with the bandwidth higher than 100 Mbits/sec and LSES of higher sparsity ratios. Such a research would give an idea of how parallel solver performance depends on network bandwidth and how stable different parallel iterative methods are from the point of view of the LSES of different sparsity ratio.

References

[1] Michael Creel (October 2004). *ParallelKnoppix - Rapid Creation of a Linux Cluster for MPI Parallel Processing Using Non-Dedicated Computers*. http://pareto.uab.es/wp/2004/62504.pdf

[2] Michael Creel (December 2004). *ParallelKnoppix Tutorial*. http://pareto.uab.es/wp/2004/62504.pdf

[3] LAM team (2004). *LAM/MPI Parallel Computing*. http://www.lam-mpi.org/

[4] William Gropp, Ewing Lusk, David Ashton (November 2005). *MPICH2 User's Guide. Version 1.0.3*. Mathematics and Computer Science Division Argonne National Laboratory. http://www-unix.mcs.anl.gov/mpi/mpich/downloads/mpich2-doc-user.pdf

[5] R. S. Tuminaro, M. Heroux, S. A. Hutchinson and J. N. Shadid (December 1999). *Official Aztec User's Guide: Version 2.1*. http://www.cs.sandia.gov/CRF/pspapers/Aztec_ug_2.1.ps

[6] Yousef Saad (1996). Iterative Methods for Sparse Linear Systems. ISBN 0-534-94776-X

[7] S. Riyavong (2003). *Experiments on Sparse Matrix Partitioning*, CERFACS Working Note WN/PA/03/32. http://www.cerfacs.fr/algor/reports/2003/WN_PA_03_32.pdf

[8] Brendan Vastenhouw, Rob H. Bisseling (2005). *A Two-Dimensional Data Distribution Method for Parallel Sparse Matrix-Vector Multiplication*. Siam Review Vol. 47, No. 1, pp. 67–95, Society for Industrial and Applied Mathematics