Version 2

SkePU

Flexible, Type-Safe, Auto-Tunable, Multi-Backend Skeleton Programming Framework for Multicore CPU and Multi-GPU Systems

August Ernstsson
Linköping University, Sweden

Lu Li
Linköping University, Sweden

Christoph Kessler
Linköping University, Sweden

Skeleton Programming

- **High-level** parallel programming paradigm
- Inspired by higher-order functions from **functional programming**
- Skeletons are reusable **components** which may have efficient parallel implementations
- Skeletons represent **computational patterns** (control and data flow)
  - E.g., data-parallel map, reduce, stencil, or scan
- Skeletons **encapsulate** parallelism and memory management
- Skeletons are configured with **user**-defined **functions**

User Functions

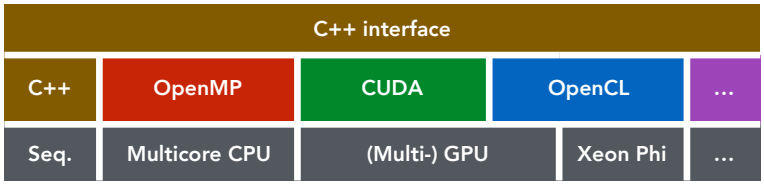
- User-provided C++ functions or function templates
- Defined as **free functions** or C++11 **lambdas**
- **Variadic** parameter arity in three aspects:
  - **Element-wise access** container operands
  - **Random access** container operands (unrestricted read/write)
  - **Uniform** scalar operands (i.e., ordinary C++ parameter)

Available Skeletons

Map	Data-parallel application of user function
Reduce	Reduction with 1D and 2D variations
MapReduce	Efficient combination of Map and Reduce
MapOverlap	Stencil operation in 1D and 2D
Scan	Generalized prefix sum
Call	Generic multi-variant component

Backends

- SkePU supports a set of **heterogeneous** backends
- **Single source** supports all backends
  - Sequential CPU, multicore CPU, (multi-) GPU, Xeon Phi
- Auto-tuning backend **selection** targeting **time** or **energy**
  - **Execution plan** computed by machine learning
- Other experimental backends, e.g., clusters, embedded
  - StarPU backend for task parallelism and hybrid execution



Example Code: Sum of Squares

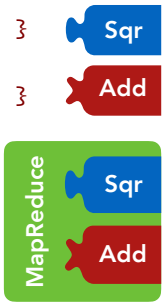
```
// User function for mapping
float sqr(float a)          { return a * a; }

// User function for          reduction
float add(float a, float b) { return a + b; }

// Instantiate skeleton
auto sumsq = MapReduce<1>(sqr, add);

// Define          smart vector container
Vector<float>   vec(N);

// Invoke on smart vector
float res = sumsq(vec);
```

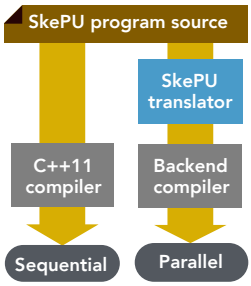


Smart Containers

- Smart containers are **STL-like** backend-aware data structures
  - **Vector**, **matrix**, and **sparse matrix** formats
- C++ class templates parameterizable by **custom structs**
- Using **software caching** between host and device
  - Reuse of device memory allocations
  - Device-to-device transfer optimizations

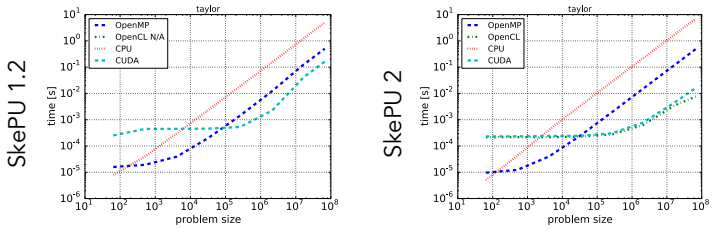
Source-to-Source Translation

- SkePU uses **Clang** as a translation tool
- Translator generates platform-specific code for OpenCL, CUDA, OpenMP
- Translator knows the **semantics** of SkePU skeletons and containers
- Programs are **valid C++11** and run sequentially without precompilation



Performance

- Flexible skeleton set allows for optimization of algorithms
- E.g. Taylor series expansion with smaller memory footprint:



Selected Publications

- A. Ernstsson, L. Li, C. Kessler: SkePU 2: Flexible and Type-Safe Skeleton Programming for Heterogeneous Parallel Systems. *Int. J. of Parallel Programming*, to appear, 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
- U. Dastgeer, J. Enmyren, C. Kessler: Auto-tuning SkePU: A Multi- Backend Skeleton Programming Framework for Multi-GPU Systems. Proc. IWMSE-2011, 2011, ACM.

LINKÖPING UNIVERSITY

SEVENTH FRAMEWORK PROGRAMME

EXCESS

Open Source (GPLv3)
www.ida.liu.se/labs/pelab/skepu/

CUGS

SeRC Swedish e-Science Research Centre