10191 Abstracts Collection
Program Composition and Optimization:
Autotuning, Scheduling, Metaprogramming and Beyond
— Dagstuhl Seminar —

Christoph W. Kessler[1], Welf Löwe[2], David Padua[3] and Markus Püschel[4]

[1] Linköping University, SE
chrke@ida.liu.se
[2] Linnaeus University, Växjö, SE
welf.lowe@lnu.se
[3] University of Illinois - Urbana, US
padua@uiuc.edu
[4] Carnegie Mellon University, Pittsburgh, USA
pueschel@ece.cmu.edu

Abstract. From May 9 to 12, 2010, the Dagstuhl Seminar 10191 "Program Composition and Optimization: Autotuning, Scheduling, Metaprogramming and Beyond" was held in Schloss Dagstuhl – Leibniz Center for Informatics. 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. Software composition, program optimization, components, parallel computing, scheduling, auto-tuning, adaptivity, performance prediction, library synthesis, meta-programming

10191 Executive Summary – Program Composition and Optimization: Autotuning, Scheduling, Metaprogramming and Beyond

Components are a well-proven means of handling software complexity. Reusable components and software composition support the construction of large and reliable software systems from pre-defined and tested partial solutions. When maximizing reusability, we end up with components that are very general and do not fit one particular scenario perfectly. Therefore, adaptation, especially optimization, is established as a technique to deal with such mismatches.

Keywords: Software composition, program optimization, components, parallel computing, scheduling, auto-tuning, adaptivity, performance prediction, library synthesis, meta-programming

Joint work of: Kessler, Christoph W.; Löwe, Welf; Padua, David; Püschel, Markus

Extended Abstract: http://drops.dagstuhl.de/opus/volltexte/2010/2571

Program Composition and Optimization: An Introduction

Software composition connects separately defined software artifacts. Such connection may be in program structure (such as inheritance), data flow (such as message passing) and/or control flow (such as function calls or loop control).

Keywords: Software composition, program optimization, components, parallel computing, scheduling, auto-tuning, adaptivity, performance prediction, library synthesis, meta-programming

Joint work of: Kessler, Christoph W.; Löwe, Welf; Padua, David; Püschel, Markus

Extended Abstract: http://drops.dagstuhl.de/opus/volltexte/2010/2573

From modules, aspects to reusable composition operators

Mehmet Aksit (University of Twente, NL)

This talk introduces the aspect-oriented programming from the perspective of our own contribution to this field.

In late 80's, we designed the Sina language; Sina introduced the concept of interface predicates (OOPSLA'88), which generalized the object-oriented composition mechanisms such as aggregations, inheritance and delegation. One could express a large set of composition mechanisms just by defining appropriate interface predicates. This work can be considered as the first example of aspect-oriented programming languages; every interface predicate can be seen as a domain specific aspect which implements message-dispatch based composition.

In begin 90's, the problem of inheritance anomaly was defined (Matsuoka, Bergmans, etc.). It was found out that class inheritance as it is defined by OO languages cannot inherit synchronization code as desired.

This was the start of discussions on software composition problems. As a continuation of the work on the Sina language, and inspired by the work on inheritance anomalies, we have generalized the interface predicate concept to different domains such as synchronization, real-time, coordinated behavior (OOPSLA'92, ECOOP'92, ECOOP'94, JPDC'96); This approach was termed as the Composition-Filters model, and since then, it has been adopted by a number of practical languages.

Towards end 90's, the term *aspect-oriented programming* was introduced by Kiczales. At that time there were several (about 4) aspect-oriented research

languages available, each with different ways of tackling the software composition problem.

During 2000's, the Compose* language was defined and implemented, which enhanced the Composition Filters model with new features such as prolog-based composition designator and a set of built-in tools to verify the composed software (CACM'02, AOSD'09). Compose* has some unique feature such as declarative aspect specifications, clear separation of aspects from objects and components, and implementation language independent composition language. Compose* can also be considered as a multi-paradigm language, which combines imperative, declarative and functional programming styles together.

During the last 2 years, we have been working on the Co-op language (AOSD'10), which allows definition of reusable composition operators as first class entities and it unifies both object-based and aspect-based composition mechanisms together.

In this talk we also refer to the future work which we aim to carry out.

Keywords: Software composition aspect-oriented programming composition filters

FastFlow: high-level yet efficient streaming applications on multi-cores

Marco Aldinucci (University of Torino, IT)

FastFlow is an open source programming environment specifically targeting streaming applications on cache-coherent shared-memory multi-cores (http://mc-fastflow.sourceforge.net). FastFlow is implemented as a stack of C++ template libraries built on top of lock-free (and fence-free) synchronization mechanisms.

In FastFlow, different layers are targeted to support different kind of programmers. FastFlow can be directly used to set up an arbitrary network of parallel activities (low-level programming layer); at this level, similarly to what happens programming with POSIX threads, any orchestration of parallel activities can be expressed. However, as for POSIX threads, writing a correct and efficient program is a non-trivial activity.

At the next layer up (high-level programming layer), FastFlow provides programmers with a number of pre-defined parametric programming patterns (i.e. skeletons); at this level, similarly to what happens programming with Intel TBB, some orchestration of parallel activities can be expressed: programs are composed by configuring and combining patterns (skeletons), which carry a optimised implementation; writing a correct and efficient program at this level is fairly easy.

The FastFlow high-level skeletal layer can be further abstracted (using skeletons as object factories) to define Problem Solving Environments (PSEs), which are programming frameworks designed to ease the development of efficient parallel applications in a specific domain. As an example, we are currently working on the following PSEs: FastFlow software accelerator and self-offloading; Parallel Monte Carlo and Gillespie simulations (FastFlow Stochkit); Parallel macro

data-flow interpretation with automatic parallelization feature supporting skeletal programming; a (blazing fast) parallel memory allocator.

The three described layers are thought for three kind of users, respectively: FastFlow designers, skilled programmers (with some knowledge of parallel programming), and casual programmers (e.g. application domain experts).

Keywords: Multi-core, parallel programming, streaming, skeletons, accelerator, non-blocking, synchronization, lock-free, function offload

Safe Feature Composition

Sven Apel (Universität Passau, DE)

Feature-oriented software development (FOSD) is an emerging paradigm that provides a multitude of formalisms, methods, languages, and tools for building well-structured, customizable, and extensible software systems. The idea is to decompose software along its end-user visible features and to generate tailored software systems based on feature selections of users. The set of valid feature combinations of a domain is called a software product line.

In this talk, I will give an overview of some recent developments in this field. Especially, I will concentrate on recent attempts to ensure correctness properties throughout the FOSD process. This includes work on type checking, formal verification, and feature interaction analysis of feature-oriented software product lines.

Keywords: Feature-Oriented Software Development, Software Product Line, Feature Interaction

Component-Based Software Engineering is like Bierkasten Research

Uwe Assmann (TU Dresden, DE)

Component-based software engineering (CBSE) was initiated as a research field at the first Int. Conf. on Software Engineering in 1968, pushed by a talk of Doug McIlroy, in which he challenged his discipline to research into a component technology for component-based software markets.

Over time, the CBSE discipline has discovered that component technology needs component models and composition languages. Many such composition systems have been developed, providing a component model, composition technique and composition language. These composition systems can be arranged in a ladder, showing progress over time. The newer approaches (grey-box compositions) do no longer work require black-box components, but allow for merging of design-time components to run-time components, enabling the component-based development of tightly-integrated systems.

Finally, we present three research challenges for CBSE: weaving of parallel aspects, reuse languages for language-independent composition, and multi-staged composition.

Keywords: Software composition, component-based software engineering, component models

See also: U. Aßmann, *Invasive Software Composition*, Springer, 2003.

At the Heart of the Automation of Linear Algebra Algorithms

Paolo Bientinesi (RWTH Aachen, DE)

It is well understood that in order to attain high performance for linear algebra operations over multiple architectures and settings, not just one, but a family of loop-based algorithms have to be generated and optimized. In the past we have demonstrated that algorithms and routines can be derived automatically, using a procedure based on formal correctness and classical formal derivations techniques.

At the heart of such a procedure lie the Partitioned Matrix Expressions (PMEs) of the target operation; these expressions describe how parts of the output operands can be represented in terms of parts of the input operands. The PMEs are the unifying element for all the algorithms in the family, as they encapsulate the necessary knowledge for generating each one of them. Until now, the PMEs were considered inputs to the derivation procedure, i.e., the users had to provide them. In this talk we discuss how from a high-level formal description of the operation it is possible to generate automatically even the PMEs. We conclude demonstrating how automation becomes critical in complex, high-dimensional, scenarios.

Keywords: Symbolic computations, PME, automation

Joint work of: Bientinesi, Paolo; Fabregat, Diego

Management of non functional concerns in component applications

Marco Danelutto (University of Pisa, IT)

We discuss the problem of autonomic management of non-functional features in component based parallel computations. We show that a single non functional concern (such as performance, security, power management, fault tolerance, etc.) can be efficiently managed when the structure of the parallel component composition is known using a rule based MAPE control loop.

A single non functional concern may be even better managed if a hierarchy of managers is considered, modeled after hierarchical composition of components using well know parallel composition patterns.

When tackling co-management of multiple non functional concerns, several problems have to be solved, related to the coordination of independent autonomic manager decisions.

We discuss preliminary results defining a common ground to be agreed/shared among independent autonomic managers to ensure feasibility of manager cooperation, as well as simple distributed agreement protocols ensuring that coordinated decisions can be taken and ineffectively ones avoided.

Keywords: Structured parallel programming, autonomic management, control loop, self tuning

Joint work of: Danelutto, Marco; Aldinucci, Marco; Kilpatrick, Peter, Xhagijka, Vamis

Automatic Generation of SIMD-Vectorized DSP Kernels

Franz Franchetti (Carnegie Mellon University - Pittsburgh, US)

SIMD Vector instruction set extensions like SSE and AVX, AltiVec/VMX, and the Larrabee new instructions offer the potential of high performance gains by providing fine-grain parallel operations on subwords. These extensions provide 2-way to 16-way single-precision and 2-way to 8-way double-precision vector units, as well as integer vector support. Many compilers have vectorization and SIMDizing support and provide substantial speed-up. Among the leading compilers are Intel's C++ compiler, the GNU C compiler, IBM's XL C compiler, and the PGI compiler. However, for computational kernels from the signal processing domain, compiler-based utilization of SIMD instruction sets does not provide the possible speed-up. The major reason is that the arithmetic density of kernels like the fast Fourier transform is low, i.e., $O(n \log n)$ operations for $O(n)$ data. Thus, the cost of vector shuffle operations is substantial, and developers of high-performance libraries often resort to hand-tuned assembly code (or C code with SIMD intrinsics) to obtain the necessary performance.

Spiral (www.spiral.net) is a program and hardware design 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 unpractical at the code or design level.

In this talk we present Spiral's SIMD vector code generation framework. It is based on a high-level structural model of SIMD instructions, a rewriting engine that uses backtracking search to span a space of fully vectorized algorithms,

and empirical search in this space to pick the best (autotuned) implementation. Moreover, architecture-specific rewriting rules fine-tune the instruction selection process. Spiral's vector code generation for the FFT and DCTs first uses short vector algorithm variants that ensure a low shuffle count, with all shuffles localized in a small number of small building blocks. The most important of these building blocks are automatically generated from the instruction set specification, using algebraic identities or superoptimization.

Structural Scoping of Behavioral Variations

Robert Hirschfeld (Hasso-Plattner-Institut - Potsdam, DE)

Context-oriented Programming, or COP, provides programmers with dedicated abstractions and mechanisms to concisely represent behavioral variations that depend on execution context. By treating context explicitly, and by directly supporting dynamic composition, COP allows programmers to better express software entities that adapt their behavior late-bound at run-time.

So far, most COP language extensions (including ours) solely support dynamic or global scoping of behavioral variations. While working on several applications, we realized that these scoping strategies, while interesting in several application scenarios, need to be complemented with others. One such strategy is structural scoping.

In our talk we will illustrate these concepts, their application, and their implementation by developing a sample scenario, and demonstrate that they are largely independent of other commitments to programming style.

Keywords: Context-oriented Programming, COP, Run-time Adaptation, Structural Scoping, Behavioral Variations

See also:
http://www.hpi.uni-potsdam.de/swa/

See also:
http://www.hpi.uni-potsdam.de/swa/cop/

Tool Demo: Reuseware Composition Framework

Jendrik Johannes (TU Dresden, DE)

In this tool presentation we demonstrate the Reuseware Composition Framework that is an open-source model and code composition tool for the Eclipse platform.

Reuseware allows language developers to extend modelling and programming languages to support new kinds of components (e.g., aspects). Language users can then use the tooling offered by Reuseware inside Eclipse in combination with other modelling tools. We demonstrate both the tooling for language developers to quickly add component support to an arbitrary language and and the tooling for language users to specify and compose model components.

Keywords: Software composition

Access/execute metadata for composition in multicore and manycore applications

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

"Access/execute decoupling" is really the essence of what people mean by streaming - the ability to separate access to data from the computation part, and ideally, to express the mapping from points in the kernel's iteration space to data locations explicitly, and declaratively. This talk will present some instances of this, what we have called the "AEcute" model, in image processing and unstructured-mesh CFD. AEcute functions as a unifying intermediate representation for several different domain-specific manycore tools we are developing - and is perhaps also a useful model for programming directly. Along the way I'll show a couple of different domain-specific program generation tools we're working on, and show how they succeed in isolating high-level algorithmic concerns from architecture-specific optimisation choices - and that raising the level of abstraction can yield cleaner code *and* higher performance.

Context-aware Composition

Welf Löwe (Linnaeus University - Växjö, SE)

To maximize their reuse in different contexts, library components come in variants. Variants range from different algorithms and data-structures implementing a certain component interface to different numbers of processors and schedules used for the implementation.

Variants induce different qualities of the component and, hence, the composed system. Qualities are non-functional properties like execution time, the system's footprint size, memory and energy consumption, required number of processors etc.

Software composition becomes an optimization problem: find the best-fit variants for a system. What is considered the optimization goal, i.e., the definition of "best", depends on the system requirements. For instance, the goal could be to optimize performance or memory consumption or a merger of the two. Even the optimization of some qualities under the constraints that others hold a threshold is possible. However, in general, it is impossible to select the best-fit variant in a static composition process. The performance, e.g., is usually data-dependent which is unknown before runtime.

Context-aware composition is a novel software composition technique which composes variants of components dynamically. Based on a composition context situation evaluated at runtime, context-aware composition selects and invokes the best-fit component variant. What the best-fit variant is for a certain context situation for a certain optimization goal is pre-computed in a training phase before runtime, e.g., when the system is deployed, using training data. The

training infrastructure and the training data are designed statically, just like the test infrastructure and test data are today.

We present an approach that generates context-aware, optimized components.

The search space contains combinations of implementation variants of algorithms, their schedules to processors and the data structures used including dynamically switching and converting between them. Based on profiling, the best implementation for a certain context is precomputed at deployment time and selected at runtime.

In our experiments, the context-aware composition approach outperforms the individual variants in almost all cases.

Keywords: Dynamic composition, optimization

Joint work of: Löwe, Welf; Kessler, Christoph

See also: C. Kessler, W. Löwe: Optimized composition of performance-aware parallel components. Proc. 15th Int. Workshop on Compilers for Parallel Computers (CPC-2010), Vienna, Austria, 2010.

Making Parallel Programs Auto-Tunable: A View From Software Engineering

Victor Pankratius (KIT - Karlsruhe Institute of Technology, DE)

Multicore systems with several processors on a chip have arrived on every desktop. Software developers need to write and optimize parallel programs for performance. This contribution presents several case studies of complex non-numerical programs, showing that automatic performance tuning will become indispensable in the software engineer's portfolio. The results show that software architecture information can be exploited to automatically prune the search space and improve performance on different software abstraction layers. Moreover, parallel code becomes easier to write and its quality improves (e.g., with respect to readability, portability, hard-coded optimizations, maintenance, debugging).

Program Composition for Performance Portability with PEPPHER

Sabri Pllana and Jesper Larsson Träff (Universität Wien, AT)

PEPPHER is a newly started EU FP7 project on enhancing programmability and performance portability for heterogeneous many-core architectures.

We outline the aims of the project, and discuss the role of software component composition towards realizing the goals of PEPPHER (www.peppher.eu).

Keywords: PEPPHER, (performance) portability, annotation, composition, run-time, autotuning, libraries

Joint work of: The PEPPHER Consortium, www.peppher.eu

Computer Generation of General Size Linear Transform Libraries

Markus Püschel (Carnegie Mellon University - Pittsburgh, US)

The development of high-performance libraries has become extraordinarily difficult due to multiple processor cores, vector instruction sets, and deep memory hierarchies. Often, the library has to be reimplemented and reoptimized, when a new platform is released. In this paper we show how to automatically generate general input-size libraries for the domain of linear transforms. The input to our generator is a formal specification of the transform and the recursive algorithms the library should use; the output is a library that supports general input size, is vectorized and multithreaded, provides an adaptation mechanism for the memory hierarchy, and has excellent performance, comparable to or better than the best human-written libraries.

Keywords: Automatic performance tuning, library generation, high-performance computing, decision trees, statistical classifier, machine learning, fast Fourier transform, FFT

Joint work of: Yevgen Voronenko, Frédéric de Mesmay and Markus Püschel

Full Paper:
http://spiral.ece.cmu.edu:8080/pub-spiral/abstract.jsp?id=129

See also: Proc. International Symposium on Code Generation and Optimization (CGO), pp. 102-113, 2009

Autotuning

Markus Püschel (Carnegie Mellon University - Pittsburgh, US)

We give a small introduction to autotuning.

Keywords: Software, performance optimization

Scheduling and auto-tuning techniques in the context of time-stepping methods

Thomas Rauber (Universität Bayreuth, DE)

In this talk, we discuss how scheduling algorithms and auto-tuning techniques can be integrated into time-stepping methods.

For a parallel execution, this allows a re-organization of the mapping of computations to execution resources dynamically between time steps to adapt the execution to the workload of the execution platform.

The basis for such a re-organization are task-based and block-based formulations of the computations, which do not a priori fix the mapping to the resources of the execution platform.

As example for time-stepping methods, we consider solution methods for ordinary differential equations, which typically perform a large number of sequential time-steps.

The STAPL Parallel Container Framework

Lawrence Rauchwerger (Texas A&M University, US)

The Standard Template Adaptive Parallel Library (STAPL) is a parallel programming infrastructure that extends C++ with support for parallelism. STAPL provides a run-time system, a collection of distributed data structures (pContainers) and parallel algorithms (pAlgorithms), and a generic methodology for extending them to provide customized functionality.

Parallel containers are data structures addressing issues related to data partitioning, distribution, communication, synchronization, load balancing, and thread safety. In this talk we will present the STAPL Parallel Container Framework (PCF), which is designed to facilitate the development of generic parallel containers. We introduce a set of concepts and a methodology for assembling a pContainer from existing containers and data distribution information. The STAPL PCF distinguishes itself from existing work by providing a large number of basic data structures (e.g., pArray, pList, pVector, pMat, pGraph, pMap, pSet) and allowing users to compose and customize existing pContainers for improved application expressivity and performance.

We evaluate the performance of the STAPL pContainers on a CRAY XT4 massively parallel processing system. We show that the pContainer methods, generic pAlgorithms, and different graph applications, all provide good scalability on more than 10K processors.

Keywords: Parallel, STAPL, STL, container, distributed

Programming support for applications structured by parallel tasks

Gudula Rünger (TU Chemnitz, DE)

The programming with parallel tasks is a suitable programming technique to implement parallel applications consisting of a set of well-defined submodules.

In this programming model, the application can be coded as a parallel program with mixed parallelism in which the submodules represent parallel tasks each of which can be executed on one or more processors of the target platform.

To facilitate the programming with parallel tasks, the specification of a parallel task program can be separated from its actual execution.

Suitable scheduling and mapping algorithm can then be employed to find an efficient implementation variant for a specific parallel platform.

Invasive program composition using aspects

Mario Südholt (Ecole des Mines de Nantes, FR)

Software composition frequently requires the definition ("extraction") of composition interfaces for functionalities that previously have been implicit. We present recent results supporting such invasive composition tasks using aspect-oriented programming, corresponding formal foundations and applications to the composition and optimization of sequential and distributed programs.

Architectural programming using invasive distributed patterns

Mario Südholt (Ecole des Mines de Nantes, FR)

Software composition frequently requires the definition ("extraction") of composition interfaces for functionalities that previously have been implicit. We present recent results supporting such invasive composition tasks using aspect-oriented programming, corresponding formal foundations and applications to the composition and optimization of sequential and distributed programs.

Programming models and autotuning for generalized n-body problems

Richard Vuduc (Georgia Institute of Technology, US)

I outline our on-going effort to build a programming model and parallel software infrastructure for an important class of computations known as generalized n-body problems (GNPs; Gray & Moore, NIPS 2000). GNPs appear in both the physical sciences and in massive-scale data analysis, and prominent examples include the fast multipole method (FMM) for particle physics, and k-nearest neighbor search and kernel density estimation for data analysis.

The key development challenge our work addresses is that an "optimal" implementation of a particular GNP solver on current multicore and manycore systems requires a complex combination of careful data layouts, vectorization,

mixed precision, and automated algorithmic and code tuning. We present a detailed example in the context of the FMM, including a surprising finding in the debate on the performance and energy-efficiency of general-purpose multicore CPU vs. GPU processors.

This talk is joint work with a cast of characters from applied math, machine learning, and HPC, including Alex Gray, George Biros, Sam Wiliams, Lenny Oliker, Aparna Chandramowlishwaran, Ryan Riegel, and Aashay Shringarpure.

Optimizing and tuning the fast multipole method for state-of-the-art multicore architectures

Richard Vuduc (Georgia Institute of Technology, US)

This work presents the first extensive study of single- node performance optimization, tuning, and analysis of the fast multipole method (FMM) on modern multicore systems. We consider single- and double-precision with numerous performance enhancements, including low-level tuning, numerical approximation, data structure transformations, OpenMP parallelization, and algorithmic tuning.

Among our numerous findings, we show that op- timization and parallelization can improve double- precision performance by 25× on Intel's quad-core Nehalem, 9.4× on AMD's quad-core Barcelona, and 37.6× on Sun's Victoria Falls (dual-sockets on all systems). We also compare our single-precision version against our prior state-of-the-art GPU-based code and show, surprisingly, that the most advanced multicore architecture (Nehalem) reaches parity in both perfor- mance and power efficiency with NVIDIA's most advanced GPU architecture.

Joint work of: Chandramowlishwaran, Aparna; Williams, Samuel; Oliker, Leonid; Lashuk, Ilya; Biros, George; Vuduc, Richard

See also: IPDPS'10