SkePU
Auto-tunable Multi-Backend Skeleton Programming Library for Multi-GPU Systems

Usman Dastgeer  Johan Enmyren  Christoph Kessler
Linköping University, Sweden

Multi-GPU Systems

Challenges
- Portability
- Programmability
- Performance portability

OpenCL™? Portable but low-level...

Solution: Skeleton Programming
Skeletons are pre-defined, reusable, parameterized generic components with well defined semantics, for which efficient parallel or accelerator-specific implementations may be available.

- Higher-order functions
- for frequent algorithmic (control and data flow) patterns e.g. map, reduce, scan, farm, pipe, DC, ...
- encapsulate all parallelism and platform-specific implementation details
- parameterized in sequential user code, programmer interface is sequential

SkePU Example: Dot product

```
#include <iostream>
#include "skepu/vector.h"
#include "skepu/mapreduce.h"

int main()
{
  skepu::MapReduce<mult, plus>
    .dotProduct(new mult, new plus);
  skepu::Vector<double> v0(1000,2);
  skepu::Vector<double> v1(1000,2);
  double r = dotProduct(v0,v1);
  std::cout << "Result: " << r <<"\n";
  return 0;
}
```

SkePU
- C++ template library, open-source
- 6 data-parallel skeletons
  - map, reduce, scan, mapreduce, maparray, mapoverlap
  - 1 task-parallel skeleton: farm
- Smart containers for user data: Vector, Matrix, ...
- Generation of platform-specific user functions
- Back-ends for C, OpenMP, OpenCL, CUDA, StarPU
- Multi-GPU support
- Overhead below 10%
- Tunable

Offline-tunable selection
- Expected best back-end + tunable parameters: #threads, #thread blocks, tiling factor, ...
- Execution plan for each skeleton, generated from off-line training data by machine learning

On-line tunable selection
- Using the history-guided selection in StarPU

Hybrid execution
- CPUs + GPUs in parallel

Selected publications

Download: www.ida.liu.se/~chrke/skepu