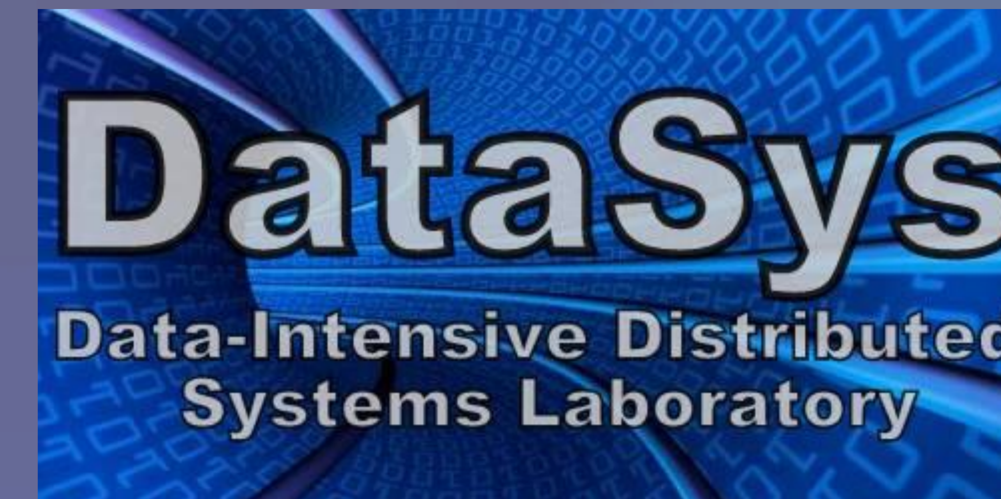
ILLINOIS INSTITUTE OF TECHNOLOGY

Early Experiences in running Many-Task Computing workloads on GPGPUs



Scott J. Krieder
Dept. of Computer Science
Illinois Institute of Technology
skrieder@iit.edu

Benjamin Grimmer
Dept. of Computer Science
Illinois Institute of Technology
bgrimmer@hawk.iit.edu

Dr. Ioan Raicu
Dept. of Computer Science
Illinois Institute of Technology
iraicu@cs.iit.edu

Many-Task Computing

Many-task computing (MTC) aims to bridge the gap between two computing paradigms, high throughput computing (HTC) and high-performance computing (HPC). MTC emphasizes using many computing resources over short periods of time to accomplish many computational tasks (i.e. including both dependent and independent tasks), where the primary metrics are measured in seconds. MTC denotes high-performance computations comprising multiple distinct activities.

Swift Parallel Programming

Swift is a particular implementation of the MTC paradigm, and is a parallel programming system that has been successfully used in many large-scale computing applications across the TeraGrid and now XSEDE. It has been adopted by the scientific community as a great way to increase productivity in running complex applications via a dataflow driven programming model, which intrinsically allows implicit parallelism to be harnessed based on data access patterns and dependencies.

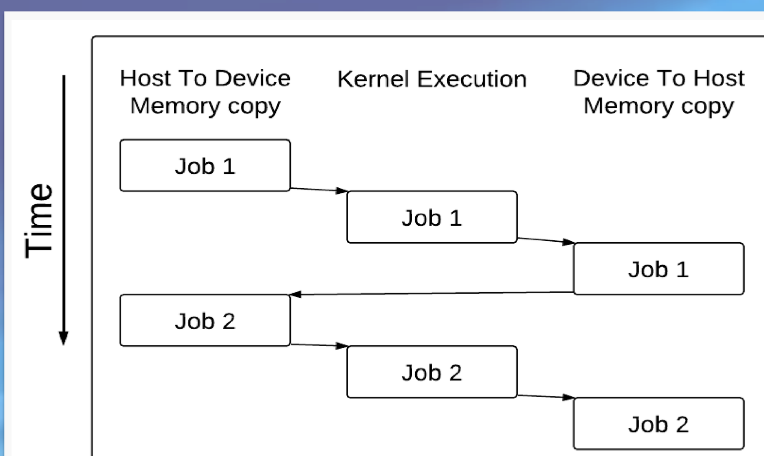
Proposed Work

This work aims to enable Swift to efficiently use accelerators (such as NVIDIA GPUs) to further accelerate a wide range of applications. This work presents preliminary results in the costs associated with managing and launching concurrent kernels on NVIDIA Kepler GPUs. We expect our results to be applicable to several XSEDE resources, such as Forge, Keeneland, and Lonestar, where currently Swift can only use the general processors to execute workloads and the GPUs are left idle.

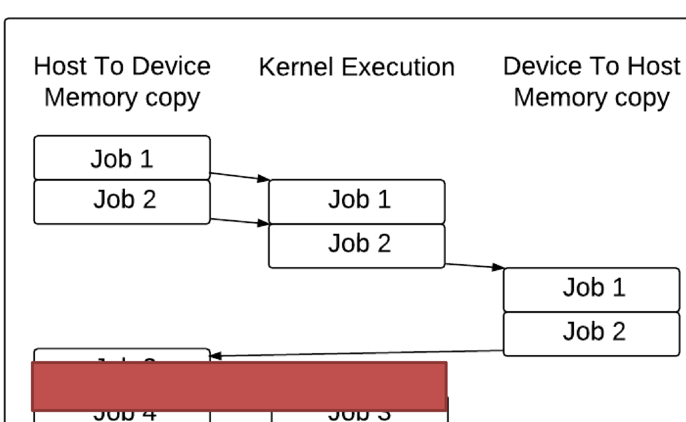
References

Swift - http://www.ci.uchicago.edu/swift/main/
NVIDIA - nvidia.com/object/cuda_home_new.html

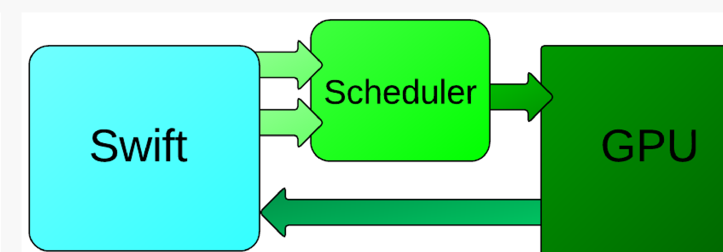
Scheduler Architecture



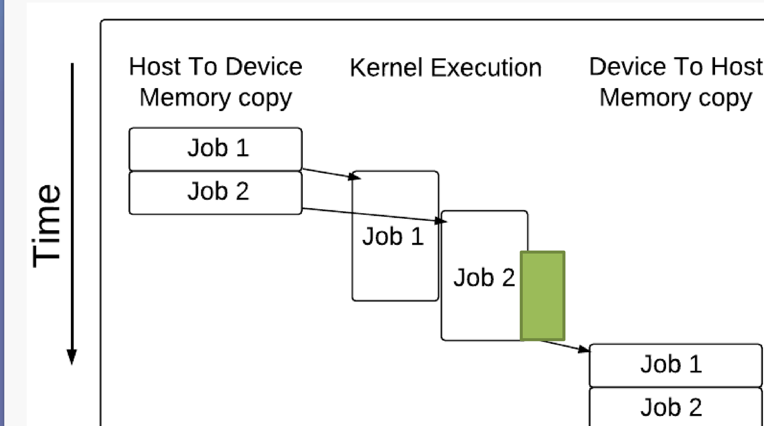
Without a scheduler, jobs run on the GPU following the pattern copy-compute-copy.



Copy-compute-copy produces inefficiencies. Some of which are shown in the red area above.

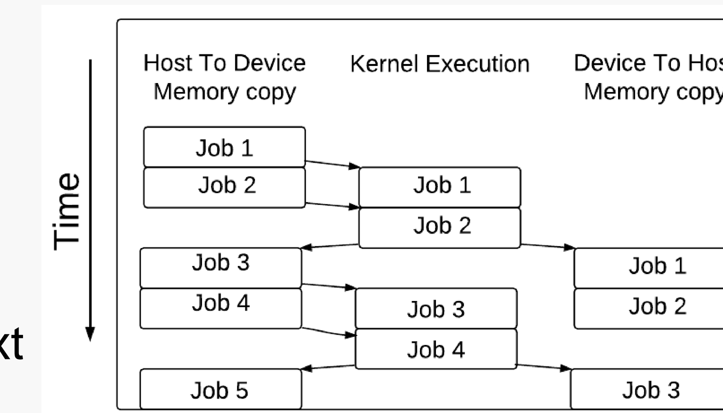


Our scheduler sits in-between Swift and GPU. Handles multiple inputs from Swift and condenses these into single GPU calls.

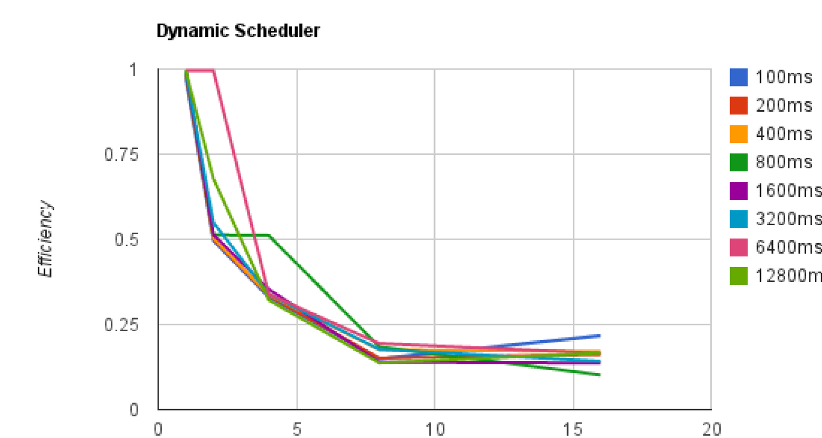


Use of a scheduler allows for overlapping kernel execution.

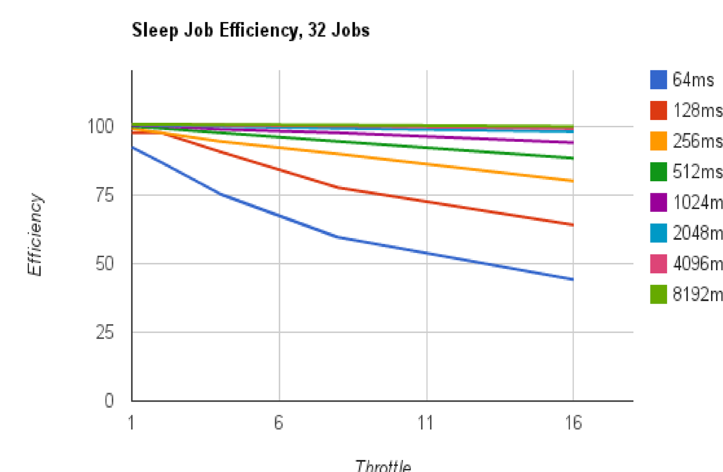
Our scheduler overlaps data transfers from last solution and next problem to increase efficiency.



Results

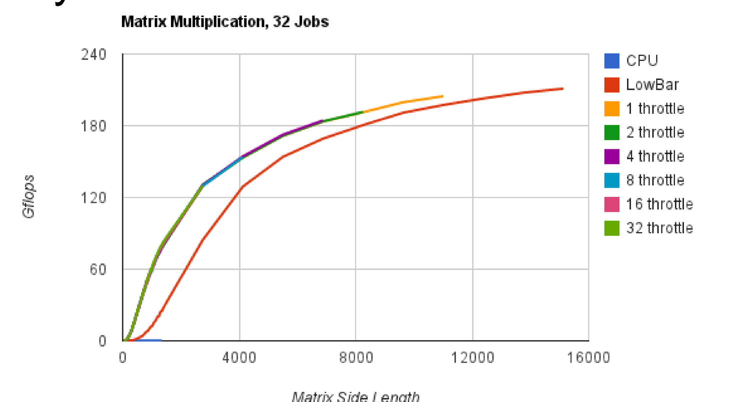


Initial implementation of Dynamic scheduler approaches serialized due to race condition when threads synchronize with streams.

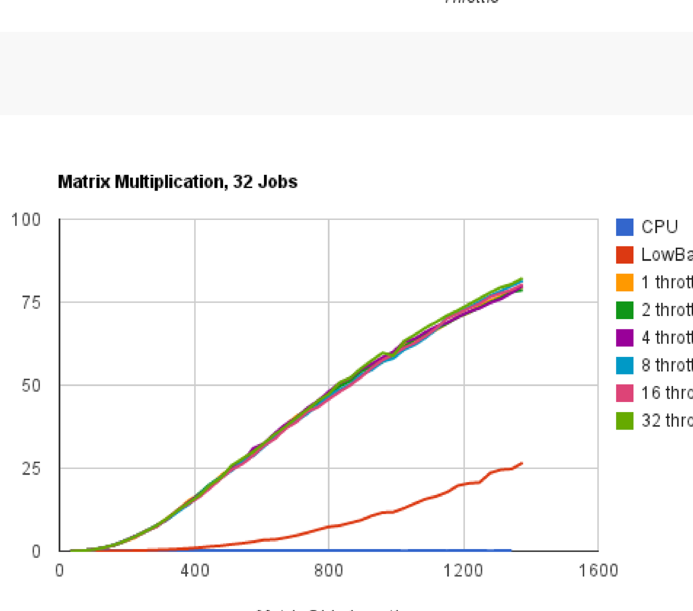


Our current implementation is a static batch FIFO scheduler.

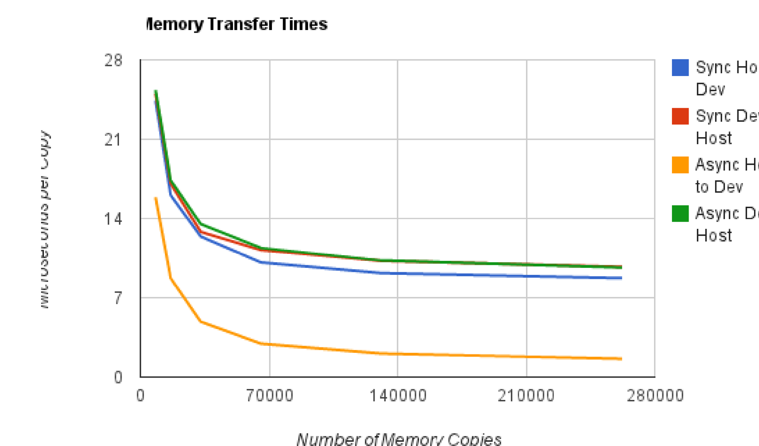
Sleep jobs demonstrate our schedulers ability to overlap kernels.



Matrix Multiplication demonstrates our ability to overlap data transfers for increased efficiency.



A closer look at the most improved portion of the curve highlighting benefit gained from overlapping data transfers.



Finally, with PCI-e 2 we are able to complete copies in just a few microseconds. This leads us to believe a SuperKernel managed MTC scheduler will not bottleneck in regards to the number of copies.

Accelerators & Coprocessors

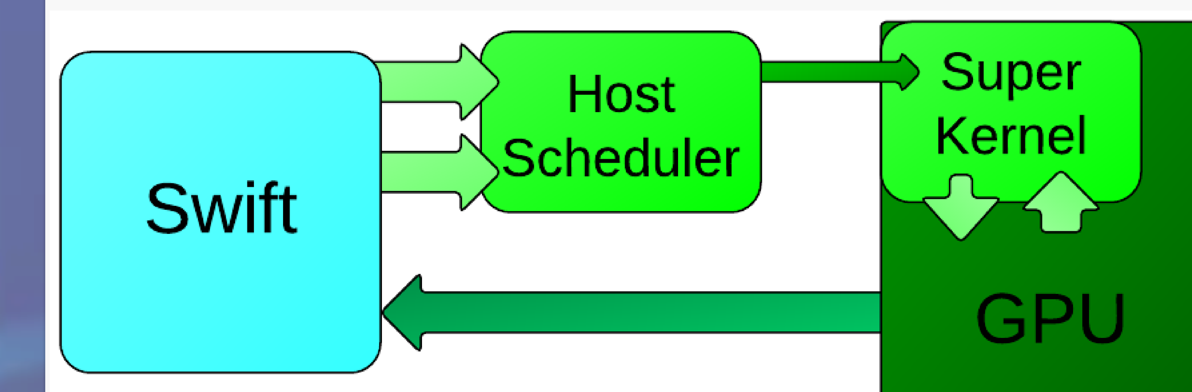
Device	Pro	Con
NVIDIA Kepler	Maturity Raw Perf.	Programming
Intel MIC	Programming	Availability
AMD GPU	Openness	Adoption

Conclusions

- Scheduler provides concurrent kernel execution from built in library of CUDA kernels
- Scheduler overlaps memory transfers for increased performance
- Concurrent kernels limited to up to 16 kernels
- Kernel execution overheads limit workloads to coarse granularity kernels

Future Work

- Migrate scheduler into GPU (SuperKernel manages MicroKernels)



- Integrate GPGPU MTC Scheduler into Swift
- Explore Intel MIC as a co-processor to run MTC workloads