Using a Multicore Processor for Rover Autonomous Science

Benjamin Bornstein, Tara Estlin, Bradley Clement, Paul Springer
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109
{firstname.lastname}@jpl.nasa.gov

Abstract—Multicore processing promises to be a critical component of future spacecraft. It provides immense increases in onboard processing power and provides an environment for directly supporting fault-tolerant computing. This paper discusses using a state-of-the-art multicore processor to efficiently perform image analysis onboard a Mars rover in support of autonomous science activities.

Techniques for onboard rover-data processing provide significant mission benefits for both data prioritization and opportunistic science. These capabilities automatically analyze collected data (e.g., visual images) onboard a spacecraft and then use that analysis to either prioritize collected data for downlink or identify new science opportunities for collection of valuable science data. Onboard data analysis is already in use on the Mars Exploration Rover (MER) mission rovers to autonomously select and gather new data on interesting rock targets.

One of the key elements of this MER capability is identifying rocks in MER navigation camera images. Surface rocks are one of the primary targets for science investigation on the surface of Mars. Automated identification of these rocks is a critical element of rover autonomous science algorithms but is also one of the most time intensive. To enable better performance on future missions, this capability has been adapted to the Tilera TILE64™ multicore processor. This paper discusses how this adaption was performed as well as presenting results on performance improvements provided through the use of multiple cores.[1] [2]

TABLE OF CONTENTS

1. INTRODUCTION.....1
2. AEGIS SYSTEM.....2
3. ROCKSTER.....3
4. TILERA MULTICORE PLATFORM.....4
5. ROCKSTER PARALLELIZATION.....5
6. RESULTS.....6
8. SUMMARY AND CONCLUSIONS.....8
REFERENCES.....8
BIOGRAPHY.....9

1. INTRODUCTION

Multicore processors could provide future spacecraft immense increases in onboard processing performance and an environment for directly supporting fault-tolerant computing. As part of a three-year research and technology development initiative we are investigating the detailed benefits of onboard autonomy and multicore computing for future NASA missions.

Long-range driving, increased autonomous operations, and onboard science have been repeatedly identified as needed capabilities for many future rover missions, including the planned 2018 Mars Astrobiology Explorer-Cacher (MAX-C) Rover Mission and the planned Mars Sample Return (MSR) Rover Mission. These future missions have significant distance driving requirements as well as goals for increased science. Multicore computing will enable the efficient execution and coordination of rover activities. Further, the ability to rapidly perform onboard science will be beneficial to a large class of missions. Other relevant mission concepts include future missions to Titan, Europa, Venus and other Mars missions. These missions may have in-situ and/or orbital spacecraft..

We are adapting three high-level autonomy capabilities for current and future rover surface missions to a multicore processor and performing detailed performance evaluations. We chose each capability for its unique, and complementary, parallelization challenges. The first capability is the Rock Segmentation Through Edge Regrouping (ROCKSTER) rock and target detection algorithm, which is currently in use onboard the MER Opportunity rover as a critical part of the larger onboard science autonomy system, AEGIS (Autonomous Exploration for Gathering Increased Science) [1]. ROCKSTER has many of the hallmarks of a classic machine vision algorithm and, as such, is well suited to data parallelization. We will focus on our experiences adapting ROCKSTER to the 64-core Tilera TILE64™ multicore platform [6] throughout most of this paper. The National Reconnaissance Office's OPERA program has contracted with Boeing to develop MAESTRO, a 49-core, radiation-hardened Tilera-class processor. As such, MAESTRO will provide a reasonably direct space flight qualification path for algorithms ported to the Tilera multicore platform.

[1] 978-1-4244-7351-9/11/$26.00 ©2011 IEEE.
[2] IEEEAC paper #1104, Version 1, Updated January 10, 2011

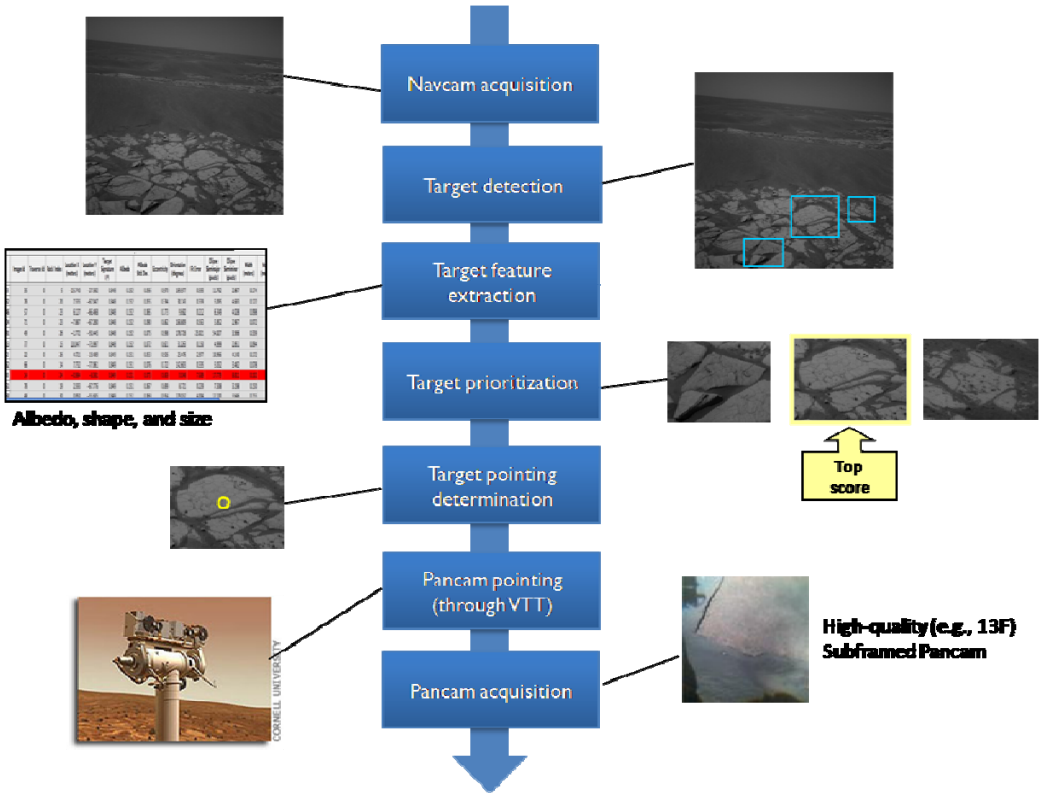

Figure 1 – AEGIS Process Pipeline

When AEGIS is sequenced, the above series of steps is executed onboard the MER Opportunity rover. Parameters can be set during ground command sequencing to specify navigation camera pointing, the "target rock signature" (e.g., rocks of large size and low albedo), and settings for the panoramic camera (e.g., what filter set to use). ROCKSTER is the first AEGIS data processing step (target dection, second block) and is thus critical for all subsequent AEGIS operations. As the most computationaly intensive aspect of AEGIS, ROCKSTER is a prime candidate for multicore parallelization.

Over the next two years, we will also adapt a visual texture and onboard planning and scheduling algorithms to the Tilera multicore platform. Visual texture is another machine vision algorithm but one that lends itself even more easily to task-based parallelism (e.g. one scale-orientation filter per core). Onboard planning and scheduling is another enabling mission technology that provides capabilities for onboard sequence modification and resource management. However its adaptation to multicore is challenging since there are multiple ways the underlying algorithm and computation could be migrated to a multicore processor.

Section 2 gives an overview AEGIS to provide context for the ROCKSTER algorithm and also to give a flavor of current state-of-the-art spacecraft autonomy. Section 3 introduces ROCKSTER in its original serial form. Section 4 describes the TILE64™ processor and platform and also its extension to the space-qualified MAESTRO processor. Section 5 details ROCKSTER parallelization. Section 6 presents multicore performance results. Finally, Section 7 closes with a summary, conclusions, and areas of future work.

2. AEGIS SYSTEM

The Autonomous Exploration for Gathering Increased Science (AEGIS) system provides autonomous deployment of science instruments that target specific terrain features [1]. A number of rover remote sensing instruments, such as the MER Mini-TES spectrometer, have a very narrow field-of-view and thus require selection of specific focused targets for sampling. Selecting targets for these instruments by mission personnel on Earth is currently a lengthy process. Typically operators will manually identify the targets in images that have already downloaded on a previous sol (Martian day). These context images are collected with wide field of view (FOV) cameras such as the MER navigation cameras, which have a 45 degree FOV, or the MER panoramic cameras in a full-frame low-resolution (single filter) mode, using a 16 degree FOV. After reaching an end-of-day location, the rover performs only untargeted data collection until the context images can be analyzed and new measurement commands uplinked. At

best this will happen on the next sol, but it may never happen if it is decided the rover should immediately proceed to a new location due to other goals or engineering constraints.

By analyzing image data onboard, AEGIS can autonomously select targets for these instruments and execute a set of measurement activities. These techniques could be used, for example, on the Mars Science Laboratory (MSL) mission to select targets for the ChemCam spectrometer instrument to sample at the end of a long rover drive. For MER AEGIS has demonstrated taking additional measurements with the panoramic camera in a quarter-frame high-quality (multiple filter) mode, which uses a 4 degree FOV.

AEGIS is run as part of the MER onboard flight software. All AEGIS components run onboard the MER 20 MHz RAD6000 flight processor, which has an early PowerPC instruction set, with 128 MB of RAM and 256 MB flash memory. AEGIS was required to run using less than 4 MB of RAM to ensure other onboard processes were not impacted.

AEGIS was originally developed as part of a large autonomous science framework called OASIS (Onboard Autonomous Science Investigation System) [2]. OASIS is designed to operate onboard a rover identifying and reacting to serendipitous science opportunities. OASIS analyzes data the rover gathers, and then, using machine learning techniques, prioritizes the data based on criteria set by the science team. This prioritization can organize data for transmission back to Earth or search for specific targets specified by the science team. If one of these targets appears, the system attempts to act on the new science opportunity by taking new instrument measurements. The AEGIS technology focuses on this second task of using onboard data analysis to acquire new instrument data on science targets, typically rocks, which have been identified in an opportunistic fashion after a drive.

AEGIS performs seven major steps to autonomously acquire new data on an interesting science target. These steps are shown in Figure 1 and described below:

1. Acquire an image with the MER navigation camera: Scientists and other sequence team members select image parameters, such as the pointing direction and resolution, during the AEGIS sequencing process. The navigation camera is typically pointed at a terrain area where potential science targets may be in view.

2. Analyze the navigation camera image for potential terrain targets: Targets for AEGIS typically correspond to rocks. AEGIS uses the ROCKSTER algorithm to look for intensity edges in grayscale imagery. This algorithm is further detailed in Section 3.

3. Extract relevant target features: AEGIS calculates a set of target features (or properties) for each candidate rock. These properties include measures of size, albedo (reflectance), and shape.

4. Prioritize targets and select top target: This component uses a prioritization algorithm to analyze rock property data and determine a top candidate. Scientists provide a "target rock signature" in the command sequence. This signature specifies what property values are of interest in the local terrain. Example signatures are "high albedo," "round shape," "large rocks with low albedo," etc.

5. Determine 3D target pointing requirements: After identifying the best scoring candidate rock, AEGIS selects a center point on the target using an inscribed circle method.

6. Point remote sensing instrument: AEGIS points the panoramic cameras at the new target using the resulting center point.

7. Acquire new data: AEGIS then acquires additional data with the panoramic cameras. The ground sequencing team can select the exact filters and other imaging parameters to use for each individual run. Typical command sequences take a quarter-framed, multiple filter image with both left and right cameras. The rover downlinks these opportunistic images with other standard data products.

3. ROCKSTER

To identify potential targets, primarily rocks, of interest, the AEGIS system employs the Rock Segmentation Through Edge Regrouping (ROCKSTER) algorithm [3]. This section provides an overview of the target detection algorithm in its original serial form. Section 5 details the parallelization of the algorithm on the Tilera TILE64™.

ROCKSTER focuses on the intensity of edges in grayscale imagery and connecting them to form closed contours. Existing MER rover engineering instrument packages and compute resources drove the choice of input data format and the level of processing applied to images. The MER navigation cameras produce 1024x1024 grayscale images at 12 bits per pixel [4]. Navigation camera images are quick to collect (30 seconds) and are acquired frequently during rover drives. As previously described, the MER compute element is a RAD6000 flight processor capable of executing 20 millions of instructions per second (MIPS). The entire AEGIS system, and in particular the ROCKSTER subsystem, is designed to operate efficiently on this processor. Efficient operation includes sharing compute resources with the always-running rover flight software, which is responsible for vehicle health and safety, telemetry, and communication.

ROCKSTER locates edges in an image of the terrain taken by one of the navigation cameras using a process similar to the well-known Canny edge detection algorithm [5]. In particular, the input image is first smoothed, using a 5x5 Gaussian kernel, to reduce the detection of small, spurious edges. Then central difference kernels are used to estimate the intensity gradient in the horizontal and vertical directions. Ridges in the intensity gradient are linked together using non-maximum suppression, hysteresis thresholding, and edge-following to produce a set of contours.

Unfortunately, this initial set of contours does not directly provide a usable segmentation of the rocks from the background due to various problems, including: (1) spurious contours from the sky-ground boundary (horizon line), texture within individual rocks, and texture present in the background, (2) incorrect linking choices at the junctions between contours, (3) and unclosed contours around an object due to gaps in the gradient information (for example, areas along the rock boundary where the rock intensity and background intensity are too close to reliably separate). ROCKSTER attempts to resolve these problems by splitting the initial contours into low-curvature fragments. A gap-filling mechanism is then applied to add new contour fragments between existing fragment endpoints. The final step is to regroup the edge fragments into coherent contours, which is accomplished through background flooding. Conceptually, water is poured into the image from the sides but the water is not allowed to cross over any edge fragments; thus, regions that are totally enclosed by edge fragments remain "dry" while other areas become "wet." Extracting contours around the dry areas yields the final rock segmentation (Figure 2).

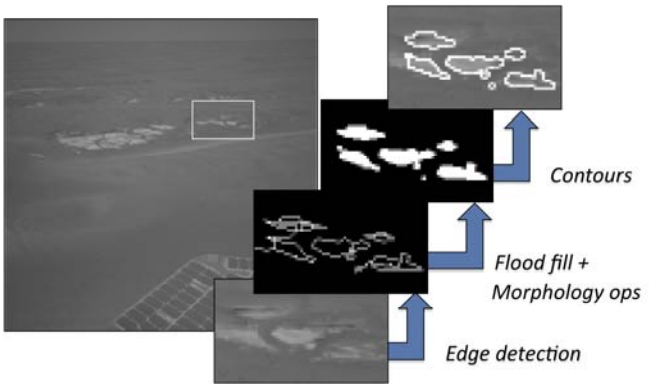

Figure 2 – ROCKSTER (Serial)

ROCKSTER operates on grayscale intensity images by performing Canny-like edge detection and linking, followed by salient point determination, splitting, and gap-filling. To remove salt-and-pepper noise, morphology operations are performed on flood-filled binary mask image. Final rock and other target contours are extracted from this binary image.

4. TILERA MULTICORE PLATFORM

Hardware

The Tilera TILE64™ runs at 750 MHz and has a parallel architecture with 64 processing elements (PEs) arranged in an 8x8 grid. Each PE is a full-featured processor, with a 32-bit arithmetic logic unit and a three-way Very Long Instruction Word (VLIW) architecture allowing up to three instructions per cycle. A single PE contains eight kilobytes of instruction L1 cache, eight kilobytes of data L1 cache, and 64 kilobytes of combined L2 cache. The cache coherency mechanisms on the TILE64™ also allow for a virtual L3 cache, where each PE can access the contents of any other PE's L2 cache. Each PE also contains its own Direct Memory Access (DMA) engine and Translation Lookaside Buffer (TLB) allowing memory virtualization and the ability to run a modern operating system on each core. Processing elements are connected to their four nearest neighbors by five data channels. There are four external double data rate (DDR2) memory channels, and each processor can access external memory directly without explicit user code to transfer messages. The external memory architecture is monolithic and flat, as a specific memory location has the same physical address on every PE. The processor also includes multiple input/output (I/O) ports, including two Peripheral Component Interconnect Express (PCIe) interfaces, multiple Serial/Deserializer (SERDES) interfaces, General Purpose I/O, and Xilinx 10 Gigabit Attachment Unit Interface (XAUI). [6]

For this study, an off-the-shelf evaluation board from Tilera was used. The evaluation board was a full-length PCIe form factor board containing a TILE64™ processor, two gigabytes of DDR2 random access memory (RAM), a PCIe interface, and all the high speed interfaces brought out to connectors. The gigabit Ethernet interface is brought out through a 12 port gigabit Ethernet switch. There is also an on-board non-volatile RAM that allows the board to run as either a daughtercard in a desktop system, or as a standalone single board computer booting into Linux.

Development environment

Tilera provides a software architecture based on Linux and the GNU toolchain. No changes were required to compile the ROCKSTER algorithm for the Tilera as a non-parallel application. The toolchain includes the standard GNU C and C++ compilers as well as all debugging, profiling, and object manipulation tools. The development system also includes a full-featured Linux based operating system with all of the standard libraries. Any application written for a Linux system can be simply re-compiled and can run on the Tilera system with little or no change. A straightforward compile of the original code, however, produced a binary file capable of running on only a single Tilera processing element. The existing ROCKSTER code base had to be rewritten to make use of the Tilera architecture and meet the runtime requirements. [7]

Space Flight Qualification

The National Reconnaissance Office (NRO) has for several years funded the OPERA program to develop MAESTRO, a radiation hardened by design version of the TILE64™ with additional features such as a dual precision IEEE-754 floating-point unit and hardened I/O features. The project is mature, and is currently testing MAESTRO, with a schedule to complete a 49-core version by the end of 2010.

The MAESTRO processor is currently projected to run at 300 MHz as opposed to the COTS 750 MHz. Inclusion of the IEEE-754 floating-point hardware unit will compensate for part of the clock speed discrepancy between the TILE64™ and MAESTRO processors. The Tilera TILE64™ uses software-emulated floating-point. While the issues of floating-point and clock speeds could have an impact on the ROCKSTER performance numbers presented in Section 6, as we have explained in Section 5, we have made a significant effort to reduce ROCKSTER's use of floating-point operations, leaving only a difference in clock speed between the two processors.

5. ROCKSTER PARALLELIZATION

ROCKSTER presents many possible avenues for parallelization. In this section, we describe the parallelization approaches we considered, their motivation, and respective tradeoffs. After each approach is outlined, we provide implementation details. In the next Section we present our performance findings for our preferred approach.

One possible parallelization strategy is quite general and requires no algorithm modifications whatsoever. The MER (and future) rovers' camera hardware can acquire images much faster than onboard software can process them. With operating system level process (or thread) partitioning for multicore, it is relatively easy to run a new and separate instance of ROCKSTER on an idle core whenever a new image is acquired. In the case of a MER rover, a navigation camera image has a 45 degree FOV and requires about 30 seconds to acquire. Therefore, a full 360 panorama could be analyzed for rocks of interest using only eight of the TILE64™ 64 cores in four minutes plus the amount of time required for ROCKSTER to execute on a single processor core. The approach applies equally well to other onboard cameras. For instance, the high-resolution MER panoramic camera has a 16 degree FOV and a filter wheel to image at multiple wavelengths. While we implemented this type of parallel processing on the TILE64™, the scaling achieved exactly matches the theoretical optimum, and therefore is not particularly interesting. We were more interested in understanding what benefits could be gained from a deeper algorithmic parallelization.

Moving-up the parallelization sophistication scale, the next seemingly simple strategy leverages fine-grained data parallelism. The smoothing, edge detection, and morphology operations at the heart of ROCKSTER are agnostic to the content of the image and more importantly, the image size. Therefore, it is quite natural to divide an image into subimages and process each subimage through the major phases of ROCKSTER on a separate core (Figure 3). What distinguishes this approach from the one previously described is that the ROCKSTER is cognizant of the parallelization and is therefore responsible for dividing the image into subimages, assigning subimages to each core, and aggregating rock detections after each subimage has been processed. Such tight algorithmic parallelism offers opportunities for load-balancing (e.g. subimages containing mostly sky will require less processing overall) and is required to address more challenging implementation issues like connecting rock contours that span subimage boundaries. This latter issue requires communication among multiple cores.

The previous two strategies have both emphasized data parallelism. Indeed, our choice of ROCKSTER was motivated partly by its natural fit to a data-parallel algorithm decomposition and our desire to gain experience with data parallelism in a multicore environment. Still, we would be remiss if we did not at least mention the possibility of a task-parallel decomposition. The ROCKSTER edge detection algorithm is amenable to a systolic-array like computation where the Gaussian smoothing kernel and gradient edge kernels are decomposed into individual pixel-level filters run on each core. For instance, edge detection, although currently performed with a central difference operator, could be decomposed into separate Sobel horizontal and vertical kernels and run in parallel. While we have begun to investigate this parallelization strategy, our work in this area is still in-progress.

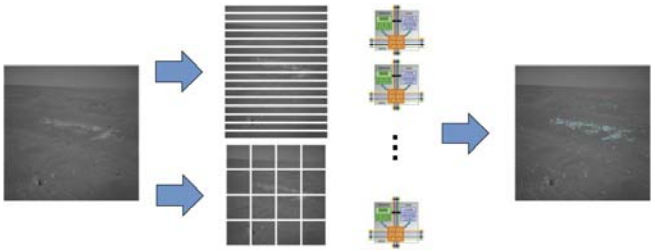

Figure 3 –ROCKSTER (Parallel)

Parallel ROCKSTER divides the input image into subimage strips or tiles and runs the major phases of ROCKSTER on each core. Challenges include choosing optimal subimage sizes, allocating to cores (load-balancing), and communication of contours across cores during edge linking.

Implementation

As Figure 3 depicts, images may be split into either subimage strips or tiles. Dividing the input images into

strips required very few changes to ROCKSTER image kernels as they are optimized to work across image columns and images are stored internally in row-major (row pixels contiguous) format. However, we quickly discovered that, in practice, the long horizontal subimages cut across far too many rock boundaries when more than eight cores were used.

Parallel ROCKSTER relies on the Message Passing Interface (MPI) [8] for communication across cores. The MPI package was created specifically for Tilera and MAESTRO by USC's ISI East as part of the OPERA program. At a low-level, core-to-core communication is accomplished by using Tilera's iLib (supported, but deprecated) or TMC C programming libraries. ISI's MPI is a particularly efficient implementation layered on top of the TMC library.

Parallel ROCKSTER can utilize either a shared memory model or a separate partition of memory per core. Shared memory serves only to reduce the initial overhead associated with MPI data passing. At the level of implementation, ROCKSTER image operations are restricted to the subimage assigned to that core. For shared memory, we use a TMC application programming interface (API) call to receive an initial pointer to shared memory. From then on, all application logic treats the pointer as a vanilla C pointer; no special access APIs are required. We have not studied the tradeoffs in shared versus local memory. Similarly, we have not focused on load balancing with respect to the four memory ports on the Tilera chip (each port is assigned to a specific subset of cores). As our initial results indicate, optimizing memory access patterns is worthy of further study.

Floating-Point Operations

Our initial port of serial ROCKSTER to single Tilera TILE64™ core was unexpectedly slow. That is, runtime was not commensurate with the 750 MHz per-core clock speed. As part of their standard development environment, Tilera has ported the open-source oProfile runtime profiling tool to their platform. We used oProfile to investigate the cause of the slow runtime. The culprit was ROCKSTER's heavy use of floating-point operations.

As mentioned previously Tilera processors do not possess floating-point hardware. Instead all floating-point data types and operations are emulated in software. Tilera's customized GCC-based development tool chain makes floating-point software emulation completely transparent to software developers. The drawback, of course, is the hidden complexity of software emulation can come at the cost of performance.

In the case of ROCKSTER, the software-emulated floating-point performance hit was extreme. ROCKSTER spent nearly 75% of its TILE64™ single-core runtime performing floating-point operations in software. Since the emulation is CPU-bound, we were concerned any performance and scaling numbers obtained after ROCKSTER parallelization would be skewed and not directly comparable to the radiation hardened 49 core MAESTRO processor, which will contain a floating-point coprocessor per core. As a result, we made a decision early on in our development to convert ROCKSTER to integer operations and data types where possible. This effort was particularly time consuming but paid dividends in the end. After conversion, ROCKSTER now spends less-than 10% of its total TILE64™ runtime performing floating-point operations in software.

6. RESULTS

In this Section we present overall performance and scaling numbers for both serial and parallel ROCKSTER across a large, representative set of MER navigation camera images.

As part of the AEGIS development effort, we curated a set of 116 MER navigation camera images from the mission's surface operation. The images form a representative sample of Martian terrain types encountered by the Spirit and Opportunity rovers and are classed according to the predominant geologic feature displayed in the image (e.g. cobbles, outcrop, and wind-swept dunes). The original purpose of the image set was to tune AEGIS parameters prior to surface operations. Thereafter the images and corresponding rock detections were the basis of a regression test set. This same set of images was used for both serial and parallel ROCKSTER to assess overall performance and scaling with increasing numbers of processor cores.

Floating-Point versus Integer Operations

The conversion of the majority of ROCKSTER's floating-point operations to primarily integer-only operations reduced the average TILE64™ single-core runtime by an average of five fold (Figure 4, images are ordered by complexity). The TILE64™ does not have hardware accelerated floating-point, whereas the space-flight qualified MAESTRO processor contains a floating-point coprocessor per core. In order to ascertain how parallel ROCKSTER performance on TILE64™ would map to MAESTRO hardware, factoring-out the contribution of CPU-bound, software-emulated floating-point operations was essential.

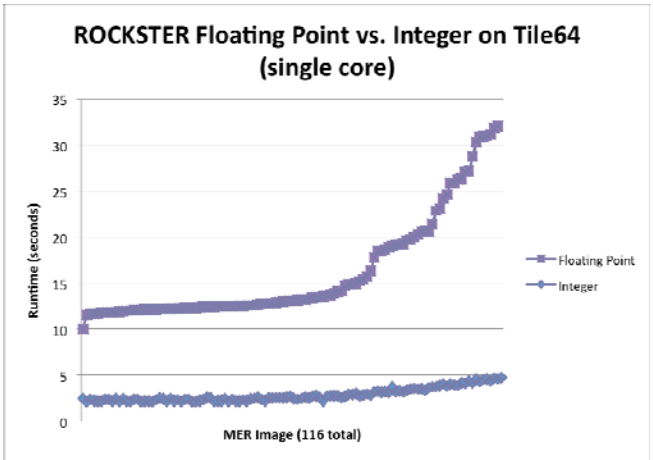

Figure 4 – ROCKSTER Floating-Point vs. Integer

The conversion of the majority of ROCKSTER's floating-point operations to primarily integer operations reduced single-core TILE64™ runtime by an average of fivefold. Each point along the x-axis represents a specific MER navigation camera image and is ordered by increasing runtime. The y-axis is runtime in seconds.

Parallel Performance and Scaling

We used the MER navigation camera regression test set described previously to assess both the overall performance of parallel ROCKSTER and the scaling achieved with increasing numbers of cores. Figure 5 shows runtimes for 1, 2, 4, 8, 16, and 32 cores, with images ordered by complexity. Table 1 shows both the average performance increase over a single core and the average runtime. For 32 cores, parallel ROCKSTER achieved nearly a 10-fold increase in runtime performance.

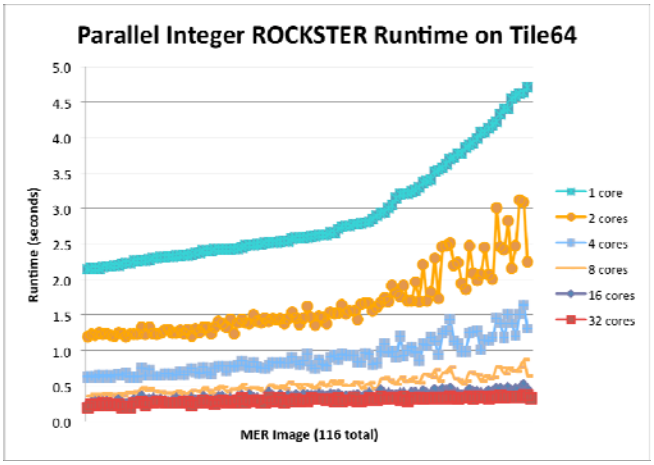

Figure 5 – Parallel Integer ROCKSTER Runtimes

Parallel Integer ROCKSTER runtimes for 1, 2, 4, 8, 16, and 32 cores operating on image strips are plotted above. The x-axis represents a specific MER navigation camera image and is ordered by increasing runtime. The y-axis is runtime in seconds.

Table 1 – ROCKSTER FtesRuntimes and Speedup Factors

	Number of Cores					
	1	**2**	**4**	**8**	**16**	**32**
Average speedup factor per image (versus single core)	1	1.8	3.3	5.5	8.0	9.7
Average runtime, all 116 MER Navcam images	2.9 s	1.6 s	0.9 s	0.5 s	0.4 s	0.3 s

Parallel Integer ROCKSTER average speedup factor compared to a single core (first row) and average runtime across all 116 MER navigation camera images in our regression test set (second row). For 32 cores, parallel ROCKSTER achieved nearly a 10-fold increase in runtime performance.

The runtimes for parallel ROCKSTER exhibit an interesting jagged structure from one run to the next after the 80th image. The x-axis is ordered by runtime, which is proportional to the overall complexity of the scene in terms of total number of edges. With increasing scene complexity, and smaller subimages, ROCKSTER is allowed to find a greater number of potential targets in some subimages. This is due to an intrinsic limit in the total number of targets ROCKSTER can track at any one time. This limit is a vestige of the memory constraints of the MER rovers. After a certain scene complexity threshold is passed, the number of targets found per subimage surpasses the intrinsic limit, which leads to greater variability in runtimes. For assessing only scalability, not overall performance, it is best to ignore the runtime variability past the 80th image. We mention this detail here only to illustrate one of the many subtle, algorithm-specific issues that are often encountered during parallelization and scalability studies.

An order of magnitude decrease in processing time certainly speaks to the benefits of multicore processing for this and similar onboard autonomy applications. The performance improvement is more dramatic when compared to current ROCKSTER runtimes on the surface of Mars. With a full, always-running flight software load, ROCKSTER often takes between 10–15 minutes to run on the MER RAD6000 20 MIPS processor. Utilizing half of the available TILE64™ cores (e.g. reserving the remaining cores for other flight software tasks), this runtime is reduced to 0.3 seconds. Since the MAESTRO processor will operate at 40% the clock speed of the TILE64™, it may be appropriate to scale reported runtimes by a factor of 2.5.

The performance gains we achieved through ROCKSTER parallelization are impressive, but still fall short of the theoretical optimum. Figure 6 shows ideal versus actual speedup as a function of cores. The achieved speedup curve suggests ROCKSTER performance asymptotes at approximately 10-fold. While we have not fully investigated the reason for actual versus theoretical performance discrepancy, our initial profiling suggests both memory and cache bottlenecks. As part of MAESTRO tool development program, several TILE64™/MAESTRO memory and cache profiling tools are available. Hopefully these tools will prove value in testing our performance bottleneck hypothesis.

Core-to-Core Communication

Image data decomposition is a natural parallelization strategy for many machine vision applications. While ROCKSTER is no exception, parallelization is not without its challenges. Maintaining contour coherency across subimage boundaries, and therefore cores, is required to maximize rock detections. This is especially true for large rocks, which tend to be scientifically interesting, yet are more likely to be divided among multiple subimages. We have not completed an implementation of ROCKSTER that perfectly preserves rock edge contours across subimages. However, we have assessed the communication overhead required for parallel ROCKSTER to transfer edge information from one core to its neighbor. Figure 7 shows messages per image, across number of cores. Although not shown, the messages required for image strips (as opposed to tiles) is three times greater on average. This finding is consistent with our empirical discovery that long horizontal subimages cut across far too many rock boundaries when more than eight subimages are used. The Tilera architecture is such that data can be sent from one core to its neighbor in a few clock cycles. Thus, we expect the impact of core-to-core communications to be negligible, even for the most demanding Martian terrains.

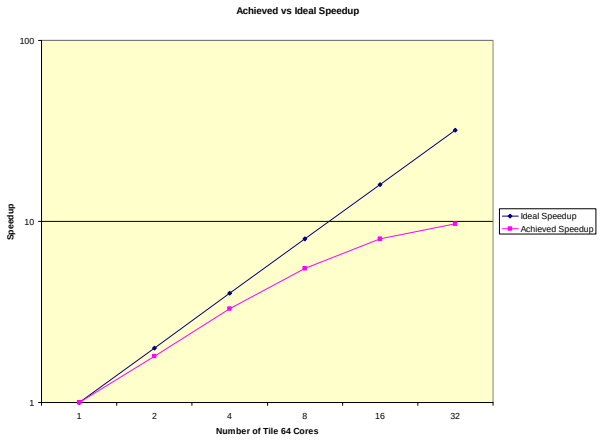

Figure 6 – ROCKSTER Speedup versus Ideal

Parallel Integer ROCKSTER average speedup versus the ideal speedup is plotted above. The achieved speedup curve suggests ROCKSTER performance asymptotes at approximately 10-fold. Memory and cache bottlenecks are likely culprits, though investigation is ongoing.

8. SUMMARY AND CONCLUSIONS

We are adapting three high-level autonomy capabilities for current and future rover surface missions to a multicore processor and performing detailed performance evaluations. We chose each capability for its unique, and complementary, parallelization challenges. The first capability is the ROCKSTER rock and target detection algorithm, which is currently in use onboard the MER Opportunity rover as a critical part of the larger AEGIS onboard science autonomy system. ROCKSTER has many of the hallmarks of a classic machine vision algorithm and as such, is well suited to data parallelization. By parallelizing for the TILE64™ multicore platform, we have made a significant step towards flight qualifying the code for the radiation hardened 49-core MAESTRO processor based on the same Tilera architecture.

After converting ROCKSTER to mostly integer operations, in order to cope with the lack of floating-point hardware on the TILE64™, we observed an approximately 10-fold decrease in average algorithm runtime at a maximum of 32 cores. However, compared to the ideal performance improvement with perfect scaling, ROCKSTER performance falls short. We suspect this is due to memory access patterns and inefficient use of cache, both of which are currently under detailed investigation. Communicating edge information from one core to its neighbor is challenging from an implementation perspective, but initial results indicate the runtime impact will be minimal.

REFERENCES

[1] T. Estlin, B. Bornstein, D. Gaines, D.R. Thompson, R. Castaño, R.C. Anderson, C. de Granville, M.C. Burl, M. Judd, and S. Chien, "AEGIS Automated Targeting for the MER Opportunity Rover," 10th International Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS), September 2010.

[2] R. Castano, T. Estlin, R. C. Anderson, D. Gaines, A. Castano, B. Bornstein, C. Chouinard, M. Judd, "OASIS: Onboard Autonomous Science Investigation System for Opportunistic Rover Science," Journal of Field Robotics, Vol 24, No. 5, May 2007.

[3] M.C. Burl, "Rockster: Rock Segmentation Through Edge Regrouping," JPL Internal Memo B-0120.

[4] Maki, J. N., et al., "The Mars Exploration Rover engineering cameras," Journal of Geophysical Research, 108(E12), 8071, 2003, doi:10.1029/2003JE002077.

[5] J. Canny, "A Computational Approach to Edge Detection," IEEE T-PAMI, vol. 8, pp. 679-698, 1986.

[7] Tilera Corporation, "Tile Processor Architecture Overview," Doc #UG100 Release 1.0, October 2007.

[6] Tilera Corporation, "Multicore Development Environment User Guide," Doc #UG201 Release 1.2, February 2008.

[8] M. Kang, E. Park, M. Cho, J. Suh, D.-I. Kang, and S. P. Crago "MPI Performance Analysis and Optimization on TILE64™/Maestro," Workshop on Multicore Processors for Space – Opportunities and Challenges (held in conjunction with SMC-IT), Pasadena, CA, July 2009.

BIOGRAPHY

Ben Bornstein *is a senior member of the Machine Learning and Instrument Autonomy group at the Jet Propulsion Laboratory in Pasadena, CA. He has over 10 years experience in developing onboard autonomy software for spacecraft and instruments. Ben led the instrument flight software and onboard autonomy software efforts for the Vehicle Cabin Atmosphere Monitor (VCAM), a GC/MS instrument currently monitoring the crew cabin atmosphere onboard the International Space Station (ISS). He led the development of the onboard pattern recognition software for JPL's most recent Electronic Nose (ENose). Ben is currently a technical lead for the AEGIS Project, which is providing new automated targeting technology for the MER rovers. He helped develop the MER cloud and dust-devil detectors, an atmospheric science autonomy technology uplinked to the MER rovers in 2006. Ben enjoys bringing machine learning techniques and considerable hacking (programming) skills to bear to solve problems in geology, remote sensing, chemistry, bioinformatics, and systems biology. Ben received a B.Sc. in Computer Science from the University of Minnesota Duluth in 1999.*

Tara Estlin *is a senior member of the Artificial Intelligence Group at the Jet Propulsion Laboratory. She has over 10 years of experience in developing spacecraft autonomy software. A primary goal of these efforts is to support onboard sequencing and opportunistic science handling for future rover missions. She is currently leading the AEGIS Project, which is providing new automated targeting technology for the MER rovers. Dr. Estlin is alsopresently a rover driver for the Mars Exploration Rover (MER) mission where she is responsible for sequencing drive and arm deployment commands for the MER Spirit and Opportunity rovers. She holds a B.S. in computer science from Tulane University and a Ph.D. in computer science from the University of Texas at Austin.*

Brad Clement *is a senior member of the Artificial Intelligence Group at the Jet Propulsion Laboratory in Pasadena, CA. He received a bachelor degree in computer engineering from the Georgia Institute of Technology and M.S. and Ph.D. degrees in computer science and engineering from the University of Michigan, Ann Arbor. His interests include artificial intelligence, planning, scheduling, multiagent coordination, robotics, abstraction, execution, distributed systems, real-time systems, integrated AI, control architectures, and AI in games.*

Paul Springer *is a senior researcher in JPL's High Capability Computing and Modeling group. He has very broad experience in developing and modifying applications to run on parallel hardware, as well as writing simulators of advanced parallel computers, such as the IBM BG/L and advanced processor-in- memory architectures. He has spent most of the past two years working with a TILE64™ development system, modifying applications to run on it, and analyzing their performnce. He has filed 20 NASA new technology reports, and authored or co-authored over 20 papers, as well as a book.*