Advanced Computing Research Institute

Theory Center

Cornell University

Semi-annual Research Activity Report

April 1992 - September 1992

Thomas F. Coleman, Director

The Advanced Computing Research Institute (ACRI) is a unit of the Cornell Theory Center and is affiliated with the Cornell Computer Science Department.

The ACRI is concerned with research in scientific computation and its application to engineering and scientific problems. Of particular importance is the use and potential of advanced computer architectures and environments. Research areas include parallelizing compilers for scientific computation and the design of algorithms for numerical linear algebra, optimization, and partial differential equations.

Currently, ACRI researchers are collaborating on several large-scale applications in the computational sciences, including: protein-folding and related molecular chemistry problems, structural optimization and biomechanics, particle methods for turbulent combustion, discrete-control problems, and the application of boundary element methods.

The parallel computers available to the ACRI for research include the Theory Center machines – a 64-node KSR computer, an IBM ES/9000, a network of IBM RS/6000s, as well as Computer Science resources: an 8K CM-200, a 32-node Intel iPSC/860, and a 64-node BBN Butterfly.

This report consists of two parts. The first part contains a short summary of the progress made in the last six months on each of the four main projects: parallelizing compilers, computational linear algebra, computational optimization, and numerical methods for partial differential equations. Included also are a list of ACRI researchers and their research interests, a list of technical reports produced in the last six months, and a list of ACRI seminars.

In the second part we highlight one of the projects, the parallelizing compiler work, where we give a more detailed introduction into this area and sketch our novel approach.

Contents

List of Figures

Part I
Project Summaries and Activities

There are four main projects in the ACRI: parallelizing compilers, computational linear algebra, computational optimization, and numerical methods for partial differential equations. We briefly describe our recent progress in each project.

1 A System for Programming NUMA Architectures

Pingali

Scalable parallel machines are often organized as networks of processor-memory pairs; examples of such machines are the BBN TC 2000, the Kendall Square Research 'all-cache' machine, and multi-computers like the Intel iPSC/i860. These machines are called *non-uniform memory access* (NUMA) machines because a processor can access data in its local memory ten to a thousand times faster than it can access non-local data. Moreover, when many non-local accesses are required, it is usually more efficient to use a single long message than many small messages.

Our approach to programming these machines starts with FORTRAN-77 extended with data decomposition information. We support most of the commonly used data decompositions such as wrapped and blocked row and column distributions. Existing compiling technology for such machines has focused on code generation mechanisms; although it was known that way of applying loop transformations was known. We have developed a novel approach to loop transformations that is based on the use of non-singular matrices to model loop transformations [19]. This approach is more general than existing ones and incorporates common loop transformations like permutation, skewing, reversal and scaling. The major advantage of our approach is that there is a simple completion algorithm which, given a partial transformation matrix, produces a complete transformation matrix respecting all program dependences. We are using this approach to transform loops for locality on NUMA machines [18], as well as to restructure loops for execution on long instruction word machines.

We are applying these techniques to benchmark programs from the molecular simulation community including Professor Keith Gubbins of the Chemical Engineering Department and Professor Michael Teeter of the Applied Physics Department at Cornell. In particular, our techniques for restructuring loop nests for parallelism and locality are applicable to molecular codes using density functional methods. We have also started developing analysis and restructuring techniques for codes using recursive data structures such as trees and graphs; these techniques will be useful for parallelizing molecular simulation codes like Barnes-Hut and Greengard codes. We have recently obtained funding from ONR to pursue this project.

We have also continued our work on using the dependence flow graph to develop novel program optimizations. The dependence flow graph is a new program dependence representation that promises to support more powerful optimization technology than existing program representations like the control flow graph or data dependence graph. We have just completed the design of an algorithm to eliminate partial redundancies in programs, and we are implementing it to see how well it runs in practice. This work is being carried out as part of a joint project with Bob Rau's advanced architectures group at the Hewlett-Packard Corporation.

2 Computational Linear Algebra

Rothman, Sun, Trefethen, Van Loan

2.1 Kronecker Product Preconditioners

Research in the block matrix computation area has proceeded along several lines.

The idea of using Kronecker product preconditioners has been developed in several ways. First, we have shown how to solve the problem of computing the nearest Kronecker product $B \otimes C$ to a given matrix A using the singular value decomposition. Indeed, $\mathrm{vec}(B)$ and $\mathrm{vec}(C)$ are multiples of the singular vectors associated with the largest singular value of a permuted version of A. We have shown that if A is symmetric positive definite, then so are the optimum Kronecker factors. Numerous other 'structured' Kronecker approximation problems such as this have been solved. As preconditioners, these optimum Kronecker products compete very well with the incomplete Cholesky on some of the canonical model problems, e.g., $[(I \otimes A) + (B \otimes I)]x = b$. For example, on a problem of size $n = 256^2$, conjugate gradients with the Kronecker preconditioner converges in 93 iterations.

The use of Kronecker products in photogammetry has led us to formulate a rank-revealing QR factorization method for Kronecker products. We also have developed a Kronecker product interpretation of some well-known lumping techniques in the analysis of Markov processes.

In September of 1992, we gave an invited talk at the IBM sponsored SupEur 92 conference in Umea, Sweden. The talk surveyed the block matrix computation area and featured Cornell work in the development of new block methods and our implementation experiences on the RS/6000 cluster.

2.2 Iterative methods for large-scale linear algebra problems

Increasingly, large systems of linear equations tend to be solved by preconditioned iterative methods. The prototype of such methods is the preconditioned conjugate gradient iteration, applicable when the matrix is symmetric and positive definite. Nonsymmetric matrices, however, pose challenges that are a topic of active current research in the U.S. and Europe. In the last three years alone, important developments in this field include the new iteration called QMR (Freund and Nachtigal), another new iteration called Bi-CGSTAB (Van der Vorst), and techniques for enhancing stability of such algorithms by 'lookahead' strategies (Gutknecht, Golub, Freund, Nachtigal, and others). Recent work by Trefethen in this area is represented by a pair of publications in v. 13 of the *SIAM J. of Matrix Analysis and Applications* ["How fast are nonsymmetric matrix iterations?", with Nachtigal and Reddy, and "A hybrid GMRES algorithm for nonsymmetric linear systems," with Nachtigal and Reichel.]

Together with Anne Greenbaum (New York University), André Weideman (Oregon State University), and others, Trefethen is currently involved with developing a general understanding of the convergence of matrix iterations for these large-scale nonsymmetric problems. The spectrum of the matrix is not enough to determine the convergence rate, nor enough to determine the effectiveness of preconditioners. Information about the norm of the resolvent, or equivalently the pseudospectra, may be more useful. A recent ACRI report by Greenbaum and Trefethen [CTC92TR92, "GMRES/CR and Arnoldi/Lanczos as matrix approximation problems," submitted to *SIAM J. Numer. Anal.*] provides the beginning of a theory of convergence by connecting nonsymmetric matrix iterations with approximation theory in a way that does not depend on the spectrum. Ongoing work with Weideman and others aims at making the connection precise between convergence and pseudospectra via the unifying concept of *lemniscates of iteration polynomials.*

The aim of this work is an improved understanding of how nonsymmetric iterative methods converge, and how preconditioners can be designed to make them converge faster.

2.3 Preconditioning Legendre Spectral Collocation Approximations to Elliptic Problems

Preliminary results of this joint work (with Professor Seymour Parter of the University of Wisconsin - Madison) were presented at the "Workshop on Advanced Scientific Computing in the Nineties", Tel Aviv, Israel, May 24-28, 1992, as an invited paper entitled, "Preconditioning of Elliptic Problems: Mixing up the Discretizations". Those results, essentially described in the last report, were limited to the one- dimensional case. Since then the work has been extended to two-dimensional problems and is nearing completion in the form of a paper, "Preconditioning Legendre Spectral Collocation Approximations to Elliptic Problems". Briefly put, we applied the theory of equivalent operators on Hilbert spaces to the numerical solution of elliptic partial differential equations. This resulted in iterative methods that are independent of the size of the discretization. Our basic problem is the solution of the linear system $Au = f$, where A is a uniformly elliptic operator that is invertible, but not necessarily definite. We approximate A by a family of spectral collocation discretization based on the Legendre-Gauss-Lobatto points which arises from a variational representation of the operator A. The actual solution of this linear system is difficult because the spectral approximation to A is ill-conditioned even if A is symmetric, positive-definite operator. We apply a preconditioned iterative approach using a finite-element discretization. Letting B be the stiffness matrix and M be the mass matrix of the finite-element method, we can precondition with $B^{-1}M$ using damped Jacobi, or GMRES, for example. These and similar iterative methods are effective when the eigenvalues of the preconditioned matrix $B^{-1}MA$ all have positive real parts. Since one usually uses a positive definite B, such an approach can be used only when A itself is positive definite. In the more general case, one could consider CG applied to normal equations. Another preconditioning is given by $B^{-1}WA$, where W is a diagonal matrix whose entries are the Legendre-Gauss-Lobatto quadrature weights. We gave a complete analysis of the B-singular values of this preconditioned matrix. We show that 'most' B-singular values associated with the general problem (A indefinite) cluster in a small interval (determined by preconditioning of A definite and $B = A$). Numerical experiments suggest that L_2 singular values behave similarly. In addition to submitting this paper for publication (probably in Applied Numerical Mathematics), a Theory Center Technical Report will be written as well.

2.4 Parallel QR Factorizations of Large Sparse Matrices on Distributed-Memory Multiprocessors

We have been investigating the issues involved in the design and implementation of an efficient parallel multifrontal algorithm for computing the QR factorization of a large sparse matrix on distributed- memory multiprocessors. Typically, a sparse QR factorization of an $m \times n$ ($m \geq n$) matrix A with full column rank involves three steps. First, determine a permutation matrix P such that the QR factorization of AP requires less arithmetic work and less storage than that of A. Second, obtain the symbolic structure of R. Third, perform the numerical factorization.

In a distributed-memory computing environment, the available parallelism needs to be carefully analyzed and exploited in addition to the sparsity consideration. This presents a formidable task because of the complexity of the data structures involved. The critical issue is how to partition the overall factorization into subtasks and map them onto processors such that load balance is maintained and low communication overhead is achieved.

Our strategy for exploiting parallelism is based on the concept of row merge trees introduced by as a means of reducing arithmetic work in a sparse QR factorization. The overall factorization is formulated as a sequence of merges of two dense upper trapezoidal matrices organized around the row merge tree. Our algorithm has the following novel features. First, a supernodal tree computed from the sparsity structure of R is used to organize the numerical factorization instead of the row merge tree. The supernodal structure of R is fully exploited to decrease stack operations and hence reduce data movement in numerical factorization. Second, the supernodal tree is mapped onto processors by a simple and yet effective proportional mapping scheme. Third, an efficient algorithm has been designed for the most crucial task in this context—the merging of two upper trapezoidal matrices of different sizes in parallel by Givens rotations. Fourth, the overall factorization is accomplished by a mixture of Householder and Givens transformations. Experimental results on an Intel iPSC/860 for a collection of problems with grid structures and random structures have been obtained. The initial results show that our algorithm is efficient in view of the running times and floating-point operations per second obtained. Further experiments are needed. Our algorithm will be compared with those reported in the literature. [21]. We have completed our work on parallel solutions of large sparse SPD systems on distributed-memory multiprocessors and our experience is summarized in. [34]

3 Computational Optimization

Coleman, Chinchalkar, Li, A. Liao, L. Liao, Ralph, Wu

3.1 Energy Minimization and Protein Folding

We have begun work with D. Shalloway (Biochemistry) to develop and implement parallel algorithms on the Intel iPSC/860 hypercube for solving the protein folding problem in molecular biology. The problem is important for the biological study of protein structure and function, but also computationally very challenging: it is NP-hard in general and so far most biologically significant problem instances (proteins with 1,000 to 10,000 atoms) remain unsolvable (in theory) due to the very large amount of computation usually required.

Two algorithms have been studied: the effective energy simulated annealing algorithm developed by our research group recently and the packet annealing algorithm proposed by Shalloway in 1991. The former algorithm is a stochastic approach, and the latter a deterministic method.

The effective energy simulated annealing algorithm combines the simulated annealing with a class of effective energy functions which are transformed from the original energy function based on the theory of renormalization group in statistical physics. The algorithm has been implemented on the Intel iPSC/860 hypercube, and numerical experiments have been conducted for the conformation of molecule clusters of up to 100 atoms interacting with the Lennard-Jones potential. The results show that the algorithm is effective and efficient for finding low energy states and hence stable structures of given molecule clusters, and it is much more efficient than a general simulated annealing method. A technical report on this work is just being completed [10]. We will continue to work on further improvement of the algorithm, and will apply it to some special proteins.

The packet annealing algorithm is a deterministic algorithm. Similar to the simulated annealing, the algorithm searches for energy minima of given molecule system at different temperatures. But different from the simulated annealing, the search is done deterministically by first constructing the approximation to the energy distribution, and then locating the conformation regions with high probability densities. To obtain the approximation, multiple Gaussian distribution functions, called Gaussian packets, need to be constructed, which, computationally, requires solving a number of large nonlinear systems of equations (or nonlinear local minimization subproblems). For the parallel implementation of this algorithm, we have made progress on:

1. parallelizing energy function evaluation;

2. computing pseudo-inverses of large dense matrices efficiently, which is required in constructing multiple Gaussian packets;

3. developing feasible procedures for solving the large nonlinear systems of equations (most of them are too large to apply standard techniques);

4. developing a small software system for parallel algebraic operations basic to high level parallel implementation.

The small software system has been completed. The system contains basic routines for parallel algebraic operations such as matrix factorization, triangular system solution, matrix-matrix, matrix-vector and vector-vector operations, etc. A users' guide for the system has been completed as a Theory Center technical report [38].

A presentation titled "a new computational approach to the protein folding problem" was given at the Fourth SIAM conference on optimization, May 1992. We have been invited to participate in a mini-symposium on computational methods for protein-folding problems at the TIMS/ORSA meeting in May 1993 on "isotropic effective energy simulated annealing for the search of low energy states of molecule clusters"

3.2 Parallel Optimal (Discrete-Time) Control

Optimal control problems arise in many engineering areas such as aerospace, electrical, environmental and structural engineering. Due to the special structure in optimal control problems, several optimal control algorithms have been developed to take advantage of this special structure. Among these algorithms, Differential Dynamic Programming (DDP) method [23] has been considered the best one.

In our recent numerical study with [20] Christine Shoemaker (Civil & Environmental Engineering and Applied Mathematics Center), DDP has achieved 24 times faster than a stagewise Newton's method [11, 27]. The detailed anlysis [20] indicates that the DDP algorithm is numerically superior to the standard optimization algorithms for optimal control problems. The results were presented in the Fourth SIAM Conference on Optimization in May, 1992.

The success of DDP lies in the efficient exploration of optimal control structure. Unfortunately DDP represents a fine-grain algorithm in the parallel processing point of view. The fine-grain structure in DDP prevents efficient massive parallel computation, especially across different time periods. Based on this fact, a new 'Parallel Clustered Dynamic Programming' algorithm has been developed for the parallel environment. The Parallel Clustered Dynamic Programming algorithm is a combination of two efficient serial optimal control algorithms, DDP and stagewise Newton's method. The current serial numerical results from the Parallel Clustered Dynamic Programming are quite encouraging compared to the numerical results from serial DDP. The parallel implementation of this Parallel Clustered Dynamic Programming is in process. The target multiprocessor computer for this algorithm is an Intel iPSC/860 at Cornell Theory Center with 32 nodes.

We have also designed and implemented a stagewise decomposition method [31], a non-linear extension of [3], in which each processor is given a different group of consecutive stages of the problem, and iterations are performed using efficient recurrences by each processor on its data. Communication between processors is needed to update separator variables — which link a group of stages to the previous group — the following group — and retain quadratic local convergence.

This parallel method has been tested [31] on a 32 node Intel iPSC/860 hypercube, a message passing computer. Our preliminary results are amongst the best in the field: on medium to large sized problems we routinely obtain parallel efficiencies (comparing the parallel code to the 'best' serial code) of above 70% while, with the exception of [3], other methods [29, 36, 37] have trouble approaching efficiencies of 60%. The question of handling inequality constraints, for which sequential quadratic programming rather than Newton's method is the building block, is the subject of current research.

3.3 Parallel Structural Optimization

We are working on problems in structural optimization that involve minimization of a nonlinear function with simple bounds on the independent variables. The nonlinear function being minimized is a function of the finite element response (displacements/stresses) and the independent variables are the densities or Young's moduli in individual finite elements. Problems in bone remodeling can be formulated in the above form, where the function being minimized is a linear combination of the mass of the bone and the strain energy in the bone and the independent variables are the densities in individual finite elements. It has been conjectured that the changes in density and stiffness of bone can be predicted by solving the above minimization problem [33].

Structural optimization problems of the above type involve thousands of independent variables and require a finite element analysis for each evaluation of the objective function. Parallel code for fem had been developed earlier; code for parallel gradient computation was written. Three approaches were used for optimization:

- an active-set method with quasi-Newton updates using a commercial code, "NPSOL". This method used an active set method for the QP subproblem. The QP subproblem was solved on a SUN workstation while the objective function evaluation and gradient computation was performed in parallel on the Intel iPSC/860 hypercube.
- a reflective Newton method with quasi Newton updates. The IPSC/860 was used to solve the linear systems in parallel.
- a reflective Newton method with finite difference Hessian.

The code developed was used to solve bone remodeling problems, which have been formulated as optimization problems [8].

Future work will involve using reflective Newton methods with analytical calculation of the Hessian, with the possibility of parallelization.

3.4 Minimization with Bounds on The Variables

We have been developing a new approach to bound-constrained optimization problems [8, 7] – the new method is called a reflective Newton method. This approach is particularly suitable for large-scale problems and parallel computation. We presented this work at a recent SIAM meeting on optimization in May (Chicago); two reports are now in preparation. This work represents a new way to deal with box constraints and appears very promising from a practical computational point of view. This approach builds upon previous work of Coleman and Li on various convex optimization problems and on the work of Coleman and Hulbert for convex quadratic minimization subject to bounds (recently accepted for publication).

However, the new reflective Newton approach is not restricted to convex problems and does not require that every variable have finite upper and lower bounds.

In collaboration with Stephen B. Pope (Department of Mechanical & Aerospace Engineering, Cornell University), we have been investigating the issues involved in implementing a particle method for modeling turbulent combustion on distributed-memory multiprocessors. A central computation in the particle method is the solution of very large and sparse box-constrained least squares problems. We have developed a parallel adaptation of our reflective Newton method for this important special case. Experimental results for a collection of sparse box-constrained least squares problems from the particle method with over 10,000 variables have been obtained on an iPSC/860 machine with 32 nodes. Our results were presented in the SIAM 40th Anniversary Meeting in Los Angeles from July 20, 1992 to July 24, 1992.

With PhD student Jianguo Liu we are looking at extensions of this work to include linear equality constraints. In addition, with PhD student Mary Ann Branch we are researching a fully iterative and parallel variant, suitable for very large problems.

To contrast with our reflective Newton methods, we are also developing efficient interior point algorithms for linearly constrained problems with bounds. We are investigating how to combine the interior point method with the SQP method and/or trust region method and we will compare this approach, empirically, to our new method.

4 Numerical Methods for Partial Differential Equations

Rothman, Trefethen, Vavasis

4.1 Hydrodynamic instability and transition to turbulence

One of the fundamental questions of fluid mechanics is how incompressible fluid flows become unstable and eventually turbulent. Not only turbulence but even the simpler problem of instability is far from fully understood at present, but this is a field undergoing rapid change. In the past two years it has become clear that *linear but non-modal* effects (i.e., unrelated to eigenvalues) are at the heart of instability for certain fundamental fluid flows such as flow of water through a pipe. These effects had heretofore gone unrecognized essentially because of widespread misunderstanding of the physical significance of eigenvalues for highly non-self-adjoint problems.

The key people involved in these new developments have been Butler and Farrell at Harvard, Reddy at New York University, Henningson and Schmid and Ioannou at MIT, and Trefethen at Cornell together with his collaborators Anne Trefethen (Cornell Theory Center) and Toby Driscoll (PhD student in the Center for Applied Mathematics). A recent Cornell Computer Science department technical report by Trefethen, Trefethen, and Reddy [TR 92-1291, "Pseudospectra of the linear Navier-Stokes evolution operator and instability of plane Poiseuille and Couette flows,"] presents many of these effects and posits a mechanism for 'bootstrapping' linear, non-modal energy growth into the ultimately nonlinear phenomenon of turbulence. This work is developing rapidly and a paper by Trefethen, Trefethen, Reddy, and Driscoll is in preparation to be submitted to *Science:* “A new direct method in hydrodynamic stability: beyond eigenvalues.”

In September, 1992, L.N. and A.E. Trefethen ran a three-day workshop sponsored by the Theory Center at which all of the people listed above except Henningson were present. One of the effects of this workshop has been to focus our work for the upcoming months especially on two topics: (a) new explanations of and alternatives to the theory of ‘secondary instability’ of such flows, and (b) attempts to pin down the nonlinear details so as to develop a more complete model of transition to turbulence. Both of these problems require large computations, including direct numerical simulations of the Navier- Stokes equations in three dimensions by means of spectral methods. Up to now we have used workstations, Connection Machines, and IBM RS/6000s for this work. In the upcoming months we expect it to expand and expect to make more extensive use of the RS/6000s and perhaps the IBM System 9000.

Trefethen, Trefethen, Reddy and Driscoll expect to be heavily involved in these hydrodynamics computations in the upcoming year.

4.2 Turbulent Boundary Layers

This work is in collaboration with Professor Lumley and his research group in Mechanical and Aerospace Engineering (Cornell). The general topic here is the control of the turbulent boundary layer. We envision a grid of sensors on a wall, and software which can locate the fluid dynamical system in a suitable phase space, based on the input from the sensors. We have a low-dimensional dynamical-systems model of the fluid flow in the wall region, and envision using this to make short term projections of the system motion in the phase space. The system is known to make jumps along heteroclinic cycles from one fixed point to another, and then back. We wish to delay these jumps, keeping the system in the vicinity of one of the fixed points, and are developing a control strategy to do this. We envision a grid of actuators, which raise Gaussian bumps on the wall. Current computations involve simulating the interaction of these with the flow. Our current efforts are focused on modifying and extending a method developed by J. Kim, P. Moin, and R. Moser using spectral methods and to investigate the parallel implementation of these methods.

4.3 Wavelets, Mesh Generation

We continued work in several areas related to numerical solution of partial differential equations. PhD student Scott Mitchell presented our joint work on mesh generation in three dimensions at the ACM Geometry conference this past summer. A full version of the paper has been submitted to *SIAM J. Computing.* Scott’s recent work involves mesh generation in the case that new nodes may not be introduced on the boundary of the region (only the interior). This occurs in modeling physical problems where boundary nodes must occur at prespecified locations corresponding, for instance, to sensors /actuators. It also applies to the problem of remeshing part of a domain which case the boundary must match up to the boundary nodes of the existing mesh.

Scott has accepted a postdoctoral position at Sandia National Laboratories starting in November.

PhD student Dave Bond presented our joint work on wavelets and multigrid for boundary element methods at the 1992 Copper Mountain Meeting on iterative methods, and a preliminary version of the work appeared in the proceedings. We are currently analyzing in more depth the use of wavelets in the process to asymptotically speed up iterative methods. The main paper so far in this area is by Beylkin, Coifman and Rokhlin: our work aims to show that the wavelet transform can be applied as a 'black box' to a variety of boundary element problems without introducing special wavelet-based shape functions or quadrature rules.

5 Other Accomplishments

- We have identified some new level-3 BLAS with graduate student Greg Henry. These BLAS have led to improved implementations of the unsymmetric QR algorithm for eigenvalues and related eigenvector calculations.
- We developed a system called IPSC-MATLAB [4], designed to provide a flexible environment for parallel programming. IPSC-MATLAB uses a SUN workstation as a front-end running MATLAB and the Intel iPSC/860 hypercube as a back-end performing numerically intensive computations. This system can be used to run existing matlab programs on large problems on the Intel iPSC/860 hypercube.
- We have written up and submitted a work on stable numerical algorithms for equilibrium systems. These are linear systems arising in optimization, electrical engineering, civil engineering, and the method of finite elements. The work shows that even when the system is severely ill-conditioned, which can happen in many applications, it is still possible to recover an accurate solution if a certain algorithm we have proposed is used.
- We have written up work on mesh partitioning, joint with Miller, Teng and Thurston, to appear in a book of proceedings of the IMA. This paper is being reviewed.
- We also produced a large-scale optimization research overview [6] now accepted for publication in Encyclopedia of Computer Science and Technology (Marcel Dekker)
- An interesting convergence result [9] was established concerning quasi-Newton methods for constrained problems and this short paper will soon appear in Applied Mathematics Letters.

ACRI Technical Reports, April - September 1992

IPSC-MATLAB Reference Manual. Shirish Chinchalkar, Advanced Computing Research Institute, Cornell Theory Center, Cornell University, CTC92TR**106**, September 1992.

Stable Numerical Algorithms for Equilibrium Systems. Stephen A. Vavasis, Advanced Computing Research Institute, Cornell Theory Center, Cornell University, CTC92-TR**105**, September 1992.

Quality Mesh Generation in Three Dimensions. Scott A. Mitchell and Stephen A. Vavasis, Advanced Computing Research Institute, Cornell Theory Center, Cornell University, CTC92TR**104**, September 1992.

Efficient Parallel Solutions of Large Sparse SPD Systems on Distributed-Memory Multiprocessors. Chunguang Sun, Advanced Computing Research Institute, Cornell Theory Center, Cornell University, CTC92TR**102**, August 1992.

Increasing Data Reuse in the Unsymmetric QR Algorithm. Greg Henry, Advanced Computing Research Institute, Cornell Theory Center, Cornell University, CTC92TR**100**, July 1992.

Access Normalization: Loop Restructuring for NUMA Compilers. Wei Li and Keshav Pingali, Advanced Computing Research Institute, Cornell Theory Center, Cornell University, CTC92TR**99**, July 1992.

A Singular Loop Transformation Framework Based on Non-singular Matrices. Wei Li and Keshav Pingali, Advanced Computing Research Institute, Cornell Theory Center, Cornell University, CTC92TR**98**, July 1992.

Advantages of Differential Dynamic Programming Over Newton's Method for Discrete-time Optimal Control Problems. Li-zhi Liao, Advanced Computing Research Institute, Cornell Theory Center, Cornell University, CTC92TR**97**, July 1992.

GMRES/CR and Arnoldi/Lanczos as Matrix Approximation Problems. Anne Greenbaum, Courant Institute of Mathematical Sciences; and Lloyd N. Trefethen, Computer Science Department and Advanced Computing Research Institute, Cornell Theory Center, Cornell University, CTC92TR**96**, June 1992.

A Parallel Row Distributed Linear Algebra System (PRDLA User's Guide). Zhijun Wu and Thomas F. Coleman, Advanced Computing Research Institute, Cornell Theory Center, Cornell University, CTC92TR**93**, April 1992.

ACRI Seminar Schedule, April - September 1992

Date	Invited Speaker and Topic of Talk
April 6	Professor Nicholas H. Higham Department of Mathematics University of Manchester, Manchester, England Title: Block Matrix Algorithms: Stable or Unstable?
April 20	Professor Gilbert Strang Institute of Information Processing Department of Mathematics Massachusetts Institute of Technology Title: Wavelet Transforms Versus Fourier Transforms
September 7	Professor Desmond J. Higham Department of Mathematics & Computer Science University of Dundee, Scotland Title: Sensitivity Issues in Numerical Linear Algebra
September 14*	Dr. Li-zhi Liao Advanced Computing Research Institute Cornell University Title: A Parallel Processing Algorithm for discrete Time Optimal Control Problems
September 21*	Professor Robert G. Bland Operations Research Industrial Engineering Cornell University Title: An Empirical Study of Minimum Cost Flow Algorithms
September 28*	Dr. Daniel Ripoll CSERG - Cornell Theory Center Cornell University Title: A Parallel Monte Carlo Search Algorithm for the Conformational Analysis of Polypeptides

*** Joint seminars with Cornell Computational Optimization Project**

ACRI Researchers, September 1992.

Name	Position	Research Interests
S. Chinchalkar	Postdoctoral Associate	Finite-element Analysis and Parallel Computation
T. Coleman	Professor, Computer Science Director, ACRI	Numerical Optimization, Parallelism for Scientific Computation
Y. Li	Research Associate	Numerical Optimization
A.P. Liao	Postdoctoral Associate	Optimization, Interior Point Methods
L. Liao	Postdoctoral Associate	Numerical Optimization, Optimal Control, Parallel Computation
K. Pingali	Assistant Professor, Computer Science	Parallelizing Compilers
D. Ralph	Postdoctoral Associate	Numerical and Parallel Optimization
E. Rothman	Research Associate	Numerical Partial Differential Equations
C. Sun	Postdoctoral Associate	Large-scale Linear Algebra and Parallelism
N. Trefethen	Associate Professor, Computer Science	Numerical Linear Algebra and Partial Differential Equations
C. Van Loan	Professor, Computer Science	Numerical Linear Algebra, Parallelism
S. Vavasis	Assistant Professor, Computer Science	Optimization, Parallelism, Boundary Element Methods
Z. Wu	Postdoctoral Associate	Numerical Optimization, Global Minimization and Parallelism

Associated with the ACRI are approximately 15 PhD students, primarily from the fields of computer science and applied mathematics.

Part II
Highlight Section

In this section we introduce one of the projects, parallelizing compilers, in more detail. We sketch the problem and our approach.

Compiling for NUMA Parallel Machines

Wei Li
Keshav Pingali

1. Motivation

We are building a system for programming NUMA machines that takes programs written in FORTRAN-77 extended with data distribution information and generates code for parallel machines such as the KSR1 and the BBN Butterfly. There is a wide range of applications that may find parallel machines useful. They come from areas such as linear programming, matrix algebra, computational physic, financial modeling, weather and climate modeling, electro-magnetic fields simulation, and numerical aerodynamic simulation [5, 12, 26].

In general, the effective use of parallel machines requires careful attention to the following four issues.

- Exposing and exploiting parallelism: Unless the algorithm underlying a program has lots of parallelism, it is pointless to run the code on a parallel machine. Moreover, the compiler must be able to expose and exploit this parallelism.
- Locality: Scalable high-performance machines are built as interconnections of processor-memory pairs in which a processor can access its local memory ten to a thousand times faster than non-local memory. Therefore, proper data management to maximize local accesses is essential.
- Block transfers: Communication between processors can be viewed as a pipelined process in which the start-up time is large compared to the time to transfer a unit of data. Therefore, when non-local accesses must be made, it is more efficient to use a single block transfer of data rather than many individual transfers.
- Load balancing: It is important to avoid swamping a few processors with work when other processors are idle.

Pnuma is a system we are building that will permit scientific application programmers to express their algorithms in a high-level language like FORTRAN-D and exploit massive parallelism on parallel machines like the KSR1 and the BBN Butterfly.

2. *Pnuma* System Overview

Our project at Cornell and the Parascope project at Rice, among others, have taken the approach of using a language like FORTRAN-D in which the programmer has control over how arrays are mapped to the local memories of the machine [32, 2, 16, 39, 15]. The data distributions commonly used by programmers of NUMA machines are *wrapped* and *blocked* row and column mappings among others. In *wrapped column* mapping, columns of an array are distributed in a round-robin way to the processors: if P is the number of processors, processor 0 gets columns *0, P, 2P* and so on, while processor 1 gets columns *1, P+1, 2P+1,* etc. Blocked mappings are similar, except that processors get contiguous columns or rows.

In the traditional approach to parallelization (pioneered by the Cedar project at Illinois), iterations of the loops in a loop nest are distributed among the processors. Synchronization instructions are introduced to take care of dependencies carried by this loop; to reduce the amount of synchronization, transformations like loop interchange are performed to move parallel loops outermost [24, 28]. This 'outside-in' strategy is simple but it does not perform any data management, and may result in many non-local accesses during the execution of the loop nest.

A different approach to compiling is to generate code 'inside-out' using the so-called ownership rule — the owner of the left-hand side variable of an assignment statement is responsible for computing the right-hand side expression. A processor executes a loop iteration if it has any work to do in the body for that iteration. Although this strategy takes data mappings into account, code generation is very complex, compared to the traditional approach, and the code generated can be very inefficient if the structure of the loop nest does not match the data distribution [32, 13].

Our project has made a major advance recently in overcoming these problems. We use data distribution information to drive a loop transformation strategy called *data access normalization.* Once a loop nest has been transformed by data access normalization, we can generate code simply by distributing outermost loop iterations among the processors. Data accesses in the resulting code are local wherever possible, and non-local accesses are performed using block transfers if possible. Our preliminary experiments show that the resulting code is superior to that produced by the two extant approaches discussed above.

Program transformations, such as loop restructuring, are critical to achieving high performance. We have developed a framework based on *non-singular* matrices and integer lattice theory for the systematic development of loop transformations [19]. The main benefit of this framework is that it provides an approach to tackling the so-called 'phase-ordering problem' — for many problems where there is no obvious order in which the transformations should be performed, it is often possible to generate a non-singular matrix from which the desired order of loop transformations can be determined easily. This framework can be used in parallelizing compilers for MIMD machines as well as in compilers for fine-grain parallel architectures such as VLIW and superscalar machines. We have implemented a loop restructuring tool-kit called *Lambda* based on this framework.

We are using this loop transformation framework in *Pnuma* which is a parallelizing compiler for NUMA machines being developed at Cornell. *Pnuma* incorporates a novel loop restructuring strategy called *access normalization* which restructures loop nests to expose

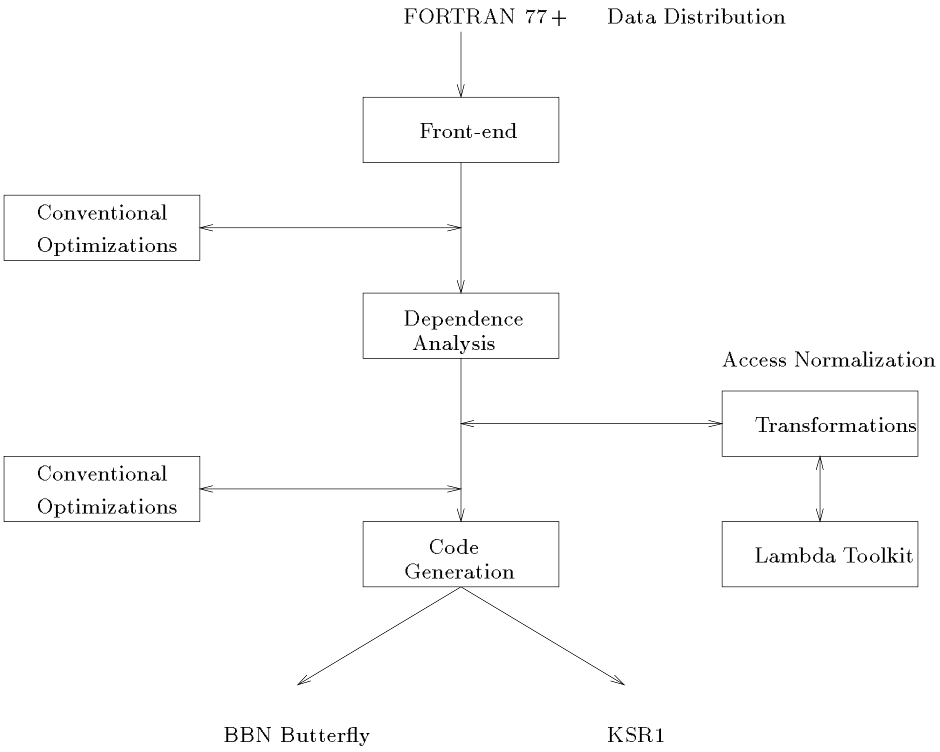


Figure 1: *Pnuma* System Overview

opportunities for parallel execution and for block transfers, while keeping data accesses local wherever possible [18]. Loop restructuring is followed by a code generation phase that generates parallel code and makes use of block transfers [17].

Compiling for distributed memory machines is the goal of a number of projects [2, 13, 39, 32, 16, 15, 25, 14]. Existing work focuses on code generation techniques, not on loop restructuring, although the importance of loop restructuring in generating good code for vector and parallel machines is widely recognized. Recent work on generalized loop transformations can be found in [1, 35, 22].

The system we are building takes programs written in FORTRAN-D, and generate code for the KSR1 and the BBN Butterfly. To avoid duplication of effort, we are using the SIGMA front-end from Dennis Gannon's group at Indiana. SIGMA takes FORTRAN-D and C as input and builds abstract syntax trees and symbol tables; these data structures are the input to our system. Figure 1 shows the individual modules in the system.

- *Front-end*: A front-end from SIGMA is used to generate abstract syntax trees from FORTRAN-77 programs. We have adopted the data distribution syntax of FORTRAN-D and we are augmenting our front-end to handle this information.

- *Optimizations*: It is beneficial to perform constant propagation and strength increasing before doing dependence analysis. (Strength increasing is a transformation that looks for loop variables assigned values using linear recurrences and converts them into functions of the loop index.) Performing these transformations improves the quality of the dependence information extracted by the dependence analyzer.

- *Dependence analyzer*:

  To give as much latitude as possible for loop transformations, it is important to have precise data dependence information. Our current dependence analyzer uses the classical GCD test and Banerjee's test to perform dependence analysis of array subscripts. These are the most commonly used tests, but they have short-comings, especially when loop bounds are functions of loop indices. Recently, a number of more sophisticated dependence tests such as the Omega test have been developed [30]. The Omega test uses a fast integer programming problem solver to solve systems of Diophantine equations for solutions which lie in specific regions of the possible solution space, which is the essential idea behind dependence analysis.

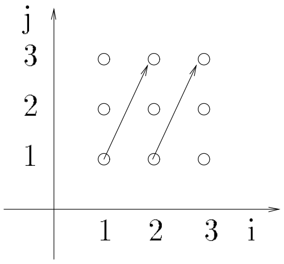
- *Transformations to enhance locality and block transfers*: *Access normalization* matches code and data distributions to exploit both parallelism and locality. As an added bonus, access normalization exposes opportunities for block transfers of data.

- *Lambda Toolkit*: Although loop transformations such as permutation and skewing are useful for generating good code for parallel machines, it is usually difficult to determine the best order in which to perform these transformations. We have developed a powerful loop transformation framework that uses *non-singular matrices* as its foundation; this framework subsumes existing ones and solves the problem of deciding the best order to apply loop transformations.

- *Back-ends*: We have implemented a back-end that generates parallel C code for the BBN Butterfly, and we are generating parallel FORTRAN for the KSR1.

3. *Lambda* Loop Transformation Theory

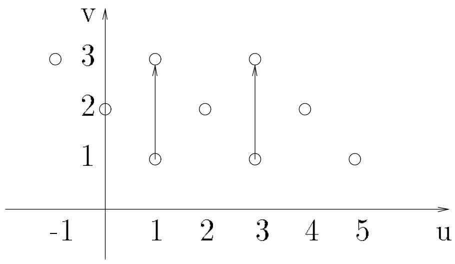
This transformation theory is based on the use of *integer lattices* as the model of loop nests and the use of *non-singular matrices* as the model of loop transformations. In this section, we give a sketch of our approach. Consider the iteration space in Figure 2(a), which is derived from a loop nest where i is the outer loop and j is the inner loop. The outer loop is not parallel because of the data dependence $\begin{pmatrix} 1 \\ 2 \end{pmatrix}$. For coarse-grain parallelism, it is preferable to have a parallel outer loop, i.e. an outer loop that does not carry data dependencies. This can be accomplished by the loop transformation shown in Figure 2. The iteration space in Figure 2(c) is the transformed iteration space; in this iteration space, loop u is a parallel loop. It is possible to obtain this transformation by a sequence of simple loop transformations like skewing and interchange, but it is non-trivial to determine the order in which these transformations must be applied to achieve the desired effect. The matrix-based approach permits the generation of the transformation matrix from the dependencies, and thereby produces the composite transformation directly.



(a) Iteration Space

$$\begin{pmatrix} 2 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} u \\ v \end{pmatrix}$$

(b) Transformation



(c) New Iteration Space

Figure 2: Loop transformation for coarse-grain parallelism

Given a loop nest and a non-singular matrix that represents a legal transformation of the loop nest, it is non-trivial to generate the code for the transformed program. Fortunately, we have shown that the Hermite normal form of the transformation matrix can be used to solve this problem. The details of code generation may be found in a related paper [19]. This paper also gives a *completion procedure* which, given some initial rows of a desired transformation matrix, produces a complete transformation matrix that represents a legal restructuring of the original loop nest.

This toolkit can be used in a number of applications such as restructuring for machines exploiting coarse-grain or fine-grain parallelism, for exploiting spatial and temporal data locality and for enhancing data reuse. The next section describes the use of this framework for exploiting data locality.

4. Access Normalization

Access normalization is a loop restructuring technique to exploit data locality between local memory and non-local memory in distributed memory MIMD machines. It changes the data access patterns so that the parallelized code can exploit data locality. To understand the need for loop restructuring in NUMA compilers, consider the program in Figure 3(a), which is a simplified version of the banded SYR2K code. Assume that both A and B have a wrapped column distribution. Distributing iterations of the outer loop among the processors results in processor p executing iterations p, $p + P$, etc. Consider accesses to elements of array B. Each iteration of the outer loop makes $N_2(b - b/P)$ non-local accesses, and the total number of non-local accesses is $N_1 N_2 b(1 - 1/P)$.

Access normalization transforms the program of Figure 3(a) to that of Figure 3(b) by deriving a legal loop transformations from the data access patterns in the program. The key data structure is the *Data Access Matrix* (Figure 3(c)) that captures the access information from the program, more specifically the subscripts from the array references. The product of each row and the column vector $(i, j, k)^T$ is a subscript from the program. A transformation matrix is constructed by taking rows from the data access matrix. In this example, the

for i = *0,* $N_1 - 1$
  for j = *i, i+b-1*
    for k = *0,* $N_2 - 1$
      B[i, j-i] = B[i, j-i] + A[i, j+k]

(a) Original Program

for u = *0, b-1*
  for $v = u,\ u + N_1 + N_2 - 2$
    for w = *0,* $N_1 - 1$
      B[w, u] = B[w, u] + A[w, v]

(b) Transformed Program

$$\begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

(c) Data Access Matrix

$$\begin{pmatrix} -1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

(d) Transformation Matrix

Figure 3: Access Normalization

transformation matrix in Figure 3(d) is just a permutation of the rows from the data access matrix. The transformation matrix needs to be both non-singular and satisfying the data dependencies. In general, the data access matrix is not qualified as the transformation matrix. The algorithm for deriving the transformation matrix from the data access matrix is presented in [18].

Consider the example again. If we distribute the outermost loop among the processors as before, there are no non-local accesses to B, since the outermost loop index is also the the subscript of the distribution dimension of array B. We cannot accomplish this for both A and B simultaneously since the subscripts in the distribution dimensions of A and B are different; therefore, there will be non-local accesses to A. However, since the subscript in the distribution dimension of the reference to A is the second loop index, we can perform block transfers for accesses to A.

5. Localizing and Amortizing Data Accesses

After the program is transformed by access normalization, we must generate the code that will run on each processor. We discuss the special case of wrapped and blocked data distributions. For these distributions, it can be shown that it is sufficient to distribute the iterations of the outermost loop of the transformed loop nest that carries the data distribution among the processors. In access normalization, the data distribution function that appeared the most in the data references was chosen to be "normalized" to be the outermost loop index. Then by mapping the iterations to the processors according to the data distribution function, an iteration should be executed by a processor if the corresponding data element is mapped to its local memory.

For lack of space, we only consider the case when the step size is 1. For a wrapped

(a) unit step	(b) task p for wrapped distribution	(c) task p for blocked distribution
for $i = l,\ u$	*for* $i = \lceil \frac{l-p}{P} \rceil * P + p,\ u,$ *step* P	*for* $i = max(l,\ p * S),$ $min(u,\ (p+1) * S - 1)$

Figure 4: Distributing loops among processors

distribution, processor p owns the *data segments* $p,\ p + P,\ p + 2P,$ .. etc, where a data segment is a column in the wrapped column distribution or a row in the wrapped row distribution. Since the iterations that access the data segments on processor p are assigned to processor p, it is easy to verify that the iterations executed by processor p are the ones shown in Figure 4(b). The lower bound $\lceil \frac{l-p}{P} \rceil * P + p$ is the first iteration between l and u that belongs to process p. For a blocked mapping, the corresponding iterations are shown in Figure 4(c).

There are three basic techniques to reduce the overhead of remote accesses: block pre-fetch, block post-store and data reuse. Since the overhead of remote data accesses is much higher with respect to the data size, it is profitable to "bundle" the remote data accesses into a single block data transfer. For data reads, it translates to block pre-fetch, and for data writes, it translates to post-store. Both block pre-fetching and block post-store can be used to overlap communications and computation. Furthermore, remote data access is unnecessary if a local copy of the remote data exists. The reader who is interested in the details of these mechanisms is referred to the forthcoming thesis of one of the authors [17].

6. Experiments

We have compiled routines from the BLAS (Basic Linear Algebra Subprograms) library. The target machine is a BBN Butterfly GP-1000. For routines that can be localized successfully, the speedup increased mostly from localizing data accesses. For example, speedup for GEMM raised from 5 (with no transformation) to 12 (with localizing) and to 15 (with both localizing and block transfers) on 20 processors. For routines that can not be localized completely, block transfers play a major role. For example, the speedup for banded SYR2K increased from 6 (with no transformations) to 7 (with localizing) and to 12 (with both localizing and block transfers). Detailed experimental results can be found in [18].

General matrix multiplication (GEMM) is one of the central subroutines in BLAS. The sequential code for this routine is shown in Figure 5(a). All arrays are of size 400 by 400 and are distributed in wrapped column manner. By distributing the outermost loop among the processors without doing any transformations, we obtain the graph labeled *gemm* in Figure 5(i).

The data access matrix and dependence matrix produced by our system are shown in Figure 5(e) and (f). The non-singular matrix for the transformation is shown in Figure 5(g).

```
for i = 1, N
  for j = 1, N
    for k = 1, N
      C[i, j] = C[i, j] + A[i, k] * B[k, j]
```

(a) The original code

```
for u = 1, N
  for v = 1, N
    for w = 1, N
      C[w, u] = C[w, u] + A[w, v] * B[v, u]
```

(b) The transformed code

```
for u = p, N, step P
  for v = 1, N
    for w = 1, N
      C[w, u] = C[w, u] + A[w, v] * B[v, u]
```

(c) The parallel code for node p

```
for u = p, N, step P
  for v = 1, N
    read A[*, v];
    for w = 1, N
      C[w, u] = C[w, u] + A[w, v] * B[v, u]
```

(d) The parallel code with block transfer

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(e) Access Matrix

$$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

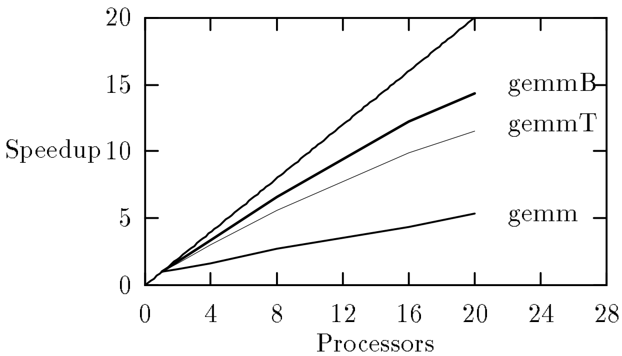
(f) Dependences

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

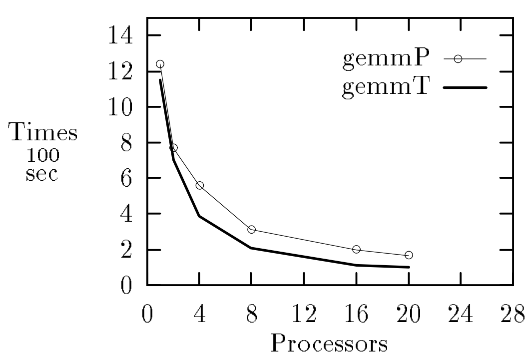
(g) Transformation

Nodes/Prog	gemm	gemmT	gemmB
1	1,245.2	1,150.0	1,117.8
2	1,072.5	704.3	630.6
4	784.4	387.5	328.1
8	462.5	206.2	169.3
16	283.1	116.8	92.5
20	230.6	100.0	78.7

(h) Execution Times (sec) of 400 × 400 GEMM



(i) Speedup



(j) Projected times

Figure 5: GEMM

The transformed loop nest of Figure 5(b) yields the code of Figure 5(c), and the corresponding execution times and speed-ups are labeled *gemmT* in Figure 5(h,i). Inserting block transfers for accesses to A, we get the code shown in Figure 5(d), and the performance of this program is labeled *gemmB* in Figure 5(h,i).

References

[1] U. Banerjee. Unimodular transformations of double loops. In *Proceedings of the Workshop on Advances in Languages and Compilers for Parallel Processing*, pages 192–219, August 1990.

[2] D. Callahan and K. Kennedy. Compiling programs for distributed memory multiprocessors. *The Journal of Supercomputing*, 2(2), October 1988.

[3] S.C. Chang, T.S. Chang, and P.B. Luh. A hierarchical decomposition for large-scale optimal control problems with parallel processing structure. *Automatica*, 25:77–86, 1989.

[4] Shirish Chinchalkar. Ipsc-matlab reference manual. Technical Report CTC92TR106, Advanced Computing Research Institute, Theory Center, Cornell University, September 1992.

[5] T. Coleman and C. Van Loan. *Handbook for Matrix Computations*. SIAM Publication, Phil, 1988.

[6] Thomas F. Coleman. Large-scale numerical optimization: Introduction and overview. Technical Report TR91-1236, Computer Science Department, Cornell University, September 1991.

[7] Thomas F. Coleman and Yuying Li. On the convergence of reflective newton methods for large-scale nonlinear minimization subject to bounds. In preparation. Advanced Computing Research Institute, Theory Center, Cornell University, 1992.

[8] Thomas F. Coleman and Yuying Li. A reflective newton method for minimizing a general quadratic function subject to bounds on variables. In preparation. Advanced Computing Research Institute, Theory Center, Cornell University, 1992.

[9] Thomas F. Coleman and A-Ping Liao. On the local convergence of the byrd-schnabel algorithm for constrained optimization. Technical Report TR91-1268, Computer Science Department, Cornell University, February 1992.

[10] Thomas F. Coleman, David Shalloway, and Zhijun Wu. Isotropic effective energy simulated annealing for searches for low energy molecular cluster states. in preparation, 1992.

[11] J. Dunn and D.P. Bertsekas. Efficient dynamic programming implementations of newton's method for unconstrained optimal control problems. *JOTA*, 63:23–38, 1989.

[12] K. Ekanadham and Arvind. SIMPLE Part I: An exercise in future scientific programming. Technical Report RC12686, IBM, 1987.

[13] S. Hiranandani, K. Kennedy, and C. Tseng. Compiler optimizations for FORTRAN-D on MIMD distributed-memory machines. Technical Report TR91-156, Rice University, April 1991.

[14] K. Ikudome, G. Fox, A. Kolawa, and J. Flower. An automatic and symbolic parallelization system for distributed memory parallel computers. In *Proc. of the 5th Distributed Memory Comp. Conf.*, April 1990.

[15] C. Koelbel and P. Mehrotra. Compiling global namespace parallel loops for distributed execution. *IEEE Transactions on Parallel and Distributed Systems*, 2, October 1991.

[16] J. Li and M. Chen. Compiling communication efficient program for massively parallel machines. *IEEE Transactions on Parallel and Distributed Systems*, 2:361–376, July 1991.

[17] W. Li. *Compiling for Parallel Machines with Memory Hierarchies*. PhD thesis, Dept. of Computer Science, Cornell University, August 1993. (expected).

[18] W. Li and K. Pingali. Access Normalization: Loop restructuring for NUMA compilers. In *Proc. 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1992.

[19] W. Li and K. Pingali. A singular loop transformation framework based on non-singular matrices. In U. Banerjee, D. Gelernter, A. Nicolau, and D. Padua, editors, *Languages and Compilers for Parallelism*. Springer-Verlag, 1993.

[20] L-Z. Liao and C. A. Shoemaker. Advantages of differential dynamic programming over newton's method for discrete-time optimal control problems. Technical Report CTC92TR97, Advanced Computing Research Institute, Theory Center, Cornell University, July 1992.

[21] J.W-H. Liu. On general row merging schemes for sparse givens transformations. *SIAM - J. Sci. Stat. Comput.*, 7:1190–1211, 1986.

[22] L. Lu. A unified framework for systematic loop transformations. In *3rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 28–38, April 1991.

[23] D. Mayne. A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems. *Intnl. J. Control*, 3:85–95, 1966.

[24] S. P. Midkiff and D. A. Padua. Compiler algorithms for synchronization. *IEEE Transactions on computers*, C-36:1485–1495, December 1987.

[25] R. Mirchandaney, J. Saltz, R. Smith, D. Nicol, and K. Crowley. Principles of runtime support for parallel processors. In *Proc. of the 2nd Int. Conf. on Supercomputing*, July 1988.

[26] A. Mohamed, G. Fox, G. Laszewski, M. Parashar, T. Haupt, K. Mills, Y. Li, N. Lin, and N. Yeh. Applications benchmark set for Fortran-D and High Performance Fortran. Technical Report SCCS 327, Syracuse University, 1992.

[27] J.F.A. De O. Pantoja. Differential dynamic programming and newton's method. *Intnl. J. Control*, 47:1539–1553, 1988.

[28] C. Polychronopoulos. The Parafrase-2 restructurer. In *Proc. of the Second Workshop on Languages and Compilers for Parallel Computing*, August 1989.

[29] M.L. Psiaki and K. Park. A parallel solver for trajectory optimization search directions. Manuscript, Mechanical and Aerospace Engineering, Cornell University, 1991.

[30] W. Pugh. The Omega test: A fast and practical interger programming algorithm for dependence analysis. In *Proceedings of Supercomputing 91*, 1991.

[31] D. Ralph. Stagewise decomposition in discrete-time optimal control. Manuscript in progress, Advanced Computing Research Institute, Theory Center, Cornell University, 1992.

[32] A. Rogers and K. Pingali. Process decomposition through locality of reference. In *Proc. of the 1989 SIGPLAN Conference on Programming Language Design and Implementation*, 1989.

[33] G. Subbarayan. *Bone construction and reconstruction: A variational model and its applications*. PhD thesis, Dept. of Mechanical Engineering, Cornell University, 1991.

[34] C. Sun. Efficient parallel solutions of large sparse spd systems on distributed-memory multiprocessors. Technical Report CTC92TR102, Advanced Computing Research Institute, Theory Center, Cornell University, August 1992.

[35] M. Wolf and M. Lam. A loop transformation theory and an algorithm to maximize parallelism. *IEEE Transactions on Parallel and Distributed Systems*, October 1991.

[36] S.J. Wright. Solution of discrete-time optimal control problems on parallel computers. *Parallel Computing*, 16:221–237, 1990.

[37] S.J. Wright. Partitioned dynamic programming for optimal control. *SIAM J. Optimization*, 1:620–642, 1991.

[38] Zhijun Wu and Thomas F. Coleman. A parallel row distributed linear algebra system. Technical Report CTC92TR93, Advanced Computing Research Institute, Theory Center, Cornell University, April 1992.

[39] H. P. Zima, H. Bast, and M. Gerndt. SUPERB: A tool for semi-automatic MIMD/SIMD parallelization. *Parallel Computing*, 6:1–18, December 1988.