Multiparadigm Space Processing for Hyperspectral Imaging

Adam Jacobs, Chris Conger, and Alan D. George
High-performance Computing and Simulation (HCS) Research Laboratory
Department of Electrical and Computer Engineering, University of Florida
{jacobs, conger, george}@hcs.ufl.edu

Abstract—Projected demands for future space missions, where on-board sensor processing and autonomous control rapidly expand computational requirements, are outpacing technologies and trends in conventional embedded microprocessors. To achieve higher levels of performance as well as relative performance versus power consumption, new processing technologies are of increasing interest for space systems. Technologies such as reconfigurable computing based upon FPGAs and vector processing based upon SIMD processor extensions, often in tandem with conventional software processors in the form of multiparadigm computing, offer a compelling solution. This paper will explore design strategies and mappings of a Hyperspectral Imaging (HSI) classification algorithm for a mix of processing paradigms on an advanced space computing system, featuring MPI-based parallel processing with multiple PowerPC microprocessors each coupled with kernel acceleration via FPGA and/or AltiVec resources. Design of key components of HSI including auto-correlation matrix calculation, weight computation, and target detection will be discussed, and hardware/software performance tradeoffs evaluated. Additionally, several parallel-partitioning strategies will be considered for extending single-node performance to a clustered architecture. Performance factors in terms of execution time and parallel efficiency will be examined on an experimental testbed. Power consumption will be investigated, and tradeoffs between performance and power consumption analyzed. This work is part of the Dependable Multiprocessor (DM) project at Honeywell and the University of Florida, one of the four experiments in the Space Technology 8 (ST-8) mission of NASA's New Millennium Program. [1] [2]

TABLE OF CONTENTS

1

[1] 1-4244-1488-1/08/$25.00 ©2008 IEEE.
[2] IEEEAC paper #1416, Version 3, Updated December 11, 2007

1. INTRODUCTION

Many current and future space applications will generate enormous amounts of raw data which must be transmitted to the ground. Unfortunately, this communication link can only support a limited amount of data and transmission time must be shared with other satellites in orbit. One approach to alleviate the bottleneck of the downlink channel is to add additional processing capabilities to the satellite to reduce the amount of information that must be transmitted and also to allow some degree of autonomy. In order to keep up with real-time constraints, the processing capabilities of future satellites must be increased substantially. The performance of traditional space-borne computer systems is severely limited by the use of radiation-hardened components, which are necessary in order to be protected from the harsh radiation environment in which they operate. While the radiation hardening process makes the components much more resilient to both transient and permanent faults, the processors used in these systems are very expensive and lag in performance compared to modern commercial-off-the-shelf (COTS) components.

The Dependable Multiprocessor (DM) project aims to achieve reliable computation in space with the use of COTS technologies in order to provide a small, low-power supercomputer in space [1]. The DM system uses a cluster of high-performance COTS PowerPC CPUs connected via Gigabit Ethernet to obtain high-performance data processing, while employing a reliable software middleware to handle SEUs when they occur. One of the goals of the DM system is to provide a familiar software interface for developers of scientific applications through the use of standard programming tools such as C and MPI running on Linux. Fault tolerance features, such as checkpointing and replication, can be added by using API calls within a program. Additional performance can be achieved by using the on-chip AltiVec vector processing engine, or through external Field-Programmable Gate Array (FPGA) co-processors.

In a power- and space-constrained system, one possible approach to increasing performance is to use FPGAs as co-processors available to the system. These reconfigurable devices can be configured to perform almost any task in hardware, and can be modified at any time, even dynamically at application run-time. Since these devices

apply algorithms directly to hardware, they hold the potential to provide large performance gains over software execution. Additionally, through the use of reconfiguration, multiple ASICs could potentially be replaced with a single FPGA, resulting in substantial power, space, and cost savings.

One potential application that could benefit from increased processing capabilities is Hyperspectral Imaging (HSI). HSI algorithms process large amounts of raw sensor data through filters in order to classify or detect targets in the data. Raw images that are input to an HSI application can be on the order of several gigabytes of data, while the output is a much smaller list of possible targets and classifications. These algorithms are very computationally intensive, requiring both large amounts of processing and memory bandwidth. In this paper we explore several options for exploiting the parallelism of an example hyperspectral target classification application on a system similar to the DM. Parallelism at several different scales will be exploited using a variety of tools: low-level data parallelism can be exploited with AltiVec SIMD instructions; high-level data parallelism can be exploited with multiple processing nodes using Message Passing Interface (MPI); and intermediate levels of parallelism will be candidates for acceleration using FPGAs.

The remaining sections of this paper are organized as follows. Section 2 surveys previous work related to this topic. Section 3 gives an overview of the algorithm that will be used in the following analysis. Section 4 examines several parallel partitioning strategies that will be examined on the target system. Section 5 discusses the design of components that are to be used on an FPGA for application speedup. Section 6 presents results and analysis from experiments on the target platform. Section 7 discusses the power consumption of the different parallelization strategies and explores power/performance tradeoffs. Finally, Section 8 presents conclusions and provides directions for future research.

2. RELATED WORK

As mentioned previously, the goal of the DM project is to create a reliable supercomputer for space using COTS components. For space systems, the increase in computational capacity will allow for new, previously intractable problems to be solved. In previous papers, several traditional image processing and scientific applications were demonstrated running on a prototype system [1,2]. This previous work focused on single-processor performance and reliability with simple algorithm kernels.

High-performance designs of certain hyperspectral imaging applications have been explored previously. Beauchamp et al. evaluated several different HSI operations, including Principal Component Analysis (PCA) and two distinct classifiers, on an Itanium architecture machine and the effects of using optimized BLAS libraries are discussed [3]. They were able to show significant performance increases using BLAS libraries, as well as identify performance differences between two different microprocessor architectures (32-bit x86 vs. 64-bit VLIW).

Plaza et al. discuss an unsupervised hyperspectral classification algorithm that takes advantage of both spatial and spectral data that is developed and parallelized for use for cluster-based systems [4, 5]. Results are shown for an SGI Origin machine as well as for a 256-node commodity cluster. Their algorithm achieves high-quality results with a fraction of the computation requirements from other unsupervised algorithms. These unsupervised algorithms are very computationally intensive, and may exceed the abilities of modern and near-future space platforms. Supervised algorithms can substantially reduce the amount of computation required, but need certain knowledge about images being processed.

Using FPGAs as a means for hyperspectral image compression was explored by Fry et al. [6]. A Set Partitioning in Hierarchical Trees (SPIHT) compression routine was implemented in hardware in order to reduce the amount of data needed for communication in a satellite system. Such a reconfigurable system would allow a satellite system to change compression algorithms while in operation, either to adapt to processing requirements or to correct algorithmic errors. Other HSI operations, such as Independent Component Analysis (ICA) [7] and other dimension-reducing algorithms [8], have also been performed using FPGAs. These solutions usually target high-performance, high-powered machines, and may be ill-suited for space systems.

3. BACKGROUND

There are many different types of hyperspectral imaging algorithms in the literature, each algorithm having different properties that may be useful for specific scenarios. The case-study HSI algorithm adopted for this research uses a linearly constrained minimum variance (LCMV) beamforming approach, as described in [9], and has a similar computational structure to many other HSI algorithms. This approach does not require *a priori* knowledge of the environment; knowledge of the desired targets is sufficient. Therefore this algorithm would be classified as a supervised classification algorithm. Additionally, this algorithm can be designed to process full images or to process images on a line-by-line basis, allowing for real-time processing.

For the algorithm discussed in this paper, target detection and classification of hyperspectral images can be divided into three stages: metric calculation, weight computation, and target classification. The metric of interest is the autocorrelation between the spectral bands of each pixel. The autocorrelation sample matrix (ACSM) is the metric used to determine specific qualities of a given image. The equation for calculating the ACSM is shown in Equation 1, where $\boldsymbol{x}$ is a pixel vector consisting of L spectral bands, N is the total number of pixel vectors in the image, and $\boldsymbol{R}_{LxL}$ is the final autocorrelation matrix.

$$R_{L\times L} = \frac{1}{N}\sum_{i=0}^{N} x_i \times x_i^T \quad (1)$$

Once the ACSM metric is calculated it can be used, along with information about targets of interest and possible classifications, to compute the optimal weight matrix. The optimal weight calculation is shown in Equation 2, where $\boldsymbol{T}$ is the target matrix, $\boldsymbol{C}$ is the constraint matrix, and $\boldsymbol{W^*}$ is the optimal weight matrix. For more information on the target and constraint matrices, we refer the reader to [9].

$$W^* = R_{L\times L}^{-1} T (T^T R_{L\times L}^{-1} T)^{-1} C \quad (2)$$

Finally, the weights are multiplied with the original data to find the classifications, $\boldsymbol{Y}$, as shown in Equation 3. The original image data is represented by the term $\boldsymbol{X}$, which is a single two-dimensional matrix composed of all pixel vectors x_i. After classification, as a final step, simple thresholding functions are used to determine to which class, if any, each specific pixel belongs. The output of this HSI application can be a simple list of pixels containing a certain class or, by color-coding each pixel according to its classification, the output can be a complete and viewable 2-D image.

$$Y = X^T \times W^* \quad (3)$$

4. PARALLELIZATION STRATEGIES

There are two approaches to exploiting the parallelism inherent in all hyperspectral data: spatial parallelism and spectral parallelism. Spatial parallelism refers to using a data decomposition method in which whole pixel vectors are divided among processing elements, and processed independently. Many HSI algorithms do not rely on the position of a specific pixel within the larger image; any random permutation of the pixels will result in identical classifications, making this approach feasible. Spectral parallelism, in contrast, refers to dividing each pixel vector into multiple sub-pixel vectors, each with fewer spectral bands. For efficiency and practicality purposes, spatial parallelism is a much more attractive target for exploitation than spectral parallelism. Spectral parallelism is difficult to use, mostly due to the amount of communication that would be required amongst processing elements for every pixel vector.

Before creating a parallelized version of a program, it is essential to understand the properties of the original, serial execution. In the following sections the sequential HSI classification algorithm will be profiled and several possible parallelization strategies will be explored.

Serial Baseline Profiling

By profiling the serial execution of the HSI application, the major computational areas of the software can be identified and prioritized for parallelization. As discussed previously, the three major sections of the application are ACSM calculation, weight computation, and target classification. Table 1 shows a breakdown of the execution between these sections for various dataset sizes. The number of spectral bands selected was chosen to provide data for current (64, 256) and future (1,024) spectral sensor capabilities. The timing values were measured by manually placing timing functions before and after each function of interest, as well as at the beginning and end of the program. As the number of pixel vectors increases, the computation is dominated by the ACSM calculation and the remainder of computation occurs during target classification. Weight calculation time is proportional to the number of spectral bands and is not affected by image size (i.e. number of pixel vectors in the input image). As the image size grows for a fixed number of spectral bands, the significance of the weight computation stage diminishes. For target classification, the amount of computation needed is proportional to the number of distinct classes, the number of pixel vectors in the image, as well as the number of spectral bands in each pixel vector. For this analysis, eight distinct classifications were used. On average, approximately 90% of execution is spent performing the ACSM calculation and most of the remainder is in performing classification.

Table 1 — HSI Application Profile

Pixel Vectors (N)	Spectral Bands (L)	% ACSM	% Weight	% Classification
16,384	64	88	0.1	11
262,144	64	88	0.01	11
16,384	256	96	0.9	3
262,144	256	97	0.06	3
16,384	1,024	81	18	0.6
262,144	1,024	88	8	4

These results reveal that the first and last stages of this application, ACSM calculation and target classification, should be the main targets for speedup through parallelization. We will use three different technologies to provide parallel processing at different levels of granularity, including AltiVec engines, FPGA co-processors, and parallel software written using the Message Passing Interface (MPI) communication library. The following sections discuss these technologies, and how they are applied to this particular HSI algorithm, in more detail.

AltiVec Parallelism

The AltiVec engine present in the PowerPC 7455 of our next-generation testbed (and also in the 7447A used on the current DM system) uses SIMD instructions to operate on 128-bit vectors that can contain four 32-bit floating point values, four 32-bit integers, eight 16-bit integers, or sixteen 8-bit integers. Since current compilers cannot effectively create vectorized code, optimized AltiVec code usually needs to be written at a low level. In some cases, critical sections of the original, baseline code would need to be completely rewritten to take full advantage of AltiVec.

One alternative to hand-coding certain mathematical functions for SIMD support is to use Automatically Tuned Linear Algebra Software (ATLAS). ATLAS automatically determines important system parameters and builds Basic Linear Algebra Subprogram (BLAS) libraries that are optimal for that system, taking into account the memory subsystem and architectural features such as MMX, SSE, or AltiVec [10]. The BLAS library contains many functions that are used extensively in scientific programming, such as matrix and vector multiplication [11]. By using ATLAS, a simple recompilation of the original code will enable the use of AltiVec. The ACSM calculation can make use of the vector-vector functions, weight computation uses matrix-matrix multiplication, and target classification uses an optimized dot product. Using AltiVec resources, these operations can be calculated in parallel. The use of AltiVec instructions has a maximum speedup potential of 4× when using floating-point computations.

FPGA Parallelism

The ACSM and target classification sections both contain a large amount of fine-grained parallelism that can be exploited by using reconfigurable hardware. In this section, we will briefly discuss where the parallelism exists in each function. Meanwhile, Section 5 will go into detail on the actual design used for our analysis.

For a single pixel vector, there is a large amount of parallelism present in the autocorrelation calculation. As seen in Equation 1, a single L-element vector is used to generate a matrix containing L^2 elements. Using spectral data decomposition, each element of this matrix could be calculated in parallel, given enough FPGA and memory resources. This approach may be feasible for data sets whose ACSM can fit within FPGA internal memory, but is not scalable to architectures with relatively few external memory connections. This approach could be used to gain 2× or 4× parallelism on modern FPGA architectures, as many platforms have 2 or 4 independent banks of memory. Alternately, spatial parallelism in the ACSM calculation can be exploited by processing multiple pixels in parallel and the intermediate results can be reduced on-chip. The reduced data lowers memory requirements and can temporarily be stored in on-board memory, which can be accessed when the next "batch" of pixels is processed. On modern FPGAs such as the Xilinx Virtex-4 and Virtex-5 series, this approach may be able to achieve up to 16-way parallelism if using floating-point operators, and even more parallelism if fixed-point arithmetic is acceptable.

During target classification, each target class calculates its classification values independently using a common pixel vector but with differing weight values, specific to each class. Using FPGA resources, each of these classifications can be calculated in parallel. An FPGA-assisted version of target classification could potentially compute all classifications in parallel, given a large enough FPGA.

Multi-Node Parallelism

The previous parallelization strategies that were explored focused on parallelism within a single node. By connecting multiple nodes together with an interconnect such as Gigabit Ethernet we can harness their shared computational power, using MPI to communicate between nodes. Internally, each node can take advantage of the previously mentioned AltiVec and FPGA parallelization strategies. Additionally, a new level of parallelism between nodes allows for the division of work at a higher level.

The ACSM calculation, as stated previously, is simply the sum of the autocorrelations of each pixel vector in a given image. Since there is no dependence between pixel vectors, a coarse-grained, spatial data decomposition approach can be employed. In a multiprocessor system, a portion of each image can be distributed to each processor which will calculate a partial ACSM. This partial result can be collected and reduced at the end of the computation. Communication between processors only occurs when receiving data or sending results, a situation that normally allows for high parallel efficiency.

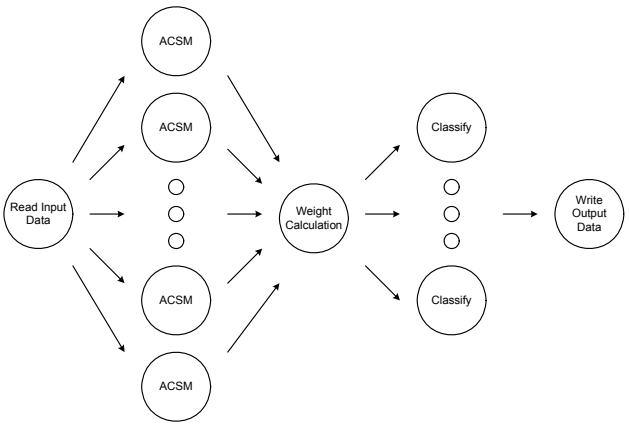

Figure 1 — Generic Parallel-Pipelined HSI Architecture

Similarly, the target classification calculations are performed on a per-pixel vector basis. Each processor in the system must have a copy of the calculated weights and a portion of the image data. A coarse-grained approach to this calculation has the same benefits as in the ACSM case.

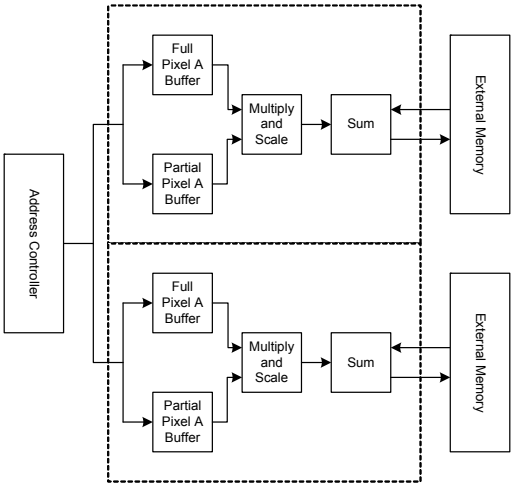

Figure 2a — Spectral Decomposition

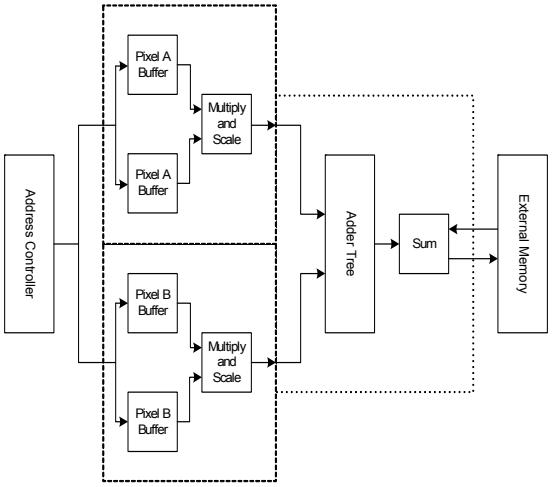

Figure 2b — Spatial Decomposition

If the same nodes are used for both ACSM and target classification, the original data can be sent during the ACSM stage and reused during classification, limiting the required amount of communication.

Another approach for obtaining parallelism is the use of a software functional pipeline. With this method, it is possible to decrease the average latency of completing images. This reduction of average execution time is accomplished by partitioning the compute nodes into groups, so that each group performs certain specific operations, and then passes the resulting data to the next group of nodes, which perform the next function. This approach can be used to adapt the algorithm to meet certain real-time deadlines that may be needed for specific missions. Additionally, dedicated nodes can be used for I/O in order to avoid performance penalties associated with disk access. A pipelined approach achieves the most benefit when each pipeline stage performs an equal amount of work, a situation that does not exist with this application. However, it would be possible to exploit enough data parallelism in the ACSM calculation that the load balancing between stages becomes more favorable for a pipeline parallelization, as shown in Figure 1. Each stage in the parallel-pipeline approach may contain any number of nodes, indicated as circles in Figure 1, and data must be divided or combined as it is passed from one stage to the next.

5. FPGA CORE DESIGN

Taking these parallelization strategies into account, we can design one or more FPGA cores that exploit the low-level parallelism of the ACSM and target classification functions. In this section, we will present several different hardware architectures and will perform a brief analysis to verify the amenability of the hardware options on the target platform.

The Alpha Data ADM-XRC-4 FPGA boards used for this work each consist of a Xilinx Virtex4-SX55 FPGA along with 16MB of SRAM split into four separate banks. The board connects to the remainder of the system through the 32-bit PCI bus running at 33 MHz. The Xilinx Virtex4 has a very limited amount of on-chip memory (approximately 6 Mbits total) and the ADM-XRC-4's SRAM is also limited, especially considering the normal dataset sizes for most HSI applications. However, the memory should be used as much as possible because the number of data transfers from the CPU to the FPGA should be minimized to alleviate the bottleneck presented by the PCI bus. Transferring data across the PCI bus is much more efficient for a few, large segments than it is for many small segments. Intermediate results should be stored in the on-board memory, instead of being communicated back to the host processor in order to avoid unnecessary communication.

ACSM Hardware Core

As discussed in the previous section, there are two possible approaches to exploiting the parallelism of this algorithm. Figures 2a and 2b show general block diagrams for each design.

Figure 2a shows a spectral decomposition using two processing elements in parallel, each creating one half of the intermediate result matrix. Each processing element needs access to store its intermediate values in external memory. Because the ADM-XRC platform does not allow concurrent reads and writes to a single bank of memory, each processing element requires access to two banks of external memory. In the first pass one bank is read, while the other is written; on alternate passes it is reversed. The ADM-XRC platform limits the total parallelism available with this approach to 2× due to the number of independent memory banks available. In Figure 2b, there are again two processing elements. In this case, a spatial decomposition approach is used where each processing element calculates an entire intermediate matrix. Results from each intermediate value are summed before being written to the on-board memory. This method only requires a single memory bank, but additional FPGA resources are used to

construct the adder tree. While two processing elements are shown here, this approach can be scaled to the size of the FPGA in use.

The input data is 16-bit integer data from an image sensor or a preprocessing stage. Fortunately, this numerical format allows for the use of integer multiplication and addition for the majority of the hardware. Referring to Figure 2b, the multiplication, scaling, and adder tree all use integer arithmetic to reduce the amount of logic resources required. Before writing data values to temporary storage (or back to main memory), the fixed-point result is converted into single-precision floating point format. There are two benefits from using this approach. First, only two floating-point operations are required in the design, fixed-to-floating-point conversion and addition. Second, processing on a traditional microprocessor would result in floating-point numbers; therefore, the results are already in the correct format for subsequent processing steps. An additional benefit of converting to floating point after the adder tree is better preservation of precision than compared to adding each result individually in floating point.

Additional computational savings can be obtained by taking advantage of the symmetric nature of the autocorrelation matrix. Instead of calculating the entire matrix, only the upper triangular portion is truly needed. At the end of the computation, the lower triangular portion can be filled in, if desired. This approach leads to almost 2× savings in computation, but it complicates memory addressing and load balancing between processing elements when using a spectral decomposition. The necessary modifications are straightforward for the spatial decomposition approach.

Before implementing a design in hardware, some rough estimates of performance can be gathered. By using the RC Amenability Test [12], we can get an estimate of performance based on communication and computation ratios in a given algorithm. This simple test requires information about the I/O capabilities and requirements of the system, as well as rough estimates of throughput of the hardware design. Figure 3 shows the results from using this analysis for a 64-core, spatial decomposition hardware design targeting 1,024 spectral bands. A small amount of speedup was observed with as few as four hardware cores, and with 64 cores, a speedup of 24× should be possible. Additional improvement can be achieved by using double buffering (i.e. transfer the next set of data while the current set is still processing). The bandwidth limitations of the PCI bus do not significantly limit the performance of this hardware design. Based on the performance estimates, and given a large enough FPGA, the PCI bus should be able to sustain a 512-core design. An important trend that can be seen using this analysis is that the hardware performance advantage over traditional software processing grows as the number of spectral bands is increased; speedup may be difficult to achieve when using less than 64 spectral bands.

Target Classification Core

The Virtex4-SX55 has a large number of built-in 18×18 multiply-and-accumulate blocks available for use. In fact, if the data and weight information were to be presented as 18-bit fixed-point numbers, it would be possible to classify up to 512 classes in parallel. (Due to routing difficulties, this

Constants/User-Defined Parameters

Communication Parameters

throughput(ideal)	(MB/s)	133	PCI-X, PCI-Express, etc…
α(input)	0<α<1	0.75	Throughput efficiency to FPGA
α(output)	0<α<1	0.75	Throughput efficiency from FPGA
# of Input Elements	#	1048576	# of elements written to FPGA
# of Output Elements	#	1048576	# of elements read from FPGA
Bytes per element	(B)	4	precision

Software Parameters

t(soft)	(sec)	146.381	all SW, RC-ported section only
N	(iterations)	8	# calls to FPGA for HW-assisted app

Constants/User-Defined Parameters

Computation Parameters

# of Comp Elements	#	2048	# of elements of computation
Ops per element	(ops/elem)	1048576	related to order of complexity
throughput(proc)	(ops/cycle)	64	# operations performed per cycle
f(clock)	(MHz)	50	clock frequency of FPGA core

Calculated Sub-metrics (in seconds)

	Double Buffered	*Single Buffered*	
t(soft)	1.464E+02	<= (same)	original software execution time
t(comm)	8.410E-02	<= (same)	time to send & recv data to/from FPGA
t(comp)	6.711E-01	<= (same)	time to process one input buffer of data
t(RC)	5.369E+00	6.041E+00	estimated FPGA execution time

Calculated Key Metrics

	Double Buffered	*Single Buffered*
util(proc)	100.0%	88.9%
util(IO)	12.5%	11.1%
speedup(kernel)	27.3	24.2

Figure 3 — RC Amenability Test for ACSM Core

Constants/User-Defined Parameters

Communication Parameters

throughput(ideal)	(MB/s)	500	PCI-X, PCI-Express, etc…
α(input)	0<α<1	0.75	Throughput efficiency to FPGA
α(output)	0<α<1	0.75	Throughput efficiency from FPGA
# of Input Elements	#	1048576	# of elements written to FPGA
# of Output Elements	#	16384	# of elements read from FPGA
Bytes per element	(B)	4	precision

Software Parameters

t(soft)	(sec)	13.854	all SW, RC-ported section only
N	(iterations)	128	# calls to FPGA for HW-assisted app

Constants/User-Defined Parameters

Computation Method A

# of Comp Elements	#	2048	# of elements of computation
Ops per element	(ops/elem)	8192	related to order of complexity
throughput(proc)	(ops/cycle)	8	# operations performed per cycle
f(clock)	(MHz)	50	clock frequency of FPGA core

Calculated Sub-metrics (in seconds)

	Double Buffered	*Single Buffered*	
t(soft)	1.385E+01	<= (same)	original software execution time
t(comm)	1.136E-02	<= (same)	time to send & recv data to/from FPGA
t(comp)	4.194E-02	<= (same)	time to process one input buffer of data
t(RC)	5.369E+00	6.823E+00	estimated FPGA execution time

Calculated Key Metrics

	Double Buffered	*Single Buffered*
util(proc)	100.0%	78.7%
util(IO)	27.1%	21.3%
speedup(kernel)	2.6	2.0

Figure 4 — RC Amenability Test for Classification Core

number would not truly be achievable.) The weight vectors are loaded once for each image at startup. When computation begins, pixel vectors are loaded onto the FPGA, and the classifications are stored in on-board memory. Each classification is calculated independently, with each hardware core containing one multiply-accumulate block and one weight buffer. A pixel buffer containing the current pixel vector being classified is shared among every core.

When operating on a traditional microprocessor, target classification would normally work on floating-point numbers. If an FPGA design uses fixed-point arithmetic, an analysis on the precision effects would need to be performed. Conversion from floating-point precision to fixed-point precision can be handled easily on the FPGA, but prior knowledge of typical weight ranges can ensure acceptable amounts of precision loss. A fully floating-point target classification design would not scale well due to the limited resources on the FPGA.

Again, we can look at some performance estimates before spending resources on constructing a hardware design. The results of the target classification amenability test are shown in Figure 4. Unlike the previous example, target classification is not likely to see significant gains from using a hardware-specific core. In this case, the I/O is a large bottleneck and adding additional hardware resources would have no effect on performance. A throughput of approximately 500 MB/s from the host processor to the FPGA would be needed to achieve a 2× speedup over the AltiVec-optimized target classification stage. As a result, we will not migrate the target classification stage to an FPGA co-processor for our experimental analyses.

6. Experimental Results and Analysis

This section will combine and analyze the approaches used in Sections 4 and 5, by discussing the complete architecture of our experimental platform and presenting performance results obtained by executing and analyzing the case study HSI application with various levels of parallelism.

Experimental Testbed

A 10-node PowerPC cluster was used to gather results. Each node contains a 1.4 GHz PowerPC 7455 with AltiVec and 1GB of SDRAM. The nodes are connected together with Gigabit Ethernet. Additionally, four nodes are equipped with ADM-XRC-4 FPGA boards from Alpha Data, each containing a Xilinx Virtex4-SX55 FPGA and four independent 4MB SRAM modules. This testbed is intended to simulate a possible "next-generation" Dependable Multiprocessor system. The major differences between the testbed and the current DM system are a faster CPU (1.4GHz vs. 1.0GHz) and the use of an FPGA coprocessor.

AltiVec Performance Results

Using ATLAS 3.6.0 compiled for the testbed system, the ACSM calculation, weight computation, and classification function calls were replaced with their BLAS equivalents. Table 2 shows the comparison of the full application performance using AltiVec-enabled code versus the original baseline. Through the use of cache prefetching, AltiVec instructions, and other optimizations, the program using the ATLAS libraries was able to decrease execution time by at least 80% for a range of image parameters. Optimizations that account for symmetry in the autocorrelation matrix calculation were not used in either the baseline or the AltiVec version. The maximum amount of speedup occurs when processing images with 256 spectral bands, corresponding to the largest tested matrix size that is capable of fitting in the L2 cache of a PowerPC 7455 processor.

Table 2 — Full, HSI Speedup: AltiVec versus Baseline

Spectral Bands	Baseline (s)	AltiVec (s)	Speedup
64	43.34	6.70	6.47
256	624.83	85.47	7.31
1024	10634.35	1964.11	5.41

FPGA Performance Results

Table 3 shows the amount of speedup that is achievable during the ACSM calculation when using an FPGA. The results were gathered from three 512×512 images, each with a differing number of spectral bands. The performance of a single FPGA-assisted node was compared to the AltiVec-enabled execution times for ACSM portion of the application. As we expected from the analysis in the previous section, images with more spectral bands experience a larger benefit from the hardware solution, reaching a 20× improvement when processing images with 1,024 spectral bands. For images that only contain a few spectral bands, the AltiVec-enabled version has the fastest execution.

For full application performance on a single node, the FPGA-assisted version of HSI will use the FPGA for ACSM calculations while the weight computation and target classification functions will be identical to those used by the AltiVec-enabled version. For subsequent results, we will focus on images with either 256 or 1,024 spectral bands. The former value is approximately the number of spectral bands available on current sensors, such as AVIRIS or the Hyperion sensor on EO-1 [13, 14]. The larger hyperspectral image will be used to show trends for possible future sensors.

Table 3 — ACSM Speedup: FPGA versus AltiVec

Spectral Bands	AltiVec (s)	FPGA (s)	ACSM Speedup	Full HSI Speedup
64	5.55	1.79	3.10	2.27
256	81.14	7.17	11.31	7.48
1024	1917.22	94.46	20.30	14.23

Multi-Node Performance Results

The HSI algorithm was parallelized with MPI using the spatial data decomposition method described in Section 4. The ACSM calculation and target classification stages were parallelized such that each node processes an independent portion of the original image, and the results are returned to a master node. Weight computation is performed sequentially on the single master node, after which the weights are broadcasted to all nodes before the beginning of the target classification stage.

Figure 5 shows the results of this parallelization using both the AltiVec-enabled software, as well as the FPGA-assisted version for the entire HSI application. In this experiment, the image is a 512×512 image with either 256 or 1,024 spectral bands. The 10-node cluster using the AltiVec-enabled HSI code was able to complete a 256-spectral-band image in 14 seconds and a 1,024-spectral-band image in 272 seconds, for speedups of 6 and 7.2 over the single-node performance, respectively. The 4-node FPGA-assisted system can finish a 256 spectral band image in 6 seconds and a 1,024 spectral band image in 60 seconds, a speedup of 13 and 26, respectively, over a single AltiVec-enabled node.

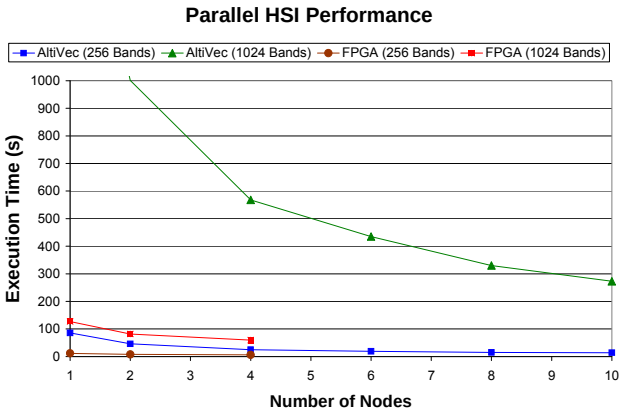

Figure 5 — Parallel System Performance (512×512 Image)

For the AltiVec-enabled version of the application, the scalability of the parallel code is very good for small node sizes. At two or four nodes, the program is able to achieve approximately 90% parallel efficiency. However, as the number of nodes scales higher the parallel efficiency, shown in Figure 6, begins to drop; 10-node performance achieves between 60% and 70% efficiency. As the number of nodes in the system increases, the computation-to-communication ratio begins to decrease, and the effects of the network start to impact performance.

For the FPGA-assisted version of the full HSI application, the single-node baseline executes 14 times faster than the AltiVec-enabled version, when using 1,024 spectral bands. This fact has the effect of changing the application profile so that the weight computation stage, which was not parallelized, is no longer an insignificant portion of the application. For example, weight computation takes approximately 22 seconds for an image using 1,024 spectral bands, accounting for 16% of the total execution time. For that reason, the parallel efficiency of the hardware-accelerated version is significantly lower than the original software version. Additionally, since the FPGAs complete the ACSM computation so quickly, the communication time using Gigabit Ethernet becomes a significant performance factor. A system using all four FPGA-equipped nodes in parallel achieves less than 60% parallel efficiency. Figure 6 shows the parallel efficiency for both the AltiVec-enabled and FPGA-assisted applications. In order to significantly improve the efficiency of the parallel algorithm, it will be necessary to parallelize the weight computation section of the application.

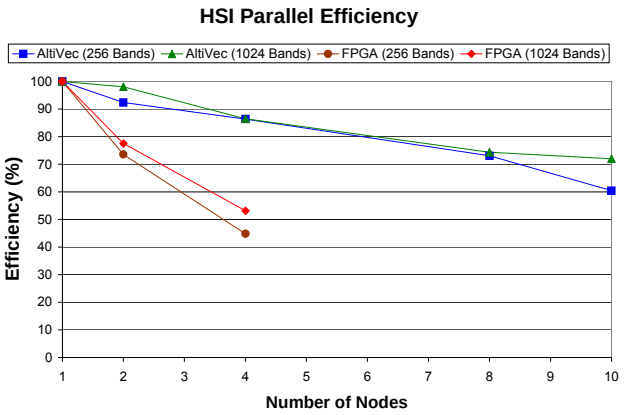

Figure 6 — Parallel Efficiency (512×512 Image)

7. POWER CONSUMPTION

Power consumption is a significant design criterion for embedded systems, and space systems have a particularly strict power envelope. This section will explore the power consumption of the systems previously discussed. Table 4 shows the power consumption of several important components used in this analysis. The FPGA power consumption was gathered using the Xilinx XPower Estimator spreadsheet, with an average toggle rate of 25%. The other power values were collected from the appropriate datasheets. When fully utilized, the ADM-XRC-4 FPGA board, including the on-board SRAM, will consume approximately 8W. On the other hand, the maximum power consumption of the PowerPC 7455 is 45W. Other contributors to power consumption, such as the network interconnect or system memory, could not be reliably measured and are not included in the following analysis.

For the single-node AltiVec-enabled version of the HSI application, we will assume that the processor reaches its maximum power consumption, 45 Watts, since all sections of the code are computationally intensive. During weight computation on a multi-node system, only one processor

Table 4 — Power Consumption Values

Variable	Description	Value
P_{CPUmax}	Maximum CPU power consumption	45W
P_{CPUtyp}	Typical CPU power consumption	34W
P_{FPGAon}	FPGA power consumption during computation	8.0W
$P_{FPGAoff}$	FPGA power consumption while idle	7.1W

will consume maximum power, while the other processors consume less power as they wait for results. The average power consumption of the AltiVec-enabled system is given by Equations 4 through 7, where N is the number of nodes in the system and t_{ACSM}, t_{WC}, and t_{TC} are the execution times of the ACSM, weight computation, and target classification stages, respectively. The power consumptions during each major function, P_{ACSM}, P_{WC}, and P_{TC}, are estimated, and those values are applied proportionally to estimate the average power, P_{avg}, of the system.

$$P_{\text{ACSM}} = N \cdot P_{\text{CPUmax}} \tag{4}$$

$$P_{\text{WC}} = P_{\text{CPUmax}} + (N-1) \cdot P_{\text{CPUtyp}} \tag{5}$$

$$P_{\text{TC}} = N \cdot P_{\text{CPUmax}} \tag{6}$$

$$P_{\text{avg}} = \frac{P_{\text{ACSM}} \cdot t_{\text{ACSM}} + P_{\text{WC}} \cdot t_{\text{WC}} + P_{\text{TC}} \cdot t_{\text{TC}}}{t_{\text{ACSM}} + t_{\text{WC}} + t_{\text{TC}}} \tag{7}$$

For the FPGA-assisted version of the application, we will assume the CPU consumes only 34 Watts (typical) when the FPGA is being utilized, and the FPGA is in idle mode when the CPU is performing weight computation or target classification. The average power consumption of a node is determined by Equation 7, shown above, and Equations 8 through 10, shown below. During the ACSM calculation, each node consumes maximum power from the FPGA and typical power from the CPU. During weight computation, the CPU on one node consumes maximum power while all other devices consume their typical values. During target classification, each CPU is assumed to consume maximum power while the FPGAs consume their idle values. The average power of each system is determined by the percentage of time spent in each function.

$$P_{\text{ACSM}} = N \cdot \left(P_{\text{FPGAon}} + P_{\text{CPUtyp}}\right) \tag{8}$$

$$P_{\text{WC}} = P_{\text{CPUmax}} + (N-1) \cdot P_{\text{CPUtyp}} + N \cdot P_{\text{FPGAoff}} \tag{9}$$

$$P_{\text{TC}} = N \cdot \left(P_{\text{CPUmax}} + P_{\text{FPGAoff}}\right) \tag{10}$$

To compare algorithms, we will use an energy metric, power multiplied by time. Table 5 augments the parallel performance data with estimates of average power for each system, using the previous equations. The most energy-efficient system is the single-node, FPGA-assisted system. The FPGA not only provides a 15× speedup, but the combined power consumption of the FPGA and the CPU in a low computational scenario ($P_{FPGAon} + P_{CPUtyp}$) is less than the maximum power consumption of the CPU (P_{CPUmax}). As more nodes are added to the system, shifting performance bottlenecks to non-parallelized sections of code, the additional CPUs will be unused for longer periods of time consuming a reduced amount of power. While the energy usage of the FPGA-assisted system is much lower than the AltiVec-enabled system, the difference in power consumption is small, less than 5% of the total system power consumption. The reduction in energy consumption comes directly from the reduced execution time for FPGA-assisted systems.

8. CONCLUSIONS

The use of non-traditional processing resources such as FPGAs or AltiVec engines is an effective method for increasing performance in systems where computational power is the largest concern. Using a few simple profiling and estimation techniques on an original sequential program, candidates for acceleration are easily determined. The ACSM calculation, accelerated using an FPGA co-processor, was able to achieve a 15× speedup. While FPGAs are capable of large performance gains in many situations, I/O bandwidth can limit their potential, as seen with target classification, which was not able to achieve a speedup with the current platform's bandwidth capabilities. By optimally combining FPGA and AltiVec resources to create a multi-paradigm HSI application, single-processor performance could be improved by as much as two orders of magnitude over the original software baseline. Additional performance gains were possible with multiple nodes communicating using MPI, achieving a 33× speedup over a single AltiVec-enabled node using four FPGA-equipped nodes. The major factor limiting parallel

Table 5 — Parallel System Performance and Power Estimates

	AltiVec-Enabled				FPGA-Assisted			
Nodes	Execution Time (s)	Speedup	Avg. Power (W)	Energy (kJ)	Execution Time (s)	Speedup	Avg. Power (W)	Energy (kJ)
1	1964.11	1.00	45.00	88.4	127.00	15.46	44.39	5.6
2	1001.16	1.96	89.76	89.9	81.90	23.98	87.20	7.1
4	567.73	3.46	178.75	101.5	59.76	32.86	172.51	10.3
6	434.92	4.52	267.14	116.2				
8	330.16	5.95	354.82	117.1				
10	272.71	7.20	441.90	120.5				

scalability was the sequential nature of the weight computation stage. The average power consumption of an FPGA-assisted node was estimated to be almost equivalent to a non-FPGA node (45W), due to the FPGA power consumption offsetting the reduced amount of power consumed by the CPU while idle.

Future work may explore methods for efficiently parallelizing the weight computation section, which involves a large matrix inversion and several small matrix multiplications. Additionally, we plan to explore the use of algorithm-based fault tolerance (ABFT) in this HSI algorithm. ABFT is an efficient method of protecting against data corruption, which can be useful for high-altitude or space missions where single-event upsets are likely.

ACKNOWLEDGMENTS

This work was supported in part by the NMP Program at NASA, our Dependable Multiprocessor project partners at Honeywell Inc., and the Florida High-Technology Corridor Council. The authors would like to thank Casey Reardon and Eric Grobelny of the HCS Lab for their support in writing this paper.

REFERENCES

[1] J. Samson, J. Ramos, I. Troxel, R. Subramaniyan, A. Jacobs, J. Greco, G. Cieslewski, J. Curreri, M. Fischer, E. Grobelny, A. George, V. Aggarwal, M. Patel, and R. Some, "High-Performance, Dependable Multiprocessor," *Proc. of IEEE Aerospace Conference*, Big Sky, MT, March 4-11, 2006.

[2] J. Greco, G. Cieslewski, A. Jacobs, I. Troxel, and A. George, "Hardware/Software Interface for High-Performance Space Computng with FPGA Coprocessors," *Proc. of IEEE Aerospace Conference*, Big Sky, MT, March 4-11, 2006.

[3] W. Lugo-Beauchamp, K. Cruz, C. Carvajal-Jimenez, and W. Rivera, "Performance of Hyperspectral Imaging Algorithms Using Itanium Architecture," *Proc. of IASTED International Conference*, pp. 327-332, November 2004.

[4] A. Plaza, D. Valencia, J. Plaza, and C. Chang, "Parallel Implementation of Endmember Extraction Algorithms from Hyperspectral Data," *IEEE Geoscience and Remote Sensing Letters*, Vol. 3, no. 3, pp. 334-338, July 2006.

[5] A. Plaza, D. Valencia, J. Plaza, and P. Martinez, "Commodity cluster-based parallel processing of hyperspectral imagery," *Journal of Parallel and Distributed Computing*, no. 66, pp. 345-358, 2006.

[6] T. Fry, and S. Hauck, "Hyperspectral Image Compression on Reconfigurable Platforms," *Proc. of 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 251-260, 2002.

[7] H. Du, H. Qi, "A Reconfigurable FPGA System for Parallel Independent Component Analysis," *EURASIP Journal on Embedded Systems*, Vol. 2006, Article ID 23025, 12 pages, 2006.

[8] E. El-Araby, T. El-Ghazawi, J. Le Moigne, and K. Gaj, "Wavelet spectral dimension reduction of hyperspectral imagery on a reconfigurable computer," *Proc. of IEEE International Conference on Field-Programmable Technology*, pp. 399-402, Dec. 6-8, 2004.

[9] C. Chang, H. Ren, and S. Chiang, "Real-time processing algorithms for target detection and classification in hyperspectral imagery," *IEEE Transactions on Geoscience and Remote Sensing*, Vol. 39, no. 4, pp. 760-768, April 2001.

[10] R. Whaley, A. Petitet, and J. Dongarra, "Automated Empirical Optimization of Software and the ATLAS Project," *Parallel Computing*, Vol. 27, no. 1-2, pp. 3-25, 2001.

[11] C. Lawson, R. Hanso, D. Kincaid, and F. Krogh, "Basic Linear Algebra Subprograms for Fortran Usage," *ACM Transactions on Mathematical Software (TOMS)*, Vol. 5, issue 3, pp. 308-323, September 1979.

[12] B. Holland, K. Nagarajan, C. Conger, A. Jacobs, and A. George, "RAT: A Methodology for Predicting Performance in Application Design Migration to FPGAs," *First International Workshop on High-Performance Reconfigurable Computing Technology and Applications (HPRCTA)*, Supercomputing Conference (SC), Reno, NV, Nov. 11, 2007.

[13] R. Green, M. Eastwood, C. Sarture, T. Chrien, M. Aronsson, B. Chippendale, J. Faust, B. Pavri, C. Chovit, M. Solis, M. Olah, O. Williams, "Imaging spectroscopy and the Airborne Visible/Infrared Imaging Spectrometer (AVIRIS)," *Remote Sensing of Environment*, Vol. 65, no. 3, pp. 227-248, September 1998.

[14] S. Ungar, J. Pearlman, J. Mendenhall, and D. Reuter, "Overview of the Earth Observing One (EO-1) mission," *IEEE Transactions on Geoscience and Remote Sensing*, Vol. 41, issue 6, pp. 1149-1159, June 2003.

BIOGRAPHY

Adam Jacobs *is a Ph.D. student in Electrical and Computer Engineering at the University of Florida. He is a research assistant in the Advanced Space Computing group at the High-Performance Computing and Simulation Research Laboratory. His research interests include fault-tolerant FPGA architectures and high-performance, multi-paradigm computing. He is a student member of the IEEE.*

Chris Conger *is a Ph.D. student in Electrical and Computer Engineering at the University of Florida. He is a research assistant with the NSF Center for High-performance Reconfigurable Computing (CHREC), investigating dynamic partial reconfiguration and reconfigurable fault tolerance in FPGA systems. His research interests include reconfigurable and fault-tolerant embedded computing, space systems, and defense applications.*

Alan D. George *is Professor of Electrical and Computer Engineering at the University of Florida, where he serves as Director of the HCS Research Lab and Director of the new NSF Center for High-performance Reconfigurable Computing (CHREC). He received the B.S. degree in Computer Science and the M.S. in Electrical and Computer Engineering from the University of Central Florida, and the Ph.D. in Computer Science from the Florida State University. Dr. George's research interests focus upon high-performance architectures, networks, systems, and applications in reconfigurable, parallel, distributed, and fault-tolerant computing. He is a senior member of IEEE and SCS, a member of ACM and AIAA, and can be reached by e-mail at ageorge@ufl.edu.*