SkePU

Multi-variant User Functions for Platform-aware Skeleton Programming

August Ernstsson
Linköping University, Sweden

Christoph Kessler
Linköping University, Sweden

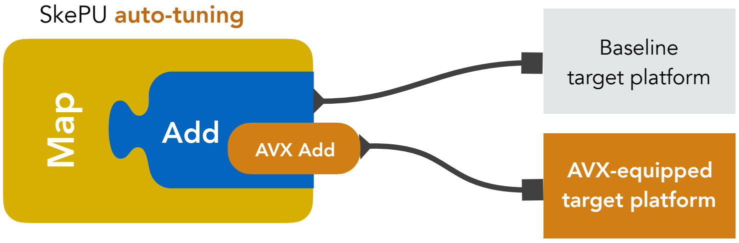
Skeleton Programming

- **High-level** parallel programming paradigm
- Skeletons are reusable **components** which may have efficient parallel implementations
- Skeletons **encapsulate** parallelism and memory management
- Represent **computational patterns** (control and data flow) such as:

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

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)
- **Multi-variant user functions** for targeting specific platforms
  - Multiple elements per user function enabling optimizations
  - Multiple variants for each user function, **selectable** directly or with SkePU **auto-tuning**



XPDL Platform Description

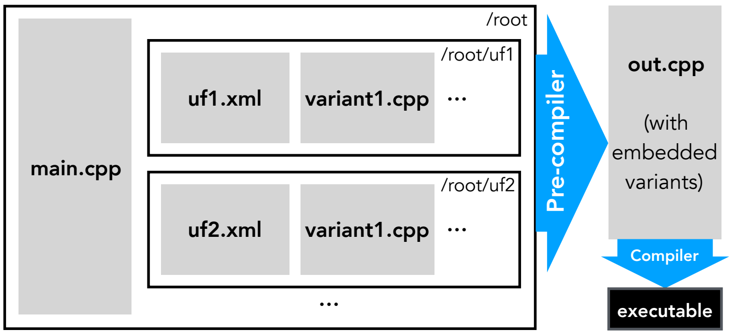
- XPDL model for Intel Xeon multi-core system with AVX instructions

```
<?xml version="1.0" encoding="UTF-8"?>
<xpdl:model xmlns:xpdl="http://www.xpdl.com/xpdl_cpu"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.xpdl.com/xpdl_cpu xpdl_cpu.xsd ">
   <xpdl:component type="cpu" />
   <xpdl:cpu name="Intel_Xeon_Gold_6130" num_of_cores="16"
      num_of_threads="32" isa_extensions="avx avx2">
   <xpdl:group prefix="core_group" quantity="16">
      <xpdl:core frequency="2.1" unit="GHz" />
      <xpdl:cache name="L1" size="32" unit="KiB" set="16" />
      <xpdl:cache name="L2" size="1" unit="MiB" set="16" />
   </xpdl:group>
   <xpdl:cache name="L3" size="22" unit="MiB" set="1" />
   <xpdl:power_model type="power_model_Gold_6130" />
   </xpdl:cpu>
</xpdl:model>
```

AVX-equipped target platform

Tool Flow

- Directory-driven **variant lookup**, one directory per user function, one file per variant
- SkePU precompiler **enables** variants and assembles program



Example: Vectorized Addition

main.cpp

```
// Main user function definition
float add(float a, float b) { return a + b; }
```

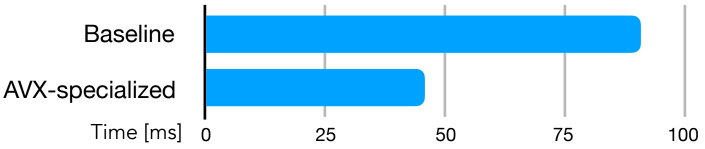
add/variant.cpp

```
// Specialized variant of add
// for platforms with AVX instructions
#pragma skepu vectorize 8
void add(float* c, const float *a,
                   const float *b)
{
      __m256 av = _mm256_load_ps(a);
      __m256 bv = _mm256_load_ps(b);
      __m256 cv = _mm256_add_ps(av, bv);
      _mm256_store_ps(c, cv);
}
```

AVX-specialized user function variant

Performance

- Early experimental performance evaluation shows over 2x speedup with the selectable vectorized user-function variant



Selected SkePU Publications

A. Ernstsson, L. Li, C. Kessler. SkePU 2: Flexible and Type-Safe Skeleton Programming for Heterogeneous Parallel Systems. *Int J Parallel Prog.* (2018) 46: 62

A. Ernstsson and C. Kessler. Extending smart containers for data locality–aware skeleton programming. *Concurrency Computat Pract Exper.* (2019) 31:e5003.

T. Öhberg, A. Ernstsson, C. Kessler. Hybrid CPU–GPU execution support in the skeleton programming framework SkePU. J Supercomput (2019). To appear.

LINKÖPING UNIVERSITY

exa2pro

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

CUGS