Designing a Modern Skeleton Programming Framework for Heterogeneous and Parallel Systems

Licentiate seminar
August Ernstsson
Contents

• Introduction
• Individual contributions
• Dissemination
• Conclusions and future work
Introduction
Programmable computers are everywhere

• Society is increasingly dependent on computer systems
• In all shapes and sizes
• Increasingly more diverse and complex!
• **Problem:** Expert knowledge is needed to efficiently utilize such systems
Algorithmic skeletons

- Approach to parallel programming proposed by Cole in 1989
- Based on functional programming
- Many implementations exist today
  - **Scientific**: SkePU, Musket, GrPPI, FastFlow, ...
  - **Industry**: Nvidia Thrust, SYCL, C++ parallel STL, ...
- Different flavors of parallelism: *data parallelism, task parallelism*
- Different targets: *multi-core, heterogeneous, cluster, ...*
The SkePU framework

• Developed and maintained at Linköping University
• C++-based
• Source-to-source compiler

• Goals
  • Multi-backend
  • Automatic memory management
  • Accessible interface
SkePU skeleton and container set

- Skeletons
  - Map
  - MapPairs
  - MapOverlap
  - Reduce
  - Scan
  - MapReduce
  - MapPairsReduce

- Containers
  - Vector
  - Matrix
  - 3D Tensor
  - 4D Tensor
SkePU programming interface

```c
int add(int a, int b)
{
    return a + b;
}

auto vec_sum = Map(add);

vec_sum(result, v1, v2);
```

<table>
<thead>
<tr>
<th>v1</th>
<th>v2</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>Add 3</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>Add ..</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>Add ..</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>Add ..</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>Add ..</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>Add ..</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>Add ..</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>Add ..</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>Add ..</td>
</tr>
</tbody>
</table>
SkePU backend structure

- Multi-backend with selection and tuning

<table>
<thead>
<tr>
<th></th>
<th>C++</th>
<th>OpenMP</th>
<th>OpenCL</th>
<th>CUDA</th>
<th>MPI + StarPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Multi-core CPU</td>
<td>Accelerator</td>
<td>GPU</td>
<td>Cluster</td>
<td></td>
</tr>
<tr>
<td>C++ Interface</td>
<td>skeletons, smart containers, …</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Contributions
Main contributions of this research

2016 SkePU 2
2017 Lazy eval with tiling
2018 Hybrid backend
2019 Multi-variant UF
2020 SkePU 3 + cluster
Contribution
SkePU 2 with pre-compiler architecture

Abstract
In this article we present SkePU 2, the next generation of the SkePU C++ skeleton programming framework for heterogeneous parallel systems. We critically examine the design and limitations of the SkePU 1 programming interface. We present a new, flexible and type-safe, interface for skeleton programming in SkePU 2, and a source-to-source transformation tool which knows about SkePU 2 constructs such as skeletons and user functions. We demonstrate how the source-to-source compiler transforms programs to enable efficient execution on parallel heterogeneous systems. We show how SkePU 2 enables new use-cases and applications by increasing the flexibility from SkePU 1, and how programming errors can be caught earlier and easier thanks to improved type safety. We propose a new skeleton, Call, unique in the sense that it does not impose any predefined skeleton structure and can encapsulate arbitrary user-defined multi-backend computations. We also discuss how the source-to-source compiler can enable a new optimization opportunity by selecting among multiple user function specializations when building a parallel program. Finally, we show that the performance of our prototype SkePU 2 implementation closely matches that of SkePU 1.

Keywords
Skeleton programming · SkePU · Source-to-source transformation · C++11 · Heterogeneous parallel systems · Portability

August Ernstsson1 · Lu Li1 · Christoph Kessler1

1 PELAB, Department of Computer and Information Science, Linköping University, Linköping, Sweden

Received: 30 September 2016 / Accepted: 18 January 2017 / Published online: 28 January 2017
© The Author(s) 2017. This article is published with open access at Springerlink.com

SkePU 2: Flexible and Type-Safe Skeleton Programming for Heterogeneous Parallel Systems

August Ernstsson1 · Lu Li1 · Christoph Kessler1
SkePU 2

• Original prototype of SkePU 2 from master’s thesis project
• Macro-based library -> source-to-source compiler toolchain
• **New interface**: shift to C++11 (”modern C++”)
  • Great flexibility
  • Improved type safety
• **New implementation**: variadic template meta-programming
  • Builds on algorithms from SkePU 1
SkePU 2 compilation flow

1. Program sources
2. Source-to-source compiler
3. Backend sources (C++, OpenCL, etc.)
4. Backend compiler (e.g., GCC)
5. SkePU runtime library
6. Executable
Contribution

Lazy evaluations with access locality optimization

Summary

We present an extension for the SkePU skeleton programming framework to improve the performance of sequences of transformations on smart containers. By using lazy evaluation, SkePU records skeleton invocations and dependencies as directed by smart container operands. When partial results are required in different parts of the program, the run-time system will process the entire lineage of skeleton invocations. Tiling is applied to keep chunks of container data in the working set for the whole sequence of transformations. The approach is inspired by big data frameworks operating on large clusters where good data locality is crucial. We also consider benefits other than data locality with the increased run-time information given by the lineage structures, such as backend selection for heterogeneous systems. Experimental evaluation of example applications shows potential for performance improvements due to better cache utilization, as long as the overhead of lineage construction and management is kept low.

KEYWORDS

lazy evaluation, loop tiling, skeleton programming, SkePU, smart containers
Lazy evaluation with lineages

- **Inspiration**: Big data analytics, e.g. Apache Spark
- **Idea**: Delay skeleton evaluation
- Collect state information and form dependency graph
- At an *evaluation point*, evaluate the DAG
  - Optimize the computations with global run-time information
Tiling optimization on lineages

- **Observation**: Data-parallel skeleton lineages are separable along the element boundaries
- A full skeleton invocation need not be evaluated in one go
- For a sequence of, e.g., maps, evaluate slices of the data set along the lineage
- Process *chunks* along *cache line size* => temporal access locality
Tiling optimization on lineages

- Parallel polynomial evaluation using Horner’s method

```cpp
skepu::Vector<float> horner_eval_nonfused(
    skepu::Vector<float> &coeffs, skepu::Vector<float> &x_vals)
{
    size_t degree = coeffs.size() - 1;
    auto mult = skepu::Map(mult_f);
    auto add = skepu::Map<1>(add_f);

    skepu::Vector<float> res(x_vals.size(), coeffs(degree));

    for (int i = degree - 1; i >= 0; --i)
    {
        mult(res, res, x_vals);
        add(res, res, coeffs(i));
    }

    return res;
}
```
Hybrid CPU–GPU execution support in the skeleton programming framework SkePU

Tomas Öhberg · August Ernstsson · Christoph Kessler

Published online: 25 March 2019
© The Author(s) 2019

Abstract

In this paper, we present a hybrid execution backend for the skeleton programming framework SkePU. The backend is capable of automatically dividing the workload and simultaneously executing the computation on a multi-core CPU and any number of accelerators, such as GPUs. We show how to efficiently partition the workload of skeletons such as Map, MapReduce, and Scan to allow hybrid execution on heterogeneous computer systems. We also show a unified way of predicting how the workload should be partitioned based on performance modeling. With experiments on typical skeleton instances, we show the speedup for all skeletons when using the new hybrid backend. We also evaluate the performance on some real-world applications. Finally, we show that the new implementation gives higher and more reliable performance compared to an old hybrid execution implementation based on dynamic scheduling.

Keywords: Heterogeneous computing · Hybrid execution · Skeleton programming · Workload partitioning
Hybrid backend

• **Goal**: To optimize utilization of a heterogeneous CPU+GPU system
  • All execution units should be working in tandem
  • Split the workload into smaller tasks and distribute among the system
    • Task scheduling system: *StarPU*?
  • *Partition ratio*: how much work to give to the CPU vs. the GPU?
Hybrid backend – Work partitioning

- Partitioning Map

<table>
<thead>
<tr>
<th>Input 1</th>
<th>Input 2</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>CPU Thread #1</strong></td>
</tr>
</tbody>
</table>

Partition ratio
Hybrid backend – Work partitioning

- Partitioning MapOverlap
Hybrid backend – Work partitioning

- Partitioning Scan
Contribution

Multi-variant user functions

Abstract. Today’s computer architectures are increasingly specialized and heterogeneous configurations of computational units are common. To provide efficient programming of these systems while still achieving good performance, including performance portability across platforms, high-level parallel programming libraries and tool-chains are used, such as the skeleton programming framework SkePU. SkePU works on heterogeneous systems by automatically generating program components, “user functions”, for multiple different execution units in the system, such as CPU and GPU, from a high-level C++ program. This work extends this multi-backend approach by providing the possibility for the programmer to provide additional variants of these user functions tailored for different scenarios, such as platform constraints. This paper introduces the overall approach of multi-variant user functions, provides several use cases including explicit SIMD vectorization for supported hardware, and evaluates the result of these optimizations that can be achieved using this extension.

1. Introduction

Programming of complex multi-core and heterogeneous computer architectures can be a difficult task, especially when there is a desire to fully and efficiently utilize the available processing resources. Managing the required workload distribution, synchronisation, and data management often requires expert knowledge and long-time experience. This is especially true if also performance portability is desired, as different systems can vary widely in terms of both the number and types of processing cores, as well as in other characteristics such as memory hierarchy.

High-level parallel programming frameworks aim to improve on this situation by reducing the user-facing complexity of programs. A small number of highly optimized but still general programming building blocks are presented through a high-level interface. This category of frameworks include application specific languages, PGAS (Partitioned Global Address Space) interfaces, dataflow models, and more, but most importantly for this paper: the skeleton programming concept, borrowing the higher-order operations of functional programming such as map and reduce, and implemented as an abstraction level that is portable across both multi-core and heterogeneous computers and larger supercomputer clusters. Skeleton programming uses generic building blocks encoding V

Parallel Computing: Technology Trends
I. Foster et al. (Eds.) © 2020 The authors and IOS Press. This article is published online with Open Access by IOS Press and distributed under the terms of the Creative Commons Attribution Non-Commercial License 4.0 (CC BY-NC 4.0).
doi:10.3233/APC200074

Multi-Variant User Functions for Platform-Aware Skeleton Programming

August ERNSTSSON1 and Christoph KESSLER1

1 PELAB, Dept. of Computer and Information Science
Linköping University, 581 83 Linköping, Sweden

Abstract. Today’s computer architectures are increasingly specialized and heterogeneous configurations of computational units are common. To provide efficient programming of these systems while still achieving good performance, including performance portability across platforms, high-level parallel programming libraries and tool-chains are used, such as the skeleton programming framework SkePU. SkePU works on heterogeneous systems by automatically generating program components, “user functions”, for multiple different execution units in the system, such as CPU and GPU, from a high-level C++ program. This work extends this multi-backend approach by providing the possibility for the programmer to provide additional variants of these user functions tailored for different scenarios, such as platform constraints. This paper introduces the overall approach of multi-variant user functions, provides several use cases including explicit SIMD vectorization for supported hardware, and evaluates the result of these optimizations that can be achieved using this extension.

Keywords. Skeleton programming, SkePU, Heterogeneous computing, Multi-variant user functions, Vectorization

Parallel Computing: Technology Trends
I. Foster et al. (Eds.) © 2020 The authors and IOS Press. This article is published online with Open Access by IOS Press and distributed under the terms of the Creative Commons Attribution Non-Commercial License 4.0 (CC BY-NC 4.0).
doi:10.3233/APC200074

Multi-Variant User Functions for Platform-Aware Skeleton Programming

August ERNSTSSON1 and Christoph KESSLER1

1 PELAB, Dept. of Computer and Information Science
Linköping University, 581 83 Linköping, Sweden

Abstract. Today’s computer architectures are increasingly specialized and heterogeneous configurations of computational units are common. To provide efficient programming of these systems while still achieving good performance, including performance portability across platforms, high-level parallel programming libraries and tool-chains are used, such as the skeleton programming framework SkePU. SkePU works on heterogeneous systems by automatically generating program components, “user functions”, for multiple different execution units in the system, such as CPU and GPU, from a high-level C++ program. This work extends this multi-backend approach by providing the possibility for the programmer to provide additional variants of these user functions tailored for different scenarios, such as platform constraints. This paper introduces the overall approach of multi-variant user functions, provides several use cases including explicit SIMD vectorization for supported hardware, and evaluates the result of these optimizations that can be achieved using this extension.

Keywords. Skeleton programming, SkePU, Heterogeneous computing, Multi-variant user functions, Vectorization
Multi-variant user functions

- **Inspiration**: Multi-variant components
- **Idea**: Allow expert programmers to provide *hand-tuned* user function variants
  - For use on specific backends only
- SkePU single-source approach otherwise makes a *single* algorithm run on all backends
- Variants are enabled at *compile-time* when the target hardware supports it
  - E.g., A CPU with vectorization instructions
  - *XPDL* platform modeling toolchain is used for feature lookup
Multi-variant user functions

User function variants (subdirectories)
- manifest.hpp
- variant1.cpp
- variant2.cpp

Executable
Program sources
Source-to-source compiler
Backend sources (C++, OpenCL, etc.)
SkePU headers
Backend compiler (e.g., GCC)

Platform.xml
XPDL Compiler

Executable
Multi-variant user functions – Evaluation

- Median image filtering, three approaches to find median value in pixel region

<table>
<thead>
<tr>
<th>Variant</th>
<th>Time complexity</th>
<th>Memory complexity</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double loop</td>
<td>$\mathcal{O}(n^2)$</td>
<td>$\mathcal{O}(1)$</td>
<td>None</td>
</tr>
<tr>
<td>Histogram</td>
<td>$\mathcal{O}(n +</td>
<td>D</td>
<td>)$</td>
</tr>
<tr>
<td>qsort</td>
<td>$\mathcal{O}(n \log n)$</td>
<td>$\mathcal{O}(n)$</td>
<td>C standard library</td>
</tr>
</tbody>
</table>

(Region size grows quadratically)
Contribution
SkePU 3 with new skeletons and cluster backend
SkePU 3

- Collaborations within the *EXA2PRO* research project
- *Application-framework co-design*
  - SkePU framework team working with application partners
- Cluster backend added for *exascale* computations
- Real-world applications being ported to SkePU
- Improved distribution and compatibility
SkePU 3 – New features sample

- Cluster backend
  - StarPU + MPI
  - C and Fortran wrappers

- Tensors
  - MatCol and MatRow
  - Tuple return syntax

- MapPairs
  - Simplified memory consistency model
  - New MapOverlap syntax

- MapPairsReduce
  - Dynamic scheduling on multi-core
  - Performance optimizations
SkePU 3 performance – Brain modeling on cluster

- Brain simulation with 90,000 neurons and 200 time steps
Dissemination and user feedback
Tutorials and labs

• The SkePU toolchain is being used in teaching
  • Part of the multi-core and GPU programming course
• SkePU provides perspective on high-level parallel programming
• Student feedback is used to influence SkePU development
  • E.g.: Revising the MapOverlap interface in SkePU 3
• SkePU has also been demonstrated in several hands-on tutorials in the scientific community
Conclusions and future work
Conclusions

- *Algorithmic skeletons* is one approach for bridging the widening gap between programming interfaces in parallel and heterogeneous systems.
- SkePU implements skeletons with C++ interface and a source-to-source compiler toolchain.
- This research is improving SkePU in several ways:
  - **Programmability** is enhanced with new features and by listening to user experiences.
  - **Performance** is optimized with *lazy evaluation*, *hybrid backends*, and *user function variants*.
  - **Portability** is increased as new systems and application domains can be targeted through the *cluster* backend.
Future work on high-level parallel programming and SkePU

- Work on SkePU continues with several research-oriented and feature-oriented ideas planned
  - **Modernized tuner:** Target more of the full feature set in SkePU 3
  - **Skeleton fusion:** Complements run-time lineage optimization
  - Further application case studies
  - And more... see the thesis!
Thank you for listening.