Carole Dulong, Rajiv Gupta, Robert Kennedy,
Jens Knoop, Jim Pierce (editors):

Code Optimisation – Trends, Challenges, and Perspectives

Dagstuhl-Seminar-Report; 286
17.9.–22.9.2000 (00381)

Dagstuhl Seminar

on

Code Optimisation – Trends, Challenges, and Perspectives

Organized by :

Carole Dulong (Intel Corp., Santa Clara, CA)
Rajiv Gupta (University of Arizona, AZ)
Robert Kennedy (Tensilica, Santa Clara, CA)
Jens Knoop (Universität Dortmund, Germany)
Jim Pierce (Intel Corp., Santa Clara, CA)

Schloss Dagstuhl, September 17 - 22, 2000

Contents

1 Preface

From September 17 to 22, 2000, the Dagstuhl Seminar 00381 on "Code Optimisation: Trends, Challenges, and Perspectives" took place in Schloß Dagstuhl covering research being conducted on a broad range of topics related to code optimization. A major goal of this seminar was to bring together researchers and practitioners from industry and academia working in the field of optimizing compilation, and to create a setting for interaction among the attendees leading to exchange of ideas and future collaborations. This goal was clearly met. Attendees from three continents, 24 from Europe, 5 from Asia, and 11 from North America, 12 of which were from industry and 28 from academia, represented a broad crossection of the community. This was also reflected by the broad range of topics covered at the meeting including:

- Classical code optimizations (PRE, inlining etc.)
- Instruction scheduling and register allocation
- Mobile code - representations and optimizations
- Profiling and profile guided optimization
- Memory optimizations
- Code generation/optimization for DSP/Network/Embedded processors
- Optimizations for Java
- Analysis techniques (data flow, symbolic etc.)

The scientific programme of this seminar consisted of 32 contributed talks, plenary discussion sessions, and informal meetings in the evenings. This report summarizes the abstracts of the presentations, presents the final seminar programme, and reports the statement written by the organizers of the seminar, which was included with the invitations to the participants. Predominating topics in the plenary discussion sessions were the impact of compiler optimization on performance, the relationship between code optimization and code verification, and future challenges for code optimization. The questions below summarize some of the key issues discussed here:

- Compiler Optimizations for Performance:
  - The impact of many optimizations is small.
  - How do the advances in performance from hardware techniques compare with those achieved through compiler techniques?
  - Have we not paid sufficient attention to optimizations with big payoff (e.g., memory optimizations)?
- Code Optimization and Code Verification:
  - Systems being deployed must function correctly or consequences can be disastrous. Do the goals of optimization and verification conflict?

An immediate outcome of this workshop is an *International Workshop on "Compiler Optimization Meets Compiler Verification* (*COCV 2002*)*"* going to be organized as a satellite event of the *5th European Joint Conferences on Theory and Practice of Software* (*ETAPS 2002*), Grenoble, France, April 6 - 14, 2002. This workshop will take place on April 13, 2002, in Grenoble, France. More information on this event can be found on the home page of COCV 2002 at

`http://sunshine.cs.uni-dortmund.de/~knoop/cocv02.html`

Further information including contact information of participants, slides of presentations, as well as brief position statements of future challenges in code optimization can be found at the Seminar's home page:

`http://sunshine.cs.uni-dortmund.de/~knoop/dagstuhl_00381/dag_00381.html`

In closing, we would like to thank Reinhard Wilhelm, the Scientific Director of Schloß Dagstuhl and the staff of the Dagstuhl office in Saarbrücken for the smooth co-operation during the organization of this Seminar. In particular, we thank the staff in Schloß Dagstuhl for making the stay a very pleasant and convenient one for all of us. We gratefully acknowledge the financial support of Intel for sponsoring this seminar. Finally, we would like to thank the participants of the Seminar for a notable donation devoted to helping Schloß Dagstuhl "to fill its empty walls." This donation will be used to purchase the painting "Grüner-Ausschnitt" by Gabriele Eickhoff. We hope that you enjoy viewing this painting the next time you are in Dagstuhl!

August 2001 — Rajiv Gupta
Tucson and Dortmund — Jens Knoop

2 Final Seminar Programme

Monday, September 18, 2000

08:50 Opening
J. Knoop

Session 1 MOTIVATION AND CHALLENGES: **09:00 – 10:30**
Chair: Jens Knoop

09:00 Discussion: Is Code Optimization Relevant?

09:55 Optimization Challenges in Object Code Translation
M. Stadel, Germany

∗ ∗ ∗ BREAK (10:30 – 11:00) ∗ ∗ ∗

Session 2 BACKEND OPTIMIZATION I: **11:00 – 12:10**
Chair: Evelyn Duesterwald

11:00 Generic Postpass Program Optimisation and Analysis Framework
R. Wilhelm, Germany

11:35 Retargetable Code Optimisation by Integer Linear Programming
D. Kästner, Germany

+ + + + + LUNCH (12:15 – 14:00) + + + + +

Session 3 ADVANCED COMPILER ANALYSES AND TRANSFORMATIONS I: **14:00 – 15:45**
Chair: Robert Schreiber

14:00 Optimizations based on Probabilistic Data Flow Systems
E. Mehofer, Austria

14:35 Symbolic Analysis of Programs for Optimizations
B. Scholz, Austria

15:10 Stratification of Rewrite Systems for Optimizing Transformations
U. Aßmann, Germany

∗ ∗ ∗ BREAK (15:45 – 16:30) ∗ ∗ ∗

Session 4 DYNAMIC AND JUST-IN-TIME OPTIMIZATION: **16:30 – 17:40**
Chair: Helmut Seidl

16:30 Enabling High Overhead Analysis in a Dynamic Compiler with
Quasi-Static Compilation
S. P. Midkiff, USA

17:05 LaTTe: An Open-Source Java VM Just-in-Time Compiler
Byung-Sun Yang, Korea

Session 5 Breakout Session I: **20:00 – 21:15**

... on the impact of compiler optimization on performance.

Tuesday, September 19, 2000

Session 6 Classical Analyses and Optimizations Reconsidered: **09:00 – 10:10**

Chair: Rajiv Gupta

09:00 Bidirectional Data Flow Analysis: Myths and Reality
U. Khedker, India

09:35 Dynamic Currency Determination Using Minimal Unrolled Graph
D. M. Dhamdhere, India

∗∗∗ Break (10:10 – 11:00) ∗∗∗

Session 7 Mobile Code: **11:00 – 12:10**

Chair: Samuel P. Midkiff

11:00 Mobile Code Representations Supporting Code Optimization
M. Franz, USA

11:35 Typed Static Single Assignment Form – A Structured and Typed Intermediate Representation for Mobile Code
W. Amme, USA

+++++ Lunch (12:15 – 14:00) +++++

Session 8 Scheduling: **14:00 – 15:45**

Chair: Alan Mycroft

14:00 Global Instruction Scheduling
D. Gregg, Austria

14:35 Instruction Scheduling for Minimum Register Need
Ch. W. Kessler, Germany

15:10 Compiler-Controlled Dual-Path Branch Execution
E. Zehendner, Germany

∗∗∗ Break (15:45 – 16:30) ∗∗∗

Session 9 Optimization under New Perspectives: **16:30 – 17:40**

Chair: Dhananjay M. Dhamdhere

16:30 Code-Size Sensitive Code Motion
O. Rüthing, Germany

17:05 Correctness Issues in Optimizing Transformations
W. Zimmermann, Germany

Session 10 Breakout Session II: 20:00 – 21:15

... on the interdependencies of optimizing and verifying compilation

Wednesday, September 20, 2000

Session 11 Advanced Compiler Analyses and Transformations II: 9:00 – 10:10

Chair: Uwe Aßmann

09:00 Decreasing the Cost of Array Bounds Checks
R. Amir, Israel

09:35 From Recursion to Iteration: What are the Optimizations?
A. Liu, USA

∗∗∗ Break (10:10 – 11:00) ∗∗∗

Session 12 Profiling and Code Selection Techniques: 11:00 – 12:10

Chair: Michael Franz

11:00 Software Profiling for Hot Path Prediction
E. Duesterwald, USA

11:35 Optimization During Tree Parsing Code Selection
M. A. Ertl, Austria

+++++ Lunch (12:30 – 14:00) +++++

∗∗∗ Afternoon Excursion ∗∗∗

Thursday, September 21, 2000

Session 13 Register Optimization: 09:00 – 10:10

Chair: James R. Larus

09:00 Coalescing as an Aid to Interference-Graph Coloring
M. Hailperin, USA

09:35 Dynamic Register Optimization: Analysis vs. Optimization
S. Pande, USA

∗∗∗ Break (10:10 – 11:00) ∗∗∗

Session 14 Backend Optimization II: 11:00 – 12:10

Chair: Carl v. Platen

11:00 A Study of Postpass Code Optimization Techniques for the Infineon 80C166 Microcontroller
Ch. Ferdinand, Germany

11:35 Optimization for Segmented Memory Architectures
F. Martin, Germany

+++++ Lunch (12:30 – 14:00) +++++

Session 15 Hardware-Related Issues I: 14:00 – 15:45

Chair: Santosh Pande

14:00 Cache-Conscious Optimization: Can Compilers Hack It?
J. R. Larus, USA

14:35 PlayDoh, Epic, ... And What's Next?
F. Müller, Germany

15:10 Efficient Use of DSP Addressing Modes
E. Eckstein, Austria

∗∗∗ Break (15:45 – 16:30) ∗∗∗

Session 15 Hardware-Related Issues II: 16:30 – 17:40

Chair: Manfred Stadel

16:30 Optimising Register Placement in Hardware from Flow Graphs
A. Mycroft, UK

17:05 Design and Evaluation of an Induction Pointer Prefetch Algorithm
J. C. Dehnert, USA

Friday, September 22, 2000

Session 16 COMPILER INFRASTRUCTURES: **09:00 – 10:15**
Chair: Jens Knoop

09:00 The SGI Pro64 Compiler Infrastructure
J. C. Dehnert, USA

∗ ∗ ∗ BREAK (10:15 – 11:00) ∗ ∗ ∗

Session 17 BREAKOUT SESSION III : **11:00 – 12:15**

... on challenges and perspectives of optimizing compilation.

+ + + + + LUNCH (12:15 – 14:00) + + + + +

∗ ∗ ∗ FAREWELL ∗ ∗ ∗

3 Abstracts of Presentations

The following abstracts appear in alphabetical order of speakers.

Decreasing the Cost of Array Bounds Checks

Roy Amir
University of Tel-Aviv
Israel

A special architecture mechanism for checking array bounds is described. It is similar to some existing memory-protection mechanisms and modulo addressing computations already in use in commercial machines, and thus can be easily implemented. The main idea of the mechanism is to have a small set of designated registers, each guarding accesses to a single array, and to perform array-bounds checks in parallel with the actual array accesses. We present a simple algorithm for a compiler to use the mechanism to reduce the cost of array-bounds checking in Java. The algorithm is parametric in the number of designated registers. A prototype of the algorithm was implemented. It shows that with 4 registers 61% of the cost of run-time checking is eliminated.

This is joint work with Nurit Dor and Mooly Sagiv.

Typed Static Single Assignment Form – A Structured and Typed Intermediate Representation for Mobile Code

Wolfram Amme
University of California at Irvine, CA
USA

The Java Virtual Machine's byte-code format (JVM-code) has become the de-facto standard for transporting mobile code across the Internet. However, it is generally acknowledged that JVM-code is far from being an ideal mobile code representationa considerable amount of preprocessing is required to convert JVM-code into a representation more amenable to an optimizing compiler, and this pre-processing step translates into precious time lost in a dynamic compilation context. We introduce TSSA, a type-safe mobile code representation based on static single assignment form. We are developing TSSA as a replacement technology to the Java Virtual Machine (JVM), over which it has several advantages: (1) TSSA is better suited as input to an optimizing dynamic code generator and allows CSE to be performed at the code producer's site. (2) TSSA provides incorruptible referential integrity and uses "type separation" to achieve intrinsic type safety. These properties reduce the code verification effort at the code consumer's site considerably. (3) TSSA can transport the results of type and bounds-check elimination in a tamper-proof manner. Despite these advantages, TSSA is usually considerably more compact than JVM-code. The approach taken with TSSA is radically different from JVM's stack-based virtual machine. The TSSA representation is a genuine static single assignment variant in that it differentiates not between variables of the original program, but only between unique values of these variables. TSSA contains no assignments or register moves, but encodes the equivalent information in phi-instructions that model dataflow. Unlike straightforward SSA representations, however, TSSA provides intrinsic and tamper-proof referential integrity as a well-formedness property of the encoding itself. Another key idea of TSSA is “type separation:” values of different types are kept separate in such a manner that even a hand-crafted malicious program cannot undermine type safety and concomitant memory integrity. Interestingly enough, type separation also enables the elimination of type and range checks on the code producer’s side in a manner that cannot be falsified.

This is joint work with Michael Franz and Jeffery Von Ronne.

Stratification of Rewrite Systems for Optimizing Transformations

Uwe Aßmann
Universität Karlsruhe
Germany

Many optimizing transformations can be described by rewrite systems, in particular relational graph rewrite systems. However, most often those systems are indeterministic and the specification developper does not know whether all solutions are apt, adaquate, or correct.

In this talk, we propose a new method to order the rule set of an indeterministic graph rewrite system in layers, so-called strata. When rules are executed along the order of these layers, they produce a unique normal form for the rewrite system. In many cases, this normal form is quite natural.

Applications of the Minimal Unrolled Graph

Dhananjay M. Dhamdhere
Indian Institute of Technology at Powai, Mumbai
India

Compiler optimizations pose many problems to source level debugging due to reordering of computations in a program. One such problem concerns whether a variable has an “expected” value at a breakpoint or at an exception point. The currency determination problem aims at determining whether the value of a variable is current, i.e., whether its actual value is the same as its expected value, at a point in a program.

We have developed a minimal representation of an execution of a program, called a minimal unrolled graph, for the purpose of dynamic currency determination. Adequacy of this representation for dynamic currency determination has been proved using preliminary results from the theory of bit-vector data flow analysis. The size of a minimal unrolled graph is $O(n)$, where n is the number of nodes in a program flow graph.

The minimal unrolled graph can also be used to perform a variant of dynamic slicing of programs.

Software Prefetching of Induction Pointers

James C. Dehnert
Silicon Graphics SGI, Mountain View, CA
USA

We present an automatic approach to prefetching data in linked list data structures. The main idea is based on the observation that linked lists are often allocated at regular intervals in memory, resulting in nearly constant address increments in loops processing the lists. We use this regularity to prefetch linked lists by speculating that future address increments will equal recent ones.

Because this property cannot be guaranteed, it is critical to avoid degradation when it does not hold. We evaluate whether prefetching code can be inserted without degradation by an analysis similar to the minimum iteration interval calculation from modulo scheduling, but generalized to loop bodies with complex control flow, and specifically considering the target machine's limits on outstanding memory fetches. The latter evaluation uses a coloring of a novel memory access interference graph to count likely cache misses along a path.

Our method has been evaluated by an implementation in the SGI MIPSpro compilers, with measurements of benefits on the SPECint2000 benchmark suite running on a MIPS R10000 processor.

This work was done primarily by Artour Stoutchinin, also in collaboration with Guang Gao, Nelson Amaral, Suneel Jain, and Alban Douillet.

The SGI Pro64 Compiler Infrastructure

James C. Dehnert
Silicon Graphics SGI, Mountain View, CA
USA

Pro64 is a suite of compilers and related tools for C, C++, and Fortran95 on IA-64/Linux systems, recently released as open source software by SGI. It is a production commercial compiler, with extensive optimizations in phases performing interprocedural analysis and optimization, loop nest optimization and parallelization, SSA-based global optimization, and code generation including hyperblock formation, software pipelining, integrated global code motion and local scheduling, and global and local register allocation.

In this talk, we present an overview of the compilers' structure, intermediate representations, and optimization capabilities.

(Prepared with Guang Gao, Nelson Amaral, and Ross Towle.)

Software Profiling for Hot Path Prediction

Evelyn Duesterwald
HP Laboratories, Cambridge, MA
USA

Recently, there has been a growing interest in exploiting profile information in adaptive systems such as just-in-time compilers, dynamic optimizers and, binary translators. In this talk, we show that sophisticated software profiling schemes that provide highly accurate information in an offline setting are ill-suited for these dynamic code generation systems. Hot path predictions must be made early in order to control the rising cost of missed opportunity that result from the prediction delay. We will show that existing sophisticated path profiling schemes, if used in an online setting, offer no prediction advantages over simpler schemes that exhibit much lower runtime overheads. Based on these observation we developed a new low-overhead software profiling scheme for hot path prediction. Using an abstract metric we compare our scheme to path profile based prediction and show that our scheme achieves comparable prediction quality. In our second set of experiments we include runtime overhead and evaluate the performance of our scheme in a realistic application: Dynamo, a dynamic optimization system. The results show that our prediction scheme clearly outperforms path profile based prediction and thus confirm that less profiling as exhibited in our scheme will actually lead to more effective hot path prediction.

Efficient Use of DSP Addressing Modes

Erik Eckstein
Atair Software GmbH, Vienna
Austria

The presented algorithm otimizes the use of linear addressing modes, provided by most DSP architectures. The goal is to remove explicit address calculations and to minimize the costs of the addressing modes used. The global heuristic algorithm uses an optimal solution for basic blocks.

Optimization During Tree Parsing Code Selection

M. Anton Ertl
Technische Universität Wien, Vienna
Austria

Tree parsing is well-known as a method for code selection. This talk presents a technique for also using it for some optimizations. The basic principle is to introduce additional nonterminals that correspond to additional data representations. For example, a nonterminal representing complemented values can be used to distribute complement operations to the optimal position (for a machine) using DeMorgan's laws. Other examples are other unary operators, constant folding across intervening non-constants, and optimizing the conversion between different representations, such as various flag representations, tagged and untagged representations, or various data sizes. The advantages of this technique is that it optimizes for the machine, not some intermediate code metric and that it has no compile-time overhead when used with tree parsing automata. The limitations are that there are only a finite number of nonterminals and thus data representations, that optimality is limited to trees, that otherwise it is limited to single-entry regions, and that it results in large grammars. This talk also mentions further work in factoring the grammar and in dealing with DAGs.

Postpass Code Optimization and Code Compaction for the Infineon 80C16x Microcontroller

Christian Ferdinand
AbsInt Angewandte Informatik GmbH, Saarbrücken
Germany

The size of compiled C code is becoming increasingly important in embedded systems, where the economic incentives to reduce ROM sizes are often very compelling. By combining advanced static program analysis methods and pattern matching techniques it is possible to reduce the code size of programs, while preserving the ability to run the program executable directly, i.e. without an intervening decompression stage. The postpass approach allows for a smooth integration of the optimizer into existing development tool chains. Practical experiments on real applications have shown compaction rates of more than 20%.

Mobile Code Representations Supporting Code Optimization

Michael Franz
University of California at Irvine, CA
USA

We are designing a secure and efficient mobile-code transportation scheme that can replace the Java Virtual Machine and is "better" than the JVM in terms of (1) being more scalable to large applications and (2) providing better support for optimizations leading to excellent final code quality.

Rather than using yet another virtual machine, our approach is based on compressing a compiler-related graph-based intermediate representation. Our scheme exploits the many commonalities between security-related information, optimization-enhancing information, and also information useful for data compression, rather than considering them separately.

We use syntax-directed compression as a means of obtaining guaranteed referential integrity, reducing the code verification effort at the target machine. Our format contains compiler-related annotations to obtain top-level performance on the eventual target machine and uses a proof-based approach to guard the compiler-related annotations from falsification in transit.

Two sub-projects are underway, one encoding the semantics of a mobile program at a high level close to the source language, and a second encoding them in a format related to Static Single Assignment form. By implementing both of these options simultaneously, we are exploring the design space rather than designing an ad-hoc solution. The relative trade-offs (encoding density vs. decoding/dynamic compilation speed vs. code quality) are can only be determined by collecting experience with actual prototypes.

Comparing Tail Duplication with Compensation Code in Global Instruction Scheduling

David Gregg
Technische Universität Wien, Vienna
Austria

Global instruction scheduling allows operations to move across basic block boundaries to create tighter schedules. When operations move above control flow joins, some code duplication is generally necessary to preserve semantics. Tail duplication and compensation code are two approaches to duplicating the necessary code, used by Superblock Scheduling and Trace Scheduling respectively. Compensation code needs much more engineering effort to implement, but offers the possibility of less code growth. I implemented both algorithms to see if the extra effort is worthwhile. Initial results suggest that Trace Scheduling does not always create less code growth, and sometimes produces more.

Coalescing as an Aid to Interference-Graph Coloring

Max Hailperin
Gustavus Adolphus College, St. Peter, MN
USA

In graph-coloring register allocators, it is conventional to coalesce a non-interfering pair of vertices when this allows a copy instruction to be eliminated. This can also have an impact on coloring. We find that in practice, the number of colors needed is more often reduced than increased. We have recently proposed coalescing other pairs of vertices as well, solely for the impact on coloring. In the present work, we distinguish between two different ways in which coalescing can aid coloring, and through measurement studies quantify both kinds of benefits, as well as the lesser negative impacts on coloring. We show that the largest effect is from reduction in chromatic number when the graph is rebuilt after coalesces that eliminate copy instructions. A smaller effect comes from coalesces of either kind allowing heuristic coloring to more closely approach the chromatic number (i.e., optimal coloring). In fact, coalescing can make heuristic coloring very nearly optimal for the interference graphs we studied. The only reason why this is a smaller effect than

the chromatic number reduction is that even without coalescing the heuristic colorings are not terribly far from optimal. Our data also show occasional slight increases in chromatic number and worsening of the heuristic coloring. Our experiments use real compiler-generated interference graphs, but ask the question how many colors (registers) are needed to avoid spilling, rather than trying to do actual allocation (with spilling) for a fixed set of registers. Although less realistic for compiler applications, this methodology allows us to avoid the arbitrary discontinuity of the register set size.

This is joint work with Steve Vegdahl of the University of Portland and my undergraduate student John Engebretson.

Retargetable Code Optimisation by Integer Linear Programming

Daniel Kästner

Universität des Saarlandes, Saarbrücken
Germany

In the area of embedded systems stringent timing constraints in connection with severe cost restrictions have led to the development of specialised, irregular hardware architectures designed to efficiently execute typical applications of digital signal processing. The code quality achieved by traditional high-level language compilers for irregular architectures often cannot satisfy the requirements of the target applications. Thus many DSP applications are still developed in assembly language. However due to the increasing software complexity and the shrinking design-cycles of embedded processors there is an urgent demand for code generation techniques that are able to produce high-quality code for irregular architectures. The PROPAN system has been developed as a retargetable framework for high-quality code optimisations and machine-dependent program analyses at postpass, i.e. assembly level. The postpass orientation allows PROPAN to be integrated in existing tool chains with moderate effort. The retargetability concept of PROPAN is based on the combination of generic and generative mechanisms. All relevant information about the target architecture is specified in a dedicated machine description language TDL. From that description a phase-coupled optimiser is generated which can perform global instruction scheduling, register reassignment, and resource allocation by integer linear programming. PROPAN allows to select between an order-indexed and a time-indexed ILP formulation such that the more appropriate modelling can be chosen individually for each target architecture. The generated integer linear programs can be solved either exactly providing a provably optimal solution to the modelled problems, or by the use of ILP-based approximations. The basic idea of the approximative methods is the iterative solution of partial relaxations of the original problem. This way the computation time can be reduced significantly and still a very high solution quality can be obtained. With PROPAN ILP-based postpass optimisers for two widely used contemporary digital signal processors, the Analog Devices ADSP-2106x and the Philips Trimedia TM1000 have been generated. Additionally PROPAN is integrated in a framework for calculating worst-case execution time guarantees for real-time systems where a TDL specification of the Infineon TriCore is used. Finally PROPAN has been successfully used in a commercial postpass optimiser for the Infineon C166 microprocessor.

Instruction Scheduling for Minimum Register Need

Christoph W. Kessler
Universität Trier
Germany

Local instruction scheduling reorders the instructions of a basic block. The goal is to minimize either the space requirements, that is, the number of registers used, or the execution time, that is, the number of CPU cycles used, or some combined optimization criterion. The data dependences among the instructions imply precedence constraints in the form of a directed acyclic graph (DAG) that must be preserved in the schedule to be computed. Except for very special target architectures and certain restricted DAG structures like trees, these optimization problems are NP-complete. We first present an algorithm that computes a space-optimal schedule with worst-case runtime complexity $O(n \cdot 2^n)$ and very good average behavior. The algorithm is based on the enumeration of all alternatives for topological sorting of the DAG. It is made practical by dynamic programming, exploiting domain-specific properties for pruning, structuring the space of partial solutions as a grid, and constructing the partial solutions in increasing order of register need. From experiments with randomly generated DAGs and large DAGs taken from application programs we observe that our algorithm is able to defer the combinatorial explosion and to generate a space-optimal schedule for basic blocks with up to 50 instructions. Note that most basic blocks in real-world programs have less than 50 instructions. Moreover, massive parallelism can be exploited in the algorithm. A straightforward modification of our algorithm to optimize execution time for pipelined and superscalar processors turns out to be less efficient and is practical only for basic blocks with up to 25 instructions, even if the grid structure of the space of partial solutions is extended by an additional third axis for the execution time. In order to avoid this problem, we introduce the new concept of time profiles, which are used to additionally classify subsolutions. This idea allows to apply a similar pruning strategy as for the space optimization. An implementation of this idea does not yet exist, but we expect a considerable improvement in the size of DAGs for which a time-optimal schedule can be computed in practice.

Bidirectional Data Flow Analysis: Myths and Reality

Uday Khedker
University of Pune
India

Research in bidirectional data flow analysis seems to have come to a halt due to an impression that the case for bidirectional data flow analysis has been considerably weakened by a plethora of investigations based on decomposability of known bidirectional placement algorithms into a sequence of purely unidirectional components. This paper shows that the approach of decomposability is not general enough in that it derives its power from the simplifying graph transformation of edge-splitting and the favourable nature of flows in partial redundancy elimination (PRE). This follows from the fact that in the

absence of edge-splitting, PRE cannot be performed using a sequence of cascaded unidirectional flows. Further, edge-splitting inherently converts data flows involved in PRE into unidirectional flows.

In our opinion, this obviates the need of an alternative formulation. We also show that edge-splitting cannot convert data flows involved in “truly” bidirectional data flow problems into unidirectional flows. Thus not every bidirectional data flow problem can be converted into unidirectional flows. Besides, we argue that the premise that bidirectional analysis is more complex than unidirectional analysis, is invalid.

Cache-Conscious Optimization: Can Compilers Hack It?

James R. Larus
Microsoft Research, Redmond, WA
USA

Recent work has demonstrated that cache-conscious data structures and cache-conscious software architecture can improve program performance by large amounts (30-100compiler optimizations, and the disparity is likely to increase with the ever-increasing processor-memory latency gap. This talk surveys some work on cache-conscious optimization and explores the challenges in automating these techniques in compilers.

From Recursion to Iteration: What are the Optimizations?

Yanhong A. Liu
SUNY at Stony Brook, NY
USA

Transforming recursion into iteration eliminates the use of stack frames during program execution. It has been studied extensively. This paper describes a powerful and systematic method, based on incrementalization, for transforming general recursion into iteration: identify an input increment, derive an incremental version under the input increment, and form an iterative computation using the incremental version. Exploiting incrementalization yields iterative computation in a uniform way and also allows additional optimizations to be explored cleanly and applied systematically, in most cases yielding iterative programs that use constant additional space, reducing additional space usage asymptotically, and run much faster. We summarize major optimizations, complexity improvements, and performance measurements.

This is joint work with Scott D. Stoller.

Optimization for Segmented Memory Architectures

Florian Martin
Universität des Saarlandes, Saarbrücken
Germany

In this talk a bunch of analyses are proposed to to optimize the cost of memory accesses on segmented memory machines in terms of code size.

This problem of reducing code size for relatively old processors is still highly relevant in the embedded market.

On processors with segmented memory architectures, like the Infineon C166, the memory access for data has to be made in two steps. First a segment register has to be loaded and the data can be accessed in a second instruction via an offset and a segment register.

We try to minimize this overhead by carefully selecting places and values to insert loads of the segment registers.

The improvements reached by these techniques on real life test programs are up to 7%.

Optimizations Based on Probabilistic Data Flow Systems

Eduard Mehofer
Universität Wien, Vienna
Austria

Probabilistic program optimization consists of probabilistic data flow analysis followed by a transformation which takes probabilities of data flow facts into account. In the following we will address both issues.

Classical data flow analysis is done statically without utilizing runtime information. All paths are equally weighted irrespectively whether they are heavily, rarely, or never executed. In contrast probabilistic data flow analysis takes runtime information into account by using edge probabilities to distinguish between frequently and rarely executed branches. The resulting solution gives us the probabilities with what data flow facts may hold true during execution at some program point.

We present a novel probabilistic data flow framework which takes execution history into account while propagating probabilities through the control flow graph. Practical experiments with spec95 show that in this way significantly better results can be achieved.

On the transformational side we present two applications: Probabilistic communication optimizations and parallelization for distributed-memory systems, and probabilistic procedure cloning for high-performance systems.

Enabling High Overhead Analysis in a Dynamic Compiler with Quasi-Static Compilation

Samuel P. Midkiff
IBM T. J. Watson Research Center
Yorktown Heights, NY
USA

The optimizations that can be performed by dynamic or just-in-time compilers are severely constrained by the need to limit the time to perform the optimizations. Global, (whole function) and interprocedural optimizations are limited to programs whose execution time is long enough to amortize the high cost of performing the optimizations. In Java, pure static compilation cannot be used without violating language semantics related to dynamic class loading. In this talk we propose a quasi-static compilation model. Quasi-static compilation optimistically statically compiles Java classes, and during the compilation retains information (dependences) about other classes on which the correctness of the compilation depends. When methods in the class are used in another execution, the precompiled code is accessed, specialized for the current instance of the Java Virtual Machine, and, if the dependences are fulfilled, executed. If the dependences are not fulfilled, the method is dynamically compiled. By performing quasi-static compilation, we achieve performance gains of from 9 to 90 percent on the specJVM98 size 100 benchmarks, and from 55 to 365 percent on the size 10 benchmarks. With the more aggressive optimization and analysis techniques enabled by quasi-static compilation, these performance gains will increase.

PlayDoh, Epic, ... And What's Next?

Frank Müller
Humboldt-Universität zu Berlin
Germany

The contributions of this talk are twofold. First, past proposals for architectural changes are briefly highlighted and compared with current trends in emerging processors. At the same time, the impact of explicitly parallel instruction computing on code optimization is studied, in particular with respect to prospective trends for the memory hierarchy. Second, aspects of exploiting instruction parallelism for VLIWs by trace scheduling in static compilation and dynamic translation environments are studies. The impact of code reordering and code replication on the control flow is discussed with an emphasis on techniques to handle irreducible regions of control flow.

Register Placement in the SAFL-to-Hardware Compiler

Alan Mycroft
Cambridge University
UK

We introduce the language SAFL which is used as the hardware specification language in the FLaSH (Functional Languages for Synthesising Hardware) system. SAFL is a functional languages (hence easy to transform) which is (i) parallel and (ii) statically allocated; we argue this matches hardware. The SAFL *ccompiler* is *resource aware* (compiles function definitions to hardware blocks 1–1) and is preceded by a *transformer* to choose the Area-Time division. We show how it compiles functions as a input register and logic (combinatorial if no function calls). Intermediate values need to be saved in a register if there is (parallel or sequential) *conflict* for the value of a called function. Variants on this *callee-save* mechanism are described.

See `http://www.cl.cam.ac.uk/users/am/papers` dated 2000/2001.

This is joint work with Richard Sharp.

A Fast Dynamic Register Allocator with Superior Code Quality for Small Register Set Embedded Processors

Santosh Pande
Georgia Tech, Atlanta, GA
USA

We describe a usage density based register allocator geared towards dynamic compilation systems. The main attraction of the allocator is that it does not make use of the traditional live range and interval analysis nor performs advanced optimizations based on range splitting or spilling but results in very good code quality. We circumvent the need for traditional analysis by using a measure of usage density of a variable. The usage density of a variable at a program point represents both the frequency and the density of the uses. We contend that using this measure we can capture both range and frequency information which is essentially used by the good allocators based on splitting and spilling. We describe a two-pass framework based on this measure which has a linear complexity in terms of the program size. We perform comparisons with static allocators based on graph coloring and dynamic ones based on simple scan (linear scan) of live ranges and show that our allocator maintains the speed of dynamic allocators and improves the quality of generated code.

This is joint work with Sathyanarayanan Thammanur.

Is Code Optimization (Research) Relevant?

Bill Pugh
University of Maryland, College Park, MD
USA

This originally scheduled talk had to be cancelled on short notice because of a sudden affection of the speaker preventing him from attending the seminar. The slides of the talk are available at Bill Pugh's home page: ***`http://www.cs.umd.edu/~pugh/`***

Code-Size Sensitive Code Motion

Oliver Rüthing
Universität Dortmund
Germany

Program optimization focuses usually on improving the run-time efficiency of a program. Its impact on the code size is typically not considered a concern. In fact, classical optimizations often cause code replication without providing any means allowing a user to control this. This limits their adequacy for applications, where code size is critical, too, like embedded systems or smart cards. In this talk, we demonstrate this by means of partial redundancy elimination (PRE), one of the most powerful und widespread optimizations in contemporay compilers, which intuitively aims at avoiding multiple computations of a value at run-time. By modularly extending the classical PRE-approaches we develop a family of code-size sensitive PRE-transformations, whose members in addition to the two traditional goals of PRE (1) reducing the number of computations and (2) avoiding unnecessary register pressure, are unique for taking also (3) code size as a third optimization goal into account. Each of them optimally captures a predefined choice of priority between these three goals. The flexibility and aptitude of these techniques for size-critical applications is demonstrated by various examples.

This is joint work with Jens Knoop and Bernhard Steffen, University of Dortmund.

Symbolic Analysis for the VFC Compiler

Bernhard Scholz

Technische Universität Wien, Vienna
Austria

The quality of many optimizations and analyses for parallelizing compilers significantly depends on the ability to evaluate symbolic expressions and on the amount of information available about program variables at arbitrary program points. We describe an effective and unified symbolic evaluation framework that statically determines the values of variables and symbolic expressions, assumptions about and constraints between variable values and the condition under which control flow reaches a program statement. The framework computes program contexts at arbitrary program points, which are a novel representation for comprehensive and compact control and data flow analysis information. All of our techniques target both linear and non-linear expressions and constraints. The efficiency of symbolic analysis is highly improved by aggressive simplification techniques. To illustrate the effectiveness of our approach we present an example for communication vectorization.

Optimization Challenges in Object Code Translation

Manfred Stadel

Fujitsu Siemens AG, München
Germany

When introducing new processor architectures you need to provide migration support to make existing applications executable on the new processors. Emulation as well as various kinds Object Code Translation (OCT) are appropriate means therefor.

The common goal of OCT is to translate code written for some original processor architecture to object code for a different target processor architecture.

Assembly language translators (AssTran) accept the original object code in assembly language notation. Static Object Code Translators (SOCT) accepts original object module files as input. AssTran and SOCT generate target object module files. Dynamic Object Code Translators (DOCT) have the loaded image of original binary code as input and write generated target binary code in memory for immediate execution. DOCTs are used to accelerate emulation: Emulation starts with interpretation. A piece of code which has been interpreted a certain number of times is translated.

All kinds of OCTs have less information than higher level language compilers:

- There is no sophisticated type system,
- at least DOCTs cannot clearly separate text and data segments,
- memory attributes like "volatile" and "constant" are not available,
- most variables are addressed indirectly,
- control flow is often not entirely known.

Classical optimization algorithms are not sufficient and must at least be enhanced and adapted to the situation of OCTs. Additional optimization techniques like speculation and specialization are required.

Generic Program Optimisation and Analysis Framework

Reinhard Wilhelm
Universität des Saarlandes, Saarbrücken
Germany

We present a framework that analyses and transforms programs on the executable/assembler level. Emphasis is put on the generic and generative approaches employed in the framework to cope with the retargetability problem.

A brief overview is given of the application to the determination of worst case execution times (WCET). WCET determination needs a static prediction of the cache and the pipline behaviour of the program and the identification of the worst case execution path. The first two problems are solved using abstract interpretation, the latter using integer linear programming (ILP).

LaTTe: An Open-Source Java VM Just-in-Time Compiler

Byung-Sun Yang
Seoul National University
Korea

Java grows a prominent programming language with a wide application spectrum from embedded systems to enterprise servers. One of the major issues in using Java is performance, specially of the Java Virtual Machine (JVM). As well as optimizing the various JVM runtime components such as garbage collection, exception handling, and threads, Just-in-Time (JIT) compilation, an example of dynamic compilations, is a must to reduce overheads of the machine-independent bytecode execution.

This talk introduces LaTTe, an open-source JVM with a highly-engineered JIT compiler for RISC machines. LaTTe first translates the bytecode into pseudo RISC code with symbolic registers, which is then optimized and register allocated while coalescing copies corresponding to pushes and pops between local variables and the operand stack. Runtime components of LaTTe are also optimized in close coordination with the JIT compilation.

Experimental results with various benchmarks including SPECjvm98 and Java Grande reveals that LaTTe achieves performance better than or comparable to the latest SUN JVMs (JDK 1.1.8, JDK J2SE 1.3 HotSpot, and JDK 1.2 production release).

Compiler-Controlled Dual-Path Branch Execution

Eberhard Zehendner

Friedrich-Schiller-Universität Jena
Germany

Unpredictable branches in machine programs hinder static scheduling and introduce misprediction penalties. Multithreading extensions to standard wide-issue superscalar processors, that are assumed to appear as part of those architectures in the future, can be used to avoid these problems. A conditional branch instruction is replaced by a fork instruction spawning a new thread starting at the branch target. Synchronisation instructions at the beginning of both branch paths check for the branch predicate and cancel the wrongly executing thread.

Following some rules similar to those of ordinary global instruction scheduling, instructions may be moved to any speculative section between a fork instruction and a corresponding synchronisation instruction. These rules are designed such that the results of instructions in speculative sections can be safely commited, whereas instructions behind a synchronisation instruction may be executed but commit only if on the correct path and after the speculation is resolved.

Our method – called Simultaneous Speculation Scheduling – completely removes misprediction penalties for the replaced branches. Due to the additional freedom w.r.t. global instruction scheduling, critical paths can be further optimised. We get some advantages over pure software speculation techniques as well as over predicated execution since the first thread finishing its synchronisation instruction stops fetching and executing instructions from the wrongly executed thread (may be itself).

Correctness Issues in Optimizing Transformations

Wolf Zimmermann
Universität Karlsruhe
Germany

In many articles, it is said that optimizing transformations should preserve the semantics of programs. However, the notion of semantics preservation remains undefined. The talk shows that is far from being trivial to define an adequate notion of "semantics preservation". The notion of "semantics preservation" should be based on a notion of compiler correctness: The users (or environment) can only observe I/O-operations that are performed by a program. A compiler is correct iff the target program preserves this observable behaviour of the source program. Since in higher level programming languages resources are usually unbounded while they are bounded on target processors, it may be allowed that the target program may abort because resource bounds are exceeded. Based on this notion of correctness - stemming from discussions in the DFG-project "Verifix" - the talk discusses correctness of code motion, dead code elimination, and procedure inlining. The first two demonstrate that analysis of exception behaviour is necessary to perform these optimizations. The procedure inlining example demonstrates that interaction of privacy concepts and inheritance can destroy correctness of procedure inlinling.

A Appendix

In this section we document the motivation for the seminar.

Motivation of the Seminar

The last decades have been witness to continuous, rapid, and far reaching progress in code optimisation. Currently, optimisation faces new challenges caused by the increasing importance of advanced programming paradigms like the object-oriented, (data-) parallel and distributed ones, the emerging dissemination of new innovative processor architectures, and the explosive proliferation of new application scenarios like Web-computing in the retinue of the meanwhile ubiquitious Internet.

New paradigms, new architectures, and new application scenarios apparently demand new compilation and optimisation techniques, but also offer new potentials for optimisation on both machine-dependent and machine-independent levels.

In the light of this situation the aim of the seminar is to bring together researchers and practitioners from both industry and academia working on any phase of optimising compilation to exchange views, share experiences, identify potentials and current and future challenges, and thus, to bridge gaps and stimulate synergies between theory and practice and between diverse areas of optimisation such as machine-dependent and machine-independent ones.

Central issues which should be discussed in the seminar are:

- *Paradigm and software/hardware boundaries*:
  Do we require new techniques to reasonably accommodate the specialities of new paradigms, architectures, or Web-driven application scenarios? Will unifying approaches that transcend paradigms be superior or even indispensible because of the economic demands for reusability, portability, and automatic generability? Similarly, the boundaries between hardware and software optimisations are changing and redefined as *e.g.*, by IA64. Does the boundary lie in the right place? What are the missing architecture hooks for the compiler to really be as good as the hardware? Is that even possible?

- *Optimisation of running time vs. memory use*:
  Will there be a renascence of storage-saving optimisations and a shift away from emphasis on running time due to the growing importance of embedded systems and the distribution of executables across the Internet?

- *Static vs. dynamic and profile-guided optimisation*:
  Very wide architectures are very sensitive to profile-guided optimisations. The profile data set, however, is most commonly not known at compile time. What are practical ways of gathering profile information without slowing down applications, and practical ways of using this information for dynamic optimisations? Concerning Internet-based applications, must approaches for just-in-time computation be complemented by approaches for just-in-time optimisation? What are the key issues here?

- *Formal methods*:
  What is the role of formal methods in code optimisation regarding the requirements

of reliability, validation or even verifiability of correctness and (formal) optimality of an optimisation? What should be its role? How can the benefits possibly offered by formal methods best be combined with those of empirical evaluations?

- *Mastering complexity*:
  The increased complexity of compiler optimisation can lead to validation nightmares and can increased compiler team size to a counterproductive level. This phenomenon proves to be a key problem in practice. How can it be mastered? In particular, how do we avoid the problem faced by growing software and hardware teams? Can formal methods improve on this situation? What could be their impact?

- *Experimental evaluations*:
  Do we need a common, publicly available compiler testbed for experimental evaluations and comparisons of competing approaches? What would be the key requirements? Is it indispensable for reasonably pushing synergies between theory and practice?

- *Synergies*:
  What and how can people from different communities working on code optimisation, e.g. on a machine-dependent and machine-independent level, learn from each other?

At the threshold of a new millenium, and in face of the rapid change of paradigms on both the software and hardware sides, it seems to be worthwile to take stock of the state of the art, to reflect on recent trends, and to identify current and future challenges and perspectives on code optimisation.

We believe that a Dagstuhl Seminar will provide an ideal setting for this endeavor, and will be a stimulating venue for people from all communities working seriously, but often with (too) little contact on these issues, to come together and exchange views and ideas, and to share their different experiences in order to push synergies and further progress in the field.

April 1999
Schloß Dagstuhl, Wadern

Carole Dulong
Rajiv Gupta
Robert Kennedy
Jens Knoop