J. Parallel Distrib. Comput. 74 (2014) 3191–3201

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Towards a performance-portable description of geometric multigrid algorithms using a domain-specific language

Richard Membarth [a,b,*], Oliver Reiche [c], Christian Schmitt [c], Frank Hannig [c], Jürgen Teich [c], Markus Stürmer [d], Harald Köstler [d]

[a] German Research Center for Artificial Intelligence, Germany
[b] Computer Graphics Lab & Intel Visual Computing Institute, Saarland University, Germany
[c] Hardware/Software Co-Design, Department of Computer Science, University of Erlangen-Nuremberg, Germany
[d] System Simulation, Department of Computer Science, University of Erlangen-Nuremberg, Germany

HIGHLIGHTS

- DSL extension to handle image pyramids and grid hierarchies.
- DSL extension to model different multigrid cycle types.
- Generated GPU code shows similar performance compared to hand-tuned implementation.
- We apply the algorithm to high dynamic range compression of 2D X-ray images.

ARTICLE INFO

Article history:
Received 18 July 2013
Received in revised form
22 April 2014
Accepted 19 August 2014
Available online 28 August 2014

Keywords:
Multigrid
Multiresolution
Image pyramid
Domain-specific language
Stencil codes
Code generation
GPU
CUDA
OpenCL

ABSTRACT

High Performance Computing (HPC) systems are nowadays more and more heterogeneous. Different processor types can be found on a single node including accelerators such as Graphics Processing Units (GPUs). To cope with the challenge of programming such complex systems, this work presents a domain-specific approach to automatically generate code tailored to different processor types. Low-level CUDA and OpenCL code is generated from a high-level description of an algorithm specified in a Domain-Specific Language (DSL) instead of writing hand-tuned code for GPU accelerators. The DSL is part of the Heterogeneous Image Processing Acceleration (HIPA^{cc}) framework and was extended in this work to handle grid hierarchies in order to model different cycle types. Language constructs are introduced to process and represent data at different resolutions. This allows to describe image processing algorithms that work on image pyramids as well as multigrid methods in the stencil domain. By decoupling the algorithm from its schedule, the proposed approach allows to generate efficient stencil code implementations. Our results show that similar performance compared to hand-tuned codes can be achieved.

1. Introduction

Mapping algorithms in an efficient way to the target hardware poses challenges for algorithm designers. This is in particular true for heterogeneous systems hosting accelerators like graphics cards. While algorithm developers have profound knowledge of the application domain, they often lack detailed insight into the underlying hardware of accelerators in order to exploit the provided processing power. To tackle this problem, OpenCL,[1] a new industry-backed standard Application Programming Interface (API) that inherits many traits from CUDA, was introduced in order to provide software portability across heterogeneous systems: correct OpenCL programs will run on any standard-compliant implementation. OpenCL per se, however, does not address the problem of *performance portability*; that is, OpenCL code optimized for one accelerator device may perform dismally on another, since performance may significantly depend on low-level details, such as data layout and iteration space mapping [11].

In this paper, a different approach is taken by decoupling algorithms from their schedule in a DSL. This allows to map

* Corresponding author at: German Research Center for Artificial Intelligence, Germany.
E-mail addresses: richard.membarth@dfki.de (R. Membarth), oliver.reiche@cs.fau.de (O. Reiche), christian.schmitt@cs.fau.de (C. Schmitt), hannig@cs.fau.de (F. Hannig), teich@cs.fau.de (J. Teich), markus.stuermer@cs.fau.de (M. Stürmer), harald.koestler@cs.fau.de (H. Köstler).

http://dx.doi.org/10.1016/j.jpdc.2014.08.008
0743-7315/

[1] http://www.khronos.org/opencl.

algorithms efficiently to a target platform. The DSL is part of the HIPA^{cc} framework [23] that provides also a source-to-source compiler to translate not only the high-level algorithm description into low-level CUDA/OpenCL code, but also to apply transformations and optimizations on the code. To describe algorithms that work on multiple resolutions of the same data, image pyramids are used in image processing [7] and multigrids for stencil computations [5,14]. We present in this work a concise syntax for creating such multiresolution data structures and for processing the data on different resolutions. Kernels described in HIPA^{cc} remain unchanged, only a schedule for iterating over the grid has to be specified.

We consider image pyramid construction and High Dynamic Range (HDR) compression of 2D images as example applications. HDR compression can be done efficiently in the gradient space. For it, the image has to be transformed to gradient space and back. While the forward transformation to gradient space is fast by using simple finite differences, the backward transformation requires the solution of a Partial Differential Equation (PDE).

Multigrid methods are one of the most efficient numerical methods to solve large, sparse linear systems arising for example when discretizing elliptic PDEs. Elliptic PDEs are used to model various physical or technical effects in many application fields. One of the most popular elliptic PDEs is the Poisson equation in order to model diffusion processes. In this work we consider a simple multigrid solver in 2D that employs stencil codes for the Poisson equation and apply it to image processing. In previous work we developed a suitable parallel multigrid algorithm and provided a hand-tuned implementation for this task [17]. Our first description of this multigrid solver in HIPA^{cc} had several drawbacks: images had to be provided for each level and the (mostly identical) computation had to be described repeatedly for each level [22]. Here, for concise modeling of multigrid algorithms, we introduce a suitable representation for multigrid and multiresolution data sets in the DSL as well as a concise syntax for describing the operations on each multigrid level and between different multigrid levels.

The focus of this work is on the concise description of algorithms that operate on different resolutions of the same data:

- We present language constructs in our DSL that allow to describe image pyramids. Data for pyramids is managed by the framework and only the data for the finest level has to be provided. Furthermore, we allow to specify how the pyramid is traversed: this includes typical traversals for construction of pyramids in image processing as well as the V-cycle and W-cycle for multigrid stencil computations.
- We evaluate the implementation of image pyramid construction and of a multigrid application using our image pyramid representation. We show that the proposed representation improves productivity significantly. Furthermore, we show that the description in HIPA^{cc} provides portability across different architectures and allows to achieve competitive performance compared to Halide [30] as well as hand-tuned implementations.

The paper first introduces related work on DSLs and frameworks for stencil codes. Then, an overview of the HDR compression application and the used multigrid solver is given. The HIPA^{cc} framework and its extensions used to model multigrid algorithms in its DSL is introduced thereafter. The paper concludes with an evaluation of the domain-specific approach for stencil codes including productivity, portability, and performance aspects.

2. Related work

In the past, several approaches captured and used knowledge about the domain of stencil codes and their applications in the form of domain-specific languages. The idea is to provide abstractions within the language that are tailored to the domain of stencil-code engineering.

Liszt [9] and Pochoir [32] are stencil compilers for code written in a simple domain-specific language. Pochoir compiles to C++ with Cilk++ extensions and fits the optimized stencil code into a generic, divide-and-conquer template. Pochoir pays particular attention to cache obliviousness on multi-core workstations. However, both languages (Liszt and Pochoir) provide only limited support for the characteristics of the hardware platform.

The hypre library [1] is a collection of high-performance preconditioners and solvers for large sparse linear systems of equations on massively parallel machines. It offers, for example, a stencil-based interface for computations on structured or block-structured grids and also incorporates different multigrid solvers. DUNE [3] is a modular and generic C++ library for the solution of partial differential equations on different kinds of grids. It supports structured or block-structured grids and a variety of algebraic solvers including multigrid is provided as external modules. Both libraries, hypre and DUNE, are flexible and can easily be adapted for stencil applications, but there are neither a domain-specific syntax nor proper editing and debugging facilities, and the stencil code has to be optimized by setting configuration options by hand. There is no specialized syntax for the definition of multigrid solvers. Setting parameters such as the number of pre- and postsmoothing steps and the cycle type is done via functions. Custom cycle types or user-defined restriction and interpolation operators are not supported. All three frameworks provide no native options for execution on GPUs.

PATUS (Parallel Auto-Tuned Stencils) [16,8] is a code generation and auto-tuning framework for stencil computations on shared-memory architectures. The algorithms and stencils are provided by the user and *strategies* can be specified that define parallelization and optimization.

The parallel Optimized Sparse Kernel Interface (pOSKI) [4] is a collection of algorithms for operations involving sparse matrices on uniprocessor and multi-core machines. It includes auto-tuning at installation- and run-time and is suitable for stencil computations yielding special sparse matrices.

Physis [21] provides a DSL for stencil computations based on C with support for GPU accelerators. They hide communication cost by overlapping boundary exchange with stencil computation.

[15] introduces a small DSL for Jacobi-like iterative methods. Efficient code is generated for GPU accelerators by using overlapping tiles for multiple iterations.

Ypnos [28] and Paraiso [27] provide a functional DSL embedded in Haskell for structured grid computations with support for GPU accelerators. Similarly, Halide [30] uses a functional representation to describe image processing algorithms and stencil codes. The programmer then specifies a schedule in Halide for a pipeline of computations separately. This gives the programmer the flexibility to reuse the same algorithm description for different target architectures by specifying target-specific schedules.

In [10] a graphical DSL based on UML activity diagrams is proposed to model multigrid algorithms for applications in variational imaging.

While these frameworks allow to model stencils and to generate efficient code, the stencils are limited to a single level in most cases. Halide [30] and the UML activity diagrams [10] are the only of the aforementioned approaches that allow users to define custom multigrid algorithms to the best of our knowledge. In [10], the data-flow between kernels is modeled and multiple levels can be modeled through a cycle. The implementation of each UML component is provided by the user and, hence, arbitrary computation can be expressed. Halide, in contrast, allows to define arbitrary control flow. For multigrid applications, a loop iterating over the different levels can be used and the results of each level can be stored to an array.

An alternative approach to code optimization and generation in stencil-code engineering is the use of the polyhedron model [13].