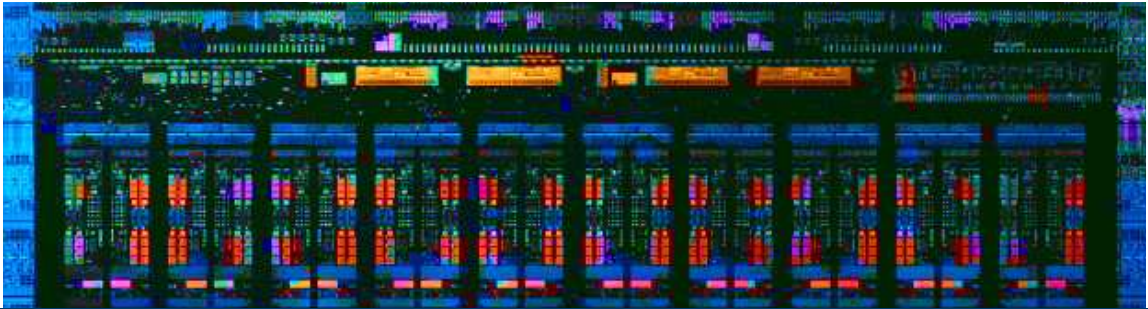


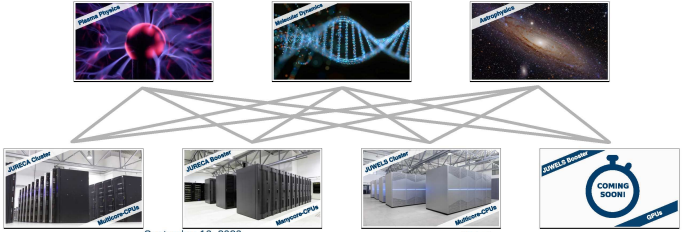
Tasking Meets GPUs
A Closer Look at Locking

September 10, 2020 | L. Morgenstern, M. Zych, I. Kabadshow | Jülich Supercomputing Centre

JÜLICH Forschungszentrum

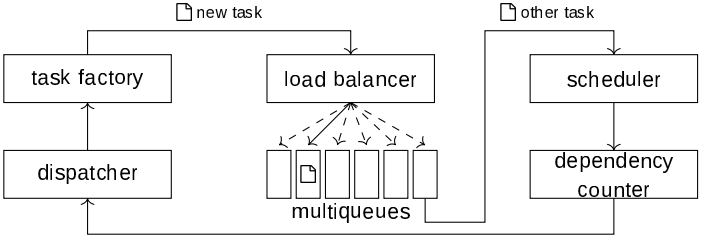
Motivation

- **Goal**:
  - Task-based programming model to exploit fine-grained task parallelism on heterogeneous hardware
- **Use case**:
  - FMSolvr: C++ implementation of Fast Multipole Method for molecular dynamics
  - Based on Eventify, a task-parallel programming library for applications with many, tiny, dependent tasks
- **Idea**:
  - Extend Eventify to GPUs
  - Provide an API that allows algorithm developers to write one function per task type that runs on host and device side



Eventify on CPUs in a Nutshell

- Input: data-flow graph → translated into task graph at compile time
- Follow critical path: each thread owns a type-driven priority queue
- Tasks are inserted into queue as soon as ready-to-execute
- Static load balancer determines task-thread mapping
- Dynamic load balancing via work stealing
- ℹ For more information on Eventify, see Proceedings of PASC20



Eventify Meets GPUs

- Employ a persistent thread kernel for each task type T
- Each kernel produces and consumes tasks of its type T
- Only one kernel launch per task type

```
produce_and_consume<T>()
{
  while(!finished())
  {
    block_master:
      mutex.lock();
    syncthreads();
    block:
      queue.front(threadIdx.x).execute();
    syncthreads();
    block_master:
      solveDependencies();
      mutex.unlock();
    syncthreads();
  }
}
```

Eventify Meets GPUs

First Performance Optimizations

- Multiple queues instead of a single, shared queue
- Hierarchical Queues
- Use shared memory for task objects

ℹ For more information on GPU Eventify, see our talk at GTC Digital 2020
https://developer.nvidia.com/gtc/2020/video/s21189

Eventify Meets GPUs

Experiment with Locking Mechanisms

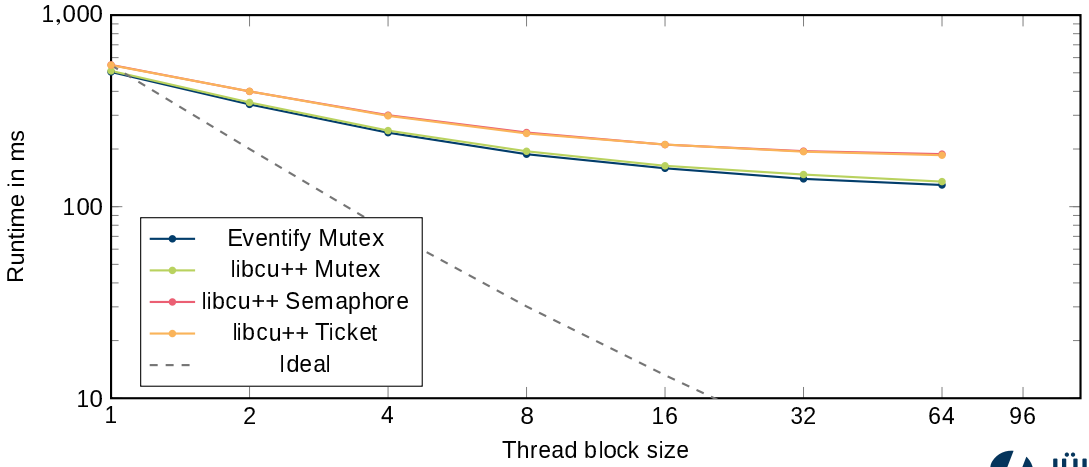
- Eventify Mutex: spin-lock via atomicCAS(), atomicExch() and memory fencing
- Locking mechanisms provided by lib *freestanding* based on libcu++
  - mutex
  - sem_mutex
  - ticket_mutex

ℹ For more information on lib *freestanding*, see https://github.com/ogiroux/freestanding

ℹ For more information on libcu++, see talk of B. Adelstein Lelbach at GTC Digital 2019
https://on-demand.gputechconf.com/supercomputing/2019/pdf/sc1942-the-cuda-c++-standard-library.pdf

Locking Mechanisms

Performance Analysis: Multipole Order p=0, Depth d=6, Grid Size g=320 on NVIDIA Tesla V100



Next Steps

- Finalize merge of FMSolvr and GPU-Eventify
- Further investigate performance difference of locking mechanisms
- Consider the use of lock-free data structures
- Message passing for multi-GPU tasking
- Message passing for tasking on heterogeneous CPU/GPU architectures
- Eventify/FMSolvr on FPGAs

Questions or Comments?

Please feel free to contact us!

Ivo Kabadshow

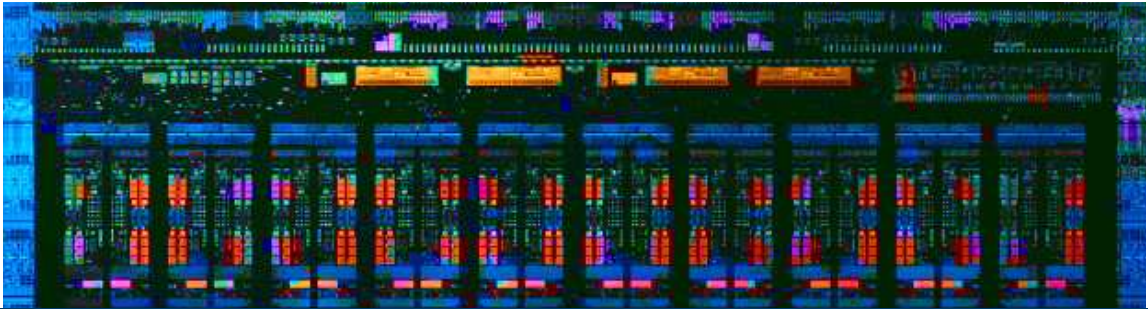
- CPU Tasking
- Fast Multipole Method

i.kabadshow@fz-juelich.de

Laura Morgenstern

- GPU Tasking
- GPU Performance Analysis

l.morgenstern@fz-juelich.de



Tasking Meets GPUs
A Closer Look at Locking

September 10, 2020 | L. Morgenstern, M. Zych, I. Kabadshow | Jülich Supercomputing Centre

JÜLICH Forschungszentrum