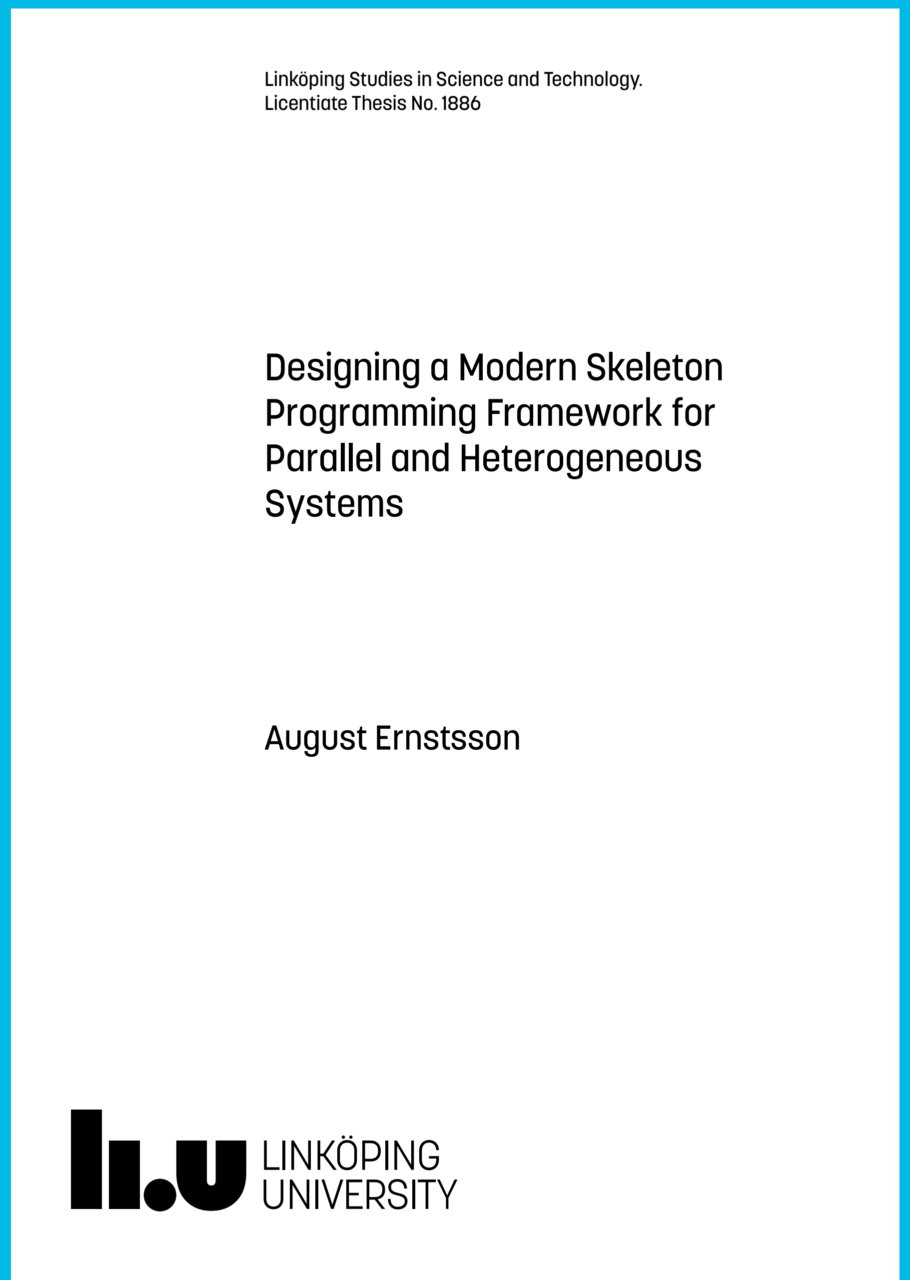
Designing a Modern Skeleton Programming Framework for Heterogeneous and Parallel Systems

Licentiate seminar

August Ernstsson


Linköping Studies in Science and Technology.
Licentiate Thesis No. 1886

Designing a Modern Skeleton Programming Framework for Parallel and Heterogeneous Systems

August Ernstsson

LINKÖPING UNIVERSITY

LINKÖPING UNIVERSITY

Contents

Introduction

Programmable computers are everywhere

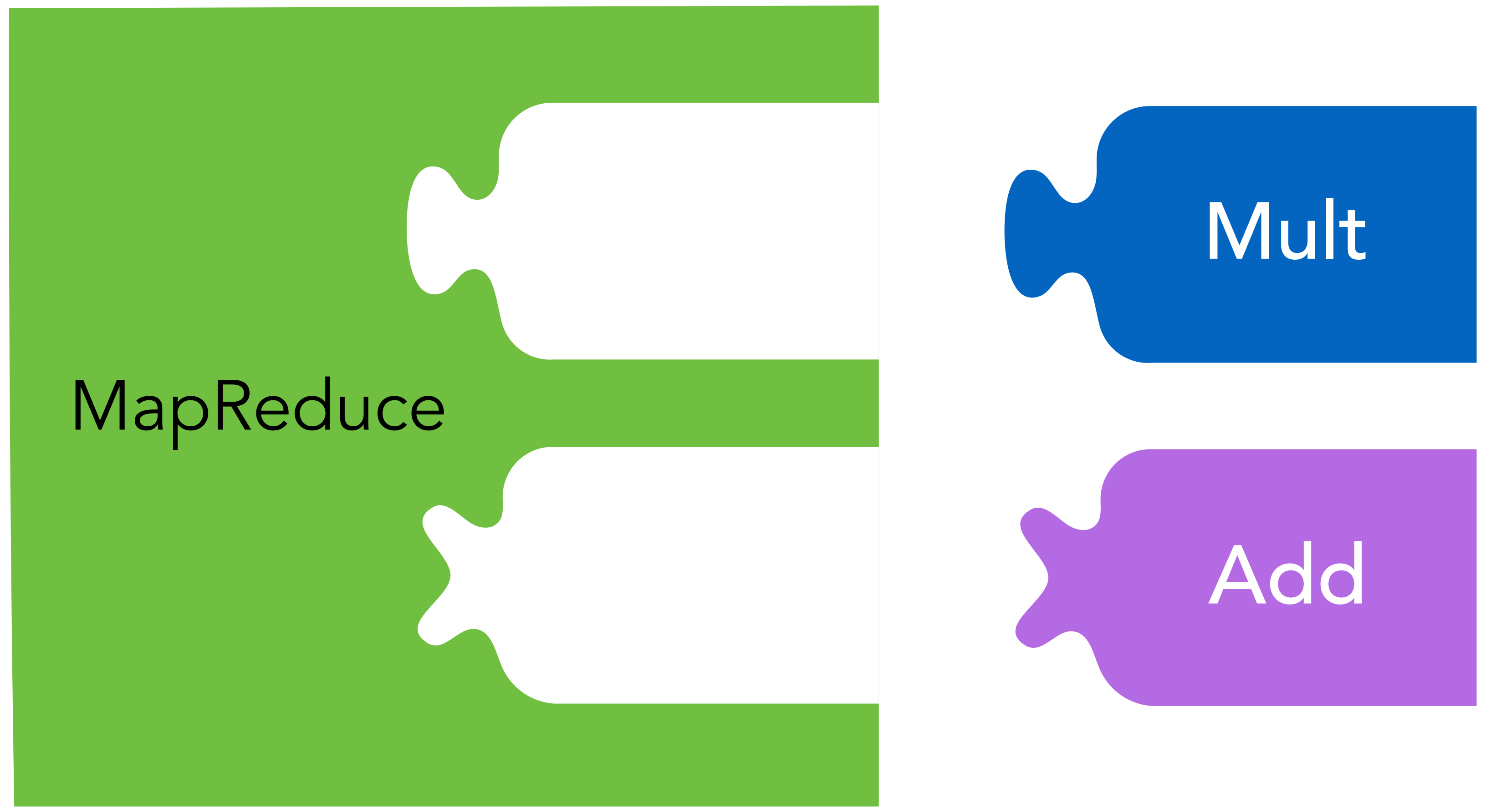
- Society is increasingly dependent on computer systems
- In all shapes and sizes
- Increasingly more diverse and complex!
- **Problem**: Expert knowledge is needed to efficiently utilize such systems

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

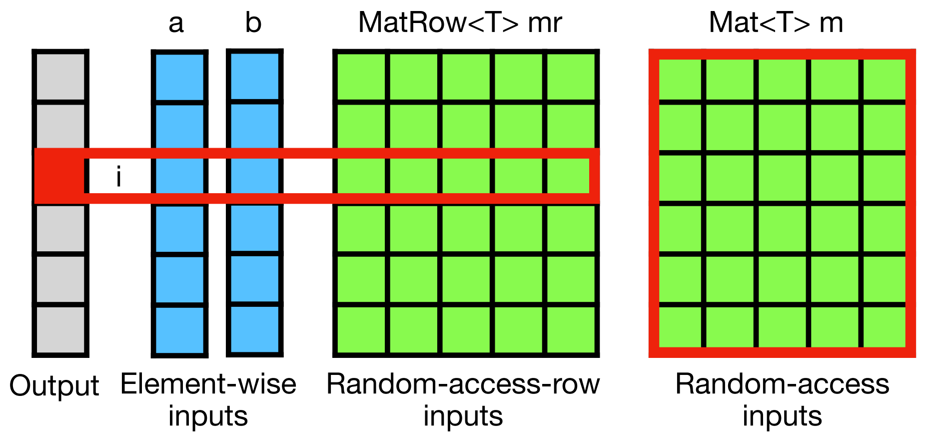
- Developed and maintained at Linköping University
- C++-based
- Source-to-source compiler
- **Goals**
  - Multi-backend
  - Automatic memory management
  - Accessible interface

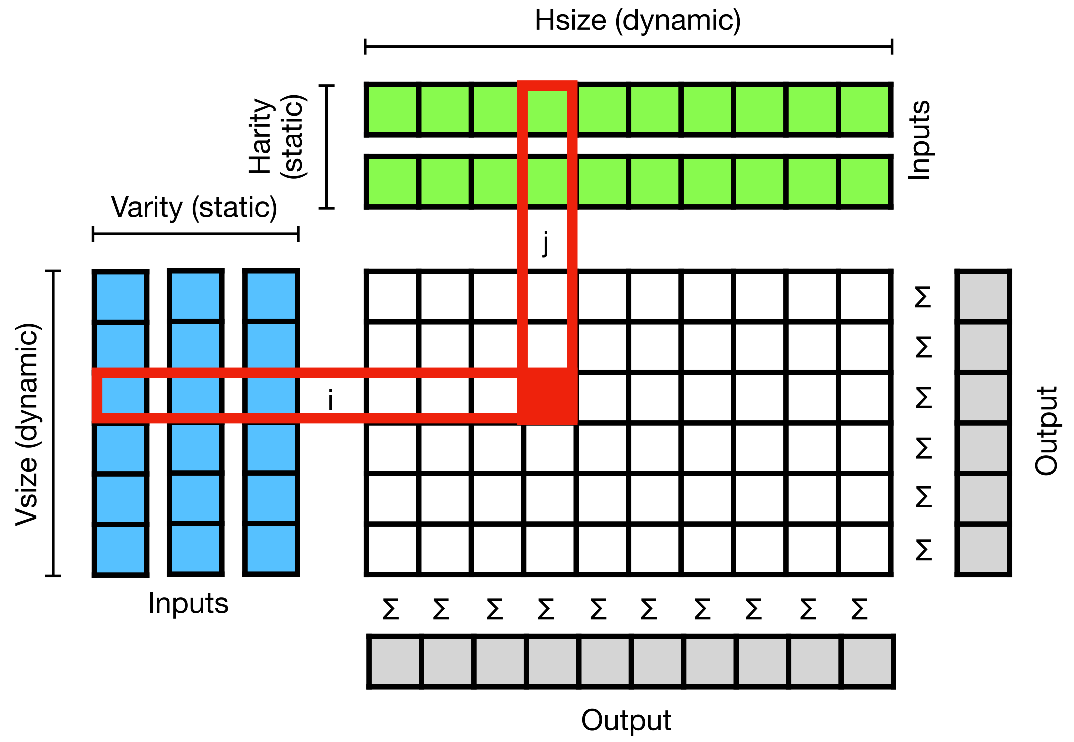


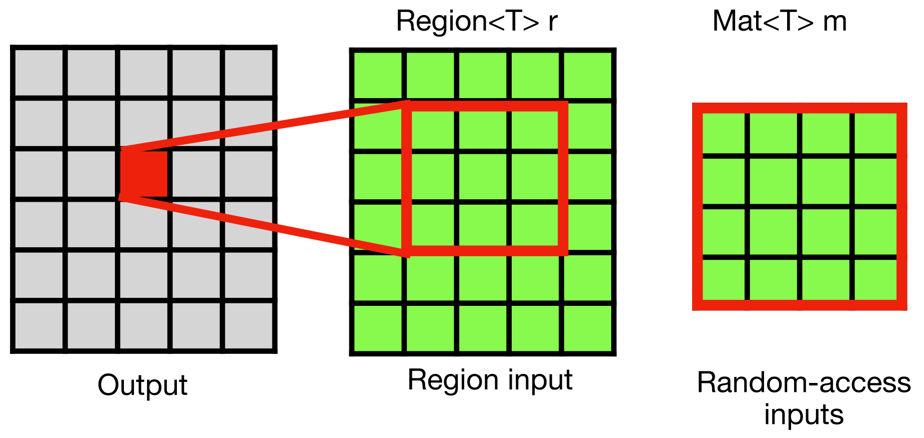
Conceptual illustration of dot product computation in SkePU

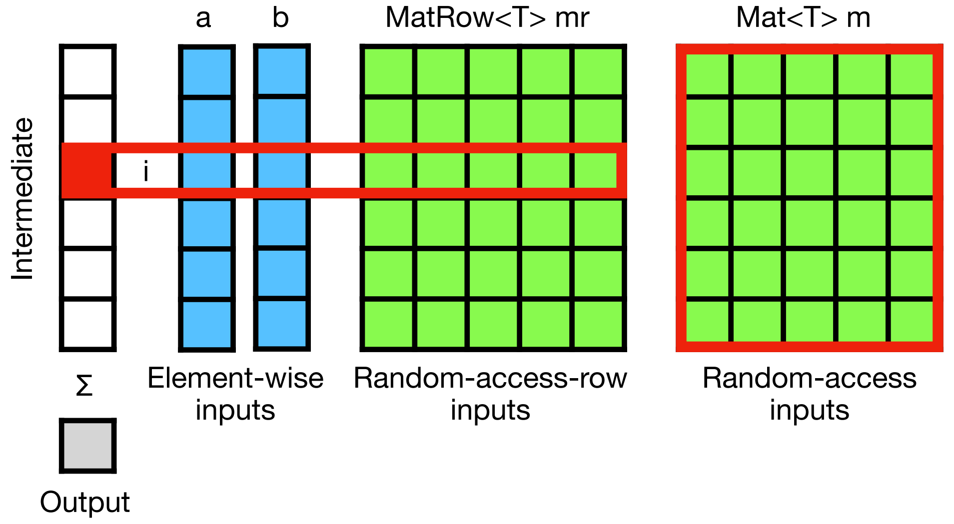
SkePU skeleton and container set

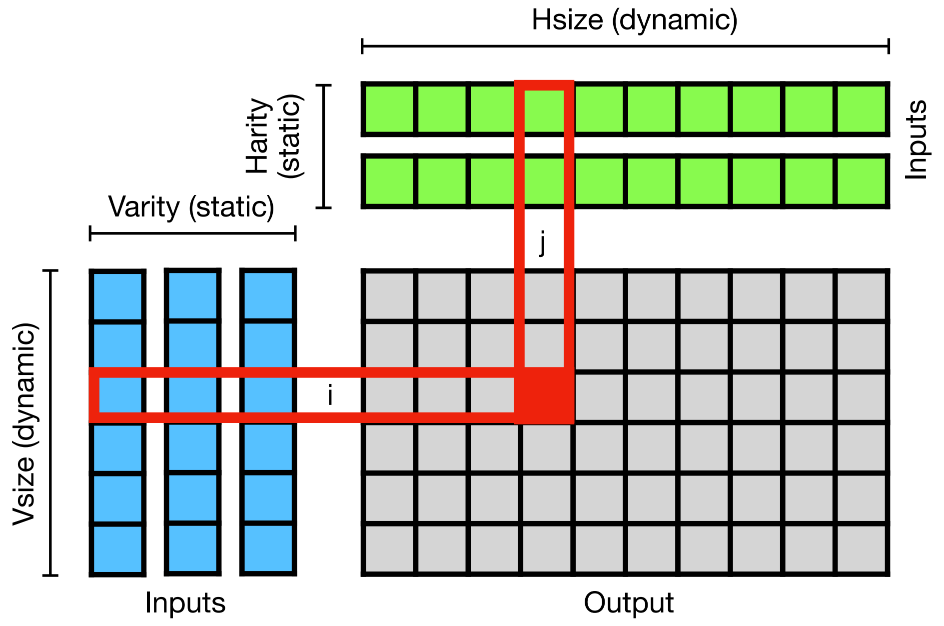
- Skeletons
  - Map
  - MapPairs
  - MapOverlap
  - Reduce
  - Scan
  - MapReduce
  - MapPairsReduce
- Containers
  - Vector
  - Matrix
  - 3D Tensor
  - 4D Tensor









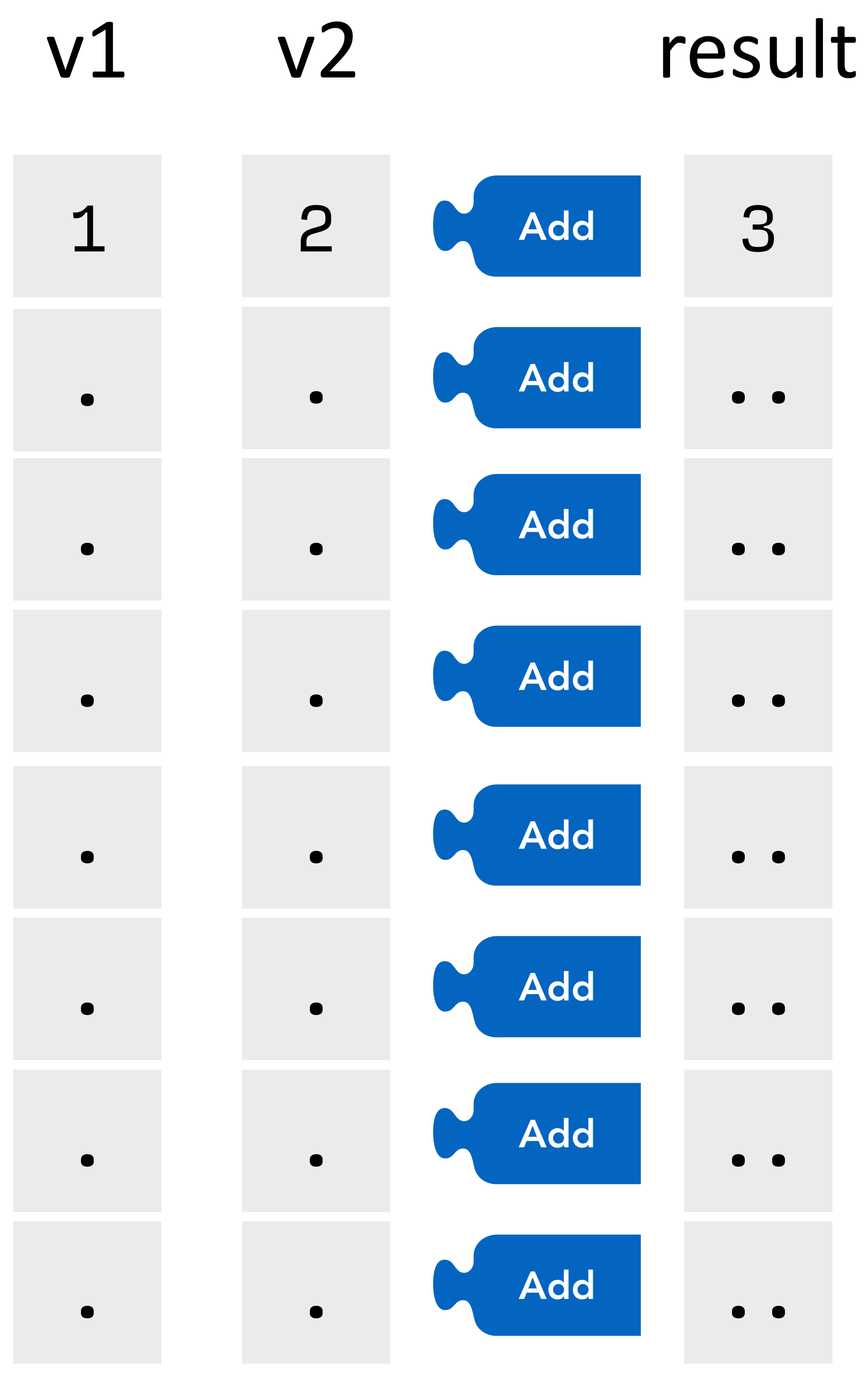


SkePU programming interface

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

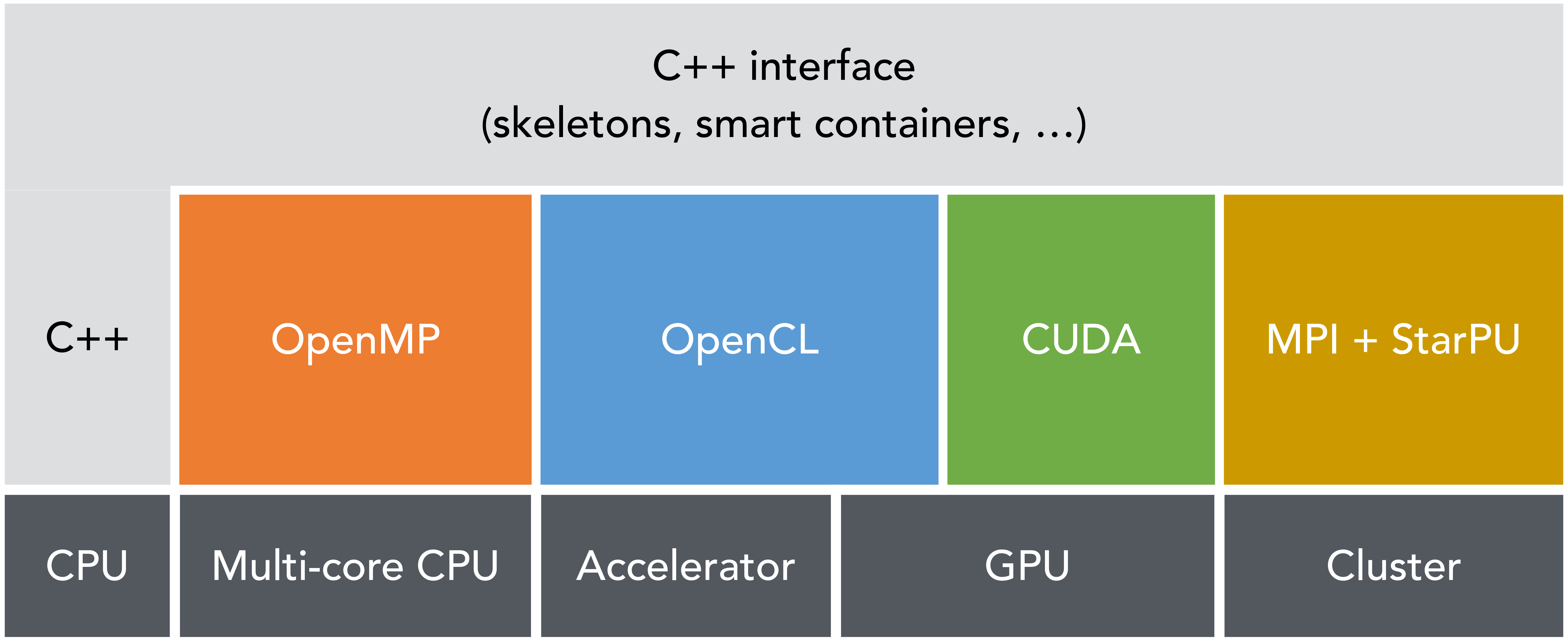
```
auto vec_sum = Map(add);
```

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



SkePU backend structure

- Multi-backend with selection and tuning



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

EXCESS

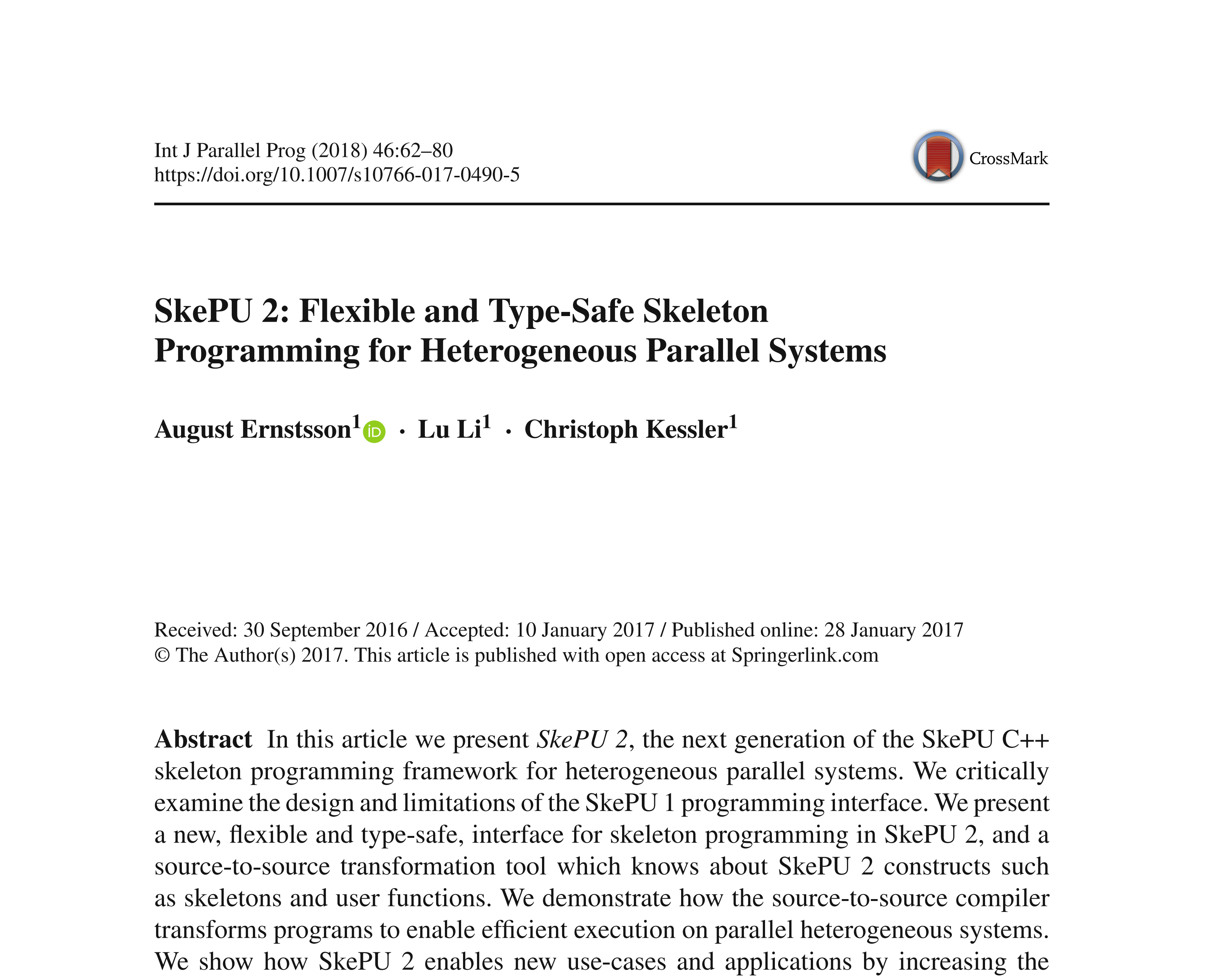
exa2pro
Enhancing Programmability and boosting Performance Portability For Exascale Computing Systems

2016

SkePU 2

Contribution

SkePU 2 with pre-compiler architecture


Int J Parallel Prog (2018) 46:62–80
https://doi.org/10.1007/s10766-017-0490-5

CrossMark

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

August Ernstsson[1] · Lu Li[1] · Christoph Kessler[1]

Received: 30 September 2016 / Accepted: 10 January 2017 / Published online: 28 January 2017

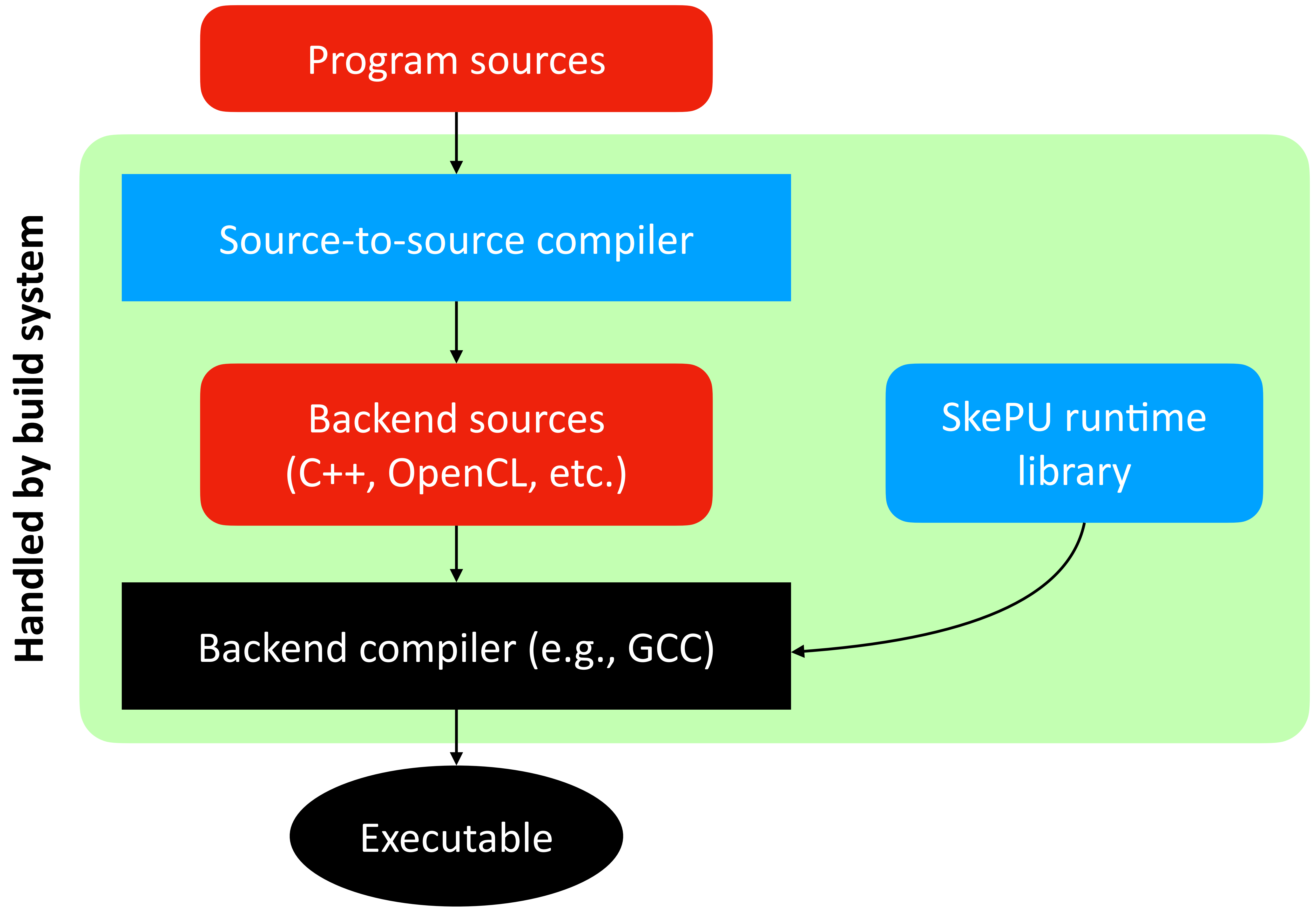
Abstract In this article we present *SkePU 2*, the next generation of the SkePU C++ skeleton programming framework for heterogeneous parallel systems. We critically examine the design and limitations of the SkePU 1 programming interface. We present a new, flexible and type-safe, interface for skeleton 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

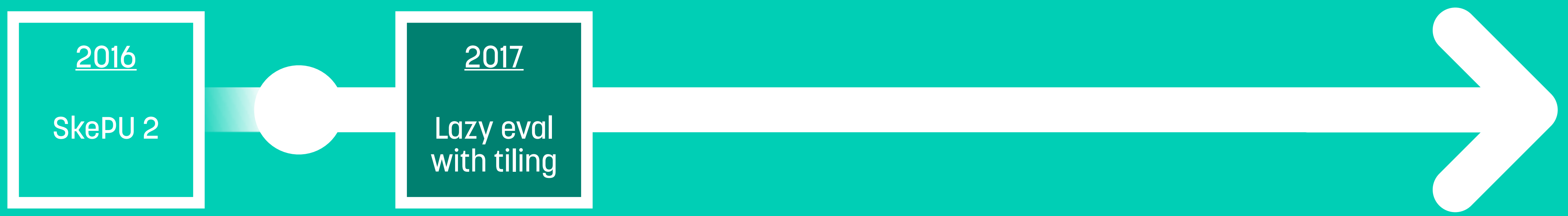
LINKÖPING UNIVERSITY

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

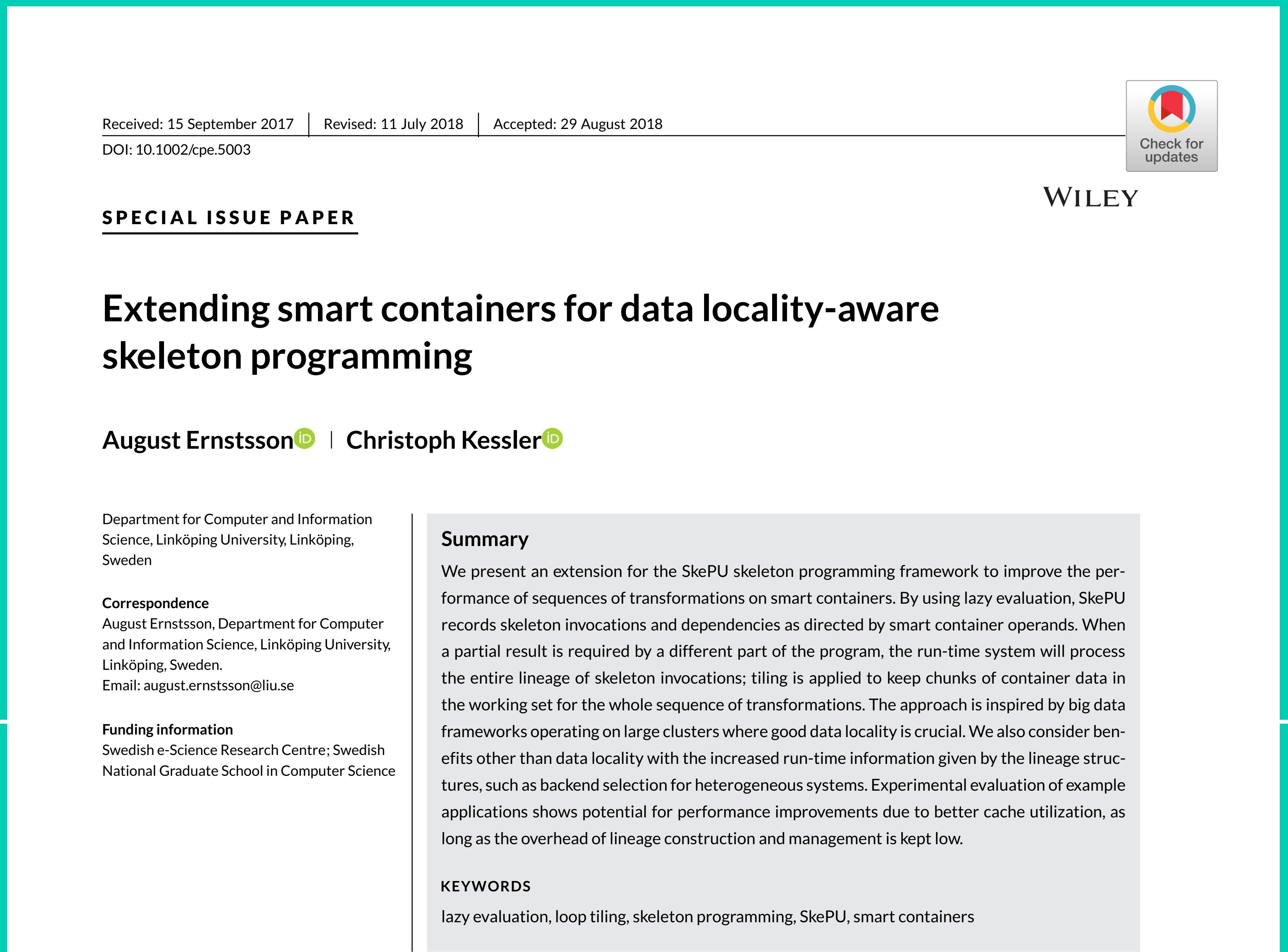
SkePU 2 compilation flow





Contribution

Lazy evaluations with access locality optimization


Received: 15 September 2017 | Revised: 11 July 2018 | Accepted: 29 August 2018
DOI: 10.1002/cpe.5003

WILEY

SPECIAL ISSUE PAPER

Extending smart containers for data locality-aware skeleton programming

August Ernstsson | Christoph Kessler

Department for Computer and Information Science, Linköping University, Linköping, Sweden

Correspondence
August Ernstsson, Department for Computer and Information Science, Linköping University, Linköping, Sweden.
Email: august.ernstsson@liu.se

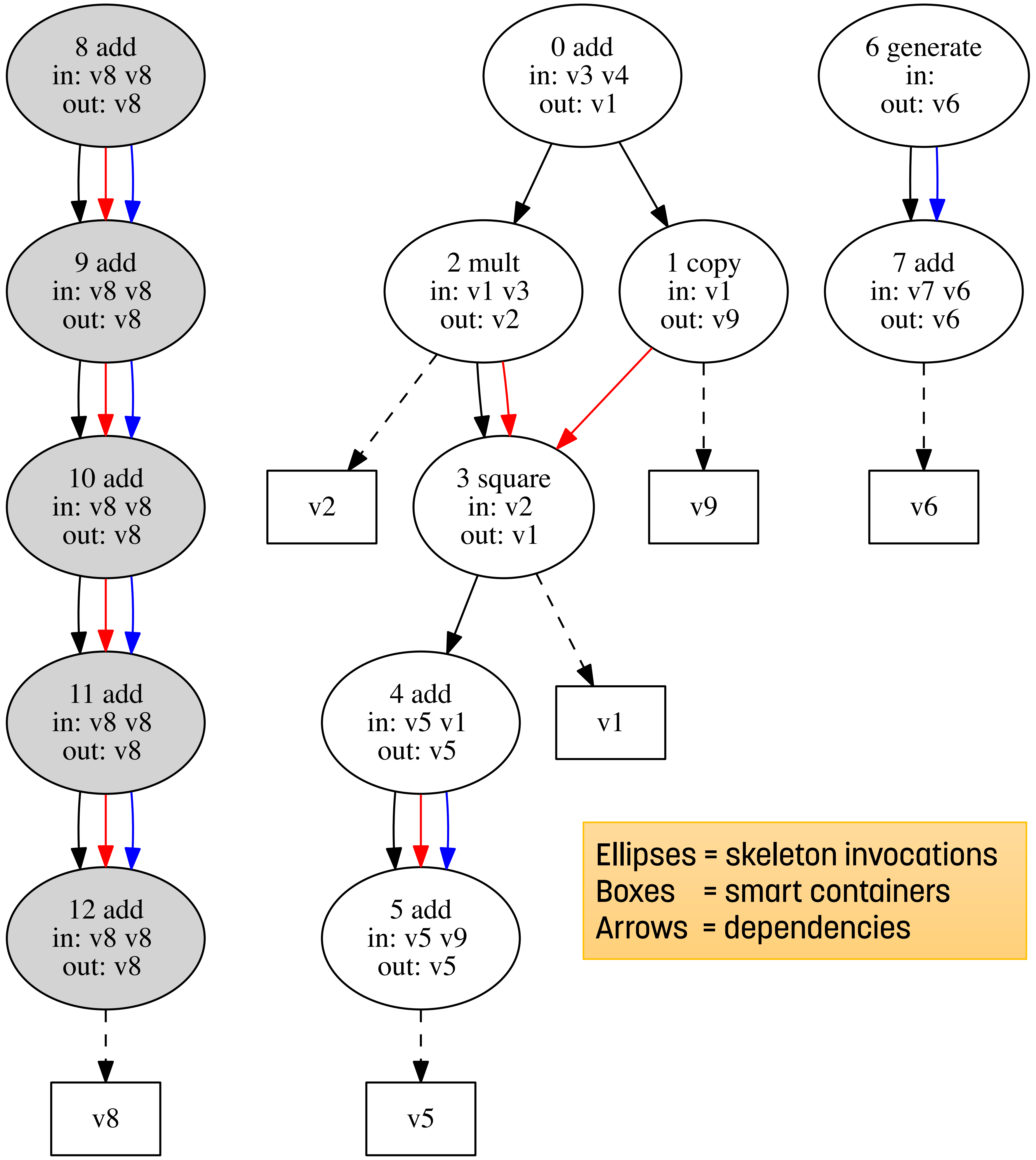
Funding information
Swedish e-Science Research Centre; Swedish National Graduate School in Computer Science

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 a partial result is required by a different part 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

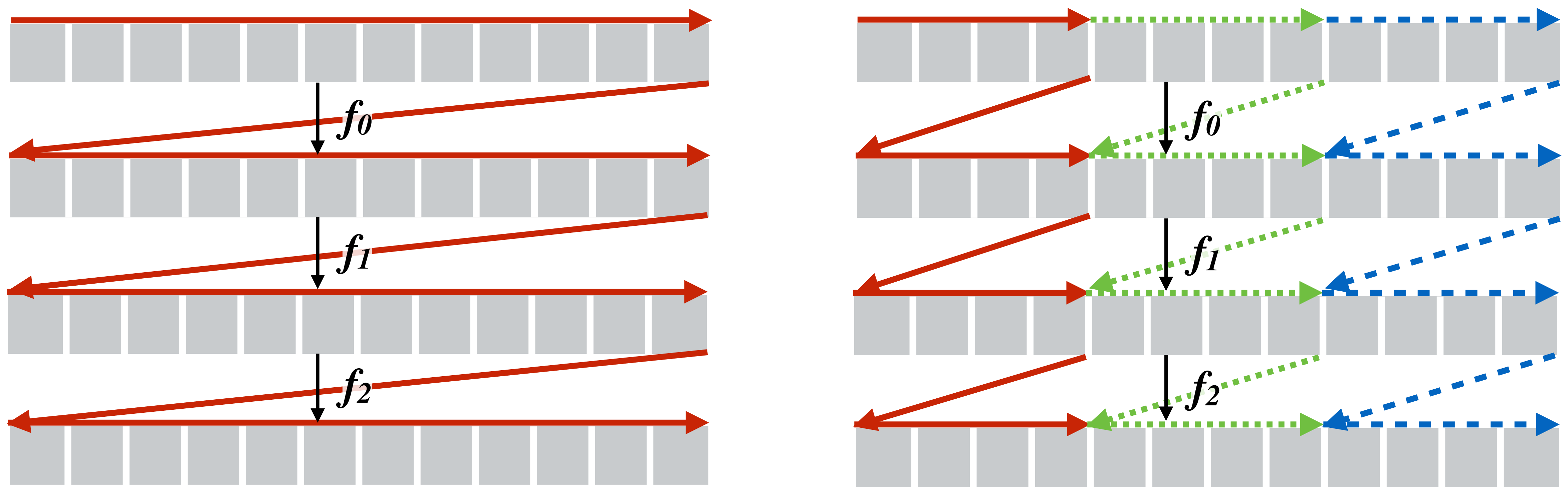
Lazy evaluation with lineages

- **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



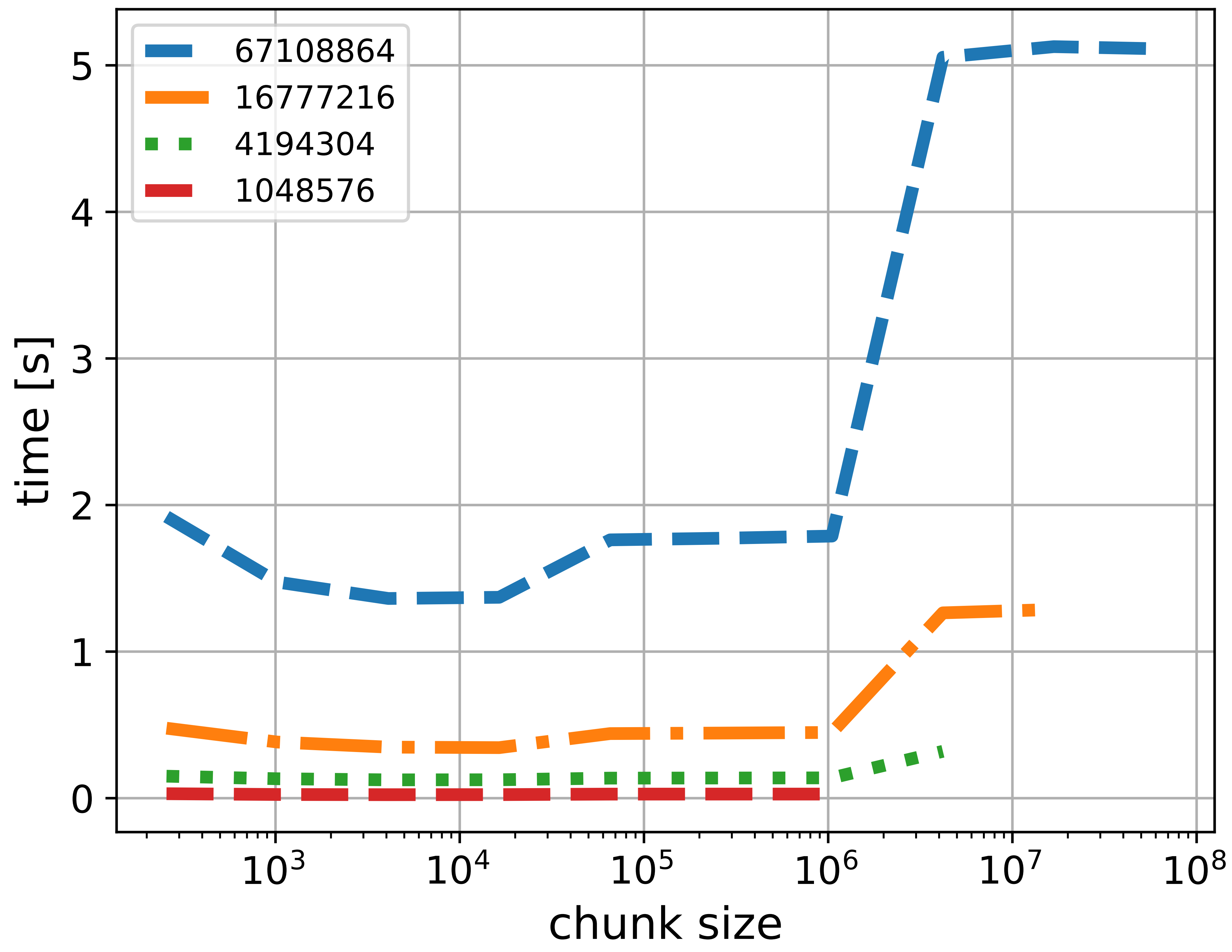
No tiling

Tiling on

Tiling optimization on lineages

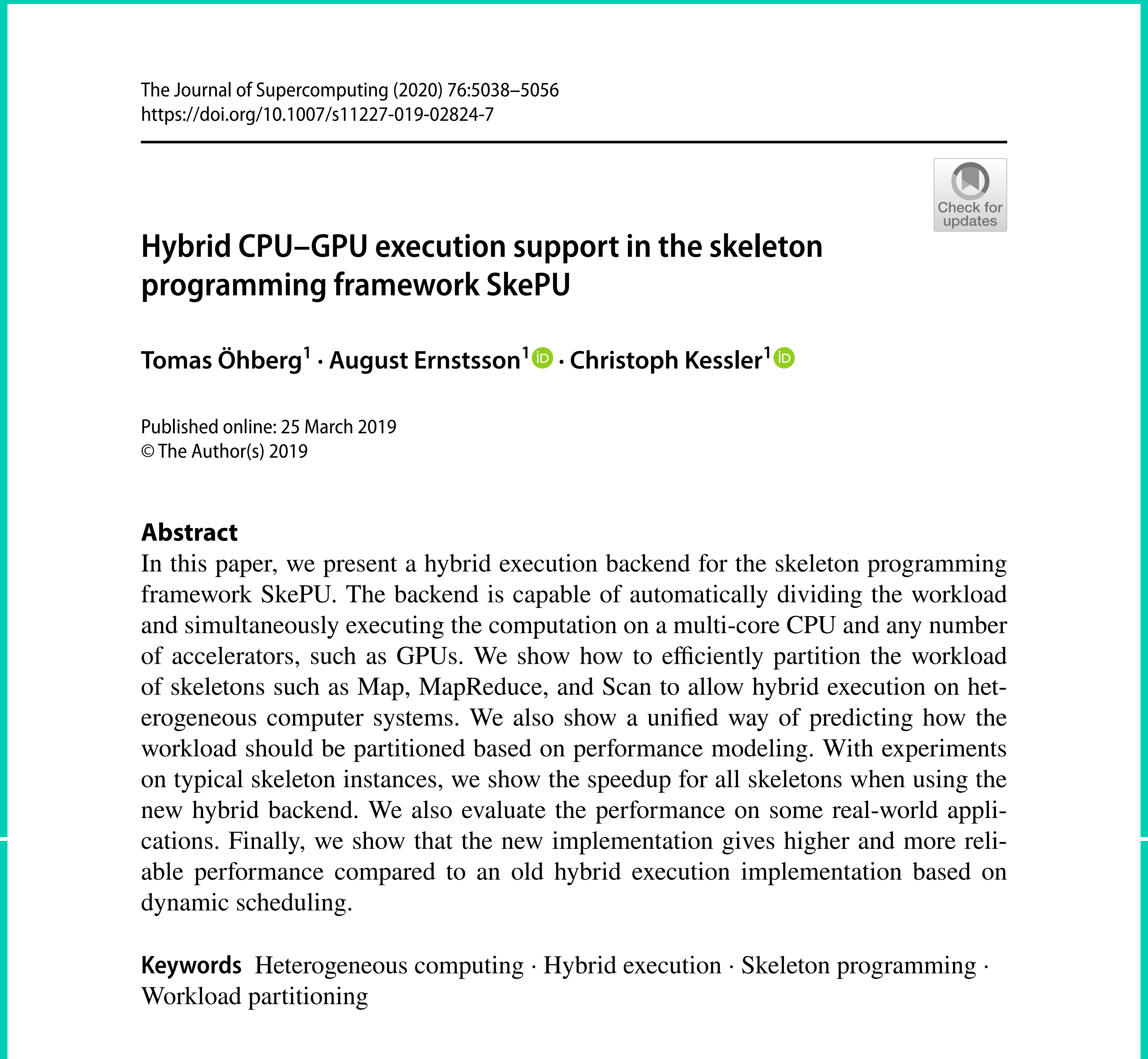
- Parallel polynomial evaluation using Horner’s method

```
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;
}
```



Contribution

Hybrid backend


The Journal of Supercomputing (2020) 76:5038–5056
https://doi.org/10.1007/s11227-019-02824-7

Check for updates

Hybrid CPU–GPU execution support in the skeleton programming framework SkePU

Tomas Öhberg[1] · August Ernstsson[1] · Christoph Kessler[1]

Published online: 25 March 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.

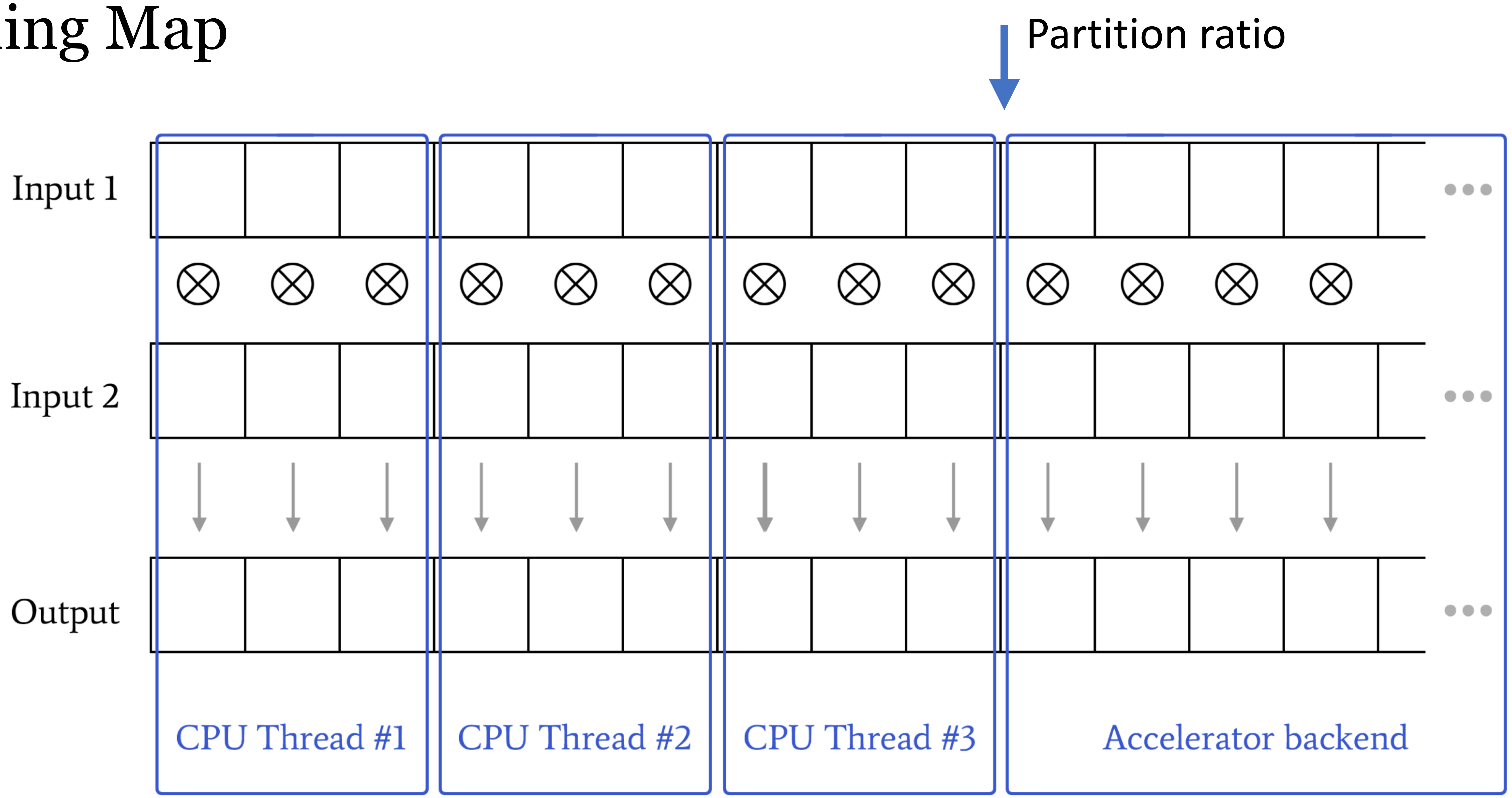
Keywords Heterogeneous computing · Hybrid execution · Skeleton programming · Workload partitioning

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?

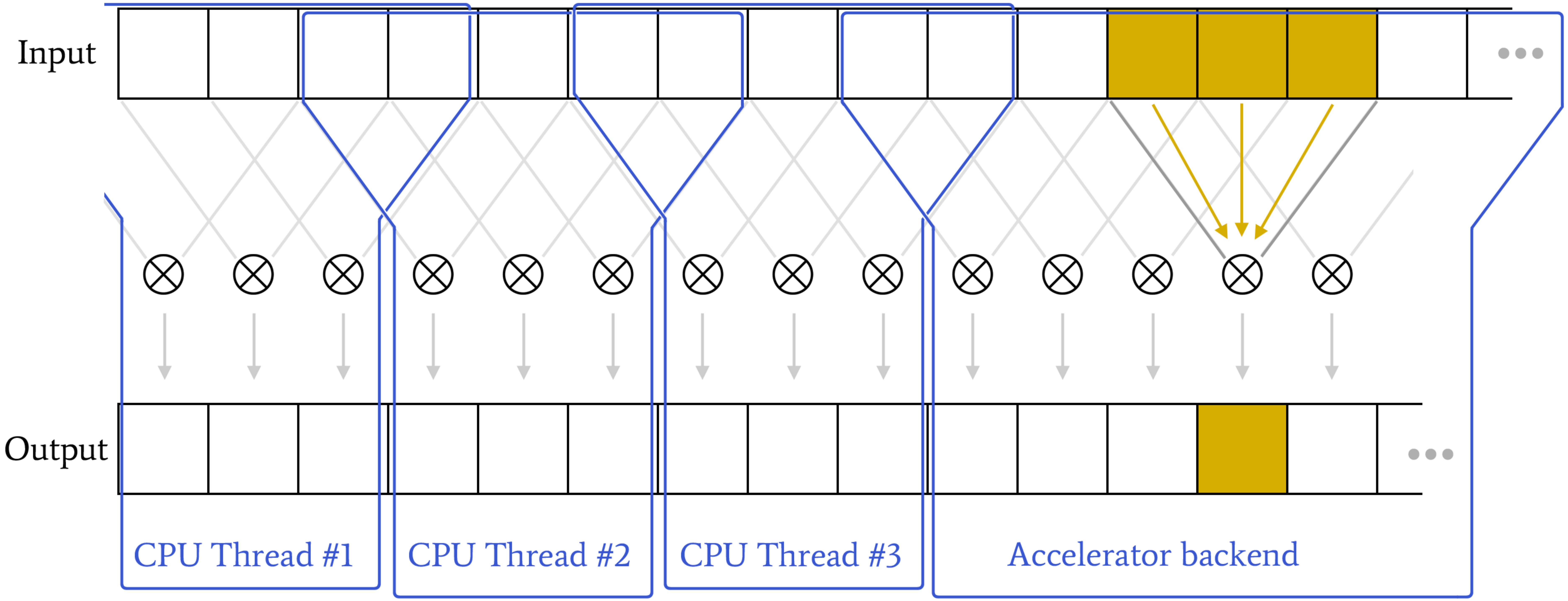
Hybrid backend – Work partitioning

- Partitioning Map



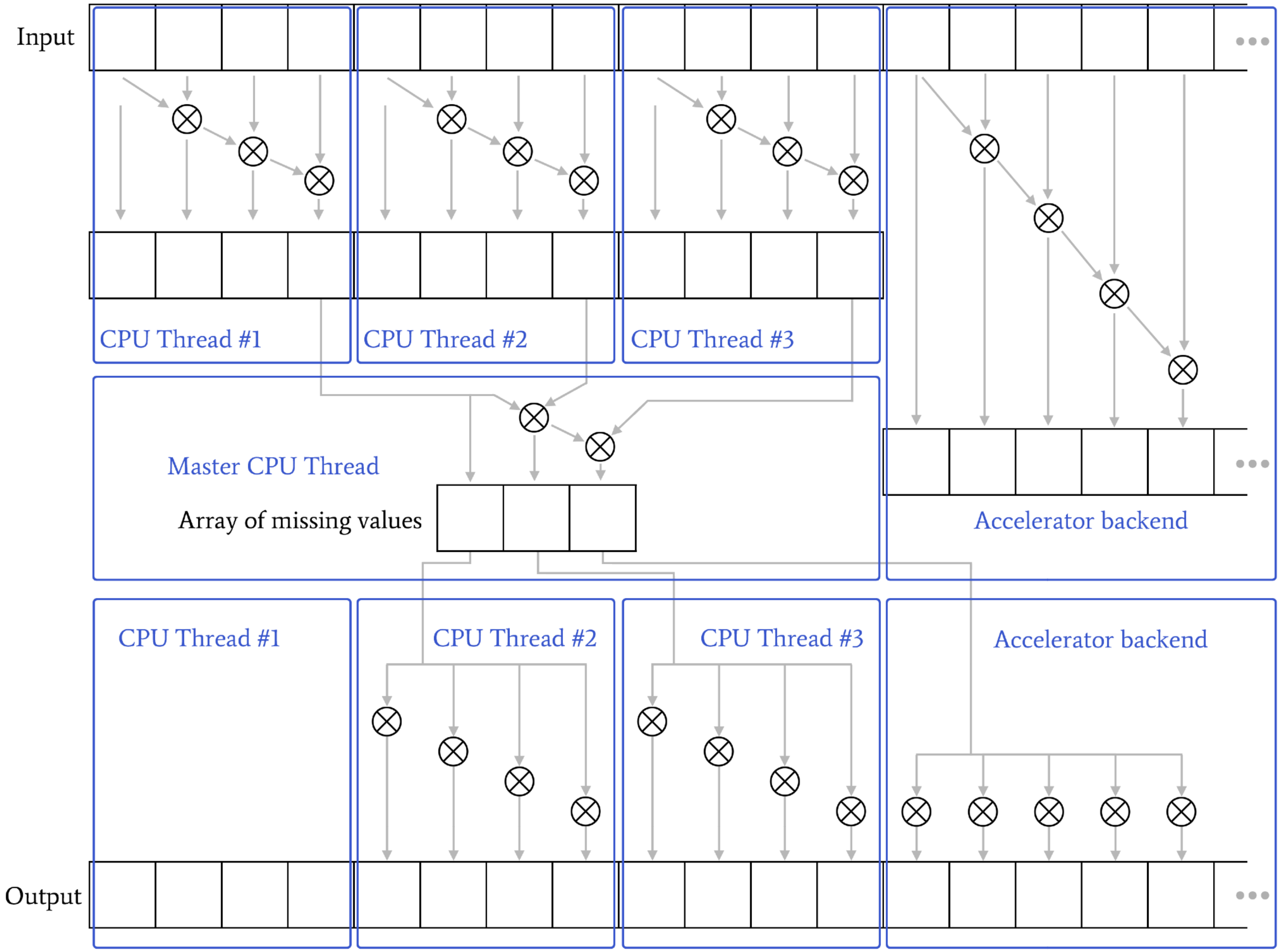
Hybrid backend – Work partitioning

- Partitioning MapOverlap



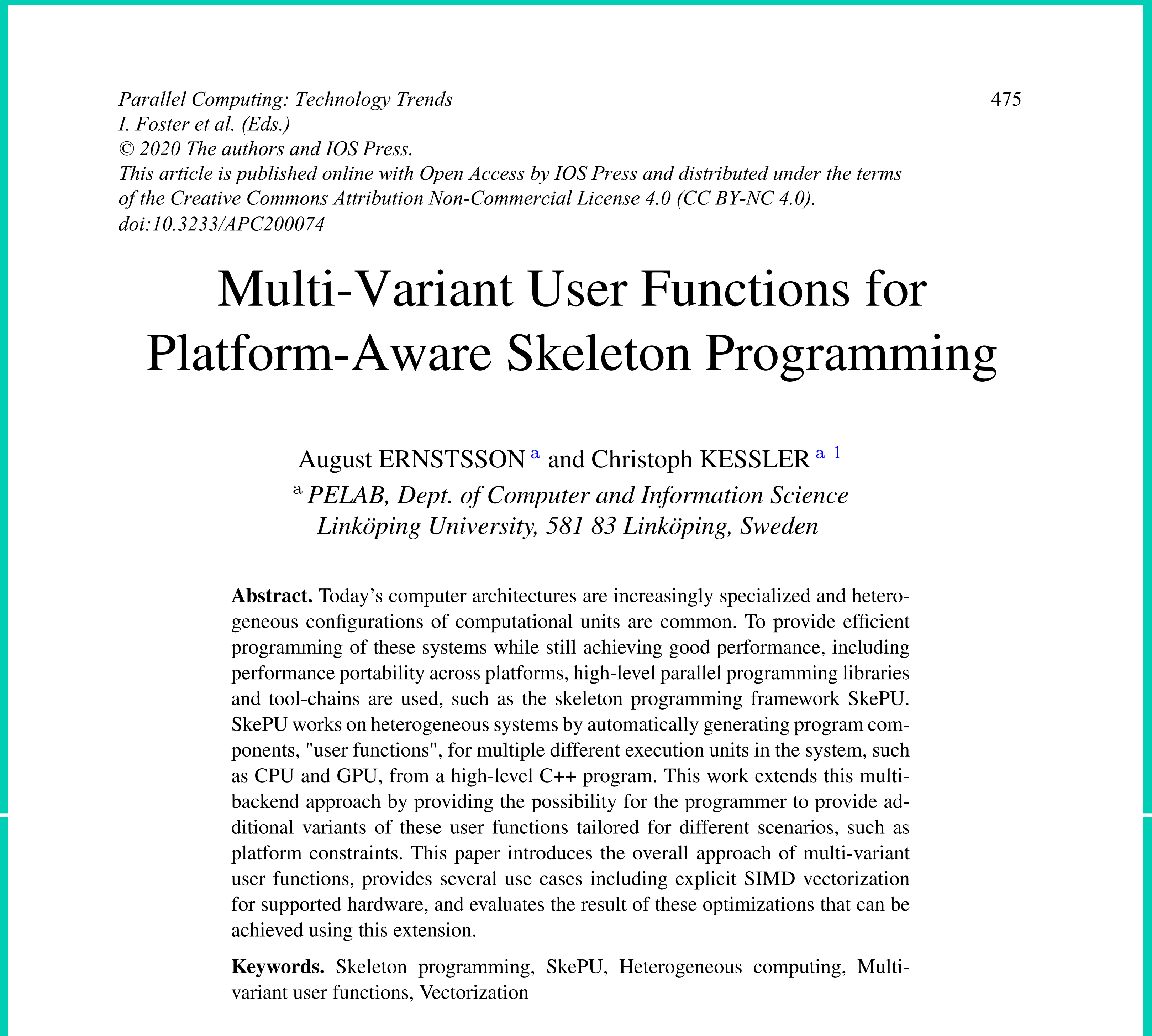
Hybrid backend – Work partitioning

- Partitioning Scan



Contribution

Multi-variant user functions



Parallel Computing: Technology Trends 475
I. Foster et al. (Eds.)

doi:10.3233/APC200074

Multi-Variant User Functions for Platform-Aware Skeleton Programming

August ERNSTSSON [a] and Christoph KESSLER [a] [1]
[a] *PELAB, Dept. of Computer and Information Science*
Linköping University, 581 83 Linköping, Sweden

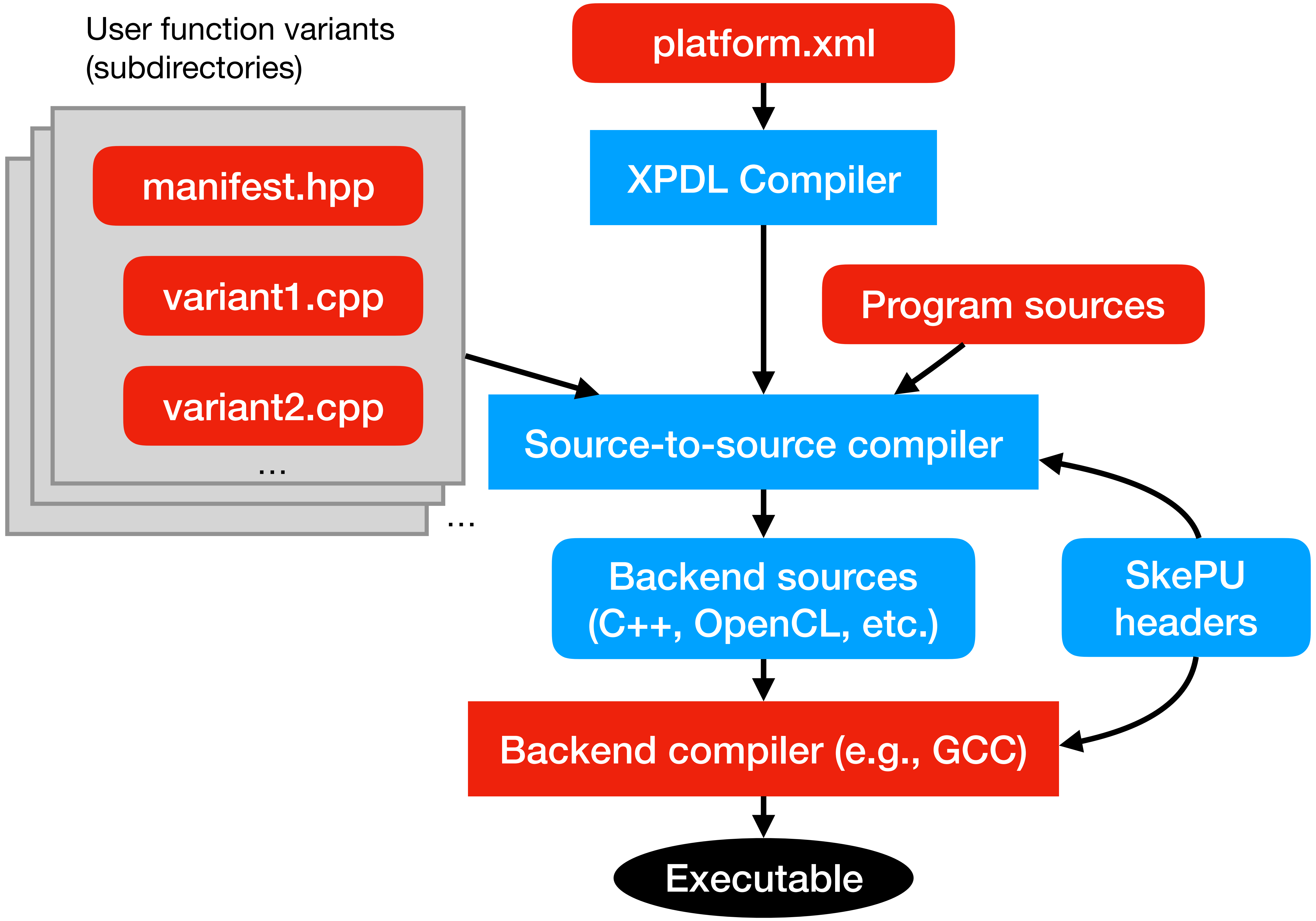
Abstract. Today's computer architectures are increasingly specialized and heterogeneous configurations of computational units are common. To provide efficient programming of these systems while still achieving good performance, including performance portability across platforms, high-level parallel programming 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



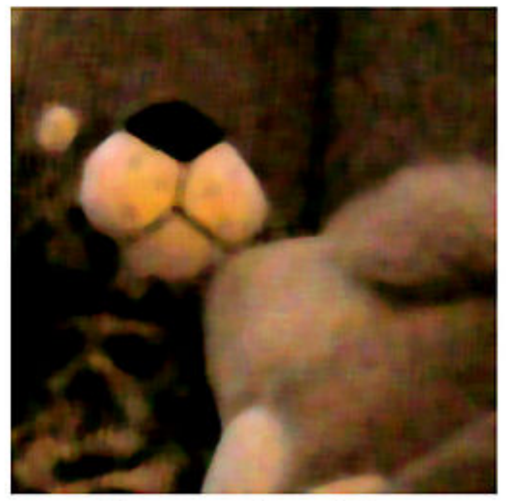
Multi-variant user functions – Evaluation

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

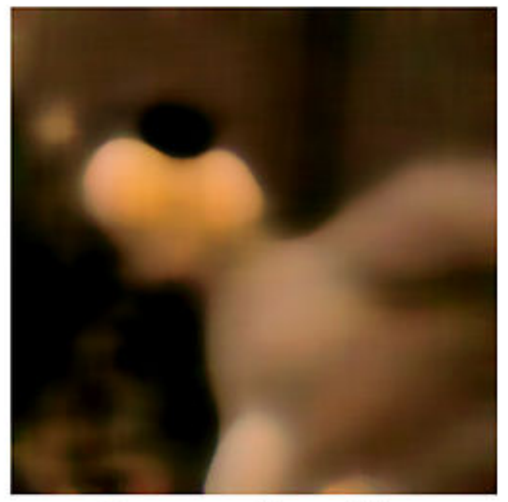
Variant	Time complexity	Memory complexity	Dependencies
Double loop	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	None
Histogram	$\mathcal{O}(n + \|D\|)$	$\mathcal{O}(\|D\|)$	None
`qsort`	$\mathcal{O}(n \log n)$	$\mathcal{O}(n)$	C standard library

original image

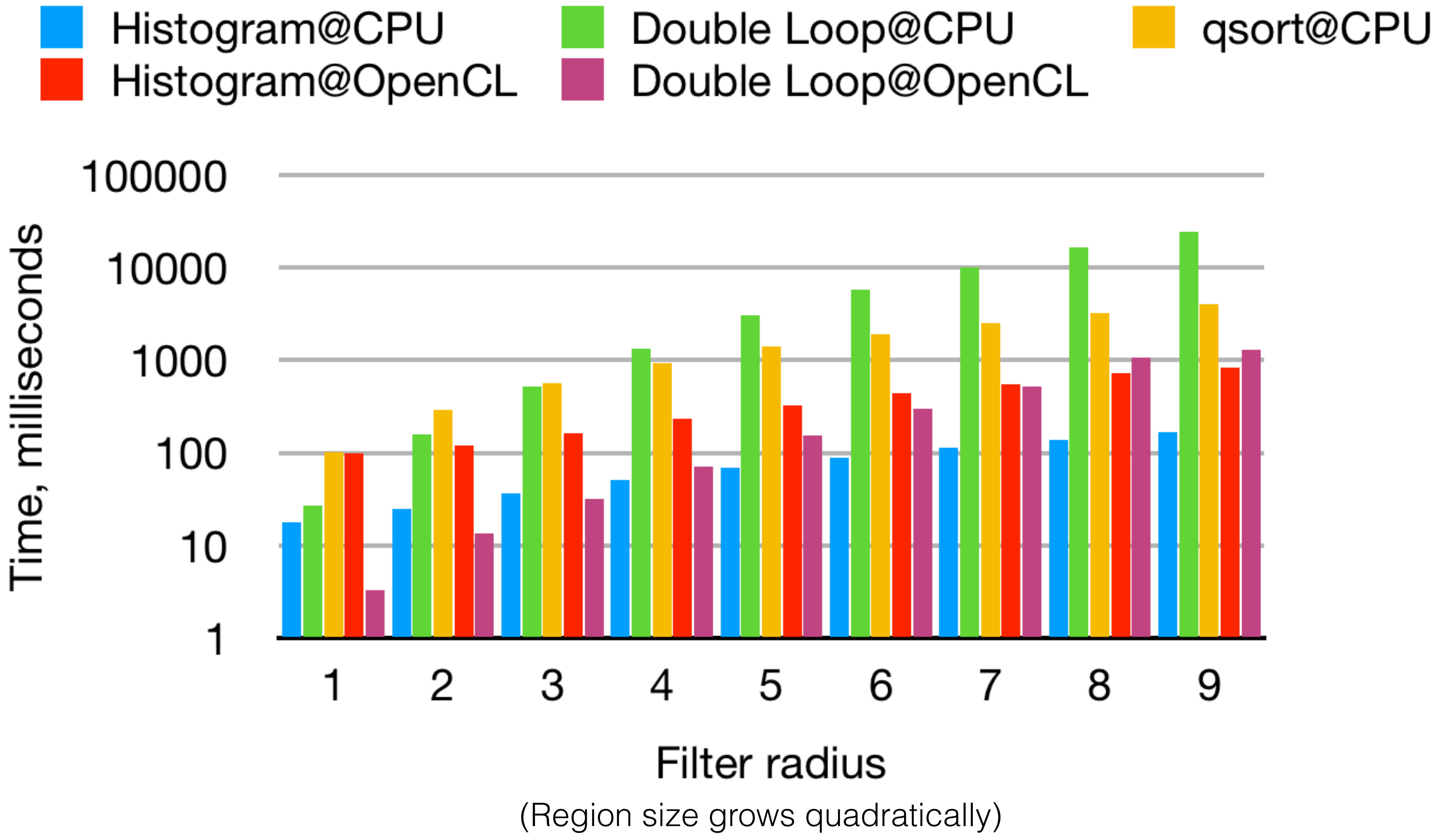
1px median filter



3px median filter



10px median filter



2016 SkePU 2 → 2017 Lazy eval with tiling → 2018 Hybrid backend → 2019 Multi-variant UF → **2020 SkePU 3 + cluster**

Contribution

SkePU 3 with new skeletons and cluster backend

Portable exploitation of parallel and heterogeneous HPC architectures in neural simulation using SkePU

Sotirios Panagiotou, National Technical University of Athens, Greece, spanagiotou@microlab.ntua.gr
August Ernstsson, Linköping University, Sweden, august.ernstsson@liu.se
Johan Ahlqvist, Linköping University, Sweden, johan.ahlqvist@liu.se
Lazaros Papadopoulos, National Technical University of Athens, Greece, lpapadop@microlab.ntua.gr
Christoph Kessler, Linköping University, Sweden, christoph.kessler@liu.se
Dimitrios Soudris, National Technical University of Athens, Greece, dsoudris@microlab.ntua.gr

ABSTRACT

The complexity of modern HPC systems requires the use of new tools that support advanced programming models and offer portability and programmability of parallel and heterogeneous architectures. In this work we evaluate the use of SkePU framework in an HPC application from the neural computing domain. We demonstrate the successful deployment of the application based on SkePU using multiple back-ends (OpenMP, OpenCL and MPI) and present lessons-learned towards future extensions of the SkePU framework.

KEYWORDS

accelerator backend. Tools that assist application developers in the process of exposing parallelization and exploiting accelerators by reducing the required programming effort are highly desirable. SkePU[1] [10] falls in this category. It is an open-source skeleton programming framework for multicore CPUs and multi-GPU systems. (Algorithmic) *skeletons* [6] are generic parallelizable high-level programming constructs based on higher-order functions such as Map, Reduce, Stencil or Scan, which model common dependence and data access patterns and which can be parameterized in problem-specific sequential code. Skeletons provide a high degree of abstraction and portability with a quasi-sequential programming interface, as their

Noname manuscript No.
(will be inserted by the editor)

SkePU 3: Portable High-Level Programming of Heterogeneous Systems and HPC Clusters

August Ernstsson · Johan Ahlqvist · Stavroula Zouzoula · Christoph Kessler

Received: date / Accepted: date

Abstract We present the third generation of the C++ based open-source skeleton programming framework SkePU. Its main new features include new skeletons, new data container types, support for returning multiple objects from skeleton instances and user functions, support for specifying alternative platform-specific user functions to exploit e.g. custom SIMD instructions, generalized scheduling variants for the multicore CPU backends, and a new

LINKÖPING UNIVERSITY

SkePU 3

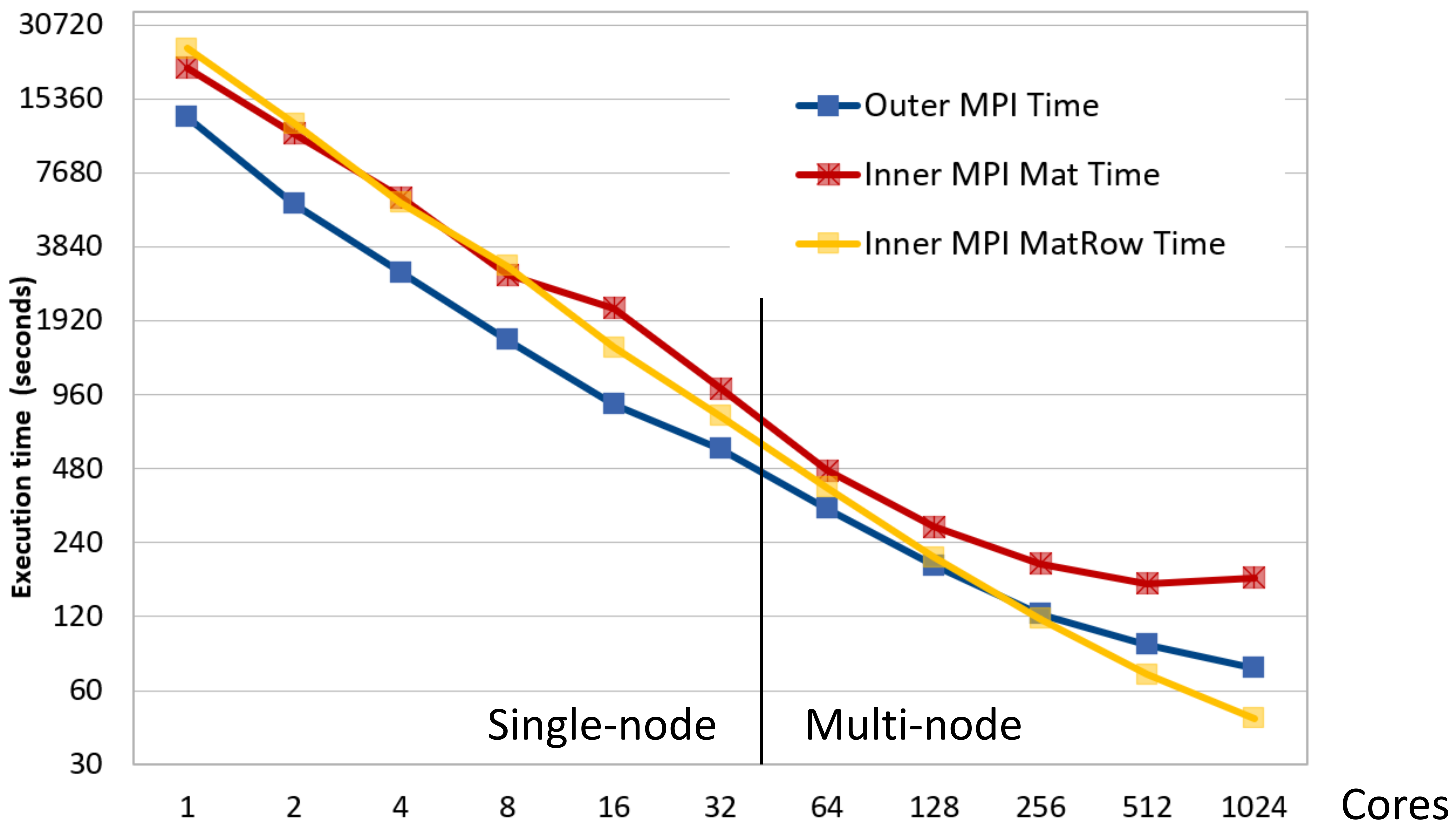
- Collaborations within the *EXA2PRO* research project
- *Application-framework co-design*
  - SkePU framework team working with application partners
- Cluster backend added for *exascale* computations
- Real-world applications being ported to SkePU
- Improved distribution and compatibility

SkePU 3 – New features sample

- Cluster backend
  StarPU + MPI
- Tensors
- MapPairs
- MapPairsReduce
- MatCol and MatRow
- Tuple return syntax
- Dynamic scheduling on multi-core
- C and Fortran wrappers
- Simplified memory consistency model
- New MapOverlap syntax
- 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.

LINKÖPING UNIVERSITY