

# Speedup of Distributed Iterative Solvers of Large Sparse Systems of Linear Equations

W.M. ZUBEREK and T.D.P. PERERA

Department of Computer Science  
Memorial University  
St.John's, Canada A1B 3X5  
wlodek@cs.mun.ca

*Abstract:* Many applications of computational methods in science and engineering require the solution of large sparse systems of linear equations. For such systems, iterative methods are often more attractive than direct methods because of their small (and constant) memory requirements. Moreover, the performance of iterative solvers can easily be improved by using distributed systems. For distributed applications, a common performance characteristic is the speedup, usually defined as the ratio of the execution time of an application on a single processor to the execution time of the same workload on a system composed of  $P$  processors. This paper estimates the speedup of distributed linear iterative solvers, analyzes the influence of different communication schemes on the speedup, and compares the estimates with the measurements of real distributed programs.

*Key-Words:* Distributed computing, speedup, computation-to-communication ratio, sparse systems of linear equations, iterative methods.

## 1 Introduction

Increasingly complex problems encountered in many areas of science and engineering require increasingly powerful high-performance computing systems to be solved. There are two basic ways of increasing the performance of computer systems; one approach increases the performance of a uniprocessor system, while the other improves system's performance by increasing the number of processors. The current computer technology favors multiprocessor systems because they are more economical [7].

Distributed systems are often considered as less expensive and more easily available alternatives to parallel systems [17]. The Beowulf cluster [20] is probably the most popular example of a system composed of (many) standard (or "off-the-shelf") components, connected by a communication medium that exchanges messages among the components of the system. Different forms of communication medium are used in distributed systems, from high-speed specialized interconnects to the

internet.

Due to steadily increasing performance of microprocessors, which, for several decades, has been doubling every 18 months (the so called Moore's Law [12]) and to improving communication bandwidth, distributed computing is becoming an attractive platform for high-performance computing. Indeed, a recent survey of the most powerful supercomputing systems shows that seven out of 10 most powerful systems are clusters [13], and the list of 500 most powerful systems includes 208 clusters. The trend towards cluster computing is expected to continue.

Although the number of practical applications of distributed computing is still somewhat limited [6] and the challenges – in particular, the standardization – are still significant, there are some spectacularly successful examples of large-scale distributed computing, such as SETI@home [22] or ClimatePrediction [21] projects, in which hundreds of thousands and even millions of processors are performing coordinated computations within the same project.

In distributed applications, the total workload is divided among the processors of the system with an expectation of concurrent execution of the distributed tasks. One of the main performance characteristics of a distributed application is its speedup [19], which is usually defined as the ratio of the application's execution time on a single processor,  $T(1)$ , to the execution time of the same workload on a system composed of  $P$  processors,  $T(P)$ :

$$S(P) = \frac{T(1)}{T(P)}.$$

The speedup depends upon a number of factors which include the number of processors and their performances, the connections between the processors, the algorithm used for the distribution of the workload, etc. Some of these factors may be difficult to take into account when estimating the speedup of a distributed application. Therefore, in many cases, a simplified analysis is used to characterize the steady-state behavior of an application. This simplified analysis is based on a number of assumptions, such as a uniform distribution of workload among the processors, constant communication times, and so on.

This paper estimates the speedup of iterative solvers of (large) sparse systems of linear equations. It shows the speedup of distributed solvers as a function of the number of processor used and the "computation-to-communication ratio," that relates the performance of processors to the bandwidth of the communication medium used. The speedup estimates are compared with measurements of implemented distributed linear solvers. Effects of different communication schemes on the performance of speedup of distributed iterative solvers are also discussed.

## 2 Distributed Iterative Solvers

The solution of large, sparse systems of linear equations is an inherent part of many computational method used in scientific and engineering applications [1]. Large systems of sparse linear equations arise in applications of finite element and finite difference approximations, they are used in analysis of electronic circuits, in the steady-state solutions of discrete systems whose behavior is represented by Markov chains, and many other cases. For large

sparse systems of equations, iterative methods [2], [9], [11] are more attractive than direct methods because they are less demanding with respect to memory and can require significantly less computational power. The standard Gaussian elimination applied to a sparse system typically leads to fill-ins, so that the total number of operations required for a solution of a system of  $N$  equations is estimated as  $N^{7/3}$  [18]; for a system of 1000 equations with the density of 0.01 (i.e., with the average of 10 nonzero elements in each equation), the computational requirement of Gaussian elimination is equivalent to an iterative 1000-step solution of the same system of equations. For 10,000 equations (with the same number of nonzero elements per equation), iterative solution with 1000 steps is computationally more than 10 times less demanding than the direct Gaussian elimination. Distributed computing can further reduce the solution time of the iterative approach. Iterative approach can also be used to improve the accuracy of a direct solution of large systems of equations, when cumulative effects of rounding errors can distort the results [8]. Iterative approach can be very attractive for applications in which a fast, approximate solution is needed, that can be obtained in just a few iteration steps. On the other hand, the convergence of the iterative approach can be a troublesome issue, and some additional techniques may be needed to improve the convergence of the iterative process [5].

For distributed processing, the (large number of) equations is divided into approximately equal sections allocated to different processors assuming that all equations have a similar number of nonzero elements and that the processors have similar performance characteristics (if these assumptions are not valid, a different scheme of load distribution is needed). After distributing the system of equations among the processors, the iterative process repeatedly executes the following three consecutive steps:

1. the current approximation to the solution is distributed to all processors,
2. sections of the next approximation to the solution are calculated (concurrently) by the processors,
3. the new approximation is collected from all processors and the convergence is checked.

Although there are several techniques that can be used for iterative solution of (sparse) linear systems [14], the Gauss-Seidel method has been chosen because it is simple to implement and for some applications it converges to the solution regardless of the initial approximation [8]. The discussion and all examples used in this paper are based on the Gauss-Seidel method, however, only minor changes are needed to adjust the presentation to a different iterative method.

The distributed iterative process can be described by the following pseudocode in which, in order to simplify the presentation, the sparsity of the system of linear equations is not taken into account, and in which it is assumed that processor 1 is the “main processor” which initiates the iterative process, and which checks the convergence of the iteration:

```

M := integer(N/P);
for i := 2 to P do
  Send M rows of A and B to processor_i
endfor;
Count := 0;
converged := FALSE;
X := Xinit;
while not converged do
  Xold := X;
  Broadcast X to all processors;
  for i := 1 to M do
    sum := B[i];
    for j := 1 to i - 1 do
      sum := sum - A[i, j] * X[j]
    endfor;
    for j := i + 1 to N do
      sum := sum - A[i, j] * X[j]
    endfor;
    X[i] := sum/A[i, i]
  endfor;
  for i := 2 to P do
    Receive a section of evaluated X
  endfor;
  Count := Count + 1;
  converged := compare(X, Xold)
endwhile;
Broadcast “END” message to all processors;

```

In the above pseudocode,  $N$  is the size of the system of equations,  $P$  is the number of processors,  $M$  is the number of equations assigned to each pro-

cessor,  $\mathbf{A}$  is the coefficient matrix,  $\mathbf{B}$  is the vector of the righthand sides, and  $\mathbf{X}$ ,  $\mathbf{Xold}$  and  $\mathbf{Xinit}$  are the vectors of the current approximations to the solution, previous approximation and the initial approximation, respectively; moreover, a procedure “compare” returns a logical value TRUE or FALSE depending whether or not the convergence criteria are satisfied.

All remaining processors are “controlled” by the main processor:

```

Receive M rows of A, B from processor_1;
flag := 0;
while flag = 0 do
  Receive data from processor_1;
  if data = “END” then
    flag := 1
  else
    for i := 1 to M do
      sum := B[i];
      for j := 1 to i - 1 do
        sum := sum - A[i, j] * X[j]
      endfor;
      for j := i + 1 to N do
        sum := sum - A[i, j] * X[j]
      endfor;
      X[i] := sum/A[i, i]
    endfor;
    Send new values of X to processor_1
  endif
endwhile;

```

The first step typically uses a broadcast (or multicast) operation, and it can be assumed that its execution time,  $T_b$ , does not depend upon the number of processors (in fact, it usually depends on this number, but this dependence is ignored here as it is rather inessential). The last step requires a transfer from each processor to a single processor which performs the convergence check, so the transfers are performed sequentially. It is assumed that the total execution time of this step is equal to  $P * T_c$ , where  $P$  is the number of processors and  $T_c$  is the communication time of a single processor. Although  $T_c$  depends upon  $P$ , the dependence is not very strong [16], and is neglected here.

Let  $T_s$  denote the total (sequential) computation time of one iteration. The (approximate) time of a

distributed execution of a single iteration is then:

$$T(P) = T_b + T_s/P + P * T_c.$$

For simplicity, it can also be assumed that  $T_b = T_c$ , which is an oversimplification, but not very significant, as it appears. With this additional assumption, the speedup is:

$$S(P) = \frac{T_s}{T_s/P + (P + 1) * T_c}.$$

Let  $r_{comp/comm}$  denote the ratio  $T_s/T_c$ , i.e., the ratio of total (sequential) computation time (per iteration) to the communication associated with a single processor (such a ratio makes sense only for programs with cyclic behavior). Then the speedup becomes a function of two variables,  $P$  and  $r_{comp/comm}$ :

$$S(P) = \frac{r_{comp/comm}}{r_{comp/comm}/P + P + 1}.$$

Fig.1 shows the values of  $S(P)$  for  $P = 2, \dots, 50$  and for  $r_{comp/comm} = 10, \dots, 100$ .

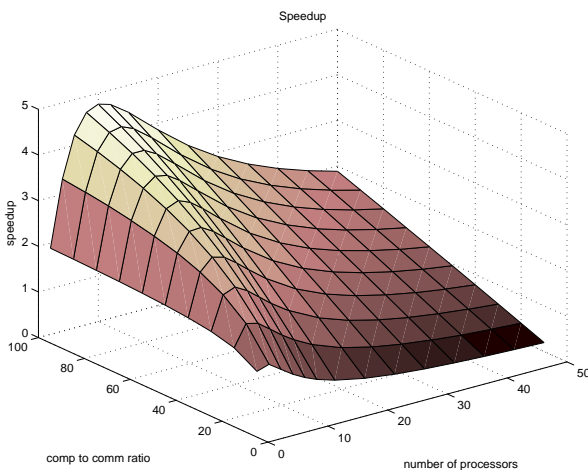


Fig.1. Speedup of distributed iterative solvers.

Reasonable speedups (around 5) can be obtained only when the computation-to-communication ratio is sufficiently high. The best speedup is obtained for a rather small number of processors (5 to 10); for a larger number of processors, the execution time actually increases as it is dominated by the communication time.

It should be observed that the simplifying assumptions are not really important because they affect terms which do not have significant influence on the speedup, especially for large values of  $P$ .

For larger values of  $P$ , the communication time becomes the dominating term in speedup estimation of linear solvers. Therefore the collection of results (step (3)) can be organized in a hierarchical, 2-level manner, in which first the results of computations are collected in groups of, say,  $K$  processors, and then the results of groups are combined together. It can be shown that the number of groups that minimizes the total communication time is equal to  $\sqrt{P}$ , and then the speedup becomes:

$$S(P) = \frac{r_{comp/comm}}{r_{comp/comm}/P + 2 * \sqrt{P} + 1}.$$

Fig.2 shows the values of  $S(P)$  for  $P = 5, \dots, 50$  and for  $r_{comp/comm} = 10, \dots, 100$  for this modified approach.

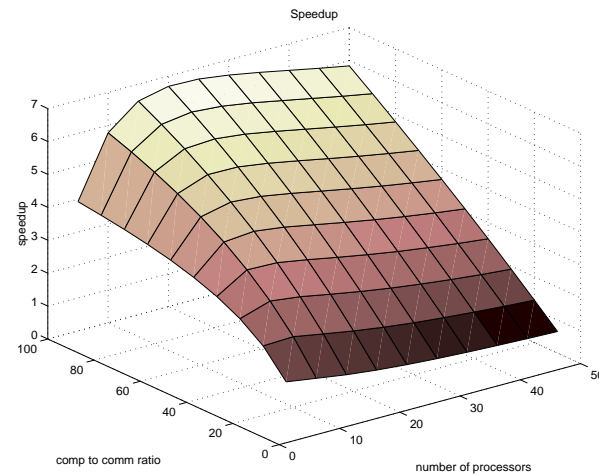


Fig.2. Speedup of modified iterative solvers.

In Fig.2, the relation between the speedup and the number of processors is quite different than in Fig.1; better speedup values can be obtained and the speedup is much less sensitive to the number of processors. For large values of  $P$  (i.e.,  $P$  greater than 100), further improvement can be obtained by additional levels of the hierarchical collection of results.

If the broadcast operation of step (1) is replaced by a sequence of  $P$  unicast send operations, the speedup is not affected in a significant way. This is due to a staggered execution phase in which the consecutive processors start their computation (step (2)), one after another, right after receiving the current approximation to the solution. The return of results is staggered in a similar way. Consequently,



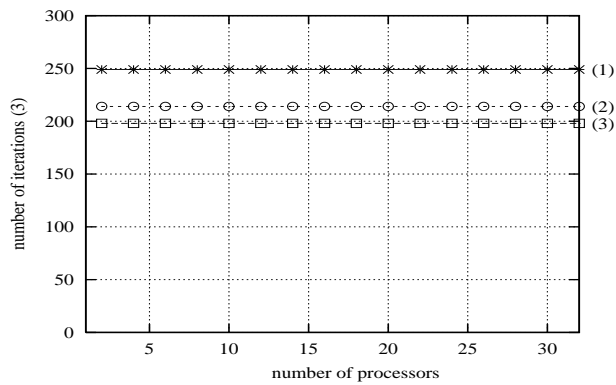


Fig.4. Numbers of iterations of distributed iterative solvers for systems of 500 (1), 1500 (2) and 2500 (3) 5-diagonal equations (sparsity pattern (3)).

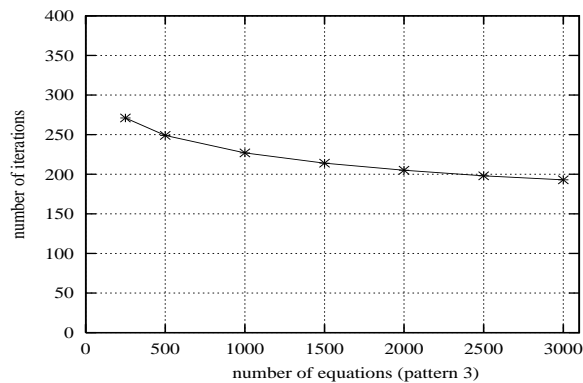


Fig.5. The number of iterations of distributed iterative solvers as a function of the number of equations (sparsity pattern (3)).

In distributed iterative solvers, all processors independently evaluate the sections of the linear systems using the same approximation to the solution. Consequently, the updates of the Gauss-Seidel method are restricted to a single section (which is different for different processors). The convergence properties of distributed solvers can be studied (on a uniprocessor system) by organizing the iterations in exactly the same way as is used in a  $P$ -processor system i.e., each group of equations (that would be assigned to one of processors in a distributed system) evaluates the next approximation to the solution step without the influence of other groups. As before,  $N$  is the size of the linear system of equations,  $P$  is the number of processors,  $\mathbf{A}$  is the coefficient matrix,  $\mathbf{B}$  is the vector of righthand sides,  $\mathbf{Xold}$  and  $\mathbf{Xnew}$  are the “previous” and the “new” versions of the solution vector while  $\mathbf{Xinit}$  is the

initial approximation to the solution:

```

Count := 0;
converged := FALSE;
Xnew := Xinit;
while not converged do
  Xold := Xnew;
  M := integer(N/P);
  i := l := 0;
  for k := 1 to M do
    if k = P then M := N - i endif;
    for m := 1 to M do
      i := i + 1;
      sum := B[i];
      for j := 1 to l do
        sum := sum - A[i, j]*Xold[j]
      endfor;
      for j := l + 1 to i - 1 do
        sum := sum - A[i, j]*Xnew[j]
      endfor;
      for j := i + 1 to N do
        sum := sum - A[i, j]*Xold[j]
      endfor;
      Xnew[i] := sum/A[i, i]
    endfor;
    l := l + M
  endfor;
  Count := Count + 1;
  converged := compare(Xold, Xnew)
endwhile;

```

For example, it can be shown that the influence of the number of processors is more pronounced for small numbers of equations. Fig.6 shows this relationship for a system of 64 equations with the sparsity pattern (2).

The same scheme can be used to study the effects of relaxation, preconditioners, and other techniques used for improving the convergence of the iterative process.

## 5 Concluding Remarks

This paper analyzes, in very general terms, the speedup that can be obtained in distributed iterative solvers of large sparse systems of linear equations. The paper shows that the straightforward

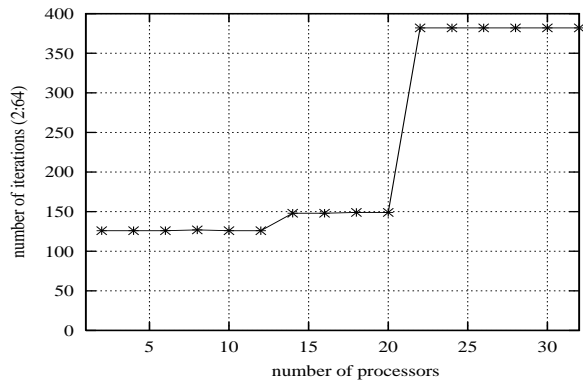
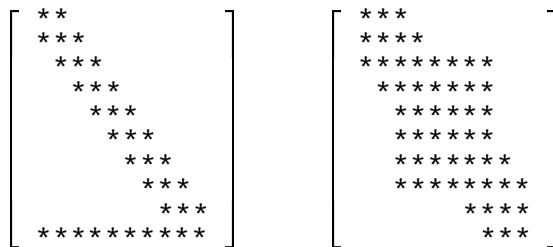


Fig.6. The number of iterations of distributed iterative solvers of 64 3–diagonal equations (sparsity pattern (2)).

distribution of equations among the processors of a distributed system introduces a limitation on the speedup, and that using more processors may actually reduce the speedup and increase the solution time.

The very simple workload distribution (based on the same numbers of equations assigned to different processors) is satisfactory only when the sparsity structure is uniform over the set of equations (as shown in Section 3). For other systems of linear equations, for example, for systems of equations with a “margin” (which is characteristic for the solution of Markov chains) or block-diagonal structures:



a different workload distribution is needed, in which approximately equal numbers of nonzero elements are assigned to processors rather than the same number of equations, otherwise a significant imbalance can occur and reduce the performance of the solver.

Moreover, the iterative method used in this paper is an example of so-called stationary iterative methods, which, in each iteration step, perform the same operations. Nonstationary methods are a relatively recent development; they often are more difficult to

use, but they can be highly effective [?, ?, BB93] so they should also be taken into consideration.

The solution of systems of linear equations is just one example of applications which may benefit from distributed environments. Other applications which are well suited for distributed execution include:

- Complex modeling and simulation techniques that increase the accuracy of results by increasing the number of random trials; the trials can be run concurrently on many processors, and the results combined to achieve greater statistical significance.
- Applications that require exhaustive search through a huge number of results that can be distributed over the many processors, such as drug screening [6].
- Simulations of complex systems (such as VLSI designs) in which the design is partitioned into a number of smaller parts, and these parts are simulated concurrently on different processors [15].

In the future, traditional distributed systems are expected to be used in specialized domains, such as transaction processing for banking applications, in mainstream applications, however, grid computing is emerging as the next evolutionary platform for large-scale high-performance computing [4].

### Acknowledgement

The Natural Sciences and Engineering Research Council of Canada partially supported this research through grant RGPIN-8222.

### References

[1] G.W. Althaus, E. Spedicato, *Algorithms for large scale linear algebraic systems: applications in science and engineering*; Kluwer Academic Publ. 1998.  
 [2] O. Axelsson, *Iterative solution methods*; Cambridge University Press 1994.

- [3] R. Barrett, M.W. Berry, T.F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. van der Vorst, *Templates for the solution of linear systems: building blocks for iterative methods*; SIAM 1993.
- [4] F. Berman, G. Fox, T. Hey, *Grid computing: making the global infrastructure a reality*; J. Wiley 2003.
- [5] J.J. Dongarra, I.S. Duff, D.C. Sorenson, H.A. van der Horst, *Numerical linear algebra for high-performance computers*; SIAM 1998.
- [6] L. Erlander, "Distributed computing: an introduction"; *Extreme Tech*, April 4, 2002.
- [7] V.K. Garg, *Principles of distributed systems*; Kluwer Academic Publ. 1998.
- [8] G.H. Golub, C.F. van Loan, *Matrix computations*; The Johns Hopkins Univ. Press 1983.
- [9] A. Greenbaum, *Iterative methods for solving linear systems* (Frontiers in Applied Mathematics 17); SIAM 1997.
- [10] W. Gropp, E. Lusk, A. Skjellum, *Using MPI: portable parallel programming with the message-passing interface* (2-nd ed.); MIT Press 1999.
- [11] W. Hackbusch, *Iterative solution of large sparse systems of equations* (Applied Mathematical Sciences 95); Springer-Verlag 1995.
- [12] S. Hamilton, "Taking Moore's law into the next century"; *IEEE Computer Magazine*, vol.32, no.1, 1999, pp.43-48.
- [13] R. Merritt, "Intel, clusters on the rise in 'Top 500 Supercomputer' list"; *EE Times Online*, November 18, 2003.
- [14] Y. Saad, *Iterative methods for sparse linear systems* (2-nd ed.); SIAM 2003.
- [15] R.A. Saleh, K.A. Gallivan, M-C. Chang, I.N. Hajj, D. Smart, T.N. Trick, Parallel circuit simulation on supercomputers; *Proceedings of the IEEE* vol.77, no.12, 1989, pp.1915-1931.
- [16] M.R. Steed, M.J. Clement, "Performance prediction of PVM programs"; Proc. 10-th Int. Parallel Processing Symposium (IPPS), 1996, pp.803-807.
- [17] J.A. Stankovic, "Distributed computing"; in *Distributed computing systems*, IEEE CS Press 1994.
- [18] H. van der Vorst, "Iterative methods for linear systems and implementation on parallel computers"; in *Iterative methods in scientific computing*; R.H. Chan, T.F. Chan, and G.H. Golub (eds.), Springer-Verlag 1999, pp.1-44.
- [19] B. Wilkinson, *Computer Architecture – Design and Performance* (2-nd ed.); Prentice Hall 1996.
- [20] "www.beowulf.org" is Beowulf home page.
- [21] "www.climateprediction.net" is ClimatePrediction home page.
- [22] "setiathome.ssl.berkeley.edu" is SETI@home home page.