Tasking Meets GPUs
Fighting Deadlocks and Other Monsters

April 15, 2019 | L. Morgenstern, I. Kabadshow | Jülich Supercomputing Centre
Motivation

Task Parallelism Everywhere

- **Goal:**
  - Task-based programming model to exploit fine-grained task parallelism on heterogeneous hardware

- **Use case:**
  - Fast Multipole Method (FMM) for molecular dynamics (MD)
  - Computation of all $N^2$ pair-wise long-range interactions between $N$ particles in $O(N)$
  - Execution time below 1 ms per simulation step – no matter, how many particles to simulate :-)

- **Requirements:**
  - Strong scalability
  - Performance portability
Motivation

Use Case: Schematic Task Graph of FMM
GPU Tasking
Programming Model for GPUs
Pitfalls

Performance Portability

Diverse GPU programming approaches:
- OpenCL
- CUDA
- SYCL

Our requirements:
- Strong subset of C++11
- Portability between GPU vendors
- Tasking features
- Maturity

(Intermediate) Solution

Use CUDA for reasons of performance. Not portable out of the box.
Pitfalls

Reusability of CPU Tasking

- Goal: reuse as much of the CPU tasking as possible
- Problem: STL missing on GPUs
Pitfalls

Architectural Differences

Pitfalls for Load Balancing

- No thread pinning
- No cache coherency

Pitfalls for Mutual Exclusion

- Weak memory consistency
- Missing forward progress guarantees
Very First Evaluation

Conditions

- Tasking with global queue only
- Measurements without work load to determine enqueue and dequeue overhead
- Measurements on P100 with 56 thread blocks with 1024 threads each
- Measurements on V100 with 80 thread blocks with 1024 threads each
Very First Evaluation

Tasking Overhead on P100 and V100

#T asks
Runtime in ms

P100
V100

Graph showing the runtime in ms for tasks on P100 and V100.
Conclusion

- Fine-grained task parallelism pays off on CPUs
- Developed mapping between CPU and GPU concepts
- (Partly) overcome pitfalls:
  - Performance portability
  - Reusability of CPU tasking code
  - Architectural differences between CPU and GPU
- Successfully implemented proof-of-concept tasking on GPUs
Next Steps

- Analyze and solve performance issues in dependency resolution
- Use memory pool for dynamic allocations
- Implement hierarchical queues
  - Transfer priority queue to GPU
  - Exploit data-parallelism through warps
- Consider the use of lock-free data structures
- Implement FMM based on GPU tasking