aspects of computation on asynchronous parallel processors

edited by
m. wright

IFIP

north-holland

ASPECTS OF COMPUTATION ON ASYNCHRONOUS PARALLEL PROCESSORS

Proceedings of the IFIP WG 2.5 Working Conference on
Aspects of Computation on Asynchronous Parallel Processors
Stanford, CA, USA, 22–26 August, 1988

edited by

Margaret WRIGHT
AT & T Bell Laboratories
Murray Hill
New Jersey, U.S.A.

1989

NORTH-HOLLAND
AMSTERDAM · NEW YORK · OXFORD · TOKYO

ISBN: 0 444 87310 4

Published by:

ELSEVIER SCIENCE PUBLISHERS B.V.
P.O. Box 103
1000 AC Amsterdam
The Netherlands

Sole distributors for the U.S.A. and Canada:

ELSEVIER SCIENCE PUBLISHING COMPANY, INC.
655 Avenue of the Americas
New York, N.Y. 10010
U.S.A.

Library of Congress Cataloging-in-Publication Data

```
IFIP WG 2.5 Working Conference on Aspects of Computation on
  Asynchronous Parallel Processors (1988 : Stanford, Calif.)
    Aspects of computation on asynchronous parallel processors :
  proceedings of the IFIP WG 2.5 Working Conference on Aspects of
  Computation on Asynchronous Parallel Processors, Stanford, CA, USA,
  22-26 August, 1988 / edited by Margaret Wright.
        p.   cm.
    Includes index.
    ISBN 0-444-87310-4 (U.S.)
    1. Parallel processing (Electronic computers)--Congresses.
  I. Wright, Margaret H.  II. Title.
  QA76.5.I2775  1988
  004'.35--dc19                                          88-37537
                                                              CIP
```

PRINTED IN THE NETHERLANDS

PREFACE

This book includes the papers presented at Working Conference 5, "Aspects of computation on asynchronous parallel processors", organized by Working Group 2.5 (Numerical Software) on behalf of Technical Committee 2 of the International Federation for Information Processing. The conference was held at Stanford University, Stanford, California, from August 22–26, 1988, with 75 invited participants from 14 countries.

Planning for the conference began in the summer of 1984, when it appeared that increasing availability of asynchronous parallel processors was beginning to provide major opportunities for original and useful work in scientific computing. In the intervening four years, both interest and activity in parallel computation have grown to a dramatic extent, and many other conferences have been devoted to parallel computing. Nonetheless, our conference had a distinctive character because of its deliberately limited size and informal atmosphere.

The field of parallel computing is still in a highly volatile state, and researchers display a wide range of opinion about many fundamental questions such as models of parallelism, approaches for detecting and analyzing parallelism of algorithms, and tools that allow software developers and users to make effective use of diverse forms of complex hardware. Accordingly, the conference was intended to be neither a state-of-the-art survey nor a general review of the field. Rather, the hope was that the conference would provide an opportunity for productive exchanges of ideas among researchers who specialize in different aspects of parallel computing, and for the emergence of tentative overall themes. This hope was amply fulfilled.

It is traditional at these working conferences to build substantial time for discussion into the schedule, and to include the transcribed discussion in the conference proceedings; readers of this volume are thus able to recapture some of the lively flavor of the conference. To encourage the sharing of new ideas, an "open session" is always scheduled for short presentations. This volume differs from previous conference proceedings in that extended abstracts or short versions of papers from the open session are included, along with the ensuing discussion.

The conference co-chairmen were Richard J. Hanson and Brian T. Smith, and Margaret H. Wright was the local arrangements chairman. In addition, the program committee included Thomas J. Aird, Françoise Chatelin, L. Dekker, L. M. Delves, Bo Einarsson, Brian Ford, Morven Gentleman, Robert Huddleston, James R. McGraw, George Paul, Jr., John K. Reid and John R. Rice.

The meeting received generous support from several co-sponsors: the Computation Department, Lawrence Livermore National Laboratory, Livermore, California; Applied Dynamics International, Ann Arbor, Michigan; and the Algorithms Group, Systems Optimization Laboratory, Department of Operations Research, Stanford University. Special thanks are due to Robert Huddleston of Lawrence Livermore National Laboratory for his support and encouragement of the conference.

The meeting was officially hosted by the Department of Operations Research, Stanford University, and we accordingly thank its chairman, Donald L. Iglehart, and the Algorithms Group of the Systems Optimization Laboratory (Philip E. Gill, Walter Murray and Michael A. Saunders).

Equipment that formed an essential part of two presentations was generously provided by Bonnie Toy on behalf of Sun Microsystems, Inc., Mountain View, California.

The session chairmen and discussants served a crucial role in the success of the conference. The session chairmen were W. J. Cody, Jr., Theodorus J. Dekker, Bo Einarsson, Brian Ford, Lloyd D. Fosdick, George Paul, Jr., John K. Reid and Mladen A. Vouk. The discussion in this volume was recorded through the diligent work of the discussants: Thomas J. Aird, Maurice Clint, Stuart Feldman, Lloyd D. Fosdick, W. Morven Gentleman, Fred G. Gustavson, Sven J. Hammarling, Richard J. Hanson, Elias Houstis, Olin G. Johnson, Charles L. Lawson, Paul C. Messina, Theodore Papatheodorou, John R. Rice, Frederic N. Ris, Brian T. Smith and Danny C. Sorensen. The discussants were impeccably organized by Thomas J. Aird, who deserves particular mention for his efforts.

Arrangements before and during the conference itself were coordinated by Gail L. Stein and by Patricia A. Schermerhorn of the Department of Operations Research, Stanford University. Their organization, dedication, helpfulness and calm contributed immeasurably to the smooth running of the conference. SLW Associates—in particular, Margaret London and Susan Sweeney—provided expertise that made even the most complex details of conference organization seem effortless for the participants.

As editor of these proceedings, I thank Jerri Rudnick for her help in preparing the TEX versions of the papers that appeared in the conference booklet, and Carol Galgano for her skillful assistance with the final copy.

Margaret H. Wright
Murray Hill, New Jersey
November 2, 1988

CONFERENCE WELCOME

On behalf of Working Group 2.5 of the International Federation of Information Processing Societies, I welcome you to this working conference, our fifth conference on aspects of numerical computation. Our working group interprets this subject, numerical computation, broadly. We are interested in more than algorithms for solving numerical problems; we are interested in the whole framework and the mechanics of numerical computation. Thus, in addition to algorithms, our working conferences have been concerned with software tools and libraries, with programming languages, with arithmetic, and with the portability and reliability of programs—all in the context of numerical computation.

As with all conferences, the principal objective of this conference is the direct exchange of ideas, and of research results. To promote this and to ensure good coverage our working conferences are one week long. Attendance is limited in order to help ensure that all participants have a good opportunity to ask questions and discuss their ideas. Publication of the conference proceedings provides a record of the talks and a summary of the discussions.

Ours is an international organization and so we have another objective, indeed an important obligation. That is to bring together scientists from all over the world, and to assure a world-community representation among speakers and attendees. This is not easy to do. Costs of travel are prohibitive for many. Large-scale computation requires expensive facilities, thus restricting research opportunities for some of us. Nevertheless, in organizing these conferences we seek participants from all parts of the world and we do our best to assure fair and balanced representation.

The subject of this working conference is computation on asynchronous parallel processors. The formal presentations have been grouped into general areas as follows: scientific applications; languages; and libraries, environments and tools. Each morning or afternoon session is devoted to one of these areas. On Thursday afternoon we have allocated time for an "open session" where short, unscheduled presentations can be made. We have allocated time for discussion after each paper and a longer time for general discussion at the end of each morning and afternoon session.

A tradition we follow is to have an afternoon excursion midway through the conference followed by a dinner and, on another evening, a banquet. Our excursion will be Wednesday afternoon, and the banquet will be Thursday evening. We hope that you will come to these events and enjoy further discussions in a relaxed atmosphere and have fun.

Brian Smith, Richard Hanson and Margaret Wright were the principals in putting this conference together. They deserve thanks from all of us. Brian and Richard co-chaired the program committee, and Margaret took care of the local arrangements.

We want all of you to take this opportunity to learn and discuss ideas in parallel computation, to share your research ideas, to make new research contacts, and thus to help make this meeting a success for all of us.

Lloyd D. Fosdick
Chairman, Working Group 2.5

CONTENTS

EFFECTIVE SCIENTIFIC APPLICATIONS, Session 1
Chair: Lloyd D. Fosdick
Discussants: Charles L. Lawson, John R. Rice

Solving Elliptic Equations on the Cedar Multiprocessor 1
E. Gallapoulos and A. Sameh

Parallel Eigenvalue and Singular Value Algorithms for Signal Processing 13
K. V. Fernando and S. J. Hammarling

EFFECTIVE SCIENTIFIC APPLICATIONS, Session 2
Chair: John K. Reid
Discussants: Thomas J. Aird and Richard J. Hanson

Analysis of a Two-Level Asynchronous Algorithm for PDES 23
John R. Rice and Dan C. Marinescu

Parallelization on an MIMD Computer with Real-Time Scheduler, Gauss-Jordan Example 35
Serge G. Petiton

A Vector and Parallel Implementation of the FFT Algorithm on the IBM 3090 . . 45
Ramesh C. Agarwal

A Language Comparison for Scientific Computing on MIMD Architectures 55
Mark T. Jones, Merrell L. Patrick and Robert G. Voigt

EFFECTIVE SCIENTIFIC APPLICATIONS, Session 3
Chair: T. J. Dekker
Discussants: Sven J. Hammarling and Maurice Clint

Multilevel Parallel Solver for Banded Linear Systems 69
Ibrahim N. Hajj and Stig Skelboe

Asynchronous Parallel Iterative Methods 79
Avi Lin

A Project for Developing a Linear Algebra Library for High-Performance Computers 87
J. Demmel, J. J. Dongarra, J. DuCroz, A. Greenbaum, S. J. Hammarling and D. C. Sorensen

Monte Carlo Simulations of Lattice Gauge Theories 93
Philippe de Forcrand

LANGUAGES AND PROGRAMMING ISSUES
Chair: Mladen A. Vouk
Discussants: Morven Gentleman and Fred Gustavson

Asynchronous Parallel Execution of a Multiply Nested For Loop by Vector Indirect Addressing 101
Takao Tsuda

Ultra-High Performance Multiprocessor Supersystems with Programmable Architecture 111
A. V. Kalyaev

LIBRARIES, ENVIRONMENTS AND TOOLS
Chair: W. J. Cody, Jr.
Discussants: Paul C. Messina, Olin G. Johnson

Tools and Methodology for Programming Parallel Processors 125
Jack J. Dongarra, Danny C. Sorensen and Orlie Brewer

Monitoring Parallel Programs on Message-Passing Multiprocessors 139
A. P. W. Böhm, J. R. Gurd and M. C. Kallstrom

Numerical Library Facilities for Transputer Arrays 159
L. M. Delves and N. G. Brown

Aspects of the Design of Portable and Efficient Basic Linear Algebra in Ada . . 171
M. J. Erl, G. S. Hodgson and L. Steenman-Clark

LANGUAGE DESIGN ISSUES
Chair: Bo Einarsson
Discussants: Stuart Feldman and Danny C. Sorensen

Automatic MIMD Parallelization 181
Hans P. Zima

Parallel Programming Experiments with Linda 193
Nicholas Carriero, David Gelernter, Susanne Hupfer, James Narem and Raman Sundaresh

Parallel Processing based on Active-Data for Asynchronous Systems 201
Chris Jesshope

SISAL Implementation and Performance 211
Stephen Skedzielewski

OPEN SESSION
Chair: John K. Reid
Discussants: Thomas J. Aird, Elias Houstis, Theodore Papatheodorou

A Parallel Implementation of Matrix Multiplication and *LU* Factorization on the IBM 3090 217
Ramesh C. Agarwal and Fred G. Gustavson

A Parallel *QR* Decomposition Algorithm 222
Lars Eldén

Optimizing a Reconfigurable Transputer Array for Line-of-Sight Communications 223
J. S. Ward, J. B. G. Roberts and P. Simpson

Solution of Ordinary Differential Equations by Waveform Relaxation Methods 225
Stig Skelboe

Recursive Task Generation 226
Geoffrey Hunter and Tyler A. Ivanco

DINO–A New Language for Numerical Computation on Distributed Memory Multiprocessors 232
Matthew Rosing and Robert B. Schnabel

Mapping SISAL Programs for Efficient Execution on the Connection Machine 234
Jack B. Dennis

Parallel Computing Forum (PCF) Fortran 235
Brian T. Smith

A Model of Fault-Tolerant Dataflow Computers 236
Milan Ojsteršek and Viljem Zumer

SYSTEM DESIGN ISSUES

Chair: Brian Ford
Discussants: Thomas J. Aird and Brian T. Smith

Synchronous versus Asynchronous Communication in High Performance Multicomputer Systems 239
P. M. Behr, W. K. Giloi and W. Schröder

EFFECTIVE SCIENTIFIC APPLICATIONS, Session 4

Chair: George Paul, Jr.
Discussants: Lloyd D. Fosdick and Frederic N. Ris

Distributable Algorithms for the Symmetric Eigenproblem 249
M. Clint, D. Roantree and A. Stewart

Coarse Grid Acceleration of Some Domain Decomposition Methods on Multiprocessors 255
Garry Rodrigue and Ted Ferretta

Future Scientific Programming 261
Arvind

PARTICIPANTS 265

AUTHOR INDEX 271

Aspects of Computation on
Asynchronous Parallel Processors
M. H. Wright (Editor)
Elsevier Science Publishers B.V. (North-Holland)
© IFIP, 1989

SOLVING ELLIPTIC EQUATIONS ON THE CEDAR MULTIPROCESSOR

E. GALLAPOULOS and A. SAMEH

Center for Supercomputing Research and Development
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

1. Introduction

We present some of the work in progress at CSRD for the solution of elliptic (Laplace, Poisson, and more general) partial differential equations on the Cedar system. The decribed algorithms are block cyclic reduction, boundary based domain decomposition, and conjugate gradient. These algorithms take advantage of the parallelism and vectorization available on the vector multiprocessors which constitute a single Cedar cluster. We discuss the effect of the memory hierarchy on performance and study some implementation issues for multiple clusters. For a detailed explanation of the numerical algorithms involved we refer to [10, 9, 8, 18].

2. Computational Environment

The Cedar system of the University of Illinois is characterized by the hierarchical organization of both its computational capabilities and memory system. It consists of multiple clusters, each of which is a multivector processor comprising 8 computational elements (CE's) (see Figure 1). Thus parallelism can be exploited at three levels. Within each CE, operations on vectors can be done in vector mode, then, as each cluster is a tightly coupled multivector processor, fine grained parallelism can be exploited. Finally, the clusters can be used for medium and large grain parallelism (and certain types of fine grain parallelism). Presently, each cluster is a modified Alliant FX/8.

The memory organization is hierarchical as well with communication increasing in cost at each level. At the lowest level each CE has a set of private scalar and vector registers. The next two levels, a cache and cluster memory, are shared by the CE's within the same cluster. Finally, all clusters have access to a large global memory. This global memory is accessed through two unidirectional switching networks, one for downloading data from memory, the other for uploading. These switching networks are 2-stage omega networks built from 8×8 cross-bar switches. For a more detailed description of the Cedar system see [14].

The Cedar operating system, Xylem, is a modification of Alliant's Concentrix operating system extended for multitasking and virtual memory management of the Cedar memory hierarchy ([5,17]). A Xylem process consists of one or more *cluster-tasks*. Multiple cluster-tasks execute asynchronously across the Cedar system. Xylem provides system calls for starting and stopping tasks, and waiting for tasks to finish. System calls are also provided for coarse-grained inter-task synchronization. In addition to multitasking, Xylem supports multiprogramming whereby multiple processes can be executing simultaneously. In addition to multitasking, Xylem supports multiprogramming whereby multiple processes can be executing simultaneously. The Xylem virtual memory system provides convenient access to the Cedar physical memory hierarchy.

Fortran is the focus of language and compiler development for Cedar because of its dominance in scientific programming. Cedar Fortran is derived from Alliant FX/Fortran with extensions for memory allocation, concurrency control, multitasking, and synchronization [11]. New data type specification statements reflect the Xylem memory access and locality structure. Vector concurrency is available through array section notation, conditional vector statements, and vector reduction functions. DOALL and DOACROSS constructs specify parallel execution of loop iterations on processors within a single cluster task or spread across multiple cluster tasks. Multitasking routines provide an interface between Cedar Fortran and Xylem for task creation and control. A set of synchronization functions allows access to the Cedar hardware synchronization primitives. Cray-style synchronization operations are also provided. Multitasking and synchronization routines are implemented as part of a Cedar Fortran run-time library. Compiler optimizations for vectorization, parallelization, and memory allocation are being developed for the Cedar machine based on the *Parafrase* restructurer ([15]).

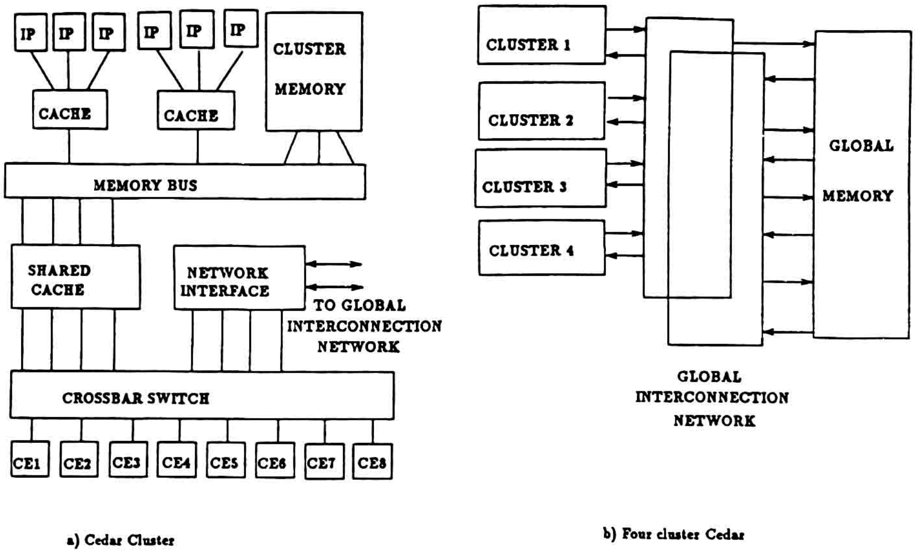

Figure 1. Cedar cluster and multicluster organization

3. Dense Matrix Computations on Cedar

A large effort has been made to develop efficient dense matrix operations for a single Cedar cluster ([2]). It quickly became apparent that the memory hierarchy - in this case the Alliant vector registers, cache and main memory - could penalize performance unless algorithms with sufficient data locality were developed ([7]). This led to exploiting the concept of block algorithms and the design and implementation of BLAS3 ([13, 6]). Indeed, algorithms designed based on BLAS3 kernels demonstrated impressive speedups on a single cluster. Curve QR1 in Figure 2 shows the performance improvements for QR factorization on 8 CEs of the Alliant FX/8 when a BLAS3 based block Householder algorithm is used as opposed to BLAS2 (curve QR2) and BLAS1 (curve QR1) ([12]).

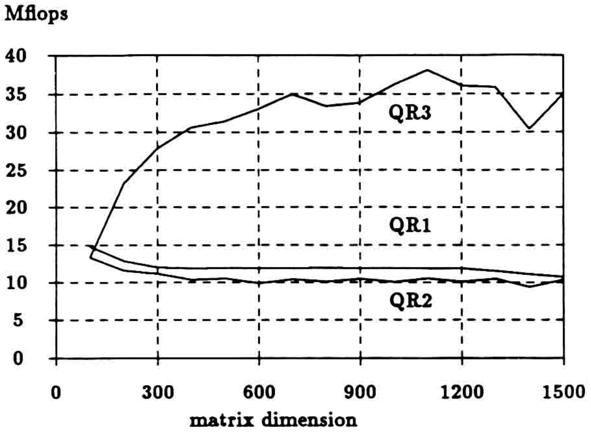

Figure 2. Using QR and BLAS3 ([12]).

4. Block Cyclic Reduction

The discretization of the separable elliptic equation

$$a(x)\frac{\partial^2 u}{\partial x^2} + b(x)\frac{\partial u}{\partial x} + c(x)u + \frac{\partial^2 u}{\partial y^2} = f(x,y) \tag{1}$$

with Dirichlet boundary conditions and a five-point stencil on a naturally ordered $n \times m$ grid defined on a rectangular region leads to a system of the form $\mathcal{A}u = f$. In this case $\mathcal{A}$ is the n block tridiagonal matrix diag$[-I, A, -I]$, where A, I are respectively tridiagonal and identity matrices of order m.

Block cyclic reduction (BCR for short) dates from the work of Hockney and was presented in [3] in its stabilized form due to Buneman. The work in [21, 20, 19] resulted in the development of FISHPAK, a package based on BCR for the solution of (1) and extensions thereof. BCR is a rapid elliptic solver (RES) having sequential computational complexity $O(nm \log n)$. Assuming that $n = 2^k - 1$, the idea of the method for reduction steps $r = 1, \dots, k-1$ is to combine the current $2^{k-r+1} - 1$ vectors into $2^{k-r} - 1$ ones, and then solve a system of the form

$$p_{2^{r-1}}(A)X = Y$$

where $Y \in \Re^{m \times (2^{k-r}-1)}$ and $p_{2^{r-1}}(A)$ is a Chebyshev polynomial of degree 2^{r-1} in A. Since its roots $\lambda_i^{(r-1)}$ are known, it can be written in product form, where each factor is tridiagonal. Hence the system to be solved becomes

$$\prod_{i=1}^{2^{r-1}} (A - \lambda_i^{(r-1)} I)[x_1 | \cdots | x_{2^{k-r}-1}] = [y_1 | \cdots | y_{2^{k-r}-1}]. \tag{2}$$

Clearly as r increases,the effectiveness of a parallel or vector machine to handle (2) decreases rapidly. A parallel version of BCR was recently discovered ([22,10]). In summary, the method is based in expressing the matrix rational function $[p_{2^{r-1}}(A)]^{-1}$ as a partial fraction, i.e. as a linear combination of the 2^{r-1} components $(A - \lambda_i^{(r-1)} I)^{-1}$.

$$[x_1 | \cdots | x_{2^{k-r}-1}] = \sum_{i=1}^{2^{r-1}} \alpha_i^{(r-1)} (A - \lambda_i^{(r-1)} I)^{-1} [y_1 | \cdots | y_{2^{k-r}-1}]. \tag{3}$$

Coefficients $\alpha_i^{(r-1)}$ are equal to $1/(p'_{2^{r-1}}(\lambda_i^{(r-1)}))$ and can be derived analytically. Figure 3 shows the performance of the parallel and standard BCR on the Alliant FX/8.

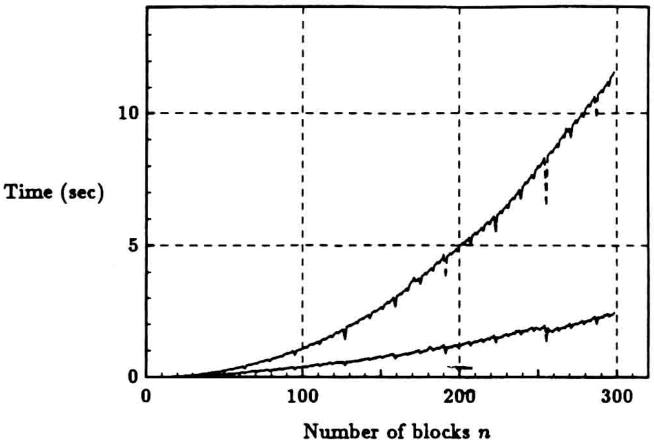

Figure 3. Parallel and standard BCR on $n \times n$ grid on Alliant FX/8

One very important trend in parallel processing is the development of powerful restructuring compilers, which would minimize the need for manual tuning when porting codes to parallel machines ([15]). FISHPAK subroutine hwscrt, which performs block cyclic reduction for rectangles in Cartesian coordinates, presents many challenging issues to the restructurer especially when oriented towards a system with a computational hierarchy. The data dependencies in hwscrt typically make very difficult the automatic recognition of more than one level of loop parallelism. Figure 4 shows part of the (manually restructured) code corresponding to the r^{th} step of the parallel version of the reduction. We observe that even here, which of the loops to vectorize and which to make exploit the concurrency depends on the relation between the lengths of the inner and outer loops, which in turn depends on r. Advanced compiler techniques such as *loop interchanging* ([23]) and generation of *multiple version loops* ([4]) would be useful to generate efficient code. Even such techniques, however, are not fully adequate in the context of Figure 4, since depending on r and m, a different multiple tridiagonal solver may be chosen. For our experiments we have generated manually multiple version loops. The most appropriate version of each loop to be used at every step of the computation was chosen after performance comparisons.

```
c   this is part of step r of the reduction
c   currently jst = 2**(r-1), l = 2**r, jsp = 2**k-2**r
    do j=l,jsp,l
       do i=1,m
          d(i,j) = q(i,j)-q(i,j-jst/2)-q(i,j+jst/2) +q(i,j-jst)+q(i,j+jst)
          e(j/l,i) = d(i,j)+q(i,j)-q(i,j-jst-jst/2)-q(i,j+jst+jst/2)
          q(i,j) = d(i,j)
       enddo
    enddo
c   solve for jsp/l right hand sides, each corresponding to jst tridiagonals
    call tridiagonal-solver
```

Figure 4. Code fragment for parallel BCR

The computational kernel for the algorithm is the multiple tridiagonal system solver, together with DAXPY ($a \leftarrow a + sb$) (cf. Figure 6) or DOTPRODUCT (cf. Figure 5) for the combination of the partial fraction terms. The number of data loads for the first two is within a small constant factor from the number of operations, resulting in turn to poor cache utilization and lower performance for the entire algorithm.

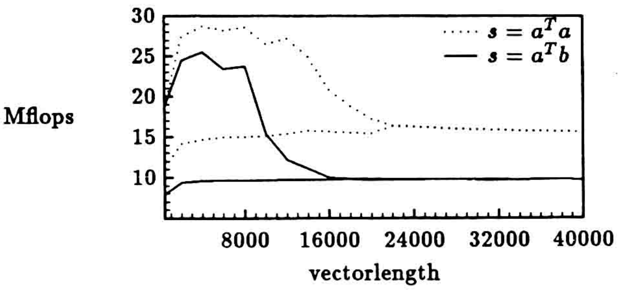

Figure 5. DOTPRODUCT out of cache and out of memory [18]

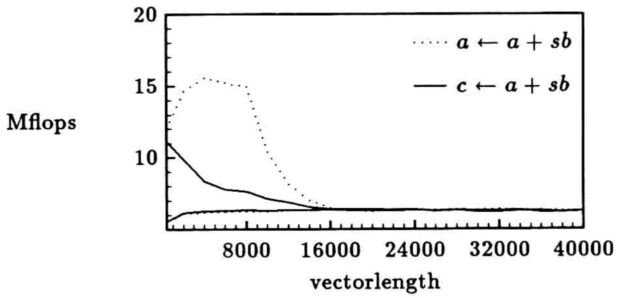

Figure 6. DAXPY out of cache and out of memory [18]

4.1. Multicluster algorithm

We assume initial Cedar configurations of $P = 2$ or $P = 4$ clusters and $n = 2^k - 1$. Minimizing intercluster communication time and partitioning the problem across clusters so that each one is doing useful work are keys to an efficient implementation. Since the tridiagonal matrix A is the seed for all the computations we can assume that at the start of the computation it is copied in each of the two cluster memories and remains there. In Cedar terminology, A has the CLUSTER memory attribute. The total number of tridiagonal systems to be solved in step r is $2^{k-1} - 2^{r-1}$. This is due to the $(2^{k-r} - 1)$ right-hand-sides, each to be solved with 2^{r-1} matrix coefficients.

A variety of cluster splitting strategies can be designed based on these observations. In the simplest one we let each cluster solve the systems corresponding to $(2^{k-r} - 1)/P$ of the righthand side in (2). The problem with this strategy is that there will be workload imbalance between the clusters. For example, when $P = 2$ then in steps $1, 2, \ldots, k - 1$ there will be 1, 2, $\ldots, 2^{k-2}$ more systems to be solved in one of the clusters (the extreme is step $k - 1$ with the one cluster remaining idle). The total load imbalance is thus $2^{k-1} - 1$ extra systems to be solved in the one of the two clusters, which is clearly significant. Because of the synchronization between every step of the computation, alternating the extra load between the 2 clusters is not a satisfactory remedy. Observe that the problem occurs because we have chosen $n = 2^k - 1$ to make the BCR algorithm more efficient whereas P is either 2 or 4. Another option is to base the distribution on the number of coefficient matrices. Unfortunately this strategy involves substantial communication overhead at each step due to required combination of the partial fractions.

A solution can be found by examining the algorithm from a finer-grain point of view, namely (3). Now the total number of systems is $2^{r-1} \times (2^{k-r} - 1)$, which is exactly divisible by 2 and

4 if $r > 1$ and $r > 2$ respectively. Hence, except for the first 1 or 2 steps, the same number of systems (namely $2^{r-1-\log P} \times (2^{k-r} - 1)$) can be assigned to each of the clusters, thus achieving computational load balance. In this case, however, the partial fraction decomposition (3) for a few vectors is calculated in stages. Once the systems corresponding to these vectors are solved, not all the terms of the decomposition are in the same cluster. Instead, the available terms are summed in each cluster, followed by communication of each partial result to the appropriate cluster for the final summation.

Figure 4 characterizes the combinations necessary during step $r < k$ of the reduction, when the algorithm is mapped on a single cluster. It reveals that column j of array q needs to be combined with columns $j \pm 2^{r-1}, j \pm 2^{r-2}, j \pm (2^{r-1} + 2^{r-2})$. The values j takes at that step are $1 \times 2^r, 2 \times 2^r, \ldots, (2^{k-r} - 1) \times 2^r$. Distributing q uniformly across the P clusters it can be shown that only few (e.g., three when $P = 2$) m vectors are needed in a cluster from the adjacent clusters at each step.

5. Boundary Integral Domain Decomposition

A new method (BIDD) was recently proposed for the solution of Laplace's equation ([9], [8]). The method is characterized by the decoupling of the problem into independent subproblems on subdomains. An approximation $\hat{u}$ to the solution u is sought as a finite linear combination of N fundamental solutions ([16]) $\phi_j(z) = -\frac{1}{2\pi} \log |z - w_j|$ of $\nabla^2 u = 0$:

$$\hat{u}(z) = \sum_{j=1}^{N} \sigma_j \phi_j(z) \tag{4}$$

For a given set of N points w_j lying outside the domain, $\sigma \in \Re^N$ is computed to minimize $\|g - G\sigma\|_p$ for some norm p. $G \in \Re^{\nu \times N}$ is the influence matrix consisting of fundamental solutions based at w_j for each boundary point. $g \in \Re^{\nu}$ consists of boundary values for u. Once σ has been computed, the solution at any μ points on the domain is $\hat{u} = H\sigma$, with $H \in \Re^{\mu \times N}$ being the influence matrix for the μ points. Choosing these μ points to be subdomain boundary points, the solution can be computed by applying the elliptic solvers most suitable for each subdomain.

The only part of the computation which is not trivially decoupled is the computation of σ. This is performed on a single or across multiple clusters depending on the availability of a multicluster solver which is efficient for these dimensions. For the remainder of this description we assume that σ has to be computed on a single cluster by means of the block Householder-based QR algorithm (cf. Figure 2). After some initialization work, a task is started on each cluster (using **ctskstart**). Each task computes its share of H and the grid coordinates on (private) CLUSTER variables.

Whether a similar split of the computation should be performed for G and g depends on the communication and event synchronization overhead required so that the cluster responsible for the computation of σ receives all remaining sections of G and g. Each cluster task allocates space for its own copy of σ. All tasks not responsible for σ wait for it to become available (e.g. calling **evwait**). Once σ is available in the responsible clusters, it is copied to a GLOBAL variable in global memory and signals the other tasks (e.g. **evpost**) to continue. Each task copies σ from global memory to the corresponding cluster variable and proceeds. In [8] it was shown that by allowing each task to do a little more work, it can compute its own boundary (interface) values, thus eliminating the need for any further communication. A suitable elliptic solver is then applied to each of the subdomains. Aside from the self-evident advantage of complete decoupling which in turn allows each cluster to be used for each subdomain with no intercluster communication, there is the added advantage that the domain can be partitioned to subdomains which are small enough so that any computation with any data locality in the subdomain solver can exploit the cache effectively. It is also worth noting that the efficient computation of the elements of G and H strongly depends on the availability of fast routines for the computation of the logarithm of an array of values. Other such elementary functions will be required in the case of operators with different fundamental solutions (cf. [8, 1]).

As an example of the issues involved, we cite [8], where horizontal strips were used for the decomposition when solving Laplace's equation on a rectangle. This approach was driven from the expression $O(nm \log n)$ characterizing the sequential computational complexity of BCR. Size reductions in the n direction are expected to contribute more heavily in the reduction of the time required to solve the problem. Figure 7 shows the expected behavior of BIDD when the domain is partitioned into p horizontal strips and $P = p$ clusters are used.

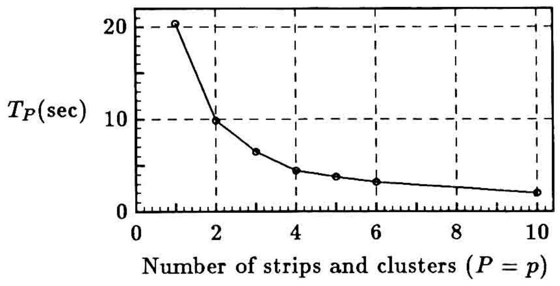

Figure 7. Estimated parallel time, $\nu = 1680$, $n = 418$, $N = 40$.

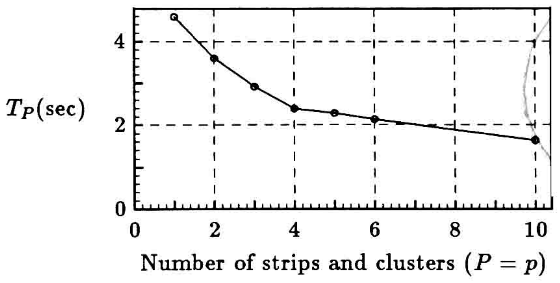

Figure 8. Estimated parallel time, $\nu = 1680$, $n = 418$, $N = 40$, and parallel BCR.

In [8] it was also observed that with the parallel version of BCR described in Section 4, the expected performance improvements on more than 1 clusters are considerably reduced (cf. Figure 8). The slopes of the curves of Figure 3 and figures in [10] show that the time for the parallel algorithm is much less sensitive to changes in the number of blocks n. Instead, reductions in the size (m) of the tridiagonal systems weigh more heavily.

This is demonstrated in Figures 9 and 10, where the method is applied on a $n \times n$ square with $4n$ collocation points, using $N = 40$ fundamental solutions and splitting into 2 horizontal and 2 vertical strips respectively. Note that the effect of vertical stripping is equivalent with the horizontal if vertical instead of natural ordering is used. In any case, the results indicate that BIDD, is also useful for speeding up the solution corresponding to a simple domain on which a rapid elliptic solver like BCR is directly applicable. Moreover, performance considerations following from the architecture indicate strong preference for the one instead of the other decomposition.

6. Conjugate Gradient Methods

More general elliptic equations on more complicated domains are handled by means of Conjugate Gradient methods (CG), frequently combined with a preconditioner (PCG) to modify the spectrum and reduce the number of iterations to convergence. The computational kernels in CG are the DOTPRODUCT, DAXPY, and pentadiagonal matrix-vector multiply. From Figures 5, 6 and 11 and the discussion in Section 4, low performance does not come as a surprise.

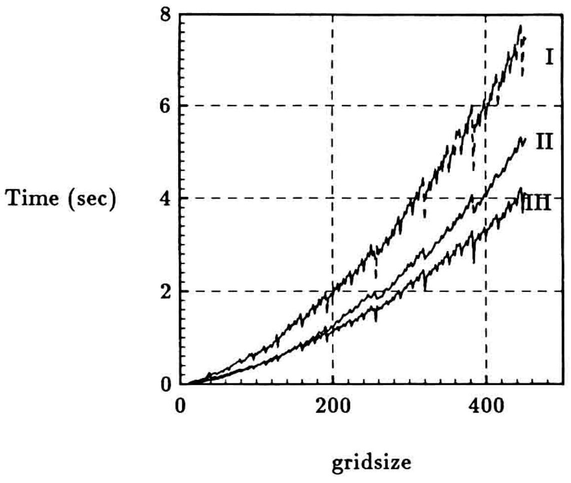

Figure 9. Horizontal strips
I, total FX/8 time; II, parallel BCR; III, expected 2-cluster time.

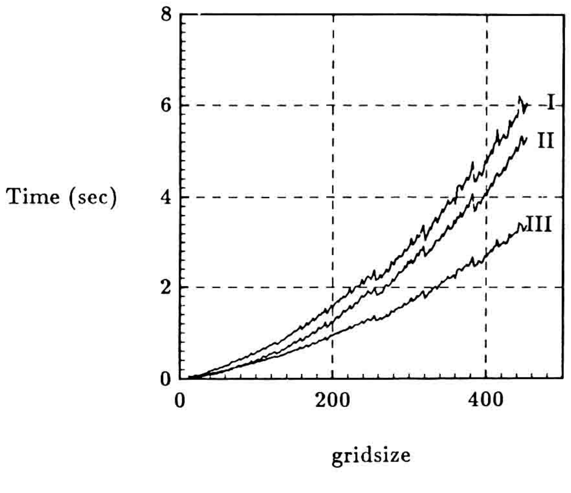

Figure 10. Vertical strips
I, total FX/8 time; II, parallel BCR; III, expected 2-cluster time.