07361 Abstracts Collection
Programming Models for Ubiquitous Parallelism
— Dagstuhl Seminar —

Albert Cohen[1], María J. Garzaran[2], Christian Lengauer[3], Samuel P. Midkiff[4] and David Chi-Leung Wong[5]

[1] INRIA Futurs - Orsay, FR
Albert.Cohen@inria.fr
[2] Univ. of Illinois - Urbana, US
garzaran@cs.uiuc.edu
[3] Univ. Passau, DE
lengauer@fim.uni-passau.de
[4] Purdue Univ., US
smidkiff@purdue.edu
[5] Intel Corp., US

Abstract. From 02.09. to 07.09.2007, the Dagstuhl Seminar 07361 "Programming Models for Ubiquitous Parallelism" 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. Links to extended abstracts or full papers are provided, if available.

Keywords. Parallel programming models, transactional memory, languages, compilers, optimizations, architecture, automatic parallelization

07361 Introduction – Programming Models for Ubiquitous Parallelism

The goal of the seminar is to present a broad view of the research challenges and ongoing efforts to improve productivity, scalability, efficiency and reliability of general-purpose and embedded parallel programming.

Keywords: Parallel programming models, transactional memory, languages, compilers, optimizations, architecture, automatic parallelization

Joint work of: Cohen, Albert; Garzarán, María J.; Lengauer, Christian; Midkiff, Samuel P.; Wong, David Chi-Leung

Extended Abstract: http://drops.dagstuhl.de/opus/volltexte/2008/1373

High productivity at what price? a look at the IBM UPC implementation

Gheorghe Almasi (IBM TJ Watson Research Center, USA)

Large scale machines are becoming cheaper and faster, but programming these machines has not become easier. In this talk we examine the promise of the UPC (Unified Parallel C) language to combine productivity with high performance: what the shortcomings of the language are and how we are addressing these to allow UPC to scale to large machines.

Keywords: Performance, scalability, productivity

Joint work of: Almasi, Gheorghe; Cascaval, Calin; Nishtala, Rajesh

The Berkeley View 2.0

Krste Asanovic (Univ. California - Berkeley, USA)

Following on from the Berkeley View survey, the Par Lab team at UCB has been thinking about how to develop a programming system that will make it easy to write correct programs that run efficiently on manycore systems. Our approach is based on two layers: a productivity layer, which sidesteps most concurrency issues, to be used by the majority of programmers; and an efficiency layer, which can achieve close to "bare metal" performance, for use by expert programmers.

Other important ingredients include a composition and coordination language, used to safely compose parallel libraries and frameworks, and a lightweight hypervisor layer that provides protected partitions within which user code controls all scheduling decisions.

High Performance Library Generation with Hierarchical Decomposition

Denis Barthou (University of Versailles, F)

Performance libraries are the building blocks of high performance applications. These libraries, architecture-dependent, are mostly hand-tuned. On-going research efforts have led to the design of automatic library generators such as ATLAS for linear algebra functions, SPIRAL or FFTW for signal processing and DFT functions. These generators, domain-specific, automatically tune the code according to the target architecture features.

In this talk I will present a new approach for the generation of performance libraries. This approach is not application-specific: it relies on performance measures of source code kernels and on a very simple performance model in order to build library functions from these building blocks. Performance results will be

compared with ATLAS and vendor libraries on Itanium2 and Pentium architectures. Challenges of this approach, in particular of the performance model, will be discussed.

An Intermediate Language for Productive and Efficient Parallel Programming: A Stream-Computing Experiment

Albert Cohen (INRIA Futurs - Orsay, F)

Moore's law on semiconductors is coming to an end. Scaling the von Neumann architecture over the 40 years of the microprocessor has led to unsustainable circuit complexity, very low compute-density, and high power consumption. On the other hand, parallel computing practices are nowhere close to the portability, accessibility, productivity and reliability levels of single-threaded software engineering. In terms of productivity, the dominant parallel programming models are often worse level than sequential programming ... in assembler. Moreover, these models hide the key decisions about scheduling and resource management on a variety of distributed, multi-level and heterogeneous parallel architectures. This lack of expressiveness may be desirable for a high-level language, but inside a compiler and at the interface with the run-time system and architecture, it leads to unnecessary overheads, lack of scalability and performance portability.

In addition, designers of physically constrained embedded systems are used to attaching temporal and resource requirements to the functional semantics of programs, a requirement often contradicting the modularity and abstraction expectations of modern programming models.

This talk advocates for an intermediate-language-driven approach to these challenges. We report on work in progress, building on recent advances of polyhedral compilation (affine scheduling and partitioning) on one side, and on new insights from the synchronous data-flow paradigm. We will highlight the main features of a stream-oriented intermediate language to bridge the productivity and efficiency gaps while offering higher levels of control to the compiler and to the expert (library) programmer. This intermediate language is based on a data-flow, stream-oriented generalization of the classical (scalar) SSA form, and also share some commonalities with transactional memory.

Keywords: Data-flow, primitive constructs, intermediate language, efficiency, optimization

Predictable performances for embedded systems

Marc Duranton (NXP Semiconductors - Eindhoven, NL)

Programmable embedded systems are ubiquitous nowadays, and their number will further increase with the emergence of Ambient Intelligence.

One of the first challenges for embedded systems is mastering the increasing complexity of future Systems on Chip (SoC). The complexity will increase relentlessly because the applications will become more and more demanding with the algorithmic complexity growing exponentially over time. Performances can only be reached by using all forms of parallelism. This will bring the challenges of designing multi-core systems using all possible levels of parallelism to reach the required performance density, of extracting all the parallelism from the application(s) and of efficiently mapping this efficiency to the hardware.

But for most embedded systems, a main challenge is in having sustained performances, not peak: guaranteed performances, and predictable timing behavior are important, together with Quality of Service, safety, reliability and dependability. The notion of time is key in embedded systems, and most of the current methods and tools, inherited from mainstream computer science, did not really cope with this extra requirement.

The new technology nodes (65nm, soon 45nm) will also bring their own challenges: the global interconnect delay does not scale with logic, the leakage power will be more and more important, and the increasing variability of components will be major problems for the design of complex systems.

Assembling systems with *unpredictable* elements will increase the global system unpredictability. Also, systems are not really designed with *separation of concern*in mind, and due to shared resources, a slight change can have a drastic impact.

Major breakthroughs will be required in compiler technology and in mapping tools, both in term of correctness and in terms of performance to achieve an efficient use of resources (performance- and power-wise), but also for the debug, validation and test of the system. Systems can no longer be verified with simulations, and we will need new validation approaches. Otherwise, the unpredictability and unreliability due to the combination of use cases will make the systems practically unusable.

Keywords: Predictability, complexity, nanophysic, concurrence, parallelism

Design Issues in Parallel Array Languages for Shared Memory

Basilio B. Fraguela (Universidad da Coruna, E)

In this talk I shall give a brief introduction to the Hierarchically Tiled Array (HTA), a data type on which researchers from UIUC, IBM and UDC (Spain), have been working for some years. HTAs enable the specification of locality and parallelism in object-oriented languages in a straightforward way by means of operations on tiles.

We have found that codes written with HTAs are very readable, and their performance is similar to that of other approaches to express parallelism. Implementations in MATLAB and C++ have been developed, with the focus to date

being on distributed memory systems. In the second part of the talk I shall discuss the design options, challenges and opportunities we face when considering a shared memory implementation of HTAs .

Keywords: Array languages, parallelism, shared memory

Parallelism in Spiral

Franz Franchetti (CMU - Pittsburgh, USA)

Spiral (www.spiral.net) is a program generation system for linear transforms such as the discrete Fourier transform, discrete cosine transforms, filters, and others. For a user-selected transform, Spiral autonomously generates different algorithms, represented in a declarative form as mathematical formulas, and their implementations to find the best match to the given target platform. Besides the search, Spiral performs deterministic optimizations on the formula level effectively restructuring the code in ways impractical at the code level.

In this talk we give a short overview on Spiral and then explain how Spiral generates efficient programs for parallel platforms including vector architectures, shared and distributed memory platforms, the Cell processor, and GPUs.

Library Generators for Parallel Machines

Maria J. Garzaran (Univ. of Illinois - Urbana, USA)

The growing complexity of processors has made the generation of efficient code increasingly difficult. As a result, hand-tuned code can be orders of magnitude faster than compiled code.

To address this problem library generators such as ATLAS, FFTW, or SPIRAL use empirical search to find the parameter values such as tile size or degree of unroll that deliver the best performance for a particular machine.

In this talk, I will present our experience in the generation of an adaptive sorting library for sequential and parallel machines, and discuss the issues that appear when applying this technology for the automatic generation of libraries for parallel machines.

Some Experiments on Tiling Loop Programs for Shared-Memory Multicore Architectures

Armin Größlinger (Universität Passau, D)

The model-based transformation of loop programs is a way of detecting fine-grained parallelism in sequential programs.

One of the challenges is to agglomerate the parallelism to a coarser grain, in order to map the operations of the program to the available cores in a multicore architecture. We consider shared-memory multicores as target architecture for space-time mapped loop programs and make some observations concerning code generation, load balancing and cache effects.

Keywords: Multicore, automatic parallelization, loop transformations, polyhedron model

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2008/1374

A Unified Retargetable Design Methodology for Dedicated and Re-Programmable Multiprocessor Arrays - Case Study and Quantitative Evaluation

Frank Hannig (Universität Erlangen-Nürnberg, D)

The efficient mapping of algorithms onto parallel architectures is of utmost importance since many state-of-the-art embedded digital systems have to deploy parallelism in order to increase their computational power.

This talk deals with the mapping of nested loop programs onto massively parallel processor arrays. We present a unified design methodology in order to achieve highly parallel implementations for two kinds of architectures: (a) dedicated, application-specific arrays and (b) coarse-grained, "weakly programmable" processor arrays. We describe which steps of the design flow can be conducted for both architecture types in common.

The hardware synthesis of dedicated hardware accelerators is mostly automated and only relatively few architectural constraints have to be considered.

Whereas, when targeting coarse-grained processor arrays, a large number of architectural parameters have to be incorporated during the backend code generation.

The proposed unified retargetable design methodology is applied in several case studies. Implementations for both target architectures with respect to performance, area cost, and reconfiguration time are evaluated. The results show that both approaches have their specific benefits and drawbacks.

Keywords: Embedded systems, Parallel architectures, Multiprocessor, Compilation, Domain-specific computing

Joint work of: Hannig, Frank; Ruckdeschel, Holger; Dutta, Hritam; Kissler, Dmitrij; Stravert, Andrej; Teich, Jürgen

It's not just the CPU - managing resources matters more than ever

Tim Harris (Microsoft Research UK - Cambridge, GB)

I would like to give a short talk about some systems issues that are often overlooked in discussions about using multi-core processors.

When thinking about desktop workloads machines vary massively in the resources devoted to a given application, either because of hardware differences, or because many applications and services are running concurrently on the same machine.

The challenge is not just designing programming models that enable applications to run effectively across different levels of hardware parallelism, but also arbitrating the conflicting requirements of concurrent applications – e.g. I do not want my spare CPU time used to run a disk indexing workload when the disk is already saturated. I would rather that some cores be left idle if that processor's memory bus is already saturated.

If two non-time-critical background tasks can each fill all of memory then let us run them in series rather than parallel.

What can we do to deal with these kinds of resource management problems?

A Case for Deconstructing Hardware Transactional Memory Systems

Mark Hill (University of Wisconsin - Madison, USA)

Major hardware and software vendors are curious about transactional memory (TM), but are understandably cautious about committing to hardware changes.

Our thesis is that deconstructing transactional memory into separate, interchangeable components facilitates TM adoption in two ways. First, it aids hardware TM refinement, allowing vendors to adopt TM earlier, knowing that they can more easily refine aspects later.

Second, it enables the components to be applied to other uses, including reliability, security, performance, and correctness, providing value even if TM is not widely used. We develop some evidence for our thesis via experience with LogTM variants and preliminary case studies of scalable watchpoints and race recording for deterministic replay.

Keywords: Hardware transactional memory

Joint work of: Hill, Mark D.; Hower, Derek; Moore, Kevin E.; Swift, Michael M.; Volos, Haris; Wood, David A.

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2008/1375

Full Paper:
http://www.cs.wisc.edu/multifacet/papers/tr1594_dtm.pdf

A model for the design and programming of multi-cores

Chris Jesshope (University of Amsterdam, NL)

This talk will describe a machine/programming model for the era of multi-core chips. It is derived from the sequential model but replaces sequential composition with concurrent composition at all levels in the program except at the level where the compiler is able to make deterministic decisions on scheduling instructions. These residual sequences of instructions are called microthreads and they are small code fragments that have blocking semantics. Dependencies that would normally be captured by sequential programming are captured in this model using dataflow synchronisation on variables in the contexts of these microthreads. The resulting model provides a foundation for significant advances in computer architecture as well as operating systems and compiler development. I will take a high-level perspective on the field of asynchronous distributed systems and draw conclusions that indicate dynamic and concurrent models are the only viable solution for this era but that these should not necessarily be visible to the users of the system.

Keywords: Multi-cores, many-cores, programming model, resource management

Active Library Infrastructure for Multicore Performance

Paul H. J. Kelly (Imperial College London, GB)

Active libraries are libraries that play an active part in the compilation, and especially the optimization, of their client code.

This talk motivates active libraries with a couple of examples we have built, for linear algebra and for visual effects for the movie industry. The main objective is to press the argument for active libraries as the delivery medium for performance optimisation technologies, and to map out some of the issues. As well as concrete examples, I will explore some of the issues in developing tools and interoperable infrastructure.

Keywords: Active Libraries, Compilers, Parallel Processing, Locality

Ubiquitous HPC Computing by Means of Reparallelizable and Migratable OpenMP Applications

Michael Klemm (Universität Erlangen, D)

Researchers often can access various computing resources through Computational Grids. However, to actually use these resources is far from user-friendly because of different architectures, network interconnects, memory bandwidths, etc.

Users are also faced with the systems' schedulers that assign CPUs/nodes to jobs. Schedulers ask the user to provide an estimate of what amount of resources their application will demand at runtime. It is especially difficult to estimate wall clock times, as the runtime not only depends on the algorithms and the degree of parallelism used in the application.

Moreover, it is influenced by environmental issues (e.g. the current system load or network load) that are often difficult to predict and change in unforeseen ways. If a job's runtime is underestimated the scheduler terminates it. This causes a loss of computation.

There are two common ways out to deal with the runtime estimation problem. First, the application can be decomposed into smaller phases after each of which the application can resume if it was terminated.

However, this increases both programming effort and code complexity.

Second, the runtime can be overestimated extensively (twice what is estimated or maximal allowed runtime). By accepting the penalty of extra long waiting in the cluster queues users "buy" a likely complete run of their job. Besides waiting penalties this also causes reservation holes in the job schedules and hence reduced overall cluster throughput.

We propose a solution to this problem that does not suffer from the above mentioned drawbacks and that also makes Grid computing more transparent. If an OpenMP application is about to exceed the reserved time, it is automatically checkpointed and transparently migrated to either a new local reservation or to a reservation on a different accessible (possible remote) system. The target system may have a different architecture or network interconnect. If the available CPU count is changed in this migration, the application is automatically reparallelized and adapts to the new degree of parallelism by adding to or removing threads from the parallel region that was being executed when checkpointing and migration commenced.

Our current research focuses on the migration strategy. We investigate a centralized bidding system. Each accessible cluster runs a small daemon that continuously monitors its load. In regular intervals, the daemon reports a bid that indicates what compute power the cluster can deliver at what time in its future job schedule.

Based on the incoming bids, the runtime system selects the most suitable cluster as the target for migration, i.e., the one that delivers the highest computing power at the earliest time. This repeats until the application has finished.

Keywords: HPC, Cluster, OpenMP, Migration, Reparallelization

Joint work of: Klemm, Michael; Philippsen, Michael

Full Paper:
http://www2.informatik.uni-erlangen.de/Forschung/Publikationen/download/migration-OpenMP.pdf

See also: Michael Klemm, Matthias Bezold, Stefan Gabriel, Ronald Veldema, and Michael Philippsen. Reparallelization and Migration of OpenMP Programs. In: Proceedings of the 7th International Symposium on Cluster Computing and the Grid, pages 529-537, Rio de Janeiro, Brazil, May 2007.

Parallel Programming or High-level Programming?

Christos Kozyrakis (Stanford University, USA)

Parallel computing has always been the ideal field for math wizards and hackers. Consistency models, locking protocols, non-blocking communication, lock-free algorithms, and open-nesting are examples of the complex topics that the experts love to debate. Will mere mortals ever be able to master such mechanisms in order to process petabytes of data using thousands of cores? Unlikely...

This talk will argue that for massive parallelism to become ubiquitous, we need to adopt higher-level forms of programming. The abstractions exposed to the programmers should be closer to the mathematical or physical entities they want to process (sets, collections, relations, probability distributions, arrays). Computations should be expressed in a declarative manner that focuses on the algorithmic properties of the computation rather than its low-level implementation and management. Notice that nothing about these high-level programming models says that they have to be parallel. Nevertheless, I will argue that such models often reveal the concurrency in the computation in the most flexible, portable, and scalable manner.

For high-level programming to be effective for ubiquitous parallelism, we need an extensive infrastructure that can extract, map, and manage the concurrency it captures. This talk will hopefully initiate a discussion on some of the many open issues:

- What are the useful high-level abstractions?
- How do we synthesize multiple abstractions into one program?
- Are domain-specific languages, libraries, or coding patterns the way to write high-level programs?
- What are the necessary low-level primitives to express the concurrency and locality characteristics in the high-level program (iterators, messages, futures, transactions, ...)?
- How much of the mapping & managing problem can we solve statically vs. dynamically?
- How do we manage scalability and portability?
- How do we manage locality? Is scheduling memory more important that scheduling cores?
- How does the high-level programmer tune such a program?
- How do we express and enforce QoS requirements in this environment (real-time, power, ...)?

- What performance overhead does the high-level approach incur? Do we need to spend half of the resources to continuously monitor, manage, and compile program? Is that the best we can do?
- Does knowledge of the high-level requirements of the computation helps us bypass or optimize some of the age-old issues with parallel computing?

Keywords: High level programming models, scalable parallelism

Full Paper:
http://csl.stanford.edu/~christos

Reconsidering Transactional Memory

James Larus (Microsoft Research - Redmond, USA)

Transactional Memory (TM) has been seen by many as a panacea for parallel programming problems. After several years experience implementing TM and a smaller amount of experience using TM, it is possible to make a preliminary assessment of the value of this programming mechanism. Implementations of TM have exposed a number of unresolved issues in the transactional programming model and in the integration of TM into existing programming languages and run-time environments. This talk surveys some of these problems and suggests some open research problems that need to be resolved before TM is adopted in its most general form.

Domain-Specific Programming

Christian Lengauer (Universität Passau, D)

A short basis for discussion of what a domain is and what the types and uses of domain-specific programming are.

Keywords: Domain-specific programming

Domain-Specific Languages for Ubiquitous Parallelism

Calvin Lin (Univ. of Texas at Austin, USA)

This talk explains how the Broadway system offers a promising mechanism for creating parallel domain-specific libraries. The key idea is to encapsulate domain-specific information that a compiler can exploit.

The current Broadway compiler accepts as input the source code of an application, the source code of a library, and a set of annotations that describe the library. With this information, the Broadway compiler can perform domain-specific data-flow analyses and transformations.

This talk summarizes empirical results that show that Broadway successfully optimizes programs written using the PLAPACK parallel linear algebra package, matching manually-optimized code that requires deep knowledge of the PLAPACK implementation. This talk also shows how the same compiler and annotation language, configured with a different set of annotations, can be used to detect a number of security vulnerabilities in a set of 18 Open Source C programs, consisting primarily of systems utilities and servers. Finally, this talk conjectures that the Broadway architecture is ideal for conveying domain-specific information that could be used to create parallel libraries from sequential libraries.

This new goal is significantly more ambitious and would require changes to both the Broadway annotation language and the Broadway compiler.

Keywords: Broadway compiler, domain-specific information, parallelization

Embedded Systems & Parallel Programming

Peter Marwedel (Universität Dortmund, D)

Embedded Systems typically have to respect tight constraints. In particular, they have to respect tight constraints for the consumed energy. Therefore, hardware platforms for embedded systems are characterized by their energy efficiency. The efficiency of embedded processors has been increased in order to get close to the efficiency of bare silicon.

Ed Lee showed in 2005 that the characteristics of threads do not match well with the requirements for specification techniques for embedded systems. This observation motivates research on models of computation other than the Von Neumann approach.

Software generation is being used for safety-critical systems. For example SCADE is employed for generating code for Airbus planes and dSPACE Targetlink is employed for generating control software in the automotive domain. This removes some of the problems with generating parallel programs manually.

Many systems are specified in the form of task graphs. Software exists which maps such task graphs onto parallel processors. In particular, we consider Thiele's approach for generating optimized architectures for signal processing and Symtavision's approach of mapping automotive applications onto execution control units (ECU). This way, parallel implementations are generated without having to care about locks, monitors etc.

Using accelerators based on reconfigurable logic attached to a standard processor is another approach to parallel programming of embedded systems. Techniques from high-performance computing should be applicable to this approach.

Keywords: Embedded Systems, Parallel Programming

ARTIST2 and ArtistDesign networks of excellence

Peter Marwedel (Universität Dortmund, D)

For its 6th Framework (round of funding), the European Commission decided to introduce so-called *networks of excellence* (NoEs). Such networks consist of excellent researchers (known for a particular area) and their research groups. The main purpose of these networks is to improve the cooperation between researchers and between researchers and academia. Certain amounts of money are made available to support cooperating partners. The ARTIST2 network on Advanced Real-Time and Embedded Systems is an example of such networks. The scope of this network includes all aspects of real-time and embedded systems.

Due to the good amount of cooperation, there will be a follow-up network, called ArtistDesign. ArtistDesign will start in 2008.

Keywords: Embedded System, network of excellence, ARTIST, ArtistDesign

Into the abyss: Parallel Programming for the masses

Samuel P. Midkiff (Purdue University, USA)

Multicore processors have fundamentally changed the way software is developed.

In particular, no longer can features be added to software with increases in processor clock speeds covering any overhead. Rather, programs must be parallel to see performance gains with new processors. This forces the average programmer to perform duties previously only expected of heroic programmers.

A partial solution to this problem is domain specific languages. Three examples were given: HPF, a compiler for polymer chemistry, and Aspen, a language for network services. We show these languages raise the level of abstraction high enough that respective domain experts can achieve good performance on parallel machines while remaining oblivious to the parallel structure of final application.

Parallelism through Digital Circuit Design

John O'Donnell (University of Glasgow, GB)

We commonly assume a sharp distinction between computer hardware and software: the digital circuit designer takes logic gates as primitives to build processors, while the programmer takes processors as primitives to build applications.

Specialists in one of these domains seldom cross into the other.

For many years this division of labour has sufficed. The increased circuit density that resulted from improved chip manufacturing has enabled circuit designers to produce faster processors, and everyone benefited. That era has now ended, and the current trend is toward multiprocessors rather than bigger processors.

But there is an alternative way to exploit all those extra transistors that are now available: you can use them to provide special assistance to ordinary processors, or even to design circuits that solve problems directly.

That approach is often unreasonable, because traditional processors solve many problems efficiently. There are, however, some applications where it does make sense to combine circuit design with programming. Two examples are specialised functional units and data parallel processors. There has been progress in languages and software tools to support such hybrid systems as well as algorithms that make the most of their capabilities. Combining hardware with software also raises some intriguing new questions in computational complexity.

Keywords: Hardware software codesign, data parallel, FPGA, models of computation

Parallelism through Digital Circuit Design

John O'Donnell (University of Glasgow, GB)

Two ways to exploit chips with a very large number of transistors are multicore processors and programmable logic chips. Some data parallel algorithms can be executed efficiently on ordinary parallel computers, including multicores. A class of data parallel algorithms is identified which have characteristics that make implementation on multiprocessors inefficient, but they are well suited for direct design as digital circuits. This leads to a programming model called circuit parallelism. The characteristics of circuit parallel algorithms are discussed, and a prototype system for supporting them is described.

Keywords: Circuit parallelism, data parallelism, FPGA

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2008/1372

GRAIL: A Generic Reconfigurable Affine Interconnection Lattice

Sanjay Rajopadhye (Colorado State University, USA)

Affine control loops (ACLs) constitute an important class of compute and data intensive programs. Domain specific SIMD architectures are an attractive target for direct hardware implementation of such programs.

This compilation requires, in the most general case, interconnections which (i) are affine functions of the PEs and (ii) change dynamically with program execution. We propose a reconfigurable, on-chip, interconnection network, called GRAIL, that supports such temporally changing affine interconnections through constant time reconfiguration. It is generic in the sense that the same fabric can be (re)used to support any affine communication pattern. The GRAIL may be viewed as a non-blocking circuit-switched multistage interconnection network that has an attractive cost-connectivity tradeoff vis-a-vis traditional networks. We describe the properties of interconnections realized by a GRAIL, and show how the switch settings as well as the dynamic reconfigurations can be determined systematically.

Keywords: Polyhedral model, affine dependence programs, dynamically reconfigurable interconnect, massively parallel architectures, non-programmable arrays

A High Productivity Programming Infrastructure for Parallel and Distributed Computing

Lawrence Rauchwerger (Texas A&M University, USA)

The Standard Template Adaptive Parallel Library (STAPL) is a collection of generic data structures and algorithms that provides a high productivity, parallel programming infrastructure with an approach that draws heavily in design from the C++ Standard Template Library (STL).

By abstracting much of the complexities of parallelism from the end user, STAPL provides a platform for high productivity by enabling the user to focus on algorithmic design instead of lower level parallel implementation issues. In this talk, we provide an overview of the major STAPL components, discuss its framework for adaptive algorithm selection, and show that several important scientific applications can be written with relative ease in STAPL and still have scalable performance.

Automatic Parallelization with Hybrid Analysis

Lawrence Rauchwerger (Texas A&M University, USA)

Hybrid Analysis (HA) is a compiler technology that can seamlessly integrate all static and run-time analysis of memory references into a single framework capable of generating sufficient information for most memory related optimizations.

In this talk, we will present Hybrid Analysis as a framework to perform automatic parallelization of loops.

For the cases when static analysis does not give conclusive results, we extract sufficient conditions which are then evaluated dynamically and can (in)validate the parallel execution of loops.

The HA framework has been fully implemented in the Polaris compiler and has parallelized 22 benchmark codes with 99% coverage and speedups superior to the Intel Ifort compiler.

Keywords: Hybrid analysis, program analysis, automatic parallelization

Experiences with shared memory programming in scientific computing

Gudula Rünger (TU Chemnitz, D)

Shared memory programming has a long history in parallel programming and is especially appropriate for adaptive or irregular applications.

With the advent of multi-core processors there are new opportunities for this programming style and shared memory will spread out into many areas of software development.

The talk will present and discuss experiences with shared memory programming on recent parallel machines and multi-core processors.

Keywords: Shared memory programming, multi-core programming , applications

Unleashing the power of simdization

Ayal Zaks (IBM - Haifa, IL)

Fine-grain data-level parallelism can be exploited using SIMD or short- vector architectures, which have become ubiquitous in DSP's, desktops and servers. Programmers have benefited from such instructions for some time using both programming language extensions as well as automatic vectorization or simdization by compilers. Yet some limitations still hinder wider usage and efficiency of SIMD extensions, mostly involved with gathering data to feed the SIMD processing engine, including its programmability, compilability and portability. In this short talk we'll discuss several approaches that aim to overcome these limitations.

Controlling Nondeterminacy in Parallel programs with Shared Memory

Christoph von Praun (IBM TJ Watson Research Center, USA)

When sequential programs are extended with parallel code, shared memory multithreading is the common programming model. The transition from a sequential to a correct parallel program is however a difficult task.

Race conditions are a common source of error, which can introduce (undesired) nondeterminism. We show how language support can alleviate the task of parallelization: Parallel programs are by default determinate, nondeterminism can be introduced selectively. The key idea is to define an implicit order among parallel tasks and resolve race conditions among tasks accordingly.

The ideas are illustrated as extensions of the X10 programming language. The approach leads quickly to a correct parallelization of an application with irregular memory access and unstructured data dependencies (umt2k). We also show the importance of task scheduling and data access locality and demonstrate how such performance critical aspects can be addressed at an algorithmic level.

Keywords: Parallel programming, thread level speculation, determinacy

Full Paper:
http://portal.acm.org/ft_gateway.cfm?id=1229443&type=pdf