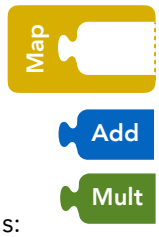
SkePU

Extending Smart Containers for Data Locality-Aware Skeleton Programming

August Ernstsson
Linköping University, Sweden

Christoph Kessler
Linköping University, Sweden

Skeleton Programming with SkePU



- **High-level** parallel programming paradigm
- **Skeletons** are higher-order functions with efficient parallel implementations
- **SkePU**: C++11 framework with Map, Reduce, MapReduce, MapOverlap, and Scan skeletons
- Skeletons are **parameterized** with user functions: such as **add**, **mult**, or more complex computations
- For heterogeneous systems: **multi-backend** (CPU+GPU)
- **Smart containers** wrap operand data and handle data management and minimize data transfers with lazy copying

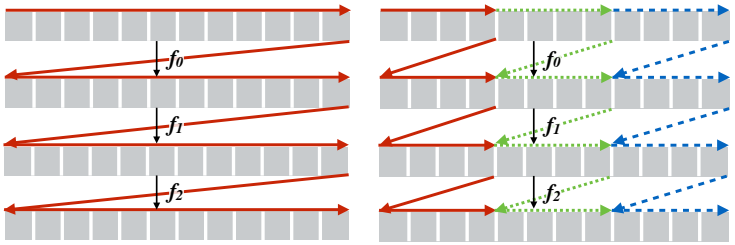
Example Program

```
Vector<float> v1, v2, v3, v4, v5, v6, v7, v8, v9;
auto add = Map<2>([](float a, float b){ return a+b;});

add(v1, v3, v4);  // transformation using Map instance
copy(v9, v1);
mult(v2, v1, v3); // copy, mult, square, and reduce
square(v1, v2);   //    are also skeleton instances
add(v5, v5, v1);
add(v5, v5, v9);
add(v6, v7, generate(v6, 5.f));
for (int i = 0; i < 5; i++)
       add(v8, v8, v8);
reduce(v8); // action point, causes evaluation
```

Goal: Tiling of Map Skeletons

- With each Map skeleton invocation handled in isolation, the access patterns are not ideal for **cache performance**
- Example: a sequence of three Maps with f_0, f_1, f_2:

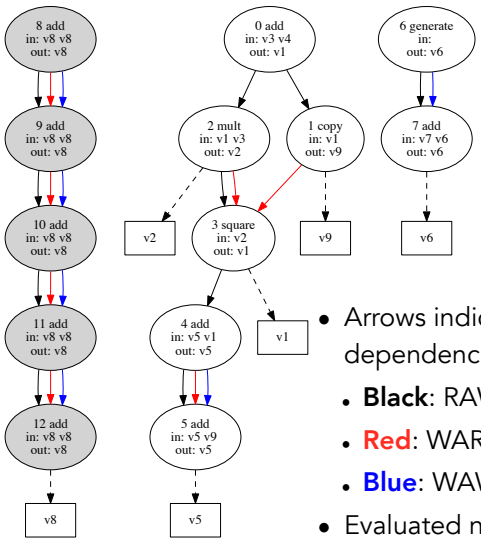


Access pattern without tiling *Access pattern with tiling*

Approach: Lazy Evaluation and Skeleton Lineages

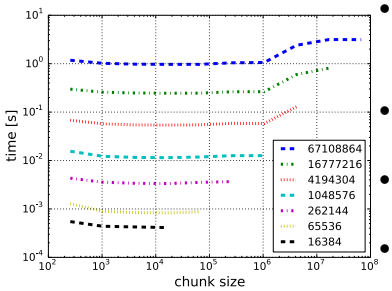
- For **data locality optimization**: consider sequences of skeleton invocations to enable tiling across entire sequence
- Inspiration from the **Spark** framework for big data analytics
  - Locality is even more important on Hadoop clusters where the access times are much higher
  - Solved with lazy evaluation: a **lineage** (DAG) of **transformations** on data is built up until an **action** is required
  - Spark containers are single-assignment, unlike SkePU
- In SkePU, Maps are "transformations" and other operations are "actions", extensible to other skeletons as well
- Lineages give **run-time** information of actual program flow (dynamic rather than static analysis)
- Once an action is required on a container, the lineage is traversed backwards, following dependencies, and its nodes are evaluated starting from the roots
  - Skeleton calls may be evaluated globally **out-of-order**, but still in-order w.r.t. data dependencies

Produced Lineage Graph



- Leaf nodes indicate starting points for evaluating a smart container.
- Internal nodes are skeleton invocations, corresponding to a call in the program.
- Arrows indicate data dependencies:
  - **Black**: RAW dependency
  - **Red**: WAR dependency
  - **Blue**: WAW dependency
- Evaluated nodes are marked gray

Performance



- Microbenchmark: iterative sequence of Map invocations (repeated squaring)
- Varying container size and tiling chunk size
- Chunk size approaching data size causes behaviour as if no tiling
- Chunk size possible to expose as tuning parameter

Selected SkePU Publications

- A. Ernstsson, C. Kessler: Extending Smart Containers for Data Locality-Aware Skeleton Programming. *Presented at HLPP 2017*, Valladolid, July 2017
- A. Ernstsson, L. Li, C. Kessler: SkePU 2: Flexible and Type-Safe Skeleton Programming for Heterogeneous Parallel Systems. *Int. J. of Parallel Programming*, 2017
- U. Dastgeer and C. Kessler. Smart Containers and Skeleton Programming for GPU-based Systems. *Int. J. of Parallel Programming* 44(3):506-530, June 2016

LINKÖPING UNIVERSITY

SkePU is open source (GPLv3)
www.ida.liu.se/labs/pelab/skepu/

Swedish e-Science Research Centre