Benchmarking Image Processing for Space: Introducing the SPACER Architecture Laboratory

Andrew C. Pineda, Jesse K. Mee
U.S. Air Force Research Laboratory
Space Electronics Branch
Space Vehicles Directorate
3550 Aberdeen Avenue SE
Kirtland AFB, NM 87117
505-853-2509, 505-846-3749
afrl.rvse@us.af.mil

Phillip M. Cunio, Reed A. Weber
U.S. Air Force Research Laboratory
Battlespace Environment Division
Space Vehicles Directorate
3550 Aberdeen Avenue SE
Kirtland AFB, NM 87117
505-846-0057, 505-853-4130
afrl.rvby@us.af.mil

Abstract — In order to better make investment decisions for future space processing, we are equipping an architectures laboratory to investigate the power and computing performance of candidate computing architectures for future space applications. The picture for future space processing is increasingly complicated by ever increasing data rates/sizes and limited communications bandwidth, both of which will require more data processing, in the form of either data reduction or compression, to be performed on orbit rather than on the ground. Candidate architectures for the laboratory are being drawn from a range of COTS processing architectures including low power multicore processors, FPGAs, and GPUs. As the drivers for these investments are likely to be data-intensive image processing applications, we have selected two representative applications, Synthetic Aperture Radar (SAR) and Hyper-Temporal Imaging (HTI), and tested them on a variety of low-power multicore processors, and for comparison, on modern conventional processors. Both applications were parallelized using OpenMP and/or pthreads. The processors employed include from four to eight cores. State-of-the-art numerical libraries were used to extract the most performance possible. The multi-core processors selected included examples of both homogeneous and heterogeneous computing architectures. Effects of varying the parameters such as the amount of memory made available to the processors, which affects how data decomposition is accomplished, are also studied. In general, homogeneous computing architectures performed better than heterogeneous ones. In some cases, better performance could be achieved with a single processor core with large memory than with multiple processors. These results are a function of the employed algorithm's ability to efficiently utilize architecture features, and cannot be attributed to all application/architecture pairings, thus highlighting the need for a concerted effort to explore processing requirements for future space missions.

TABLE OF CONTENTS

1. INTRODUCTION.....	1
2. PROCESSORS STUDIED	2
3. APPLICATIONS STUDIED	2
4. RESULTS AND ANALYSIS.....	4
5. SUMMARY	8
ACKNOWLEDGEMENTS	8
REFERENCES.....	8
BIOGRAPHY	9

1. INTRODUCTION

The problem of predicting requirements for future space processors is a challenging one. The long lead times required for hardening microprocessors against the space radiation environment and the power constraints imposed by operating on a satellite mean that currently available space qualified processors lag modern terrestrial processing hardware by several generations in performance. In terrestrial systems, this performance improvement over space systems is due to many factors. Owing to traditional Moore's Law scaling of processor feature size and clock rates, terrestrial processors currently enjoy a roughly 100X clock speed advantage. [1,2] Further complicating matters is the greatly increased diversity of processor features and architectures. Single processor cores now feature more execution units – e.g. integer units, floating point units, Single Instruction-Multiple Data (SIMD) units. Other advancements such as improved pipelining of instructions wring out greater performance per clock cycle. Multiple cores are now combined in a single processor. These multiple cores can either be homogeneous or heterogeneous as in the case of modern implementations of the ARM architecture. Going beyond “conventional” microprocessors, a number of newer architectures that attempt to exploit parallelism at a variety of levels have found application in various high performance computing applications. These range from very fine-grained parallelism that can be exploited by field programmable gate arrays (FPGAs) to graphical processing units (GPUs). The difficulty in determining which of these solutions will work best over a wide range of space processing applications is that their performance is highly dependent on how well the algorithms employed map to those architectures. This has led us to set up a laboratory to study the performance of applications of importance to the Air Force on a variety of processor architectures.

In this paper, we report on recent efforts at benchmarking the performance of data intensive image processing applications, namely Synthetic Aperture Radar (SAR) and Hyper-Temporal Imaging (HTI), on a range of low power ARM-based computing platforms. These ARM platforms include a number of architectural features that we hope to investigate, including 32-bit and 64-bit implementations,

homogeneous and heterogeneous computing features (such as big.LITTLE architectures), and GPU capability. Our reference applications have been implemented using the best available numerical libraries for each platform, and parallelized using threaded and message passing implementations. We report parallel speedups and relative performance for these applications, and compare performance with conventional multi-core architectures.

In section 2, the processors included in this study are described in detail.

In section 3, the applications included in this study are described. The parallelization strategies and the libraries used and their configurations are also described.

In section 4, the parallel performances of these applications are analyzed using generalizations of Amdahl's law.

2. PROCESSORS STUDIED

In this paper, we report results obtained using the ODROID-XU3 development board, the NVidia Jetson TK1 development board, and for reference, a DELL T7500 workstation. The development boards are examples of heterogeneous and homogeneous computing architectures based on the ARM processor.

The ODROID-XU3 development board is based on the Samsung Exynos5422 System on a Chip. This chip includes a quad core ARM Cortex-A15 cluster operating at 2.1 GHz and a quad core ARM Cortex-A7 cluster operating at 1.5GHz. The A7 and A15 cores are binary compatible. They differ in clock rate and in the structure of the instruction processing pipeline. In normal low power operation, tasks run on the A7 cores, while for computing-intensive tasks the faster, more power-hungry A15 cores are used. The board contains 2GB of 32-bit dual channel LPDDR3 RAM operating at 933 MHz (14.9 GB/s memory bandwidth). Storage includes 64GB of eMMC5.0 HS400 Flash Storage pre-installed with Ubuntu 14.04LTS.

The NVidia Jetson TK1 development board is based on the NVidia Tegra K1 System on a Chip. This chip includes a quad core ARM Cortex-A15 and a Tegra K1 GPU. The GPU contains 192 SM3.2 CUDA cores delivering 326 GFlops. This board includes 2GB DDR3L 933MHz EMC x16 DRAM using 64-bit data width. Storage includes 16 GB of eMMC4.51 flash memory and a 16 GB SD card slot. The flash memory is installed with Ubuntu Linux 14.04LTS.

The performance of these boards is compared against a DELL 7500 Workstation containing dual quad-core Intel Xeon X5677 CPUs operating at 3.47GHz and 48 GB of Samsung 1333 MHz DDR3 ECC Registered DIMM Memory. The workstation runs Ubuntu Linux version 12.04LTS. Prior to the tests in Figure 7, our test workstation had been upgraded to Ubuntu Linux version 14.04LTS.

The GCC compiler version included with Ubuntu Linux version 14.04LTS is version 4.8.4 which supports the OpenMP 3.1 specification. Ubuntu 12.04LTS includes the 4.6.3 GCC compiler which supports version 3.0 of the OpenMP specification.

Several other development boards were not received in time for inclusion in this paper. These include a 64-bit Juno ARM development board and a Xilinx Zynq-7000 based Mini-ITX development kit. Likewise, the CUDA GPU (graphics processing unit) compute capability of the NVidia Jetson TK1 is not studied in this paper as we did not have time to rewrite and validate either of our benchmark codes. We expect to include these boards and capabilities in future work.

3. APPLICATIONS STUDIED

A SAR application implementing the basic range compression, azimuth transformation, range migration and azimuth compression steps of the SAR algorithm was written by Chris Conger and Adam Jacobs of the National Science Foundation (NSF) Center for High-Performance Reconfigurable Computing (CHREC).[3-6] The CHREC consortium includes partners from academia, industry and government. The original codes included a serial version and an MPI-based (MPI=Message Passing Interface) version implementing a crude master-slave domain decomposition algorithm on top of the SAR algorithm used in the serial code. Fast Fourier Transforms (FFTs) from the GNU Scientific Library (GSL) are used.[7] The codes were based on a sequential SAR application developed by David Sandwell and Evelyn Price of the Scripps Institution of Oceanography.[8]

In the original CHREC MPI implementation, the master processor just distributed data to the slave processors; we changed this to allow the master processor to share in the computational workload. This was critical because of the small number of processors on our test boards. Figure 1 illustrates, for the case of a quad-core environment, how the data distribution and SAR processing are performed in our parallel benchmark. The distributed memory message passing implementation was converted to a shared memory implementation using OpenMP threading. On our reference platform the MPI and OpenMP implementations were found to perform identically. In this paper, we compare the performance of the OpenMP implementations across our test platforms. This SAR code reads in a 326MB file of raw radar echoes and an associated parameter file and generates a raw SAR image file. No additional input decks were available. Conversion of the raw image to a standard Windows BMP image is performed with a separate application that is not part of the benchmark and not included in the timing results.

An HTI application that uses modern state of the art algorithms for the extraction of temporal signals from synthetic high frame rate (>100 Hz) data was developed within our laboratory using MATLAB. The code was

Figure 1. Overview of the SAR Data Distribution and Parallel Processing Algorithm. The SAR data is broken in to panels and assigned to a processor by the master, p0. The master processor, p0, handles all input and output. Processing times are not shown to scale. The cycle above repeats until no more panels need to be processed. Panel processing includes range compression, azimuth transform, range migration, and azimuth compression.

subsequently converted to FORTRAN 90 and C. In this conversion process, heavy use was made of industry standard LAPACK¹ and BLAS linear algebra libraries as heavily optimized versions of these libraries are usually available in either vendor-supplied or open source implementations that are tuned to take advantage of architectural features such as cache size and structure.

These libraries are typically available in both serial and thread parallel implementations, so some level of parallelization is obtainable simply by swapping a serial library for a parallel one. For the platforms studied in this paper, the libraries used consisted of versions 3.4.2 (Xeon) and 3.5.0 (Odroid, Jetson) of the LAPACK library and version 3.10.1 of the Automatically Tuned Linear Algebra Subroutine (ATLAS) BLAS library. The parallel implementation of the ATLAS library makes use of pthreads. By default, the parallel ATLAS library makes use of all of the processors on the platform. For our Xeon reference platform this is 8 cores, for the ODROID XU3 this is a total of 8 cores (4 A15 cores and 4 A7 cores), for the NVidia Jetson TK1 this is 4 A15 cores. The configuration options of the ATLAS library also allowed us to build an ATLAS implementation that only uses the A15 cores on the ODROID board. This capability was unique to ATLAS.

The HTI processing chain can be broken into several standard image processing and data manipulation steps that can be expressed in terms of elementary operations such as convolution, matrix multiplication, eigenvector

decomposition, conversion of data to double precision floating point, local averaging, and covariance matrix calculations. Many of these operations can be expressed as BLAS Level 1 (vector), 2 (matrix-vector) and 3 (matrix-matrix) linear algebra operations. Other operations within the code were threaded using OpenMP. The application reads and processes 500 256x256 16-bit pixel images from a synthetic 420 MB data file.

The HTI code makes use of explicit thread parallelization via OpenMP directives and indirect parallelization through the use of parallel linear algebra libraries. This is possible because no linear algebra library call appears in any parallel region used by OpenMP. In this application we make use of a Basic Linear Algebra Subroutine (BLAS) library from the Automatically Tuned Linear Algebra Software (ATLAS) project [9] and LAPACK 3.5.0 [10]. When the ATLAS library is compiled both serial and thread parallel versions of the library are built. The serial version is optimized for a single processor core including the use of SIMD instructions such as the SSE2 instruction set in the Xeon or NEON instructions in an ARM processor. The default thread support used in the ATLAS library is pthreads². Also, the number of threads used by the ATLAS library is fixed at compile time and defaults to the number of available cores on the target machine. The LAPACK library is a serial library, but benefits from any parallelization inherited from the BLAS library calls it makes. Since the OpenMP directives can be enabled or disabled via a compiler flag, four different versions of the benchmark can be compiled: a purely serial implementation, an implementation in which

¹ LAPACK is a reimplementation of the LINPACK and EISPACK numerical libraries. Its routines are written using high level block operations that can be optimized for each architecture.

² POSIX threads

Figure 2. Speedup of parallel CHREC SAR application as a function of number of patches (patch size) and number of OpenMP threads used on our Xeon reference platform.

the serial HTI code is linked with the thread parallel ATLAS library, an implementation in which an OpenMP compiled HTI code is linked to a serial ATLAS library, and a version in which an OpenMP compiled HTI code is linked with the thread parallel ATLAS library.

OpenMP's parallel constructs, typically **omp parallel**, **omp for** and **omp parallel for**, were used extensively to speed up loops and iterative operations within the convolution, local averaging and data conversion blocks of code. Blocks of code that could be implemented using standardized linear algebra library calls, including Basic Linear Algebra Subroutines (BLAS) and LAPACK, are made parallel using parallel implementations of numerical libraries such as ATLAS. BLAS and LAPACK calls include: DGEMM³, DGER⁴, DSCAL⁵, DSYEV⁶.

4. RESULTS AND ANALYSIS

SAR Benchmark

In this implementation of the SAR benchmark, there are two parameters under user control: (1) the size of a patch (or number of patches) and (2) the number of OpenMP threads employed. As the data set is of fixed size and the patch dimension is a power of 2, varying the size of a patch changes the size of the image produced. The amount of data processed increases as the number of patches is increased, so comparing run times for different numbers of patches is

³ DGEMM is the double precision BLAS LEVEL 3 Matrix-Matrix Multiply.

⁴ DGER is the double precision BLAS LEVEL 2 Rank 1 Matrix Update operation.

⁵ DSCAL is the double precision LAPACK library routine for scaling a vector (or matrix) by a constant.

⁶ DSYEV is the double precision LAPACK library routine for computing eigenvalues and eigenvectors of a matrix.

Figure 3. Number of iterations of the parallel loop in the parallel SAR implementation.

not valid. However, varying the patch size allows us to see the impact of swapping to disk when the memory required exceeds the amount of RAM available. This can happen depending on the combination of patch size and number of processors chosen. Figure 2 illustrates the parallel speedup of the SAR application as a function of the number of OpenMP threads for different numbers of patches on our Xeon reference platform. Each speedup curve is computed using the serial run time appropriate to that number of patches. Available memory on this platform greatly exceeds that needed by the code, so the observed performance is not influenced by memory constraints. Instead, the observed performance is easily understood in terms of the number of iterations of the parallel loop body; that is, one iteration of the block of operations shown in Figure 1. The number of iterations that are performed for a given combination of number of patches and number of OpenMP threads are shown in Figure 3. Note that when the number of patches is

Figure 4. Parallel speedup of the CHREC SAR application on the NVidia Jetson TK1.

much greater than the number of available threads (processors), the number of parallel iterations becomes smoothly varying. When the number of patches becomes comparable to the number of available threads, there are ranges where adding a thread/processor does not change the number of parallel iterations. In these situations the speedup is found to worsen slightly as there is little benefit to adding a processor. Clear speedup improvements are seen when the number of parallel iterations decrease. As expected, utilization of processors is optimal when the number of patches is evenly divisible by the number of threads.

Figure 4 shows the parallel speedup of the SAR application on the NVidia Jetson TK1 for 9 and 35 patches. For 35 patches, reasonably good scaling is observed. For 9 patches, good scaling is observed up to 3 threads/processors. Addition of a 4th thread/processor does not reduce the number of iterations, so the 9 patch speedup decreases with the addition of a 4th thread.

The situation becomes more complex for the ODROID XU3 platform. Figure 5 shows the parallel speedup of the SAR application on the ODROID XU3. Recall that the ODROID has a cluster of 4 fast ARM A15 cores and a cluster of 4 slower ARM A7 cores. Initially, the process scheduler assigns tasks to the fast cores, so the speedup curves for less than or equal to 4 threads look very similar to that of the NVidia Jetson TK1. For more than 4 threads, speedup suffers dramatically as slower A7 cores are incorporated into the processing. For the case of 35 patches, there remains a performance benefit to using more threads because the gain from reducing the number of parallel

Figure 6. Parallel speedup of the CHREC SAR application on the ODROID XU3.

iterations outweighs the cost of adding slower processor cores to the calculation. Peak performance in this case occurs at about 6 cores. In a power constrained environment, such as aboard a spacecraft the minimal performance gains (20% higher speedup at 6 processors than at 3 processors) may not justify the use of additional processors.

Figure 5. Parallel speedup of the HTI application on our reference Xeon platform. The dashed lines are a single fit to both curves using a generalization of Amdahl's law.

HTI Benchmark

The analysis of the performance of the HTI benchmark is challenging because the number of pthreads used by the parallel ATLAS library is fixed when the library is compiled whereas the number of OpenMP threads can be varied at run time. For our Xeon reference platform, the number of pthreads used by the parallel ATLAS library is fixed at 8. For the NVidia Jetson TK1, the number of pthreads used by the parallel ATLAS library is fixed at 4. Since the ODROID XU3 is a heterogeneous architecture in the sense that its cores may be divided into two classes of cores that are binary compatible but run at different speeds, we studied the performance of the ODROID using two builds of the ATLAS library. In the first build, the number of pthreads is the default (8), i.e., the four fast ARM A15 cores are pooled with the four slower ARM A7 cores. In this case, scheduling of tasks on cores is left to the operating system. In the second case, we take advantage of build options in the ATLAS library to restrict the pthreads to run exclusively on the faster A15 cores.

Figure 6 shows the performance of our HTI implementation on our reference Xeon platform. Two curves are shown. One curve shows the impact of OpenMP thread parallelization when the code is linked with serial linear algebra libraries (ATLAS BLAS + LAPACK), the other includes the effect of using parallel linear algebra (parallel ATLAS) libraries. The difference in the two curves reflects the amount of parallelizable code distributed between the main body of the code and the numerical libraries. Since the run time associated with the parallel numerical library, which is using all 8 cores in this case, scales as well as or better than the OpenMP code which is using 1 to 8 cores, the code scales reasonably well with the number of OpenMP threads on this platform. The parallel efficiency of the code

when using all 8 cores for both OpenMP (main code) and pthreads (ATLAS) is approximately 70%.

Figure 7. Comparison of the performance of OpenMP schedulers on the OpenMP + pthreads implementation of the HTI application on our reference Xeon platform.

We also studied the performance impact on the code of the various loop schedulers available within OpenMP by adding the `schedule(runtime)` clause to all of the OpenMP parallelizable loops in the code. This allowed the loop scheduler to be varied at run time by assigning a value to the `OMP_SCHEDULE` environment variable. The environment variable allows both a schedule type and an optional “chunk size” value, which controls the size of blocks of loop iterations distributed to threads, to be set.

The available schedule types include **auto**, **dynamic**, **guided** and **static**. [11] The meaning of the optional “chunk size” value varies with the schedule type selected. When the schedule type is **auto**, the decision regarding scheduling is delegated to the compiler and/or runtime system. When the schedule type is **dynamic**, the loop iterations are distributed to threads in blocks of size specified by the

Figure 8. Parallel speedup of the HTI application on the NVidia Jetson TK1. The dashed lines are a single fit to all the data using Amdahl’s law.

Figure 9. Parallel speedup of HTI application on the ODROID XU3. The ATLAS library was compiled to use all 8 cores (4 ARM A15+4 ARM A7). The dashed lines are a single fit to all the data using a generalization of Amdahl’s law.

“chunk size” parameter as the threads request them. The default value of “chunk size” for the dynamic scheduling option is 1. When the schedule type is **static**, iterations are divided into blocks of size “chunk size” and the blocks are assigned to the threads in round-robin fashion in order of thread number. If “chunk size” is not specified, its default value for **static** is 1. When the schedule type is **guided**, iterations are again distributed to the threads as threads request them. The “chunk size” value, with the possible exception of the final chunk, is a lower bound on the size of a chunk. For the **guided** schedule type, the default value of the chunk size is 1.

Figure 7 shows the parallel speedup of the HTI application on the Xeon platform for various combinations of schedule type and chunk size. Also included is the speedup when the `schedule(runtime)` clause is not included in the loop parallelization specification, i.e. the default OpenMP scheduling. This is labeled as “default” in the figure. Chunk sizes were varied from 1 to 32 in powers of 2, but for clarity, curves for only a few representative values are shown. We observed that **dynamic** scheduling yielded very poor performance for small chunk sizes, but became comparable to other methods when the chunk size reached 32. The “default” scheduling performed as well as the best run time scheduling options.

Figure 8 shows the performance of the application on the NVidia Jetson TK1. Qualitatively, the behavior is similar to that of the Xeon. The parallel efficiency achieved when using parallel libraries with all 4 cores was about 78%.

Figure 9 shows the performance of the code on the ODROID XU3 using a version of the ATLAS BLAS library compiled to use all 8 cores (4 A15 cores and 4 A7 cores). The parallel performance is quite poor. We suspect that because of the mix of slow and fast cores in the pthreads

portion of the code, the pthreads execution is a larger portion of the run time than on the other boards tested. This acts to limit the scalability of the code with respect to OpenMP threads. We conducted experiments varying the OpenMP scheduler type and chunk size as well as the number and types of cores used by the ATLAS library to determine the likely culprit.

Figure 10 shows the performance of the code on the ODOROID XU3 when the parallel linear algebra library is restricted to run only on the four faster ARM A15 cores. The parallel performance is improved by about 45% by restricting the BLAS operations to the ARM A15 cores.

The effect of the OpenMP scheduler options selected on the

Figure 10. Parallel speedup of HTI application on the ODOROID XU3. The ATLAS library was compiled to use the 4 ARM A15 cores only. The dashed lines are a fit to all the data using a generalization of Amdahl's law.

Figure 12. Comparison of the performance of OpenMP schedulers on the HTI application on the ODOROID XU3. The ATLAS library was compiled to use all 8 cores (4 ARM A15+4 ARM A7).

performance of the code is shown the next few figures.

Figure 11 shows the effect of OpenMP scheduler options on the HTI application when the parallel ATLAS library is compiled to use both ARM A15 and A7 cores. Figure 12 shows the effect of scheduler options on the HTI application when the ATLAS library is compiled to only use the A15 cores. In both cases, the best performing run time scheduling option offers the same performance as the default OpenMP scheduler.

We have found that the performance of the HTI application can be well characterized by a slight generalization of Amdahl's law.[12] We have found that the parallel speed up can be written as

$$S = \frac{t_{\text{serial}} + t_{\text{OpenMP}} + t_{\text{pthreads}}}{t_{\text{serial}} + t_{\text{OpenMP}}/n_{\text{OpenMP}} + t_{\text{pthreads}}/n_{\text{pthreads}}}$$

where t_{serial} is the execution time for the serial portion of the code, t_{OpenMP} is the serial execution time for the portion of the code using OpenMP threads, and t_{pthreads} is the serial execution time for the portion of the code parallelized using pthreads. The dashed lines in figures 6, 8, 9 and 10 are least squares fits of the data to the expression above using the ratios $t_{\text{OpenMP}}/t_{\text{serial}}$ and $t_{\text{pthreads}}/t_{\text{serial}}$ as the fitting parameters. For homogeneous architectures, the fit is excellent while for the heterogeneous architecture the fit is still reasonable. For this problem size, extrapolation of the curves suggests that at most 16 processors can be used for this application. The fitted ratios are platform dependent reflecting compiler differences and differences in SIMD units.

Figure 11. Comparison of the performance of OpenMP schedulers on the HTI application on the ODOROID XU3. The ATLAS library was compiled to use only the ARM A15 cores.

Finally, in Figure 13 we compare the relative performance of the codes on the various platforms. Here we present speeds relative to the Xeon platform. The relative speed shown here is the ratio of the Xeon run time for k OpenMP threads to the run time for the various platforms for k OpenMP threads; thus the figure shows to what extent the other platforms underperform the Xeon platform as a

function of the number of OpenMP threads used. The curve for the NVidia Jetson is flatter, reflecting differences that are solely due to clock speed and differences in performance of SIMD units on the Xeon and Jetson. The slight downward trend in this curve reflects the fact that the ATLAS library used with the Jetson was built to use 4 cores whereas the ATLAS library used with the Xeon platform uses 8 cores. The curves for the ODROID reflect differences in clock speed and the impact of using slower ARM A7 cores in the computation.

5. SUMMARY

In this paper, we have examined the performance to two space applications on low power COTS processors based on the ARM architecture. The processors included examples of both homogeneous and heterogeneous (big.LITTLE) architectures. The applications examples included two image processing applications. The heterogeneous architecture fared poorly as the slower LITTLE cores were a performance bottleneck.

The new laboratory provides us the capability to study how next generation mission applications map to current and future space processing platforms. We hope to work with other groups to develop other benchmarks in the future.

Figure 13. Parallel performance of the 32-bit ARM implementations relative to the 64-bit Xeon using the same number of OpenMP threads. The serial code is shown as having zero OpenMP threads.

ACKNOWLEDGEMENTS

The authors thank the Air Force Office of Scientific Research for funding this project through its IRP program. We also thank Dr. Alan D. George and Chris Conger of the University of Florida for supplying the SAR code.

REFERENCES

- [1] T. M. Lovelly, D. Bryan, K. Cheng, R. Kreymin, A. D. George, A. Gordon-Ross and G. Mounce, "A Framework to Analyze Processor Architectures for Next-Generation On-Board Space Computing," *IEEE Aerospace*, pp. 1-10, 2014.
- [2] D. Rudolph, C. Wilson, J. Stewart, P. Gauvin, A. George, H. Lam, G. Crum, M. Wirthlin, A. Wilson and A. Stoddard, "CSP: A Multifaceted Hybrid Architecture for Space Computing," *28th Annual AIAA/USU Conference on Small Satellites*, paper SSC14-III-3.
- [3] Web site: www.chrec.org.
- [4] D. Bueno, C. Conger, A. Leko, I. Troxel and A. George, "RapidIO-based Space System Architectures for Synthetic Aperture Radar and Ground Moving Target Indicator," *Proc. of High-Performance Embedded Computing (HPEC) Workshop*, MIT Lincoln Lab, Lexington, MA, Sep. 20-22, 2005.
- [5] C. Conger, A. Jacobs, and A. George, "Application-level Benchmarking with Synthetic Aperture Radar," *Proc. of High-Performance Embedded Computing (HPEC) Workshop*, MIT Lincoln Lab, Lexington, MA, Sep. 18-20, 2007.
- [6] A. Jacobs, G. Cieslewski, and A. George, "Multiparadigm Computing for Space-Based Synthetic Aperture Radar," *Proc. of International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, Las Vegas, NV, July 14-17, 2008.
- [7] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, P. Alken, M. Booth, F. Rossi, and R. Ulerich, *GNU Scientific Library Reference Manual*, Edition 1.16 for GSL Version 1.16 (17 July 2013).
- [8] D. T. Sandwell and E. J. Price, "Phase gradient approach to stacking interferograms," *J. Geophysical Res.*, Vol. 103, No. B12, pp. 30,183-30,204, December 10, 1998. Paper number 1998JB900008.
- [9] Clint R. Whaley and Jack J. Dongarra, "Automatically Tuned Linear Algebra Software", *Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*, SC '98, pp. 1-27 (1998).
- [10] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenny and D. Sorensen, *LAPACK User's Guide*, 3rd Ed., Society for Industrial and Applied Mathematics, Philadelphia, PA (1999).
- [11] The OpenMP API specification, including version 3.1, may be found on the OpenMP website, www.openmp.org.
- [12] G. M. Amdahl, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities", *AFIPS Conference Proceedings*, vol. 30, p. 483-485 (1967).

BIOGRAPHY

Andrew C. Pineda (Member, IEEE) received a B.S. in Chemistry from Harvey Mudd College, Claremont, CA in 1982, and A.M. degree and Ph.D. degree in Chemistry and Physical Chemistry from Harvard University, Cambridge, MA in 1984 and 1994, respectively. From 1997 to 2012, he was a Sr. Research Scientist and an adjunct/research faculty member at the University of New Mexico affiliated with the UNM Center for High Performance Computing, the Department of Chemistry and the Department of Electrical and Computer Engineering. After working with AFRL as a contractor from 2012 to 2015, Dr. Pineda recently joined the AFRL Space Vehicles Directorate. Dr. Pineda has co-authored over 20 papers on electronics materials-related topics ranging from defects in SiO₂, optical properties of III-V materials, molecular electronics, nanotubes, chalcogenides and dielectric phenomena. His research interests include electronic structure theory of electronics materials and high performance computing.

Jesse K. Mee received the M.S. degree (distinction) in Electrical Engineering (EE) and the Ph.D. degree (distinction) in EE with a concentration in Optoelectronics from the University of New Mexico in 2010 and 2013, respectively. His M.S. thesis examined reliability physics of microelectronics for

space application, concentrating on the reliability degradation phenomenon known as Negative Bias Temperature Instability. In his doctoral research, he undertook an extensive investigation of time-domain characteristics of monolithic quantum dot passively mode-locked lasers for high data-rate transmission architectures. Dr. Mee has been working for AFRL Space Vehicles Directorate since 2008 and is the Technical Area Coordinator for the Space Electronics Technology Program's Systems Studies Group where he leads a collaboration of military, civilian, contractors & academic S&Es under an ambitious spacecraft architecture modeling and simulation project. He has led approximately 20 in-house and contractor research efforts on satellite optical backplane, flexible power electronics for space, data compression and advanced microelectronics. Dr. Mee is on the UNM Electrical and Computer Engineering Department's Advisory Council, and is the president of the Phillips Research Site Junior Force Council.

Reed A. Weber received a Ph.D. in Optical Science and Engineering from the University of New Mexico, following an M.S. from UNM and a B.S. in Physics from Bethel University. He is currently the lead data analyst for the Air Force Research Laboratory's hyper-

temporal imaging space experiment where he has focused on developing advanced algorithms for signal detection specific to high frame rate data. He is a former National Science Foundation GK-12 Fellow and DOD SMART Scholar. He is the author or co-author of numerous papers and conference presentations related to imaging diagnostics of both fluid dynamic phenomena and advanced materials.

Phillip M. Cunio graduated from the Massachusetts Institute of Technology in 2012 with a Ph.D. in Space Systems, after completing his undergraduate studies in aerospace and mechanical engineering at the University of Florida. He works as an acquisition engineer for spaceborne data processing systems in the Air Force Research Laboratory's Space Vehicles

directorate. His past work includes testing for a prototype lunar hopping vehicle, system-level commonality analysis for life support systems, and Mars exploration via human-robot teams.