A Hybrid Task Graph Scheduler for High Performance Image Processing Workflows

Timothy Blattner, Walid Keyrouz, Shuvra Bhattacharyya, Milton Halem, and Mary Brady
Dep't of Computer Science and Electrical Engineering, University of Maryland, Baltimore County (UMBC)
National Institute of Standards and Technology (NIST)

ABSTRACT

Having applications take advantage of the capabilities available in hybrid and cluster computing is key to improving their performance. Scheduling code to utilize parallelism is difficult, particularly when dealing with data dependencies, memory management, data motion, and processor occupancy. The Hybrid Task Graph Scheduler (HTGS) increases programmer productivity when implementing hybrid workflows that scale to multi-core and multi-GPU systems. HTGS manages dependencies between tasks, represents CPU and GPU memories independently, overlaps computations with disk I/O and memory transfers, keeps multiple GPUs occupied, and uses all available compute resources.

We present an implementation of hybrid microscopy image stitching using HTGS that reduces code size by ~25% and shows favorable performance compared to a similar hybrid workflow implementation without HTGS. The HTGS-based implementation reuses the computational functions of the hybrid workflow implementation.

INTRODUCTION

Hybrid clusters now play a prominent role in high performance computing; they make up five of the top ten fastest supercomputers as of Nov 2015. [1] These petascale clusters consist of nodes that contain one or more CPUs with one or more co-processors (Intel Xeon Phi/NVIDIA Tesla). Clusters are approaching the exascale level; the next generation of hybrid architectures will contain fat cores coupled with many thin cores/accelerators on a single chip, as seen on Intel's Knights Landing. Programming these exascale machines for performance will be challenging. This will require new methods to emphasize minimizing data movement and maximize the number of computations done on that data.[2]

APPLICATION - IMAGE STITCHING

Image Stitching algorithm:

(S1) The fast Fourier transform (FFT) of an image,

(S2) The phase correlation image alignment method (PCIAM) that acts on two neighboring images' FFTs

(S3) The cross correlation factors (CCFs) between two neighboring images focused around a maximum intensity point identified from the PCIAM.

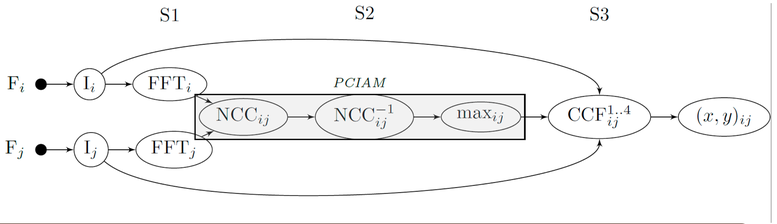
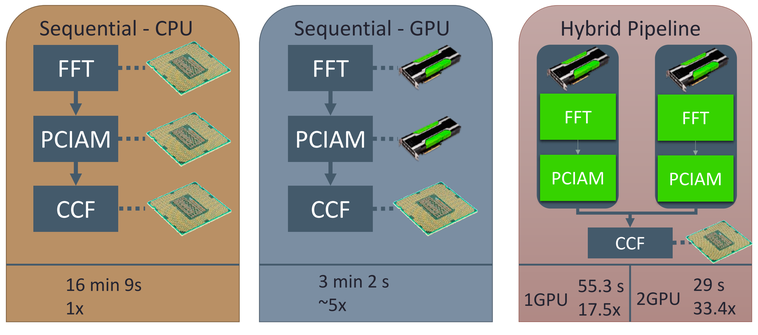


IMAGE STITCHING HYBRID PIPELINE WORKFLOW



HYBRID TASK GRAPH SCHEDULER

The Hybrid task graph scheduler (HTGS) aids in implementing hybrid pipeline workflows

Task graph - a series of vertices and edges, where the vertex is a task, and an edge is data flow between tasks.

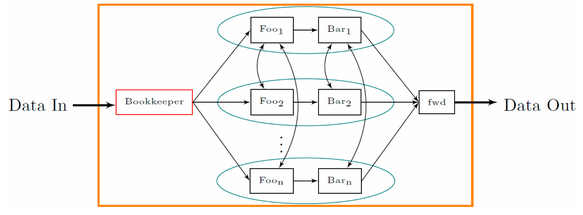
Task – implements a user-defined function applied on data, each tasks have input and output data

Example tasks in HTGS: Memory bookkeeping (memory management), Bookkeeper (Dependencies), GPUTask (GPU computation), Execution Pipeline (scaling)

Data – used by tasks and contains any number of elements required by the task implementation

EXECUTION PIPELINE

Execution pipeline is a special type of task that encapsulates task graphs and duplicates them. The execution pipeline can distribute data to each of its task graphs through rules (domain decomposition). Each pipeline executes concurrently.



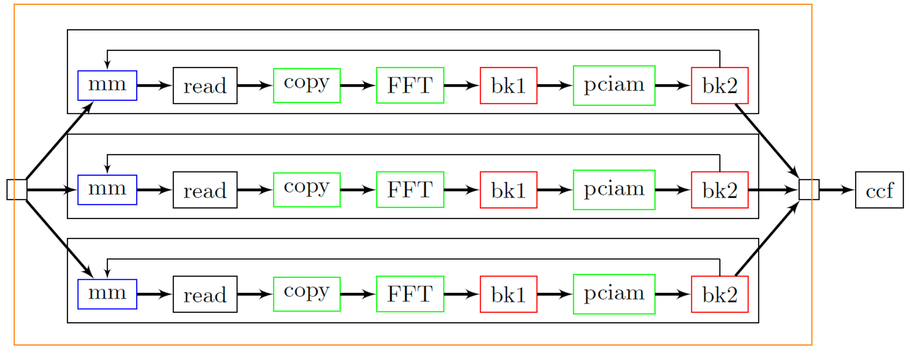
HTGS COMPONENTS

There are only four components that a programmer needs to define a task graph:

1. Data
2. Tasks
3. Dependency Rules
4. Memory Rules

Using these components a task graph can be created and edges between tasks can be made. If each of the tasks in a graph are implemented with attention to pipeline ids (rank), then a graph can be scaled using execution pipelines.

HTGS IMAGE STITCHING EXAMPLE



HYBRID IMAGE STITCHING IMPLEMENTATION

Task implementations:

- Read – Reads an image from disk
- FFT – Computes the forward fast Fourier transform on GPU
- PCIAM – Computes the phase correlation image alignment method on GPU
- CCF – Computes cross correlation on CPU

Data transport:

- FFT GPU Data – Stores image tile
- PCIAM GPU Data – Stores the origin tile, neighbor tile, and direction of displacement
- CCF Data – Stores array of possible displacements, origin tile, and neighbor tile

Dependency rules:

- Stitching Rule – Identifies neighboring images that both have their FFTs computed
- Stitching Memory Rule – Forwards memory data to the memory book keeper
- Stitching CCF Rule – Forwards CCF data to the CCF task

Memory allocations:

- FFT Memory Rule – Defines how FFT memory is used and released
- FFT Memory Allocator – Defines how FFT memory is allocated

HYBRID IMAGE STITCHING RESULTS (PROTOTYPE)

Machine specs: dual socket Xeon E5620 (8 total physical cores, 16 logical) + 2x Tesla C2070s + 1x GTX 680

Language: Java ; Dataset: 42x59 grid of tiles (~6.6 GB); Libraries: FFTW, JCuda, JCuFFT

Test Case	CPU Threads	Num GPUs	Runtime (sec)	Lines of Code
w/o API	16	3	24.9	985
w/o Pipeline w/ API	16	1	43.3	725
w/ Pipeline w/ API	16	1	41.4	726
w/ Pipeline w/ API	16	2	26.6	726
w/ Pipeline w/ API	16	3	24.5	726
CPU-only w/ API	16	0	141.0	479

CONCLUSIONS

This work represents an early prototype of the hybrid task graph scheduler. The results compared to the original implementation and HTGS using 3 GPUs show a 23.6% reduction in code size and similar runtime (24.5 s versus 24.9 s). Hybrid workflows are effective at parallelizing an algorithm, hiding data motion, and keeping processors busy. HTGS reduces the effort required to represent hybrid workflows in image stitching, while maintaining the performance of manually creating a hybrid workflow. HTGS also provides a framework for representing algorithms and tools for complex, data-intensive applications that require very high performance.

References:
[1] TOP500 super computer sites (Date last accessed: 2015-05-04). http://top500.org
[2] Ang. Et. Al. "Abstract machine models and proxy architectures for exascale computing" in Co-HPC '14. IEEE Press, 2014, pp 25-32