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 code...

Solution: Skeleton Programming
Skeletons are pre-defined, reusable, parameterizable 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 ...
- Parameterized in sequential user code, programmer interface is sequential
- Encapsulate all parallelism and platform-specific implementation details
- Well suited for internal autotuning
- Enforce well structured code

SkePU Example: Dot product

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

BINARY_FUNC(plus, double, a, b; return a+b;
) BINARY_FUNC(mult, double, a, b; return a*b;
)

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

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 tuneable selection
- Using the history-guided selection in StarPU

Hybrid execution
- CPUs + GPUs in parallel

Selected publications

Open Source:
www.ida.liu.se/~chrke/skepu