Scientific Computing on Bulk Synchronous Parallel Architectures

R. H. Bisseling[a] * and W. F. McColl[b] †

[a] Department of Mathematics, Utrecht University, P.O. Box 80010, 3508 TA Utrecht, The Netherlands (bisseling@math.ruu.nl)

[b] Programming Research Group, Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford OX1 3QD, UK (mccoll@comlab.ox.ac.uk)

We theoretically and experimentally analyse the efficiency with which a wide range of important scientific computations can be performed on bulk synchronous parallel architectures.

Keyword Codes: C.1.2; F.1.1; G.1.3
Keywords: Multiprocessors; Models of Computation; Numerical Linear Algebra

1. Introduction

Bulk synchronous parallel (BSP) architectures [1,2] offer the prospect of achieving both scalable parallel performance and architecture independent parallel software. They provide a robust model on which to base the future development of general purpose parallel computing systems. In this paper we theoretically and experimentally analyse the efficiency with which a wide range of important scientific computations can be performed on BSP architectures. The computations considered include the iterative solution of sparse linear systems, molecular dynamics, linear programming, and the solution of partial differential equations on a mesh. We analyse these computations in a uniform manner by formulating their basic procedures as a sparse matrix-vector multiplication. In our analysis, we introduce the *normalised BSP cost* of an algorithm as an expression of the form $a + bg + cl$, where a, b, and c are scalar values which depend on the algorithm, on the number of processors, and on the chosen data distribution. The scalars g and l are the parameters that characterise a BSP architecture. An ideal parallel algorithm has the values $a = 1$, $b = 0$, and $c = 0$; an algorithm with load imbalance has a value $a > 1$; an algorithm with communication overhead has a value $b > 0$; and an algorithm with synchronisation overhead has a value $c > 0$.

As an example, consider the execution of a five-point Laplacian finite difference operator on a two-dimensional toroidal mesh. This operator computes new values at a mesh point using the old values at the mesh point and its direct neighbours to the north, east,

*This work was initiated while this author was a Research Mathematician at Koninklijke/Shell-Laboratorium, Amsterdam.

†Part of this work was done while this author was a Visiting Scientist at Koninklijke/Shell-Laboratorium, Amsterdam.

south, and west. Our BSP algorithm for this computation has a normalised cost on 100 processors of $1.0 + 0.022g + 0.00056l$ for a mesh of size 200×200. This low cost is achieved by distributing the mesh by orthogonal domain partitioning over the processors, assigning a square block of 20×20 mesh points to each processor. The resulting cost value implies that this computation can be performed efficiently on BSP computers with $g \leq b^{-1} \approx 45$ and $l \leq c^{-1} \approx 1800$.

In the design of efficient BSP algorithms, it is important to find a good data distribution. In fact, the choice of a data distribution is one of the main means of influencing the performance of the algorithm. In the BSP model, the partitioning of the data is a crucial issue, as opposed to the mapping of the resulting partitions to particular processors, which is irrelevant. This leads to an emphasis on problem dependent techniques of data partitioning, instead of on hardware dependent techniques that take network topologies into account. The algorithm designer who is liberated from such hardware considerations may concentrate on exploiting the essential features of the problem. In our case, this leads, surprisingly, to the application of sphere packing techniques to reduce communication in molecular dynamics simulations and in mesh computations.

2. The MLIB test set of sparse matrices

Linear algebra is of crucial importance in scientific computing. This paper focuses on one particularly important linear algebra operation, the multiplication of a sparse matrix by a vector. This operation arises in many areas, e.g. it is the basis of most iterative methods for the solution of sparse linear systems $Ax = b$, it represents the execution of the finite difference operator in certain PDE solvers, and it can be used to model two-particle interactions in molecular dynamics simulations. For more details, see [3].

We present experimental results on the efficiency with which sparse matrix-vector multiplication can be performed on BSP architectures. The experiments are performed on a sparse matrix test library MLIB, which we developed with the aim of capturing the essence of a wide range of important scientific computations in the uniform format of a sparse matrix. MLIB consists of 34 sparse matrices and their generating programs. It contains the following classes of matrices:

- `hyp`.r.d.D, the hypercube matrix with radix r, dimension d, and distance D, $r, d, D \geq 1$. For $D = 1$, this is the adjacency matrix of the directed r-ary, d-dimensional hypercube graph. The vertices of this graph form a d-dimensional mesh of $n = r^d$ points; they are numbered lexicographically. Each vertex has directed edges to itself and to its immediate neighbours in each direction. The size of the hypercube matrix is $n \times n$. For $D > 1$, the hypercube graph is obtained by connecting each vertex to those vertices that can by reached by a path of length $\leq D$ in the original $D = 1$ graph. This models certain higher-order finite difference operators.

- `dense`.n, the dense matrix of size $n \times n$. All elements of this matrix are nonzero.

- `random`.n.ρ^{-1}, an $n \times n$ matrix with a random sparsity structure and a nonzero density ρ.

- `hb.x`, the matrix `x` from the Harwell-Boeing collection [4].

- `md`.n.$r_c{}^{-1}$, an $n \times n$ matrix which corresponds to n particles in a three-dimensional molecular dynamics simulation with short-range potentials. The particles i and j interact, i.e. $a_{ij} \neq 0$, if $||\mathbf{r}_i - \mathbf{r}_j|| \leq r_c$, where $\mathbf{r}_i$ is the position of particle i and r_c the cut-off radius. The positions $\mathbf{r}_i = (x_i, y_i, z_i)$, with $0 \leq x_i, y_i, z_i \leq 1$, are given at the end of the file. The interactions assume periodic boundaries.

- `mdr`.n.$r_c{}^{-1}$.ρ^{-1}, an $n \times n$ matrix which corresponds to n particles in a three-dimensional molecular dynamics simulation with short-range potentials and, additionally, an artificial long-range potential for certain randomly selected particle pairs. The sparsity pattern of this matrix is the union of the sparsity patterns of a short-range molecular dynamics matrix with cut-off radius r_c and a random sparse matrix with density ρ. Here, long-range interactions between selected particles represent interactions between distant clusters of particles. The aim of this procedure is to mimic e.g. multipole expansions.

- `lp`.n, an $n \times n$ matrix which resembles certain matrices that occur in the solution of linear programming problems.

The matrices in the library range in size from `hb.impcol` ($n = 59$, #nonzeros=312) through to `hyp.20.4.1` ($n = 160000$, #nonzeros=1440000).

3. Sparse matrix-vector multiplication

In this section we present a parallel algorithm for the multiplication of a sparse matrix A by a dense vector $\mathbf{v}$ to produce a vector $\mathbf{u}$. The matrix $A = (a_{ij},\ 0 \leq i, j < n)$ has size $n \times n$ and the vectors $\mathbf{u} = (u_i,\ 0 \leq i < n)$ and $\mathbf{v} = (v_i,\ 0 \leq i < n)$ have length n. We assume that the matrix is distributed by a *Cartesian distribution* [5]. This means that the processors are numbered by two-dimensional identifiers (s, t), with $0 \leq s < q_0$ and $0 \leq t < q_1$, where $p = q_0 q_1$ is the number of processors, and that there are mappings $\phi_0 : \{0, 1, \ldots, n-1\} \to \{0, 1, \ldots, q_0 - 1\}$ and $\phi_1 : \{0, 1, \ldots, n-1\} \to \{0, 1, \ldots, q_1 - 1\}$ such that matrix elements are distributed according to

$$a_{ij} \longmapsto \text{processor}(\phi_0(i), \phi_1(j)).$$

Vectors are distributed in the same manner as the diagonal of the matrix, i.e. according to

$$u_i \longmapsto \text{processor}(\phi_0(i), \phi_1(i)).$$

This particular distribution scheme is flexible enough to accommodate many commonly used distribution methods while it is also sufficiently restrictive to impose efficient communication patterns. A detailed discussion and motivation of this distribution scheme in the context of sparse matrix-vector multiplication is given in [6].

Sparse matrix-vector multiplication can be performed on a BSP computer in four supersteps: a fan-out of vector components to the processors that need them; a multiplication of the local part of the sparse matrix by the corresponding part of the input vector; a fan-in of partial sums; and, finally, the computation of the local part of the output vector. The fan-out and the fan-in are h-relations [1]; the other supersteps are local computations.

$\{\ A:\ n \times n,\ \mathrm{distr}(A) = \phi,$
$\ \mathbf{v}:\ n,\ \mathrm{distr}(\mathbf{v}) = \mathrm{distr}(\mathrm{diag}(A))\ \}$
{ fan-out }
for all $j : 0 \le j < n \ \wedge\ \phi_0(j) = s \ \wedge\ \phi_1(j) = t$ **do**
    send v_j to processors $\{(\phi_0(i), t) : 0 \le i < n \ \wedge\ a_{ij} \neq 0\}$;
{ local sparse matrix-vector multiplication }
for all $i : 0 \le i < n \ \wedge\ \phi_0(i) = s \ \wedge\ (\exists r : 0 \le r < n \ \wedge\ \phi_1(r) = t \ \wedge\ a_{ir} \neq 0)$ **do**
begin
    $u_{it} := 0$;
    for all $j : 0 \le j < n \ \wedge\ \ \wedge \phi_1(j) = t \ \wedge\ a_{ij} \neq 0$ **do** $u_{it} := u_{it} + a_{ij} v_j$
end;
{ fan-in }
for all $i : 0 \le i < n \ \wedge\ \phi_0(i) = s \ \wedge\ u_{it} \neq 0$ **do**
    send u_{it} to processor $(s, \phi_1(i))$;
{ summation of partial sums }
for all $i : 0 \le i < n \ \wedge\ \phi_0(i) = s \ \wedge\ \phi_1(i) = t$ **do**
begin
    $u_i := 0$;
    for all $k : 0 \le k < q_1 \ \wedge\ u_{ik} \neq 0$ **do** $u_i := u_i + u_{ik}$
end
$\{\mathbf{u}:\ n,\ \mathbf{u} = A\mathbf{v},\ \mathrm{distr}(\mathbf{u}) = \mathrm{distr}(\mathbf{v})\}$

Figure 1. Sparse matrix-vector multiplication program for processor (s, t)

Figure 1 gives the program text for processor (s, t), with $0 \le s < q_0$ and $0 \le t < q_1$. The total BSP cost of the algorithm is obtained by adding the computation and communication costs of the four supersteps. Denoting the BSP cost for p processors by $T(p)$ we define the *normalised BSP cost* $C(p)$ to be $pT(p)/T_{\mathrm{seq}}$ where T_{seq} is the cost of the sequential algorithm. In other words, the normalised BSP cost $C(p)$ of an algorithm is the ratio between the time $T(p)$ of that algorithm on a BSP computer and the time T_{seq} of a perfectly parallelised sequential algorithm. The normalised BSP cost of an algorithm is an expression of the form $a + bg + cl$. The normalised cost of an ideal BSP algorithm would be $1 + 0g + 0l$.

4. Results for structure independent distributions

We have implemented a program that computes the normalised BSP cost $a + bg + cl$ of sparse matrix-vector multiplication for a given sparse matrix and a given data distribution. Our cost statistics can be used to predict the computing time on an actual BSP computer, provided that the g and l parameters of the machine are available. For our experiments, we fixed the number of processors at $p = 100$.

The c coefficient depends only on p, T_{seq} and the number of supersteps, i.e. it is independent of the data distribution. Many data distributions give values close to 1 for the a coefficient. The results in [3] show, however, that the b coefficient depends crucially on the data distribution. For example, consider the MLIB matrix `hyp.200.2.1` ($n = 40000$,

#nonzeros=200000). The program shows that for a "PRAM distribution", i.e. random allocation of matrix elements to processors, $(a, b, c) = (1.06, 0.96, 0.0011)$ whereas for a "block-grid distribution" it is $(1.00, 0.23, 0.0011)$. For the matrix `dense.500` ($n = 500$, #nonzeros=250000) we have $(1.12, 0.42, 0.0008)$ for PRAM and $(1.00, 0.02, 0.0008)$ for block-grid. For details of these results, and many others, see [3].

5. Results for structure dependent distributions

The b coefficient for `hyp.200.2.1` can be further reduced to 0.022 by distributing the matrix elements according to an orthogonal partition of the corresponding 200×200 mesh into 20×20 contiguous blocks. This is an immediate consequence of the surface-to-volume effect, where the communication across the block boundaries grows as the number of points near the surface, and the computation as the number of points within the volume of the block. It is possible to improve the b coefficient even further, to about 0.016, by partitioning the 200×200 mesh into "digital circles" where all mesh points with a Manhattan distance less than or equal to a radius r from the centre of such a circle are allocated to the same processor. The circles wrap around the boundaries of the mesh. Figure 2 illustrates this kind of distribution. These "digital sphere packing" techniques can be generalised to give very efficient BSP domain decomposition methods for higher dimensional hypercube matrices and for molecular dynamics matrices. For details, see [3].

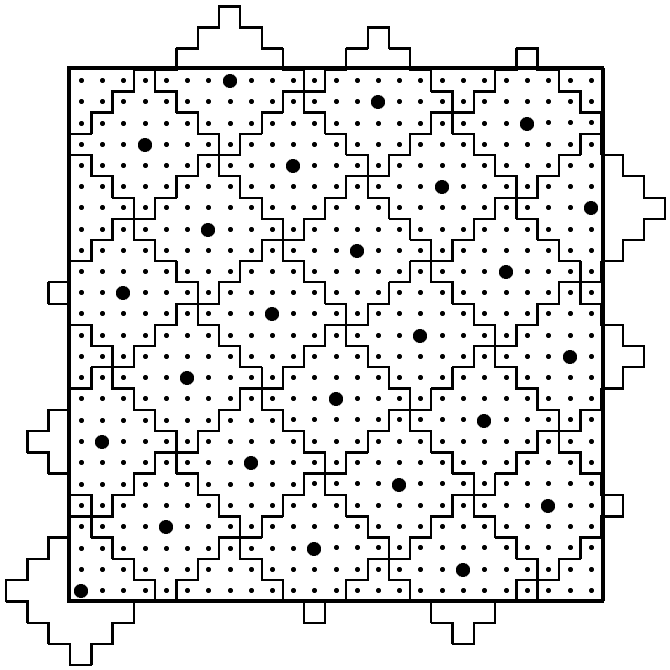

Figure 2. Partitioning of a 25×25 mesh into 25 digital circles of radius 3

6. Conclusion

The BSP model provides a new foundation for the development of scalable parallel computing systems. It offers a robust framework within which we can unify the various

classes of parallel computers which are being produced (distributed memory architectures, shared memory multiprocessors, networks of workstations). The model permits and encourages the development of efficient parallel algorithms and programs which are both scalable and portable.

In this paper and [3] we provide the first theoretical and experimental analysis of the efficiency with which a wide range of important scientific computations can be performed on bulk synchronous parallel architectures. Our analysis shows that the exploitation of knowledge about the underlying structure of the problem is extremely important in achieving efficient parallel computations on a BSP computer. Highly irregular sparse matrix problems without a known structure are likely to be very difficult to solve efficiently on any parallel architecture which has a large g value. Our results show clearly that many structure independent parallel computations require extremely high communication performance and demand values of g that at present are difficult to achieve. Providing a library of parallel algorithms to solve general sparse problems is a first step towards efficient parallel scientific computing, but to make further progress, this should be combined with developing algorithms that find structure in the problems [7].

The initial techniques and results described here show clearly that the network independent approach of the BSP model gives rise to a whole range of interesting new theoretical questions concerning load balancing, communication complexity, and domain partitioning for parallel scientific computing. In contrast to the many network specific (e.g. hypercube, mesh, or butterfly) process mapping and domain decomposition methods which were developed over the last decade, the techniques and results described here have an advantage in that they are of relevance to any parallel computing system.

REFERENCES

1. L. G. Valiant, A bridging model for parallel computation, *Commun. ACM*, **33** (1990), pp. 103–111.
2. W. F. McColl, General purpose parallel computing. In: A. M. Gibbons and P. Spirakis (eds.), *Lectures on Parallel Computation. Proc. 1991 ALCOM Spring School on Parallel Computation*, Cambridge University Press, Cambridge, UK, 1993, pp. 337–391.
3. R. H. Bisseling and W. F. McColl, Scientific Computing on Bulk Synchronous Parallel Architectures, Preprint No. 836, Department of Mathematics, Utrecht University, 1993, 31pp.
4. I. S. Duff, R. G. Grimes, and J. G. Lewis, Users' guide for the Harwell-Boeing sparse matrix collection (Release I). Technical Report TR/PA/92/86, CERFACS, Toulouse, France, 1992.
5. R. H. Bisseling and J. G. G. van de Vorst, Parallel LU decomposition on a transputer network. In: G. A. van Zee and J. G. G. van de Vorst (eds.), *Parallel Computing 1988, Shell Conference*, Lecture Notes in Computer Science, **384**, Springer-Verlag, Berlin, 1989, pp. 61–77.
6. R. H. Bisseling, Parallel iterative solution of sparse linear systems on a transputer network, In: A. E. Fincham and B. Ford (eds) *Proc. IMA Conf. on Parallel Computing, Oxford 1991*, Oxford University Press, Oxford, UK, 1993, pp. 253–271.
7. A. Pothen, H. D. Simon, and K.-P. Liou, Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, **11** (1990), pp. 430–452.