### 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

<table>
<thead>
<tr>
<th>Skeleton</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Map</strong></td>
<td>Data-parallel application of user function</td>
</tr>
<tr>
<td><strong>Reduce</strong></td>
<td>Reduction with 1D and 2D variations</td>
</tr>
<tr>
<td><strong>MapReduce</strong></td>
<td>Efficient combination of Map and Reduce</td>
</tr>
<tr>
<td><strong>MapOverlap</strong></td>
<td>Stencil operation in 1D and 2D</td>
</tr>
<tr>
<td><strong>Scan</strong></td>
<td>Generalized prefix sum</td>
</tr>
<tr>
<td><strong>Call</strong></td>
<td>Generic multi-variant component</td>
</tr>
</tbody>
</table>

### 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

### 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