Swathbuckler: HPC Processing and Information Exploitation

Scot Tucker, Robert Vienneau
Advanced Engineering & Sciences
ITT Industries
Rome, NY
Scot.Tucker@rl.af.mil, Robert.Vienneau@itt.com

Joshua Corner, Richard W. Linderman
Information Directorate
Air Force Research Laboratory, USAF
Rome, NY
{cornerj, lindermanr}@rl.af.mil

***Abstract*—This real-time system recorded raw returns and processed continuous strip-map, high-resolution (< 1m), wide-swath (37 km) imagery on-board the Convair 580 aircraft with an embedded high performance computing system costing under $100K. The SAR image formation algorithm was optimized at all levels of the computer architecture to achieve real-time processing over 9000 times faster than the original algorithm specification. The information management system implemented a pub-sub-query environment to allow real-time offboard exploitation of the on-board multi-terabyte database within seconds. Real-time SAR image formation was demonstrated on five flight tests. The completed flights provided 7.3 terabytes of raw and processed imagery to support future algorithm research.**

I. INTRODUCTION

The Swathbuckler experiment is an effort to develop and demonstrate wide-swath, high resolution, real-time strip map SAR image formation. The program was conducted under the auspices of the signal/image processing panel of The Technical Cooperation Program (TTCP) which involves the defense research organizations of the US, UK, Canada, and Australia [1]. Following lab and aircraft integration activities, the Swathbuckler demonstration culminated in five flights in Ontario, Canada during the fall of 2005. During the flights, the aircraft flew hexagons with 40 km sides with the radar looking inward, as shown in Fig. 1. Each leg produced roughly a 40 km wide x 37 km deep imaged strip 20 to 57 km from the aircraft which overlapped with the imaged areas of the other legs. While Swathbuckler covers many technical areas, the focus of this paper is the embedded high performance computing and information exploitation. Three other papers deal with the system architecture [2], processing front-end [3], and radar system [4].

Some of the most difficult challenges for this experiment involved meeting the real-time requirements for embedded signal/image processing and devising a means to exploit the data both on-board and offboard the aircraft. The SAR processing algorithm was provided by Defense Research and Development Canada - Ottawa. It was implemented using the MATLAB® signal processing environment and could process 60k range lines of 16384 points into imagery using 16k azimuth blocks in 12.5 hours time. To run in real-time, processing had to be measured not in hours but tens of seconds. AFRL and ITT Industries, Advanced Engineering & Sciences Division, implemented the MATLAB® image formation algorithm in C++ and optimized it for the High Performance Computing (HPC) platform on 24 backend dual Xeon nodes. AFRL also developed the information management system both for the embedded system and the data repository using the Joint Battlespace Infosphere (JBI) framework [5]—a grid-based pub-sub-query information management environment.

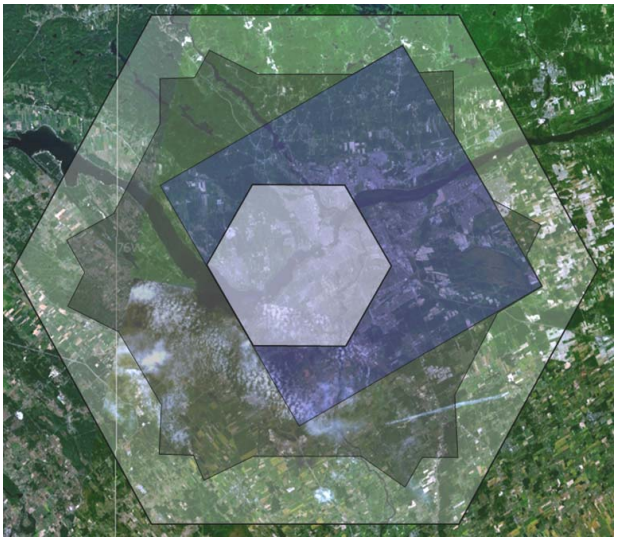

Figure 1. Ottawa, Ontario Hexagon Flight Pattern

II. OBJECTIVES

Three objectives drove the design of the Swathbuckler processing system. These objectives have an interlinked dependency on each other.

0-7803-9497-6/06/$20.00 © 2006 IEEE.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2006		**00-00-2006 to 00-00-2006**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Swathbuckler: HPC Processing and Information Exploitation	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Advanced Engineering & Sciences,ITT Industries,Rome ,NY**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
2006 IEEE Radar Conference, held in Verona, NY on April 24-27, 2006

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**8**	

A. *Real-time Stripmap SAR Processing*

Two attributes of this objective are affordable and real-time. Affordability improves the ability to transition the technology by both decreasing cost and also complexity of the solution by aggressively leveraging commercial technology advances. The objective was to demonstrate that well understood cluster computing technology costing less than $100K could do the job.

The real-time aspect has significant implications on the computational design because the complex SAR processing had to be done within a time limit on-board a resource constrained aircraft. For example, the power limit was 8 kW and the chassis had to fit in two shortened racks onboard the aircraft. The goal here was to produce imagery across a swath width 40 km as fast as possible. The goal PRF value of 800 translates into processing SAR imagery at 6 km^2 /sec. However, since the PRF was slaved to aircraft velocity this was a primary control available to adjust the processing rate.

B. *Image Exploitation*

Once the imagery has been processed there is great value in having secure fast custom access to the imagery. Thus the images had to be accessed easily and quickly both onboard and offboard the aircraft. Remote users had to be made aware of newly created images and be able to query the imagery database onboard the aircraft by their custom-defined parameters. The goal was to support this exploitation with latency of seconds—not minutes.

C. *Real-time Data Recording*

Post-demonstration analysis and continued algorithm research motivated the storage of both the raw image returns and processed imagery. The raw data supports post-processing using new algorithm configurations. Note that the recording of processed data to an onboard database is implied by the exploitation objective. Both sets of data are real-time flows and must be captured to disk without falling behind.

III. APPROACH

To achieve real-time SAR image formation across a wide swath it was important to maximize the performance of the processing architecture. The approach developed from estimation of processing potential, to distribution of the data, communication overhead, influence of aircraft and radar variables, process distribution, memory management, optimization, data storage and finally the exploitation approach.

A. *Comparing Peak to Required Performance*

Dual Xeon processors in 1U rack-mountable format are widely used in business as web servers, but for $2K-$3K also can provide considerable signal processing. Each 3.2 GHz Xeon processor features a peak performance of 12.8 billion single precision FLoating-point OPerations per Second (FLOPS). Twenty four dual nodes have a peak of 614 billion FLOPS (GFLOPS).

Analysis determined the stressing parts of the algorithm to be the FFT and Inverse FFT operations for compression in the range and azimuth directions, motion compensation, Range Cell Migration Correction (RCMC) and auto focusing. At 800 Hz PRF, these tasks amount to sustaining approximately 100 GFLOPS which is 26% of peak. Based on FFT benchmarks, this seemed achievable.

Another stressing condition was introduced by memory limitations from the corner turning operations and memory management of the required memory address space for each process—limited to around 3 GB on the 32 bit Xeon architectures. However, by mapping shared memory blocks into the 3 GB process space, larger memory could be tapped, albeit as sequential overlays of process memory.

The basic approach to processing the SAR imagery in parallel was to divide the swath width (40 km) into 24 subswaths about 1.5 km each. Each subswath holds 16384 points in range, about 10000 of which are "new", the rest are overlapped with the adjacent node to support overlap-save range compression. Thus the node 1 would process the range closest to the aircraft and node 24 would process the range lines furthest from the aircraft.

An important attribute of the processing is that the azimuth compression work gets harder at further ranges because the length of the azimuth filter increases and so the overlap regions (which are thrown away) increase. This motivates going to larger azimuth FFTs at further ranges to make more headway in the presence of the overlap inefficiency. In this case, 16K azimuth FFTs are used for the first 8 range subswaths and 32K FFTs are used for the remaining 16 subswaths as shown in Fig. 2.

B. *Radar Data Input*

Another key challenge to achieving a simple and affordable solution was converting raw A/D input data into packetized information objects with headers in a form communicable to the COTS dual Xeon processing nodes. These nodes featured dual Gigabit Ethernet links, so one was devoted to the real-time inputs while the other provided the normal cluster networking. An FPGA at the A/D front-end solved this interfacing problem efficiently. It converted the A/D data into separate UDP packets steered toward each of the 24 nodes, replicated the data for range overlaps, attached headers, and optionally also performed IF to baseband downconversion as discussed in [3].

C. *Communication Overhead*

Using the overlap-save method to break the range compression across the wide swath into several subswaths introduces a communications overhead proportional to the range pulse compression ratio. The replication of data could be accomplished either by the FPGA or by message passing amongst the cluster nodes. In this case, the FPGA-side did the replicating using the high capacity of the gigabit

Ethernet, and this allowed the heavily tasked cluster nodes to be spared the extra work of inter-node communications.

There is a data dependency inherent in the SAR algorithm--the range processing precedes the azimuth processing. Thus, the intermediate data structure needs to be passed between these processes. Because this communication was kept local to each node, shared memory could be used instead of slower message passing protocols. Communications, via shared files, is however needed to get the current aircraft position/altitude for the motion compensation from the cluster headnode.

Even with shared memory, the corner-turning between the two Xeons is expensive. The large cube is turned once to perform the azimuth compression FFTs, but there was a problem with RCMC wishing to turn back to a range orientation and then return to azimuth orientation. This was avoided by altering the RCMC algorithm to use cache lines in the azimuth direction and avoid re-turning the cube.

D. Related Variables: Velocity and PRF

Real-time SAR processing depends on a number of variables but one critical to the Swathbuckler processing design was the PRF. Because the aircraft speed dictates the rate at which the returns are passed to the HPC, changing the aircraft velocity effectively changes the PRF. As the aircraft speed increases, the PRF increases. So the PRF linearly throttles the data rate into the processing system and thereby the computations. To stay aloft and level, the minimum PRF is roughly 400 PRF. Thus, the desired minimum PRF was set at 500 and if the SAR processing lags, the system was designed to gracefully degrade by dropping blocks of images as buffers fill.

E. Processing Distribution and Pipeline

A logical division for a dual-processor node was to put the range processing functions on one of the processors and the azimuth processing on the other. The original algorithm sequentially processed all range lines first and then processed the azimuth lines. The implementation processes the first 16k (or 32k) range lines then starts the second CPU azimuth processing on that block in parallel while the first CPU continues onto the next block of range lines. Once an image is finished, it is immediately written to disk. Internal buffers ensure efficient flow between the input, range, azimuth, and writing portions. This represents a 4-stage pipeline.

F. SAR Algorithm Optimization

The original algorithm implemented in MATLAB® took 12.5 hours to process 60000 range lines of one subswath into images. For a 500 Hz PRF which was used in the flight tests, 500 range lines, each with 24 subswaths, are produced

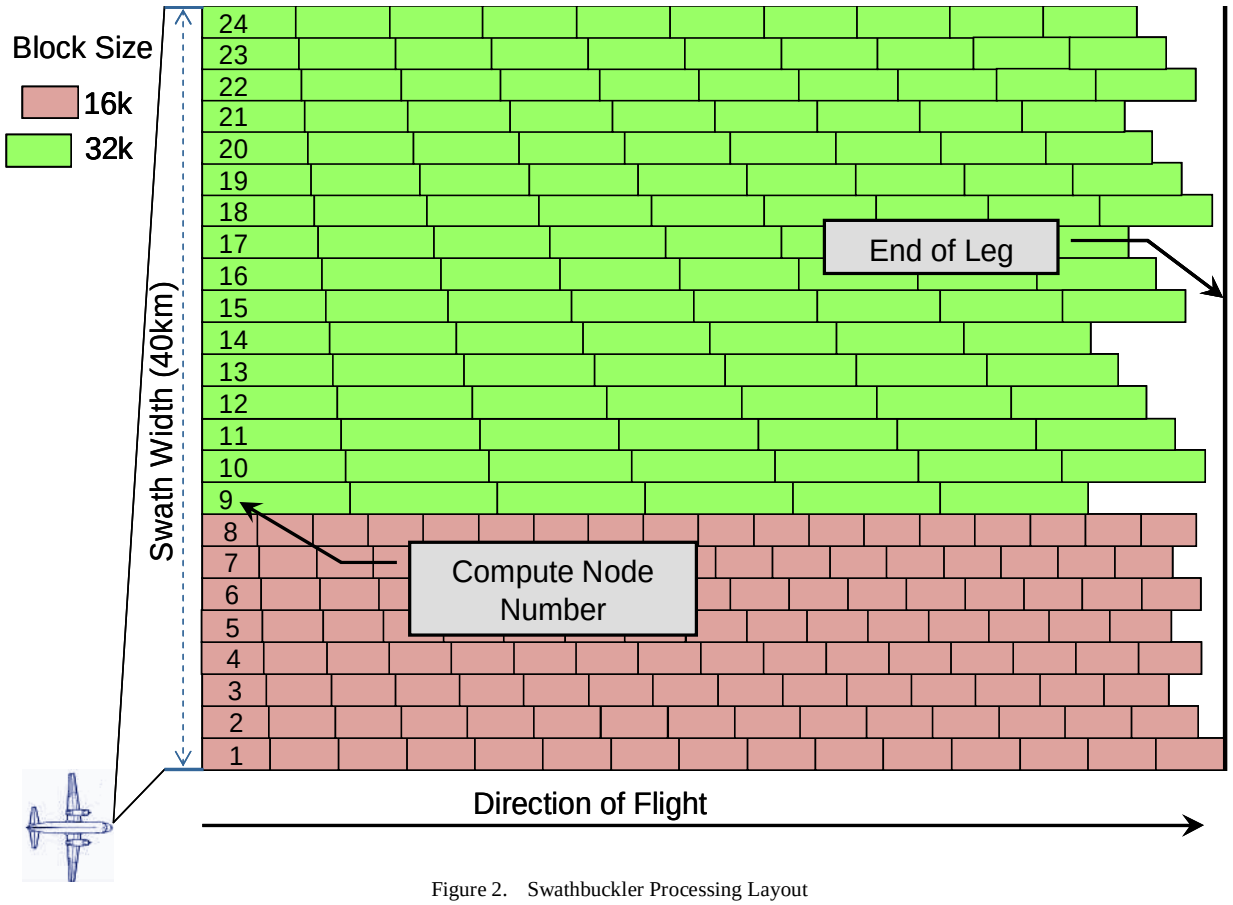

Figure 2. Swathbuckler Processing Layout

each second. This is a factor of 9022X speedup. So the original code, though being a functional prototype, was far from real-time. To start, the algorithm was re-targeted to a 24 node Beowulf cluster. Over time, processor clocks increased from 2.55 GHz to 3.06 and 3.2 GHz. During the flights, eight 3.06 GHz and sixteen 3.2 GHz dual-Xeon nodes comprised the cluster with an average of 6 gigabytes of shared memory per node.

Using optimization tools, a series of advances were implemented to reduce the execution time.

1) Corner Turn

This is a very expensive operation because of the large memory latency. These Xeon microprocessors have a 2 or 3 level cache hierarchy which is critically important to achieving good performance. The Level 2 (L2) cache sized at 512KB on the 3.06 GHz processors, and 1 MB on the 3.2 GHz processors was especially important. The corner turn function reads one line and then writes it back out-of-stride to a distant part of memory—essentially performing a matrix transpose operation. Because the corner turn moves data outside the cache's locality on a frequent basis, the 3+ GHz CPU clock is effectively slowed to the speed of the front side bus—566 MHz later increased to 800 MHz. In addition, this already costly operation was repeated four times during the baseline image formation task.

To work with the cache design, we changed it into a piece wise operation that took advantage of the cache lines which are the quanta of information moving through the memory hierarchy. This greatly reduced time consuming cache-misses. We reduced the number of corner turns required by rewriting the RCMC to run across cache lines. A final corner turn after image formation which wrote the image out unit-stride in range was also eliminated. In the end, the four corner turns were reduced to one.

2) Auto Load Balancing

Because of the pipeline architecture, if one CPU gets ahead, it can share the work that may be holding up the other. Typically, the azimuth processor was running behind range processing throughout the optimization effort. Therefore, the system allowed the first two steps of the azimuth processing to be dynamically taken over by the range processor.

G. Data Storage

The impact of storing the both the raw and processed data led to the need for increased disk capacity for the Coyote cluster. Maintaining the independence of each data set allowed for quick reprocessing and data re-distribution tasks. Each node was equipped with two separate 200 GB hard disks—one for processed imagery and the other for raw data. A minor price for the continuous recording capability is that 7% of each CPU's time is dedicated to this data recording activity. Using ultra mode 5 direct memory access and separate IDE controllers the impact would have been much worse. Special attention had to be given to data capacity rating of the connection ribbon between the hard disk and the motherboard. There were several occasions when the wrong type of connection ribbon doubled the transfer times.

H. Exploitation

Meeting the exploitation objective required an information management system that provided the desired characteristics—custom query, remote accessibility, security, and flexibility. We were able to leverage a development of such a system called the Joint Battlespace Infosphere (JBI) that had been developed by AFRL and already shared under the Information Management panel of TTCP.

The JBI uses a publish and subscribe communication method [5]. All communication between clients is defined in an XML schema. This allows clients to be loosely coupled which assists rapid prototyping and flexible systems. The Swathbuckler user console displayed JBI publications of cluster node status to track performance and health. This functionality proved invaluable during integration and flight testing.

Metadata associated with image formation, including the locations of the image corners in lat/long, was published as each node finished an azimuth frame. All these publications were stored permanently. The rectangles described by image information publications were plotted on a live map display as shown in Fig. 3. As the aircraft flew in its hexagon pattern, the images show up on the map at the corresponding geo-registered location on the display. Each leg of the hexagon is portrayed in a different color. At the time of the photo in Fig. 3, four legs of the hexagon were complete. A latitude and longitude position could be selected with a mouse click which allowed the user to query the hexagonal data for any image containing that point. With results returned from the query, the user could then request sub-regions of the images for viewing. These were excised from

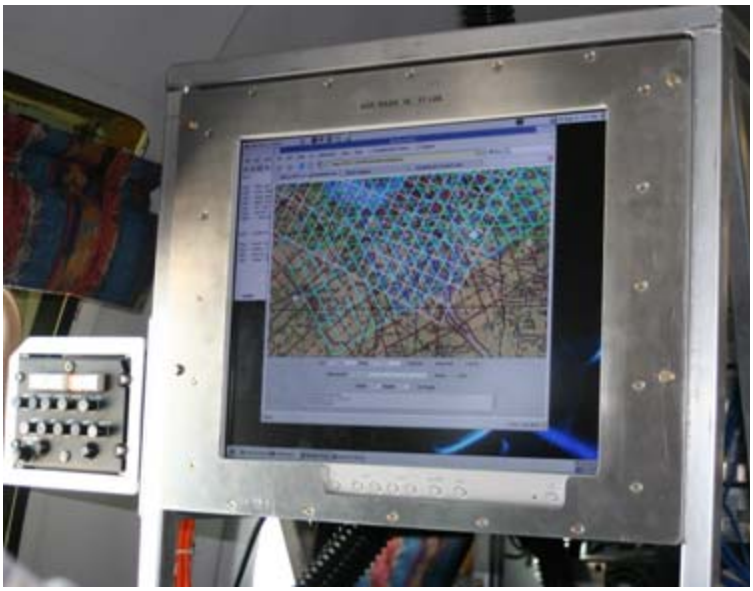

Figure 3. Swathbuckler Moving Map Display On-board Flight 36

the larger images per user specifications input at the console.

Security provided by the JBI implementation was crucial for connecting the remote user to the real-time system. The JBI uses virtual private networking with tunneling technology to allow communication connections over insecure networks. In the case of the Swathbuckler demonstration, a JBI client connected from the airplane over the internet to a JBI server at AFRL in Rome, NY allowing other users connected to the same server to communicate and receive information from the airplane with the same interface as onboard the aircraft, albeit with less bandwidth for download of imagery.

IV. Implementation

Designing a real-time HPC algorithm can be done in days—implementing it takes much longer with attention to understanding the often subtle interactions with the computing architecture. This section details significant events of that process. First, the overlap calculations reveals the upper bound time limit for azimuth processing, then the optimization chronology, the integration and flight test challenges, and finally the challenges of integrating the JBI.

A. Real-time Deadline and Overlap

There is a hard time limit by which the SAR imagery must be finished on each node. Since the azimuth overlap is thrown away, and this inefficiency grows with range, the time available for the node to complete a block (of 16K or 32K range lines) is:

$$t = (azimuth\ block — azimuth\ overlap) / PRF . \quad (1)$$

In general, as the processing range increases the overlap increases. The further out in range the larger the overlap and thus the reduction in time till the next block must be processed. For example, the first node has to finish its block in 45 seconds and the last node has to finish processing the same size block in 15 seconds. At further ranges there is more overlap, thus less forward progress, for the same size block. This is evident on Fig. 1 where the block sizes progressively decrease in size as range increases.

For this demonstration, the real-time requirement for continuous processing of 16k azimuth block with PRF at 500 results in a 15 second worst case on time between blocks. When 1 MB L2 cache became available, the inefficiency could be cut down by as much as a factor of 1.8 on the most critical nodes, by moving to 32k block sizes.

B. Algorithm Optimization

The azimuth processing was identified as the most time constrained portion by the Swathbuckler code and received the majority of the optimization effort. The first step ported the MATLAB® code to C++ and Vector Signal Image Processing Library++ (VSIPL++) [6] on a line-by-line and vector-for-vector basis. This allowed for very fine grain validation between the two codes. After initial validation, the code was instrumented by a timing algorithm to identify the slowest areas. The goal PRF of 800 necessitated that we reduce this run time down to 15 seconds for a 16k block.

The following chronology shows a timeline improvement dates, the specific computing architecture technique applied, and the impact of the change (% reduction in execution time):

- Aug 14, 2003 –Ported code was modified to run in memory. Disk input/output (IO) replaced with Random Access Memory (RAM) access. The MATLAB® implementation created intermediate data files, and the initial C++ implementation did also. Intermediate data structures used to support corner turning. (baseline)
- Aug 21, 2003 – Code modified to allow for compiler optimization options. (44%)
- Aug 27, 2003 – Various vector processing tweaks made. Moved VSIPL++ vector allocations out of loops. VSIPL++ is a package providing vector and matrix operations common in signal and image processing. Modified the VSIPL++ implementation to allocate memory at 32 byte-aligned addresses. (11%)
- Aug 28, 2003 – Added vendor optimized library functions underneath the VSIPL standard API to support azimuth compression. Improved FFTs by converting to the Spiral FFT <http://www.spiral.net/> [7]. (68%)
- Aug 29, 2003 – Eliminated redundancy in main control loops, optimized azimuth compression multiply functions and eliminate an interpolation step. (48%)
- Sept 2, 2003 – Added optimized cosine and sine calculations. (3%)
- Sept 4, 2003 – Accelerated memory functions and various minor optimizations. Eliminated unnecessary copying of FFT data. The original C++ implementation, based on the MATLAB® version, contained unneeded vector assignments. (13%)
- Feb 01, 2004 – System run in pipelined state, more platform specific optimized vector operations added, and RCMC done out of stride removing two corner turns. Bypassed corner turns, where feasible. The corner turns between range and azimuth processing and before and after RCMC are not bypassed. Implemented load balancing between range and azimuth processing. Corner turning between range and azimuth processing is part of range processing. If the range processor is waiting for the azimuth processor, the range processor performs the first FFTs in azimuth processing. (52%)

- Feb 04, 2004 – Corner turn optimizations. The need for two corner turns removed by rewriting the RCMC function to run out-of-stride and optimized to use Intel caching architecture. This resulted in the function running faster out-of-stride than in-stride. (27%)
- Feb 27, 2004 – RCMC replaced with assembly level Streaming SIMD[1] Extensions (SSE) code, memory aligned on 128 byte boundaries, azimuth compression rewritten for speed, work around to improve speed of spiral FFTs, and the select lines algorithm within auto focus was optimized. Implemented FFTs and, for example, vector math for element-by-element multiplies with Intel Performance Primitives. Begin expressing existing logic in terms of vector processing. (28%)
- Mar 2, 2004 - Rest of azimuth processing optimized by replacing the existing logic with vector operations—the auto focus portion being the most difficult. (31%)
- Mar 10, 2004 – Spiral FFTs replaced with Intel FFTs because the 16K Spiral FFT exceeded the cache size and became variable and unpredictable. (12%)
- Mar 12, 2004 – Memory movement optimized (14%)
- Mar 25, 2004 – Various optimizations taken advantage of new processing nodes with 1MB L3 cache (32%)
- Apr 01, 2005 – System running on new nodes with 1MB of L2 cache (40%). L2 much more effective than L3 cache.

Fig. 4 summarizes the results of the optimization contributions listed above in terms of reducing the overall execution. It highlights the most significant contributions to the final optimized azimuth processing of 15 seconds.

C. *Integration*

This section begins with our optimized code fully exercised in the lab facility at the Canadian radar test site. Several challenges during integration required major algorithm modifications such as changing block sizes, implementing circular buffering, and creative use of the memory address space.

1) *32k Block Upgrade*

During lab integration a change to the RF envelope moved the original start and stop swath locations 10 km further from the aircraft. The optimized algorithm met the real-time processing constraints based on a 40 km swath starting at 10 km and ending at 50 km with respect to the aircraft. The new range (20-60 km) led to more overlap thus reducing the time to the next block to a 12 second deadline.

The implementation to that time would not keep up at 500 PRF for the furthest ranges so the algorithm was changed to use 32k blocks for the furthest 16 nodes.

Two memory management problems surfaced with the new block size.

- Stage 3 (Azimuth processing) of the pipeline now required 3.9 gigabytes of addressable memory which does not leave room for critical operating system processes without causing thrashing. This is a hard limit of the 32-bit computing architecture. Moving to a 64-bit architecture and having the available memory would solve it. Instead we did in-place processing of the images thus reducing the memory footprint back to a manageable 2.6 gigabytes.
- Shared memory is the second area that needed a change to work with the 32k block size. The original shared memory implementation used an address mapping into shared memory to transfer data from stage 2 to stage 3 to stage 4. In addition to stage 3's memory address space, the total node's memory cannot exceed the installed amount (6-8 GB). When moving to 32k block size the total needs to be 11 gigabytes which exceeded resources. To overcome this, a circular buffer in shared memory was used to keep the active parts of stage 2, 3, and 4 in shared memory requiring a total of ~7 gigabytes.

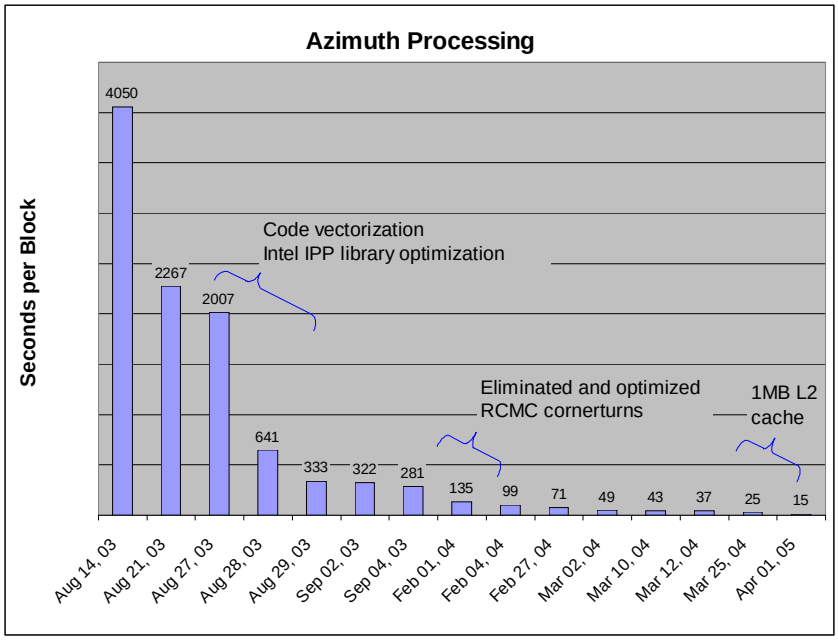

Figure 4. Azimuth Processing Reduction Timeline

[1] SIMD: Single instruction stream, multiple data streams

After regression testing the 32k block algorithm during aircraft integration, a problem with FPGA preprocessing and a desire to record IF rather than I,Q data led to a request to take on the downconversion processing. We were able to add that function to stage 1 of the pipeline.

2) Heterogeneous Configuration

To maintain the real-time processing, a mixture of block sizes and memory configurations was necessary across the embedded HPC. Even after the 32k upgrade, the last 8 nodes in range could still not keep up with 500 PRF because the overlaps were still too great.

Not yet mentioned was yet a different SAR parameter that affects processing performance. It is the presum value used in the algorithm. Changing the azimuth presum by a factor of 2 effectively divides the block size in half. This then increases the available processing time by sacrificing azimuth pixel size. Thus, for the farthest 8 nodes, the presum value was increased to 4 so those nodes could keep up. This was acceptable because with the SAR grazing angle at 3.5 degrees, the quality of returns is already degraded at those distant ranges.

Tab. 1 shows the final flight configuration of the HPC's presum value, block size, clock rate, and memory for the three groups of ranges from the aircraft. It was advantageous to put the slower nodes further out with a larger presum value because even the faster nodes with the larger block size still infrequently dropped blocks.

TABLE I. FLIGHT TEST PROCESSING CONFIGURATION

Nodes	SAR Algorithm, Memory, CPU Detail			
	Presum	***Block Size***	***CPU***	***Memory***
1-8	2	16k	3.2 GHz	4G
9-16	2	32k	3.2 GHz	8G
17-24	4	32k	3.06 GHz	6G

D. Exploitation

Establishing a working communications link proved to be the hardest implementation aspect of the exploitation. By contrast, linking the onboard and offboard JBI information management systems was straightforward. Several flights went by without any usable downlink bandwidth until the radio antenna was placed on the belly of the aircraft. With strategic placement of the ground antenna at the hexagon center, a 100 kbits/second downlink bandwidth connection was established over the license-free radio through the use of a commercial LAN Bridge [8]. A remote user 300 miles away then connected over the internet to the distributed JBI with Kerberos authentication and received current SAR imagery as it occurred during a flight. Through the course of the flight several connections were severed due to the flight path—however—the remote user was still able to continue his subscription. Due to the loosely coupled connection provided by JBI, the service hardly noticed drop outs.

In addition to the exploitation done through the JBI during flight test, further real-time insight into current HPC processing was made available to the operators onboard the aircraft. On more than one occasion it was this tool that provided the clues to prevent losing recorded raw/processed data due to, for example, a malfunctioning Ethernet cable. Being able to see the processing utilization and remaining disk space saved flying what would otherwise be a bad collection leg.

V. RESULTS

Three objectives were pursued and accomplished during 5 flights that occurred over Ottawa and Kingston, Ontario. Each flight was composed of a hexagon track around a center of interest so that each leg of the hexagon lasted about 6 minutes.

A. Real-time SAR Image Formation

The SAR image formation task was implemented on the US HPC and flew on the Convair 580. During those flights the sustained PRF hovered around 500. The actual flight test swath width was 37km instead of 40km due to slightly more overlap in range allocated for motion compensation. We processed SAR imagery across the entire swath width keeping up at 3.43 km^2/s.

B. SAR Image Exploitation

During the last flight, remote ground personnel securely connected through the JBI to the on-board database as it was being populated by the real-time processing. The user saw on his ground station within a second that a block of SAR imagery was ready for querying. He then selected a position inside the desired block and queried for a 100x100 pixel area and received the corresponding imagery for the exact longitude/latitude rectangle only 14 seconds after the request. These returned images were smaller NITF files excised from the large image blocks stored onboard the aircraft.

C. Real-time Data Recording

We recorded a total of 7.3 terabytes of raw and processed imagery. This required 375 MB/sec of raw data being copied as well as 322 MB/sec of processed imagery being written to disk. With such high data rates, each CPU on the node had 7% utilization dedicated to writing disk data.

VI. CONCLUSION

Upon arriving at the goal of processing 37 km wide swath SAR returns into images in real-time, an important lesson learned is that initial performance on modern architectures is often far below the "conventional wisdom" of 1-10% of peak derived from simply compiling code. In fact, in this case the final performance represents approximately 20% of peak. Our "conventional wisdom" needs to be updated to reflect the heightened sensitivities of the new multilayer cache architectures.

The yearly advances of processing technologies, coupled with the ability to leverage commercial web server technologies, make it first achievable, and second affordable to benefit the real-time embedded radar signal processing community in many ways. The Swathbuckler experiment shows that by careful matching of algorithm to architecture, the real-time production and exploitation of wide swath, high resolution SAR imagery is one such way.

REFERENCES

[1] The Technical Cooperation Program, Sensors Group, http://www.dtic.mil/ttcp/sen.htm, accessed September 2005.

[2] R. Linderman, "Swathbuckler: Wide Swath SAR System Architecture," *Proceedings of the 2006 IEEE Radar Conference*, in press.

[3] S. Rouse, D. Bosworth, "Swathbuckler Wide Area SAR Processing Front End," *Proceedings of the 2006 IEEE Radar Conference*, in press.

[4] A. Damini, C. Parry, G. E. Haslam, "Swathbuckler—Radar System and Signal Processing," *Proceedings of the 2006 IEEE Radar Conference*, in press.

[5] R. Linderman, M. Linderman, C.S. Lin, "FPGA Acceleration of Information Management Services, *IEEE Military Applications of Programmble Logic Devices Conference*, September 2005.

[6] CodeSourcery, LLC, VSIPL++ Specification: 1.0 candidate rev C, 2005.

[7] Puschel, M. et al., "Spiral: Code Generation for DSP Transforms," *Proceedings of the IEEE special issue on Program Generation, Optimization, and Adaptation*, Vol. 93, No. 2, Feb. 2005.

[8] Microhard Systems Inc., *SpectraNT 920 Operating Manual*, 900MHz Spread Spectrum Industrial Ethernet Bridge, Rev 0.10, 19 October 2004.