05101 Abstracts Collection
Scheduling for Parallel Architectures: Theory, Applications, Challenges
— Dagstuhl Seminar —

Erik Altman[1], James Dehnert[2], Christoph W. Kessler[3] and Jens Knoop[4]

[1] IBM TJ Watson Research Center, US
`erik@watson.ibm.com`
[2] Transmeta - Santa Clara, US
`dehnertj@acm.org`
[3] Linköping Univ., SE
`chrke@ida.liu.se`
[4] TU Wien, AT
`knoop@complang.tuwien.ac.at`

Abstract. From 06.03.05 to 11.03.05, the Dagstuhl Seminar 05101 "Scheduling for Parallel Architectures: Theory, Applications, Challenges" was held in the International Conference and Research Center (IBFI), Schloss Dagstuhl. During the seminar, several participants presented their current research, and ongoing work and open problems were discussed. Abstracts of the presentations given during the seminar as well as abstracts of seminar results and ideas are put together in this paper. The first section describes the seminar topics and goals in general.

Keywords. Instruction scheduling, task clustering, task merging, dynamic scheduling, multiprocessor scheduling, software pipelining, hierarchical malleable task-graphs

05101 Executive Summary - Scheduling for Parallel Architectures: Theory, Applications, Challenges

E. Altman, J. Dehnert, C. W. Kessler and J. Knoop

This paper summarizes the objectives and contributions of a seminar with the same title held from March 6 to March 11, 2005 at Schloss Dagstuhl, Germany.

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2005/323

Exploiting Widely Separated Quasi-Independent Strands of Computation in a Single Thread

Erik Altman (IBM TJ Watson Research Center - Yorktown Heights)

Many studies have found large amounts of parallelism are present even in integer apps like SPECint. A machine with unbounded resources and an oracle for branch prediction and memory disambiguation could execute hundreds or thousands of instructions per cycle.

Why have we not already exploited this task level parallelism in a single thread?

- It is widely separated independent tasks/threads of computation, which are often separated by millions of dynamic instructions.
- It is too hard for a compiler to determine all memory aliases.
- It is too hard for a compiler to know the full program call graph, especially with indirect method calls and dynamically linked libraries.
- Millions of instructions is too much for the largest instruction window / reorder buffer.
- To have a 90% chance of following the correct path for a million instructions requires individual branch prediction accuracy better than 99.999947%.

Nevertheless, most long running programs have essentially a "while (TRUE)" main loop in which they repeatedly call tasks. Many of these tasks are independent of each other or are independent of the same task on the previous iteration. For example, gcc compiles one function at a time. Compilation of each function is largely independent. Given a parallel machine (SMT, CMP), a skilled programmer could parallelize gcc to compile functions in parallel. It would be preferable to perform this parallelization automatically for any program, not just gcc. This talk explores some ways that this automatic parallelization might be accomplished.

Keywords: Coarse grain parallelism, assist thread, CMP

Convergent Scheduling

Saman Amarashinghe (Massachusetts Institute of Technology)

Convergent scheduling is a general instruction scheduling framework that simplifies and facilitates the application of a multitude of arbitrary constraints and scheduling heuristics required to schedule instructions for a modern complex processor. A convergent scheduler is composed of independent passes, each implementing a heuristic that addresses a particular problem or constraint. The passes share a simple, common interface that provides spatial and temporal preferences for each instruction. Preferences are not absolute; instead, the interface allows a pass to express the confidence of its preference. By applying a series of passes that address all the relevant constraints, we show that convergent scheduler can produce a schedule better than the state-of-the-art schedulers.

Keywords: Instruction scheduling, clustered VLIW

Task Merging using Graph Rewrite Systems

Peter Aronsson (Linköping University)

The increasing gap between communication cost of data between processing elements and the cost of execution of tasks on such elements requires new methods and cost models for scheduling of task graphs. We present a method of merging tasks in a task graph such that the granularity, i.e. the ratio between execution costs and communication costs, increases. This is achieved by a Graph Rewrite System (GRS) consisting of a set of transformation rules with the goal of merging task into larger tasks. These transformation rules are applied to a task graph given that conditions, ensuring that the parallel time of the task graph is not increasing. Some of the rules use task replication to further reduce the cost of communication. The underlying cost model uses two parameters for the cost of communication, bandwidth and latency. This is important when considering merging of communication packets sent to or from a merged task.

A number of properties of the proposed GRS, such as termination and confluence has been investigated, and the GRS has been implemented in a tool for static scheduling of simulation code.

The task merging GRS has been tested task graphs from the above mentioned tool and from the Standard Task Graph Set, and succeeds in increasing the granularity substantially, mostly depending on the bandwidth and latency parameters.

Keywords: Task clustering, task merging, graph rewrite system, granularity

Recent Developments in Dynamic Scheduling for Load Balancing Scientific Computations

Ioana Banicescu (Mississippi State Univ.)

Traditionally, scheduling to achieve load balancing in large scientific applications employ heuristics that utilize information about processor workloads. In general, such information is collected via profiling, and load balancing is achieved through iterative static partitioning. However, profiling is tedious and incurs high overhead as problem sizes and the number of processors increase. Furthermore, irregularities in the runtime system make the result of profiling work obsolete. Newer scheduling strategies have been proposed to address load imbalances that arise from application and systemic irregularities that are considered known or predictable, and omit those that may occur during execution, such as algorithm adaptivity, unpredictable variations in processor loads, or network latency. These concerns highlight the need for a dynamic scheduling approach that adapts to application-inherent and system-induced irregularities at runtime. This presentation describes such an approach based on recent advances in dynamic loop scheduling: processor workloads are dynamically assigned using scheduling policies based on probabilistic analyses. These policies generate variable sized workloads, such that the workloads have a high probability of being completed before the optimal time. To underscore the effectiveness of the techniques involved, a few sample applications using this approach are presented. The performance obtained by using various dynamic loop scheduling techniques, is up to 74% over straight forward parallelization. Ongoing research collaborative work on a number of NSF research projects, and recent results obtained by integrating these techniques into a few runtime systems will also be discussed.

Keywords: Dynamic scheduling, load balancing, scientific computing

Joint work of: Banicescu, Ioana; Carino, Ricolindo

Optimal Integrated Code Generation for VLIW Architectures

Andrzej Bednarski (Linköping University)

Computer architectures for high-performance embedded systems exhibit an increasing degree of instruction-level parallelism by means of deeper pipelines, VLIW and clustered-VLIW architectures. For such processors, the quality of code relies heavily on the code generator in the compiler's back-end. Code generation includes the three main tasks of instruction selection with resource allocation, instruction scheduling, and register allocation/assignment. Classical code generation techniques solve those tasks separately in consecutive compiler phases, which can cause an unnecessarily low code quality due to strong interdependences between the subproblems solved in the different phases. To our knowledge, there are few attempts to combine the three main code generation phases into a single optimization problem and still produce optimal or highly optimized code.

We present a dynamic programming technique that can solve an integrated code generation problem at the basic block level optimally for small and (depending on the architecture) also medium-sized problem instances within acceptable time and space requirements. For larger problem instances, our dynamic programming algorithms can be relaxed by heuristic pruning techniques and still produce high code quality with shorter optimization times. We have implemented these algorithms in our retargetable optimizing code generator OPTIMIST.

Currently we are investigating extending our algorithms for software pipelining, which is an important optimization for high-performance embedded applications where most of the time is spent in loops.

Keywords: Integrated code generation, dynamic programming, VLIW, clustered VLIW architectures

Preemptive and Nonpreemptive Schedules

Peter Brucker (Universität Osnabrück)

Processor scheduling admits two options to process tasks. In a preemptive mode processing may be interrupted and resumed later possibly on a different machine. In a nonpreemptive mode interruptions are not allowed. Usually, the possibility to preempt tasks leads to better performance values. However, also examples exist where preemptions do not improve the performance. Another issue are complexity questions. Usually,preemptive scheduling problems are easier to solve or have the same complexity as their nonpreemptive counterparts.However there exist examples with the property that the preemptive version is NP-hard but the nonpreemptive version is polynomially solvable.We will discuss these problems in connection with single processor and parallel processor scheduling problems.

Keywords: Preemption, single processor scheduling, multiprocessor scheduling

"Placement of Synchronization Barriers in SPMD Codes" or "Is Theory Really Useful?"

Alain Darte (ENS - Lyon)

In the first part of the talk, I will present a linear-time algorithm that we recently developed for the optimal placement of synchronization barriers in nested loops in SPMD programs, optimal in that it requires the smallest number of barriers. I will also state a related scheduling problem that, to my knowledge, has never been addressed.

In the second part of the talk, I will take this algorithm and several other topics as examples to give my (somehow pessimistic) feeling on the question addressed in this workshop: "how can we bridge the gap between scheduling theory and practice?".

Keywords: SPMD programs, synchronization barriers, theory, practice, NP-completeness, heuristics

Synchronization of Periodic Streams

Christine Eisenbeis (INRIA Futurs - Orsay)

In periodic streaming computations typically used in the domain of high performance video processing, the design of communicating buffers between processes whose clocks do not strictly match is tedious and error-prone. Two communicating periodic processes are defined as n-synchronous if they can be implemented in the ordinary (0-)synchronous model with a buffer of size n. Clock calculus is extended to periodic clocks defined as periodic infinite binary words. The semantics of our programming language is extended accordingly so as to accept these non-strictly synchronous processes. The required size of buffers is computed and the control code of these buffers is automatically generated by applying a systematic program transformation.

Joint work of: Cohen, Albert; Duranton, Marc; Eisenbeis, Christine; Pagetti, Claire; Pouzet, Marc

Optimal Global Instruction Scheduling with Unlimited Resources

Anton Ertl (TU Wien)

We present a method for optimal whole-procedure instruction scheduling for machines with unlimited resources: The program and its dependences are transformed into a linear programming problem, which can then be solved using an off-the-shelf linear problem solver. This scheduler is an intermediate step towards a more realistic global instruction scheduler, but it has also an immediate use: We use it to evaluate the significance of the restrictions imposed by static scheduling and for determining an upper bound for the performance of more realistic global instruction schedulers. We have applied the scheduler to several benchmarks and compared it to a dynamic scheduler with unlimited resources. For some benchmarks, they perform equally well; for others, dynamic scheduling performs much better; on closer inspection it appears that the causes for this performance difference can be reduced by performing well-known transformations before scheduling (in particular, loop transformations).

Keywords: Instruction scheduling, software pipelining, linear programming

Joint work of: Ertl, Anton; Gregg, David; Krall, Andreas

Open Block Scheduling

Aleksei Fishkin (MPI für Informatik)

In the traditional theory of scheduling, each job is processed by only one processor at a time. However, due to the rapid development of parallel computer systems, new theoretical approaches have emerged to model scheduling on parallel architectures. One of these is scheduling multiprocessor jobs where each job consists of several operations which should simultaneously run on parallel processors.

Here we introduce a new multiprocessor job scheduling model, called the *Open Block Problem*, where any two operations of each job may run either non-overlapping in time (as an *open shop* job) or, alternatively, completely simultaneously (as a *block*).

We present first hardness and approximation results.

Keywords: Scheduling, multiprocessor job

Joint work of: Ageev, Alexander; Fishkin, Aleksei; Kononov, Alexander; Sevastianov, Sergey

Scheduling in Spiral

Franz Franchetti (CMU - Pittsburgh)

Digital signal processing (DSP) transforms like the discrete Fourier transform (DFT), discrete Sine and Cosine transforms (DST and DCT) and others are at the heart of many signal processing and scientific computing applications. Spiral targets the problem of portable high-performance implementations for the domain of DSP transforms and translates it into a search problem for the best implementation of a given transform on a given computer system. The generated code is competitive with vendor libraries and the adaptive library FFTW.

In this talk we review Spiral and discuss the various abstraction levels used within the system. By introducing different mathematical and code representations we can apply optimizations at the right level, which means at the level of abstraction where they can be expressed easiest and the application costs least. Many of these optimizations are closely related to scheduling. Spiral is currently supporting scalar and short vector SIMD code as well as experimental code for shared and distributed memory parallel systems.

SPIRAL Team: José M. F. Moura (ECE, CMU), James C. Hoe (ECE, CMU), Markus Püschel (ECE, CMU), Jeremy Johnson (CS, Drexel), David Padua (CS, UIUC), Manuela Veloso (CS, CMU), Bryan W. Singer (CS, CMU), Jianxin Xiong (CS, UIUC), Franz Franchetti (ECE, CMU), Aca Gacic (ECE, CMU), Yevgen Voronenko (ECE, CMU), Kang Chen (CS, Drexel), Robert W. Johnson (Quarry Comp. Inc.), Nick Rizzolo (CS, UIUC)

Keywords: Spiral, digital signal processing, code generator

Partitioning, Mapping, and Overlapped Scheduling of Applications on Network Processors

Ramaswamy Govindarajan (Indian Inst. of Science - Bangalore)

Network processor architectures are designed to handle the inherently parallel nature of network processing applications. Programming these processors to utilize the available resources efficiently, however, remains a challenge. Also, the large variety of processor architectures in use make it difficult to port a solution for one architecture onto another.

It has been shown that efficient partitioning and scheduling, along with data allocation to reduce memory contention is crucial in realizing the performance potentials of a given network processor architecture. This work proposes a framework for automating the task of application mapping and scheduling that can be used in network processors, especially for the IXP family. The framework is extended to perform overlapped execution of multiple instances of concurrent tasks, taking into account the number of threads and the micro-engines in the network processor, to obtain a near-optimal throughput.

Keywords: Scheduling, network processor, design space exploration, genetic algorithm, software pipelined schedule

Stage Scheduling for Dummies

Guillaume Huard (Laboratoire ID-IMAG)

This presentation deals with the stage scheduling problem.

It present a complete overview of the technique:

NP-Completeness of the problem (new result), exact resolution (improved over existing resolutions) and polynomial time approximation (new result).

Keywords: Instruction level scheduling, instruction shifting, stage scheduling

Simulation Modelling and Performance Analysis of Scheduling Strategies in Distributed Systems

Helen Karatza (Aristotle University of Thessaloniki)

Distributed systems pose challenging problems and require proper scheduling in order to efficiently serve users. This presentation summarizes recent research into the performance of scheduling strategies on distributed systems. Topics relating to various aspects of distributed system scheduling are included. The technique used to evaluate the performance of scheduling strategies is experimentation using synthetic workload simulation.

In addition to traditional methods of scheduling, epoch scheduling is also presented.

In epoch scheduling, task priorities in processor queues are recalculated at the end of epochs when queues are rearranged.

Most parallel job scheduling research assumes that parameters such as job inter-arrival times, number of tasks per job and task service demands are defined by specific distributions, but we also consider distributed systems with time varying workloads. We examine scheduling of parallel jobs which consist of independent tasks that can execute at any time on any processor, as well as gang scheduling. Finally, the impact of workload parameters on performance metrics is examined.

Keywords: Simulation, modelling, performance, scheduling, distributed systems

Work in Progress: Load Balancing Strategies for Irregular Parallel Divide-And-Conquer Computations in Group-SPMD Systems

Christoph W. Kessler (Linköping University)

We consider parallel systems with a group-SPMD programming environment, such as MPI, Fork, NestStep, PCP, HPF-2, which provide a fixed number of threads or processors and allow to adapt the scope of synchronization, data sharing and consistency to finer levels of granularity by dynamic group splitting. Parallel divide-and-conquer computations map quite naturally to such systems by dynamically splitting a processor group into subgroups and assigning independent subproblems to these subgroups to solve them recursively. Here, we focus on irregular parallel divide-and-conquer computations such as parallel QuickSort and QuickHull, where the (expected) work to be performed for each subproblem depends on run-time data. While the ideal subgroup sizes can be derived from the expected work of the subproblems, load imbalances can occur because subgroup sizes must be integer and at least 1, and such imbalances can accumulate over the parallel recursion tree.

In this work, we consider several strategies for load balancing of irregular parallel divide-and-conquer computations, including repivoting, serialization of subproblem computations, and flat, fully dynamic load balancing. Based on abstract models (PRAM, BSP) and experimental frameworks (Fork, Tlib/MPI), the goal is to determine which strategies work well under what conditions, and to provide a heuristic combination of strategies that attempts to minimize the (expected) parallel execution time.

Keywords: SPMD, load balancing, irregular parallel divide-and-conquer computations

Joint work of: Eriksson, Mattias; Kessler, Christoph W.; Chalabine, Mikhail

Efficient Instruction Scheduling and Grouping in a JavaVM

Andreas Krall (TU Wien)

In this talk we discuss work in progress about instruction scheduling and grouping in a Java just-in-time compiler for a VLIW architecture.

Scheduling time adds to the run time. Therefore, we evaluated different linear complexity dependence graph construction algorithms. Classical top down list scheduling is applied afterwards. Exceptions and the dynamic optimizations complicate graph construction and scheduling and lead to more restricted dependence graphs.

Keywords: Dependence graph construction, VLIW scheduling

Joint work of: Krall, Andreas; Thalinger, Christian

Lookahead Scheduling for Reconfigurable Data-Parallel Applications

Welf Löwe (University of Växjö)

We propose an approach to continuously schedule data-parallel scientific applications with dynamically changing software architectures. Hereby we combine a dynamic runtime platform and lookahead malleable task graph scheduling.

Keywords: Lookahead malleable task graph scheduling

Joint work of: Andersson, Jesper; Ericsson, Morgan; Löwe, Welf; Zimmermann, Wolf

Generic Software Pipelining at the Assembly Level

Markus Pister (Universität Saarbrücken)

Software used in embedded systems is subject to strict timing and space constraints. The growing software complexity creates an urgent need for fast program execution under the constraint of very limited code size. Techniques for speeding up program execution with only a moderate increase in code size have become very attractive. Typically most of the program execution time is spent in loops; thus enhancing the performance of loops is a key issue for performance improvements.

Even modern compilers produce code whose quality often is far away from the optimum concerning either the execution time or the memory space consumption. One consequence is that the available parallelism in modern processors often is not exhaustively used.

Software pipelining is a static global cyclic instruction scheduling technique which improves the execution times of loops by exploiting instruction level parallelism.

This talk demonstrates software pipelining as a post pass approach at the assembly level within a retargetable framework for post pass analyses and optimizations (PROPAN).

Joint work of: Pister, Markus; Kästner, Daniel

Telescoping Languages: Exploiting Parallelism within User-Defined Abstractions

Daniel J. Quinlan (LLNL - Livermore)

"Telescoping Languages" leverage an existing base language and build compile-time support for user-defined abstractions. This approach is well suited to tailoring a general language to the domain-specific requirements of scientific computing. ROSE is a tool for building domain-specific source-to-source translators on C and C++ applications.

Where domain specific user-defined abstractions are collected within libraries, ROSE permits the definition of telescoping languages with compile-time support for the optimization of the user-defined abstractions. Thus ROSE forms a tool supporting research on telescoping languages and permitting the optimization scientific applications. ROSE permits the construction of domain-specific compile-time optimizations which can leverage any pre-defined semantics.

We expect that ROSE could be used as a tool to support research on the scheduling of parallel operations within scientific applications.

Keywords: Telescoping languages, optimization, scientific computing, object-oriented

Scheduling of M-tasks for Heterogeneous Systems and Grid Environments

Thomas Rauber (Universität Bayreuth)

A task-parallel execution has been shown to be successful on homogeneous parallel systems for many applications providing a suitable degree of multiprocessor task parallelism. We extend the model of task-parallel executions so that the same program can also be executed in heterogeneous systems and grid environments. The new model is particularly suited for large applications consisting of independent modules which can be mapped onto different parts of a distributed execution platform.

We show that a suitable representation of the execution activities is crucial for combining a flexible multi-level specification with a dynamic scheduling that can be adapted to a dynamically changing execution environment.

Keywords: M-task scheduling, heterogeneous systems, grid environments

Joint work of: Rauber, Thomas; Rünger, Gudula

Multiprocessor Task Scheduling and Data Re-Distribution

Gudula Rünger (TU Chemnitz)

Multiprocessor task (M-task) programming is a suitable parallel programming model for coding application problems with an inherent modular structure. An M-task can be executed on a group of processors of arbitrary size, concurrently to other M-tasks of the same application program. The dependencies between the M-tasks lead to a task graph for which an efficient schedule has to be chosen for execution on a specific platform. For the scheduling, the data distribution selected for the M-tasks play an important role. Moreover, data re-distributions between cooperating M-tasks have to be taken into consideration. Based on these observations, we present an extension of the Tlib library for M-task programming by including a data-re-distribution library.

Keywords: Multiprocessor tasks, data re-distribution, scheduling, library approach

Joint work of: Rünger, Gudula; Rauber, Thomas

Characterization of Aliasing Patterns and its Impact on Scheduling for DSP Codes

Markus Schordan (TU Wien)

Digital Signal Processing (DSP) is a domain specific application area in which high performance is a necessity. The use of high-level languages is only accepted if the required performance can be achieved. We analyze representative DSP codes with respect to the use of pointers and aliasing patterns, to select an appropriate alias algorithm which statically computes information that can be utilized by a basic block instruction scheduler.

The more precise the alias information, the smaller is the number of dependences which are conservatively computed by the dependence analysis. This results in a higher degree of freedom of the instruction scheduler and increases the number of possible schedules.

If a better schedule can be found, a higher level of precision of the alias information is useful.

We determine the appropriate alias analysis, ranging from almost linear time to double exponential, based on the aliasing patterns which occur in given DSP programs and investigate the impact of the alias information on a basic block scheduler with several DSP benchmarks.

Keywords: Basic block scheduling, alias analysis, DSP

On Register Requirement in Periodic Schedules

Sid Ahmed Ali Touati (University of Versailles)

In this talk, we present some new fundamental results on register requirement in periodic schedules. The literature is rich in proposing many heuristics for minimizing such factor. However, all the existing techniques use pseudo-polynomial algorithms that compute the periodic register requirement in a software pipelined loop. Indeed, the complexity of the existing algorithms that compute the exact register need of a scheduled loop depends on the initiation interval (II), which is not a polynomial factor since it does not strictly depend on the number of instructions in the loop. Our first contribution in this work is a polynomial algorithm for computing the exact register requirement of a scheduled loop with $O(n \lg n)$ complexity (n is the number of instructions in the loop).

Second, as opposed to what one thinks, we show that it is possible that the optimal minimal register requirement increases if we increment the initiation interval (II). Then, we prove a sufficient condition so that the optimal minimal register requirement decreases (or stays constant) when increasing II.

Third, we prove an interesting property that enables to optimally compute the minimal periodic register sufficiency of a loop for all its valid periodic schedules, independently of II.

Fourth and last, we prove that the problem of optimal stage scheduling under register constraints is a polynomial problem for the case of data dependence graph that fix a unique killer per variable, while such problem has been proved as NP-complete problem for an arbitrary loop. Our latter result is a generalization of the case of trees and forest of trees.

Keywords: Periodic register requirement, periodic register sufficiency, register allocation, software pipelining, stage scheduling, instruction level parallelism, optimizing compilation

Joint work of: Touati, Sid-Ahmed-Ali

About Scheduling Strategies for Distributed Heterogeneous Platforms

Frédéric Vivien (ENS - Lyon)

We consider the problem of scheduling a set of tasks on distributed heterogeneous platform. Because of the characteristics of such a platform, the scheduling practices must change. Therefore, new approaches have been designed, such as steady-state optimization or the divisible load theory. We will briefly present this first technique.

We will also focus on the need to take into account and model the communication networks and the dynamicity of these platforms.

Optimal Global Instruction Scheduling for the Itanium Processor Architecture

Sebastian Winkel (Intel - Santa Clara)

In this talk we present an optimal approach to global instruction scheduling using integer linear programming (ILP). In addition to practical results, we also show new theoretical results on the computational complexity of global instruction scheduling in general. They were obtained through an extensive polyhedral analysis of the global instruction scheduling polytope.

Extended Abstract:

The work targets the statically scheduled Itanium 2 processor. On this highly parallel architecture, effective global instruction scheduling is crucial to high performance. At the same time, it poses a challenge to the compiler: This code generation subtask involves strongly interdependent decisions and complex trade-offs that are difficult to cope with for heuristics.

We tackle this NP-complete problem with integer linear programming (ILP), a search-based method that yields provably optimal results.

This promises faster code as well as insights into the potential of the architecture. Our ILP model comprises global code motion with compensation copies, predication, and Itanium-specific features like control/data speculation.

In integer linear programming, well-structured models are the key to acceptable solution times. The feasible solutions of an ILP are represented by integer points inside a polytope. If all vertices of this polytope are integral, then the ILP can be solved in polynomial time. We define two subproblems of global scheduling in which some constraint classes are omitted and show that the corresponding two subpolytopes of our ILP model are integral and polynomial sized. This means that these subproblems can be solved in polynomial time.

This contrasts with NP-completeness proofs we find for several larger subproblems: Under the assumption P!=NP, these subproblems are not polynomial-time solvable.

The ILP formulation is extended by further transformations like cyclic code motion, which moves instructions upwards out of a loop, circularly in the opposite direction of the loop back edges.

Experiments have been conducted with a post pass tool that implements the ILP scheduler. It parses assembly procedures generated by IntelŠs Itanium compiler and reschedules them as a whole. Using this tool, we optimize a selection of hot functions from the SPECint 2000 benchmark.

The results show a significant speedup over the original code.

Scheduling Hierarchical Malleable Task Graphs

Wolf Zimmermann (Universität Halle-Wittenberg)

We show how task-scheduling techniques can be integrated into compilers for parallel languages. Such an integration allows to compile parallel languages without the need for explicit definition of data distributions and control-flow parallelism. Our approach is robust when libraries are used. The key technique is the use of hierarchically scheduling malleable tasks, i.e., tasks that can be executed on several processors. The talk discusses the relationship between parallel programs and hierarchical malleable task-graphs and discusses a scheduling approach for these task-graphs.

Keywords: hierarchical malleable task-graphs, scheduling, compiler

Joint work of: Zimmermann, Wolf; Trystram, Denis; Loewe, Welf