Tiling for Optimal Resource Utilization

Abstract

A Dagstuhl Seminar (no. 98341) on Tiling was held 24–28 August 1998 with thirty six participants from seven countries. During the meeting there were twenty five lectures, a panel session, and several informal discussions. The outcomes of the meeting include

- Fruitful and ongoing discussion on all aspects of tiling (through the creation of a web site* and a mailing list, `tiling@irisa.fr`).
- A collection of abstracts (this report) of all of the lectures, made publicly available at the web site (As usual the abstracts are also being printed in booklet form for distribution to the participants and for availability on request from the Dagstuhl office).
- A panel discussion and followup activities on the creation of a common set of benchmarks for researchers in the community.

1 Organizers

- J. Ferrante, University of California, San Diego
- W. Giloi, GMD First, Berlin
- S. Rajopadhye, IRISA, Rennes
- L. Thiele ETH-Zentrum, Zurich

2 Motivation

The following statement, written by the organizers was included with the invitations to the particiants. Its scope was deliberately somewhat narrow, and had the useful effect of forcing the participants to view a large number of compiler optimizations through "tiling colored" glasses.

Tiling is a regular partitioning of a uniform index space representing either computations (e.g., the iteration space of a loop program), or data (e.g., arrays distributed over the processors of a parallel machine). Tiling can be used to

*`http://www.irisa.fr/api/Rajopadhye/tiling`

achieve many different performance goals, such as exploiting data locality in hierarchical memory machines, communication optimization by message aggregation, communication-computation overlap, and latency avoidance.

Being such a common paradigm, tiling is used by many different communities in computer science, each with slightly different perspectives. Application writers tune a given program for performance by hand using multiple instances of tiling, some times sacrificing portability and ease of programming and debugging. Compiler writers have the same performance goals, but apply tiling automatically to a wide class of programs. Hence their considerations include feasibility of an automatic solution, reasonable computation times, and efficiency and ease of automatic code generation. VLSI processor array and embedded system designers have other constraints in the global context of the application (real-time throughputs, power consumption), but are willing to accept slow compilers (or design assistant tools). And certain problems may be best resolved at run time, since all parameters may not be available at compile time.

Underlying all of these applications of tiling is the issue of optimality in the presence of limited resources, embodied as a non-linear discrete optimization question.

- Is it possible to accurately model the machine behavior with a few cost parameters for a wide variety of machines?
- Is it possible to use such a model to predict the cost of a given program?
- How tractable is the resulting optimization problem (is it preferable to retain a less accurate model in the interests of tractability)?
- What are the consequences of separating multiple applications of tiling into separate optimization problems?
- Is a more global approach needed?

This seminar will bring together researchers from these diverse groups to foster cross fertilization between them. We will examine the differences in the problems, the models, and techniques they use, and the quality of their solutions, and how they interact with ultimate performance of an ENTIRE system.

3 Participants

- Bowen Alpern, San Diego, USA
- Corinne Ancourt, Fontainbleu, France
- Rumen Andonov, Valenciennes, France
- Scott Baden, San Diego, USA
- Hannah Bast, Saarbrucken, Germany

- Pierre-Yves Calland, Valenciennes, France
- Larry Carter, San Diego, USA
- Siddhartha Chatterjee, Chapel Hill, USA
- Alain Darte, Lyon, France
- Peter Drakenberg, Stockholm, Sweden
- Reinhardt Euler, Brest, France
- Jeanne Ferrante, San Diego, USA
- Susan Flynn Hummel, New York, USA
- Vladimir Getov, Harrow, UK
- Wolfgang K. Giloi, Berlin, Germany
- Martin Griebl, Passau, Germany
- Tony Hey, Southampton, UK
- Karin Hogstedt, San Diego, USA
- Francois Irigoin, Fontainbleu, France
- Paul H. J. Kelly, London, UK
- Jens Knoop, Dortmund, Germany
- Christian Lengauer, Passau, Germany
- Margaret Martonosi, Princeton, USA
- Keshav Pingali, Ithaca, USA
- Oscar Plata, Malaga, Spain
- Hans-Werner Pohl, Berlin, Germany
- Gudula Rünger, Leipzig, Germany
- Sanjay Rajopadhye, Rennes, France
- Fabrice Rastello, Lyon, France
- Thomas Rauber, Halle-Wittenberg, Germany
- Tanguy Risset, Rennes, France
- P. Sadayappan, Columbus, USA
- Robert Schreiber, Palo Alto, USA
- Jürgen Teich, Zurich, Switzerland
- Lothar Thiele, Zurich, Switzerland
- Chau-Wen Tseng, College Park, USA

4 Benchmarks Panel

A discussion session was held on the topic of establishing a collection of benchmarks for researchers in the field. Positions statements were presented by F. Irigoin, K. Pingali (presented by J. Ferrante in absentia), ... The ongoing discussion is carried out through a mailing list and recorded at the workshop web site.

5 Lectures

The abstracts of the twenty five lectures are given below.

Tiling and Scheduling with Architectural and Applicative Constraints

Corrine Ancourt

This talk presents a technique to map automatically a complete digital signal processing (DSP) application onto a parallel machine with distributed memory. Unlike other applications where coarse or medium grain scheduling techniques can be used, DSP applications integrate several thousand of tasks and hence necessitate fine grain considerations. Moreover finding an effective mapping imperatively require to take into account both architectural resources constraints and real time constraints. The main contribution of this paper is to show how it is possible to handle and to solve data partitioning, and fine-grain scheduling under the above operational constraints using Concurrent Constraints Logic Programming languages (CCLP). Our concurrent resolution technique undertaking linear and non linear constraints takes advantage of the special features of signal processing applications and provides a solution equivalent to a manual solution for the representative "BROAD BAND SURVEILLANCE" application.

Optimal Orhtogonal Tiling

Rumen Andonov

Iteration space tiling is a common strategy used by parallelizing compilers and in performance tuning of parallel codes. We address the problem of determining the tile size that minimizes the total execution time. We restrict our attention to *orthogonal tiling*—uniform dependency programs with (hyper) parallelepiped shaped iteration domains which can be tiled with hyperplanes parallel to the domain boundaries. Our formulation includes many machine and program models used in the literature, notably the BSP programming model. We resolve the optimization problem analytically, yielding a closed form solution.

Keywords: coarse grain pipelining, SPMD programs, loop blocking, nonlinear optimization, communication-computation overlap, supernode partitioning, automatic parallelization, macro-systolic arrays.

This is joint work with S. Rajopadhye and N. Yanev.

KeLP$^{\mathbf{N}}$ (exp (N ln KeLP))

Scott B. Baden

Hierarchically organized parallel multicomputers present opportunities for delivering high performance, but also many obstacles. Historically, parallel programming models assume a non-hierarchical view of system organization, which is the basis for most general-purpose commercial multicomputers. As a result, programmers are forced to navigate a cluttered terrain of processes, threads, messages, shared memory, and so on. Instead, I propose a more orderly programming model, $\mathrm{KeLP}^{\mathrm{N}}$, which supports hierarchical control flow, data decomposition, and data motion. These mechanisms are parameterized according to the level of the machine at which they are applied, and thus present a uniform programmer interface. The run time system hides most of the details, executing anonymous parameterized instruction sequences that carry out the desired behavior at each level of the hierarchy. I present results for a 2-level prototype of $\mathrm{KeLP}^{\mathrm{N}}$ running on a dedicated SMP cluster. A variety of hierarchically organized computations were implemented, that generally outperform their non-hierarchical counterparts. In particular, these applications are able to effectively overlap communication on a system which does not support such overlap via non-blocking communication. KeLP supports overlap by ascribing communication at the node rather than the processor level. Communication executes as a separate concurrent task, and how it is implemented on the node cannot be exploited by the programmer. I conclude by generalizing the 2-level model to the general case of N levels, and sketch the details of an overlapped version of an iterative finite difference solver, running on a 3-level multicomputer with symmetric multiprocessor nodes. Overlap in this computation is expressed at 2 distinct levels.

Dynamic Scheduling with Incomplete Information

Hannah Bast

We consider the following scheduling problem: Our goal is to execute a given amount of arbitrarily decomposable work on a distributed machine as quickly as possible. The work is maintained by a central scheduler that can assign chunks of work of an arbitrary size to idle processors. The difficulty is that the processing time required for a chunk is not exactly predictable—usually the less, the larger the chunk—and that processors suffer a delay for each assignment. Our objective is to minimize the total wasted time of the schedule, that is, the sum of all delays plus the idle times of processors waiting for the last processor to finish. We introduce a new deterministic model for this setting, based on estimated ranges [a(w),b(w)] for processing times of chunks of size w. Depending on a, b and a measure for the overall deviation from these estimates, we can prove matching upper and lower bounds on the wasted time, the former being achieved by our new balancing strategy. This is in sharp contrast with previous work that, even under the strong assumption of independent, approximately normally distributed chunk processing times, proposed only heuristic scheduling schemes supported merely by empirical evidence. Our model naturally subsumes this stochastic setting, and our generic analysis is valid for most of the existing schemes too, proving them to be non-optimal.

Tiling, the Universal Optimization

Larry Carter

Tiling is a method of *improving performance* of a program by partitioning the program's *ISG* (Iteration Space Graph) into *similar pieces* for *atomic execution* on a computer with a hierarchy of processors and memory.

In this talk, I visit each of the italicized phrases twice. The first pass gives a limited meaning to each term, and is intended to be non-controversial. The second pass expands the definitions and makes the argument that a broad range of optimization techniques are, in essence, tiling. We argue that tiling should consider storage mapping, scheduling, and communication pipelining decisions; that it encompasses inspector/executor methods; that it can facilitate register allocation, storage compaction, instruction cache optimization, fault tolerance, and adaptive computing on heterogeneous platforms; and so on.

This is joint work with everyone who has ever improved the performance of a program, and in particular with Bowen Alpern, Jeanne Ferrante, Susan Flynn Hummel, Kang Su Gatlin, Karin Hogstedt, and Nick Mitchell.

TUNE: System Support for Memory-Friendly Programming

Siddhartha Chatterjee

The pervasive use of multi-level memory hierarchies in microprocessor-based machines makes the performance of an application primarily determined by, and extremely sensitive to, its memory hierarchy mapping. Good performance therefore requires "memory-friendly programming": careful layout of data structures, and restructuring of code and/or data use patterns to improve locality. The lack of automatic tools for enhancing locality currently forces many application programmers to manually restructure their codes. Unfortunately, the sophisticated algorithms seen in modern scientific computing require equally sophisticated restructuring techniques, beyond the loop tiling transformations that some research compilers can perform automatically for dense iterative affine loop codes. Such restructuring techniques require expertise in computer architecture, burden the application programmer with tedious machine-specific details unrelated to program correctness, and reduce the readability, maintainability, and portability of the restructured code. The goal of the TUNE project is to improve our understanding of locality for a wide class of "hierarchical" problems and to develop a toolkit to aid the programmer in developing memory-friendly programs for such problems. Our ultimate goal is to demonstrate that the TUNE toolkit can be used to improve the performance of a naive implementation of an application to a level comparable to that of an implementation with extensive manual restructuring, but at a small fraction of effort on the part of the programmer.

A comprehensive solution to the problem of improving locality in hierarchical codes requires interactions among several sub-disciplines of computer science: scientific computing, numerical analysis, programming languages, compilers, performance modeling, and computer architecture. This project therefore targets all aspects of the locality problem: developing the relevant mathematical techniques for representing and manipulating locality; characterizations of the relationship between program transformations and numerical accuracy; implementing interactive and automatic locality management tools; and proposing and evaluating innovative memory architectures for future-generation systems.

The talk will discuss these various aspects of the TUNE project.

Code generation for the juggling technique

Alain Darte

This talk follows Rob Schreiber's explanations on "how to juggle". Starting from a set of perfectly nested loops, the juggling technique builds a closed-form formula for the mapping and the scheduling of computations that achieves a Locally Sequential Globally Parallel partitioning of the loops.

In this talk, we show how we can generate the code for such a partitioning. Our first approach was to use classical non unimodular loop transformations, but this resulted in a very inefficient solution where the cost of the hardware that carries out the original code was completely overwhelmed by housekeeping computations. The reason of this overhead was due to complex operators, such as div and mod, and expensive loop bounds evaluations. We present another solution based on a decision-tree of depth (n-1) where n is the number of nested loops, for which only a few adds and conditionals are needed. This solution exploits the mathematical properties of juggling. The reduction of cost, compared with the first approach, is typically an order of magnitude.

Multi-level Tiling Interactions

Jeanne Ferrante

Optimizations, including tiling, often target a single level of memory or parallelism, such as cache. These optimizations usually operate on a level-by-level basis, guided by a cost function parameterized by features of that single level. The benefits of optimizations guided by these one-level cost functions decreases as architectures tend towards a hierarchy of memory and of parallelism. We have identified three common architectural scenarios where a single tiling choice could be improved by using information from multiple levels in concert. For each scenario, we derive multi-level cost functions which guide the optimal choice of tile size and shape, and quantify the improvement gained. We give both analysis and simulation results to support our points.

This is joint work with Nicholas Mitchell, Karin Hogstedt, Larry Carter.

Specialized Tools for Performance Tuning

Vladimir Getov

The fast Fourier transform (FFT) is the cornerstone of many supercomputer applications and therefore needs careful performance tuning. Most often, however, the real performance of the FFT implementations is far below the acceptable figures. Within the frame of the hierarchical tiling approach, we explore several strategies for performance optimisations of the FFT computation, such as enhancing instruction-level parallelism, loop fusion, and reducing the memory loads and stores by using a special-purpose source code generator. Our approach is based on the principle of complete unrolling which we apply to modify the FT kernel of the NAS Parallel Benchmarks. In experiments on two different IBM SP2 platforms, we show performance improvements between 40% and 53% in comparison with the original code. Further, our 3-D FFT mega-step of the whole benchmark is faster than the corresponding FFT library call from the vendor-optimised PESSL numerical library. Finally, our approach for automatic generation of moderately optimised but specialised codes requires only a modest amount of programming effort.

Fortran Futures

Tony Hey

Abstract This paper discusses the economics of program optimisation and some of the challenges facing the parallel FORTRAN community. Industry takes a much broader view of cost optimisation than just speeding up a FORTRAN application code. Two examples of industrial "Fortran" projects undertaken by the Parallel Applications Centre at Southampton are used to illustrate these industrial concerns. The PROMENVIR project demonstrated the cost-effective use of a Europe-wide metacomputer, composed of idle workstations and parallel systems connected by a WAN, to explore the design space of an industrial application. By contrast, the TOOLSHED project was concerned with the integration of the design phase and grid generation phase with the simulation phase of the industrial design cycle. Use of the STEP data interchange format enabled the whole of the design cycle—design, simulation and visualisation—to be optimized. The paper concludes with a discussion of three important challenges for the continued health of parallel FORTRAN community. These are: the multiplicity of versions of parallel FORTRAN; competition from high-level scientific packages such as MATLAB and Mathematica;and finally, the inexorable rise in popularity of Java-based, network-centric computing.

Determining the idle time of a k-dimensional tiling

Karin Hogstedt

For a compiler to yield high-performance code we need to exploit the architectural features of the target machine. Tiling is used to improve both the locality and the utilization of the available parallelism, but with the more complicated memory hierarchies of today's computers tiling also needs to be applied at (possibly) all levels of the memory hierarchy. The goal of tiling is to minimize the execution time. Not to maximize the locality or available parallelism, which has been the optimization goal of many researchers in the past. Tiling should therefore be applied using a multi-level cost-function taking all memory levels, locality and parallelism into account.

We model the execution time by a recursive formula where the execution time of the iteration space at a certain level in the memory hierarchy depends on, among other things, the execution time of a tile, the idle time, loop overhead etc. We intend to model all these different parts with closed-form formulae, which combined give us the total execution time of the iteration space. Our work has been concentrated on deriving a formula for the idle time due to parallelism, i.e., the time a memory module spends waiting either for data computed on a different processor or at a synchronization point.

We define a sub-class of convex iteration spaces, so called rectilinear iteration spaces, for which we can derive a closed-form formula for the idle time due to parallelism, even though in the general case this problem only can be solved using linear programming. The main parameter of this formula is the rise, which intuitively is a measure of the difference between the shape of the iteration space and the shape of the tiles.

This is joint work with Larry Carter and Jeanne Ferrante.

Prospects for Effective Tiling in Java

Bowen Alpern & Susan Flynn-Hummel

To those habituated to the norms of optimizing Fortran programs, the land of Java optimizations is apt to be *terra incognita*. We will present a preliminary report on our initial forays into this strange country. Its exotic character is felt along two orthogonal dimensions (neither of them linguistic): its regulatory environment and its optimization terrain.

The regulatory environment of Fortran is quite *laissez-faire*: if there is agreement among a community that a particular program transformation preserves a program's (socially constructed) meaning, then one is free to apply the it. In Java, such transformations are expressly forbidden: a program must appear to have been executed in program order.

Fortran's optimization terrain is painfully dull: either the programmer tiles explicitly or the compiler tiles implicitly. Java's optimization terrain is exquisitely Baroque: implicit tiling might be attempted by a byte-code compiler, a byte-code optimizer, a Just-In-Time compiler, a dynamic compiler, a static compiler, a preprocessor, or even a garbage collector. In addition, native libraries might be consulted or local dialects (heavily persecuted) spoken.

Our talk is completely void of technical content. Being, rather, a travelogue, beginning with a summary of Java restrictions that tend to inhibit tiling, continuing with our (largely possitive) experience with explicit tiling in Java, moving on we present a taxonomy of implicit optimization strategies in Java. Finally, we conclude with our predictions as to whether Java will take over the world.

On-line automatic data placement in a parallel matrix library

Paul Kelly

Suppose we want to hide all the complexity of parallel programming inside a library of parallel operations. For example, we want to call it from languages for which building a parallelising compiler is unattractive. But we want to minimise data redistribution between library calls—but we can't do dataflow analysis of the calling program (in Visual Basic etc).

The problem: you can't tell how each routine's results are going to be used, so you miss the opportunity to select operand distributions and operator implementations which would avoid communication later.

To solve this we use delayed evaluation to capture the control-flow of user programs at runtime. Now we know the context in which values are used, we can propagate data placement constraints backwards. To make this work, we have to optimise really fast, so we have formulated the problem carefully using affine functions to represent data placement and constraints.

To further reduce the overheads, we detect when an equivalent DAG re-occurs, and re-use the result of optimising it. We also optimise incrementally, and perform further optimisation iterations if a potentially sub-optimal execution plan is re-used repeatedly.

Preliminary performance result were presented and there followed a discussion of extensions and applications of the work.

Distribution Assignment Placement: Cleaning up after (Data) Tiling

Jens Knoop

Data locality and workload balance are key factors for getting high performance out of data-parallel programs on multiprocessor architectures. Data-parallel languages like High Performance Fortran (HPF) thus offer means for specifying data distributions as well as for changing distributions dynamically in order to maintain these properties. Redistributions, however, can be quite expensive and significantly degrade a program's performance. In this talk, we report on a novel, aggressive approach for cleaning unnecessary distributions off a program. It works by eliminating *partially dead* and *partially redundant* distribution changes. Basically, this approach evolves from extending and combining two algorithms for these optimizations achieving each on its own optimal results. We demonstrate that combining them demands for a refined optimality investigation. Moreover, we show that the data-parallel setting leads to a family of algorithms of varying power and efficiency allowing user-customized solutions. The power and flexibility of the new approach are demonstrated by several examples ranging from typical HPF fragments to real world programs. Performance measurements additionally underline its importance and effectivity.

Keywords: Data-parallel languages, High Performance Fortran (HPF), dynamic data redistribution, data-flow analysis, optimization, partially dead and partially redundant assignment elimination.

This is joint work with Eduard Mehofer (University of Vienna, Austria).

Cache Miss Equations: Precise Miss Analysis for Program Transformations in Caches with Arbitrary Associativity

Margaret Martonosi

The peak performance of current microprocessors is improving at dramatic rates, but unfortunately memory performance has not kept pace. While caches are often effective at masking this performance gap, program transformations are often needed to allow programs to use cache most effectively.

In this talk, I discuss compile-time mechanisms for improving memory system behavior using "Cache Miss Equations" (CMEs). CMEs are a mathematical framework we have developed that allows the compiler to analyze potential cache miss points in scientific code by expressing cache conflicts in terms of a system of linear Diophantine equations. I describe and give performance results for precise loop transformation algorithms we have developed. These algorithms improve cache performance by analyzing the number of CME solution points given a particular memory hierarchy. Thus, they can specialize array positions and loop constructs to the given hardware structure. In particular, I present a tiling algorithm that uses CME analysis to eliminate self-interference misses in blocked linear algebra codes.

Data-centric Program Restructuring

Keshav Pingali

Modern high-performance machines have deep memory hierarchies, and optimizing the flow of data through the memory hierarchy usually has a dramatic impact on program performance. Unfortunately, memory hierarchies complicate the programming model enormously (for example, arrays cannot be treated as though they were random-access data structures). Carefully hand-coded libraries, such as the LAPACK library for dense numerical linear algebra, are one solution, but the utility of a library is limited to the particular problem domain for which it was developed.

In a quest for general-purpose tools, the compiler community has explored the use of program transformations like tiling (preceded by linear loop transformations) to optimize programs for memory hierarchies. However, performance improvements from tiling may be quite limited even for relatively simple codes like Cholesky factorization.

We have developed a new approach to program transformation called "data-centric program restructuring", one application of which is the optimization of programs for memory hierarchies. In this talk, we will discuss this technology and present performance numbers obtained on SGI's Octane workstations which show the performance advantages of data-centric restructuring over tiling. A practical consequence of this work is that our data-centric approach to optimizing programs for memory hierarchies is being incorporated into SGI's compiler product line.

Data Parallel Extensions for Maximizing Locality in Numerical Irregular Problems

Oscar Plata

The efficient programming of intensive numerical applications with irregular structure on parallel computers is a very complex task. In general, key properties of the problem that these codes solve are needed in order to obtain a good parallel code. However, these properties cannot be inferred from the code itself (at least, easily). Three approaches can be recognized to parallelize this class of codes, manual, user-annotated (specifically, data parallelism), and automatic parallelization. Manual parallelization usually involves drastic rewriting of the original sequential code, requiring a high development effort. However, this approach can take into account high-level problem properties, resulting in very efficient parallel codes. The data-parallel approach annotates the sequential code with directives that explicit the parallelism. Basically the directives specify data distributions and alignments. The compiler is in charge of the rest, the most tedious, part of the parallelization process. The efficiency of this approach depends on the ability of the data-parallel language to express problem properties, which is a difficult and open question for irregular applications. Finally, in the third approach, the compiler is completely in charge of the parallelization process. Currently there is active research in this area, as compiler techniques to recognize and take advantage of problem properties have to be discovered. No paralellizing compiler is, at the moment, able to efficiently parallelize complete irregular codes. This work focuses on the data-parallel approach, and how we can express problem properties using a limited number of user annotations. As simple molecular dynamics simulation code is taken as a running example. Nowadays, most of the production applications in this area have been manually parallelized. We present a parallelization strategy that offers: high efficiency similar to that of manual parallelization; original program structure is preserved in resulting parallel code; global data structures are decomposed in smaller local structures with the same organization; initial data decomposition and further communications are handled by calls to an existing runtime support. Finally, we analyze the introduction of HPF extensions to provide the compiler with information enough to guess the role of each data structure in particle codes.

This is joint work with Guillermo P. Trabado Emilio L. Zapata.

Group-Based Dependence-Free Clustering

Hans-Werner Pohl

Dependence-free clustering of data structures with affine dependences can be regarded as a non-rectangular generalization of alignment. Because affine dependences (in their infinite extension) are unions of finitely many cosets of subgroups of $\mathcal{Z}^n$, we use group-theoretic techniques to perform the clustering. All the computations can be done efficiently on basis of the Hermite normal form.

Loop Partitioning versus Tiling for Cache based Multiprocessors

Fabrice Rastello

In this paper, an efficient algorithm to implement *loop partitioning* is introduced and evaluated. We improve recent results of Agarwal, Kranz and Natarajan in several directions. We derive a new formulation of the cumulative footprint, which enables us to deal with arbitrary parallelepiped-shaped tiles, as opposed to rectangular tiles. We design an efficient heuristic to determine the optimal tile shape. We illustrate the superiority of our algorithm on the same examples as by Agarwal et al. to ensure the fairness of the comparisons.

Key words: compilation technique, hierarchical memory systems, loop partitioning, tiling, cache, data locality, footprint.

Partitioning and Structures Scheduling for SAREs

Tanguy Risset

We propose a strategy for partitioning systolic arrays expressed in the formalism of recurrence equations. This partitioning is realised with multi-dimensionnal scheduling (to obtain a linear array) and LPGS (or LSGP) partitioning at the end. One advantage of this approach is the control of the arrays described can be easily automatically generated. Also, with the help of a flexible schedule tool, structured scheduling can be added to the method, allowing to re-use hardware and keep the designer structuring.

Performance Optimization of a Class of Loops

P. Sadayappan

The problem of optimizing a class of parallel loops for parallel execution is considered. The problem is motivated from examples in computational physics where multi-dimensional summations of products of arrays are performed. The overall optimization problem is divided into three sub-problems: operation count optimization, communication optimization, and data locality optimization. The operation count optimization problem is found to be NP-complete. A polynomial time dynamic programming algorithm is developed for the communication optimization problem. The issues arising with the data locality optimization problem are discussed: loop permutation, tiling, and loop fusion. Partial solutions to the data optimization problem are presented.

How To Juggle

Rob Schreiber

We describe a new, practical, constructive method for solving the well-known conflict-free scheduling problem for the locally sequential, globally parallel (LSGP) case of systolic array synthesis. A loop nest and a linear mapping to virtual processors is given, as is a clustering of rectangular arrangements of virtual processors into physical processors. A solution to the scheduling problem is a linear map of iteration indices to time that satisfies linear inequality constraints determined by loop carried-dependences. The schedule is conflict-free if no two iterations are scheduled simultaneously on the same processor. It is tight if it juggles and, in the steady state, all processors are busy every cycle.

Earlier attempts to solve this problem by Darte and Delosme provided a solution with an important practical disadvantage, which we discuss below. Megson and Chen later used Darte's analytic technique to provide a partial solution to the problem. Here, we provide a closed form solution that enables the enumeration of all tight schedules. The new method has been incorporated into a software system for the automatic synthesis of hardware accelerators developed by HP Labs.

Regular State Machines

Jürgen Teich

In this talk, we introduce the notion of a model called regular state machines (RSM) that characterizes a class of state-transition systems with a regular, repetitive, and (frequently) unbounded number of states and state transitions. An example of first choice is the state of a fifo queue, as written by a process and read by another process.

We show that many models of parallel computation, represented ususally by process graphs, may be effectively and finitely described in the state-space by an RSM, e.g., Petri nets and subclasses thereof.

Mathematically, a RSM is described by a set of indexed states, defined over an index domain which is a lattice bounded by a polyhedron.

We further show how properties such as state reachability, existence of periodic schedules, and memory analysis can be done efficiently based on results obtained in the area of mapping regular algorithms to processor arrays and loop parallelization...

This is joint work with Lothar Thiele

Eliminating Conflict Misses for Tiled Codes

Chau-Wen Tseng

Tiling is a powerful compiler technique for exploiting data locality in scientific codes. However, previous research has shown conflict misses occurring due to caches with limited associativity can significantly degrade the performance of tiled codes. Two approaches for avoiding conflict misses are 1) carefully picking tile sizes, and 2) copying tiles to contiguous buffers at run time. In this research, we improve the flexibility and performance of existing approaches through intra-variable padding, as well as provide extensive experimental evaluation using a matrix multiply case study.

Previous techniques for avoiding conflict misses include picking the largest non-conflicting square tile [1] and some rectangular tile generated using the Euclidean GCD algorithm [2]. By varying the problem size of matrix multiply, we discover many sizes where both algorithms are forced to select tiles small relative to the cache size. As a result, the amount of loop overhead degraded the performance of the tiled code, sometimes by 20% or more on a Sun UltraSparc. We provide a simple padding algorithm for solving this problem.

Our padding heuristic simply examines tiles selected when the target array column is padded from 0 to 7 elements. The pad producing the "best" tile size is chosen. If possible, array declarations are modified. Otherwise the array is first precopied to a padded array.

We experimentally evaluate the performance of matrix multiply for problem sizes between 100 to 400 for different versions of tiling on a Sun UltraSparc, using both timings and cache simulations. Results show padding can significantly improve the performance of tiling, avoiding pathological array sizes which force small tiles. The overhead of precopying the added array is under 3% of the execution time, particularly for larger problem sizes. Experiments also show that copying tiles to contiguous buffers is quite effective, and larger tiles should be selected than proposed in previous papers. Overall, we find padding used in conjunction with tile size selection to be a useful compiler transformation for eliminating conflicts in tiled codes.

This is joint work with Gabriel Rivera.

References

[1] Lam, Rothberg, Wolf: ASPLOS'91
[2] Coleman, McKinley: PLDI'95