Dagstuhl Seminar
on
Instruction-Level Parallelism and Parallelizing Compilation

Organized by

D. K. Arvind (University of Edinburgh)
Kemal Ebcioglu (IBM T. J. Watson Research Center)
Christian Lengauer (Universsitt Passau)
Keshav Pingali (Cornell University)
Robert S. Schreiber (Hewlett-Packard)

Schlo Dagstuhl 18. ˘ 23.4.1999

Contents

1 Preface

1.1 Introduction

Parallel programming has been around for three decades and has remained a diŒcult øeld. The biggest challenge arises when the main purpose of parallelism is to increase performance, i.e., computation speed. Parallel programs are notoriously hard to get correct and eŒcient. Although progress has been made on the semantics and veriøcation of parallel programs in certain domains, no practical technique for the development of reliable, portable application parallelism for high performance has been achieved.

One approach towards this goal is to unburden the programmer from the diŒcult task of handling parallelism and delegate this to the compiler or the machine architecture. The research area which gives the compiler the control over the parallelism is parallelizing compilation, the research area which lets the machine infuse the parallelism is instruction-level parallelism (ILP).

The aim of the seminar was to bring together these two research areas, which have developed side by side with little exchange of results. Both areas are dealing with similar issues like dependence analysis, synchronous vs. asynchronous parallelism, static vs. dynamic parallelization, and speculative execution. However, the diœerent levels of abstraction at which the parallelization takes place call for diœerent techniques and impose diœerent optimization criteria.

In instruction-level parallelism, by nature, the parallelism is invisible to the programmer, since it is infused in program parts which are atomic at the level of the programming language. The emphasis is on driving the parallelization process by the availability of architectural resources. Static parallelization has been targeted at very large instruction word (VLIW) architectures and dynamic parallelization at superscalar architectures. Heuristics are being applied to achieve good but, in general, suboptimal performance.

In parallelizing compilation, parallelism visible at the level of the programming language must be exposed. The programmer usually aids the parallelization process with program annotations or by putting the program to be parallelized in a certain syntactic form. The emphasis has been on static parallelization methods. One can apply either heuristics or an optimizing algorithm to search for best performance. Resource limitations can be taken into account during the search, or they can be imposed in a later step, e.g., through tiling or partitioning the computation domain.

1.2 Summary of the Presentations

Embedded software applications have traditionally tended to use low-level languages and hand-crafted techniques for optimizing execution time and memory usage. Given the scope for exploiting parallelism in embedded software, especially in multimedia applications, and the emergence of ILP processors, such as VLIW ones, there is a growing body of work investigating automatic parallelization of high-level programs aimed at ILP targets. A number of these compiler infrastructure projects were presented at the workshop.

The Esprit OCEANS project (Eisenbeis) is aimed at embedded VLIW architectures, with emphasis on understanding and exploiting interactions between high-level optimizations, such as loop unrolling, and low-level ones, such as software pipelining

The ACROPOLIS project (Omns) at IMEC considers the impact of data organization in embedded applications on performance metrics such as instruction throughput and power consumption. The latter is fast becoming an important consideration for multimedia applications running on mobile appliances. The approach in this project is to inform the choices in the parallel compilation process of the impact of the dominant costs of data transfers and complex data manipulations on the overall performance.

The TriMedia project (Augusteijn) at Philips has developed a compilation environment for embedded programs written almost exclusively in C/C++, and targeted at the 5-issue TriMedia VLIW processor. It supports predicated execution and special operations for DSP algorithms, such as vector instructions on subwords.

Vectorizing techniques for exploiting sub-word parallelism in ILP architectures was the subject of two further talks from the University of Vienna (Krall) and the Indian Institute of Sciences, Bangalore (Govindarajan); and the method of predicated execution for exploiting ILP in the EPIC (Explicitly Parallel Instruction Computing) architecture was the subject of a paper from University of California, San Diego (Ferrante).

The traditional instruction sets of processors have been extended to exploit eŒciently sub-word parallelism, in which a number of short data elements are packed in a single register and data-parallel operations are executed on them in parallel. Examples of these so-called multimedia extensions include, the Visual Instruction Set for the UltraSPARC processor, the AltiVec for the PowerPC, the MMX extension for the Pentium processor, and the MAX-2 instruction set of the PA-RISC processor. At present, there is little or no compiler support to exploit sub-word parallelism ˘ the user is expected to handcode large parts of their application in assembly language.

Krall uses the technique of vectorization by unrolling to automate this process. Data dependence analysis and dynamic run-time checking are used to handle unaligned memory accesses. Govindarajan uses standard vectorization techniques on loops which are tailored for short vector lengths.

Predicated execution is one of many approaches used to ønding instructions that can be executed simultaneously in ILP architectures. One of the drawbacks, however, is that predicated code presents challenges to traditional compiler optimizations. Ferrante presented an extension to the well-known Static Single Assignment form, called the Predicated Static Single Assignment form, which, when used in conjunction with speculation and control height reduction, enables instructions to be issued at their true data dependence height.

The microarchitecture of future processors has been the subject of intense debate and was the topic of two talks ˘ from the University of Wisconsin (Sohi) and the University of Delft (Corporaal). Both speakers recognized common problems ˘ future architectures have to contend with increasing workloads and longer communication and memory latencies ˘ but they advocated quite diœerent solutions. Microarchitectures have traditionally been based on certain observable program behaviours, such as spatial and temporal locality. Sohi advocated the need for information about program structure ˘ the data and control relationship between instructions, i.e., the relationship which causes the observable behaviour ˘ to be the basis for the design of future microarchitectures. Corporaal recognized communication as being of primary importance in the design of future microarchitectures. In the transport-triggered architectures, for instance, the communication between functional units, and with the register øles, are programmed explicitly; the computation is now a side-eœect, triggered by the communication. All communication inside the microarchitecture is visible to the compiler, which leads to a number of communication-level optimizations that the compiler can perform to increase the performance. Embedded programs can be analyzed and implemented on a transport-triggered architecture with an optimal number of functional units and communication pattern.

Embedded systems have requirements such as low cost and low power consumption, in addition to high performance. In many cases, oœ-the-shelf processors cannot meet the performance speciøcation. The design of embedded systems tuned to a particular application in a given domain demands an approach which takes an integrated view of the software and hardware design of the system. Cadence Design Systems (Martin, Hoover) presented an approach to performance estimation of software running on the processor by using a virtual instruction set model, and a scheduling model for the real-time operating system.

Looking beyond ILP, talks from the University of Minnesota (Yew), Chalmers University (Stenstrm), and the University of Jena (Unger) presented ideas on compilers that exploit thread-level and instruction-level parallelism. Yew discussed the Agassiz compiler which is targeted at a concurrent multithreaded architecture and supports speculative execution at both the thread and instruction level, in addition to run-time data dependence checking and very fast communication between thread processing units. High-level program information, such as aliases, and cross-iteration data dependences, are passed from the thread-level compiler to the ILP one. Unlike Yew, Stenstrm does not assume that the target architecture supports speculative execution. His ideas for thread-level data speculation are implemented entirely in software. The aim is to demonstrate that the overhead is acceptably low with reasonable performance gains through the exposed parallelism. Unger presented the Simultaneous Speculation Scheduling, which is a combined compiler and architecture technique for multithreaded processors. The speculation is controlled entirely by the compiler and is aimed at simultaneous multithreaded processors.

A number of talks discussed the dependence analysis of programs and their optimization for ILP processors. The paper from the University of Jena (Zehendner) describes a method for memory reference disambiguation on assembly language code for increasing ILP. The method derives value-based dependences between memory operations and is integrated in the SALTO system. The presentation from the University of Versailles (Collard) looked at statically deriving the probability, in the case of arrays, that two references access the same memory location. This is useful in moving a load speculatively above a possibly aliasing store. Mills Strout from the University of California, San Diego, presented a method of register tiling which exploits data dependence analysis to reduce storage requirements in superscalar ILP architectures. Jens Knoop from the University of Dortmund presented an automata-theoretic approach to interprocedural data Æow analysis. The structural behavior of the program is modelled by an appropriate pushdown system; the reaching deønitions boils down to a reachability problem on pushdown systems.

One dominant target of parallelizing compilers is the domain of nested loop programs. A number of presentations in the seminar came from this domain.

The computation domain of a loop nest is often modelled by embedding the loop steps on a high-dimensional integer grid. One problem which arises is how to use the structure of this domain to advantage for an eŒcient execution. Griebl from the University of Passau presented an algorithm to shorten the parallel execution which uses breaks in the dependence structure of a polyhedral computation domain. Quiller from IRISA in Rennes presented a method for the execution of domains which are unions of polyhedra. Rather than testing at run time, whether a computation point falls into the domain or lies outside ˘ which can lead to a lot of overhead ˘ he splits the domain into pure polyhedra and scans these without any run-time tests. Darte from the ENS Lyon uses a combination of loop shifting and loop compaction to shorten the parallel execution of a program composed of separate loop nests. Robert from the same school extends static techniques of partitioning the domain to the context of limited computational resources with diœerent-speed processors.

At a lower level of abstraction, Coelho restructures mathematical expressions to evaluate them more eŒciently. Gregg addresses the software pipelining of loops with branches in the loop body. Moore presented work at the University of Frankfurt on associative architectures for the support of run-time parallelization.

1.3 Conclusions

The seminar exposed the exciting developments taking place in parallel computer architecture. It also exposed the heavy burdens which are being placed in compilers by current parallel machines. The eŒcient use of performance-increasing hardware such as cache hierarchies, pipelined functional units and predication call for highly sophisticated analysis and code generation techniques. It remains to be seen how the portability of parallel software can be maintained in this scenario. Portability is essential. After all, a parallel computer whose main purpose is high performance becomes obsolete after about øve years.

An issue of the International Journal of Parallel Programming dedicated to this seminar will appear in due course.

D. K. Arvind   K. Ebcioglu   C. Lengauer   K. Pingali   R. S. Schreiber

2 Abstracts

Instruction-Level Parallelism and Parallelising Compilation

D. K. Arvind

University of Edinburgh, Scotland

The øelds of Instruction-Level Parallelism (ILP) and Parallelising Compilation have developed concurrently, but with little exchange of results. Four issues of common interest were identiøed:

- dependence analysis,
- synchronous versus asynchronous parallelism,
- static versus dynamic parallelisation,
- speculative execution.

It is argued that the design of future compilers and architectures, and the division of responsibility between them, will be inÆuenced by a number of factors:

- the growth of new application areas such as embedded systems
- the resulting changes in the nature and volume of the workload
- new design methods for implementing entire systems in silicon

Computers in the future will be more pervasive ˘ portable rather than tethered to a desk. There will be demand for greater computational speeds at lower energy costs. Improvements in silicon technology suggest a billion transistor chip by the middle of the next decade. One of the design considerations for future architectures is the high cost of on-chip communications. Architectural structures which exploit information locality and decentralise their controls will be likely winners. Also, the processor cores have to live cheek by jowl with analogue devices operating at radio frequencies, and noisy micromechanical structures. The new application areas suggest a shorter design cycle for these systems, which emphasises architectures that can be composed quickly from pre-designed and pre-veriøed building blocks, i.e. a design method which is compositional at the diœerent levels of abstraction, right down to the silicon implementation. It was argued that a fully asynchronous design method captures the following features:

- a network of functional units which operate concurrently and communicate asynchronously using a local protocol
- they consume a fraction of the total power of an equivalent clocked design
- they are immune to electromagnetic interferences
- the architectures exposes øne grain concurrency
- the designs are compositional even at the silicon level

It was proposed that processing cores of future embedded systems could be fully asynchronous ILP processors.

Vectorizing Techniques for Exploiting Instruction-Level Parallelism

Ramaswamy Govindarajan (with N. Sreraman)

Indian Institute of Science Bangalore, India

Subword parallelism is a technique in which multiple short data elements are packed in a single register and data-parallel operations are executed on them in parallel. Extending the instruction set architecture to exploit subword parallelism is becoming increasingly popular in modern processors. Despite this popularity, compiler support for exploiting such subword parallelism is widely absent and programmers are expected to handcode (parts of) their application in assembly.
In this work, we use standard vectorizing techniques to identify vectorizable loops and generate in-lined MMX assembly code that exploits subword parallelism. The various phases of standard vectorization techniques (dependence

analysis, loop distribution and øssion, strip mining, scalar expansion and code generation) are tailored for subword parallelism which deal with short vector lengths. We use SUIF (Stanford University Intermediate Form) for performing vectorization. Initial experimental results on multimedia applications are also presented.

Circuit Retiming Applied to Decomposed Software Pipelining

Alain Darte (with Pierre-Yves Calland, Guillaume Huard and Yves Robert)

Ecole Normale Superieure de Lyon, France

The idea of decomposed software pipelining is to decouple the software pipelining problem into a cyclic scheduling problem without resource constraints and an acyclic scheduling problem with resource constraints. In terms of loop transformation and code motion, the technique can be formulated as a combination of loop shifting and loop compaction. Loop shifting amounts to moving statements between iterations, thereby changing some loop-independent dependences into loop-carried dependences and vice versa. Then, loop compaction schedules the body of the loop considering only loop independent dependences, but taking into account the details of the target architecture. In the ørst part of this talk, I recall the main theoretical results for the software pipelining problem and I survey existing heuristics, mainly modulo scheduling, kernel recognition algorithms and compact/shift algorithms, discussing their intuitive ideas and weaknesses. In the second part, I focus on decomposed software pipelining: I show how loop shifting can be optimized so as to minimize both the length of the critical path and the number of dependences for loop compaction. Both problems (and the combination) are polynomially solvable with fast graph algorithms, the ørst one by using an algorithm due to Leiserson and Saxe, the second one by designing a variant of minimum-cost Æow algorithms.

Index Set Splitting

Martin Griebl (with Paul Feautrier and Christian Lengauer)

University of Passau, Germany

There are many algorithms for the space-time mapping of nested loops. Some of them even make the optimal choices within their framework. We propose a preprocessing phase for algorithms in the polytope model, which extends the model and yields space-time mappings whose schedule is, in some cases, orders of magnitude faster. These are cases in which the dependence graph has small irregularities. The basic idea is to split the iteration domain of the loop nests into parts with a regular dependence structure and apply the existing space-time mapping algorithms to these parts individually.
This work is based on a seminal idea in the more limited context of loop parallelization at the code level. We elevate the idea to the model level (our model is the polytope model), which increases its applicability by providing a clearer and wider range of choices at an acceptable analysis cost.
Index set splitting is one facet in the eœort to extend the power of the polytope model and to enable the generation of competitive target code.

Algorithmic Issues for Heterogeneous Computing Platforms

Yves Robert (with Pierre Boulet, Jack Dongarra, Fabrice Rastello and Frdric Vivien)

Ecole Normale Superieure de Lyon, France

In the framework of fully permutable loops, tiling has been extensively studied as a source-to-source program transformation. However, little work has been devoted to the mapping and scheduling of the tiles on physical processors. Moreover, targeting heterogeneous computing platforms has, to the best of our knowledge, never been considered. In this talk, we extend static tiling techniques to the context of limited computational resources with diœerent-speed processors. In particular, we present eŒcient scheduling and mapping strategies that are asymptotically optimal. The practical usefulness of these

strategies is fully demonstrated by MPI experiments on a heterogeneous network of workstations.
Along these lines we also discuss some algorithmic issues when computing with a heterogeneous network of workstations (the typical poor man's parallel computer). How is it possible to implement eŒciently numerical linear algebra kernels like those included in the ScaLAPACK library? Dealing with processors of diœerent speeds requires the use of more involved strategies than purely static block-cyclic data distributions. Dynamic data distribution is a ørst possibility but may prove impractical and not scalable due to communication and control overhead. Static data distributions tuned to balance execution times constitute another possibility but may prove ineføcient due to variations in the processor speeds (e.g., because of diœerent workloads during the computation). There is a challenge in determining a trade-oœ between the data distribution parameters and the process spawning and possible migration (redistribution) policies. We introduce a semi-static distribution strategy that can be reøned on the Æy, and we show that it is well suited to parallelizing several kernels of the ScaLAPACK library such as LU or QR decomposition.

Storage Allocation for Register Tiling, Instruction-Level Parallelism and Parallelizing Compilation

Michelle Mills Strout (with Larry Carter and Jeanne Ferrante)

University of California at San Diego, USA

We look at the problem of exposing instruction-level parallelism under register constraints. Register tiling is one technique which can be used to expose ILP to a superscalar architecture. Typically, larger tile sizes contain more low-level parallelism. However, the number of registers available restricts the tile size. We apply the idea of the occupancy vector (OV) storage mapping to register allocation in order to reduce the storage requirements in terms of registers, thus allowing larger register tiles. The technique of determining an occupancy vector uses data dependences and schedule information to ønd an eŒcient storage reuse pattern. In simulations of an out-of-order processor, our OV register allocation along with a wavefront parallel schedule performs better than techniques based on scalar replacement.

Data Dependence Analysis of Assembly Code

Eberhard Zehendner (with Wolfram Amme and Peter Braun)

University of Jena, Germany

Determination of data dependences is a task typically performed on high-level language source code in today's optimizing and parallelizing compilers. Less work has been done in the øeld of data dependence analysis on assembly language code, but this area will be of growing importance ˘ in particular, for increasing ILP. A central element of a data dependence analysis in this case is a method for memory reference disambiguation which decides whether two memory operations must/may access the same memory location. We describe a new approach for determination of data dependences in assembly code. Our method is based on a sophisticated algorithm for symbolic value propagation, and it can derive value-based dependences between memory operations instead of address-based dependences only. We have integrated our method into the SALTO system for assembly language optimization. Experimental results show that our approach greatly improves the accuracy of the dependence analysis in many cases.

OCEANS: Optimizing Compilers for Embedded Applications

Christine Eisenbeis

INRIA Rocquencourt, France

The objectives of the OCEANS ESPRIT project is to investigate and develop advanced compiler insfrastructure for embedded VLIW processors. This combines high- and low-level optimisation approaches within an iterative

framework for compilation. We present three kinds of results that have been obtained thanks to the OCEANS compiler framework. First, one allows to trade-oœ globally between code size and code performance for a set of code pieces of the same program. Second, one compares diœerent versions of high-level loop unrolling and low-level software pipelining. Third, one analyzes the interaction between loop unrolling and loop tiling by exploring the space of a range of possible unrolling and blocking factors.

Compilation Techniques for Multimedia Processors

Andreas Krall (with Sylvain Lelait)

University of Vienna, Austria

The huge processing power needed by multimedia applications has lead to multimedia extensions in the instruction set of microprocessors. Examples of these extended instruction sets are the Visual Instruction Set of the Ultra-SPARC processor, the AltiVec instruction set of the PowerPc processor, the MMX extensions of the Pentium, and the MAX-2 instruction set of the HP PA-RISC processor. Currently, these extensions can only be used by programs written in assembly language, through system libraries or by calling specialized macros in a high-level language. Therefore, these instructions are not used by most applications.
We propose two code generation techniques to produce native code using these multimedia extensions for programs written in a high-level language: classical vectorization and vectorization by unrolling. Vectorization by unrolling is simpler than classical vectorization, since data dependence analysis is reduced to acyclic control Æow graph analysis. Furthermore, we address the problem of unaligned memory accesses. This can be handled by both static analysis and dynamic run-time checking. Preliminary experimental results for a code generator for the UltraSPARC VIS instruction set show that speedups of up to a factor of 4.8 are possible, and that vectorization by unrolling is simpler but as eœective as classical vectorization.

Compiling for TriMedia

Lex Augusteijn

Philips Research Eindhoven, The Netherlands

The TriMedia processor is a 5-issue slot VLIW, equipped with 128 32-bit registers, predicated execution and special operations for DSP algorithms, like vector instructions on subwords. The TriMedia is programmed almost exclusively in C and C++, hardly on the assembly level. The C/C++ compilation process consists of two parts: the core compiler and the scheduler, communicating via an intermediate format called decision trees. These dtrees are trees of basic blocks, containing a data Æow graph of operations. The core compiler performs global register allocation and computes the partial order of operations in the data Æow graph, the scheduler schedules them into a total order, obeying the resource constraints of the machine, while performing local register allocation. In order to exploit the potential for ILP, the compiler attempts to generate large decision trees with few dependencies. Techniques applied to achieve this are:

- grafting (tail duplication), which copies basic blocks in order to enlarge dtrees,
- loop unrolling and function inlining,
- extensive alias analysis,
- if-conversion, which introduces predicated execution and phi-nodes in the dtrees.

Proøling is used to control both these transformations in the compiler and speculation in the scheduler. At the level of the program, restrict pointers are used to support the alias analysis of the compiler and the whole TriMedia instruction set is available via built-in functions, the so-called custom ops. The compilation has been validated for correctness and performance by standard benchmark suites like Plumhall, Nullstone, Spec92 and Spec95, as well as a program generator that generates self-testing random C programs.

The Multimedia Compilation Project ACROPOLIS

Thierry J.-F. Omns

IMEC, Belgium

For most advanced real-time multimedia processing applications (video, imaging, telecommunication), the manipulation of complex data has a major or even dominant eœect on the cost of the global system, also when compiled to (parallel) multimedia processors like the Philips TriMedia, TI C60 or C80, Chromatic/Mpact, and even on general-purpose processors with multimedia extensions like the Intel MMX. This is mainly due to the large impact of the memory organization and the global data transfer between processor storage and data path units on the cycle count and system power consumption.
Currently, due to design time restrictions, the system designer usually has to select ˘ on an ad-hoc basis ˘ a single promising path in the huge decision tree from abstract speciøcation to more reøned C speciøcation, which is then input to the conventional (parallel) compiler environments. To alleviate this situation, there is a need for more support during the compilation and also for fast and early feedback at the algorithm level without going all the way to the ønal code. When the design space has been suŒciently explored at a high level and when the most promising candidate has been identiøed, also the ønal choice has to be coded in the transformed (C) speciøcation.
Therefore, based on our large experience in hardware oriented memory management (see Atomium project), we are developing a tuned methodology, formal models and system-level compilation techniques which are intended as a precompilation stage to the conventional (parallel) compiler environments. In addition, we are working on prototype tools and an environment supporting the most error-prone and critical tasks in this data storage and transfer exploration and optimisation. Promising results have been achieved by applying this approach manually to a number of complex real-life video and image processing algorithms.

Hardware-Software Codesign and Software Estimation

Grant Martin and Christopher Hoover

Cadence, USA

Two recent trends in electronics design are converging: technology-driven integration which is leading to complete Systems-On-Chip (SOC) devices for embedded systems; and advances in systems design which support the hardware-software codesign process for embedded devices. The design methodology is further changing towards integration platform-based design approaches to reduce design time for SOC.

The presentation describes some recent R&D work at Cadence which is developing a new methodology and technology for hardware-software codesign. This approach is particularly interesting when combined with the integration platform concept to create design vehicles for rapid derivative creation, although it is more general and can be applied to the creation of whole new architectures. The methodology relies on the modeling of applications behaviour using a compositional approach which allows the integration of models from diœerent computational domains. In an orthogonal manner, candidate architectures consisting of functional resources (which may be IP blocks or virtual components) and communications resources are captured. An explicit mapping between behaviours and architectural resources is used to drive a performance analysis process, which then guides designers towards optimising the architecture and choice of components to meet system design requirements.

Mapping behaviours to components which are programmable processors implies that the functions will be implemented as software tasks. The performance implications of this mapping (latency, throughput) are estimated using an abstract virtual instruction set model for the target processor, a scheduling model for the real-time operating system, and a partial compilation of the tasks into the virtual instruction set. The tasks are simulated at native workstation speed, but their estimated performance on the target processor is derived during the simulation and used as the basis for performance analysis. For intensive data manipulation tasks, e.g., running on a DSP, a diœerent approach using precharacterized kernel functions is used with a static schedule. Characterization methods for these techniques are brieÆy described.

Microarchitectural Mechanisms for Future ILP Processors

Gurindar Sohi

University of Wisconsin at Madison, USA

Microarchitectural techniques of the next decade will have to be more eŒcient and scalable, in order to handle growing workloads and longer communication and memory latencies. New techniques will have to be developed for these scenarios, and a new underlying basis will be needed for these techniques.
To date, microarchitectural mechanisms have been based on observable empirical behaviour of a program, e.g., temporal and spatial locality. For future mechanisms, a new basis will be needed. We believe that information about program structure, the data and control relationships between instructions, i.e., causal relationships which cause the observed empirical behaviour can be used as a powerful framework for new techniques. We argue that program structure information has several inherent advantages over frameworks that associate information either with instructions in isolation or with data. We present summaries of four novel methods that apply program structure information to memory system problems ranging from dynamic scheduling of memory operations and optimizing data cache bandwidth to prefetching recursive data structures and optimizing cache coherence protocols.

On Compiler Issues for Concurrent Multithreaded Architectures

Pen-Chung Yew

University of Minnesota, USA

Supporting concurrent execution of multiple threads on a single chip is a promising approach to boosting performance beyond existing superscalar architectures. Concurrent multithreaded architectures (CMAs) take advantage of the compilation techniques and run-time hardware support to exploit both instruction-level and thread-level parallelism, thus allowing a wider instruction issue rate per clock cycle.

New compiler techniques are needed to exploit both thread-level and instruction-level parallelism on such architectures for general-purpose applications (e.g., SPEC benchmark programs). They require the compiler to go beyond exploiting ILP in the innermost loops as in traditional ILP compilers. CMAs support speculative execution at both the instruction and the thread level. They also provide run-time data dependence checking and very fast communication between thread processing units. Such architectures allow do-while loops and do-across loops to be exploited very eŒciently at any loop nesting level.

The Agassiz compiler, currently being developed at the University of Minnesota, is an integrated parallelizing compiler targeting CMAs. It has a parallelizing compiler as its front-end for both C and Fortran, and an optimizing compiler based on gcc as its back-end to exploit ILP. A well deøned data structure is used to export high-level program information such as aliases, cross-iteration data dependences, and interprocedural information from the front-end parallelizing compiler to the back-end ILP compiler.

In this talk, we present some compiler techniques needed to exploit thread-level parallelism in SPECint95 programs on CMAs, and show their performance data using simulations.

The Potential of a Software-Only Thread-Level Data Speculation System for Multiprocessors

Per Stenstrm (with Peter Rundberg)

Chalmers University, Sweden

Because parallelizing compilers cannot statically detect all dependences, a lot of parallelism cannot be exploited. Thread-level data speculation systems detect data dependences at run time and allow threads that the compiler would conservatively deem dependent to be speculatively executed in parallel. Only if a dependence violation is detected must the speculatively executed threads be terminated. Previously proposed speculation systems require quite aggressive changes to memory hierarchies in multiprocessors.
We propose a software-only approach to design speculation systems. Each load and store that is potentially involved in a dependency invokes highly tuned code sequences that remove or detect potential dependences. Through simulation and analytical modeling techniques of a prototypical implementation of such a system, we show that our approach to thread-level data speculation imposes acceptably low overheads and may result in reasonable performance gains through the parallelism that is exposed.

Predicated Static Single-Assignment Form for ILP

Jeanne Ferrante (with Lori Carter, Beth Simon, Brad Calder and Larry Carter)

Uuniversity of California at San Diego, USA

In the quest for faster processors, new architectures that can issue multiple instructions per cycle have been developed. The EPIC (Explicitly Parallel Instruction Computing) architecture is an example of such a design with a special feature called Predicated Execution. Predicated Execution helps the compiler meet the challenge of ønding instructions that can be executed simultaneously. However, code that has been predicated presents challenges to performing traditional compiler optimizations. In this work, we present the Predicated Static Single-Assignment form (PSSA) to aid in the analysis and

optimization of predicated code. PSSA is an extension of traditional Static Single-Assignment form, which has been used as the basis for more eŒcient and powerful optimizations on previous architectures. We show that PSSA can be used in conjunction with speculation and control height reduction to enable instructions to be issued at their true data dependence height, assuming a machine with unlimited resources. We are currently implementing these algorithms in the Trimaran system.

Computation in the Context of Transport-Triggered Architectures

Henk Corporaal

Delft University of Technology, The Netherlands

Billions of embedded systems are sold annually. Processors for these systems often have speciøc requirements like low cost, high performance and low power consumption. Oœ-the-shelf processors can not always fuløl these requirements simultaneously. Using a templated processor architecture, which can be tuned for a certain application (domain) oœers a solution. This paper highlights such a templated architecture, called transport-triggered architecture (TTA). This architecture combines extreme Æexibility, modularity and scalability, while being simple (and therefore easy to generate automatically) and oœering good acost-performance ratio. The TTA resembles a VLIW (very long instruction word) architecture, but its programming model is completely diœerent. Instead of specifying the operations to be performed by the function units, TTA programs explicitly specify the transports between function units, and the transports of data from/to the register øles. The operations occur as a side eœect of these transports.
Giving the compiler control about all the internal data transports, opens a number of new types of (transport-level) optimizations. These optimizations are exploited by our compiler. They result in a register traŒc reduction of at least 50%. As a result the number of register ports can be reduced substantially. Furthermore, the connectivity between function units (needed for bypassing results) can be reduced with up to 80%, depending on the type of application. This has been reported in earlier papers.

After an introduction to TTAs and their characteristics, two important topics are discussed in this talk. First, a new scheduling method, exploiting the available ILP (instruction-level parallelism) and the TTA-speciøc optimizations, is described. The new aspect of this scheduling method is its integrated register allocation. We will compare results from this method with both early register allocation (i.e., before scheduling) and late register allocation. It will be shown that integrated allocation is superior when registers are scarce or, equivalently, when applications show a high register pressure.
Second, we discuss how to tune the architecture for a certain application (domain). A framework has been developed allowing a quantitative search of the design space, and giving the so-called Pareto points in the cost-performance space. This framework assumes a given set of function units, supporting a certain set of, usually, basic RISC-like operations. Although the framework supports more complex operations as well, the designer has to decide which complex operations to include. To support this decision, a tool has been designed which can detect arbitrary patterns in any data dependence graph. There is no restriction on the number of inputs and outputs of the patterns, and patterns do not have to be tree-shaped (as is the case for many other pattern detectors). It will be shown that by adding about 20% extra operations, which are simple combinations of two basic operations, the total number of operations can be reduced by 40% for a large set of applications.

Array-Tailored Analyses vs. MFP Analysis

Jean-Franois Collard

University of Versailles, France

In this presentation, I describe two analyses that are tailored for arrays. The goal of the former is to derive statically the probability that two references access the same memory location. Based on this probability, the compiler may decide to move a load speculatively above a possibly aliasing store. In this work, support for speculative loads is provided by HP PlayDoh.
The latter analysis derives reaching deønitions on array elements. In addition, the analysis is able to pinpoint precisely which instance of which write reference actually deønes the value. This analysis is contrasted with MFP analyses and with analyses based on integer linear programming.

An Automata-Theoretic Approach to Dependence Analysis

Jens Knoop (with Javier Esparza)

University of Dortmund, Germany

In this talk, we present an automata-theoretic approach to the interprocedural variant of the reaching deønitions problem, a speciøc kind of dependence analysis. Fundamental for this approach are recent results on abstract reachability problems in inønite state systems, in particular, pushdown systems. We demonstrate that these results, which have been discovered in the course of extending the automata-theoretic approach to model checking from ønite state systems to inønite state systems, have immediate applications in interprocedural data Æow analysis, in particular, reaching deønitions analysis. Modelling the structural behaviour of an interprocedural program by an appropriate pushdown system, the reaching deønitions problem boils down to a reachability problem on pushdown systems. We demonstrate that this approach can be extended uniformly to a setting with procedures and parallel threads, which improves on previous related approaches. Even though the research in this øeld is at its very beginning, e.g., the extension to subscripted variables is a matter of future research, this exemplarily highlights an immediate beneøt of applying automata techniques.

Simultaneous Speculation Scheduling

Andreas Unger (with Theo Ungerer and Eberhard Zehendner)

University of Jena, Germany

In this talk, we introduce Simultaneous Speculation Scheduling (S ^{3}) ˘ a combined compiler and architecture technique that enables speculative execution of more than one program path if the program does not contain enough parallelism to utilize the processing potentials of a multithreaded processor.

The S^3 technique can be used to treat control dependences as well as data dependences, in particular loop-carried data dependences. Speculation is controlled by the compiler. Therefore, the compiler inserts instructions for thread-handling into the program, organizes the data exchange among the threads, avoids incorrect calculations by performing a static register renaming, and generates instructions to test for the correctness of the alternative or the parallel execution threads. The generated threads are directly mapped on the hardware threads of a multithreaded processor. The S^3 technique can target any multithreaded architecture that achieves a fast thread interaction. Our methodology aims in particular at simultaneous multithreaded, nanothreaded, and microthreaded processors. We discuss the architectural characteristics of SMT processors that are essential for the S^3 technique, and the instructions that are assumed to be implemented for thread handling. We compare Simultaneous Speculation Scheduling to other static as well as dynamic speculation techniques.

Three Associative Approaches to Automatic Parallelization

Ronald Moore (with Frank Henritzi, Bernd Klauer and Klaus Waldschmidt)

University of Frankfurt, Germany

The essence of the automatic parallelization problem is ønding a mapping of computation and data onto parallel components. This talk reviews three architectures which use associative hardware to support run-time distribution schemes. The associative hardware routes messages between distributed objects, independently of the momentary locations of those objects. This supports dynamic distribution even under volatile run time conditions. The presence of associative routing has important implications for the relationship between compiler and architecture.

The ørst of the three architectures, called ADARC (Associative DataÆow ARChitecture), introduced an Associative Crossbar Network (ACN). Certain limitations inherent in this architecture placed novel requirements on the run-time scheduler. Despite these limitations, empirical results for certain applications were encouraging.

The lessons learned from ADARC have inspired two successor architectures, one focused on the uniprocessor world, and one designed for multiprocessors. The uniprocessor architecture investigates whether ADARC-like associative routing can be used to increase the eŒciency while decreasing the complexity of superscalar architectures. The multiprocessor architecture uses a hybrid approach where static program partitioning is combined with an adaptive run-time distribution protocol. This protocol extends COMAs' (Cache-Only Memory Architectures) ability to distribute data dynamically to allow additionally a similarly transparent, adaptive distribution of computation.

Code Generation for Automatic Parallelization in the Polyhedral Model

Fabien Quiller (with Sanjay Rajopadhye)

IRISA, France

Automatic parallelization in the polyhedral model is based on aŒne transformations from an original computation domain (iteration space) to a target space time domain, often with a diœerent transformation for each variable. Although an often ignored step child of this process, code generation has a signiøcant impact on the quality of the ønal result. It involves a trade-oœ between code size and control simpliøcation/optimization. Previous methods are based on loop splitting, but have non-optimal behaviour in the presence of parametrized programs. Moreover, the only methods to date which handle equalities do so in a very nave manner, and the trade-oœ involves heuristics. We present a general parametrized method for code generation and also discuss the trade-oœ between code size and control overhead. Our method, which is based on dual representations of polyhedra, uses a simple recursion on the dimensions of the domains. It is constructive and, therefore, allows precise trade-oœs.

Optimizing Expression Evaluation for ILP

Fabien Coelho

CRI-ENSMP, France

This presentation focuses on the use of algebraic properties, such as commutativity and associativity, applied to mathematical expressions computed within programs so as to enhance their performance. These transformations can have a signiøcant impact on performance, up to a factor of 3, because the number of operations is changed (e.g., by factorization) and a more balanced expression tree can give more opportunities to use parallel functional units. As the problem is highly combinatorial, and the resolution criterion must be change from one hardware target to another, two parametric greedy heuristics are presented to factorize expressions and to detect special instructions such as combined multiply-add. Then, other opportunities for optimizing code with algebraic transformations, so as to improve invariant code motion or common expression elimination, are presented. In particular, trade-oœ and interaction between these transformations are discussed.

Software Pipelining with Iteration Preselection

David Gregg

Technical University of Vienna, Austria

The software pipelining of loops containing multiple paths is a very diŒcult problem with no general solution. The approach which seems most likely to feasibly achieve a close to optimal schedule is to pipeline by moving operations across the loop entry. Several algorithms use this approach with good results.

The main problem is to decide which operations to move across the loop head and how many times. Existing algorithms use a greedy approach based on

acyclic scheduling. Greedy algorithms may move operations too much creating unnecessary code growth and register pressure, while wasting execution resources.

We break software pipelining into a two-stage process. In the ørst stage, we decide which operations should be moved across the loop entry based on inter- and intra-iteration dependences. In the second stage, we schedule the resulting loop with an acyclic algorithm. Separating the two problems allows us to pipeline less greedily, while using a stronger acyclic algorithm.

3 List of Participants