Application-level Benchmarking with Synthetic Aperture Radar

Chris Conger, Adam Jacobs, and Alan D. George

HCS Research Laboratory
College of Engineering
University of Florida

20 September 2007

Outline

I. **Introduction & Motivation**
II. **SAR Algorithm Overview**
    I. Basic application
    II. Parallel decompositions
III. **Summary of Benchmark Features**
IV. **Benchmark Results**
    I. Experimental setup
    II. Performance results
    III. Visualization and error
V. **Conclusions**

Introduction & Motivation

- **New Synthetic Aperture Radar (SAR) application-level benchmark**
  - Strip-map mode SAR
  - Sequential, multiple parallel implementations
  - Written in ANSI-C, using GSL* and MPI **
- **Based on original SAR code provided by Scripps Institution of Oceanography**
- **Why did we "re-invent the wheel?"**
  - Multiple parallelizations, unique features
  - Simple code structure, easy to modify
- **Code originally intended for internal use, *decided to share with community***

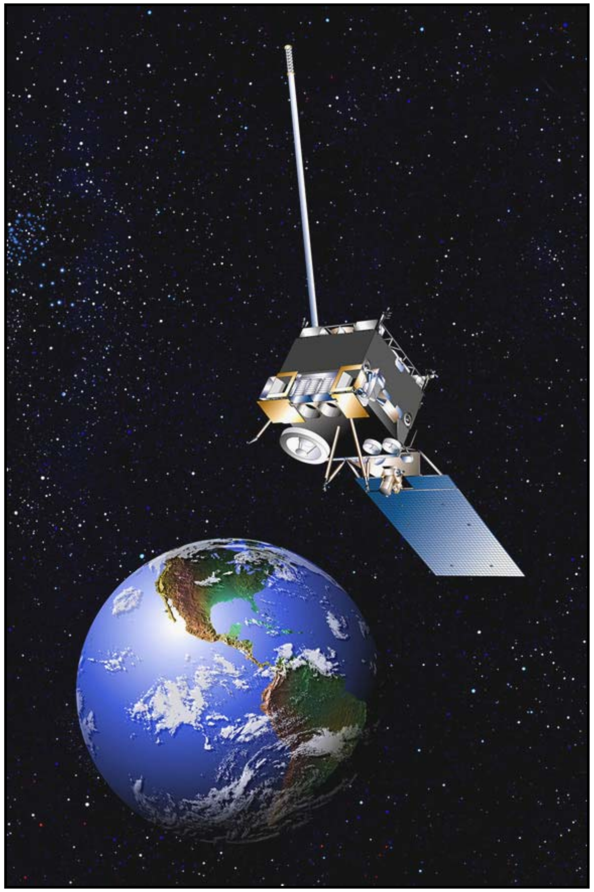


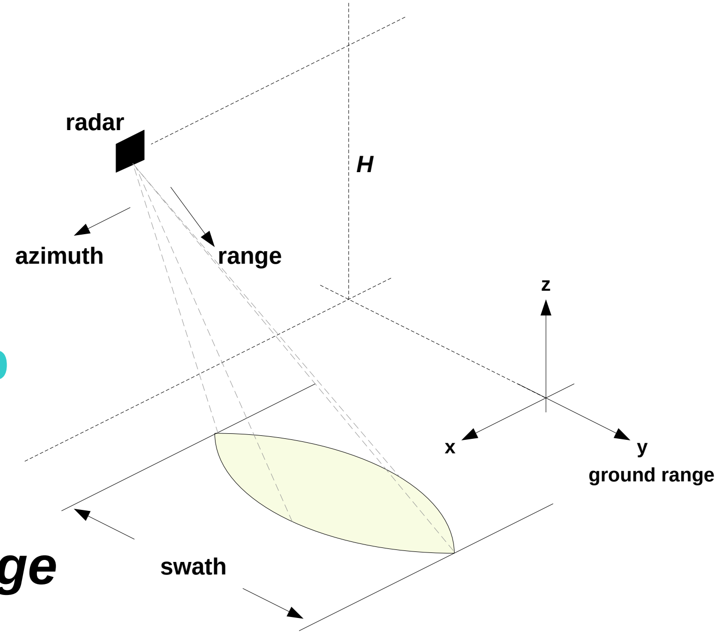
Image courtesy [1]

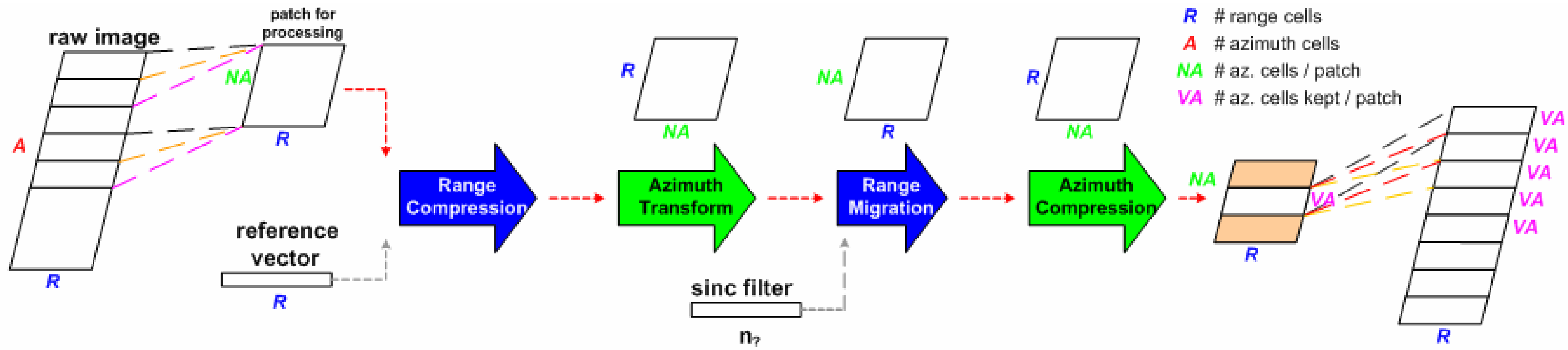
* GSL = Gnu Scientific Library
** MPI = Message Passing Interface

[1] http://www.noaanews.noaa.gov/stories2005/s2432.htm

SAR Algorithm Overview

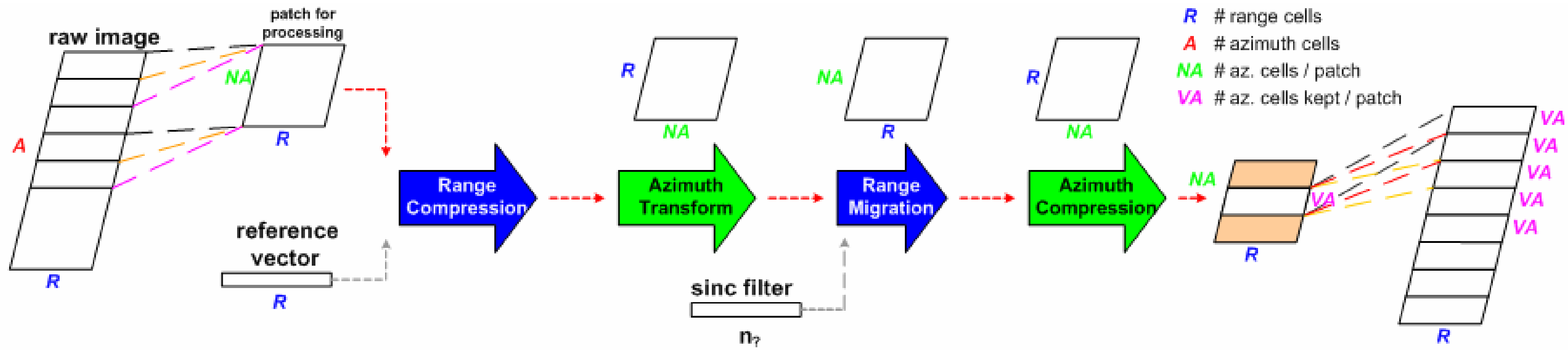
- **SAR produces high-resolution images of Earth's surface from air or space, using downward-facing radar**
- **This benchmark implements strip-map mode SAR, composed of four stages**
- **Data is complex 2-D image,** ***must be transposed between each stage***
  - Range dimension: distance from radar
  - Azimuth dimension: different radar pulses/pulse returns





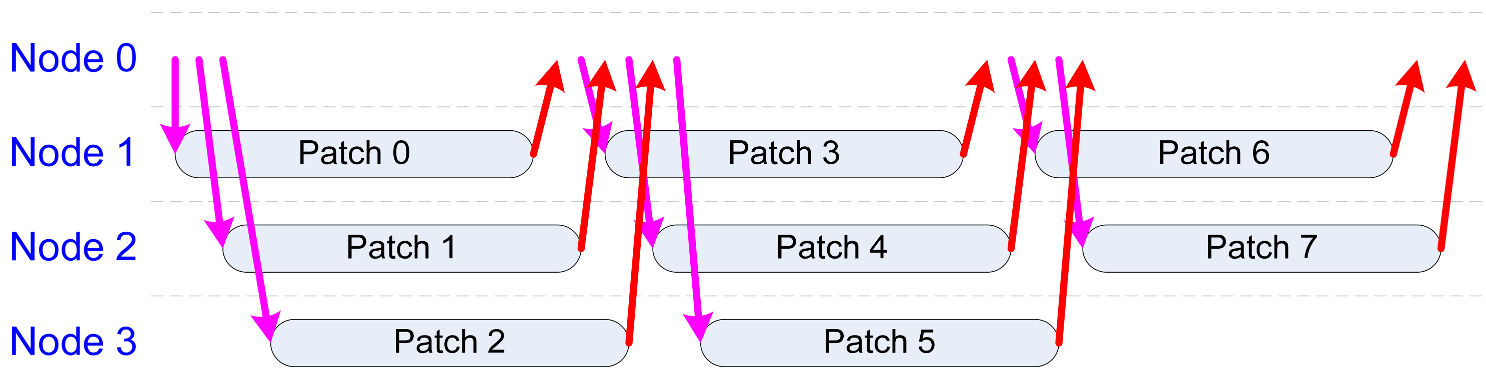
SAR Algorithm Overview

- **Sequential, baseline implementation (S1)**
  - Entire raw image is processed in patches, with *overlapping boundaries*
    - Each patch can be processed completely independently of other patches
    - Portion of each fully-processed patch is kept, appended together seamlessly
  - Patch size is variable along azimuth dimension
    - Smaller means lower memory and computational requirements per patch, however more repeated calculations across different patches
    - Larger means higher memory and computational requirements per patch, however less repeated calculations across different patches
  - Read one patch from file, process, and write to output file… repeat



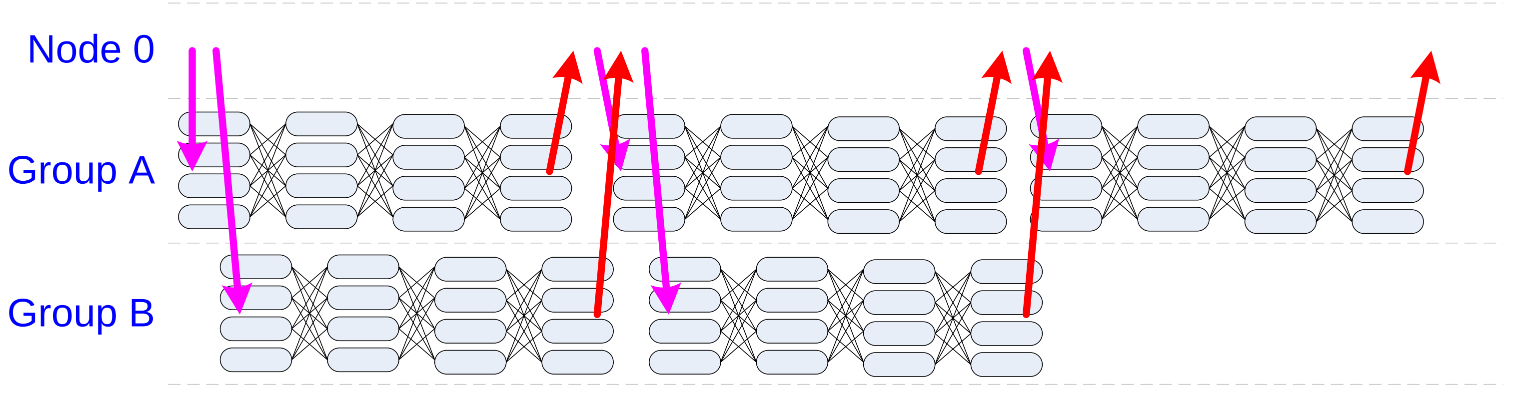
SAR Algorithm Overview

- **Parallelization #1 (P1) – Distributed Patches**
  - Master-worker partitioning of *N* nodes, one master and *N*-1 workers
    - Master node responsible for file I/O, sending and receiving patches
    - Worker nodes wait to receive data from master, perform actual SAR processing
  - One patch per worker node, two different data distribution strategies
    - P1-A: first parallelization, sends to all workers, receives from all workers, repeat
    - P1-B: optimized data distribution (shown below), workers receive new patch immediately
  - Maximum number of workers is bounded by number of patches in full image
  - ***No distributed transposes needed for this parallelization***
  - Ideally, full image processing latency reduces to single-patch processing latency



SAR Algorithm Overview

- **Parallelization #2 (P2) – Distributed Parallel Patches**
  - Master-worker partitioning of *N* nodes, one master and *N*-1 workers
    - Master node responsible for file I/O, sending and receiving patches
    - Worker nodes wait to receive data from master, perform actual SAR processing
  - Worker nodes separated into *G* groups of nodes, one patch per *group*
    - When *G* = 1, this reduces to a system-wide, data-parallel decomposition
    - When 1 < *G* < (*N* – 1), this becomes a hybrid data-parallel/distributed-patch decomposition
    - When G = (*N* - 1), this reduces to P1 parallelization
  - No inherent upper bound on number of nodes that can be used
  - Distributed transposes necessary *within each group of nodes*



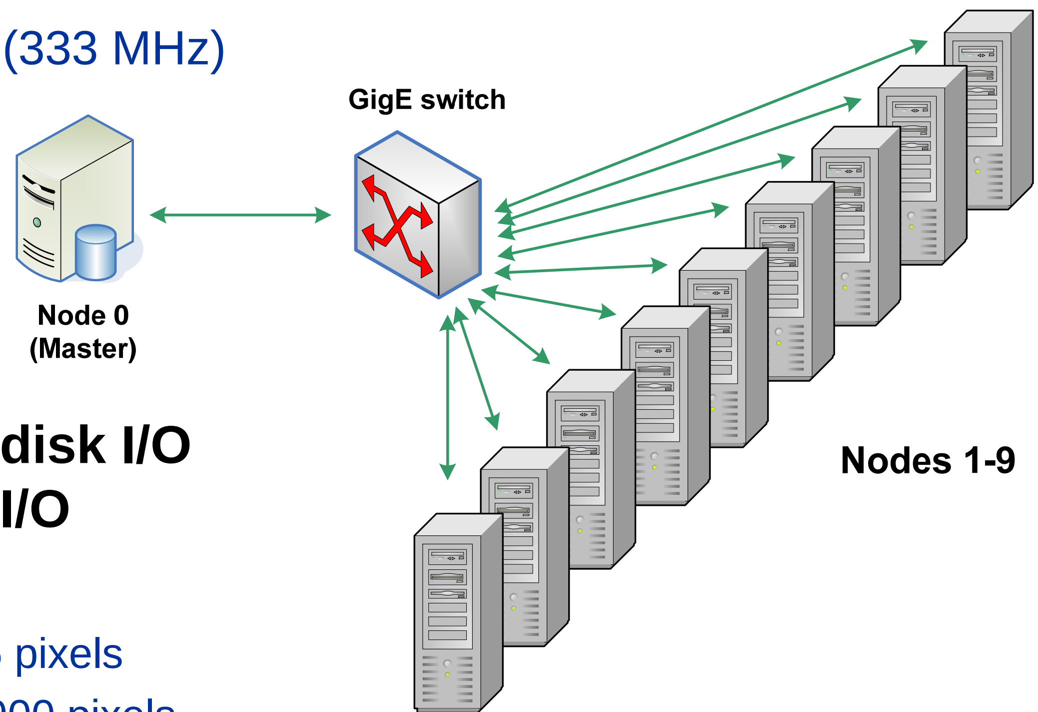
Summary of Benchmark Features

- **Selectable precision, single- or double-precision floating point**
- **Adjustable image sizes, radar parameters**
- **Adjustable memory usage**
  - Artificially limit amount of memory available to SAR application
  - Determines patch size used for processing full image
- **Data and bitmap generation tools**
  - Input data generator for arbitrary-sized input files (random data)
  - Bitmap file generator to convert benchmark output to viewable file
- **Modular code structure**
  - Can replace GSL with other math library, by editing *one source file*
  - Written to read and process raw SAR files from ERS-2 satellite, can be easily modified to interpret other file formats
- **Sample ERS-2 image provided† with benchmark source code**
- **Documentation!!**

† image can be downloaded from public website, http://topex.ucsd.edu/insar/e2_10001_2925.raw.gz (last accessed 08/25/2007)

Benchmark Results – Experimental Setup

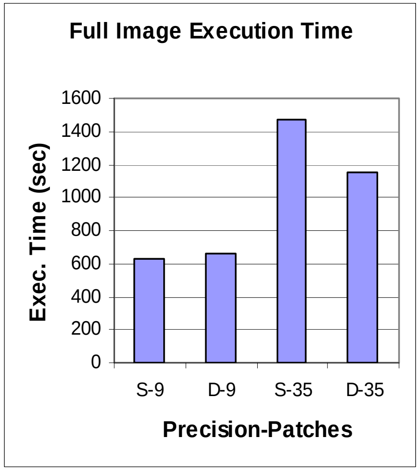
- **As an example, benchmark was run on 10-node cluster of Linux servers, connected via GigE switch; each node contains:**
  - 1.42 GHz PowerPC G4 processor
  - 1 GB of PC2700 memory (333 MHz)
  - Gigabit Ethernet NICs
  - 120 GB hard drive

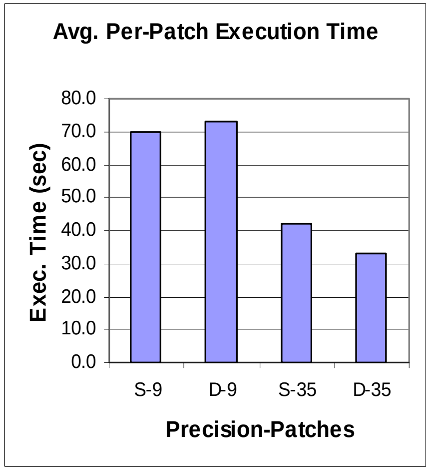


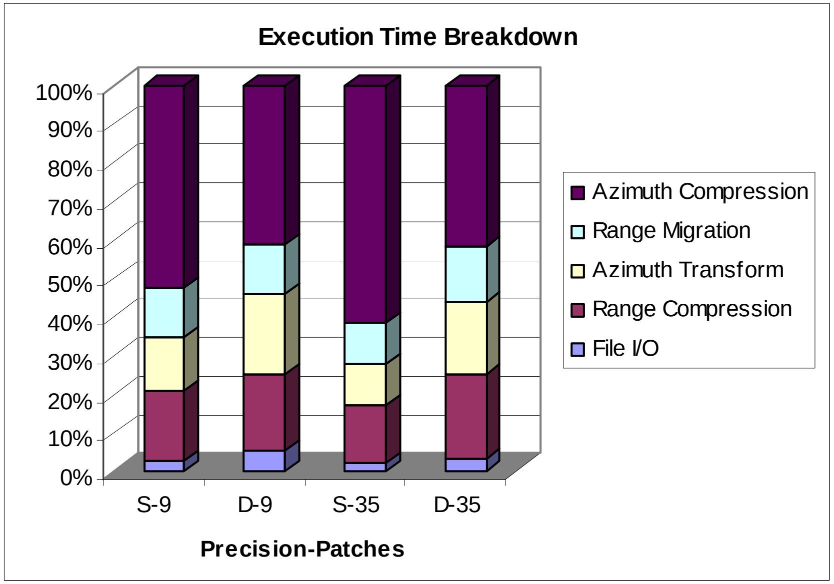
- **One server reserved for disk I/O and majority of network I/O**
- **Full image dimensions:**
  - Range dimension size: 5,616 pixels
  - Azimuth dimension size: 27,900 pixels
- **Ideally, process entire image in < 16 sec**

Benchmark Results – Sequential Baseline (S1)

- **Two valid patch sizes considered:**
  - Using 5616×4096-pixel patches, entire image can be processed in 9 patches
  - Using 5616×2048-pixel patches, entire image can be processed in 35 patches
  - Each pixel represented by complex element
- **Notation in figures:**
  - **S-9** single precision, 9 patches
  - **S-35** single precision, 35 patches
  - **D-9** double precision, 9 patches
  - **D-35** double precision, 35 patches
- ***Slower to process full image with smaller patches, however faster per-patch with smaller patches***
- **Figure to lower-right shows percentage of overall latency for each stage**
  - Transposition of patches included in azimuth-processing stage latencies
  - Azimuth compression takes longer with single-precision floating point?
  - Per-stage contribution depends on precision, but not so much on number of patches

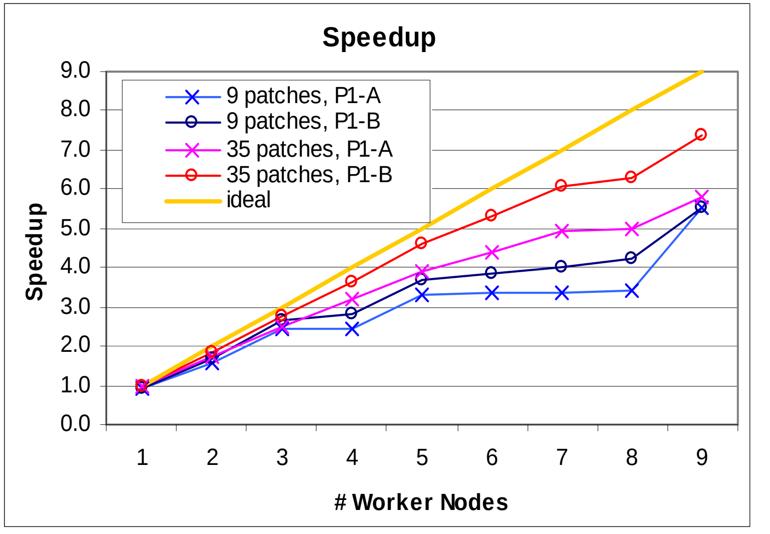




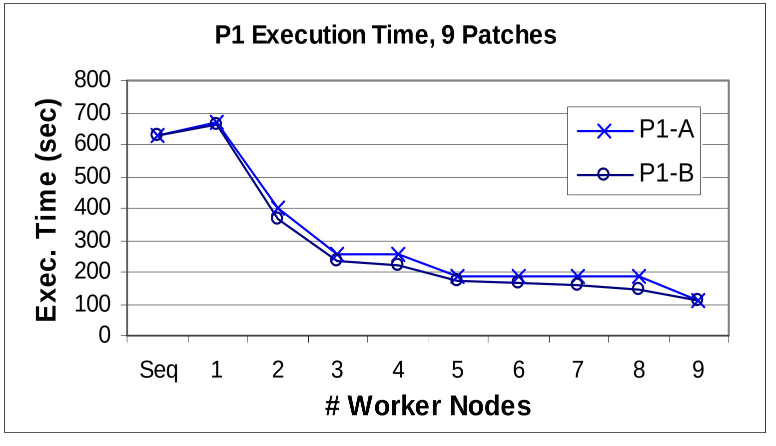


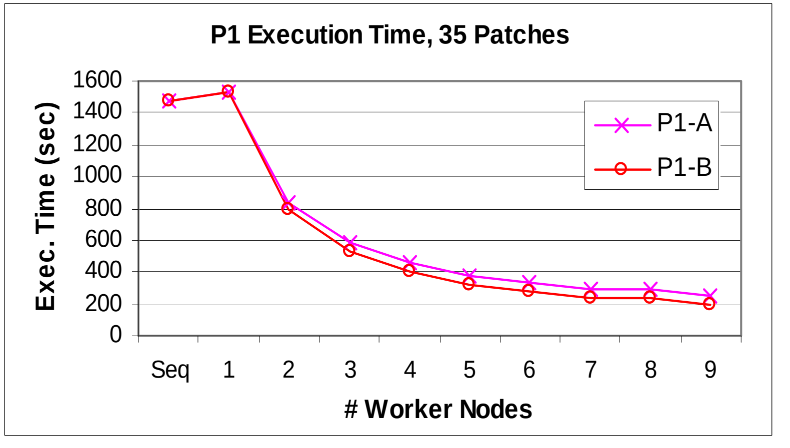
Benchmark Results – Distributed Patches (P1)

- **Recall two different data distribution strategies, P1-A and P1-B**
- **Same two patch sizes as in S1 results**
  - Smaller patches may provide better scalability, but net performance is consistently worse
  - Entire range of possible nodes shown for 9-patch case
  - For 35-patch case, could use up to 35 worker nodes (red curves extend beyond what is shown, blue do not)
- **Too many restrictions result from this coarse-grained parallelization**
  - Max number of nodes capped
  - Best possible latency same as single-patch latency (~42 sec for 35 patches, ~69 sec for 9 patches)
  - *May never be able to achieve desired performance!*



"ideal" based on number of worker nodes, not *total* number of nodes



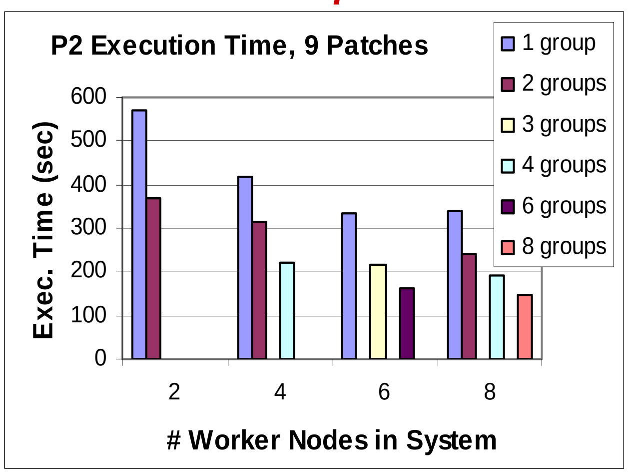


single-precision only on this slide

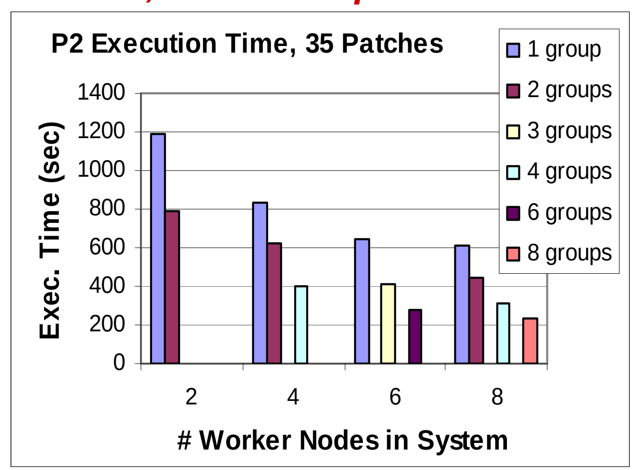
Benchmark Results – Distributed Parallel Patches (P2)

- **For this system, P2 parallelization below shows no improvement over P1**
  - Why? P2 features multi-level parallelism, but this cluster only a single-level architecture
  - In all cases, performance penalty of distributed transposes within groups overpowers performance improvement of data-parallel processing in each stage
  - Cost of all-to-all communications of corner turns over Gigabit Ethernet is prohibitive
  - Systems of multiprocessor nodes or multicore devices much better targets for P2 method
- **Using more nodes would provide better visibility into true performance limits**
- **For this parallelization, both dimensions of a patch must be divisible by number of nodes in a group (restricts valid system sizes)**

660 sec sequential



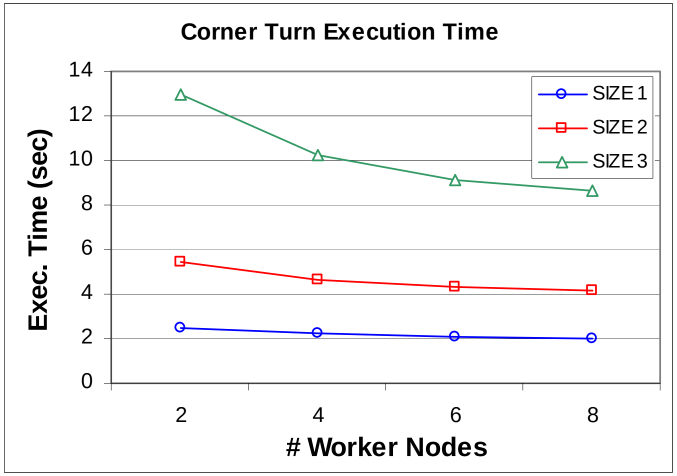
1,464 sec sequential



single-precision only on this slide

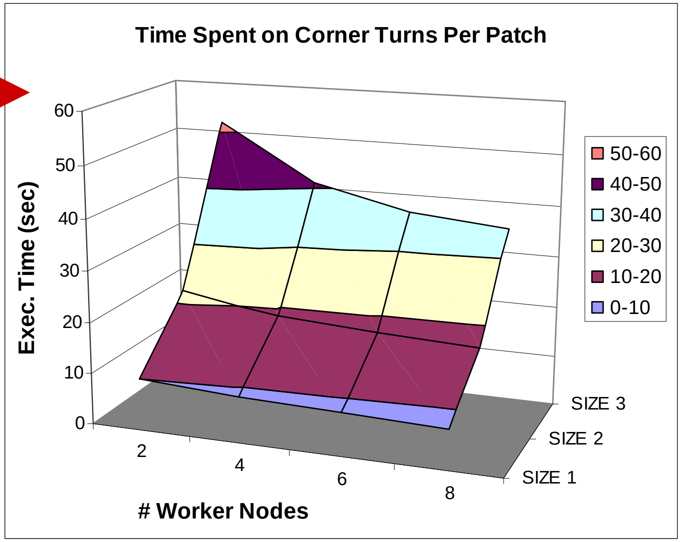
Benchmark Results – Distributed Transposes

- **Distributed transpose also called *corner turn* (CT)**
- **Third, larger patch size included, 5616 x 8192**
  - For provided example image, not valid patch size (too large)
  - Included only for CT study, for wider range of patch sizes
  - In real-time system, or with larger images, would be valid option
- **CT latency per patch is smaller for small patches, however many more patches per image as well**
  - Values in bottom-most table calculated assuming a single group of *N* worker nodes must do all patches sequentially
  - Recall, multiple groups *can* operate concurrently
- **Large values explain inability for P2 parallelization to provide better performance on this platform**



TOTAL TIME SPENT ON CORNER TURNS PER PATCH (sec)

	# corner turns	2 workers	4 workers	6 workers	8 workers
SIZE 1	4	10	9	8.44	7.96
SIZE 2	4	21.76	18.72	17.4	16.72
SIZE 3	4	51.8	40.88	36.36	34.64

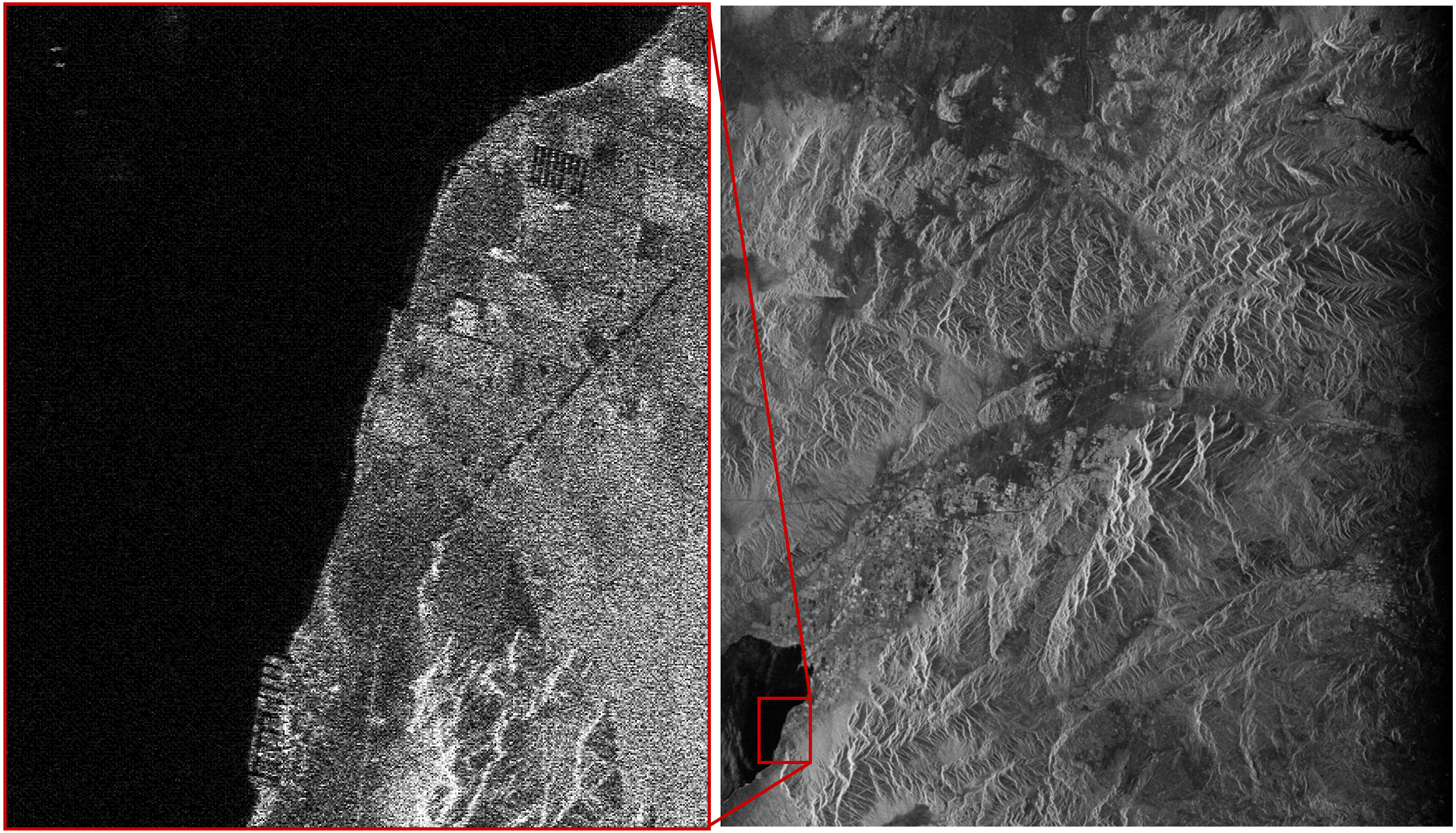


TOTAL TIME SPENT ON CORNER TURNS FOR FULL IMAGE (sec)

	# corner turns	2 workers	4 workers	6 workers	8 workers
SIZE 1	140	350	315	295.4	278.6
SIZE 2	36	195.84	168.48	156.6	150.48

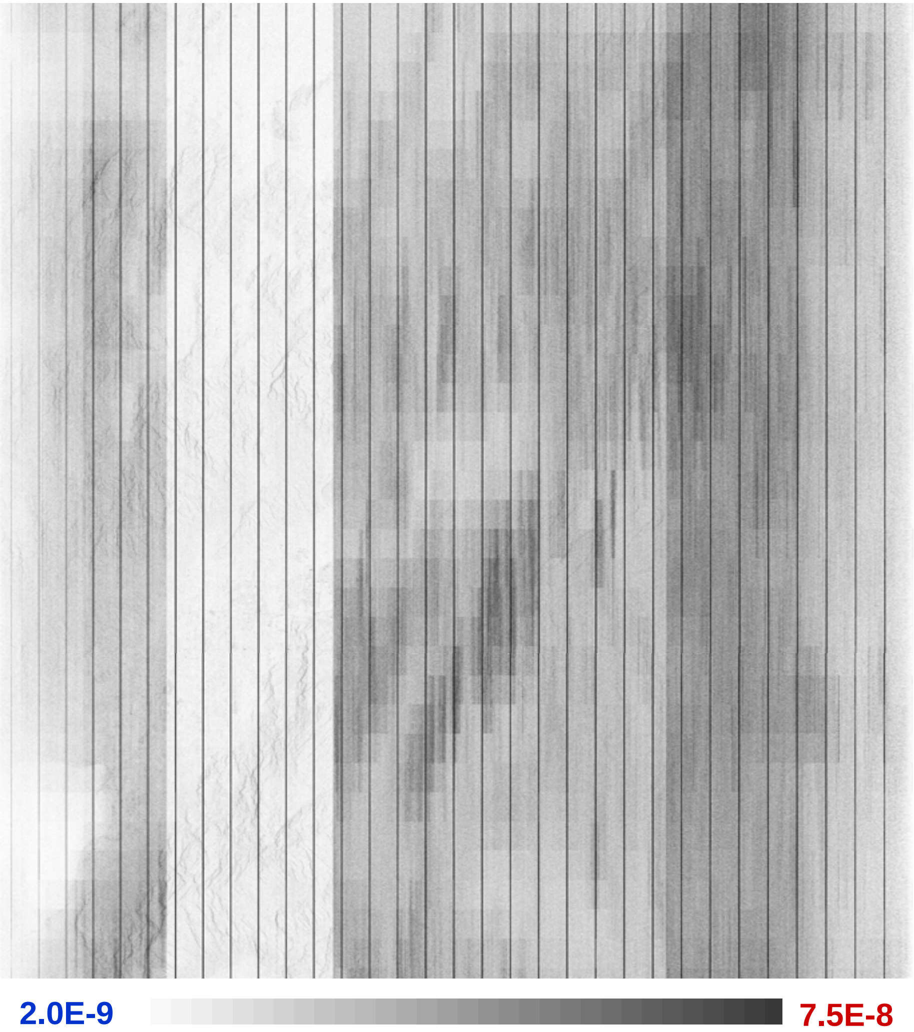
SIZE 1 – 5616 x 2048, 35 patches
SIZE 2 – 5616 x 4096, 9 patches
SIZE 3 – 5616 x 8192, 3 patches

Benchmark Results – Visualization and Error



Benchmark Results – Visualization and Error

- **Range of values in output:**
  - Maximum: 0.02979
  - Minimum: 0.00000
- **Error between outputs produced using single-/double-precision**
  - Maximum pixel error: 3.314E-6
  - Minimum pixel error: 0.000
  - Mean-squared error: **< 1.0E-9**
- **Original input file contains 5-bit fixed-point data, more bits would result in more error in output**
- **No visible differences between single-/double-precision images**
- **Single-precision data means:**
  - Only half as much data to move around the system
  - Lower processing latency from single-precision FP operations



Benchmark Results – Benchmark Materials Delivered

- **Source code and documentation provided together, but separate from example ERS-2 input file**
- **Source code package includes all three SAR implementations**
  - Sequential baseline (S1)
  - Both parallelizations, (P1) and (P2)
- **Documentation which covers:**
  - Mathematics of this SAR implementation
  - Code structure description and diagrams
  - Instructions on how to compile and run the benchmark
  - Pointers to other related reference material
- ***GSL or MPI libraries not delivered with benchmark material*... user's responsibility to ensure proper libraries are installed**

Conclusions (1)

- **Developed malleable code-base for strip-map mode of SAR, sharing with community to freely use for benchmarking or other case studies**
- **As provided, code is not optimized for any particular platform… lots of room for improving performance on specific targets**
  - Replace GSL as math library with something optimized for target architecture
  - Optimize distributed transpose algorithm for P2
  - Overlap file accesses and network communication at master node
  - Use more than one node to perform file access and/or distribution of data
- **Multi-level parallelism exploited through P2 does not map favorably to non-hierarchical system topologies (e.g. basic star)**
  - P2 better fit for multi-level parallel system architectures (e.g. clusters of SMPs/MCs)
  - Balance number of workers per group with localized processing resources
- **Based on observed performance of P1 and P2, a pipelined parallelization seems like it would most easily support real-time SAR**
  - Unless highly-optimized distributed transposes provide vast improvements in performance, may simply be too much data for data-parallel decompositions
  - Having better mapping between target system architecture and P2 parallelization could also significantly improve application performance

Conclusions (2)

- **Intended uses of this benchmark:**
  - Measurement and comparison of system performance
  - Realistic code-base for arbitrary research case studies
  - Professors could use this code for class projects (parallel computing, radar theory, etc…)
- **Other application-level benchmarks in development:**
  - Ground-Moving Target Indicator (GMTI)
  - Pixel classification with Hyper-Spectral Imaging (HSI)
  - *Searching for more ideas*
- **Potential future VSIPL++ implementation and comparison with ANSI-C/MPI/GSL baseline**

To download source code and documentation:
http://www.hcs.ufl.edu/~conger/sar.tgz

To download example input file from ERS-2 satellite:
http://topex.ucsd.edu/insar/e2_10001_2925.raw.gz

Acknowledgements

- We would like to thank Dr. David T. Sandwell of Scripps Institution of Oceanography for the generous donation of sequential SAR code that served as the basis for the implementations included in this benchmark

- We also extend thanks to Honeywell – Space Electronics Systems in Clearwater, FL for their support of this research

To download source code and documentation:
http://www.hcs.ufl.edu/~conger/sar.tgz

To download example input file from ERS-2 satellite:
http://topex.ucsd.edu/insar/e2_10001_2925.raw.gz