A Hybrid CPU-GPU System for Stitching Large Scale Optical Microscopy Images

Walid Keyrouz
NIST | ITL | SSD | ISG

National Institute of Standards and Technology
U.S. Department of Commerce

Disclaimer

- No approval or endorsement of any commercial product by NIST is intended or implied. Certain commercial software, products, and systems are identified in this report to facilitate better understanding. Such identification does not imply recommendations or endorsement by NIST nor does it imply that the software and products identified are necessarily the best available for the purpose.

Acknowledgments

- Tim Blattner
  - UMBC Ph.D. student & NIST
- Bertrand Stivalet, Joe Chalfoun, Mary Brady
  - NIST
- Anson Rosenthal
  - SURF student, Brown University
- Shujia Zhou
  - UMBC adjunct faculty
- Kiran Badhiraju
  - Univ. of Maryland, College Park
- Video games & special effects industries

Contents

Computing as an Enabler of Measurements

Goals & Objectives

Goals

- Computing as enabler & accelerator of metrology
  - Biology, Materials Science…
  - Third leg of Science
- Computationally steerable experiments

Themes

- Computational measurements
  - Augment physical measurements
- Performance
  - Computing-based measurements in "real-time"

Change Drivers

- Dramatic increases in computing power
  - Desktop supercomputer!
- Dramatic increases in storage
- Automated data collection

What $8K gets you (Sep 2011)?

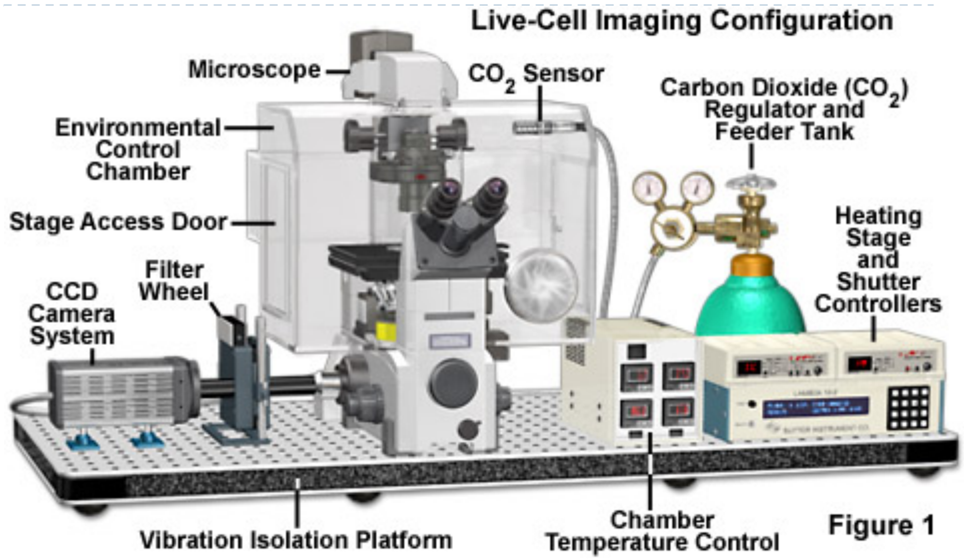
- 2 Intel quad-core Xeon E5620 2.4 GHz
  - 2 threads per core
  - Peak 38.4 GFLOPS
- 24 GB RAM
- 2 Nvidia Tesla C2070
  - 448 cores & 6 GB each
  - Peak 2 x 515 GFLOPS
- 1 Nvidia Quadro 4000
  - 256 cores
  - Peak 486.4 GFLOPS
- ~ 1.5 TFLOPS!

- June 2005 Top500:
  - R_{peak}: 1.28–183.5 TFLOPS
  - NEC SX-5/128M8 3.2ns

Desktop Supercomputer

Life Sciences

- High end microscopes:
  - Reaches $150–200K!
  - Nikon Live Cell Imaging station (URL)



- Prediction:
  - Will come with a desktop supercomputer.

Transformative Moment

- Current request:
  - Make my work easier.
- Strategic goal:
  - Change the way scientists work & discover!
- Approach:
  - Massive data & compute driven techniques.
    - Processing & data collection rates *= 100–1000.

Accelerators in HPC

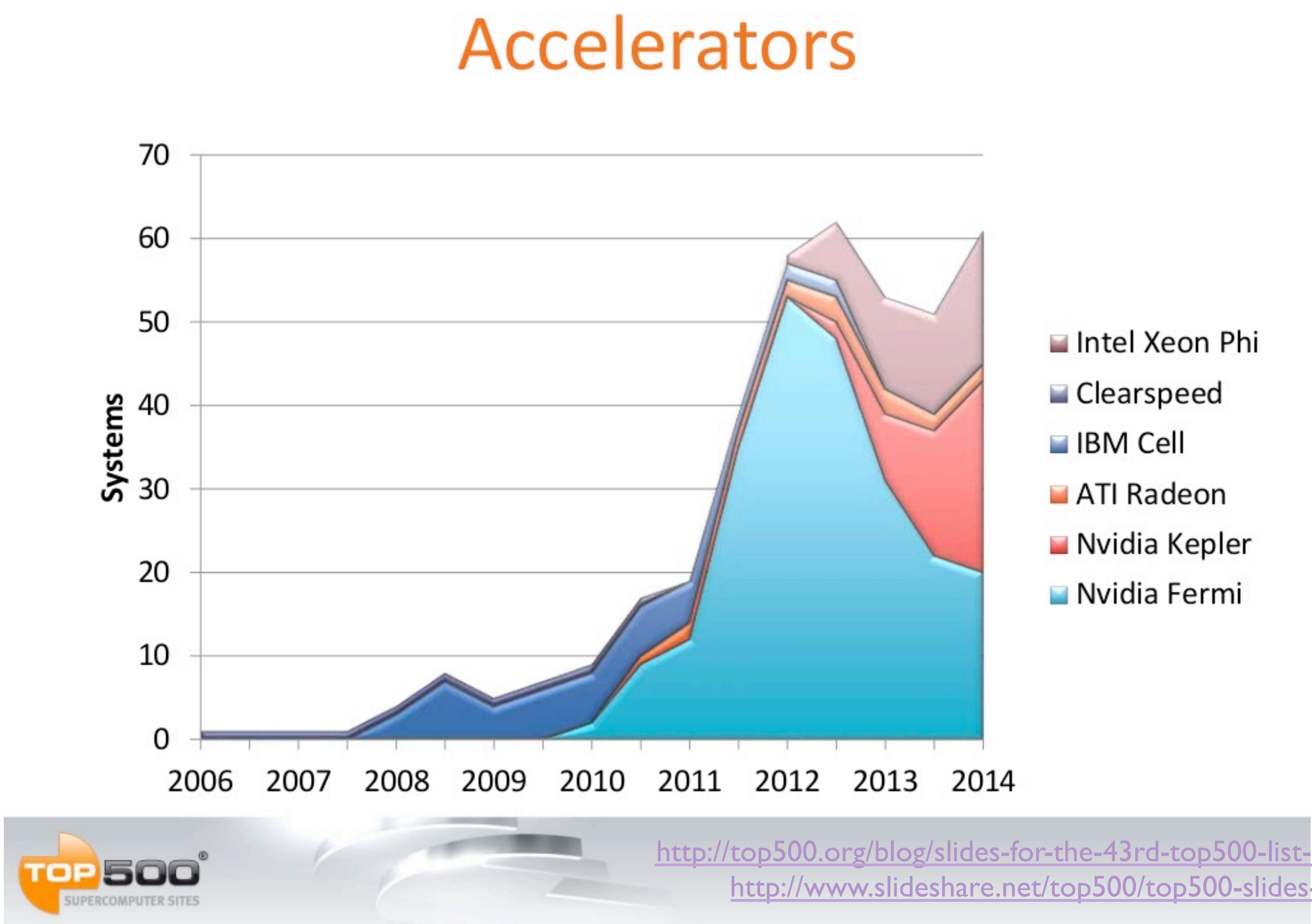
Top500 List (June 2014)

- Released twice a year (ISC & SC)

- 4 of Top 10 machines use accelerators
  - Intel Phi: Tianhe-2 (#1), Stampede (#7)
  - NVIDIA GPUs: Titan (#2), Piz Daint (#6)

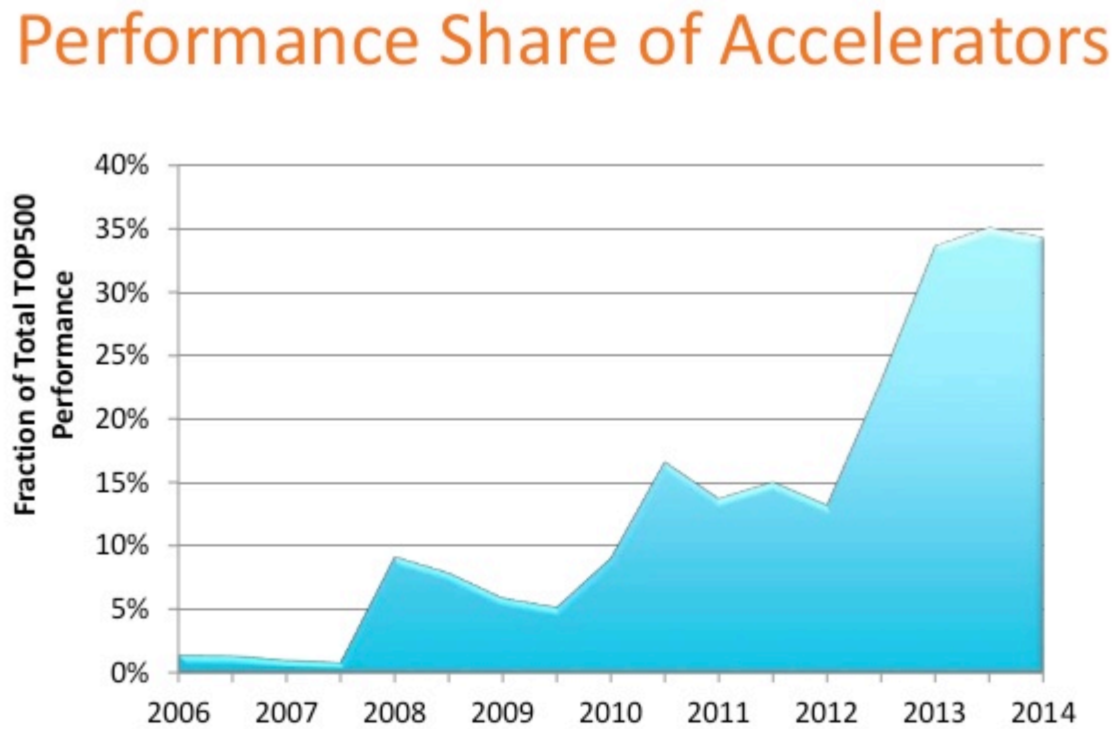
- 62 of Top 500 machines:
  - Nov 2013 edition had 53
  - NVIDIA GPUs: 44; ATI GPUs: 2
  - Intel Phi: 17

Top500 Accelerators (June 2014)



http://top500.org/blog/slides-for-the-43rd-top500-list-now-available/
http://www.slideshare.net/top500/top500-slides-for-june-2014

Top500 Accelerators (June 2014)



http://top500.org/blog/slides-for-the-43rd-top500-list-now-available/
http://www.slideshare.net/top500/top500-slides-for-june-2014

Recent Announcements (SC'14)

- DOE's Summit (ORNL) and Sierra (LLNL)
- IBM Power9 CPUs
- NVIDIA Volta-based Tesla GPUs
  - Architectures: Kepler, Maxwell, Pascal, Volta
- Mellanox EDR Infiniband
- Summit: 3400 nodes, 150--300 PFLOPS (~ 10MW)
  - Would appear on Top500 list with 4 nodes only!

NVIDIA Tesla K80

- Dual Tesla K40 card, $5K
  - NVIDIA Kepler GK210 architecture
- 2 x 2496 CUDA Cores
- Clock up to 562--875 MHz
- Memory:
  - 2 x 12 GB GDDR5
  - 2 x 384 GB/s internal bandwidth
- 300W Minimum System Power
  - Passive cooling
- Single precision: 5.6--8.75 TFLOPS
  - Double precision: 1.87--2.91 TFLOPS

Announced at SC'14

More Changes Coming

- AMD's APU
  - Combines CPU & GPU cores
- ARM's big.LITTLE
- Intel's Knights Landing
  - CPU + Xeon Phi cores on same die
  - Share memory bandwidth
- NVIDIA Pascal
  - NVLink, 3D stacked memory

Image Stitching for Optical Microscopy

Image Stitching for Optical Microscopy

Objectives

- Stitching of optical microscopy images at *interactive* rates
- General purpose library, ImageJ/Fiji plug-in, etc.
- Stitch & visualize plate between repeat experiments
  - Up to ~50 min to image a plate
  - Image plate every 1 hour

Success criterion

- Transformative impact
  - Run sample problem in < 1 min
  - > 10–100x speed improvement

Image Stitching Problem

- Optical microscopes scan a plate and take overlapping partial images (tiles)
- Need to assemble image tiles into one large image
  - Software tools
- Modern microscopy automated:
  - Scientists are acquiring & processing large sets of images

Starting Point

- ImageJ/Fiji
  - Open source Java
  - Multithreaded
- NIST prototype
  - MATLAB
- NIST data set:
  - 59x42 images (~ 7 GB)
- Hardware:
  - 2 Xeon quad-core CPUs
  - 48 GB
  - 2 NVIDIA Tesla 2070 GPUs

	ImageJ Fiji	NIST prototype
Old H/W		50 min
New H/W	~ 3.6 h	17 min

Motivation

- Proof of concept application
  - Illustrate benefits of desktop supercomputing
    - Multicore and GPU computing
- Heavy computing workloads

  > high-end desktop or small server

  < cluster

Data Set

- Grid of 59x42 images (2478)
- 1392x1040 16-bit grayscale images (2.8 MB per image)
  - ~ 7 GB

- Source:
  - Kiran Bhadriraju (NIST, Univ. of Maryland)

Evaluation Platform

Hardware

- Dual Intel® Xeon® E-5620 CPUs (quad-core, 2.4 GHz, hyper-threading)
- 48 GB RAM
- Dual NVIDIA® Tesla™ C2070 cards

Reference Implementations

- ImageJ/Fiji™ Stitching plugin, ~ 3.7 h
- MATLAB® prototype, ~17.5 min on a similar machine

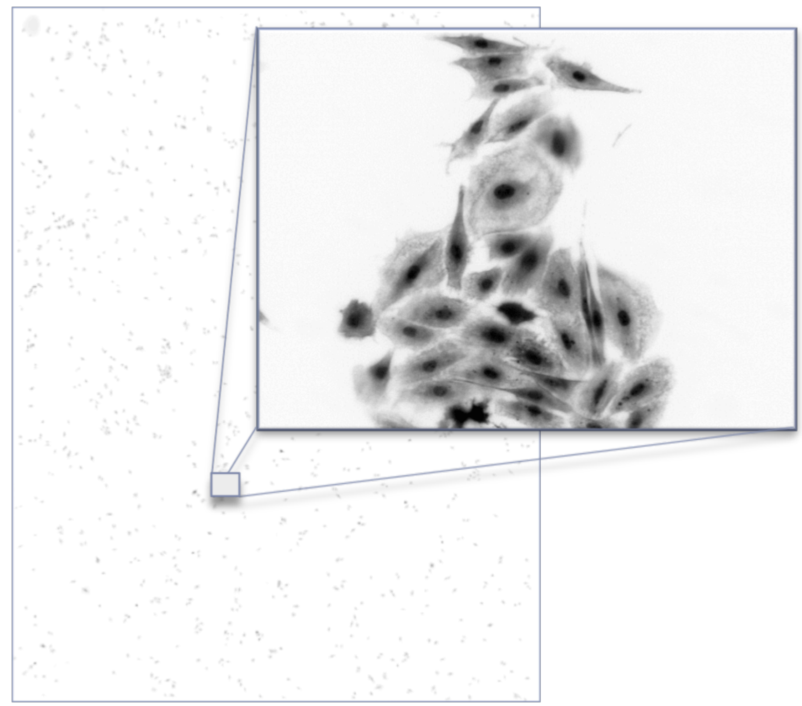
Software

- Ubuntu Linux 12.04/x86_64, kernel 3.2.0
- Libc6 2.1.5, libstd++6 4.6
- BOOST 1.48, FFTW 3.3, libTIFF4
- NVIDIA CUDA & CUFFT 5.0

Image Stitching Problem…

Three phases:

1. Compute the X & Y translations for all tiles
   - Produces over-constrained system
2. Remove over-constraint
   - Use global optimization techniques
3. Apply the translations & compose the stitched image

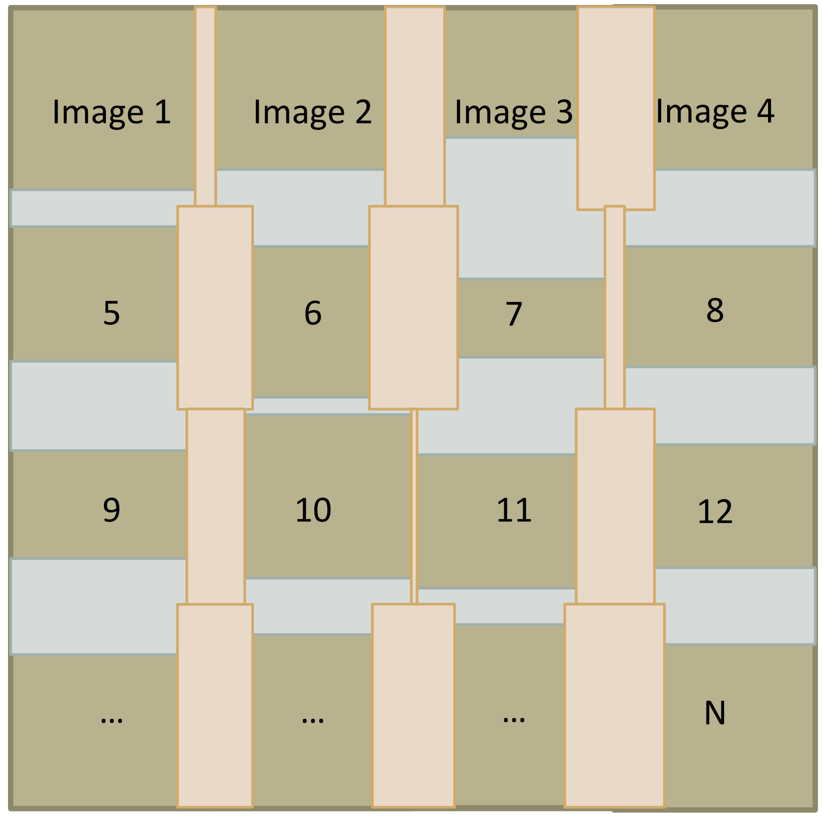


- Main focus is on phase 1

Image Stitching Algorithm

Loop over all images:

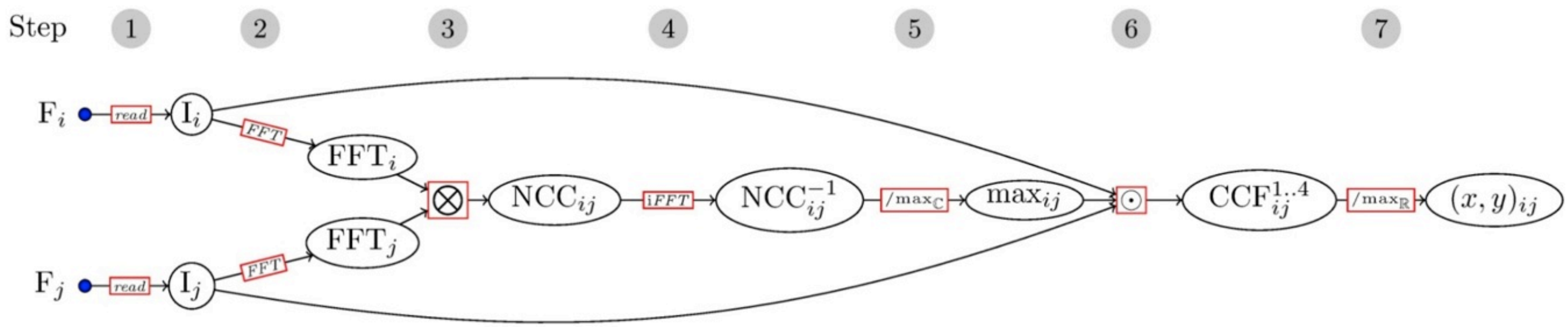
- Read an image tile
- Compute its FFT-2D
- Compute correlation coefficients with west and north neighbors
  - Depends on FFT-2D for each tile

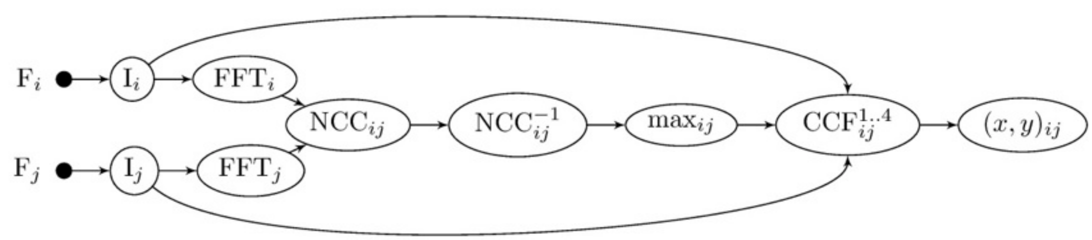


Major compute portions:

- FFT-2D of tiles
- Compute and normalize phase correlation
- Inverse FFT-2D
- Max reduction of correlation

Image Stitching Algorithm…





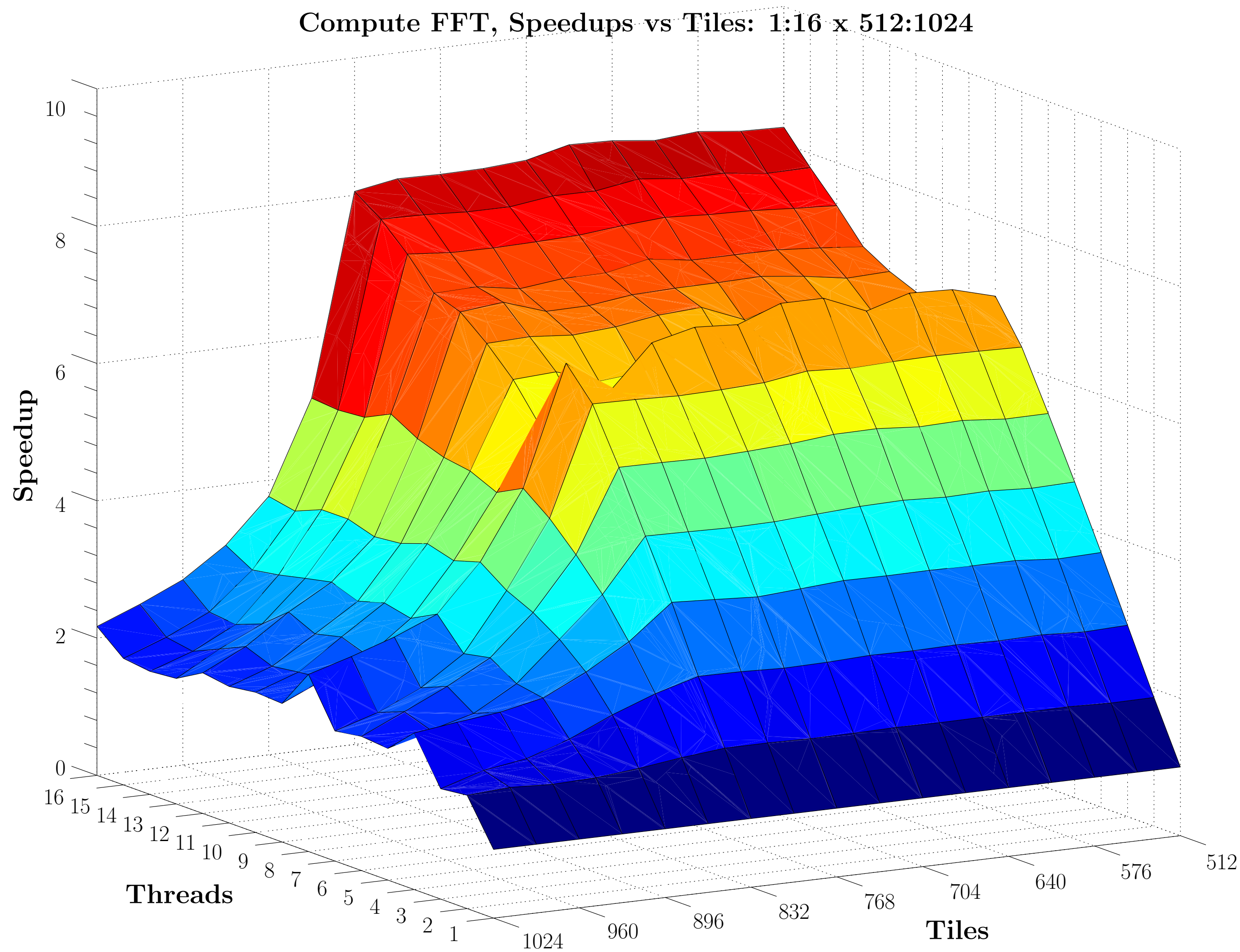
Algorithm's Parallel Characteristics

- Almost embarrassingly parallel
  - Large number of independent computations
  - For an *n* x *m* grid:
    - FFT for all images nxm
    - NCC for all image pairs $2nxm - n - m$
    - FFT^{-1} for the NCCs of all image pairs $2nxm - n - m$
    - …
- Caveats
  - Data dependencies
  - Limited memory

Memory Limitations

- Image tile:
  - 1040 x 1392 2-byte pixels: 2.76 MB
- Image transform
  - 1040 x 1392 x 16 = 22.1 MB
- Forward transforms for all tiles:
  - 22.1 MB x 59 x 42 = 53.5 GB!

Memory Limitations…



Performance-Driven Development Approach

- Extends *Edit-Compile-Debug* Cycle:
  - Edit-compile-debug
  - Measure performance
    - Identify bottleneck
  - Modify design

- Performance measurements
  - Execution time on a lightly loaded machine
    - Repeat timings, drop high & low values
  - Metered data structures
    - Examples: maximum & average queue lengths

Development Approach…

- Three main branches
  - Share code + utilities, regression tests…
  - Sequential
  - Pure multicore
  - Hybrid CPU-GPU
- Java plugin
  - Targeted for release in Feb 2015.

Implementations & Results

		Time	Speedup	*Effective* Speedup	Threads
CPU only	Sequential	10.6 min		20.3x	1
	Simple Multi-Threaded	1.6 min	6.6x	135x	16
	Pipelined Multi-Threaded	1.4 min	7.5x	154x	16
CPU-GPU	Simple GPU	9.3 min	1.14x	23.2x	1
	Pipelined-GPU, 1 GPU	49.7 s	12.8x	261x	16
	Pipelined-GPU, 2 GPUs	26.6 s	23.9x	487x	16
3 K20s	Pipelined-GPU, 3 GPUs	17 s	37.4x	759x	16

Sequential Implementation

Existing Implementations

ImageJ/Fiji

- Java code
- Multithreaded
- Timing: 3.6 hours!

NIST Prototype

- MATLAB code:
  - Timing: 17.5 min
- Remarks:
  - Caching file reads and FFT results?
  - Multi-threaded FFT routines
  - Optimized or multithreaded code for vector operations?

FFTW 3.3 (fftw.org)

Auto-tuning software

- Create plan to compute FFT
  - Based on CPU properties & FFT dimensions
- Planning mode specifies effort to find "*best*" FFT algorithm

Planning Mode	Planning Time	Execution Time
Estimate	0.02 s	137.7 ms
Measure	4 min 23 s	66.1 ms
Patient	4 min 23 s	66.1 ms
Exhaustive	7 min 1 s	66.1 ms

Amortized planning cost

- Save plan to (re)use later
- Run prior to stitching computation

Optimizations in Seq. Version

Memory I

- Free a tile's transform memory as soon as possible
  - Use reference counts

Memory II

- Traversal order:
  - Chained diagonal
- Maximum memory:
  - Short diagonal + 1
  - Pre-allocate at start & recycle

SSE intrinsics:

- Normalized Cross Correlation factors (step 3)
- Max reduction (step 5)

Results

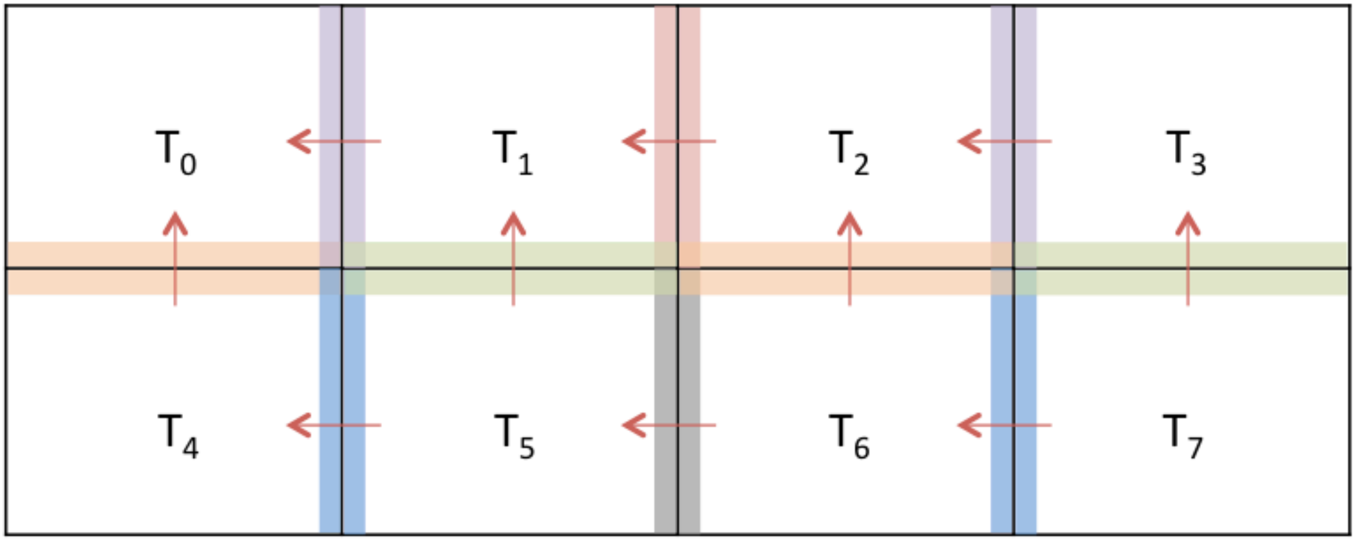
- > 80% of computation in forward and backward Fourier transforms
- Timing: 10.6 min

Speedups

ImageJ/Fiji	21
NIST prototype	1.6

Simple Multi-Threaded Implementation

Simple Multi-Threaded



- 8-way decomposition
- Arrows & shaded areas denote dependencies

- Spatial domain decomposition, one thread per partition
- Handle inter-partition dependencies via *barriers*

Simple Multi-Threaded…

Three phases for all threads separated by barriers:

1. Compute FFT of own images

   Compute relative displacements of tiles with no inter-partition dependencies

   Release memory of transforms w/o dependents

   Barrier

2. Compute relative displacements for remaining tiles (on north & east partition boundaries)

   Barrier

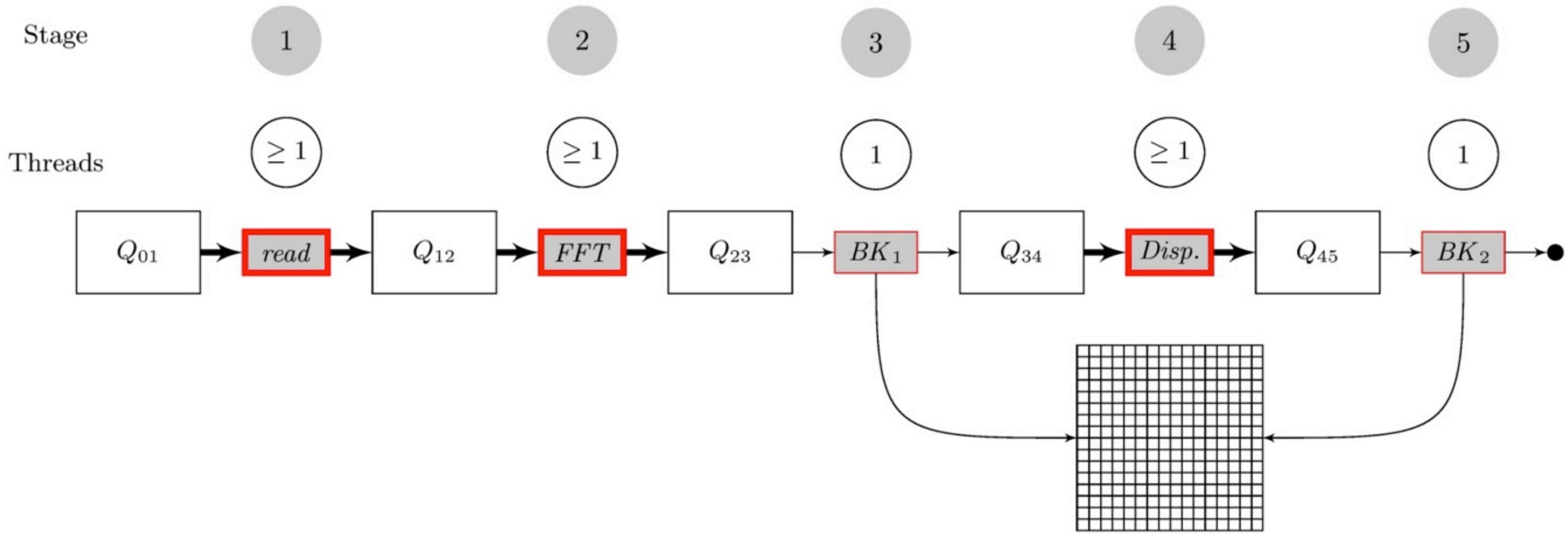
3. Release memory of remaining transforms

Simple Multi-Threaded…

- Timing: 1 min 35 s w/16 threads
    - Used to be 4 min 56 s w/8 threads in "Estimate" planning
- Load imbalance:
    - $T_{0:}$ processes all its tiles in phase 1; idle in phase II
    - T_1–T_7: cannot finish in phase I; different workloads in phase II
    - Unequal partition sizes
- Saturated I/O subsystem
    - Threads concurrently try to read image files
- Hyper-threading
    - Takes advantage of threads with mixed characteristics
    - FFTW functions optimize locality & are CPU-bound

Pipelined Multi-Threaded Implementations

Pipelined Multi-Threaded, v1

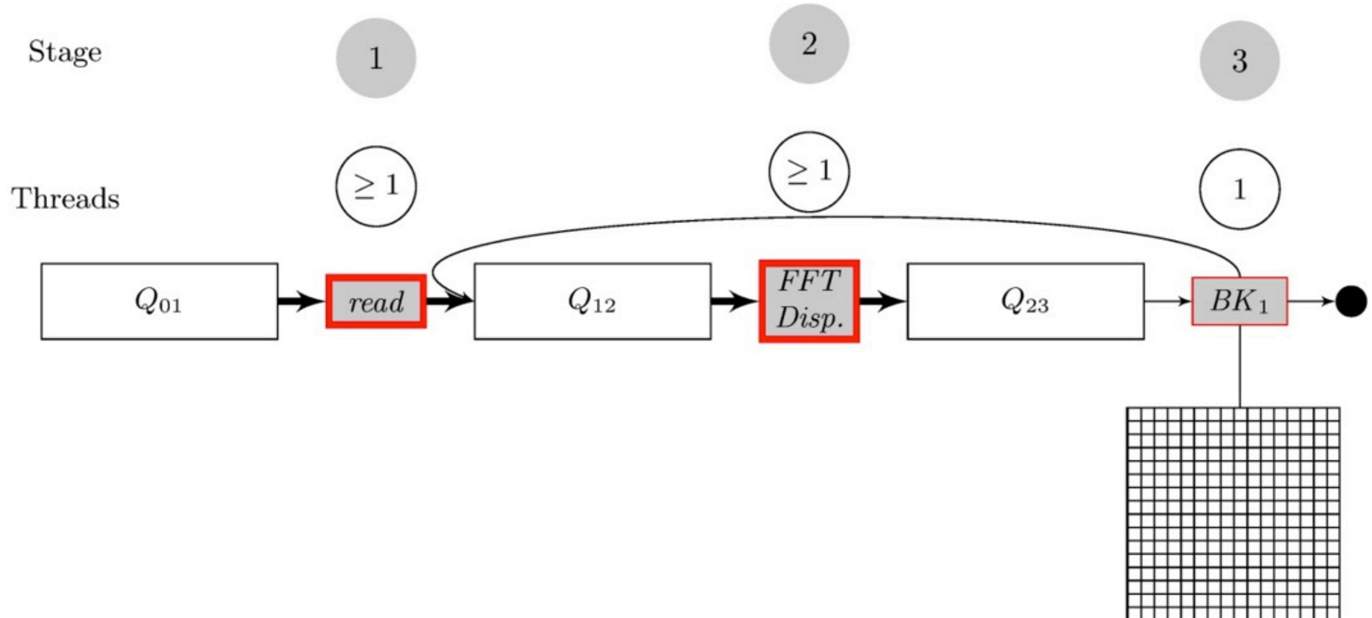


- 5-stage pipeline
  - Producer-consumer pattern
  - Queues with built-in synchronization & size limits
    - Reports max size & logs entries (for debugging)
    - First queue, Q_{01}, is optimized away
  - Memory management restricts number of images in system

Pipelined v1—Details & Optimizations

- 5-stage pipeline
- Producer-consumer pattern
- Queues
  - Built-in synchronization & size limits
    - Synchronization via mutexes & spinlocks
  - Log entries & report max size (for debugging)
- Seq. version optimizations
- Q_{01} optimized away
- Memory management
  - Restrict # images in flight
  - Throttle reader(s)
- Threads
  - 1 reader
  - 2 bookkeepers
  - *n* FFT & 2*n* Rel. Disp.

Pipelined Multi-Threaded, v2



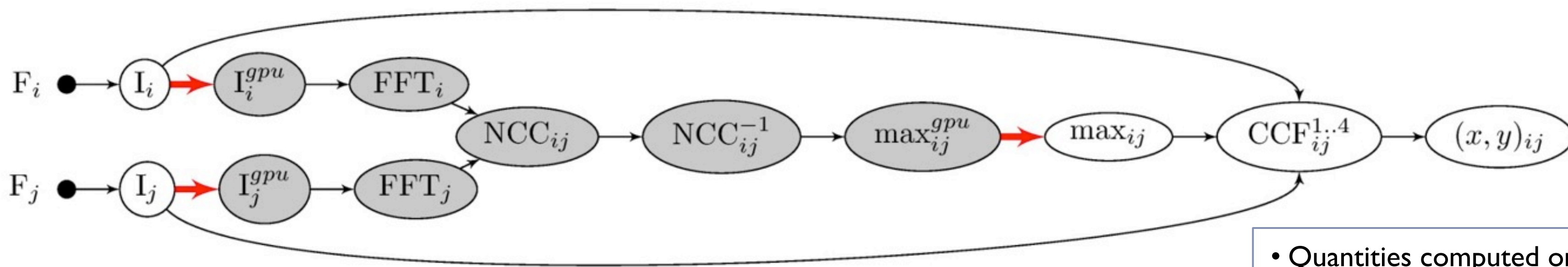
- 3-stage pipeline
  - Based on 5-stage pipeline version
  - Merge work queues into one priority queue
    - Favor relative displacement computation
    - Simpler thread allocation & better thread utilization
  - Merge bookkeeping queues

CPU-Only Speedups

	Time	ImageJ/Fiji	NIST prototype	Sequential
ImageJ/Fiji	3.7 h			
NIST proto.	17.5 min			
Sequential	10.6 min	20.3x	1.65x	1
Simple MT	96 s	135x	10.9x	6.6x
Pipelined MT	84 s	154x	12.5x	7.5x

Simple GPU Implementation

Simple GPU

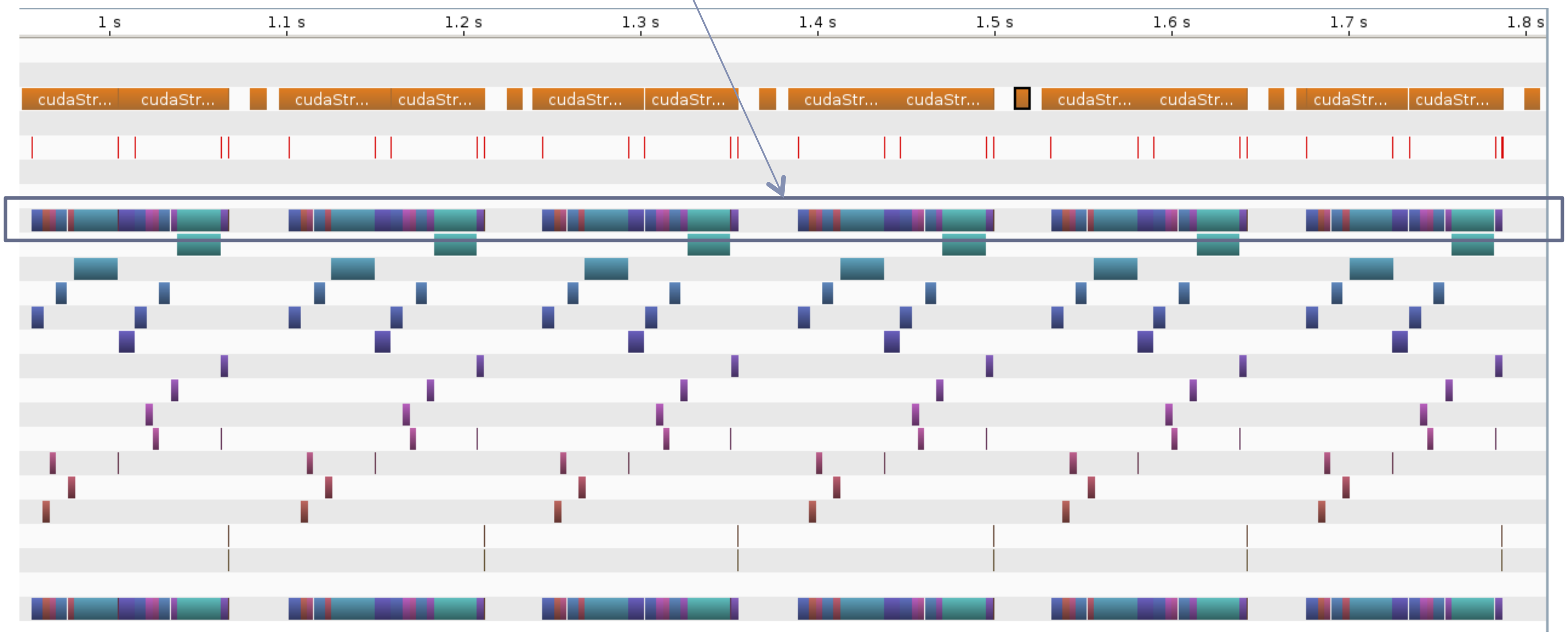


- Quantities computed on GPU appear in gray.
- Red arrows (→): CPU-GPU copy operation.

- Based on sequential implementation
- Offloads most computational tasks to GPU
  - CUFFT function calls for forward & backward transforms
  - Custom CUDA kernels for all other computations
- Copies image data to GPU memory
- Copies results back from GPU memory to RAM

Simple GPU—1 s profile

Single stream
Large repeated gaps



Simple GPU…

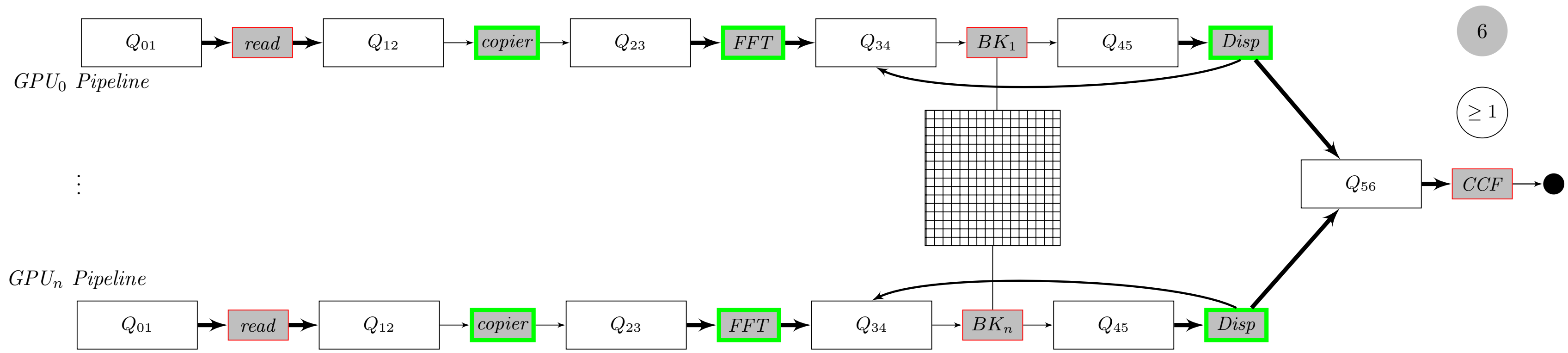
- Timing
  - 9.3 min
  - Speedup 1.14x!

- Optimizations
  - Custom kernels
  - Compute CCFs on CPU

- Conclusion
  - High cost of easy portability!

Pipelined GPU Implementation

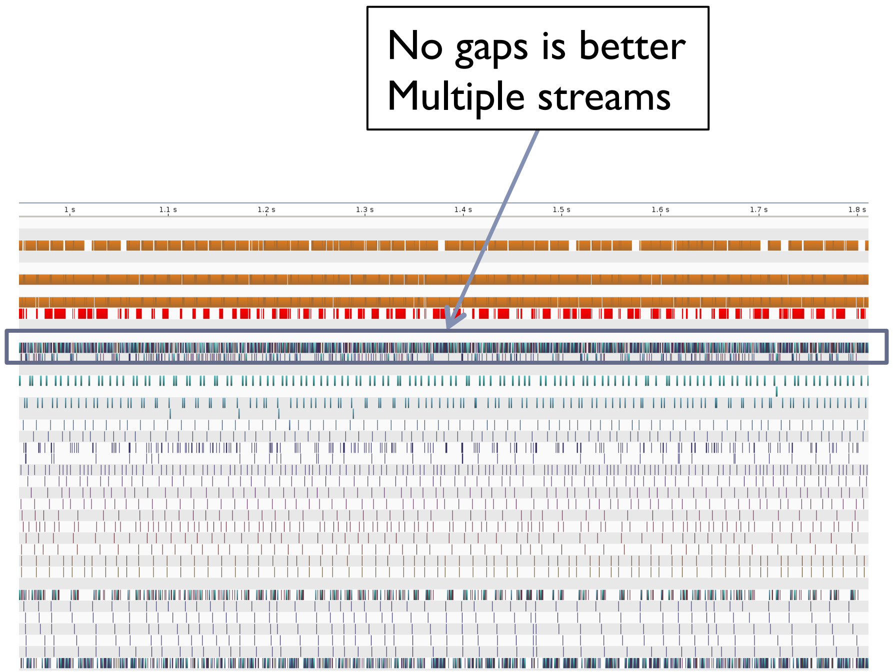
Pipelined GPU



- Seven stage pipeline:
  - Based on pipelined multi-threaded implementation
- CPU threads per GPU:
  - 1 of reader, copier, FFT compute, & rel. displ. compute
  - 2 bookkeepers
- For all GPUs
  - Multiple CPU CCF compute threads

Pipelined GPU—Optimizations

- Asynchronous copy:
  - Overlaps CPU-GPU transfers with tasks on CPU & GPU
- Peer-2-peer copy
  - Between GPUs
- Uses all available resources:
  - Two CPUs
  - Two GPUs
- Keeps GPUs busy



Same 1 s profile as "Simple GPU"

Speedups

	Time	ImageJ/Fiji	NIST prototype	Sequential
ImageJ/Fiji	3.7 h			
NIST prototype	17.5 min			
Sequential	10.6 min	20.3x	1.66x	1
Simple Multithreaded	96 s	135x	10.9x	6.6x
Pipelined MT	84 s	154x	12.5x	7.5x
Simple GPU	9.3 min	23.2x	1.79x	1.14x
Pipelined GPU, 1 GPU	49.7 s	261x	21.1x	12.8x
Pipelined GPU, 2 GPUs	26.6 s	487x	39.5x	23.9x

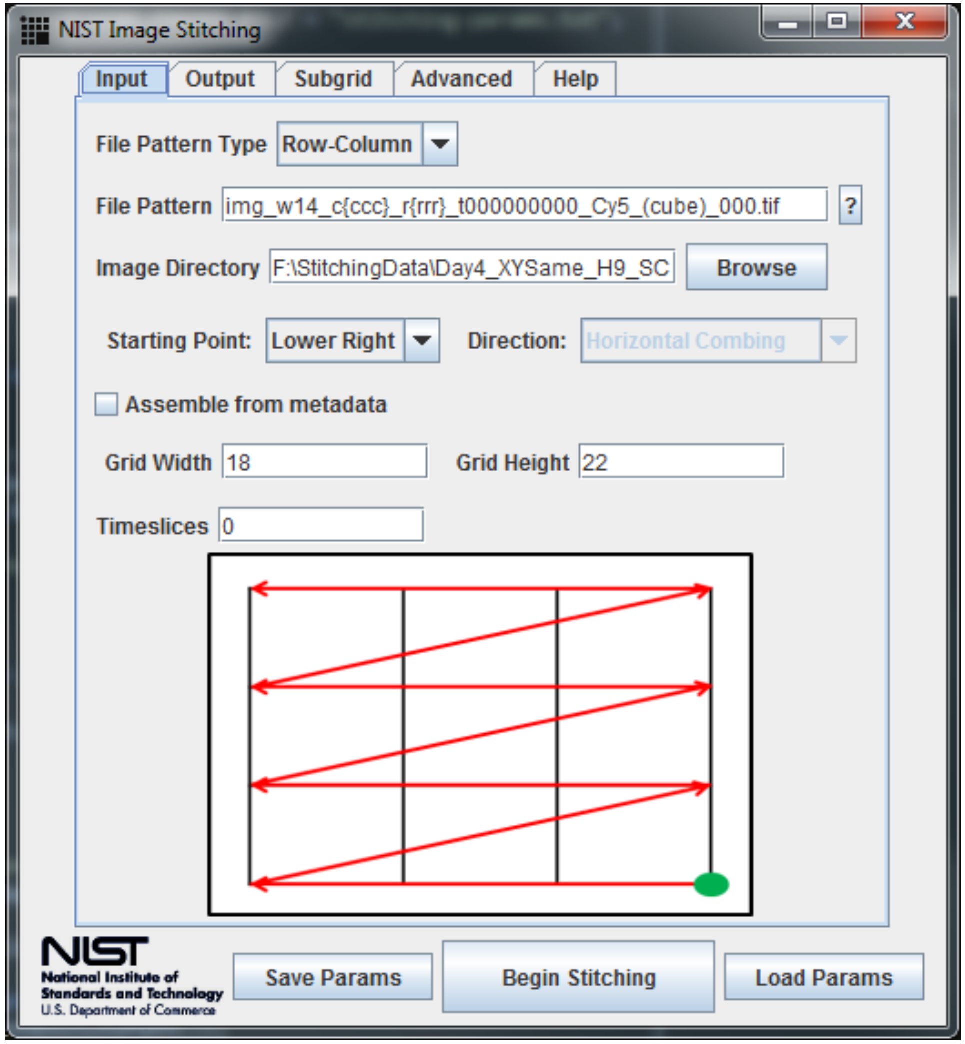
More Recent Results

- IBM Power8 + 3 x NVIDIA K40 GPUs
  - 13 s

Upcoming Code Release

- Java plugin
  - ImageJ/Fiji
  - Stitching + visualization
- Status:
  - Code review + fixes
- Release
  - Q1 2015
- Machine specs
  - Intel Xeon E5-2620 @ 2.3 GHz
    - 6 physical cores (12 logical)
  - 64 GB RAM
  - NVIDIA Tesla C2075
- Java performance
  - CUDA: 134 s
  - FFTW: 203 s
  - Java 32-bit FFT: 99.6 s

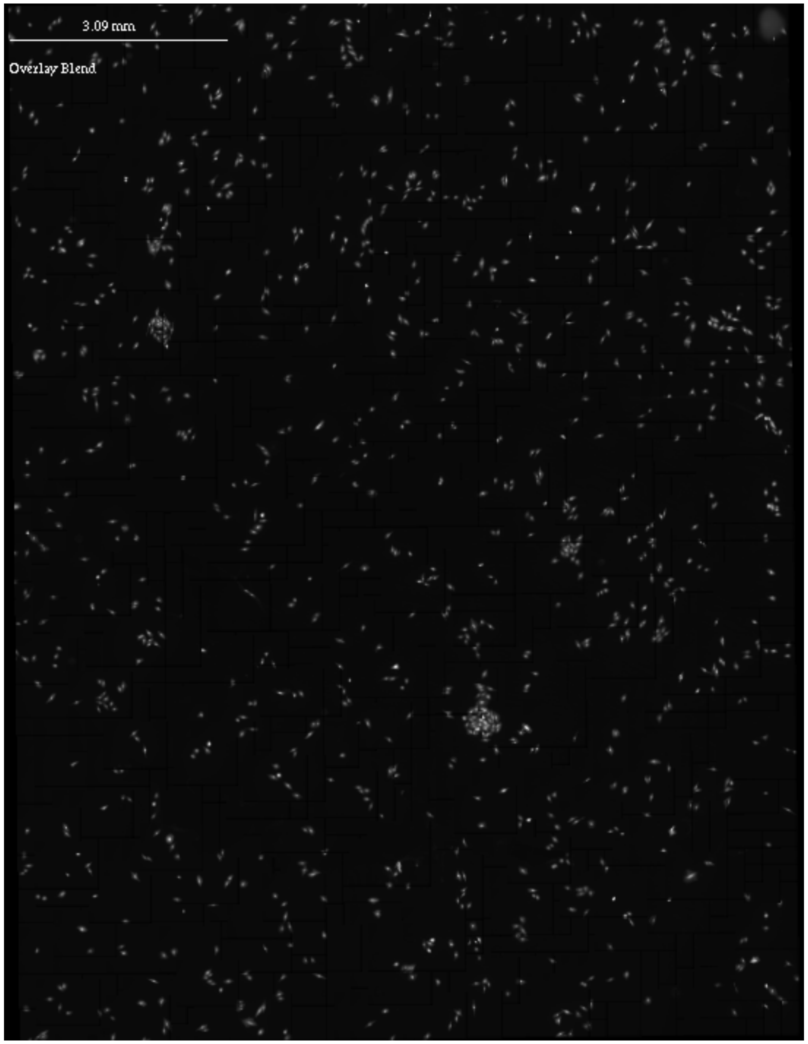
ImageJ Plugin

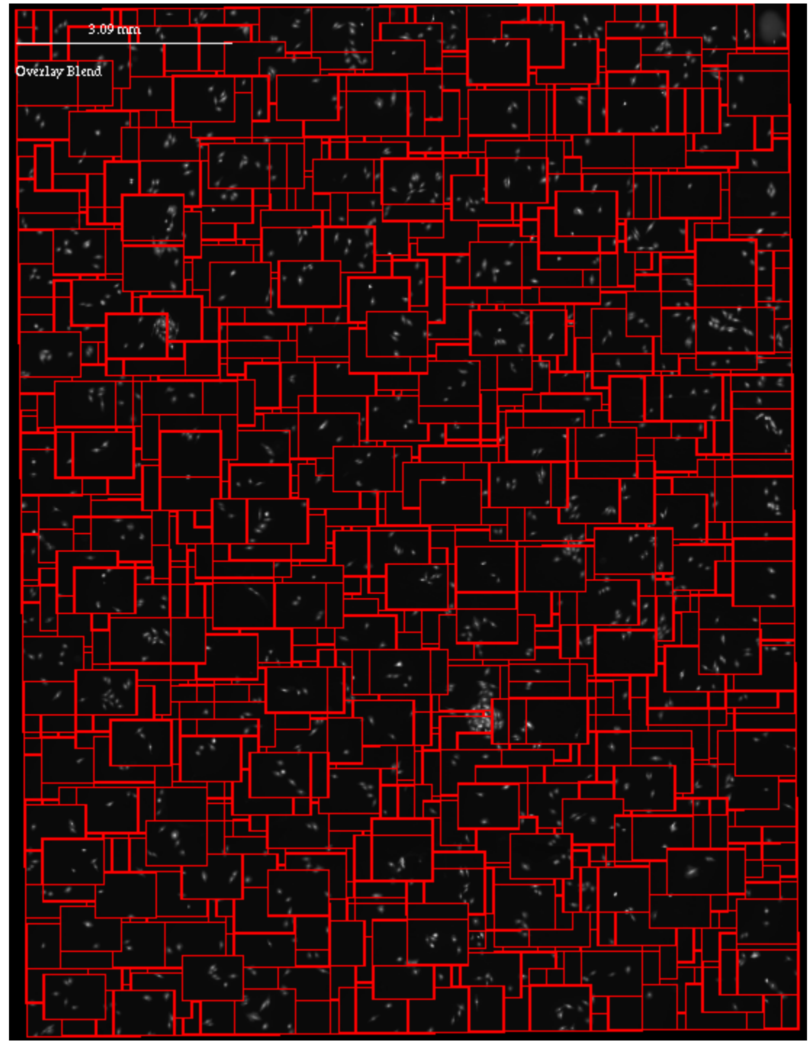


Visualization Overview

- Properties of data
  - Too big to display
  - Impossible to interpret as a whole
  - Sparse regions of interest
- Properties of data access
  - Users
    - Limited capacity for data perception
  - Programs
    - Need pixel-level precision
- Preprocessing
  - Generate and save image pyramids for all tiles
  - Advantage
    - Load minimum data to display given zoom level
- Cache lowest resolution reconstructed image
- Compose partial stitched image on demand

Stitched Images





Lessons Learned

Programs ≠ Algorithms + Data Structures

- Algorithm
  - Mathematical specification of sets of operations
- Data Structures
  - Logical organization of data
- View missing critical aspects for HPC!

Performance & Scalability

- Algorithms + Data Structures = Programs
  - Niklaus Wirth, 1973

- Algorithms + Data Structures + Scheduling + Memory Management = High Performance Programs

Coarse-Grained Parallelism

- Parallel Tasks
  - Decomposition of "Algorithm + Data Structures"
  - Data parallelism for particular operations
- Memory
  - Critical resource to manage
- Data Motion
  - Includes inter-process communication
  - Major delays!
- Schedule
  - Essential to tie together all of the above

Lessons Learned, v 0.2

- It works!
- Understand your computation
  - Correct choice of algorithm
    - $O(n \log n)$ vs. $O(n^2)$ or $O(n)$ vs. $O(n)$
  - Target 90% vs. 10%
- Think asynchronous
- Throw-away code
  - Performance oriented prototypes

Lessons Learned, v 0.2…

- Performance as 1st class citizen
  - Edit-compile-debug → edit-compile-debug-measure
    - Use visualization tools
- Performance vs. portability
- Widely applicable techniques
  - MATLAB prototype: 17 min → ~ 4-5 min
- Java implementation
  - Minutes vs. hours

Lessons Learned, v 0.2…

- Lack of tools
  - Refactoring 10 KLOC (ZENO) feasible by 1-person
  - 100 KLOC beyond scalability limit
- Kernels essentially invariant
  - Image stitching & ZENO
- Performance-oriented tools
  - Instrument code
  - Isolate code sections into kernels
  - Compose kernels into schedules
  - Meter & reuse memory

Closure—General

- 24x speedup (w.r.t. sequential implementation)
  - ~500x w.r.t. ImageJ/Fiji
- Representative data set:
  - 42x59 grid
  - ~ 0.5 minutes
- Low memory footprint by design
  - 4 GB of RAM
- Can budget compute time to:
  - Generate stitched image
  - **Carry out additional analysis**
  - Enables computationally steerable experiments

Closure—CPU & GPU Scalability

- Simple multi-threaded implementation
  - Does not scale well with threads
    - Performance tanks as the number of threads increases past 8
  - Attributed to:
    - Disk I/O being saturated
    - Load Imbalance

- Pipelined implementation
  - Scales well
    - Performance improves as the number of threads increases
    - Takes advantage of multiple GPUs
  - Attributed to:
    - Single reader thread able to keep disk busy without saturation
    - Load is balanced with pool of worker threads

Closure—Additional Work

- ImageJ plug-in + code release
  - Plug-in + ImageJ visualization
  - Visualization tool
  - C++ code
- Integrate into microscope controller software
  - Real-time feedback

Closure—Additional Work

- Systematize approach & analysis into API:
  - Tim's Ph.D. thesis

  - Task graph model
  - Execution pipeline model
    - Mapping between two

  - Memory management
  - Workflow scheduler
  - Petri nets

- Apply to other problems

Closure

- Dramatic performance improvements
  - Often within reach
- Requires software re-design
  - Tool support?
- May be at odds with portability
  - Is portability overvalued?

Questions?