Further Reading in High Performance Compilers for Parallel Computing

Michael Wolfe

November 9, 1994

To keep the book *High Performance Compilers for Parallel Computing* to a reasonable length, I left out most of the original references. The complete references are included in this document, which has one section for each chapter of the text.

1 High Performance Systems

A description of commercial and research compilers aimed at high performance and parallel computer systems can be useful to understanding what people have found important and what lessons they have learned. While specific optimizations are often published, overview papers of whole compilers are rare. General compiler techniques, such as parsing, type inferencing, and creating an intermediate form, can be found in standard textbooks such as those by Fischer and LeBlanc (1988) and Aho, Sethi, and Ullman (1986). Cohagan (1973) and Wedel (1975) describe some of the analysis used in one of the first automatic vectorizing compilers, which was targeted for the Texas Instruments Advanced Scientific Computer. Strohecker and Kolsky (1986) describe the vectorizing compiler for the IBM 3090 Vector Facility. Coleman (1987) describes a commercial compiler for the Unisys ISP vector processor. Allen (1988) and Allen and Johnson (1988) describe the compilers for the Ardent Titan, a single-user, high-performance graphics engineering vector multiprocessor workstation, and Mercer (1988) describes the Convex compiler of the same era. Padua and Wolfe (1986) give an overview of the state of parallel compiler technology in the mid-1980s. Tjiang et al. (1991) describe a compiler that uses a single intermediate representation for both parallel and scalar optimizations. Yasue, Yamana, and Muraoka (1992) discuss a Fortran compiler for a dataflow machine, which uses many of the techniques presented in this text. Zima and Chapman (1991) and Wolfe (1989b) give reasonable overviews of techniques used in parallelizing compilers of the late 1980s; Banerjee et al. (1993) also give a much briefer summary. Kennedy (1992) summarizes many of the challenges of software development, including languages and compilers, for supercomputer and parallel systems in particular.

The Parafrase Analyzer and compiler research group at the University of Illinois developed many techniques that affect modern research and commercial compilers; see Kuck et al. (1984). Parafrase-2, described by Polychronopoulos et al. (1989a, 1989b), is a successor project with even more ambitious goals. A contemporaneous project at Rice University developed the PFC parallelizing compiler (Allen and Kennedy [1984a]); this project had a direct impact

on several commercial compilers, such as at Convex (Mercer [1988]) and IBM (Scarborough and Kolsky [1986]). PTRAN was developed at IBM's T. J. Watson Research Laboratory around the same time (Allen et al. [1987, 1988a]). Together, these three research efforts developed many of the analysis methods on which modern parallel compiler research and development are based.

Several interactive tools have been developed in research environments. Smith and Appelbe (1988, 1989) describe PAT, an interactive parallelizing assistant tool. ParaScope is another interactive parallelizing tool, described by Balasundaram et al. (1989), Kennedy, McKinley, and Tseng (1991a, 1991b) and Cooper et al. (1993); it is an outgrowth of the PFC project at Rice University. Irigoin, Jouvelot, and Triolet (1991) discuss the PIPS parallelizer, which uses extensive interprocedural analysis to find additional parallelism.

A summary of various parallel computer systems is given by Hwang (1993); he includes a chapter on parallel programming models, languages, and compilers. Another summary of the features of many high performance and parallel computers can be found in the book by Stone (1990). The terms SIMD and MIMD were introduced by Flynn (1972). One of the early parallel supercomputers was the Illiac-IV, described by Barnes et al. (1968). The first widely used, commercially successful supercomputer was the Cray-1, described in an overview by Russel (1978). Three commercial supercomputer projects in the late 1970s are presented by Kozdrowicki and Theis (1980). Several supercomputers of the mid-1980s are discussed by Stevens (1986). The SPMD programming model was introduced by Darema et al. (1988) for shared-memory parallel programming. A good summary of issues related to modern computer architecture is found in the book by Hennessy and Patterson (1990).

Current high performance and parallel computer system designs can often be found in the proceedings of various conferences, such as the annual International Conference on Parallel Processing, the International Symposium on Computer Architecture, the Supercomputing conferences, or the Compcon meetings. Lundstrom and Barnes (1980) describe the Burroughs design for a Flow Model Processor, a large multiprocessor. Gottlieb et al. (1983) and Edler et al. (1985) discuss the design of the Ultracomputer, a large-scale shared-memory multiprocessor designed at New York University. Gajski et al. (1983) describe the Cedar, a hierarchical multiprocessor designed and constructed at the University of Illinois; Eigenmann et al. (1990a, 1990b) summarize the Fortran compiler for the Cedar. Behr, Giloi, and Mühlenbein (1986) discuss design of the Suprenum machine, a parallel supercomputer designed and constructed in Germany. Pfister et al. (1985) describe the IBM RP3 prototype scalable shared-memory multiprocessor. The Alliant FX/8, a shared-memory vector multiprocessor aimed at the cost- and performance- conscious market, is described by Perron and Mundie (1986). The Convex C-1, a vector uniprocessor marketed around the same time, is discussed by Wallach (1986). The DASH project constructed a prototype large-scale cache-coherent shared-memory multiprocessor, described by Lenoski et al. (1992, 1993). A high level view of the Thinking Machines Connection Machine CM-5, a modern highly, parallel supercomputer, is given by Hillis and Tucker (1993).

2 Programming Language Features

A history of the early Fortran language and compilers is given by Backus (1981); several interesting comments are made there, including some from users who were unhappy with the speed of the compiler, and an excerpt from a November 1954 Fortran Preliminary Report stating that "... Fortran should virtually eliminate coding and debugging ..." (relative to machine language programming, of course). Fortran 90 programmers can find a description of the language by Metcalf and Reid (1990), whereas compiler writers may want the complete details by Adams et al. (1992). High Performance Fortran is fully described in the handbook by Koelbel et al. (1994), which borrows a great deal from the efforts of the compiler for the German Suprenum machine (Zima, Bast, and Gerndt [1988]) and from the Fortran D project at Rice University (Hiranandani, Kennedy, and Tseng [1992b]). Both the "traditional" C language and ANSI standard C are described by Harbison and Steele (1991). C++ is described by Stroustrup (1991) and Ellis and Stroustrup (1990), and in a primer by Lippman (1991). Pros and cons of using C for numerical computing are discussed by MacDonald (1991).

The Blaze language (Mehrotra and Rosendale [1987]) had a parallel loop, with copy-in copy-out semantics, like our `dopar`. A type of loop in which the iterations can be executed in any order is described by Wolfe (1992b); the same loop model is used by Solworth (1992) to generate better parallel code for operations on linked lists. The concurrent model of parallel loop execution is described by Cytron (1986). Yang et al. (1993) give a method to integrate a parallel loop with reduction operations to improve performance. Albert et al. (1988, 1991) describe the use of a `forall` construct for programming parallel machines in data-parallel style. Philippsen and Tichy (1992) use a `forall` statement in an extended Modula-2 language with two flavors: synchronous, like the one we present here, and asynchronous, which is more like a `dopar`. Midkiff, Padua, and Cytron (1989, 1990) discuss the problems that can arise when trying to analyze and optimize programs with poorly designed parallel constructs. The general parallel loop model discussed in this chapter is presented more extensively by Li and Wolfe (1994).

Refined Fortran and Refined C were designed as extensions to Fortran and C in which it is possible to express potential aliasing relationships that the compiler can use to identify more parallelism; the extensions are described by Dietz and Klappholz (1986) and by Klappholz, Kallis, and Kong (1989, 1989). In the 1980s and early 1990s, an ad hoc group of parallel computer system vendors and academic researchers tried to design a shared-memory computational model and extensions to Fortran; the group was called the Parallel Computing Forum (PCF), and the proposed extensions were called PCF Fortran (Leasure [1991]). A summary of various parallel Fortran language dialects is presented by Guzzi et al. (1990), and general language topics for parallel programming are presented by Pancake (1993). Karp and Babb (1988) describe the use of 12 Fortran dialects for nine parallel machines to solve a simple problem; Babb (1988) describes the process that generated these 12 languages in detail.

Parallel languages are often developed for particular machines, such as for the Thinking Machines Connection Machine (Thinking Machines Corporation [1991]) or Sequent parallel processor (Sequent Computer Corp. [1986]). Sometimes fixed hardware characteristics, such as the number of processors, are visible in the language, as in Glypnir, the first high level language for the Illiac IV (Lawrie et al. [1975]), and DAP-Fortran (Flanders et al. [1991]). The C*

language was developed as a SIMD, data-parallel language for the Connection Machine; Quinn, Hatcher, and Jourdenais (1988, 1990) and Hatcher and Quinn (1991) describe a compiler for C* that is targeted at MIMD shared-memory or message-passing multiprocessors. Array languages similar to APL are often explored for parallel programming, as discussed by Ortiz, Pinter, and Pinter (1991).

Advocates of higher level programming languages have always been hampered by the relatively weak performance of these languages on numerical codes compared to sequential languages like Fortran or C with aggressive optimizing compilers. Gopinath and Hennessy (1989) describe one of the important optimizations for functional languages, copy elimination. A great deal of work in the SISAL project has attempted to address the performance complaint (Feo, Cann, and Oldehoeft [1990]). Kuck, McGraw, and Wolfe (1984) debate whether functional programming languages can replace Fortran as the premier high performance language. Cann and Feo (1990) describe a small experiment that shows that optimized SISAL code can approach and even beat the performance of compiler-optimized Fortran code; Cann (1991, 1992) follows this up with more evidence that performance is no longer an obstacle for declarative programming languages. Haskell and Crystal are declarative languages that contain primitives to support scientific computation; Anderson and Hudak (1990) describe compiler analysis for Haskell, while Chen (1986b) and Chen, Choo, and Li (1988) describe the Crystal compiler. In both cases, many of the analysis techniques are similar to those described in this text.

Sometimes particular language rules have unexpected side effects with respect to optimization. Cooper, Hall, and Torczon (1992) describe a situation where inlining in a Fortran compiler weakened the analysis. In this case, the program passed parts of a single array to distinct reference arguments; since the subroutine defined one of the arguments, the compiler used the Fortran rule that the arguments must not be aliased. After inlining the subroutine, this rule no longer applied, and the compiler analysis was less precise.

Goldberg (1990) describes the IEEE floating point arithmetic standard. Goldberg (1991) and Fateman (1982) discuss language implications of floating point arithmetic.

3 Basic Graph Concepts

Any general textbook on discrete math will cover basic concepts of sets and logic, such as the books by Stanat and McAllister (1977) or Tremblay and Manohar (1975). Logic and discrete math are described by Gries and Schneider (1993) and Stanat and McAllister (1977). Aho, Hopcroft, and Ullman (1974) give many graph algorithms that are in common use in compilers. Cormen, Leiserson, and Rivest (1990) describe and analyze many algorithms that are in common use; they give an alternate algorithm for finding strongly connected components for a directed graph. The algorithm to find strongly connected components shown here is due to Tarjan (1972), who proves it correct; it is also described by Aho, Hopcroft, and Ullman (1974).

Control flow graphs and dominators have long been used in compilers. Lengauer and Tarjan (1979) describe a fast and efficient algorithm to find the dominator tree of a CFG. Ferrante and Ottenstein (1983, 1987) define the concept of control dependence in terms of postdominators, and use the control depen-

dence graph to replace the control flow graph; they use an irreflexive definition of postdominance, and a different, but equivalent, definition of control dependence. Cytron et al. (1989, 1991) describe the algorithm for finding dominance frontiers, and show the relationship between the postdominance frontiers and control dependence graph.

Sarkar (1989) uses control dependence relations to reduce the execution cost of inserting profiling counters; one counter is used for two (or more) basic blocks that are control-equivalent, and one counter is used for both the true and false paths of conditionals. The PTRAN compiler uses the forward control dependence graph (without back edges) to simplify program analysis (Cytron, Ferrante, and Sarkar [1989]). Our definition of dominance frontier is slightly different than that used in previous work. Cytron, Ferrante, and Sarkar (1990) show a more concise algorithm for finding dominance frontiers and control dependence relations.

The definition we use for natural loop is equivalent to Tarjan's definition of interval (Tarjan [1974]). The reverse postorder numbering is equivalent to the depth-first ordering used to speed up certain data-flow analysis algorithms (Hecht [1977]).

4 Review of Linear Algebra

Many of the techniques reviewed here for dealing with integer matrices, as well as more details on integer programming, are discussed in greater detail in the books by Nemhauser and Wolsey (1988) and Schrijver (1986). Banerjee (1993) also discusses these topics in the context of restructuring compilers. General integer programming tries to find a solution that optimizes some cost function in a region bounded by linear inequalities. The techniques in which we are interested have no cost function, so we may be able to take advantage of simpler solvers.

Euclid's algorithm to find the greatest common divisor is the first mathematical algorithm, developed several thousand years ago. Lamé (1844) showed that the number of iterations for two integers is bounded by five times the number of digits in the smaller number. Bradley (1970) discusses finding the gcd of a sequence of numbers, along with finding a solution to the gcd equation.

Fourier-Motzkin projection is discussed by Williams (1976, 1983), Dantzig and Eaves (1973), and Duffin (1974). Ancourt and Irigoin (1991) use Fourier-Motzkin projection to find loop limits after restructuring (reordering) the loops. Kuhn (1980) uses Fourier-Motzkin projection to solve data dependence problems. Maydan, Hennessy, and Lam (1991) and Wolfe and Tseng (1992) both reduce the number of unknowns by first eliminating equalities, then use Fourier-Motzkin to eliminate inequalities, in data dependence problems. We will show how to use our algorithms to address these problems.

The extreme value method for rectangular regions was used by Banerjee (1976) to solve data dependence problems. Banerjee (1988a) extended it to nonrectangular regions, and also describes the algebraic techniques to reduce integer systems of equations. Another method to reduce integer systems of equations was developed by Pugh (1991b, 1992a), who introduces a new operator ($\widehat{\text{mod}}$, pronounced "mod-hat"), which allows the system to reduce more quickly. Pugh also introduced the concept of a dark shadow. His system, called the Omega Test, solves general integer programming problems as well as more

complicated Presburger formulas. In the domain studied in this chapter, when the Fourier-Motzkin projection technique is inexact and the dark shadow test fails to find a solution, Pugh's system can "splinter" the problem into a set of problems, one of which must have an exact solution for the original system to have a solution.

Feautrier (1988a) describes a scheme called parametric integer programming to solve integer programming problems that arise in the framework of a compiler.

5 Data Dependence

The concepts of data dependence have been well-known for quite some time (Bernstein [1966]); data dependence distance vectors have long been used in the systolic array community to generate regular communication patterns (Karp, Miller, and Winograd [1967]). Allen (1987) coined the terms *dependence level*, *loop carried* and *loop independent* dependence. Kuck et al. (1981) and Allen, Callahan, and Kennedy (1987) show how to use dependence graphs for parallelization and other optimizations. Before control dependence relations were widely used, some compilers handled control conditions by converting them to data dependence relations (Allen et al. [1983]).

The distinction between address-based and value-based dependence relations arises frequently. Brandes (1988) uses the term *direct dependence* for value-based dependences; using direct dependences, more aggressive dead code elimination can be done, and variable renaming can be used to eliminate false dependence relations.

Banerjee (1988b) introduces basic concepts of dependence testing. Data dependence levels are introduced by Allen (1987). Wolfe (1991b) describes the relations of various data dependence abstractions, such as dependence level, direction vector, distance vector, and semantic information such as reduction directions, and shows the strengths and weaknesses of each. Pugh (1992b) challenges some traditional definitions of dependence distance as ill-defined; we have attempted to address his concerns here.

Cartwright and Felleisen (1989) discuss the meaning of data dependence, pointing out the difference between preserving the actual behavior and the observable behavior of correct, terminating programs. Pingali et al. (1990, 1991) and Johnson, Li, and Pingali (1991) describe an intermediate representation called *dependence flow graphs*, which subsumes data and control dependence graphs and has executable semantics.

The program dependence graph is described more fully by Ferrante, Ottenstein, and Warren (1987); their description uses region nodes to collect graph nodes with identical control dependence predecessors. Horwitz, Prins, and Reps (1988) show that PDGs are adequate, in that they distinguish between programs that are not equivalent.

6 Scalar Analysis with Factored Use-Def Chains

The algorithms for finding the FUD chains (equivalent to the static single assignment, or SSA, form) are presented by Cytron et al. (1989, 1991). A factored SSA form is described by Choi, Cytron, and Ferrante (1994), which closely resembles FUD chains. An asymptotically faster algorithm for placing ϕ-terms is given by Cytron and Ferrante (1993).

An alternative "sparse" data-flow graph representation for program analysis is shown by Beck and Pingali (1990), Beck, Johnson, and Pingali (1991), and Johnson and Pingali (1993). The program dependence web (Ballance, Maccabe, and Ottenstein [1990]) is another representation that encompasses the advantages of several others. One step to build the program dependence web is construction of the gated single assignment form, where ϕ-terms are replaced by gates with a predicate to choose the appropriate operand; Havlak (1993) discusses how to construct a version of the gated single assignment form. Choi, Cytron, and Ferrante (1991) explore using sparse data-flow graphs for solving any data-flow problem.

Jouvelot and Dehbonei (1989) describe an analysis technique that finds induction variables, invariant expressions and other generalized reduction operations in loops. Ammarguellat and Harrison (1990) use abstract interpretation with a set of recurrence templates to find induction variables and other recurrence operations. Haghighat and Polychronopoulos (1992) use symbolic interpretation to find induction variables and perform other scalar analysis; they also use symbolic analysis to improve the results of parallelization (1993). Wolfe (1992a) describes the method presented here for finding induction variables using FUD chains; these advanced methods can also identify and classify a variety of other interesting patterns that are useful for dependence analysis. Note that we are interested in finding induction variables for use in dependence analysis, which is different from the goal of finding induction variables for strength reduction (Allen, Cocke, and Kennedy [1981]).

The constant propagation algorithm shown here was developed by Wegman and Zadeck (1985, 1991). An interprocedural constant propagation algorithm is described by Callahan et al. (1986). Handling aliases, such as pointer aliases, in data-flow analyses based on FUD chains or SSA form is discussed by Cytron and Gershbein (1993). Srinivasan, Hook, and Wolfe (1993) explore construction of the FUD chains or the SSA form in programs with explicitly parallel constructs.

7 Data Dependence Analysis for Arrays

Initial work in array dependence testing was done in the context of systolic array generation for uniform recurrence equations (Karp, Miller, and Winograd [1967]). Many of the ideas were also used in early compiler work before data dependence concepts were fully developed (Firestone [1973]). The extreme value test, developed by Banerjee (1976), was one of the first attempts to apply serious mathematics to the problem of dependence between general array references; its use can also be found in the paper by Banerjee et al. (1979). The gcd and extreme value test for direction vectors are described by Wolfe (1982) and Wolfe and Banerjee (1987). The exact test for single variables is described by Banerjee (1979). The direction vector hierarchy is described by Burke and Cytron (1986). Linearization of array subscripts is proposed by Banerjee et al. (1979). The idea of a self-loop was used in the PTRAN compiler (Burke and Cytron [1986]).

Wallace (1988) describes using limited Gaussian elimination on the dependence matrix to determine whether the dependence equation has a unique solution and to diagonalize the dependence matrix so the generalized gcd test can be used; if those fail to determine independence, he suggests constructing a constraint matrix (adding slack variables) and attempting to find a solution using a variation of the simplex method. Integer and linear programming techniques

are also discussed by Li and Yew (1989).

Li, Yew, and Zhu (1989, 1990) describe the λ-test for multidimensional array subscript dependence analysis, which constructs various linear combinations of the dependence equations for each dimension and solves each linear combination; in certain circumstances, the λ-test can be shown to be exact (Li [1989]). Goff, Kennedy, and Tseng (1991) describe the dependence analysis developed for the PFC and ParaScope tools; these tools substitute information from one equation to simplify another, and take advantage of the special structure of the equations to separate the system into smaller systems. Grunwald (1990) compares this analysis with the λ-test, both of which find combinations of dependence equations to eliminate unknowns. Maslov (1992) discusses splitting dependence equations.

The extreme value test was used in a great deal of early work, including that of Allen (1987) and Wolfe and Banerjee (1987), where it is usually referred to as Banerjee's test. Banerjee (1988a) describes the extensions to triangular loop limits. Psarris, Klappholz, and Kong (1991) and Klappholz, Psarris, and Kong (1990) argue that in practice, the extreme value test is accurate for single dependence equations, even when direction vector constraints are considered (Psarris [1992]). Kong, Klappholz, and Psarris (1990, 1991) and Psarris, Kong, and Klappholz (1991, 1993) describe the I-test, a refinement of the extreme value test that improves its accuracy and incorporates the gcd test as well. The I-test finds lower and upper bounds for the dependence function $L \le a_1 i_1 + a_2 i_2 + \cdots a_n i_n \le U$, and checks whether there is a multiple of $g = \gcd(a_1, a_2, \ldots, a_n)$ in the range $[L : U]$. If there is not, there is no solution to the dependence equation; if there is, then one of the unknowns is eliminated (as with the extreme value test) and the technique iterates. Petersen and Padua (1992, 1993) describe experiments to evaluate the relative accuracy of various dependence tests, and the importance (or lack thereof) of improving test accuracy.

Kuhn (1980) proposed using Fourier-Motzkin projection for dependence testing, though he did not first consider removing equalities to reduce the number of unknowns. The generalized gcd test was introduced by Banerjee (1988a). Wolfe and Tseng (1992) propose to follow the generalized gcd test with some simple analysis to find exact dependence distances, when possible, and with Fourier-Motzkin projection otherwise; they eliminate the unknowns in a fixed order, the most deeply nested loop first. Maydan, Hennessy, and Lam (1991) describe a series of techniques to eliminate inequalities in the dependence system (including a loop residue step [Shostak (1981)] not presented here) that empirically determine all dependence relations exactly. Eisenbeis and Sogno (1992) discusses an approach based on integer programming techniques, including some techniques to improve the run-time performance. The Omega test for data dependence is described by Pugh (1991b, 1992a); it includes additional analysis to find an exact solution even when Fourier-Motzkin projection gives inexact integer results.

Nontrivial dependence systems include unknown symbols other than loop induction variables, and have been studied from many points of view. Lichnewsky and Thomasset (1988) describe a dependence testing scheme for a vectorizing compiler that uses extensive symbolic manipulation, including symbolic assertions in the program. Haghighat (1990) also explores handling unknown symbols in dependence analysis. Lu and Chen (1990) point out that unknown loop limits can be treated like additional variables in the dependence system, and that additional constraints can sometimes be found in conditional expressions. Klap-

pholz and Kong (1992) also discuss adding conditional expression constraints in the extreme value test.

The complexity of the data dependence problem was studied by Shen, Li, and Yew (1989, 1990); they also show statistics on how often different dependence tests are used in their system, and the success rate of each. See the papers by Goff, Kennedy, and Tseng (1991) and Maydan, Hennessy, and Lam (1991) for comparative studies using different sets of dependence tests. Wolfe (1993) gives guidelines on effectively engineering a data dependence program.

Compiler analysis can compute kill information for scalars precisely, but for array references the analysis is more complex; classical memory-based data dependence analysis is inherently imprecise because it ignores array kills. Ribas (1990) finds the latest assignment for each element, computing self-kills by using integer programming. Maydan, Amarasinghe, and Lam (1992, 1993) present a data-flow framework for array references, which is similar in concept to, but simpler than, the method by Feautrier (1991). Pugh and Wonnacott (1992, 1993) describe methods to compute array kills by extending the Omega test to solve more general problems. Maslov (1994) describes another efficient array data-flow analysis method that uses the Omega test to reduce the problem form. Duesterwald, Gupta, and Soffa (1993) give a simpler scheme based on data-flow analysis. Kallis and Klappholz (1991) shows another scheme to integrate array references into a classical data-flow framework.

Tzen and Ni (1993b) capture the dependence relations with irregular dependence distances by computing the convex hull of the free variables for the solutions to the dependence equation.

8 Other Dependence Problems

Pointer analysis is the subject of a great deal of research. Jones and Muchnick (1981) present some of the first work to try to automatically determine the structure of the dynamic data structure from program analysis. Landi and Ryder (1991) give a classification of the general problem. Chase, Wegman, and Zadeck (1990) add a heap reference count to the storage shape graph, which counts the number of unnamed (nonvariable) pointers to each object; if the heap reference count is one, then the data structure must be a simple cycle, list, or tree. Larus and Hilfinger (1988) use an alias graph instead of a storage shape graph to represent pointer aliases. The points-to analysis presented here is shown in more detail by Emami, Ghiya, and Hendren (1994).

Hendren and Nicolau (1989, 1990) discuss a path matrix representation of pointer paths to determine aliasing or dependence for programs that construct and traverse dynamic data structures. Horwitz, Pfeiffer, and Reps (1989) build on the work of Jones and Muchnick (1981). Loeliger et al. (1991) show some results of an implementation of interprocedural pointer tracking in a commercial C compiler. Smith (1991) shows the importance of pointer analysis and pointer arithmetic when compiling C for vector machines (and by extension, for parallel machines). Guarna (1988) also looks at pointers as they are used in loops. Landi and Ryder (1992), Landi, Ryder, and Zhang (1993), and Choi, Burke, and Carini (1993) discuss interprocedural pointer alias and side effect analysis. See the short article by Marlowe et al. (1993) for a discussion of two general pointer-aliasing approaches. Realizing that pointers are used to implement complex data structures that may have regular characteristics, Hendren, Hummel, and

Nicolau (1992) have developed a language for annotating programs to describe characteristics of the actual data structure that is being constructed and/or traversed with the pointers. Hummel, Hendren, and Nicolau (1992) show how these annotations can be used to improve analysis; Hummel, Hendren, and Nicolau (1994a) describe a simple language for expressing alias properties; and Hummel, Hendren, and Nicolau (1994b) show a scheme to find data dependence by a theorem-proving technique, given a set of simple axioms defined by the annotations.

Interprocedural data-flow analysis was studied by Barth (1977, 1978). Weihl (1980) looked at a wide range of problems caused by procedure variables, label variables, pointers, etc. Cooper and Kennedy (1984) show an efficient scheme for addressing the interprocedural MOD problem. The solution presented in this chapter is derived from the work of Cooper and Kennedy (1988).

Interprocedural alias analysis gives rise to subtle problems; one of these is explored by Burke and Choi (1992), and the analysis is studied in general by Cooper (1985). A fast algorithm to find aliases for formal reference arguments and global variables is described by Cooper and Kennedy (1989); the same algorithm is refined by Mayer and Wolfe (1993). The R^n programming environment (Carle et al. [1987]), supports extensive interprocedural analysis (as described by Cooper, Kennedy, and Torczon (1986)) such as alias analysis, interprocedural constant propagation, recompilation analysis, linkage tailoring, and side effects. Callahan et al. (1986) uses a *program summary graph* with entry, exit, call, and return nodes for each procedure and call statement, to capture flow-sensitive interprocedural data-flow information. Hall et al. (1993a) describe a general interprocedural analysis tool that can be used to solve many problems. Callahan et al. (1986) developed the interprocedural constant propagation that we present here. Grove and Torczon (1993) discuss an implementation of interprocedural constant propagation, with the costs and benefits of more powerful jump function implementations analyzed.

Tang (1993) describes a method to summarize the region of arrays that are referenced or defined by projecting the actual references from the loop iteration space onto the array dimension space.

Triolet, Irigoin, and Feautrier (1986) show how summarizing the effects of a procedure can be used to parallelize loops containing calls to that procedure. Callahan and Kennedy (1987, 1988a) summarize the array subregions that are used or modified by a subroutine call using a structure they call "regular sections." Balasundaram and Kennedy (1989b) and Balasundaram (1990) give a simpler, more efficient scheme to address the same problem. Havlak and Kennedy (1990, 1991) describe experience with an implementation of regular section analysis and give a good summary of various regular section methods. Li and Yew (1988a, 1988b, 1988c) describe an alternative scheme that does not summarize references, but describes the region of the array accessed by each array reference. A fast data-flow algorithm that uses rectangular regions is used by Gross and Steenkiste (1990).

Constructing the call graph in the presence of procedure parameters is discussed by Ryder (1979) if there is no static or dynamic recursion. An extension that allows recursion is presented by Callahan et al. (1987). Hall and Kennedy (1992) show the efficient but less precise method presented here.

9 Loop Restructuring

Some of the transformations presented in this chapter have been well known for some time; see the descriptions by Allen and Cocke (1972) and Loveman (1976, 1977).

Automatic loop interchanging was first implemented in the compiler for the TI Advanced Scientific Computer (Cohagan [1973]); it was discussed more generally by Wolfe (1982) and Allen (1983). Simple loop interchanging was published by Allen and Kennedy (1984b), and handling nontrivial loop limits is shown by Wolfe (1986a). A more general discussion is presented by Banerjee (1989). Ancourt and Irigoin (1991) discussed using Fourier-Motzkin projection to find loop limits after restructuring (reordering) the loops.

Smith, Appelbe, and Stirewalt (1990) and Wolfe (1991a) discuss incrementally updating the dependence graph after restructuring the program. These techniques were used in Tiny, an experimental loop restructuring research tool (Wolfe [1991c]). The ParaScope interactive tool updates both the program source and the dependence information after an edit by the programmer (Kennedy, McKinley, and Tseng [1993]). Warren (1984) discusses a hierarchical representation of data dependence to support statement reordering and loop fusion.

Fission for loops containing conditionals requires computing the expression that controls the condition and saving the result in a temporary array. Several approaches have been described in the literature. If all the statements with the same flow constraints are kept in the same loop after fission, no conditional arrays are necessary. Towle (1976) and Baxter and Bauer (1989) suggest using conditional arrays only when statements with the same control constraints end up in different loops after fission. Kennedy and McKinley (1990) describe the process of loop fission when the loop contains conditionals or a loop dependent exit. Hsieh, Hind, and Cytron (1992) demonstrate loop fission using a general control flow graph and with multiple exits.

Eigenmann and Blum (1991, 1991, 1993) describe additional program transformations, aimed at allowing the generation of parallel code in more cases; in particular, these include array privatization and classifying generalized induction variables. Polychronopoulos (1987b) introduces loop coalescing with the goal of simplifying code generation and scheduling for parallel computers.

There have been many attempts to place a number of transformations in a single framework, such as a matrix formulation. Banerjee (1990) and Wolf and Lam (1990, 1991a) explore unimodular transformations for tightly nested loops. Sarkar and Thekkath (1992) and Wolfe (1991a) present similar schemes for characterizing a number of transformations, though the transformations are not composable in the same sense that matrix operations are composable. Linear loop transformations have long been used in the systolic array community to map regular algorithms onto a regular processor ensemble (Karp, Miller, and Winograd [1967], Chen [1986a]). Nonlinear and piecewise linear transformations are described by Lu (1991).

Loop skewing (Wolfe [1986b]) was developed as an alternative implementation of the wavefront (Muraoka [1971]) or hyperplane (Lamport [1974]) method of parallel execution for loops with dependence relations; the wavefront method is effectively the same as using unimodular transformations, focused on parallel execution. Circular loop skewing is presented as a loop transformation by Wolfe (1989a).

Irigoin and Triolet (1988) and Wolfe (1989c) introduce the concept of tiling

for locality and parallelism. Ramanujam (1992) gives another derivation of tiling, in terms of hyperplanes that divide the iteration space into tiles, including code generation methods. The method presented here for code generation for nonunimodular transformations is derived from the work by Li and Pingali (1992a, 1994). Barnett and Lengauer (1992) and Ramanujam (1992) also discuss nonunimodular loop transformations.

Interprocedural program transformations can also play an important role in a restructuring compiler. Procedure cloning is well known in the partial evaluation literature, where it is called *polyvariant partial evaluation*. Cooper, Hall, and Kennedy (1990) discuss cloning in the framework of a programming environment that manages interprocedural issues. Hall, Kennedy, and McKinley (1991) describe interprocedural loop extraction and loop embedding and their use in a programming environment. Cooper, Hall, and Torczon (1991) show results of an experiment with procedure inline substitution, including increases in object code size and compile time, and reductions in execution time.

Whitfield and Soffa (1991) have developed a language for writing optimizations, including the types of restructuring described in this chapter.

Scalar expansion can remove most anti- and output dependence relations due to scalar assignments in a loop. Array expansion can be used when arrays are assigned, but are not indexed with all the loop indices (the so-called "missing index" problem). Chen and Chang (1987) show array expansion for simple cases.

Torres et al. (1993) discuss code generation when alignment is combined with linear loop transformations by giving each statement an alignment offset from other statements.

10 Optimizing for Locality

McKellar and Coffman (1969) noted that using a submatrix formulation of matrix operations essentially tiling—augmented with the proper data layout will improve memory hierarchy performance; at that time, the memory hierarchy in question was paged virtual memory. The earliest work to use automatic program restructuring to optimize for the memory hierarchy was done by Abu-Sufah (1978) and Abu-Sufah, Kuck, and Lawrie (1981). The importance of properly taking advantage of the memory hierarchy is conveyed by the design of the Level-3 BLAS (Dongarra et al. [1990]). Callahan and Porterfield (1990) show that up to half the time of some supercomputer applications can be wasted in processing cache misses. Goodman (1983) shows performance characteristics when the cache memory is used to reduce traffic between the processor and the memory. Lam, Rothberg, and Wolf (1991) discuss many aspects of optimizing for locality, such as choosing the tile size and copying data to reduce cache interference. Temam, Granston, and Jalby (1993) give other heuristics for choosing data to copy to separate, conflict-free locations.

Gannon, Jalby, and Gallivan (1987, 1988) describe early work on an automatic scheme to manage a memory hierarchy automatically in a parallel programming environment using the concept of a *reference window*, which is the number of loop iterations from when the data element is first reference until it is no longer needed; minimizing reference windows will improve the performance of the memory hierarchy. Eisenbeis et al. (1990) use a similar scheme to copy frequently used data into a fast local memory, and Bodin et al. (1992)

use reference windows to manage cache utilization on a uniprocessor. Nicolau (1987, 1988) uses an optimization essentially equivalent to tiling (following by fully unrolling the element loops) to optimize for locality when compiling for fine-grain parallel machines, such as VLIW machines. Wolf and Lam (1991b) distinguish between reuse and locality, and between self-reuse and group-reuse, and show a scheme to quantify reuse and to try to improve locality. Wolf (1992) uses unimodular transformations to optimize for locality and parallelism. Carr and Kennedy (1989, 1992) show the strengths and weaknesses of tiling in a compiler; they show several cases in which tiling fails, and that index-set splitting can address some of these cases.

Ferrante, Sarkar, and Thrash (1991) show compiler analysis methods to estimate how many cache lines will be needed for each reference in a loop, which can be used to guide compiler optimizations.

Gornish, Granston, and Veidenbaum (1990) describe a scheme for inserting prefetch operations at the earliest point in the program after which the prefetched data is guaranteed to be used. Callahan, Kennedy, and Porterfield (1991) describe a simple but effective method for inserting prefetch operations for algorithms with regular data access patterns. Mowry, Lam, and Gupta (1992) evaluate another scheme for inserting prefetch operations with the compiler.

Prepaging is the analog in virtual systems to cache-line prefetching; Di Antonio, Ebert, and Vitale (1989) describe the results of some experiments showing the effectiveness of prepaging on programs with very large data sets.

11 Concurrency Analysis

Early work on parallelizing compilers is described by Padua (1979) and Padua, Kuck, and Lawrie (1980). Banerjee (1993, 1994) covers many aspects on parallelization and loop restructuring in a linear algebra framework.

Self-scheduling for nested loops is discussed by Tang and Yew (1986) and Fang et al. (1990). One of the first tapering scheduling methods was guided self-scheduling, developed by Polychronopoulos (1986) and described by Polychronopoulos and Kuck (1987). The factoring scheduling method is discussed by Hummel, Schonberg, and Flynn (1991, 1992), who show it to be experimentally better than simple chunk scheduling or guided self-scheduling; if the coefficient of variance of the execution time for the iterations is known, factoring can be modified to compute a more optimal batch size. Lucco (1992) discusses dynamic scheduling for loops with irregular iteration execution times. Tzen and Ni (1991, 1993a) describe trapezoidal scheduling, a simpler scheduling algorithm that gets much of the benefit of other tapering schemes, but with a simple (cheap) tapering method. Haghighat and Polychronopoulos (1993) use symbolic analysis to try to determine at compile-time how to schedule the loop iterations. The idea of using a queue of *generator* procedures to assign work in a parallel system was implemented in the BBN Butterfly multiprocessor. This system also allowed programs to scatter data structures across the many memory modules to reduce network contention (Rettberg and Thomas [1986]).

The cycle shrinking optimization is due to Polychronopoulos (1987a, 1988a); he also describes a scheme to strip mine a loop using dynamically chosen (at run-time) strip sizes such that the iterations in each strip are data independent, allowing the element loop to be executed in parallel. Saltz, Mirchandaney,

and Crowley (1989, 1991) describe a generalization of this where the iterations are sorted into independent groups to minimize execution time. Robert and Song (1992) describe several improvements to this approach. Ayguadé et al. (1990) show a method where each statement is scheduled independently. Shang, O'Keefe, and Fortes (1994) relate cycle shrinking to linear loop transformations. See Polychronopoulos (1988c) for a number of parallelization techniques.

An alternative to cycle shrinking, when the dependence distances are all known, is to schedule the operations to achieve as much parallelism as possible; such a scheme is described by Ayguadé et al. (1989) and Labarta et al. (1991)

Several schemes have been proposed to partition iterations so that each partition is independent of the others, and can be executed in parallel. Peir and Cytron (1987, 1989) show how to partition the loop based on unimodular transformations of the dependence matrix. Liu, Ho, and Sheu (1990) use statement alignment to improve the partitions. Fang (1990) and Fang and Lu (1991) show a scheme that takes into account cache locality, and attempts to reduce or eliminate the effects of false sharing. D'Hollander (1990, 1992) uses unimodular transformations on the dependence distance matrix to reduce it to diagonal or triangular form; applying the same transformations to the iteration space relabels the iterations such that independent partitions can easily be found. Other partitioning schemes are presented by Shang and Fortes (1988) and Huang and Sadayappan (1991).

Alignment and code replication to help parallelization are discussed by Allen, Callahan, and Kennedy (1987). Fang (1990) discuss compiler analysis and code generation for parallelizing `while` loops (noniterative loops). Dietz (1988) handles `while` loops by splitting out the part of the loop that determines the exit condition, counting the number of iterations, and then executing the remainder of the loop as a countable loop. A similar approach is described by Wu and Lewis (1990). Lu and Chen (1991) discuss parallelization of loops with pointers.

Kogge and Stone (1973) use recursive doubling techniques to solve mth order linear recurrences in parallel. Callahan (1991) discusses certain forms of recurrences that can be solved efficiently and in parallel. Ammarguellat and Harrison (1990) use abstract interpretation with a set of recurrence templates to detect and classify simple recurrence relations in loops. Redon and Feautrier (1993) can find and classify a wide range of recurrences in loops, involving both scalars and arrays.

For some algorithms, the execution order of the iterations is immaterial. An example is a set of accumulations; the order in which the accumulations occur is unimportant, if we assume that the roundoff error differences don't matter. With such an algorithm, additional parallelism can be found by allowing any iterations that don't have def-def conflicts to execute in parallel. Graph coloring techniques are used by Hege and Stüeben (1991) to find the nonconflicting iterations.

For many scalar compilers, the optimization order is chosen when the compiler is written; for instance, the compiler may perform instruction scheduling before or after register allocation (or may do scheduling twice). Some optimizations, such as dead code elimination, may be repeated several times. A compiler with a choice of many transformations to apply must also determine an order in which to apply them. For parallel computer systems, a fixed order may not be sufficient. Choosing the best sequence of transformations is a difficult problem, and depends on the characteristics of the program as well as on the target machine. This problem is addressed by Wolfe (1987, 1988a) by using a simple

machine model. Brandes and Sommer (1987a, 1987b) use a rule-based interactive parallelizing programming environment; the user can write rules to capture the program characteristics that affect performance.

A subproblem is the loop mode assignment—the choice of which loop or loops to execute in parallel, vector, or sequential mode. Koblenz and Noyce (1989) describe a mechanism to solve the loop mode assignment problem using a tree representation of the potential ways to parallelize the loops; additional cost analysis determines which of these generates the best performance. Byler et al. (1988) discuss the compiler option of generating multiple versions of a loop, and several means of choosing between the two versions.

Choosing the best transformation to apply next requires the compiler to be able to predict the performance of the target system on the transformed code, at least relative to the untransformed code; one such scheme is shown for the Cyber 205 vector supercomputer by Arnold (1983). A performance prediction scheme for the Alliant FX/8, a parallel vector multiprocessor, is described by Gallivan et al. (1989), assuming that the assembly code is available. Atapattu and Gannon (1989) describe a model of the memory subsystem for the Alliant machine. Fahringer (1994) describes a portable performance predictor that uses the results of a set of "training runs" to learn what performance parameters are important; the parameters estimate work distribution, communication overhead, and data locality. Appelbe and Smith (1992) discuss improving parallelism by using variable reduction and loop alignment.

Several schemes try to avoid the "one transformation at a time" paradigm. Lamport (1974) describes a wavefront scheme essentially equivalent to a unimodular transformation of the index set. Dowling (1990) shows that such an approach can always move all the loop-carried dependences to the outer loop, making all inner loops parallel. Wolf and Lam (1991a) use unimodular transformations to convert the loop to a fully permutable form, then find coarse-grain (outer loop) or fine-grain (inner loop) parallelism; their method, described more fully by Wolf (1992), includes tiling for locality. Punyhs (1991a) describes a scheme to find a schedule for each statement in a loop, then interleaves the schedules for each statement to get the schedule for the whole loop. Ayguadé and Torres (1993) describe a similar, but simpler, scheme.

Several interactive parallelizing tools have been developed. Smith and Appelbe (1989) and Appelbe, Smith, and McDowell (1989) describe Start/Pat, a parallelizing assistant tool, that interactively converts sequential programs to use Cray microtasking directives or other parallel Fortran dialects. Callahan et al. (1988) and Kennedy, McKinley, and Tseng (1991a) describe ParaScope, which contains a rich set of loop transformations. Hall et al. (1993b) describe experiences using this tool.

Allen et al. (1988b) describe how to use the control dependence relation to determine maximal parallelism in a program; the data dependence relations add constraints that are satisfied either explicitly (with synchronization) or implicitly (by scheduling).

Various heuristics have been developed to schedule task graphs (a la Petri nets) onto parallel processors; some of these have been adapted for use in general-purpose compilers for parallel computers, such as described by Girkar (1988). Watts, Soffa, and Gupta (1992) use transformations such as loop fission to create tasks, then construct and schedule the resulting task graph. Polychronopoulos (1988b) proposes a scheme whereby the compiler partitions the program into parallel tasks and generates drive code to schedule the tasks

onto the processors without operating system intervention or overhead. Polychronopoulos (1991) discusses how to construct task graphs from the data and control dependence relations of procedure and loop bodies, and how to generate code that will unfold the parallelism dynamically; Girkar and Polychronopoulos (1991) show how to optimize the generated code in this model.

Burke et al. (1988) shows how PTRAN extracts parallelism based on the control dependence graph. Kasahara et al. (1991) describe a scheme to extract parallelism at several granularities; large grain macrotasks are scheduled onto processor clusters, and iterations of parallel loops in a macrotask are scheduled onto processors in a cluster. Polychronopoulos and Banerjee (1987) discuss processor allocation to maximize speedup when large-grain outer loop parallelism and fine-grain inner loop parallelism are both available.

Privatization or scalar expansion can be used to eliminate false dependence relations (anti- and output dependence). If there are no loop carried flow dependence relations, privatization is feasible; if there are loop carried flow dependence relations, scalar expansion must insert the correct subscripts into the array so that the fetch gets the proper value. Either transformation is relatively simple, once the def-use information is available. To privatize or expand arrays, value-based dependence information must be available. Array privatization is feasible when there are no loop carried flow dependence relations for that array. Array expansion requires the compiler to determine the last assignment to each array element, so that the proper subscript expression can be inserted in the expanded dimensions. This is discussed in detail by Feautrier (1988b). Data-flow analysis techniques to determine when arrays can be privatized are given by Li (1992) and Tu and Padua (1993). A run-time technique to test for situations when privatization will allow execution of a loop as a doall is discussed by Rauchwerger and Padua (1994).

The problem of finding parallelism in sequential code is simply a matter of inspecting the dependence relations. While restructuring the program may allow additional parallelism, exciting the maximum parallel part of the program sequentially retains program correctness. The inverse problem is generating sequential code from a parallel program, which is discussed by Ferrante and Mace (1985) and Ferrante, Mace, and Simons (1988). Sarkar and Cann (1990) discuss compiler methods for a single assignment language, where the goal is to restrict the language parallelism to that which is useful in the target machine.

A related problem is automatically determining when a parallel program is correct. One aspect of this is finding race conditions in a parallel program, when two tasks that are executing in parallel can access the same location (and one of the accesses is a write to that location). This is studied by Balasundaram and Kennedy (1989a). Emrath, Ghosh, and Padua (1989) and Callahan, Kennedy, and Subhlok (1993) also study this for parallel programs with certain types of synchronization primitives. Schonberg (1989) and Mellor-Crummey (1991) describe methods to monitor parallel programs to detect data races dynamically. Netzer and Miller (1990) discuss a scheme to distinguish between between actual data races, data races that could have occurred but didn't, and apparent data races. Netzer and Miller (1992) show how to filter out false races to focus attention on actual problems. Dinning and Schonberg (1990) discuss several dynamic race detection methods for effectiveness and efficiency. LeBlanc and Mellor-Crummey (1987) show a scheme to reproduce the execution behavior of a parallel program, allowing deterministic debugging.

Chen and Yew (1991) argue that restricting parallel loops to those without

loop carried dependence relations reduces parallel performance to unacceptable levels. Scheduling policies for parallel loops that contain synchronization operations can affect performance, as discussed by Tang, Yew, and Zhu (1988). Krothapalli and Sadayappan (1991) discuss methods to remove redundant synchronizations from loops.

All-pairs computations are studied in the framework of linear systolic arrays by Shih, Chen, and Lee (1987). Efficient schedules for multiprocessors are discussed by Theobald and Gao (1991).

Cytron (1984) discusses synchronization and speedup for `doacross` loops. Midkiff and Padua (1986, 1987) discuss adding synchronization in parallelized loops. Wolfe (1988b) describes several forms of synchronization. Allen and Kennedy (1985) describes general parallelization techniques, including methods to reduce synchronization. Allen et al. (1986) show an interactive tool, PTOOL, to aid the user in determining when synchronization is necessary; Henderson et al. (1990) describe their experiences using PTOOL including some limitations of static analysis. Sarkar (1988) discusses minimizing the number of synchronization operations by using counting versus binary semaphores. Li (1991) discusses the problem of adding post and wait operations for event-variable synchronization, a generalization of the advance and await operations discussed here. Insertion of data synchronization operations is discussed by Zhu and Yew (1987) and Tang, Yew, and Zhu (1990), and an optimization to reduce the number of synchronizations required is presented by Li and Abu-Sufah (1987). Su and Yew (1989) argue that data synchronization is too costly unless the number of processors is large. Tzen and Ni (1993b) show how to find a set of uniform synchronization primitives to satisfy dependence relations with irregular distance in doubly nested loops.

When parallelizing sequential loops, finding special operations, such as reductions, is important. Generalized reduction operations are discussed by Jouvelot and Dehbonei (1989).

Hill and Larus (1990) give a short introduction to the problems of caches in multiprocessor systems. Stenström (1990) summarizes several multiprocessor cache coherence schemes in common use. Torrellas, Lam, and Hennessy (1994) discuss how false sharing and spatial locality affects the performance of multiprocessor caches. Early work on multiprocessor cache coherence protocols is presented by Papamarcos and Patel (1984), who introduce the Illinois protocol, and by Katz et al. (1985), who introduce the Berkeley protocol. The scalable coherent interface is intended to serve as an extended computer backplane interface for large numbers of processors, into the thousands; a brief introduction to the coherence protocol is given by James et al. (1990).

Venugopal and Eventoff (1991) discuss using strip mining or tiling to reduce interprocessor cache conflicts. Matsumoto and Hiraki (1993) explore dynamically changing the cache coherence protocols for particular data sets depending on the sharing patterns; for instance, switching between write-update and write-exclusive protocols can be done depending on whether or not the data is shared read-write.

Jalby and Meier (1986) first discussed compiling nested loops into tiles for locality in multiprocessor caches. Wolfe (1989c) discusses the use of iteration space tiles as the unit of parallel execution, while optimizing for locality within tiles. Kennedy and McKinley (1992) show a scheme to find a loop ordering that optimizes locality and parallelism; if the ordering is not feasible, a fall-back position tries to find a "nearby" loop ordering with most of the benefits. Hudak

and Abraham (1990) discuss tiling parallel loops to minimize interprocessor communication. Ramanujam and Sadayappan (1991b) describe how to find extreme vectors to allow iteration space tiling to generate atomic parallel tiles. Ramanujam and Sadayappan (1992) present methods to find extreme vectors in n-dimensional iteration spaces. Boulet et al. (1994) discuss scalable criteria for choosing the best tile shape to reduce interprocessor communication.

Kulkarni et al. (1991) use unimodular transformations to find an outer loop to execute in parallel; this will minimize the number of communication operations, the communication volume, and the load imbalance.

Sarkar and Gao (1991) and Gao et al. (1992) show a scheme to optimize a set of adjacent loops via interchanging, reversal, and fusion, in order to enable array contraction; in the general case, an array may be replaced by a small buffer of values, if the dependence distance is small enough. Kennedy and McKinley (1993) discuss the best way to use loop fission and fusion to generate the most parallelism (and the fewest sequential loops), and also discuss data reuse.

Girkar and Polychronopoulos (1988) describe a mechanism to execute the body of one loop in parallel with the body of an adjacent loop (or n adjacent loops, in general), which might be useful when loop carried dependence relations prevent execution of the loop iterations in parallel.

Collard (1994) discusses parallelizing a `while` loop that surrounds one or more nested iterative loops by using linear transformations and speculative execution.

Cytron, Lipkis, and Schonberg (1990) describe rules for determining when sequential code can be redundantly executed on multiple processors, rather than parking the slaves and letting only the master execute the sequential code.

Hatcher et al. (1991) describe generation of parallel code from Dataparallel C, a SIMD-parallel language for a variety of parallel architectures.

Bik and Wijshoff (1993a, 1993b) describe methods to compile operations on sparse matrices; they also describe (1993c) methods to optimize the parallelism in these algorithms by taking advantage of the sparsity to eliminate unnecessary synchronization.

Cytron, Hind, and Hsieh (1989) describe compiler algorithms to find nonloop parallelism, a la fork-join or cobegin-coend, but allowing a general DAG precedence graph between the parallel tasks. Burke et al. (1989) use this along with loop parallelization to identify parallelism at several levels of granularity. Girkar and Polychronopoulos (1992) use a hierarchical representation of tasks and dependence relations to find nonloop parallelism. Kasahara et al. (1990) describe a compiler that breaks the program into macro-tasks, and dynamically schedules these onto processor clusters.

Pfister and Norton (1985) and Kruskal, Rudolph, and Snir (1988) discuss *combining networks* to improve the performance of synchronization operations on large-scale multiprocessors. Goodman, Vernon, and Woest (1989) describe hardware-supported scalable synchronization schemes, including support for fetch-and-operation primitives in software. Gupta (1989) introduces the *fuzzy barrier* synchronization primitive, which allows a processor to continue to perform useful work after signaling that it has arrived at the barrier but before checking to see if all other processors have arrived. Anderson (1990) discusses methods to improve the performance of the low-level synchronization primitives (spin-waiting) typically used in shared-memory multiprocessors. Graunke and Thakkar (1990) also discuss the performance of various synchronization algorithms.

Some machine designs directly support higher level parallel primitives, such as the parallel loop control used in the Myrias computer system (Beltrametti, Bobey, and Zorbas [1988]); taking these into account can either simplify or complicate the job of the compiler. The Tera computer system (Alverson et al. [1990]) supports parallelism at many granularities (Alverson et al. [1992]).

Delivering the performance for parallel applications often requires tuning the program, which really means manual intervention. Performance analysis tools, such as Mtool (Goldberg and Hennessy [1993]) can help find critical parts of the program that need tuning.

An alternative approach to automatic parallelization is to use pattern matching, as described by Keßler and Paul (1993); such a scheme can restructure the program in ways that seem to violate data dependence relations since the patterns can incorporate semantic information. A portable yet effective parallelizer for Pascal is described by Gabber, Averbuch, and Yehudai (1991).

Sequential consistency was introduced by Lamport (1979). Weaker memory consistency models are discussed by Adve and Hill (1990). Release consistency is introduced by Gharachorloo et al. (1990) and is used in the Stanford DASH prototype multiprocessor.

12 Vector Analysis

Vectorization was the earliest method of automatically finding parallelism in sequential loops (Schneck [1972]). The Parafrase project at the University of Illinois (Kuck et al. [1980]) and the PFC project at Rice University (Allen [1987]) were two of the earliest projects to use data dependence theory to find vector-style parallelism from sequential loops.

Allen and Kennedy (1992) discuss strip mining and optimizations to enhance vector register locality.

Callahan, Dongarra, and Levine (1988) describe a small benchmark of 100 short loops that were used to test the automatic vectorization capabilities of vectorizing compilers in 1988; this benchmark was used for several years thereafter as a marketing tool by the vendors.

As mentioned earlier, choosing an order for the transformations is difficult. Bose (1988) describes an interactive aid for program development and optimization, aimed at the IBM 3090 Vector Facility.

The influence of optimizing for stride on a cached vector machine is described by Erbacci, Paruolo, and Tagliavini (1989).

Tanaka et al. (1988, 1990) use loop unrolling for loops with recurrences on a vector computer with hardware support for first-order linear recurrences to reduce the recurrence length and improve the execution time.

Tsuda and Kunieda (1988, 1990) describe a vectorizing compiler that makes extensive use of indirect addressing after loop coalescing to vectorize nested loops. Tsuda, Kunieda, and Atipas (1989) show the results of applying vectorizing compiler technology to character string and database `join` operations.

The scheme presented here for vectorizing loops with conditionals where the condition is in a dependence cycle is taken from Banerjee and Gajski (1984).

13 Message Passing Machines

Early message-passing multicomputers are summarized by Athas and Seitz (1988).

Much of the early work on automatic compilation for message-passing machines was a result of the Suprenum project in Germany in the 1980s. Gerndt and Zima (1987) and Zima, Bast, and Gerndt (1988) describe an interactive software tool called SUPERB for converting sequential programs for parallel execution. Gerndt (1989, 1990) describes the array distribution mechanism in SUPERB, including the concept of an overlap region. Kremer et al. (1988) discuss some of the problems that arise from the interactive nature. Gerndt (1991) discusses the methods used in SUPERB to distribute the iterations of parallel loops according to the data layout. Zima and Chapman (1993) give a good overview of compiler technology for message-passing systems. The collection edited by Saltz and Mehrotra (1992) contains a number of articles about programming languages and compilers for message-passing machines.

The owner-computes rule was first described by Rogers and Pingali (1989, 1994) and Pingali and Rogers (1990), who implemented one of the first global compilers for a commercial message-passing machine. Koelbel, Mehrotra, and Rosendale (1987a, 1987b) addressed simple nearest-neighbor problems in the context of a simplified Pascal-like language, Blaze. Handling nearest-neighbor communication for distributed data structures is sufficient for some applications (Koelbel, Mehrotra, and Rosendale [1990]). More early work is described by Koelbel and Mehrotra (1991b, 1991a) using Kali, a language based on Fortran. Mehrotra and Rosendale (1990) discuss Kali language constructs and the Aspect/execution model used by the compiler. Koelbel et al. (1990, 1991) discuss automatic insertion of communication operations, and the resulting performance, for irregular and regular patterns. Gupta and Banerjee (1992b) show how to generate collective communication routines by analyzing the data movement required. Callahan and Kennedy (1988b) describe the design of a general scheme for code generation for distributed memory. Ruhl and Annaratone (1990) describe a scheme to compile annotated Fortran to a research message-passing system. André, Pazat, and Thomas (1990a, 1990b) discuss Pandore, a compiler that uses a model of distributed execution similar to that of High Performance Fortran. Philippsen and Tichy (1991) describe the compiler for Modula-2*, another language for writing structured highly parallel programs for shared or message-passing computer systems. Merlin (1991) describes the ADAPT system, which converts Fortran 90 array assignments automatically into message-passing code. Mehrotra and Rosendale (1990) discuss communication patterns and windows during which the communication must take place. Chen and Hu (1992) show how to generate parallel code for several categories of loops. Gupta and Schonberg (1993) describe analysis to minimize communication by reusing data that has already been sent, either to the same processor or to a processor nearer than the owner.

Chatterjee et al. (1993b) describe a method using finite-state machines to generate the local addresses of distributed data. Hiranandani et al. (1994) discuss this as well as other problems of compiling with general block-cyclic distributions. The scheme described here was first presented by Gupta et al. (1993).

One of the early SIMD machines was the Massively Parallel Processor, or MPP, installed in May 1983 at Goddard Space Flight Center; the machine and its software are described in the collection edited by Potter (1985). The MPP and other SIMD machines are summarized by Hord (1990).

Thinking Machines Corp. delivered one of the first compilers that allowed programs to be written in data-parallel style without writing node programs; the

Connection Machine Fortran compiler used extended Fortran 90 array assignments by including a `forall` statement, from which the compiler would generate parallel code (Albert, Lukas, and Steele [1990, 1991]). The early Thinking Machines compilers used the concept of virtual processors, supported by microcode, for large data sets. Christy (1991) argues that the compiler should deal with actual processors, not virtual processors, and this is how the compilers for the Thinking Machines Connection Machine evolved. Sabot (1992) discusses how the compiler eventually was migrated to support the MIMD Connection Machine CM-5. Bromley et al. (1991) describe compiler optimizations to generate code for the SIMD Connection Machine CM-2 that could run at 10 gigaflops; the programs were restricted to have regular stencil data accesses, and the compiler used overlap regions to reduce communication, and cleverly scheduled the PE code to take maximum advantage of the registers, another example of how optimizing for parallelism is not enough.

The design of High Performance Fortran borrowed much from other languages. The `template`, `align` and `distribute` statements are borrowed from Fortran D, a language and compiler developed at Rice University (Hiranandani, Kennedy, and Tseng [1992b]). Hiranandani et al. (1991a, 1991) and Tseng (1993) describe the design of the analysis and optimizations used in the Fortran D compiler. Hall et al. (1992) show how the Fortran D compiler uses interprocedural analysis to preserve the benefits of separate compilation; it also calculates reaching layouts for procedure arguments, clones procedures based on the layouts of their formal arguments, and computes the sizes of the overlap regions based on interprocedural information. Hiranandani, Kennedy, and Tseng (1993) describe some early experiences with the Fortran D compiler, showing that for some types of problems it approaches the performance of hand-optimized parallel code, while for other cases more optimizations need to be integrated. Hiranandani, Kennedy, and Tseng (1992a, 1994) show the relative enhancements due to particular optimizations in the Fortran D system, in particular the communication optimizations. Bozkus et al. (1993a, 1993b, 1994) describe related work on a compiler based on Fortran 90 array assignments, and Choudhary et al. (1992, 1993) describe how this compiler is integrated with the parallelizing Fortran D system.

Vienna Fortran, described by Chapman, Mehrotra, and Zima (1992), is related to High Performance Fortran, but includes some features not yet addressed by that language. Fahringer, Blasko, and Zima (1992, 1993) describe a performance prediction tool for the Vienna Fortran Compilation System, a compiler that generates message-passing code; the performance prediction tool is used both by the compiler system and by users to help with program transformation and data layout decisions. Chapman et al. (1993) describe the support of Vienna Fortran for dynamic distributions, including some of the compiler analysis necessary to optimize it. Chapman, Zima, and Mehrotra (1992) discuss the procedure interface for distributed data structures, including inheritance of the data layout and conditionals to dynamically test the actual data layout. Chapman, Mehrotra, and Zima (1993) describe an approach for data distribution used in Vienna Fortran that avoids the `template` construct. An extension to include Fortran 90 constructs is described by Benkner, Chapman, and Zima (1992).

Baber (1991) and Otto and Wolfe (1992) describe early precompilers aimed at many of the goals of High Performance Fortran. The programming model of these precompilers is that the user writes a node program using global indexing;

the node program is executed on each node, and parallel work is distributed over the nodes according to the array distribution. Another language design is given by Elustondo et al. (1992). Ikudome et al. (1990) discuss ASPAR, a simple parallelizer for C programs targeted at message-passing machines. Nichols, Siegel, and Dietz (1990, 1993) describe the compiler for ELP, an explicitly parallel language, and show a way to handle mono and poly data for SIMD or SPMD execution. Socha (1990) discusses a method for handling a class of data-parallel algorithms based on grids of data points. Chase et al. (1992) describe Paragon, a language and programming methodology for writing architecture-independent data-parallel applications. Grit (1990) describes an implementation of SISAL for a distributed memory machine.

Tseng (1990b, 1990a) shows compilation of loops for a simple array language onto a systolic array, essentially a message-passing machine with high bandwidth interconnections. Chen and Cowie (1992) describe the optimizations, including transformations of the parallel code, in a prototype Fortran 90 subset compiler for the SIMD CM/2 and MIMD CM/5 by Thinking Machines. Bala, Ferrante, and Carter (1993) give a compiler intermediate representation to facilitate optimization and communication operations.

Ponnusamy, Saltz, and Choudhary (1993) discuss ways to handle problems that are caused by mapping an irregular data structure onto a linear array; in particular, they discuss ways to interface a logical data partitioner to the compiler-generated code. They also discuss scheduling each iteration on a single processor (the owner of the most left-hand sides), rather than scheduling each statement of each iteration on the owner of its left-hand side. Agrawal, Sussman, and Saltz (1993) discuss methods to integrate support into a compiler for multiple communicating meshes, such as multigrid or coupled meshes. Data-flow analysis is used to insert inspector/executor communication for indirect array accesses by von Hanxleden et al. (1992). Das, Saltz, and von Hanxleden (1993) discuss corresponding methods to use when there are multiple levels of indirection. Berryman, Saltz, and Scroggs (1991) describe a set of subprograms to support and optimizate communication corresponding to unstructured meshes. Hiranandani et al. (1991b) describe the performance of a scheme that uses hash tables to access local copies of off-processor data.

Kung and Subhlok (1991) show the importance of block-cyclic data distributions for matrix operations. The divide operations needed for implementation of block-cyclic distributions can be done with multiplications, as shown by Granlund and Montgomery (1994).

Gallivan, Jalby, and Gannon (1988) discuss options for solving the problem of finding and transferring subregions of distributed arrays that are accessed locally. Amarasinghe and Lam (1993) describe more aggressive communication optimizations based on exact data dependence analysis.

Kennedy and Roth (1994) discuss loop fusion and index set splitting to optimize SIMD context computation. Weiss (1991) discusses generalized strip mining for SIMD machines to avoid global communication. The SIMD contraint that all processors must execute the same instruction does not necessarily mean that all processors must proceed through their local iteration sets in the same order; von Hanxleden and Kennedy (1992) describe compiler transformations to improve the asymptotic performance of SIMD machines. Several optimizations for SIMD machines are summarized by Knobe, Lukas, and Weiss (1993).

An alternative approach to compiling for a message-passing machine is to treat it more like a shared-memory machine; Rudolph and Polychronopoulos

(1989) describe an experimental system where iterations are scheduled onto the nodes of a message-passing system by guided self-scheduling, rather than by looking at the data layout. Su and Yew (1991) use a similar scheme using static scheduling, allowing flow dependence relations with constant distance to be resolved by direct messages from the processor that produces the value to the memory of the processor that will execute the iteration that consumes the value. Wu and Gajski (1988) show another programming scheme where the compiler generates a static schedule and inserts synchronization given the communication structure of the tasks.

Several research groups have looked at addressing the compilation problem by partitioning the iteration space (through tiling or other mappings) and using that to generate data partitions. Sheu and Tai (1991) show a method using linear transformations. Ramanujam and Sadayappan (1990a, 1990b, 1991a) study how to partition the iteration space to find communication-free partitions, or, failing that, use extreme vectors to partition the tiles into atomic units.

Automatic data alignment and distribution are subjects of current research. An optimal solution in a restricted domain—that of computation of a single expression in a simple Cartesian processor space with one element per processor—is shown by Gilbert and Schreiber (1991). Ramanujam and Sadayappan (1989) discuss constructing a data space dependence graph in addition to the iteration space dependence graph to help with automatic data partitioning. Li and Chen (1990a, 1991b) describe the axis alignment graph and methods to partition it. Li and Chen (1990c, 1991a) show a scheme where the data is automatically aligned using an axis alignment graph approach, where collective communication patterns are recognized, and where the choices of which dimensions to distribute and how are made based on the costs of the communication for each pattern. This work is based on a functional language, Crystal, but the techniques can be used in imperative languages (Li and Chen [1990a], Li [1991]). Knobe, Lukas, and Steele (1988, 1990) and Knobe and Natarajan (1990) describe a similar scheme using alignment preferences specifically targeted at SIMD machines. Balasundaram et al. (1990, 1991) propose an interactive tool that will guide the programmer in making data distribution decisions based on a compile-time performance estimator, where the estimator module is trained with a set of kernel routines. O'Boyle and Hedayat (1992) describe a scheme for automatic data alignment for affine array accesses by finding the Hermite form of the access matrix. Wholey (1992) uses a language with high-level primitive operations and describes a scheme to decide how to allocate machine dimensions to distributed array dimensions. Anderson and Lam (1993) try to automatically discover parallelism and an appropriate data layout by distinguishing arrays that can be simply decomposed from those that require communication or reorganization; code generation and related issues are presented by Amarasinghe et al. (1993). Gupta and Banerjee (1991, 1992a, 1993) use an axis alignment graph to align arrays, followed by analysis of how to distribute each aligned set of dimensions. Feautrier (1993) uses exact value-based dependence relations and finds a data layout that maximizes the number of dependence edges that are satisfied by scheduling the dependent iterations on the same processor. Chapman, Herbeck, and Zima (1991, 1993) discuss support for automatic data distribution in Vienna Fortran, which uses intraprocedural and interprocedural analysis allowing dynamic realignment. Efficient and effective solutions for restricted cases are presented by Sinharoy and Szymanski (1994). Chatterjee, Gilbert, and Schreiber (1993b) discuss *mobile alignments*, in particular when

data should be realigned between iterations of a loop; Chatterjee, Gilbert, and Schreiber (1993a, 1993a) describe the alignment-distribution graph, a graphical form of the flow of data used for automatic alignment. Snyder and Socha (1990) describe a scheme for finding balanced almost-rectangular data partitions.

Sabot and Wholey (1993) describe CMAX, a tool that automatically converts sequential programs to parallel programs for the Connection Machine; much of the analysis has to do with data layout and avoiding problems that arise from Fortran's storage and sequence association.

Burns, Kuhn, and Werme (1992) describe a *low-copy message-passing* scheme that eliminates much of the software and system overhead associated with message-passing systems; they note that additional synchronization between the system and the application may need to take place; for instance, to insure that the buffer is filled before using or was emptied before reusing. Hsu and Banerjee (1990) discuss the design of a coprocessor that will remove most of the cost of message-passing from the software. Active messages are presented by von Eicken et al. (1992) as a method to reduce the cost of communication by associating a software handler with each message. Thakur, Choudhary, and Fox (1994) discuss details of optimizing the communication that arises from redistributing distributed arrays. Holm, Lain, and Banerjee (1994) discuss code generation strategies to change a SPMD program, such as one resulting from the methods in this chapter, into code suitable for a multithreading or message-driven environment.

Saltz et al. (1990, 1991) propose that the communication latency is so high for message passing that the compiler should leave most of the scheduling and communication decisions until run-time; the argument is that the extra cost at run-time will pay off in lower communication time. Early development of this idea is described by Mirchandaney et al. (1988).

14 Scalable Shared Memory Machines

Markatos and LeBlanc (1991, 1992, 1994) show that the potential load imbalance of using affinity scheduling is outweighed by the improved cache locality, even with small numbers of processors, compared to guided self-scheduling, factoring and other scheduling methods. Squillante and Lazowska (1993) describe several scheduling policies and explore how they are affected by cache locality. Appelbe and Lakshmanan (1993, 1993) discuss the interaction of loop transformations with affinity scheduling. Li and Pingali (1992b) use loop restructuring to modify the loop so that array accesses in the inner loop have improved locality.

Cache memories and the three Cs classification of cache misses are described by Hennessy and Patterson (1990). Scalable distributed directory cache coherence hardware is described by Thapar and Delagi (1990). Several directory cache coherence protocols are summarized by Chaiken et al. (1990). Weber and Gupta (1989) argue that a directory scheme need only have a few pointers for each cache line, if the applications are appropriately well-written. Gupta, Weber, and Mowry (1990) show several directory schemes and evaluate them according to the directory storage and network traffic requirements.

Compiler-managed cache memories for large-scale multiprocessors have been heavily studied. Cytron, Karlovsky, and McAuliffe (1988) use data-flow analysis to find where to put cache control instructions. Owicki and Agarwal (1989) eval-

uate the performance of two simple software cache coherence schemes, finding them effective for large scalable systems. Veidenbaum (1986) studies a software-cache coherence scheme with three cache control instructions: invalidate the cache, turn the cache on, and turn the cache off. Cheong and Veidenbaum (1987) evaluate a simple write-through cache that is kept coherent by flushing the cache before and after each parallel loop. This has the unfortunate effect of losing locality between adjacent parallel loops. In a modification to this approach described by Cheong and Veidenbaum (1990b), the software and the cache maintain a version number for each variable; when an access to a variable hits in the cache but that variable has an old version number, the cache value is stale and the cache line must be reloaded. Since the granularity of the version numbers is one per variable, even if that variable is an array, there is still the potential for lots of unnecessary cache traffic. Several of these schemes are summarized by Cheong and Veidenbaum (1990a). A similar scheme is described by Min and Baer (1989, 1992). Min and Baer (1990) show through simulations that software-based cache coherence can achieve cache hit rates equivalent to directory-based methods with less network traffic.

Chen and Veidenbaum (1991a) describe a modified directory scheme to hold the sharing state of each cache line. In a subsequent paper Chen and Veidenbaum (1991b) compare simulated performance of software- and hardware-managed cache memories; the software cache was better at tolerating false sharing for writes. Cheong (1992) proposes a modified cache coherence protocol with up-to-date and soon-to-be-stale cache states identified by the software and kept in the cache. Darnell, Mellor-Crummey, and Kennedy (1992) discuss a scheme where `post` and `invalidate` instructions are inserted by the compiler for each memory reference and floated out of loops using vectorization techniques. Darnell and Kennedy (1993) discuss a hardware-assisted software cache coherence scheme, and show that it can achieve a hit ratio close to a hardware snoopy cache coherence protocol; this paper presents a nice summary of various software cache coherence schemes. Lilja and Yew (1993) show compiler optimizations to reduce the memory used by software cache coherence. Cheong and Veidenbaum (1988) describe flow analysis methods to minimize the size of the cached data that must be invalidated as stale.

Larus (1993) argues that the flexibility of compiler-managed caching can outweigh the performance advantages of hardware-managed caching. Adve et al. (1991) present experiments comparing hardware and software cache coherence, showing that software schemes are comparable to hardware schemes. Bennett, Carter, and Zwaenepoel (1990) present an adaptive software cache coherence strategy. Li and Sevcik (1994) show the performance of using software cache coherence with affinity scheduling.

Picano, Brooks, and Hoag (1991) argue the importance of alleviating the programmer from dealing with cache coherence. Ju and Dietz (1991) show an approach to optimize the program using loop transformations and changing the data layout to reduce cache coherence traffic.

Hudak and Abraham (1991) describe a scheme to partition elements of global arrays into four classes: the *exclusive read-write* (ERW) set, the *shared read-exclusive write* (SREW) set, the *shared read-no write* (SRNW) set, and the *no read-write* (NRW) set. Consistency needs to be maintained for the SREW set, but the ERW can be cached without penalty for interprocessor consistencies, and the SRNW set can be cached or duplicated. They also show through experiments on the BBN TC2000 that recognizing these sets can improve per-

formance significantly. Lilja and Yew (1991) and Mounes-Toussi, Lilja, and Li (1994) discuss using compiler algorithms to optimize the use of directory-based cache memories by determining which data locations or references might need coherence operations; read-only or private data need not suffer the penalty of coherence operations or even directory pointers. Abraham and Hudak (1991) use rectangular and hexagonal tiling to increase the cache locality of programs containing parallel loops nested in a sequential loop.

An alternative design for NUMA machines is to require the software to explicitly load the local cache, rather than using hardware cache miss detection. Granston and Veidenbaum (1991) discuss a method to detect redundant cache loads, to reduce the memory traffic. Lee, Yew, and Lawrie (1987) show a method that inserts prefetch instructions to alleviate cache latencies. Lilja (1994) notes that prefetching requires the program to know what iterations will be executed on a processor, affecting the scheduling strategy.

Culler et al. (1991) describe a scheme based on multithreading at the software level, which allows a processor to tolerate long latencies for messages or synchronization.

Abdelrahman and Wong (1994) use data placement and work distribution methods essentially equivalent to the techniques used in the preceding chapter for message-passing machines to manage locality on a NUMA machine; in their target machine, the local memory at each processor is part of the global shared-memory.

Smith et al. (1987) describe the ZS-1, the first commercial machine with decoupled access and execute processors. Bird, Rawsthorne, and Topham (1993) discusses the effectiveness of a compiler for such a machine, with an additional level of decoupling, adding a control processor.

Some NUMA systems are constructed from standard workstation-class components, with the operating system detecting and handling page faults for pages that are resident on another node. Li and Hudak (1989) discuss operating system methods to resolve page faults and coherence in such a system. Granston (1993) describes compiler analysis and optimizations to reduce page faults in such an environment.

References

Abdelrahman, Tarek S., and Thomas N. Wong (1994). Distributed Array Data Management on NUMA Multiprocessors. In *Scalable High Performance Computing Conference*, held in Knoxville, Tenn., November, 551–559.

Abraham, Santosh G., and David E. Hudak (1991). Compile-Time Partitioning of Iterative Parallel Loops to Reduce Cache Coherency Traffic. *IEEE Transactions on Parallel and Distributed Systems*, July, 2(3):318–328.

Abu-Sufah, Walid A. (1978). Improving the Performance of Virtual Memory Computers. Ph.D. Dissertation UIUCDCS-R-78-945, University of Illinois, Dept. Computer Science, November, (UMI 79-15307).

Abu-Sufah, Walid A., David J. Kuck, and Duncan H. Lawrie (1981). On the Performance Enhancement of Paging Systems Through Program Analysis and Transformations. *IEEE Transactions on Computers*, May, C-30(5):341–356.

Adams, Jeanne C., et al. (1992). *Fortran 90 Handbook*. New York: McGraw-Hill.

Adve, Sarita V., and Mark D. Hill (1990). Weak Ordering: A New Definition. In *17th International Symposium on Computer Architecture*, held in Seattle, Wash., May, 2–14.

Adve, Sarita V., et al. (1991). Comparison of Hardware and Software Cache Coherence Schemes. In *18th International Symposium on Computer Architecture*, held in Toronto, Ont., May, 298–308.

Agrawal, Gagan, Alan Sussman, and Joel H. Saltz (1993). Compiler and Runtime Support for Structured and Block Structured Applications. In *Supercomputing '93*, held in Portland, Ore., November, 578–587.

Aho, Alfred V., John E. Hopcroft, and Jeffrey D. Ullman (1974). *The Design and Analysis of Computer Algorithms*. Reading, Mass.: Addison-Wesley.

Aho, Alfred V., R. Sethi, and Jeffrey D. Ullman (1986). *Compilers: Principles, Techniques, and Tools*. Reading, Mass.: Addison-Wesley.

Albert, Eugene, Joan D. Lukas, and Guy L. Steele, Jr. (1990). Data Parallel Computers and the Forall Statement. In *Third Symposium on the Frontiers of Massively Parallel Computation*, held in College Park, Md., October, 390–396.

——— (1991). Data Parallel Computers and the Forall Statement. *Journal of Parallel and Distributed Computing*, October, 13(2):185–192.

Albert, Eugene, et al. (1988). Compiling Fortran 8x Array Features for the Connection Machine Computer System. In *ACM/SIGPLAN PPEALS, Parallel Programming: Experience with Applications, Languages and Systems*, held in New Haven, Conn., July, 42–56.

Allen, Frances E., and John Cocke (1972). A Catalogue of Optimizing Transformations. In *Design and Optimization of Compilers*, R. Rustin, ed. Englewood Cliffs, N.J.: Prentice-Hall, 1–30.

Allen, Frances E., John Cocke, and Ken Kennedy (1981). Reduction of Operator Strength. In *Program Flow Analysis: Theory and Applications*, Steven S. Muchnick and Neil D. Jones, eds. Englewood Cliffs, N.J.: Prentice-Hall, 79–101.

Allen, Frances E., et al. (1987). An Overview of the PTRAN Analysis System for Multiprocessing. In *Supercomputing: 1st International Conference*, held in Athens, Greece, number 297 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 194–211.

——— (1988a). An Overview of the PTRAN Analysis System for Multiprocessing. *Journal of Parallel and Distributed Computing*, October, 5(5):617–640.

——— (1988b). A Framework for Determining Useful Parallelism. In *1988 ACM International Conference on Supercomputing*, held in St. Malo, France, June, 207–215.

Allen, J. Randy (1983). Dependence Analysis for Subscripted Variables and Its Application to Program Transformations. Ph.D. Dissertation, Rice University, Dept. Mathematical Sciences, April, (UMI 83-14916).

——— (1988). Unifying Vectorization, Parallelization and Optimization: The Ardent Compiler. In *Third International Conference on Supercomputing*, vol. 2, 176–185.

Allen, J. Randy, David Callahan, and Ken Kennedy (1987). Automatic Decomposition of Scientific Programs for Parallel Execution. In *14th Annual ACM Symposium on Principles of Programming Languages*, held in Munich, Germany, January, 63–76.

Allen, J. Randy, and Steve Johnson (1988). Compiling C for Vectorization, Parallelization and Inline Expansion. In *ACM SIGPLAN '88 Conference on Programming Language Design and Implementation*, held in White Plains, N.Y., June, 241–249.

Allen, J. Randy, and Ken Kennedy (1984a). PFC: A Program to Convert Fortran to Parallel Form. In *Supercomputers: Design and Applications*, Kai Hwang, ed. Washington, D.C.: IEEE Computer Society Press, 186–203.

——— (1984b). Automatic Loop Interchange. In *ACM SIGPLAN '84 Symposium on Compiler Construction*, held in Montreal, June, 233–246.

——— (1985). A Parallel Programming Environment. *IEEE Software*, July, 2(4):21–29.

——— (1987). Automatic Translation of Fortran Programs to Vector Form. *ACM Transactions on Programming Languages and Systems*, October, 9(4):491–542.

——— (1992). Vector Register Allocation. *IEEE Transactions on Computers*, October, 41(10):1290–1317.

Allen, J. Randy, et al. (1983). Conversion of Control Dependence to Data Dependence. In *10th Annual ACM Symposium on Principles of Programming Languages*, held in Austin, Tex., January, 177–189.

——— (1986). PTOOL: A Semi-Automatic Parallel Programming Assistant. In *1986 International Conference on Parallel Processing*, held in St. Charles, Ill., August, 164–170.

Alverson, Gail, et al. (1992). Exploiting Heterogeneous Parallelism on a Multithreaded Multiprocessor. In *1992 ACM International Conference on Supercomputing*, held in Washington, D.C., July, 188–197.

Alverson, Robert, et al. (1990). The Tera Computer System. In *1990 ACM International Conference on Supercomputing*, held in Amsterdam, The Netherlands, June, 1–6.

Amarasinghe, Saman P., and Monica S. Lam (1993). Communication Optimization and Code Generation for Distributed Memory Machines. In *ACM SIGPLAN '93 Conference on Programming Language Design and Implementation*, held in Albuquerque, N.M., June, 126–138.

Amarasinghe, Saman P., et al. (1993). An Overview of a Compiler for Scalable Parallel Machines. In *1993 Workshop on Languages and Compilers for Parallel Computing*, held in Portland, Ore., August, number 768 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 253–272.

Ammarguellat, Zahira, and W. L. Harrison, III (1990). Automatic Recognition of Induction Variables and Recurrence Relations by Abstract Interpretation. In *ACM SIGPLAN '90 Conference on Programming Language Design and Implementation*, held in White Plains, N.Y., June, 283–295.

Ancourt, Corinne, and François Irigoin (1991). Scanning Polyhedre with DO Loops. In *3rd ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, held in Williamsburg, Va., April, 39–50.

Anderson, Jennifer M., and Monica S. Lam (1993). Global Optimizations for Parallelism and Locality on Scalable Parallel Machines. In *ACM SIGPLAN '93 Conference on Programming Language Design and Implementation*, held in Albuquerque, N.M., June, 112–125.

Anderson, S., and Paul Hudak (1990). Compilation of Haskell Array Comprehensions for Scientific Computing. In *ACM SIGPLAN '90 Conference on Programming Language Design and Implementation*, held in White Plains, N.Y., June, 137–149.

Anderson, Thomas E. (1990). The Performance of Spin Lock Alternatives for Shared-Memory Multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, January, 1(1):6–16.

André, Françoise, Jean-Louis Pazat, and Henry Thomas (1990a). Pandore: A System to Manage Data Distribution. In *1990 ACM International Conference on Supercomputing*, held in Amsterdam, The Netherlands, June, 380–388.

——— (1990b). Data Distribution in Pandore. In *Fifth Distributed Memory Computing Conference*, held in Charleston, S.C., April, 1115–1119.

Appelbe, William F., Charles Hardnett, and Sri Doddapaneni (1993). Program Transformation for Locality Using Affinity Regions. In *1993 Workshop on Languages and Compilers for Parallel Computing*, held in Portland, Ore., August, number 768 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 290–300.

Appelbe, William F., and Balakrishnan Lakshmanan (1993). Optimizing Parallel Programs Using Affinity Regions. In *1993 International Conference on Parallel Processing*, vol. II, held in St. Charles, Ill., August, 246–249.

Appelbe, William F., and Kevin Smith (1992). Determining Transformation Sequences for Loop Parallelization. In *1992 Workshop on Languages and Compilers for Parallel Computing*, held in New Haven, Conn., August, number 757 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 208–222.

Appelbe, William F., Kevin Smith, and Charlie McDowell (1989). Start/Pat: A Parallel Programming Toolkit. *IEEE Software*, July, 6(4):29–38.

Arnold, Clifford N. (1983). Vector Optimization on the Cyber 205. In *1983 International Conference on Parallel Processing*, held in St. Charles, Ill., August, 530–536.

Atapattu, Daya, and Dennis Gannon (1989). Building Analytical Models into an Interactive Performance Prediction Tool. In *Supercomputing '89*, held in Reno, Nev., November, 521–530.

Athas, William C., and Charles L. Seitz (1988). Multicomputers: Message-Passing Concurrent Computers. *IEEE Computer*, August, 21(8):9–24.

Ayguadé, E., and Jordi Torres (1993). Partitioning the Statement per Iteration Space Using Nonsingular Matrices. In *1993 ACM International Conference on Supercomputing*, held in Tokyo, July, 407–415.

Ayguadé, E., et al. (1989). GTS: Parallelization and Vectorization of Tight Recurrences. In *Supercomputing '89*, held in Reno, Nev., November, 531–539.

——— (1990). Parallelism Evaluation and Partitioning of Nested Loops for Shared Memory Multiprocessors. In *Advances in Languages and Compilers for Parallel Computing, 1990 Workshop*, held in Irvine, Calif., August, Research Monographs in Parallel and Distributed Computing, Cambridge, Mass.: MIT Press, 220–242.

Babb, Robert G., II, ed. (1988). *Programming Parallel Processors*. Reading, Mass.: Addison-Wesley.

Baber, Marc (1991). Hypertasking Support for Dynamically Redistributable and Resizeable Arrays on the iPSC. In *Sixth Distributed Memory Computing Conference*, held in Portland, Ore., April, 59–66.

Backus, John (1981). The History of Fortran I, II and III. In *History of Programming Languages*, Richard L. Wexelblat, ed. New York: Academic Press, 25–74.

Bala, Vasanth, Jeanne Ferrante, and Larry Carter (1993). Explicit Data Placement (XDP): A Methodology for Explicit Compile-Time Representation and Optimization of Data Movement. In *4th ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, held in San Diego, Calif., May, 139–148.

Balasundaram, Vasanth (1990). A Mechanism for Keeping Useful Internal Information in Parallel Programming Tools: The Data Access Descriptor. *Journal of Parallel and Distributed Computing*, June, 9(2):154–170.

Balasundaram, Vasanth, and Ken Kennedy (1989a). Compile-time Detection of Race Conditions in a Parallel Program. In *1989 ACM International Conference on Supercomputing*, held in Crete, Greece, June, 175–185.

——— (1989b). A Technique for Summarizing Data Access and Its Use in Parallelism Exhancing Transformations. In *ACM SIGPLAN '89 Conference on Programming Language Design and Implementation*, held in Portland, Ore., June, 41–53.

Balasundaram, Vasanth, et al. (1989). The ParaScope Editor: An Interactive Parallel Programming Tool. In *Supercomputing '89*, held in Reno, Nev., November, 540–550.

——— (1990). An Interactive Environment for Data Partitioning and Distribution. In *Fifth Distributed Memory Computing Conference*, held in Charleston, S.C., April, 1160–1170.

——— (1991). A Static Performance Estimator to Guide Data Partitioning Decisions. In *3rd ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, held in Williamsburg, Va., April, 213–223.

Ballance, Robert A., Arthur B. Maccabe, and Karl J. Ottenstein (1990). The Program Dependence Web: A Representation Supporting Control-, Data-, and Demand-Driven Interpretation of Imperative Languages. In *ACM SIGPLAN '90 Conference on Programming Language Design and Implementation*, held in White Plains, N.Y., June, 257–271.

Banerjee, Utpal (1976). Data Dependence in Ordinary Programs. M.S. Thesis UIUCDCS-R-76-837, University of Illinois, Dept. Computer Science.

——— (1979). Speedup of Ordinary Programs. Ph.D. Dissertation, University of Illinois, Dept. Computer Science, October (UIMI 80-08967).

——— (1988a). *Dependence Analysis for Supercomputing*. Norwell, Mass.: Kluwer Academic Publishers.

——— (1988b). An Introduction to a Formal Theory of Dependence Analysis. *The Journal of Supercomputing*, October, 2(2):133–150.

——— (1989). A Theory of Loop Permutations. In *Languages and Compilers for Parallel Computing, 1989 Workshop*, held in Urbana, Ill., August, Research Monographs in Parallel and Distributed Computing, Cambridge, Mass.: MIT Press, 54–74.

——— (1990). Unimodular Transformations of Double Loops. In *Advances in Languages and Compilers for Parallel Computing, 1990 Workshop*, held in Irvine, Calif., August, Research Monographs in Parallel and Distributed Computing, Cambridge, Mass.: MIT Press, 192–219.

——— (1993). *Loop Transformations for Restructuring Compilers: The Foundations*. Norwell, Mass.: Kluwer Academic Publishers.

——— (1994). *Loop Parallelization*. Norwell, Mass.: Kluwer Academic Publishers.

Banerjee, Utpal, and Daniel D. Gajski (1984). Fast Execution of Loops with IF Statements. *IEEE Transactions on Computers*, November, C-33(11):1030–1033.

Banerjee, Utpal, et al. (1979). Time and Parallel Processor Bounds for Fortran-Like Loops. *IEEE Transactions on Computers*, September, C-28(9):660–670.

——— (1993). Automatic Program Parallelization. *Proceedings of the IEEE*, February, 81(2):211–243.

Barnes, George H., et al. (1968). The ILLIAC IV Computer. *IEEE Transactions on Computers*, August, C-17(8):746–757.

Barrett, Michael, and Christian Lengauer (1992). Uniformularity Considered Nonessential. In *CONPAR 92 – VAPP V: Joint International Conference on Vector and Parallel Processing*, held in Lyon, France, September, number 634 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 659–664.

Barth, Jeffrey M. (1977). An Interprocedural Data Flow Analysis Algorithm. In *4th ACM Symposium on Principles of Programming Languages*, held in Los Angeles, January, 119–131.

——— (1978). A Practical Interprocedural Data Flow Analysis Algorithm. *Communications of the ACM*, September, 21(9):724–736.

Baxter, W., and H. R. Bauer, III (1989). The Program Dependence Graph and Vectorization. In *16th Annual ACM Symposium on Principles of Programming Languages*, held in Austin, Tex., January, 1–11.

Beck, Micah, Richard Johnson, and Keshav Pingali (1991). From Control Flow to Dataflow. *Journal of Parallel and Distributed Computing*, June, 12(2):118–129.

Beck, Micah, and Keshav Pingali (1990). From Control Flow to Dataflow. In *1990 International Conference on Parallel Processing*, vol. II, held in St. Charles, Ill., August, 43–53.

Behr, P. M., W. K. Giloi, and H. Múhlenbein (1986). Suprenum: The German Supercomputer Architecture: Rationale and Concepts. In *1986 International Conference on Parallel Processing*, held in St. Charles, Ill., August, 567–575.

Beltrametti, Monica, Kenneth Bobey, and John R. Zorbas (1988). The Control Mechanism for the Myrias Parallel Computer System. *Computer Architecture News*, September, 16(4):21–30.

Benkner, Siegfried, Barbara M. Chapman, and Hans P. Zima (1992). Vienna Fortran 90. In *Scalable High Performance Computing Conference*, held in Williamsburg, Va., April, 51–59.

Bennett, John K., John B. Carter, and Willy Zwaenepoel (1990). Adaptive Software Cache Management for Distributed Shared Memory Architectures. In *17th International Symposium on Computer Architecture*, held in Seattle, Wash., May, 125–134.

Bernstein, A. J. (1966). Analysis of Programs for Parallel Processing. *IEEE Transactions on Electronic Computers*, October, 15(5).

Berryman, Harry, Joel H. Saltz, and Jeffrey Scroggs (1991). Execution Time Support for Adaptive Scientific Algorithms on Distributed Memory Machines. *Concurrency: Practice & Experience*, June, 3(3):159–178.

Bik, Aart J. C., and Harry A. G. Wijshoff (1993a). Compilation Techniques for Sparse Matrix Computations. In *1993 ACM International Conference on Supercomputing*, held in Tokyo, July, 416–424.

——— (1993b). On Automatic Data Structure Selection and Code Generation for Sparse Computations. In *1993 Workshop on Languages and Compilers for Parallel Computing*, held in Portland, Ore., August, number 768 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 57–75.

——— (1993c). Advanced Compiler Optimizations for Sparse Computations. In *Supercomputing '93*, held in Portland, Ore., November, 430–439.

Bird, Peter L., Alasdair Rawsthorne, and Nigel P. Topham (1993). The Effectiveness of Decoupling. In *1993 ACM International Conference on Supercomputing*, held in Tokyo, July, 47–56.

Bodin, François, et al. (1992). A Quantitative Algorithm for Data Locality Optimization. In *Code Generation: Concepts, Tools, Techniques*, Robert Giegerich and Susan Graham, eds. Berlin: Springer Verlag, 119–145.

Bose, Pradip (1988). Interactive Program Improvement via EAVE: An Expert Advisor for Vectorization. In *1988 ACM International Conference on Supercomputing*, held in St. Malo, France, June, 119–130.

Bouvet, Pierre, et al. (1994). (Per)-Ultimate Tuning. In *Scalable High Performance Computing Conference*, held in Knoxville, Tenn., November, 568–576.

Bozkus, Zeki, et al. (1993a). A Compilation Approach for Fortran 90D/HPF Compilers. In *1993 Workshop on Languages and Compilers for Parallel Computing*, held in Portland, Ore., August, number 768 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 200–215.

——— (1993b). Fortran 90D/HPF Compiler for Distributed Memory MIMD Computers: Design, Implementation, and Performance Results. In *Supercomputing '93*, held in Portland, Ore., November, 351–360.

——— (1994). Compiling Fortran 90D/HPF for Distributed Memory MIMD Computers. *Journal of Parallel and Distributed Computing*, April, 21(1):15–26.

Bradley, G. H. (1970). Algorithm and Bound for the Greatest Common Divisor of N Integers. *Communications of the ACM*, July, 13(7):433–436.

Brandes, Thomas (1988). The Importance of Direct Dependences for Automatic Parallelization. In *1988 ACM International Conference on Supercomputing*, held in St. Malo, France, June, 407–417.

Brandes, Thomas, and Manfred Sommer (1987a). Realization of a Knowledge-Based Parallelization Tool in a Programming Environment. In *Supercomputing: 1st International Conference*, held in Athens, Greece, number 297 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 338–354.

——— (1987b). A Knowledge-Based Parallelization Tool in a Programming Environment. In *1987 International Conference on Parallel Processing*, held in St. Charles, Ill., August, 446–448.

Bromley, Mark, et al. (1991). Fortran at Ten Gigaflops: The Connection Machine Convolution Compiler. In *ACM SIGPLAN '91 Conference on Programming Language Design and Implementation*, held in Toronto, Ont., June, 145–156.

Burke, Michael, and Jong-Deok Choi (1992). Precise and Efficient Integration of Interprocedural Alias Information into Data-Flow Analysis. *ACM Letters on Programming Languages and Systems*, March, 1(1):14–21.

Burke, Michael, and Ron Cytron (1986). Interprocedural Dependence Analysis and Parallelization. In *ACM SIGPLAN '86 Symposium on Compiler Construction*, held in Palo Alto, Calif., June, 162–175.

Burke, Michael, et al. (1988). Automatic Discovery of Parallelism: A Tool and an Experiment. In *ACM/SIGPLAN PPEALS, Parallel Programming: Experience with Applications, Languages and Systems*, held in New Haven, Conn., July, 77–84.

——— (1989). Automatic Generated of Nested, Fork-Join Parallelism. *The Journal of Supercomputing*, July, 3(2):71–88.

Burns, Charles M., Robert H. Kuhn, and Eric J. Werme (1992). Low Copy Message Passing on the Alliant CAMPUS/800. In *Supercomputing '92*, held in Minneapolis, Minn., November, 760–769.

Byler, Mark, et al. (1988). Multiple Version Loops. In *1987 International Conference on Parallel Processing*, held in St. Charles, Ill., August, 312–318.

Callahan, David (1988). The Program Summary Graph and Flow-sensitive Interprocedural Data Flow Analysis. In *ACM SIGPLAN '88 Conference on Programming Language Design and Implementation*, held in White Plains, N.Y., June, 47–56.

——— (1991). Recognizing and Parallelizing Bounded Recurrences. In *1991 Workshop on Languages and Compilers for Parallel Computing*, held in Santa Clara, Calif., August, number 589 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 169–185.

Callahan, David, Jack Dongarra, and David Levine (1988). Vectorizing Compilers: A Test Suite and Results. In *Supercomputing '88*, held in Orlando, Fla., November, 98–105.

Callahan, David, and Ken Kennedy (1987). Analysis of Interprocedural Side Effects in a Parallel Programming Environment. In *Supercomputing: 1st International Conference*, held in Athens, Greece, number 297 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 138–171.

——— (1988a). Analysis of Interprocedural Side Effects in a Parallel Programming Environment. *Journal of Parallel and Distributed Computing*, October, 5(5):517–550.

——— (1988b). Compiling Programs for Distributed-Memory Multiprocessors. *The Journal of Supercomputing*, October, 2(2):151–169.

Callahan, David, Ken Kennedy, and Allan Porterfield (1991). Software Prefetching. In *4th International Conference on Architectural Support for Programming Languages and Operating Systems*, held in Santa Clara, Calif., April, 40–52.

Callahan, David, Ken Kennedy, and Jaspal Subhlok (1993). Analysis of Event Synchronization in a Parallel Programming Tool. In *4th ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, held in San Diego, Calif., May, 21–30.

Callahan, David, and Allan Porterfield (1990). Data Cache Performance of Supercomputer Applications. In *Supercomputing '90*, held in New York, November, 564–572.

Callahan, David, et al. (1986). Interprocedural Constant Propagation. In *ACM SIGPLAN '86 Symposium on Compiler Construction*, held in Palo Alto, Calif., June, 152–161.

——— (1988). ParaScope: A Parallel Programming Environment. *International Journal of Supercomputer Applications*, Winter, 2(4):84–99.

——— (1987). Constructing the Procedure Call Multigraph. *IEEE Transactions on Software Engineering*, April, 16(4):483–487.

Cann, David C. (1991). Retire Fortran? A Debate Rekindled. In *Supercomputing '91*, held in Albuquerque, N.M., November, 264–272.

Cann, David (1992). Retire Fortran? A Debate Rekindled. *Communications of the ACM*, August, 35(8):81–89.

Cann, David C., and John T. Feo (1990). SISAL versus Fortran: A Comparison Using the Livermore Loops. In *Supercomputing '90*, held in New York, November, 626–636.

Carle, Alan, et al. (1987). A Practical Environment for Scientific Programming. *IEEE Computer*, November, 20(11):75–89.

Carr, Steve, and Ken Kennedy (1989). Blocking Linear Algebra Codes for Memory Hierarchies. In *Fourth SIAM Conference on Parallel Processing for Scientific Computing*, held in Chicago, December, 400–405.

——— (1992). Compiler Blockability of Numerical Algorithms. In *Supercomputing '92*, held in Minneapolis, Minn., November, 114–124.

Cartwright, Robert, and Matthias Felleisen (1989). The Semantics of Program Dependence. In *ACM SIGPLAN '89 Conference on Programming Language Design and Implementation*, held in Portland, Ore., June, 13–27.

Chaiken, David, et al. (1990). Directory-Based Cache Coherence in Large-Scale Multiprocessors. *IEEE Computer*, June, 23(6):49–58.

Chapman, Barbara M., Thomas Fahringer, and Hans P. Zima (1993). Automatic Support for Data Distribution on Distributed Memory Multiprocessor Systems. In *1993 Workshop on Languages and Compilers for Parallel Computing*, held in Portland, Ore., August, number 768 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 184–199.

Chapman, Barbara M., Heinz Herbeck, and Hans P. Zima (1991). Automatic Support for Data Distribution. In *Sixth Distributed Memory Computing Conference*, held in Portland, Ore., April, 51–58.

Chapman, Barbara M., Piyush Mehrotra, and Hans P. Zima (1992). Programming in Vienna Fortran. *Scientific Programming*, 1(1):31–50.

——— (1993). High Performance Fortran without Templates: An Alternative Model for Distribution and Alignment. In *4th ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, held in San Diego, Calif., May, 92–101.

Chapman, Barbara M., Hans P. Zima, and Piyush Mehrotra (1992). Handling Distributed Data in Vienna Fortran Procedures. In *1992 Workshop on Languages and Compilers for Parallel Computing*, held in New Haven, Conn., August, number 757 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 244–263.

Chapman, Barbara M., et al. (1993). Dynamic Data Distributions in Vienna Fortran. In *Supercomputing '93*, held in Portland, Ore., November, 284–293.

Chase, Craig M., et al. (1992). Paragon: A Parallel Programming Environment for Scientific Applications Using Communication Structures. *Journal of Parallel and Distributed Computing*, October, 16(2):79–91.

Chase, David R., Mark Wegman, and F. Kenneth Zadeck (1990). Analysis of Pointers and Structures. In *ACM SIGPLAN '90 Conference on Programming Language Design and Implementation*, held in White Plains, N.Y., June, 296–310.

Chatterjee, Siddhartha, John R. Gilbert, and Robert Schreiber (1993a). The Alignment-Distribution Graph. In *1993 Workshop on Languages and Compilers for Parallel Computing*, held in Portland, Ore., August, number 768 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 234–252.

——— (1993b). Mobile and Replicated Alignment of Arrays in Data-Parallel Programs. In *Supercomputing '93*, held in Portland, Ore., November, 420–429.

Chatterjee, Siddhartha, et al. (1993a). Automatic Array Alignment in Data-Parallel Programs. In *20th Annual ACM Symposium on Principles of Programming Languages*, held in Charleston, S.C., January, 260–272.

——— (1993b). Generating Local Addresses and Communication Sets for Data-Parallel Programs. In *4th ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, held in San Diego, Calif., May, 149–158.

Chen, Ding-Kai, and Pen-Chung Yew (1991). An Empirical Study on DoAcross Loops. In *Supercomputing '91*, held in Albuquerque, N.M., November, 620–632.

Chen, Marina C. (1986a). A Design Methodology for Synthesizing Parallel Algorithms and Architectures. *Journal of Parallel and Distributed Computing*, December, 3(4):461–491.

——— (1986b). A Parallel Language and Its Compilation to Multiprocessor Machines or VLSI. In *13th Annual ACM Symposium on Principles of Programming Languages*, held in St. Petersburg Beach, Fla., January, 131–139.

Chen, Marina C., Young-Il Choo, and Jingke Li (1988). Compiling Parallel Programs by Optimizing Performance. *The Journal of Supercomputing*, October, 2(2):171–207.

Chen, Marina C., and James Cowie (1992). Prototyping Fortran-90 Compilers for Massively Parallel Machines. In *ACM SIGPLAN '92 Conference on Programming Language Design and Implementation*, held in San Francisco, Calif., June, 94–105.

Chen, Marina C., and Y. Hu (1992). Compiler Optimizations for Massively Parallel Machines: Transformations on Iterative Spatial Loops. In *1992 Workshop on Languages and Compilers for Parallel Computing*, held in New Haven, Conn., August, number 757 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 223–247.

Chen, Yung-Chin, and Alexander V. Veidenbaum (1991a). A Software Coherence Scheme with the Assistance of Directories. In *1991 ACM International Conference on Supercomputing*, held in Cologne, Germany, June, 284–294.

——— (1991b). Comparison and Analysis of Software and Directory Coherence Schemes. In *Supercomputing '91*, held in Albuquerque, N.M., November, 818–829.

Chen, Zhiyuan, and Chuan Chi Chang (1987). Iterative Level Parallel Execution of DO Loops with a Reduced Set of Dependence Relations. *Journal of Parallel and Distributed Computing*, October, 4(5):488–504.

Cheong, Hoichi (1992). Life Span Strategy: A Compiler-Based Approach to Cache Coherence. In *1992 ACM International Conference on Supercomputing*, held in Washington, D.C., July, 139–148.

Cheong, Hoichi, and Alexander V. Veidenbaum (1987). The Performance of Software-Managed Multiprocessor Caches on Parallel Numerical Programs. In *Supercomputing: 1st International Conference*, held in Athens, Greece, number 297 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 316–337.

——— (1988). Stale Data Detection and Coherence Enforcement Using Flow Analysis. In *1988 International Conference on Parallel Processing*, vol. I, held in St. Charles, Ill., August, 138–145.

——— (1990a). Compiler-Directed Cache Management in Multiprocessors. *IEEE Computer*, June, 23(6):39–47.

——— (1990b). A Version Control Approach to Cache Coherence. In *1990 ACM International Conference on Supercomputing*, held in Amsterdam, The Netherlands, June, 322–330.

Choi, Jong-Deok, Michael Burke, and Paul Carini (1993). Efficient Flow-Sensitive Interprocedural Computation of Pointer-Induced Aliases and Side Effects. In *20th Annual ACM Symposium on Principles of Programming Languages*, held in Charleston, S.C., January, 232–245.

Choi, Jong-Deok, Ron Cytron, and Jeanne Ferrante (1991). Automatic Construction of Sparse Data Flow Evaluation Graphs. In *18th Annual ACM Symposium on Principles of Programming Languages*, held in Orlando, Fla., January, 55–66.

——— (1994). On the Efficient Engineering of Ambitious Program Analysis. *IEEE Transactions on Software Engineering*, 20(2).

Choudhary, Alok, et al. (1992). Compiling Fortran 77D and 90D for MIMD Distributed-Memory Machines. In *Fourth Symposium on the Frontiers of Massively Parallel Computation*, held in McLean, Va., October. Washington, D.C.: IEEE Computer Society Press, 4–11.

——— (1993). Unified Compilation of Fortran 77D and 90D. *ACM Letters on Programming Languages and Systems*, 2:95–114.

Christy, Peter (1990). Virtual Processors Considered Harmful. In *Sixth Distributed Memory Computing Conference*, held in Portland, Ore., April, 99–103.

Cohagan, William L. (1973). Vector Optimization for the ASC. In *Seventh Annual Princeton Conf. on Information Sciences and Systems*, held in Princeton, N.J., March, 169–174.

Coleman, Howard B. (1987). The Vectorizing Compiler for the Unisys ISP. In *1987 International Conference on Parallel Processing*, held in St. Charles, Ill., August, 567–576.

Collard, Jean-François (1994). Space-Time Transformation of While-Loops Using Speculative Execution. In *Scalable High Performance Computing Conference*, held in Knoxville, Tenn., November, 429–436.

Cooper, Keith D. (1985). Analyzing Aliases of Reference Formal Parameters. In *12th Annual ACM Symposium on Principles of Programming Languages*, held in New Orleans, La., January, 281–290.

Cooper, Keith D., Mary W. Hall, and Ken Kennedy (1990). Procedure Cloning. In *1992 International Conference on Computer Languages*, held in Oakland, Calif., March, 96–105.

Cooper, Keith D., Mary W. Hall, and Linda Torczon (1992). Unexpected Side Effects of Inline Substitution: A Case Study. *ACM Letters on Programming Languages and Systems*, March, 1(1):22–32.

——— (1991). An Experiment with Inline Substitution. *Software: Practice & Experience*, June, 21(6):581–601.

Cooper, Keith D., and Ken Kennedy (1984). Efficient Computation of Flow Insensitive Interprocedural Summary Information. In *ACM SIGPLAN '84 Symposium on Compiler Construction*, held in Montreal, June, 247–258.

——— (1988). Interprocedural Side-Effect Analysis in Linear Time. In *ACM SIGPLAN '88 Conference on Programming Language Design and Implementation*, held in White Plains, N.Y., June, 57–66.

——— (1989). Fast Interprocedural Alias Analysis. In *16th Annual ACM Symposium on Principles of Programming Languages*, held in Austin, Tex., January, 49–59.

Cooper, Keith D., Ken Kennedy, and Linda Torczon (1986). The Impact of Interprocedural Analysis and Optimization in the R^n programming Environment. *ACM Transactions on Programming Languages and Systems*, October, 8(4):491–523.

Cooper, Keith D., et al. (1993). The ParaScope Parallel Programming Environment. *Proceedings of the IEEE*, February, 81(2):244–263.

Cormen, Thomas H., Charles E. Leiserson, and Ronald L. Rivest (1990). *The Design and Analysis of Computer Algorithms*. Cambridge, Mass.: MIT Press.

Culler, David E., et al. (1991). Fine-grain Parallelism with Minimal Hardware Support: A Compiler-Controlled Threaded Abstract Machine. In *4th International Conference on Architectural Support for Programming Languages and Operating Systems*, held in Santa Clara, Calif., April, 164–175.

Cytron, Ron (1984). Compile-Time Scheduling and Optimization for Asynchronous Machines. Ph.D. Dissertation UIUCDCS-R-84-1177, University of Illinois, Dept. Computer Science, October.

——— (1986). Doacross: Beyond Vectorization for Multiprocessors. In *1986 International Conference on Parallel Processing*, held in St. Charles, Ill., August, 836–844.

Cytron, Ron, and Jeanne Ferrante (1993). Efficiently Computing ϕ-Nodes on-the-Fly. In *1993 Workshop on Languages and Compilers for Parallel Computing*, held in Portland, Ore., August, number 768 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 461–476.

Cytron, Ron, Jeanne Ferrante, and Vivek Sarkar (1989). Experiences Using Control Dependences in PTRAN. In *Languages and Compilers for Parallel Computing, 1989 Workshop*, held in Urbana, Ill., August, Research Monographs in Parallel and Distributed Computing, Cambridge, Mass.: MIT Press, 186–212.

——— (1990). Compact Representations for Control Dependence. In *ACM SIGPLAN '90 Conference on Programming Language Design and Implementation*, held in White Plains, N.Y., June, 337–351.

Cytron, Ron, and Reid Gershbein (1993). Efficient Accommodation of May-Alias Information in SSA Form. In *ACM SIGPLAN '93 Conference on Programming Language Design and Implementation*, held in Albuquerque, N.M., June, 36–45.

Cytron, Ron, Michael Hind, and Wilson Hsieh (1989). Automatic Generation of DAG Parallelism. In *ACM SIGPLAN '89 Conference on Programming Language Design and Implementation*, held in Portland, Ore., June, 54–68.

Cytron, Ron, Steve Karlovsky, and Kevin P. McAuliffe (1988). Automatic Management of Programmable Caches. In *1988 International Conference on Parallel Processing*, held in St. Charles, Ill., August, 229–238.

Cytron, Ron, Jim Lipkis, and Edith Schonberg (1990). A Compiler-Assisted Approach to SPMD Execution. In *Supercomputing '90*, held in New York, November, 398–406.

Cytron, Ron, et al. (1989). An Efficient Method of Computing Static Single Assignment Form. In *16th Annual ACM Symposium on Principles of Programming Languages*, held in Austin, Tex., January, 25–35.

——— (1991). Efficiently Computing Static Single Assignment Form and the Control Dependence Graph. *ACM Transactions on Programming Languages and Systems*, October, 13(4):451–490.

Dantzig, George B., and B. Curtis Eaves (1973). Fourier-Motzkin Elimination and Its Dual. *Journal of Combinatorial Theory (A)*, 14:288–297.

Darema, F., et al. (1988). A Single-Program-Multiple-Data Computational Model for EPEX/Fortran. *Parallel Computing*, April, 7(1):11–24.

Darnell, Ervan, and Ken Kennedy (1992). Cache Coherence Using Local Knowledge. In *Supercomputing '93*, held in Portland, Ore., November, 720–729.

Darnell, Ervan, Johm M. Mellor-Crummey, and Ken Kennedy (1992). Automatic Software Cache Coherence Through Vectorization. In *1992 ACM International Conference on Supercomputing*, held in Washington, D.C., July, 129–138.

Das, Raja, Joel H. Saltz, and Reinhard von Hanxleden (1993). Slicing Analysis and Indirect Accesses to Distributed Arrays. In *1993 Workshop on Languages and Compilers for Parallel Computing*, held in Portland, Ore., August, number 768 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 152–168.

D'Hollander, Erik H. (1990). Partitioning and Labeling of Loops in Do Loops with Constant Dependence Vectors. In *1990 International Conference on Parallel Processing*, vol. II, held in St. Charles, Ill., August, 139–144.

——— (1992). Partitioning and Labeling of Loops by Unimodular Transformations. *IEEE Transactions on Parallel and Distributed Systems*, July, 3(4):465–476.

Di Antonio, Renzo, John Eilert, and Marcello Vitaletti (1989). Using Page-Ahead for Large Fortran Programs. In *Supercomputing '89*, held in Reno, Nev., November, 511–520.

Dietz, Henry G. (1988). Finding Large-Grain Parallelism in Loops with Serial Control Dependencies. In *1988 International Conference on Parallel Processing*, held in St. Charles, Ill., August, 114–121.

Dietz, Henry G., and David Klappholz (1986). Refined Fortran: Another Sequential Language for Parallel Programming. In *1986 International Conference on Parallel Processing*, held in St. Charles, Ill., August, 184–191.

Dinning, Anne, and Edith Schonberg (1990). An Empirical Comparison of Monitoring Algorithms for Access Anomaly Detection. In *2nd ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, held in Seattle, Wash., March, 1–10.

Dongarra, Jack J., et al. (1990). A set of Level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, March, 16(1):1–17.

Dowling, Michael L. (1990). Optimal Code Parallelization using Unimodular Transformations. *Parallel Computing*, December, 16(2):157–171.

Duesterwald, Evelyn, Rajiv Gupta, and Mary Lou Soffa (1993). A Practical Data Flow Framework for Array Reference Analysis and Its Use in Optimizations. In *ACM SIGPLAN '93 Conference on Programming Language Design and Implementation*, held in Albuquerque, N.M., June, 68–77.

Duffin, R. J. (1974). On Fourier's Analysis of Linear Inequality Systems. In *Mathematical Programming Study 1*. New York: North Holland, 71–95.

Edler, J., et al. (1985). Issues Related to MIMD Shared-Memory Computers: The NYU Ultracomputer Approach. In *12th International Symposium on Computer Architecture*, June, 126–135.

Eigenmann, Rudolf, and William Blume (1991). An Effectiveness Study of Parallelizing Compiler Techniques. In *1991 International Conference on Parallel Processing*, vol. II, held in St. Charles, Ill., August, 17–25.

Eigenmann, Rudolf, et al. (1990a). Cedar Fortran and Its Restructuring Compiler. In *CONPAR 90 – VAPP IV: Joint International Conference on Vector and Parallel Processing*, held in Zurich, Switzerland, September, number 457 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 288–299.

——— (1990b). Cedar Fortran and Its Restructuring Compiler. In *Advances in Languages and Compilers for Parallel Computing, 1990 Workshop*, held in Irvine, Calif., August, Research Monographs in Parallel and Distributed Computing, Cambridge, Mass.: MIT Press, 1–23.

——— (1991). Experience in the Automatic Parallelization of Four Perfect Benchmark Programs. In *12th Workshop on Languages and Compilers for Parallel Computing*, held in Santa Clara, Calif., August, number 589 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 65–83.

——— (1993). Restructuring Fortran Programs for Cedar. *Concurrency: Practice & Experience*, October, 5(7):553–573.

Eisenbeis, Christine, and Jean-Claude Sogno (1992). A General Algorithm for Data Dependence Analysis. In *1992 ACM International Conference on Supercomputing*, held in Washington, D.C., July, 292–302.

Eisenbeis, Christine, et al. (1990). A Strategy for Array Management in Local Memory. In *Advances in Languages and Compilers for Parallel Computing, 1990 Workshop*, held in Irvine, Calif., August, Research Monographs in Parallel and Distributed Computing, Cambridge, Mass.: MIT Press, 130–151.

Ellis, Margaret A., and Bjarne Stroustrup (1990). *The Annotated C++ Reference Manual*. Reading, Mass.: Addison-Wesley.

Elustondo, Pablo M., et al. (1992). Data Parallel Fortran. In *Fourth Symposium on the Frontiers of Massively Parallel Computation*, held in McLean, Va., October, Washington, D.C.: IEEE Computer Society Press, 21–28.

Emami, Maryam, Rakesh Ghiya, and Laurie J. Hendren (1994). Context-Sensitive Interprocedural Points-to Analysis in the Presence of Function Pointers. In *ACM SIGPLAN '94 Conference on Programming Language Design and Implementation*, held in Orlando, Fla., June, 242–256.

Emrath, Perry A., Sanjoy Ghosh, and David A. Padua (1989). Event Synchronization Analysis for Debugging Parallel Programs. In *Supercomputing '89*, held in Reno, Nev., November, 580–588.

Erbacci, Giovanni, Guiseppe Paruolo, and Giancarlo Tagliavini (1989). Influence of the Stride on the Cache Utilization in the IBM 3090 VF. In *1989 ACM International Conference on Supercomputing*, held in Crete, Greece, June, 443–451.

Fahringer, Thomas (1994). Using the P^{3}T to Guide the Parallelization and Optimization Effort Under the Vienna Fortran Compilation System. In *Scalable High Performance Computing Conference*, held in Knoxville, Tenn., November, 437–444.

Fahringer, Thomas, Roman Blasko, and Hans P. Zima (1992). Automatic Performance Prediction to Support Parallelization of Fortran Programs for Massively Parallel Systems. In *1992 ACM International Conference on Supercomputing*, held in Washington, D.C., July, 347–356.

Fahringer, Thomas, and Hans P. Zima (1993). A Static Parameter based Performance Prediction Tool for Parallel Programs. In *1993 ACM International Conference on Supercomputing*, held in Tokyo, July, 207–219.

Fang, Zhixi, and M. Lu (1991). An Iteration Partition Approach for Cache or Local Memory Thrashing on Parallel Processing. In *1991 Workshop on Languages and Compilers for Parallel Computing*, held in Santa Clara, Calif., August, number 589 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 313–327.

Fang, Zhixi (1990). Cache or Local Memory Thrashing and Compiler Strategy in Parallel Processing Systems. In *1990 International Conference on Parallel Processing*, vol. II, held in St. Charles, Ill., August, 271–275.

Fang, Zhixi, et al. (1990). Dynamic Processor Self-Scheduling for General Parallel Nested Loops. *IEEE Transactions on Computers*, July, 39(7):919–929.

Fateman, Richard J. (1982). High-Level Language Implications of the Proposed IEEE Floating-point Standard. *ACM Transactions on Programming Languages and Systems*, April, 4(2):239–257.

Feautrier, Paul (1988a). Parametric Integer Programming. *Operations Research*, 22(3):243–268.

——— (1988b). Array Expansion. In *1988 ACM International Conference on Supercomputing*, held in St. Malo, France, June, 429–441.

——— (1991). Dataflow Analysis of Array and Scalar References. *International Journal of Parallel Programming*, 20(1):23–54.

——— (1993). Toward Automatic Partitioning of Arrays for Distributed Memory Computers. In *1993 ACM International Conference on Supercomputing*, held in Tokyo, July, 175–184.

Feo, John T., David C. Cann, and Rodney R. Oldehoeft (1990). A Report on the Sisal Language Project. *Journal of Parallel and Distributed Computing*, December, 10(4):349–366.

Ferrante, Jeanne, and Mary Mace (1985). On Linearizing Parallel Code. In *12th Annual ACM Symposium on Principles of Programming Languages*, held in New Orleans, La., January, 179–190.

Ferrante, Jeanne, Mary Mace, and Barbara Simons (1988). Generating Sequential Code from Parallel Code. In *1988 ACM International Conference on Supercomputing*, held in St. Malo, France, June, 582–592.

Ferrante, Jeanne, and Karl J. Ottenstein (1983). A Program Form Based on Data Dependency in Predicate Regions. In *10th Annual ACM Symposium on Principles of Programming Languages*, held in Austin, Tex., January, 217–236.

Ferrante, Jeanne, Karl J. Ottenstein, and Joe D. Warren (1987). The Program Dependence Graph and Its Use in Optimization. *ACM Transactions on Programming Languages and Systems*, July, 9(3):319–349.

Ferrante, Jeanne, Vivek Sarkar, and Wendy Thrash (1991). On Estimating and Enhancing Cache Effectiveness. In *1991 Workshop on Languages and Compilers for Parallel Computing*, held in Santa Clara, Calif., August, number 589 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 328–343.

Firestone, Roger M. (1973). Loop Optimization: Array Use Analysis. In *Seventh Annual Princeton Conf. on Information Sciences and Systems*, held in Princeton, N.J., March, 298 (abstract only).

Fischer, Charles N., and Richard J. LeBlanc, Jr. (1988). *Crafting a Compiler*. Menlo Park, Calif.: Benjamin-Cummings.

Flanders, Peter M., et al. (1991). Efficient High-Level Programming on the AMT DAP. *Proceedings of the IEEE*, April, 79(4):524–536.

Flynn, Michael J. (1972). Some Computer Organizations and Their Effectiveness. *IEEE Transactions on Computers*, September, C-21(9):948–960.

Gabber, Evan, Amir Averbuch, and Amiram Yehudai (1991). Experience with a Portable Parallelizing Pascal Compiler. In *1991 International Conference on Parallel Processing*, vol. II, held in St. Charles, Ill., August, 207–210.

Gajski, Daniel D., et al. (1983). Cedar: A Large Scale Multiprocessor. In *1983 International Conference on Parallel Processing*, held in St. Charles, Ill., August, 524–529.

Gallivan, Kyle, William Jalby, and Dennis Gannon (1988). On the Problem of Optimizing Data Transfers for Complex Memory Systms. In *1988 ACM International Conference on Supercomputing*, held in St. Malo, France, June, 238–253.

Gallivan, Kyle, et al. (1989). Performance Prediction of Loop Constructs on Multiprocessor Hierarchical-Memory Systems. In *1989 ACM International Conference on Supercomputing*, held in Crete, Greece, June, 433–442.

Gannon, Dennis, William Jalby, and Kyle Gallivan (1987). Strategies for Cache and Local Memory Management by Global Program Transformation. In *Supercomputing: 1st International Conference*, held in Athens, Greece, number 297 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 229–254.

——— (1988). Strategies for Cache and Local Memory Management by Global Program Transformation. *Journal of Parallel and Distributed Computing*, October, 5(5):587–616.

Gao, Guang R., et al. (1992). Collective Loop Fusion for Array Contraction. In *1992 Workshop on Languages and Compilers for Parallel Computing*, held in New Haven, Conn., August, number 757 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 281–295.

Gerndt, Michael (1989). Array Distribution in SUPERB. In *1989 ACM International Conference on Supercomputing*, held in Crete, Greece, June, 164–174.

——— (1990). Updating Distributed Variables in Local Computations. *Concurrency: Practice & Experience*, September, 2(3):171–193.

——— (1991). Work Distribution in Parallel Programs for Distributed Memory Multiprocessors. In *1991 ACM International Conference on Supercomputing*, held in Cologne, Germany, June, 96–104.

Gerndt, Michael, and Hans P. Zima (1987). MIMD-Parallelization for Suprenum. In *Supercomputing: 1st International Conference*, held in Athens, Greece, number 297 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 278–293.

Gharachorloo, Kourosh, et al. (1990). Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors. In *17th International Symposium on Computer Architecture*, held in Seattle, Wash., May, 15–26.

Gilbert, John R., and Robert Schreiber (1991). Optimal Expression Evaluation for Data Parallel Architectures. *Journal of Parallel and Distributed Computing*, September, 13(1):58–64.

Girkar, Milind (1988). Partitioning Programs for Parallel Execution. In *1988 ACM International Conference on Supercomputing*, held in St. Malo, France, June, 216–229.

Girkar, Milind, and Constantine D. Polychronopoulos (1988). Compiling Issues for Supercomputers. In *Supercomputing '88*, held in Orlando, Fla., November, 164–173.

——— (1991). Optimization of Data/Control Conditions in Task Graphs. In *1991 Workshop on Languages and Compilers for Parallel Computing*, held in Santa Clara, Calif., August, number 589 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 152–168.

——— (1992). Automatic Extraction of Functional Parallelism from Ordinary Programs. *IEEE Transactions on Parallel and Distributed Systems*, March, 3(2):166–178.

Goff, Gina, Ken Kennedy, and Chau-Wen Tseng (1991). Practical Dependence Testing. In *ACM SIGPLAN '91 Conference on Programming Language Design and Implementation*, held in Toronto, Ont., June, 15–29.

Goldberg, Aaron J., and John L. Hennessy (1993). Mtool: An Integrated System for Performance Debugging Shared Memory Multiprocessor Applications. *IEEE Transactions on Parallel and Distributed Systems*, January, 4(1):28–40.

Goldberg, David (1990). *Computer Arithmetic*, Appendix A. Palo Alto, Calif.: Morgan Kaufmann.

——— (1991). What Every Computer Scientist Should Know About Floating-Point Arithmetic. *ACM Computing Surveys*, March, 23(1):5–48.

Goodman, James R. (1983). Using Cache Memory to Reduce Processor-Memory Traffic. In *10th International Symposium on Computer Architecture*, held in Stockholm, Sweden, June, 124–131.

Goodman, James R., Mary K. Vernon, and Philip J. Woest (1989). Efficient Synchronization Primitives for Large-Scale Cache-Coherent Multiprocessors. In *3rd International Conference on Architectural Support for Programming Languages and Operating Systems*, held in Boston, April, 64–73.

Gopinath, K., and John L. Hennessy (1989). Copy Elimination in Functional Languages. In *16th Annual ACM Symposium on Principles of Programming Languages*, held in Austin, Tex., January, 303–314.

Gornish, Edward H., Elana D. Granston, and Alexander V. Veidenbaum (1990). Compiler-Directed Data Prefetching in Multiprocessors with Memory Hierarchies. In *1990 ACM International Conference on Supercomputing*, held in Amsterdam, The Netherlands, June, 354–368.

Gottlieb, Allan, et al. (1983). The NYU Ultracomputer–Designing an MIMD Shared Memory Parallel Computer. *IEEE Transactions on Computers*, February, C-32(2):175–189.

Granlund, Torbjörn, and Peter L. Montgomery (1994). Division by Invariant Integers Using Multiplication. In *ACM SIGPLAN '94 Conference on Programming Language Design and Implementation*, held in Orlando, Fla., June, 61–72.

Granston, Elana D. (1993). Toward a Compile-Time Methodology for Reducing False Sharing and Communication Traffic in Shared Virtual Memory Systems. In *1993 Workshop on Languages and Compilers for Parallel Computing*, held in Portland, Ore., August, number 768 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 273–289.

Granston, Elana D., and Alexander V. Veidenbaum (1991). Detecting Redundant Accesses to Array Data. In *Supercomputing '91*, held in Albuquerque, N.M., November, 854–865.

Graunke, Gary, and Shreekant Thakkar (1990). Synchronization Algorithms for Shared-Memory Multiprocessors. *IEEE Computer*, June, 23(6):60–69.

Gries, David, and Fred B. Schneider (1993). *A Logical Approach to Discrete Math*. Texts and Monographs in Computer Science. Berlin: Springer Verlag.

Grit, Dale H. (1990). A Distributed Memory Implementation of SISAL. In *Fifth Distributed Memory Computing Conference*, held in Charleston, S.C., April, 1131–1136.

Gross, Thomas, and Peter Steenkiste (1990). Structured Dataflow Analysis for Arrays and Its Use in an Optimizing Compilers. *Software: Practice & Experience*, February, 20(2):133–155.

Grove, Dan, and Linda Torczon (1993). Interprocedural Constant Propagation: A Study of Jump Function Implementations. In *ACM SIGPLAN '93 Conference on Programming Language Design and Implementation*, held in Albuquerque, N.M., June, 90–99.

Grunwald, Dirk (1990). The Lambda Test Revisited. In *1990 International Conference on Parallel Processing*, vol. II, held in St. Charles, Ill., August, 220–223.

Guarna, Vincent A., Jr. (1988). A Technique for Analyzing Pointer and Structure References in Parallel Restructuring Compilers. In *1988 International Conference on Parallel Processing*, vol. II, held in St. Charles, Ill., August, 212–220.

Gupta, Anoop, Wolf-Dietrich Weber, and Todd Mowry (1990). Reducing Memory and Traffic Requirements for Scalable Directory-Based Cache Coherence Schemes. In *1990 International Conference on Parallel Processing*, vol. I, held in St. Charles, Ill., August, 312–321.

Gupta, Manish, and Prithviraj Banerjee (1991). Automatic Data Partitioning on Distributed Memory Multiprocessors. In *Sixth Distributed Memory Computing Conference*, held in Portland, Ore., April, 43–50.

——— (1992a). Demonstration of Automatic Data Partitioning Techniques for Parallelizing Compilers on Multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, March, 2(2):179–193.

——— (1992b). A Methodology for High-Level Synthesis of Communication on Multicomputers. In *1992 ACM International Conference on Supercomputing*, held in Washington, D.C., July, 357–367.

——— (1993). PARADIGM: A Compiler for Automatic Data Distribution on Multicomputers. In *1993 ACM International Conference on Supercomputing*, held in Tokyo, July, 87–96.

Gupta, Manish, and Edith Schonberg (1993). A Framework for Exploiting Data Availability to Optimize Communication. In *1993 Workshop on Languages and Compilers for Parallel Computing*, held in Portland, Ore., August, number 768 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 216–233.

Gupta, Rajiv (1989). The Fuzzy Barrier: A Mechanism for High Speed Synchronization of Processors. In *3rd International Conference on Architectural Support for Programming Languages and Operating Systems*, held in Boston, April, 54–63.

Gupta, S. K. S., et al. (1993). On Compiling Array Expressions for Efficient Execution on Distributed-Memory Machines. In *1993 International Conference on Parallel Processing*, vol. II, held in St. Charles, Ill., August, 301–305.

Guzzi, Mark D., et al. (1990). Cedar Fortran and Other Vector and Parallel Fortran Dialects. *The Journal of Supercomputing*, March, 4(1):37–62.

Haghighat, Mohammad R. (1990). Symbolic Dependence Analysis for High-Performance Parallelizing Compilers. In *Advances in Languages and Compilers for Parallel Computing, 1990 Workshop*, held in Irvine, Calif., August, Research Monographs in Parallel and Distributed Computing, Cambridge, Mass.: MIT Press, 310–330.

Haghighat, Mohammad R., and Constantine D. Polychronopoulos (1992). Symbolic Program Analysis and Optimization for Parallelizing Compilers. In *1992 Workshop on Languages and Compilers for Parallel Computing*, held in New Haven, Conn., August, number 757 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 538–562.

——— (1993). Symbolic Analysis: A Basis for Parallelization, Optimization and Scheduling of Programs. In *1993 Workshop on Languages and Compilers for Parallel Computing*, held in Portland, Ore., August, number 768 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 567–585.

Hall, Mary W., and Ken Kennedy (1992). Efficient Call Graph Analysis. *ACM Letters on Programming Languages and Systems*, September, 1(3):227–242.

Hall, Mary W., Ken Kennedy, and Kathryn S. McKinley (1991). Interprocedural Transformations for Parallel Code Generation. In *Supercomputing '91*, held in Albuquerque, N.M., November, 424–434.

Hall, Mary W., et al. (1992). Interprocedural Compilation of Fortran D for MIMD Distributed-Memory Machines. In *Supercomputing '92*, held in Minneapolis, Minn., November, 522–534.

——— (1993a). FIAT: A Framework for Interprocedural Analysis and Transformation. In *1993 Workshop on Languages and Compilers for Parallel Computing*, held in Portland, Ore., August, number 768 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 522–545.

——— (1993b). Experiences Using the ParaScope Editor: An Interactive Parallel Programming Tool. In *4th ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, held in San Diego, Calif., May, 33–43.

Harbison, Samuel P., and Guy L. Steele, Jr. (1991). *C: A Reference Manual*, third edition. Englewood Cliffs, N.J.: Prentice-Hall.

Hatcher, Philip J., and Michael J. Quinn (1991). *Data-Parallel Programming on MIMD Computers*. Scientific and Engineering Computation Series. Cambridge, Mass.: MIT Press.

Hatcher, Philip J., et al. (1991). Architecture-Independent Scientific Programming in Dataparallel C: Three Case Studies. In *Supercomputing '91*, held in Albuquerque, N.M., November, 208–217.

Havlak, Paul (1993). Construction of Thinned Gated Single-Assignment Form. In *1993 Workshop on Languages and Compilers for Parallel Computing*, held in Portland, Ore., August, number 768 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 477–499.

Havlak, Paul, and Ken Kennedy (1990). Experience with Interprocedural Analysis of Array Side Effects. In *Supercomputing '90*, held in New York, November, 952–961.

——— (1991). An Implementation of Interprocedural Bounded Regular Section Analysis. *IEEE Transactions on Parallel and Distributed Systems*, July, 2(3):350–360.

Hecht, Matthew S. (1977). *Flow Analysis of Computer Programs*. New York: North Holland.

Heg, Hans-Christian, and Hinrich Süeben (1991). Vectorization and Parallelization of Irregular Problems via Graph Coloring. In *1991 ACM International Conference on Supercomputing*, held in Cologne, Germany, June, 47–56.

Henderson, Leslie A., et al. (1990). On the Use of Diagnostic Dependence-Analysis Tools in Parallel Programming: Experiences Using PTOOL. *The Journal of Supercomputing*, March, 4(1):83–96.

Hendren, Laurie J., Joseph Hummel, and Alexandru Nicolau (1992). Abstractions for Recursive Pointer Data Structures: Improving the Analysis and Transformation of Imperative Programs. In *ACM SIGPLAN '92 Conference on Programming Language Design and Implementation*, held in San Francisco, Calif., June, 249–260.

Hendren, Laurie J., and Alexandru Nicolau (1989). Interference Analysis Tools for Parallelizing Programs with Recursive Data Structures. In *1989 ACM International Conference on Supercomputing*, held in Crete, Greece, June, 205–214.

——— (1990). Parallelizing Programs with Recursive Data Structures. *IEEE Transactions on Parallel and Distributed Systems*, January, 1(1):35–47.

Hennessy, John L., and David A. Patterson (1990). *Computer Architecture: A Quantitative Approach*. Palo Alto, Calif.: Morgan Kaufmann.

Hill, Mark D., and James R. Larus (1990). Cache Considerations for Multiprocessor Programmers. *Communications of the ACM*, August, 33(8):97–102.

Hillis, W. Daniel, and Lewis W. Tucker (1993). The CM-5 Connection Machine: A Scalable Supercomputer. *Communications of the ACM*, November, 36(11):30–40.

Hiranandani, Seema, Ken Kennedy, and Chau-Wen Tseng (1991). Compiler Optimizations for Fortran D on MIMD Distributed-Memory Machines. In *Supercomputing '91*, held in Albuquerque, N.M., November, 86–100.

——— (1992a). Evaluation of Compiler Optimizations for Fortran D on MIMD Distributed-Memory Machines. In *1992 ACM International Conference on Supercomputing*, held in Washington, D.C., July, 1–14.

——— (1992b). Compiling Fortran D for MIMD Distributed Memory Machines. *Communications of the ACM*, August, 35(8):66–88.

——— (1993). Preliminary Experiences with the Fortran D Compiler. In *Supercomputing '93*, held in Portland, Ore., November, 338–350.

——— (1994). Evaluating Compiler Optimizations for Fortran D. *Journal of Parallel and Distributed Computing*, April, 21(1):27–45.

Hiranandani, Seema, et al. (1991a). An Overview of the Fortran D Programming System. In *1991 Workshop on Languages and Compilers for Parallel Computing*, held in Santa Clara, Calif., August, number 589 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 18–34.

——— (1991b). Performance of Hashed Cache Data Migration Schemes on Multicomputers. *Journal of Parallel and Distributed Computing*, August, 12(4):415–422.

——— (1994). Compilation Techniques for Block-Cyclic Distributions. In *1994 ACM International Conference on Supercomputing*, held in Manchester, England, July, 392–403.

Holm, John, Antonio Lain, and Prithviraj Banerjee (1994). Compilation of Scientific Programs into Multithreaded and Message Driven Computation. In *Scalable High Performance Computing Conference*, held in Knoxville, Tenn., November, 518–525.

Hord, R. Michael, ed. (1990). *Parallel Supercomputing in SIMD Architectures*. Boca Raton, Fla.: CRC Press, Inc.

Horwitz, Susan, Phil Pfeiffer, and Thomas Reps (1989). Dependence Analysis for Pointer Variables. In *ACM SIGPLAN '89 Conference on Programming Language Design and Implementation*, held in Portland, Ore., June, 28–40.

Horwitz, Susan, Jan Prins, and Thomas Reps (1988). On the Adequacy of Program Dependence Graphs Representing Programs. In *15th Annual ACM Symposium on Principles of Programming Languages*, held in San Diego, Calif., January, 146–157.

Hsieh, Bor-Ming, Michael Hind, and Ron Cytron (1992). Loop Distribution with Multiple Exits. In *Supercomputing '92*, held in Minneapolis, Minn., November, 204–213.

Hsu, Jiun-Ming, and Prithviraj Banerjee (1990). A Message Passing Coprocessor for Distributed Memory Multicomputers. In *Supercomputing '90*, held in New York, November, 720–729.

Huang, Chua-Hua, and P. Sadayappan (1991). Communication-Free Hyperplane Partitioning of Nested Loops. In *1991 Workshop on Languages and Compilers for Parallel Computing*, held in Santa Clara, Calif., August, number 589 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 186–200.

Hudak, David E., and Santosh G. Abraham (1990). Compiler Techniques for Data Partitioning of Sequentially Iterated Parallel Loop. In *1990 ACM International Conference on Supercomputing*, held in Amsterdam, The Netherlands, June, 187–200.

——— (1991). Beyond Loop Partitioning: Data Assignment and Overlap to Reduce Communication Overhead. In *1991 ACM International Conference on Supercomputing*, held in Cologne, Germany, June, 172–182.

Hummel, Joseph, Laurie J. Hendren, and Alexandru Nicolau (1992). Abstract Description of Pointer Data Structures: An Approach for Improving the Analysis and Optimization of Imperative Programs. *ACM Letters on Programming Languages and Systems*, September, 1(3):243–260.

——— (1994a). A Language for Conveying the Aliasing Properties of Dynamic, Pointer-Based Data Structures. In *8th International Parallel Processing Symposium*, held in Cancun, Mexico, April, 208–216.

——— (1994b). A General Data Dependence Test for Dynamic, Pointer-Based Data Structures. In *ACM SIGPLAN '94 Conference on Programming Language Design and Implementation*, held in Orlando, Fla., June, 218–229.

Hummel, Susan Flynn, Edith Schonberg, and Lawrence E. Flynn (1991). Factoring: A Practical and Robust Method for Scheduling Parallel Loops. In *Supercomputing '91*, held in Albuquerque, N.M., November, 610–619.

——— (1992). Factoring: A Method for Scheduling Parallel Loops. *Communications of the ACM*, August, 35(8):90–101.

Hwang, Kai (1993). *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. New York: McGraw-Hill.

Ikudome, K., et al. (1990). An Automatic and Symbolic Parallelization System for Distributed Memory Parallel Computers. In *Fifth Distributed Memory Computing Conference*, held in Charleston, S.C., April, 1105–1114.

Irigoin, François, Pierre Jouvelot, and Rémi Triolet (1991). Semantical Interprocedural Parallelization: An Overview of the PIPS Project. In *1991 ACM International Conference on Supercomputing*, held in Cologne, Germany, June, 244–251.

Irigoin, François, and Rémi Triolet (1988). Supernode Partitioning. In *15th Annual ACM Symposium on Principles of Programming Languages*, held in San Diego, Calif., January, 319–329.

Jalby, William, and Ulrike Meier (1986). Optimizing Matrix Operations on a Parallel Multiprocessor with a Hierarchical Memory System. In *1986 International Conference on Parallel Processing*, held in St. Charles, Ill., August, 429–432.

James, David V., et al. (1990). Scalable Coherent Interface. *IEEE Computer*, June, 23(6):74–77.

Johnson, Richard, Wei Li, and Keshav Pingali (1991). An Executable Representation of Distance and Direction. In *1991 Workshop on Languages and Compilers for Parallel Computing*, held in Santa Clara, Calif., August, number 589 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 122–136.

Johnson, Richard, and Keshav Pingali (1993). Dependence-Based Program Analysis. In *ACM SIGPLAN '93 Conference on Programming Language Design and Implementation*, held in Albuquerque, N.M., June, 78–89.

Jones, Neil D., and Steven S. Muchnick (1981). Flow Analysis and Optimization of Lisp-like Structures. In *Program Flow Analysis: Theory and Applications*, Steven S. Muchnick and Neil D. Jones, eds. Englewood Cliffs, N.J.: Prentice-Hall, 102–131.

Jouvelot, Pierre, and Babak Dehbonei (1989). A Unified Semantic Approach for the Vectorization and Parallelization of Generalized Reductions. In *1989 ACM International Conference on Supercomputing*, held in Crete, Greece, June, 186–194.

Ju, Y.-J., and Henry G. Dietz (1991). Reduction of Cache Coherence Overhead by Compiler Data Layout and Loop Transformation. In *1991 Workshop on Languages and Compilers for Parallel Computing*, held in Santa Clara, Calif., August, number 589 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 344–358.

Kallis, Apostolos D., and David Klappholz (1991). Extending Conventional Flow Analysis to Deal with Array References. In *1991 Workshop on Languages and Compilers for Parallel Computing*, held in Santa Clara, Calif., August, number 589 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 251–265.

Karp, Alan H., and Robert G. Babb, II (1988). A Comparison of 12 Fortran Dialects. *IEEE Software*, September, 5(5):52–67.

Karp, Richard M., Raymond E. Miller, and Shmuel Winograd (1967). The Organization of Computations for Uniform Recurrence Equations. *Journal of the ACM*, July, 14(3):563–590.

Kasahara, Hironori, et al. (1990). A Compilation Scheme for Macro-Dataflow Computation on Hierarichical Multiprocessor Systems. In *1990 International Conference on Parallel Processing*, vol. II, held in St. Charles, Ill., August, 294–295.

——— (1991). A Multi-Grain Parallelizing Compilation Scheme for OSCAR (Optimally Scheduled Advanced Multiprocessor). In *1991 Workshop on Languages and Compilers for Parallel Computing*, held in Santa Clara, Calif., August, number 589 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 283–297.

Katz, Randy H., et al. (1985). Implementing a Cache Consistency Protocol. In *12th International Symposium on Computer Architecture*, held in Boston, June, 276–283.

Kennedy, Ken (1992). Software for Supercomputers of the Future. *The Journal of Supercomputing*, February, 5(4):251–262.

Kennedy, Ken, and Kathryn S. McKinley (1990). Loop Distribution with Arbitrary Control Flow. In *Supercomputing '90*, held in New York, November, 407–416.

——— (1992). Optimizing for Parallelism and Data Locality. In *1992 ACM International Conference on Supercomputing*, held in Washington, D.C., July, 323–334.

——— (1993). Maximizing Loop Parallelism and Improving Data Locality via Loop Fusion and Distribution. In *1993 Workshop on Languages and Compilers for Parallel Computing*, held in Portland, Ore., August, number 768 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 301–320.

Kennedy, Ken, Kathryn S. McKinley, and Chau-Wen Tseng (1991a). Analysis and Transformation in the ParaScope Editor. In *1991 ACM International Conference on Supercomputing*, held in Cologne, Germany, June, 433–447.

——— (1991b). Interactive Parallel Programming Using the ParaScope Editor. *IEEE Transactions on Parallel and Distributed Systems*, July, 2(3):329–341.

——— (1993). Analysis and Transformation in an Interactive Parallel Programming Tool. *Concurrency: Practice & Experience*, October, 5(7):575–602.

Kennedy, Ken, and Gerald Roth (1994). Context Optimization for SIMD Execution. In *Scalable High Performance Computing Conference*, held in Knoxville, Tenn., November, 445–453.

Keßler, Christoph W., and Wolfgang J. Paul (1993). Automatic Parallelization by Pattern-Matching. In *ACPC '93: 2nd International Conference of the Austrian Center for Parallel Computation*, held in Gmunden, Austria, October, number 734 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 166–181.

Klappholz, David, Apostolos D. Kallis, and Xiangyun Kong (1989). Refined C: An Update. In *Languages and Compilers for Parallel Computing, 1989 Workshop*, held in Urbana, Ill., August, Research Monographs in Parallel and Distributed Computing, Cambridge, Mass.: MIT Press, 331–357.

Klappholz, David, and Xiangyun Kong (1992). Extending the Banerjee-Wolfe Test to Handle Execution Conditions. In *1992 Workshop on Languages and Compilers for Parallel Computing*, held in New Haven, Conn., August, number 757 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 464–481.

Klappholz, David, Xiangyun Kong, and Apostolos D. Kallis (1989). Refine Fortran: An Update. In *Supercomputing '89*, held in Reno, Nev., November, 607–615.

Klappholz, David, Kleanthis Psarris, and Xiangyun Kong (1990). On the Perfect Accuracy of an Approximate Subscript Analysis Test. In *1990 ACM International Conference on Supercomputing*, held in Amsterdam, The Netherlands, June, 201–212.

Knobe, Kathleen, Joan D. Lukas, and Guy L. Steele, Jr. (1988). Massively Parallel Data Optimization. In *Second Symposium on the Frontiers of Massively Parallel Computation*, held in Fairfax, Va., October, 551–558.

——— (1990). Data Optimization: Allocation of Arrays to Reduce Communication on SIMD Machines. *Journal of Parallel and Distributed Computing*, February, 8(2):102–118.

Knobe, Kathleen, Joan D. Lukas, and Michael Weiss (1993). Optimization Techniques for SIMD Fortran Compilers. *Concurrency: Practice & Experience*, October, 5(7):527–552.

Knobe, Kathleen, and Venkataraman Natarajan (1990). Data Optimization: Minimizing Residual Interprocessor Data Motion on SIMD Machines. In *Third Symposium on the Frontiers of Massively Parallel Computation*, held in College Park, Md., October, 416–423.

Koblenz, Brian D., and William B. Noyce (1989). Constraint Based Vectorization. In *1989 ACM International Conference on Supercomputing*, held in Crete, Greece, June, 195–204.

Koelbel, Charles H. (1991). Compile-Time Generation of Regular Communication Patterns. In *Supercomputing '91*, held in Albuquerque, N.M., November, 101–110.

Koelbel, Charles H., and Piyush Mehrotra (1991a). Programming Data Parallel Algorithms on Distributed Memory Machines using Kali. In *1991 ACM International Conference on Supercomputing*, held in Cologne, Germany, June, 414–423.

——— (1991b). Compiling Global Name-Space Parallel Loops for Distributed Execution. *IEEE Transactions on Parallel and Distributed Systems*, October, 2(4):440–451.

Koelbel, Charles H., Piyush Mehrotra, and John Van Rosendale (1987a). Semi-Automatic Domain Decomposition in Blaze. In *1987 International Conference on Parallel Processing*, held in St. Charles, Ill., August, 521–524.

——— (1987b). Semi-Automatic Process Partitioning for Parallel Computation. *International Journal of Parallel Programming*, October, 16(5):365–382.

——— (1990). Supporting Shared Data Structures on Distributed Memory Architectures. In *2nd ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, held in Seattle, Wash., March, 177–186.

Koelbel, Charles H., et al. (1990). Parallel Loops on Distributed Machines. In *Fifth Distributed Memory Computing Conference*, held in Charleston, S.C., April, 1097–1104.

——— (1994). *The High Performance Fortran Handbook*. Cambridge, Mass., MIT Press.

Kogge, Peter M., and Harold S. Stone (1973). A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations. *IEEE Transactions on Computers*, August, C-22(8):786–792.

Kong, Xiangyun, David Klappholz, and Kleanthis Psarris (1990). The I Test: A New Test for Subscript Data Dependence. In *1990 International Conference on Parallel Processing*, vol. II, held in St. Charles, Ill., August, 204–211.

——— (1991). The I Test: An Improved Dependence Test for Automatic Parallelization and Vectorization. *IEEE Transactions on Parallel and Distributed Systems*, July, 2(3):342–349.

Kozdrowicki, E. W., and D. J. Theis (1980). Second Generation of Vector Supercomputers. *IEEE Computer*, October, 1(11):71–83.

Kremer, Ulrich, et al. (1988). Advanced Tools and Techniques for Automatic Parallelization. *Parallel Computing*, September, 7(3):387–393.

Krothapalli, V. P., and P. Sadayappan (1991). Removal of Redundant Dependences in Doacross Loops with Constant Dependences. *IEEE Transactions on Parallel and Distributed Systems*, July, 2(3):281–289.

Kruskal, Clyde P., Larry Rudolph, and Marc Snir (1988). Efficient Synchronization on Multiprocessors with Shared Memory. *ACM Transactions on Programming Languages and Systems*, October, 10(4):579–601.

Kuck, David J., et al. (1980). The Structure of an Advanced Vectorizer for Pipelined Processors. In *Proc. 4th International Computer Software and Applications Conf.*, held in Chicago, October, 709–715.

——— (1981). Dependence Graphs and Compiler Optimizations. In *8th Annual ACM Symposium on Principles of Programming Languages*, January, 207–218.

Kuck, David J., James McGraw, and Michael Wolfe (1984). Retire Fortran? *Physics Today*, May, 37(5):66–75.

Kuck, David J., et al. (1984). The Effects of Program Restructuring, Algorithm Change, and Architecture Choice on Program Performance. In *1984 International Conference on Parallel Processing*, held in St. Charles, Ill., August, 129–138.

Kuhn, Robert H. (1980). Optimization and Interconnection Complexity for: Parallel Processors, Single Stage Networks, and Decision Trees. Ph.D. Dissertation UIUCDCS-R-80-1009, University of Illinois, Dept. Computer Science, February, (UMI 80-26541).

Kulkarni, D., et al. (1991). Loop Partitioning for Distributed Memory Multiprocessors as Unimodular Transformations. In *1991 ACM International Conference on Supercomputing*, held in Cologne, Germany, June, 206–215.

Kung, H. T., and Jaspal Subhlok (1991). A New Approach for Automatic Parallelization of Blocked Linear Algebra Computations. In *Supercomputing '91*, held in Albuquerque, N.M., November, 122–129.

Labarta, Jesús, et al. (1991). Balanced Loop Partitioning Using GTS. In *1991 Workshop on Languages and Compilers for Parallel Computing*, held in Santa Clara, Calif., August, number 589 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 298–312.

Lam, Monica S., Edward E. Rothberg, and Michael E. Wolf (1991). The Cache Performance and Optimizations of Blocked Algorithms. In *4th International Conference on Architectural Support for Programming Languages and Operating Systems*, held in Santa Clara, Calif., April, 63–74.

Lamé, G. (1844). Note sur la Limite du Nombre des Divisions dans la Recherche du Plus Grand Commun Diviseur Entre Euex Eombres Entiers. *C.R. Acad. Sci. Paris*, 19:867–870.

Lamport, Leslie (1974). The Parallel Execution of DO Loops. *Communications of the ACM*, February, 17(2):83–93.

——— (1979). How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. *IEEE Transactions on Computers*, September, C-28(9):690–691.

Landi, William, and Barbara G. Ryder (1991). Pointer-Induced Aliasing: A Problem Classification. In *18th Annual ACM Symposium on Principles of Programming Languages*, held in Orlando, Fla., January, 93–103.

——— (1992). A Safe Approximate Algorithm for Interprocedural Pointer Aliasing. In *ACM SIGPLAN '92 Conference on Programming Language Design and Implementation*, held in San Francisco, Calif., June, 235–248.

Landi, William, Barbara G. Ryder, and Sean Zhang (1993). Interprocedural Modification Side Effect Analysis with Pointer Aliasing. In *ACM SIGPLAN '93 Conference on Programming Language Design and Implementation*, held in Albuquerque, N.M., June, 56–67.

Larus, James R. (1993). Compiling for Shared-Memory and Message-Passing Computers. *ACM Letters on Programming Languages and Systems*, 2:165–180.

Larus, James R., and Paul N. Hilfinger (1988). Detecting Conflicts Between Structure Accesses. In *ACM SIGPLAN '88 Conference on Programming Language Design and Implementation*, held in White Plains, N.Y., June, 21–34.

Lawrie, Duncan H., et al. (1975). Glypnir: A Programming Language for Illiac IV. *Communications of the ACM*, March, 18(3):157–164.

Leasure, Bruce, ed. (1991). Parallel Computing Forum Parallel Fortran Extensions. *Fortran Forum*, September (special issue), 10(3).

LeBlanc, Thomas J., and John M. Mellor-Crummey (1987). Debugging Parallel Programs with Instant Replay. *IEEE Transactions on Computers*, April, C-36(4):471–482.

Lee, Roland B., Pen-Chung Yew, and Duncan H. Lawrie (1987). Data Prefetching in Shared Memory Multiprocessors. In *1987 International Conference on Parallel Processing*, held in St. Charles, Ill., August, 28–31.

Lengauer, Thomas, and Robert E. Tarjan (1979). A Fast Algorithm for Finding Dominators in a Flow Graph. *ACM Transactions on Programming Languages and Systems*, July, 1(1):121–141.

Lenoski, Daniel, et al. (1992). The Stanford Dash Multiprocessor. *IEEE Computer*, March, 25(3):63–79.

——— (1993). The DASH Prototype: Logic Overhead and Performance. *IEEE Transactions on Parallel and Distributed Systems*, January, 4(1):41–61.

Li, Hui, and Kenneth C. Sevcik (1994). Exploiting Cache Affinity in Software Cache Coherence. In *1994 ACM International Conference on Supercomputing*, held in Manchester, England, July, 264–273.

Li, Jingke (1991). Compiling Crystal for Distributed-Memory Machines. Ph.D. Dissertation, Yale University, Dept. Computer Science, October.

Li, Jingke, and Marina C. Chen (1990a). Automating the Coordination of Interprocessor Communication. In *Advances in Languages and Compilers for Parallel Computing, 1990 Workshop*, held in Irvine, Calif., August, Research Monographs in Parallel and Distributed Computing, Cambridge, Mass.: MIT Press, 402–424.

——— (1990b). Index Domain Alignment: Minimizing Cost of Cross-Referencing Between Distributed Arrays. In *Third Symposium on the Frontiers of Massively Parallel Computation*, held in College Park, Md., October, 424–433.

——— (1990c). Generating Explicit Communication from Shared-Memory Program References. In *Supercomputing '90*, held in New York, November, 865–876.

——— (1991a). Compiling Communication-Efficient Programs for Massively Parallel Machines. *IEEE Transactions on Parallel and Distributed Systems*, July, 2(3):361–376.

——— (1991b). The Data Alignment Phase in Compiling Programs for Distributed-Memory Machines. *Journal of Parallel and Distributed Computing*, October, 13(2):213–221.

Li, Jingke, and Michael Wolfe (1994). Defining, Analyzing, and Transforming Program Constructs. *IEEE Parallel & Distributed Technology*, Spring, 2(1):32–39.

Li, Kai, and Paul Hudak (1989). Memory Coherence in Shared Virtual Memory Systems. *ACM Transactions on Computer Systems*, November, 7(4):321–359.

Li, Wei, and Keshav Pingali (1992a). A Singular Loop Transformation Framework Based on Nonsingular Matrices. In *1992 Workshop on Languages and Compilers for Parallel Computing*, held in New Haven, Conn., August, number 757 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 391–405.

——— (1992b). Access Normalization: Loop Restructuring for NUMA Compilers. In *5th International Conference on Architectural Support for Programming Languages and Operating Systems*, held in Boston, October, 285–295.

——— (1994). A Singular Loop Transformation Framework Based on Nonsingular Matrices. *International Journal of Parallel Programming*, April, 22(2):183–205.

Li, Zhiyuan (1989). Intraprocedural and Interprocedural Data Dependence Analysis for Parallel Computing. Ph.D. Dissertation CSRD Rpt. No. 910, University of Illinois, Dept. Computer Science, August.

——— (1991). Compiler Algorithms for Event Variable Synchronization. In *1991 ACM International Conference on Supercomputing*, held in Cologne, Germany, June, 85–95.

——— (1992). Array Privatization for Parallel Execution of Loops. In *1992 ACM International Conference on Supercomputing*, held in Washington, D.C., July, 313–322.

Li, Zhiyuan, and Walid A. Abu-Sufah (1987). On Reducing Data Synchronization in Multiprocessed Loops. *IEEE Transactions on Computers*, January, C-36(1):105–109.

Li, Zhiyuan, and Pen-Chung Yew (1988a). Efficient Interprocedural Analysis for Program Parallelization and Restructuring. In *ACM/SIGPLAN PPEALS, Parallel Programming: Experience with Applications, Languages and Systems*, held in New Haven, Conn., July, 85–99.

——— (1988b). Interprocedural Analysis for Parallel Computing. In *1988 International Conference on Parallel Processing*, vol. II, held in St. Charles, Ill., August, 221–228.

——— (1988c). Program Parallelization with Interprocedural Analysis. *The Journal of Supercomputing*, October, 2(2):225–244.

——— (1989). Some Results on Exact Data Dependence Analysis. In *Languages and Compilers for Parallel Computing, 1989 Workshop*, held in Urbana, Ill., August, Research Monographs in Parallel and Distributed Computing, Cambridge, Mass.: MIT Press, 374–401.

Li, Zhiyuan, Pen-Chung Yew, and Chuan-Qi Zhu (1989). Data Dependence Analysis on Multi-Dimensional References. In *1989 ACM International Conference on Supercomputing*, held in Crete, Greece, June, 215–224.

——— (1990). An Efficient Data Dependence Analysis for Parallelizing Compilers. *IEEE Transactions on Parallel and Distributed Systems*, January, 1(1):26–34.

Lichnewsky, A., and F. Thomasset (1988). Introducing Symbolic Problem Solving Techniques in the Dependence Testing Phases of a Vectorizer. In *1988 ACM International Conference on Supercomputing*, held in St. Malo, France, June, 308–406.

Lilja, David J. (1994). The Impact of Parallel Loop Scheduling Strategies on Prefetching in a Shared-Memory Multiprocessor. *IEEE Transactions on Parallel and Distributed Systems*, June, 5(6):573–584.

Lilja, David J., and Pen-Chung Yew (1991). Combining Hardware and Software Cache Coherence Strategies. In *1991 ACM International Conference on Supercomputing*, held in Cologne, Germany, June, 274–283.

——— (1993). Improving Memory Utilization in Cache Coherence Directories. *IEEE Transactions on Parallel and Distributed Systems*, October, 4(10):1130–1146.

Lippman, Stanley B. (1991). *C++ Primer*, second edition. Reading, Mass.: Addison-Wesley.

Liu, Lang-Sheng, Chin-Wen Ho, and Jang-Ping Sheu (1990). On the Parallelism of Nested For-Loops Using Index Shift Method. In *1990 International Conference on Parallel Processing*, vol. II, held in St. Charles, Ill., August, 119–123.

Loeliger, Jon, et al. (1991). Pointer Target Tracking: An Empirical Study. In *Supercomputing '91*, held in Albuquerque, N.M., November, 14–23.

Loveman, David B. (1976). Program Improvement by Source to Source Transformation. In *3rd ACM Symposium on Principles of Programming Languages*, held in Atlanta, Ga., January, 140–152.

——— (1977). Program Improvement by Source-to-Source Transformation. *Communications of the ACM*, January, 20(1):121–145.

Lu, Lee-Chung (1991). A Unified Framework for Systematic Loop Transformations. In *3rd ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, held in Williamsburg, Va., April, 28–38.

Lu, Lee-Chung, and Marina C. Chen (1990). Subdomain Dependence Test for Massive Parallelism. In *Supercomputing '90*, held in New York, November, 962–972.

——— (1991). Parallelizing Loops with Indirect Array References or Pointers. In *1991 Workshop on Languages and Compilers for Parallel Computing*, held in Santa Clara, Calif., August, number 589 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 201–217.

Lucco, Steven (1992). A Dynamic Scheduling Method for Irregular Parallel Programs. In *ACM SIGPLAN '92 Conference on Programming Language Design and Implementation*, held in San Francisco, Calif., June, 200–211.

Lundstrom, S. F., and George H. Barnes (1980). Controllable MIMD Architecture. In *1980 International Conference on Parallel Processing*, held in St. Charles, Ill., August, 19–27.

MacDonald, Tom (1991). C for Numerical Computing. *The Journal of Supercomputing*, June, 5(1):31–48.

Markatos, Evangelos P., and Thomas J. LeBlanc (1991). Load Balancing vs. Locality Management in Shared-Memory Multiprocessors. In *1992 International Conference on Parallel Processing*, vol. I, held in St. Charles, Ill., August, 258–267.

——— (1992). Using Processor Affinity in Loop Scheduling on Shared-Memory Multiprocessors. In *Supercomputing '92*, held in Minneapolis, Minn., November, 104–113.

——— (1994). Using Processor Affinity in Loop Scheduling on Shared-Memory Multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, April, 5(4):379–400.

Marlowe, Thomas J., et al. (1993). Pointer-Induced Aliasing: A Clarification. *Sigplan Notices*, September, 28(9):67–70.

Maslov, Vadim (1992). Delinearization: An Efficient Way to Break Multiloop Dependence Equations. In *ACM SIGPLAN '92 Conference on Programming Language Design and Implementation*, held in San Francisco, Calif., June, 152–161.

——— (1994). Lazy Array Data-Flow Dependence Analysis. In *21st Annual ACM Symposium on Principles of Programming Languages*, held in Portland, Ore., January, 311–325.

Matsumoto, T., and K. Hiraki (1993). Dynamic Switching of Coherent Cache Protocols and Its Effects on Doacross Loops. In *1993 ACM International Conference on Supercomputing*, held in Tokyo, July, 328–337.

Maydan, Dror E., Saman P. Amarasinghe, and Monica S. Lam (1992). Data Dependence and Data-Flow Analysis of Arrays. In *1992 Workshop on Languages and Compilers for Parallel Computing*, held in New Haven, Conn., August, number 757 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 434–448.

——— (1993). Array Data-Flow Analysis and Its Use in Array Privatization. In *20th Annual ACM Symposium on Principles of Programming Languages*, held in Charleston, S.C., January, 2–15.

Maydan, Dror E., John L. Hennessy, and Monica S. Lam (1991). Efficient and Exact Data Dependence Analysis. In *ACM SIGPLAN '91 Conference on Programming Language Design and Implementation*, held in Toronto, Ont., June, 1–14.

Mayer, Herb, and Michael Wolfe (1993). Interprocedural Alias Analysis: Implementation and Empirical Results. *Software: Practice & Experience*, November, 23(11):1201–1233.

McKellar, A. C., and E. G. Coffman, Jr. (1969). Organizing Matrices and Matrix Operations for Paged Memory Systems. *Communications of the ACM*, March, 12(3):153–165.

Mehrotra, Piyush, and John Van Rosendale (1987). The BLAZE Language: A Parallel Language for Scientific Programming. *Parallel Computing*, November, 5(3):339–361.

——— (1990). Programing Distributed Memory Architectures Using Kali. In *Advances in Languages and Compilers for Parallel Computing, 1990 Workshop*, held in Irvine, Calif., August, Research Monographs in Parallel and Distributed Computing, Cambridge, Mass.: MIT Press, 364–384.

Mellor-Crummey, John M. (1991). On-the-fly Detection of Data Races for Programs with Nested Fork-Join Parallelism. In *Supercomputing '91*, held in Albuquerque, N.M., November, 24–33.

Mercer, Randall (1988). The Convex Fortran 5.0 Compiler. In *Third International Conference on Supercomputing*, vol. 2, 164–175.

Merlin, John H. (1991). ADAPTing Fortran 90 Array Programs for Distributed Memory Architectures. In *ACPC '91: 1st International Conference of the Austrian Center for Parallel Computation*, held in Salzburg, Austria, September, 184–200.

Metcalf, Michael, and John Reid (1990). *Fortran 90 Explained*. Oxford, England: Oxford University Press.

Midkiff, Samuel P., and David A. Padua (1986). Compiler Generated Synchronization for DO Loops. In *1986 International Conference on Parallel Processing*, held in St. Charles, Ill., August, 544–551.

——— (1987). Compiler Algorithms for Synchronization. *IEEE Transactions on Computers*, December, C-36(12):1485–1495.

——— (1990). Issues in the Optimization of Parallel Programs. In *1990 International Conference on Parallel Processing*, vol. II, held in St. Charles, Ill., August, 105–113.

Midkiff, Samuel P., David A. Padua, and Ron Cytron (1989). Compiling Programs with User Parallelism. In *Languages and Compilers for Parallel Computing, 1989 Workshop*, held in Urbana, Ill., August, Research Monographs in Parallel and Distributed Computing, Cambridge, Mass.: MIT Press, 402–422.

Min, Sang Lyul, and Jean-Loup Baer (1989). A Timestamp-Based Cache Coherence Scheme. In *1989 International Conference on Parallel Processing*, vol. I, held in St. Charles, Ill., August, 23–32.

——— (1990). A Performance Comparison of Directory-Based and Timestamp-Based Cache Coherence Schemes. In *1990 International Conference on Parallel Processing*, vol. I, held in St. Charles, Ill., August, 305–311.

——— (1992). Design and Analysis of a Scalable Cache Coherence Scheme Based on Clocks and Timestamps. *IEEE Transactions on Parallel and Distributed Systems*, January, 3(1):25–44.

Mirchandaney, Ravi, et. al. (1988). Principles of Runtime Support for Parallel Processors. In *1988 ACM International Conference on Supercomputing*, held in St. Malo, France, June, 140–152.

Mounes-Toussi, Farnaz, David J. Lilja, and Zhiyuan Li (1994). Evaluation of a Compiler Optimization for Improving the Performance of a Coherence Directory. In *1994 ACM International Conference on Supercomputing*, held in Manchester, England, July, 75–84.

Mowry, Todd C., Monica S. Lam, and Anoop Gupta (1992). Design and Evaluation of a Compiler Algorithm for Prefetching. In *5th International Conference on Architectural Support for Programming Languages and Operating Systems*, held in Boston, October, 62–75.

Muraoka, Yoichi (1971). Parallelism Exposure and Exploitation in Programs. Ph.D. Dissertation UIUCDCS-R-71-424, University of Illinois, Dept. Computer Science, (UMI 71-21189).

Nemhauser, George L., and Laurence A. Wolsey (1988). *Integer and Combinatorial Optimization*. New York: John Wiley & Sons.

Netzer, Robert, and Bart Miller (1990). Detecting Data Races in Parallel Program Executions. In *Advances in Languages and Compilers for Parallel Computing, 1990 Workshop*, held in Irvine, Calif., August, Research Monographs in Parallel and Distributed Computing, Cambridge, Mass.: MIT Press, 109–129.

——— (1992). Experience with Techniques for Refining Data Race Detection. In *1992 Workshop on Languages and Compilers for Parallel Computing*, held in New Haven, Conn., August, number 757 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 449–463.

Nichols, Mark A., Howard Jay Siegel, and Henry G. Dietz (1990). Data Management and Control-Flow Constructs in a SPMD/SIMD Parallel Language/Compiler. In *Third Symposium on the Frontiers of Massively Parallel Computation*, held in College Park, Md., October, 397–406.

——— (1993). Data Management and Control-Flow Constructs of an SPMD/SIMD Parallel Language/Compiler. *IEEE Transactions on Parallel and Distributed Systems*, February, 4(2):222–233.

Nicolau, Alexandru (1987). Loop Quantization or Unwinding Done Right. In *Supercomputing: 1st International Conference*, held in Athens, Greece, number 297 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 294–308.

——— (1988). A Generalized Loop Unwinding Technique. *Journal of Parallel and Distributed Computing*, October, 5(5):568–586.

O'Boyle, Michael, and G. A. Hedayat (1992). Data Alignment: Transformations to Reduce Communication on Distributed Memory Architectures. In *Scalable High Performance Computing Conference*, held in Williamsburg, Va., April, 366–371.

Ortiz, Luis F., Ron Y. Pinter, and Shlomit S. Pinter (1991). An Array Language for Data Parallelism: Definition, Compilation and Applications. *The Journal of Supercomputing*, June, 5(1):7–29.

Otto, Steve W., and Michael Wolfe (1992). The MetaMP Approach to Parallel Programming. In *Fourth Symposium on the Frontiers of Massively Parallel Computation*, held in McLean, Va., October, Washington, D.C.: IEEE Computer Society Press, 562–565.

Owicki, Susan, and Anant Agarwal (1989). Evaluating the Performance of Software Cache Coherence. In *3rd International Conference on Architectural Support for Programming Languages and Operating Systems*, held in Boston, April, 230–242.

Padua, David A. (1979). Multiprocessors: Discussion of Some Theoretical and Practical Problems. Ph.D. Dissertation UIUCDCS-R-79-990, University of Illinois, Dept. Computer Science, November.

Padua, David A., David J. Kuck, and Duncan H. Lawrie (1980). High-Speed Multiprocessors and Compilation Techniques. *IEEE Transactions on Computers*, September, C-29(9):763–776.

Padua, David A., and Michael Wolfe (1986). Advanced Compiler Optimizations for Supercomputers. *Communications of the ACM*, December, 29(12):1184–201.

Pancake, Cherri M. (1993). Multithreaded Languages for Scientific and Technical Computing. *Proceedings of the IEEE*, February, 81(2):288–304.

Papamarcos, Mark S., and Janak H. Patel (1984). A Low-Overhead Coherence Solution for Multiprocessors with Private Cache Memories. In *11th International Symposium on Computer Architecture*, held in Ann Arbor, Mich., June, 348–354.

Peir, Jih-Kwon, and Ron Cytron (1987). Minimum Distance: A Method for Partitioning Recurrences for Multiprocessors. In *1987 International Conference on Parallel Processing*, held in St. Charles, Ill., August, 217–225.

——— (1989). Minimum Distance: A Method for Partitioning Recurrences for Multiprocessors. *IEEE Transactions on Computers*, August, 38(8):1203–1211.

Perron, Robert, and Craig Mundie (1986). The Architecture of the Alliant FX/8 Computer. In *Digest of Papers: Compcon 86*, held in Washington, D.C., March, 390–394.

Petersen, Paul M., and David A. Padua (1992). Dynamic Dependence Analysis: A Novel Method for Data Dependence Evaluation. In *1992 Workshop on Languages and Compilers for Parallel Computing*, held in New Haven, Conn., August, number 757 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 64–81.

——— (1993). Static and Dynamic Evaluation of Data Dependence Analysis. In *1993 ACM International Conference on Supercomputing*, held in Tokyo, July, 107–116.

Pfister, Gregory, and V. Alan Norton (1985). "Hot Spot" Contention and Combining in Multistage Interconnection Networks. *IEEE Transactions on Computers*, October, C-34(10):943–948.

Pfister, Gregory, et al. (1985). The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture. In *1985 International Conference on Parallel Processing*, held in St. Charles, Ill., August, 764–771.

Philippsen, Michael, and Walter F. Tichy (1991). Modula-2* and Its Compilation. In *ACPC '91: 1st International Conference of the Austrian Center for Parallel Computation*, held in Salzburg, Austria, September, 169–183.

——— (1992). Compiling for Massively Parallel Machines. In *Code Generation: Concepts, Tools, Techniques*, Robert Giegerich and Susan Graham, eds. Berlin: Springer Verlag, 92–111.

Picano, Silvio, Eugene D. Brooks, III, and Joseph E. Hoag (1991). Programming Costs of Explicit Memory Localization on a Large Scale Shared Memory Multiprocessor. In *Supercomputing '91*, held in Albuquerque, N.M., November, 36–45.

Pingali, Keshav, and Anne Rogers (1990). Compiling for Locality. In *1990 International Conference on Parallel Processing*, vol. II, held in St. Charles, Ill., August, 142–146.

Pingali, Keshav, et al. (1990). Dependence Flow Graphs: An Algebraic Approach to Program Dependencies. In *Advances in Languages and Compilers for Parallel Computing, 1990 Workshop*, held in Irvine, Calif., August, Research Monographs in Parallel and Distributed Computing, Cambridge, Mass.: MIT Press, 445–467.

——— (1991). Dependence Flow Graphs: An Algebraic Approach to Program Dependences. In *18th Annual ACM Symposium on Principles of Programming Languages*, held in Orlando, Fla., January, 67–78.

Polychronopoulos, Constantine D. (1986). On Program Restructuring, Scheduling and Communication for Parallel Processor Systems. Ph.D. Dissertation UILU-ENG-86-8006, University of Illinois, Dept. Computer Science, August.

——— (1987a). Advanced Loop Optimizations for Parallel Computers. In *Supercomputing: 1st International Conference*, held in Athens, Greece, number 297 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 255–277.

——— (1987b). Loop Coalescing: A Compiler Transformation for Parallel Machines. In *1987 International Conference on Parallel Processing*, held in St. Charles, Ill., August, 235–242.

——— (1988a). Compiler Optimizations for Enhancing Parallelism and Their Impact on Architecture Design. *IEEE Transactions on Computers*, August, 37(8):991–1004.

——— (1988b). Toward Auto-Scheduling Compilers. *The Journal of Supercomputing*, November, 2(3):297–330.

——— (1988c). *Parallel Programming and Compilers*. Norwell, Mass.: Kluwer Academic Publishers.

——— (1991). The Hierarchical Task Graph and Its Use in Auto-Scheduling. In *1991 ACM International Conference on Supercomputing*, held in Cologne, Germany, June, 252–263.

Polychronopoulos, Constantine D., and Utpal Banerjee (1987). Processor Allocation for Horizontal and Vertical Parallelism and Related Speedup Bounds. *IEEE Transactions on Computers*, April, C-36(4):410–420.

Polychronopoulos, Constantine D., and David J. Kuck (1987). Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Supercomputers. *IEEE Transactions on Computers*, December, C-36(12):1485–1495.

Polychronopoulos, Constantine D., et al. (1989a). Parafrase-2: An Environment for Parallelizing, Partitioning, Synchronizing and Scheduling Programs on Multiprocessors. In *1989 International Conference on Parallel Processing*, vol. II, held in St. Charles, Ill., August, 39–48.

——— (1989b). The Structure of Parafrase-2: An Advanced Parallelizing Compiler for C and Fortran. In *Languages and Compilers for Parallel Computing, 1989 Workshop*, held in Urbana, Ill., August, Research Monographs in Parallel and Distributed Computing, Cambridge, Mass.: MIT Press, 423–453.

Ponnusamy, Ravi, Joel H. Saltz, and Alok Choudhary (1993). Runtime-Compilation Techniques for Data Partitioning and Communication Schedule Reuse. In *Supercomputing '93*, held in Portland, Ore., November, 361–370.

Potter, Jerry L., ed. (1985). *The Massively Parallel Processor*. Cambridge, Mass.: MIT Press.

Psarris, Kleanthis (1992). On Exact Data Dependence Analysis. In *1992 ACM International Conference on Supercomputing*, held in Washington, D.C., July, 303–312.

Psarris, Kleanthis, David Klappholz, and Xiangyun Kong (1991). On the Accuracy of the Banerjee Test. *Journal of Parallel and Distributed Computing*, June, 12(2):152–158.

Psarris, Kleanthis, Xiangyun Kong, and David Klappholz (1991). Extending the I Test to Direction Vectors. In *1991 ACM International Conference on Supercomputing*, held in Cologne, Germany, June, 330–340.

——— (1993). The Direction Vector I Test. *IEEE Transactions on Parallel and Distributed Systems*, November, 4(11):1280–1290.

Pugh, William (1991a). Uniform Techniques for Loop Optimization. In *1991 ACM International Conference on Supercomputing*, held in Cologne, Germany, June, 341–352.

——— (1991b). The Omega Test: A Fast and Practical Integer Programming Algorithm for Dependence Analysis. In *Supercomputing '91*, held in Albuquerque, N.M., November, 4–13.

——— (1992a). A Practical Algorithm for Exact Array Dependence Analysis. *Communications of the ACM*, August, 35(8):102–114.

——— (1992b). Definitions of Dependence Distance. *ACM Letters on Programming Languages and Systems*, September, 1(3):261–265.

Pugh, William, and David Wonnacott (1992). Eliminating False Data Dependences Using the Omega Test. In *ACM SIGPLAN '92 Conference on Programming Language Design and Implementation*, held in San Francisco, Calif., June, 140–151.

——— (1993). An Exact Method for Analysis of Value-based Array Data Dependences. In *1993 Workshop on Languages and Compilers for Parallel Computing*, held in Portland, Ore., August, number 768 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 546–566.

Quinn, Michael J., and Philip J. Hatcher (1990). Compiling SIMD Programs for MIMD Architectures. In *1990 International Conference on Computer Languages*, held in New Orleans, La., March, 291–296.

Quinn, Michael J., Philip J. Hatcher, and Karen C. Jourdenais (1988). Compiling C* for a Hypercube Multicomputer. In *ACM/SIGPLAN PPEALS, Parallel Programming: Experience with Applications, Languages and Systems*, held in New Haven, Conn., July, 57–65.

Ramanujam, J. (1992). Nonunimodular Transformations of Nested Loops. In *Supercomputing '92*, held in Minneapolis, Minn., November, 214–223.

Ramanujam, J., and P. Sadayappan (1989). A Methodology for Parallelizing Programs for Multicomputers and Complex Memory Multiprocessors. In *Supercomputing '89*, held in Reno, Nev., November, 637–646.

——— (1990a). Nested Loop Tiling for Distributed Memory Machines. In *Fifth Distributed Memory Computing Conference*, held in Charleston, S.C., April, 1088–1096.

——— (1990b). Tiling of Iteration Spaces for Multicomputers. In *1990 International Conference on Parallel Processing*, held in St. Charles, Ill., August, 179–186.

——— (1991a). Compile-Time Techniques for Data Distribution in Distributed Memory Machines. *IEEE Transactions on Parallel and Distributed Systems*, October, 2(4):472–482.

——— (1991b). Tiling Multidimensional Iteration Spaces for Nonshared Memory Machines. In *Supercomputing '91*, held in Albuquerque, N.M., November, 111–120.

——— (1992). Tiling Multidimensional Iteration Spaces for Multicomputers. *Journal of Parallel and Distributed Computing*, October, 16(2):108–230.

Rauchwerger, Lawrence, and David A. Padua (1994). The Privatizing DOALL Test: A Run-Time Technique for DOALL Loop Identification and Array Privatization. In *1994 ACM International Conference on Supercomputing*, held in Manchester, England, July, 33–43.

Redon, Xavier, and Paul Feautrier (1993). Detection of Recurrences in Sequential Programs with Loops. In *PARLE '93: Parallel Architectures and Languages Europe*, held in Munich, Germany, June, number 694 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 132–145.

Rettberg, Randall, and Robert Thomas (1986). Contention is No Obstacle to Shared-Memory Multiprocessing. *Communications of the ACM*, December, 29(12):1202–1212.

Ribas, Hudson B. (1990). Obtaining Dependence Vectors for Nested-Loop Computations. In *1990 International Conference on Parallel Processing*, vol. II, held in St. Charles, Ill., August, 212–219.

Robert, Yves, and Siang W. Song (1992). Revisiting Cycle Shrinking. *Parallel Computing*, 18:481–496.

Rogers, Anne, and Keshav Pingali (1989). Process Decomposition Through Locality of Reference. In *ACM SIGPLAN '89 Conference on Programming Language Design and Implementation*, held in Portland, Ore., June, 69–80.

——— (1994). Compiling for Distributed Memory Architectures. *IEEE Transactions on Parallel and Distributed Systems*, March, 5(3):281–298.

Rudolph, David C., and Constantine D. Polychronopoulos (1989). An Efficient Message-Passing Scheduler Based on Guided Self-Scheduling. In *1989 ACM International Conference on Supercomputing*, held in Crete, Greece, June, 50–61.

Rühl, Roland, and Marco Annaratone (1990). Parallelization of Fortran Code on Distributed-Memory Parallel Processors. In *1990 ACM International Conference on Supercomputing*, held in Amsterdam, The Netherlands, June, 342–353.

Russel, R. M. (1978). The Cray-1 Computer System. *Communications of the ACM*, January, 21(1):63–72.

Ryder, Barbara G. (1979). Constructing the Call Graph of a Program. *IEEE Transactions on Software Engineering*, May, SE-5(3):216–226.

Sabot, Gary (1992). A Compiler for a Massively Parallel Distributed Memory MIMD Computer. In *Fourth Symposium on the Frontiers of Massively Parallel Computation*, held in McLean, Va., October, Washington, D.C.: IEEE Computer Society Press, 12–20.

Sabot, Gary, and Skef Wholey (1993). CMAX: A Fortran Translator for the Connection Machine System. In *1993 ACM International Conference on Supercomputing*, held in Tokyo, July, 147–156.

Saltz, Joel H., Harry Berryman, and Janet Wu (1991). Multiprocessors and Run-Time Compilation. *Concurrency: Practice & Experience*, December, 3(6):573–592.

Saltz, Joel H., and Piyush Mehrotra, eds. (1992). *Languages, Compilers and Run-Time Environments for Distributed Memory Machines*, vol. 3 of *Advances in Parallel Computing*. New York: North Holland.

Saltz, Joel H., Ravi Mirchandaney, and Kathleen Crowley (1989). The DoConsider Loop. In *1989 ACM International Conference on Supercomputing*, held in Crete, Greece, June, 29–40.

——— (1991). Run-Time Parallelization and Scheduling of Loops. *IEEE Transactions on Computers*, May, 40(5):603–612.

Saltz, Joel H., et al. (1990). Run-Time Scheduling and Execution of Loops on Message Passing Machines. *Journal of Parallel and Distributed Computing*, April, 8(4):303–312.

Sarkar, Vivek (1988). Synchronization using Counting Semaphores. In *1988 ACM International Conference on Supercomputing*, held in St. Malo, France, June, 627–637.

——— (1989). Determining Average Program Execution Times and their Variance. In *ACM SIGPLAN '89 Conference on Programming Language Design and Implementation*, held in Portland, Ore., June, 298–309.

Sarkar, Vivek, and David C. Cann (1990). POSC: A Partitioning and Optimizing SISAL Compiler. In *1990 ACM International Conference on Supercomputing*, held in Amsterdam, The Netherlands, June, 148–163.

Sarkar, Vivek, and Guang R. Gao (1991). Optimization of Array Accesses by Collective Loop Transformations. In *1991 ACM International Conference on Supercomputing*, held in Cologne, Germany, June, 194–205.

Sarkar, Vivek, and Radhika Thekkath (1992). A General Framework for Iteration-Reordering Transformations. In *ACM SIGPLAN '92 Conference on Programming Language Design and Implementation*, held in San Francisco, Calif., June, 175–187.

Scarborough, Randolph G., and Harwood G. Kolsky (1986). A Vectorizing Fortran Compiler. *IBM Journal of Research and Development*, March, 30(2):163–171.

Schneck, Paul B. (1972). Automatic Recognition of Vector and Parallel Operations in a Higher Level Language. *Sigplan Notices*, November, 7(11):45–52.

Schonberg, Edith (1989). On-the-Fly Detection of Access Anomalies. In *ACM SIGPLAN '89 Conference on Programming Language Design and Implementation*, held in Portland, Ore., June, 285–297.

Schrijver, A. (1986). *Theory of Linear and Integer Programming*. Chichester, England: John Wiley & Sons.

Sequent Computer Corp. (1986). *Guide to Parallel Programming*. Beaverton, Ore.

Shang, Weijia, and Jose A. B. Fortes (1988). Independent Partitioning of Algorithms with Uniform Dependencies. In *1988 International Conference on Parallel Processing*, vol. II, held in St. Charles, Ill., August, 26–33.

Shang, Weijia, Matthew T. O'Keefe, and Jose A. B. Fortes (1994). On Loop Transformations for Generalized Cycle Shrinking. *IEEE Transactions on Parallel and Distributed Systems*, February, 5(2):193–204.

Shen, Zhiyu, Zhiyuan Li, and Pen-Chung Yew (1989). An Empirical Study of Array Subscripts and Data Dependencies. In *1989 International Conference on Parallel Processing*, vol. II, held in St. Charles, Ill., August, 145–152.

——— (1990). An Empirical Study of Fortran Programs for Parallelizing Compilers. *IEEE Transactions on Parallel and Distributed Systems*, July, 1(3):356–364.

Sheu, Jang-Ping, and Tsu-Huei Tai (1991). Partitioning and Mapping Nested Loops on Multiprocessor Systems. *IEEE Transactions on Parallel and Distributed Systems*, October, 2(4):430–439.

Shih, Zen-Cheung, Gen-Huey Chen, and R. C. T. Lee (1987). Systolic Algorithms to Examine All Pairs of Elements. *Communications of the ACM*, February, 30(2):161–167.

Shostak, Robert (1981). Deciding Linear Inequalities by Computing Loop Residues. *Journal of the ACM*, October, 28(4):769–779.

Sinharoy, Balaram, and Boleslaw K. Szymanski (1994). Data and Task Alignment in Distributed Memory Architectures. *Journal of Parallel and Distributed Computing*, April, 21(1):61–74.

Smith, James E., et al. (1987). The ZS-1 Central Processor. In *2nd International Conference on Architectural Support for Programming Languages and Operating Systems*, held in Palo Alto, Calif., October, 199–204.

Smith, Kevin, and William F. Appelbe (1988). PAT: An Interactive Fortran Parallelizing Assistant Tool. In *1988 International Conference on Parallel Processing*, vol. II, held in St. Charles, Ill., August, 58–62.

——— (1989). Interactive Conversion of Sequential to Multitasking Fortran. In *1989 ACM International Conference on Supercomputing*, held in Crete, Greece, June, 225–234.

Smith, Kevin, William F. Appelbe, and Kurt Stirewalt (1990). Incremental Dependence Analysis for Interactive Parallelization. In *1990 ACM International Conference on Supercomputing*, held in Amsterdam, The Netherlands, June, 330–341.

Smith, Lauren L. (1991). Vectorizing C Compilers: How Good Are They? In *Supercomputing '91*, held in Albuquerque, N.M., November, 544–553.

Snyder, Lawrence, and David G. Socha (1990). An Algorithm Producing Balanced Partitionings of Data Arrays. In *Fifth Distributed Memory Computing Conference*, held in Charleston, S.C., April, 867–875.

Socha, David G. (1990). An Approach to Compiling Single-point Iterative Programs for Distributed Memory Computers. In *Fifth Distributed Memory Computing Conference*, held in Charleston, S.C., April, 1017–1027.

Solworth, Jon (1992). On the Feasibility of Dynamic Partitioning of Pointer Structures. In *1992 Workshop on Languages and Compilers for Parallel Computing*, held in New Haven, Conn., August, number 757 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 82–96.

Squillante, Mark S., and Edward D. Lazowska (1993). Using Processor-Cache Affinity Information in Shared-Memory Multiprocessor Scheduling. *IEEE Transactions on Parallel and Distributed Systems*, February, 4(2):131–143.

Srinivasan, Harini, James Hook, and Michael Wolfe (1993). Static Single Assignment for Explicitly Parallel Programs. In *20th Annual ACM Symposium on Principles of Programming Languages*, held in Charleston, S.C., January, 16–28.

Stanat, Donald F., and David F. McAllister (1977). *Discrete Mathematics in Computer Science*. Englewood Cliffs, N.J.: Prentice-Hall.

Stenstrom, Per (1990). A Survey of Cache Coherence Schemes for Multiprocessors. *IEEE Computer*, June, 23(6):12–24.

Stevens, Kent G., Jr. (1986). Todays Supercomputers. In *Digest of Papers: Compcon 86*, held in Washington, D.C., March, 2–6.

Stone, Harold S. (1990). *High-Performance Computer Architecture*, second edition. Reading, Mass.: Addison-Wesley.

Stroustrup, Bjarne (1991). *The C++ Programming Language*, second edition. Reading, Mass.: Addison-Wesley.

Su, Hong-Men, and Pen-Chung Yew (1989). On Data Synchronization for Multiprocessors. In *16th International Symposium on Computer Architecture*, held in Jerusalem, Israel, May, 416–423.

——— (1991). Efficient Doacross Execution on Distributed Shared-Memory Multiprocessors. In *Supercomputing '91*, held in Albuquerque, N.M., November, 842–853.

Tanaka, Yoshikazu, et al. (1988). Compiling Techniques for First-Order Linear Recurrences on a Vector Computer. In *Supercomputing '88*, held in Orlando, Fla., November, 174–181.

——— (1990). Compiling Techniques for First-Order Linear Recurrences on a Vector Computer. *The Journal of Supercomputing*, March, 4(1):63–82.

Tang, Peiyi (1993). Exact Side Effects for Interprocedural Dependence Anaylsis. In *1993 ACM International Conference on Supercomputing*, held in Tokyo, July, 137–146.

Tang, Peiyi, and Pen-Chung Yew (1986). Processor Self-Scheduling for Multiple-Nested Parallel Loops. In *1986 International Conference on Parallel Processing*, held in St. Charles, Ill., August, 528–535.

Tang, Peiyi, Pen-Chung Yew, and Chuan-Qi Zhu (1988). Impact of Self-Scheduling Order on Performance of Multiprocessor Systems. In *1988 ACM International Conference on Supercomputing*, held in St. Malo, France, June, 593–603.

——— (1990). Compiler Techniques for Data Synchronization in Nested Parallel Loops. In *1990 ACM International Conference on Supercomputing*, held in Amsterdam, The Netherlands, June, 177–186.

Tarjan, Robert E. (1972). Depth-First Search and Linear Graph Algorithms. *SIAM Journal of Computing*, June, 1(2):146–160.

——— (1974). Testing Flow Graph Reducibility. *Journal of Computer and System Sciences*, 9:355–365.

Temam, Olivier, Elana D. Granston, and William Jalby (1993). To Copy or Not to Copy: A Compile-Time Technique for Assessing When Data Copying Should be Used to Eliminate Cache Conflicts. In *Supercomputing '93*, held in Portland, Ore., November, 410–419.

Thakur, Rajeev, Alok Choudhary, and Geoffrey Fox (1994). Runtime Array Redistribution in HPF Programs. In *Scalable High Performance Computing Conference*, held in Knoxville, Tenn., November, 309–316.

Thapar, Manu, and Bruce Delagi (1990). Scalable Cache Coherence for Large Shared Memory Multiprocessors. In *CONPAR 90 – VAPP IV: Joint International Conference on Vector and Parallel Processing*, held in Zurich, Switzerland, September, number 634 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 592–603.

Theobald, Kevin B., and Guang R. Gao (1991). An Efficient Parallel Algorithm for All Pairs Examination. In *Supercomputing '91*, held in Albuquerque, N.M., November, 742–753.

Thinking Machines Corporation (1991). *CM Fortran Reference Manual, version 1.0*. Cambridge, Mass.: MIT Press, February.

Tjiang, Steve, et al. (1991). Integrating Scalar Optimization and Parallelization. In *1991 Workshop on Languages and Compilers for Parallel Computing*, held in Santa Clara, Calif., August, number 589 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 137–151.

Torrellas, Josep, Monica S. Lam, and John L. Hennessy (1994). False Sharing and Spatial Locality in Multiprocessor Caches. *IEEE Transactions on Computers*, June, 43(6):651–663.

Torres, Jordi, et al. (1993). Align and Distribute-Based Linear Loop Transformations. In *1993 Workshop on Languages and Compilers for Parallel Computing*, held in Portland, Ore., August, number 768 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 321–339.

Towle, Ross (1976). Control and Data Dependence for Program Transformations. Ph.D. Dissertation UIUCDCS-R-76-788, University of Illinois, Dept. Computer Science, October, (UMI 76-24191).

Tremblay, Jean-Paul, and Ram P. Manohar (1975). *Discrete Mathematical Structures with Application to Computer Science*. New York: McGraw-Hill.

Triolet, Rémi, François Irigoin, and Paul Feautrier (1986). Direct Parallelization of Call Statements. In *ACM SIGPLAN '86 Symposium on Compiler Construction*, held in Palo Alto, Calif., June, 175–185.

Tseng, Chau-Wen (1993). An Optimizing Fortran D Compiler for MIMD Distributed-Memory Machines. Ph.D. Dissertation TR93-199, Rice University, Dept. Computer Science, January.

Tseng, Ping-Sheng (1990a). Compiling Programs for a Linear Systolic Array. In *ACM SIGPLAN '90 Conference on Programming Language Design and Implementation*, held in White Plains, N.Y., June, 311–321.

——— (1990b). A Systolic Array Programming Language. In *Fifth Distributed Memory Computing Conference*, held in Charleston, S.C., April, 1125–1130.

Tsuda, Takao, and Yoshitoshi Kunieda (1988). V-Pascal: An Automatic Vectorizing Compiler for Pascal with No Language Extensions. In *Supercomputing '88*, held in Orlando, Fla., November, 182–191.

——— (1990). V-Pascal: An Automatic Vectorizing Compiler for Pascal with No Language Extensions. *The Journal of Supercomputing*, September, 4(3):251–276.

Tsuda, Takao, Yoshitoshi Kunieda, and Piyaporn Atipas (1989). Automatic Vectorization of Character String Manipulations and Relational Operations in Pascal. In *Supercomputing '89*, held in Reno, Nev., November, 187–196.

Tu, Peng, and David A. Padua (1993). Automatic Array Privatization. In *1993 Workshop on Languages and Compilers for Parallel Computing*, held in Portland, Ore., August, number 768 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 500–521.

Tzen, Ten H., and Lionel M. Ni (1991). Dynamic Loop Scheduling for Shared-Memory Multiprocessors. In *1991 International Conference on Parallel Processing*, vol. II, held in St. Charles, Ill., August, 247–250.

——— (1993a). Trapezoid Self-Scheduling: A Practical Scheduling Scheme for Parallel Compilers. *IEEE Transactions on Parallel and Distributed Systems*, January, 4(1):87–98.

——— (1993b). Dependence Uniformization: A Loop Parallelization Technique. *IEEE Transactions on Parallel and Distributed Systems*, May, 4(5):547–558.

Veidenbaum, Alexander V. (1986). A Compiler-Assisted Cache Coherence Solution for Multiprocessors. In *1986 International Conference on Parallel Processing*, held in St. Charles, Ill., August, 1029–1036.

Venugopal, Sesh, and William Eventoff (1991). Automatic Transformation of Fortran Loops to Reduce Cache Conflicts. In *1991 ACM International Conference on Supercomputing*, held in Cologne, Germany, June, 183–193.

von Eicken, Thorsten, et al. (1992). Active Messages: A Mechanism for Integrated Communication and Computation. In *19th International Symposium on Computer Architecture*, held in Gold Coast, Australia, May, 256–266.

von Hanxleden, Reinhard, and Ken Kennedy (1992). Relaxing SIMD Control Flow Constraints using Loop Transformations. In *ACM SIGPLAN '92 Conference on Programming Language Design and Implementation*, held in San Francisco, Calif., June, 188–199.

von Hanxleden, Reinhard, et al. (1992). Compiler Analysis for Irregular Problems in Fortran D. In *1992 Workshop on Languages and Compilers for Parallel Computing*, held in New Haven, Conn., August, number 757 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 97–111.

Wallace, David R. (1988). Dependence of Multi-Dimensional Array References. In *1988 ACM International Conference on Supercomputing*, held in St. Malo, France, June, 418–428.

Wallach, Steve (1986). The CONVEX C-1 64-Bit Supercomputer. In *Digest of Papers: Compcon 86*, held in Washington, D.C., March, 452–457.

Warren, Joe (1984). A Hierarchical Basis for Reordering Transformations. In *11th Annual ACM Symposium on Principles of Programming Languages*, held in Salt Lake City, Ut., January, 272–282.

Watts, Tia M., Mary Lou Soffa, and Rajiv Gupta (1992). Techniques for Integrating Parallelizing Transformations and Compiler Based Scheduling Methods. In *Supercomputing '92*, held in Minneapolis, Minn., November, 830–839.

Weber, Wolf-Dietrich, and Anoop Gupta (1989). Analysis of Cache Invalidation Patterns in Multiprocessors. In *3rd International Conference on Architectural Support for Programming Languages and Operating Systems*, held in Boston, April, 243–256.

Wedel, Dorothy (1975). Fortran for the Texas Instruments ASC System. *SIGPLAN Notices*, March, 10(3):119–132.

Wegman, Mark N., and F. Kenneth Zadeck (1985). Constant Propagation with Conditional Branches. In *12th Annual ACM Symposium on Principles of Programming Languages*, held in New Orleans, La., January, 291–299.

——— (1991). Constant Propagation with Conditional Branches. *ACM Transactions on Programming Languages and Systems*, April, 13(2):181–210.

Weihl, William E. (1980). Interprocedural Data Flow Analysis in the Presence of Pointers, Procedure Variables and Label Variables. In *7th Annual ACM Symposium on Principles of Programming Languages*, held in Las Vegas, Nev., January, 83–94.

Weiss, Michael (1991). Strip Mining on SIMD Architectures. In *1991 ACM International Conference on Supercomputing*, held in Cologne, Germany, June, 234–243.

Whitfield, Deborah, and Mary Lou Soffa (1991). Automatic Generation of Global Optimizers. In *ACM SIGPLAN '91 Conference on Programming Language Design and Implementation*, held in Toronto, Ont., June, 120–129.

Wholey, Skef (1992). Automatic Data Mapping for Distributed-Memory Parallel Computers. In *1992 ACM International Conference on Supercomputing*, held in Washington, D.C., July, 25–34.

Williams, H. P. (1976). Fourier-Motzkin Elimination Extension to Integer Programming Problems. *Journal of Combinatorial Theory (A)*, 21:118–123.

——— (1983). A Characterisation of All Feasible Solutions to an Integer Program. *Discrete Applied Mathematics*, 5:147–155.

Wolf, Michael E. (1992). Improving Locality and Parallelism in Nested Loops. Ph.D. Dissertation CSL-TR-92-538, Stanford University, Dept. Computer Science, August.

Wolf, Michael E., and Monica S. Lam (1990). An Algorithmic Approach to Compound Loop Transformations. In *Advances in Languages and Compilers for Parallel Computing, 1990 Workshop*, held in Irvine, Calif., August, Research Monographs in Parallel and Distributed Computing, Cambridge, Mass.: MIT Press, 243–259.

——— (1991a). A Loop Transformation Theory and an Algorithm to Maximize Parallelism. *IEEE Transactions on Parallel and Distributed Systems*, October, 2(4):452–471.

——— (1991b). A Data Locality Optimizing Algorithm. In *ACM SIGPLAN '91 Conference on Programming Language Design and Implementation*, held in Toronto, Ont., June, 30–44.

Wolfe, Michael (1982). Optimizing Supercompilers for Supercomputers. Ph.D. Dissertation UIUCDCS-R-82-1105, University of Illinois, Dept. Computer Science, October, (UMI 83-03027).

——— (1986a). Advanced Loop Interchanging. In *1986 International Conference on Parallel Processing*, held in St. Charles, Ill., August, 536–543.

——— (1986b). Loop Skewing: The Wavefront Method Revisited. *International Journal of Parallel Programming*, August, 15(4):279–294.

——— (1987). Vector Optimization vs. Vectorization. In *Supercomputing: 1st International Conference*, held in Athens, Greece, number 297 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 309–315.

——— (1988a). Vector Optimization vs. Vectorization. *Journal of Parallel and Distributed Computing*, October, 5(5):551–567.

——— (1988b). Multiprocessor Synchronization for Concurrent Loops. *IEEE Software*, January, 5(1):23–32.

——— (1989a). Loop Rotation. In *Languages and Compilers for Parallel Computing, 1989 Workshop*, held in Urbana, Ill., August, Research Monographs in Parallel and Distributed Computing, Cambridge, Mass.: MIT Press, 531–553.

——— (1989b). *Optimizing Supercompilers for Supercomputers*. Research Monographs in Parallel and Distributed Computing. Cambridge, Mass.: MIT Press.

——— (1989c). More Iteration Space Tiling. In *Supercomputing '89*, held in Reno, Nev., November, 655–664.

——— (1991a). Data Dependence and Program Restructuring. *The Journal of Supercomputing*, January, 4(4):321–344.

——— (1991b). Experiences with Data Dependence Abstractions. In *1991 ACM International Conference on Supercomputing*, held in Cologne, Germany, June, 321–329.

——— (1991c). The Tiny Loop Restructuring Research Tool. In *1991 International Conference on Parallel Processing*, vol. II, held in St. Charles, Ill., August, 46–53.

——— (1992a). Beyond Induction Variables. In *ACM SIGPLAN '92 Conference on Programming Language Design and Implementation*, held in San Francisco, Calif., June, 162–174.

——— (1992b). Doany: Not Just Another Parallel Loop. In *1992 Workshop on Languages and Compilers for Parallel Computing*, held in New Haven, Conn., August, number 757 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 421–433.

——— (1993). Engineering a Data Dependence Test. *Concurrency: Practice & Experience*, October, 5(7):603–622.

Wolfe, Michael, and Utpal Banerjee (1987). Data Dependence and Its Application to Parallel Processing. *International Journal of Parallel Programming*, April, 16(2):137–178.

Wolfe, Michael, and Chau-Wen Tseng (1992). The Power Test for Data Dependence. *IEEE Transactions on Parallel and Distributed Systems*, September, 3(5):591–601.

Wu, Min-You, and Daniel D. Gajski (1988). A Programming Aid for Hypercube Architectures. *The Journal of Supercomputing*, November, 2(3):349–372.

Wu, Youfeng, and Ted G. Lewis (1990). Parallelizing While Loops. In *1990 International Conference on Parallel Processing*, vol. II, held in St. Charles, Ill., August, 7–8.

Yang, Bwolen, et al. (1993). DO&Merge: Integrating Parallel Loops and Reductions. In *1993 Workshop on Languages and Compilers for Parallel Computing*, held in Portland, Ore., August, number 768 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 169–183.

Yasue, T., H. Yamana, and Yoichi Muraoka (1992). A Fortran Compiling Method for Dataflow Machines and Its Prototype Compiler for the Parallel Processing System Harray. In *1992 Workshop on Languages and Compilers for Parallel Computing*, held in New Haven, Conn., August, number 757 in Lecture Notes in Computer Science, Berlin: Springer Verlag, 482–496.

Zhu, Chuan-Qi, and Pen-Chung Yew (1987). A Scheme to Enforce Data Dependence on Large Multiprocessor Systems. *IEEE Transactions on Software Engineering*, June, SE-13(6):726–739.

Zima, Hans P., Heinz-J. Bast, and Michael Gerndt (1988). SUPERB: A Tool for Semi-Automatic MIMD/SIMD Parallelization. *Parallel Computing*, January, 6(1):1–18.

Zima, Hans P., and Barbara M. Chapman (1991). *Supercompilers for Parallel and Vector Computers*. New York: ACM Press.

——— (1993). Compiling for Distributed-Memory Systems. *Proceedings of the IEEE*, February, 81(2):264–287.