APPLICATIONS ON HIGH PERFORMANCE CLUSTER COMPUTERS

Production of Mars Panoramic Mosaic Images[1]

Tom Cwik, Gerhard Klimeck, Myche McAuley, Robert Deen and Eric DeJong
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109

1. INTRODUCTION

The development, application and commercialization of cluster computer systems have escalated dramatically over the last several years. Driven by a range of applications that need relatively low-cost access to high performance computing systems, cluster computers have reached worldwide acceptance and use. A cluster system consists of commercial-off-the-shelf hardware coupled to (generally) open source software. These commodity personal computers are interconnected through commodity network switches and protocols to produce scalable computing systems usable in a wide range of applications. First developed by NASA Goddard Space Flight Center in the mid 1990s, the initial Caltech/JPL development resulted in the Gordon Bell Prize for price-per-performance using the 16-node machine *Hyglac* in 1997 [1]. Currently the JPL High Performance Computing Group uses and maintains three generations of clusters including *Hyglac*. The available hardware resources include over 100 CPUs, over 80Gbytes of RAM, and over 600Gbytes of disk space. The individual machines are connected via 100Mbit/s and 2.0Gbit/s networks.

Though the resources are relatively large, the system cost-for-performance allows these machines to be treated as 'mini-supercomputers' by a relatively small group of users. Application codes are developed, optimized and put into production on the local resources. Being a distributed memory computer system, existing sequential applications are first parallelized while new applications are developed and debugged using a range of libraries and utilities. Indeed, the cluster systems provide a unique and convenient starting point to using even larger institutional parallel computing resources within JPL and NASA.

A wide range of applications has been developed over the span of three generations of cluster hardware. Initial work concluded that the slower commodity networks used in a cluster computer (as compared to the high-performance network of a non-commodity parallel computer) do not generally slow execution times in parallel applications [2]. It was also seen that latency tolerant algorithms could be added to offset the slower networks in some of the less efficient applications. What followed was the development or porting of a range of applications that utilized the clusters' resources. End benefits include greatly reduced application execution time in many cases, and the availability of large amounts of memory for larger problem sizes or greater fidelity in existing models. The applications can be characterized into the following classes: a) *Science data processing*: these applications typically exploit the available file systems and processors to speed data reduction; b) *Physics-based modeling:* these applications typically use large amounts of memory and can stress the available network latency and bandwidth; and c) *Design environments:* cluster computer resources can be integrated into larger software systems to enable fast turnaround of specific design or simulation components that otherwise slow the design cycle.

The heavy use of clusters for a variety of applications requires the development of a cluster operation and maintenance infrastructure. This includes the use of commercial or open source tools and libraries. Key components involve the integration of message passing libraries (MPI) with a variety of compilers, queuing systems for effective resource utilization, utilities to monitor the health of the machine and the use of networked file systems attached to the cluster.

The rest of this paper describes the cluster machines used for a wide variety of applications, and then discusses a image data processing application in more detail. Specifically, the process for creating Martian surface mosaic images from a collection of individual images shot from a rover or lander camera will be discussed. The mosaic

[1] This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology under a contract with the National Aeronautics and Space Administration.

production and image pair correlation software was ported to a commodity cluster computer system as a proof-of-concept for near real-time processing of the downlink data. The application and algorithms, parallelization needed for use with the cluster and performance gains in using the clusters described above will be briefly outlined.

2. COMPUTING ENVIRONMENT

Three generations of cluster machines have been assembled within the High performance Computing Group at JPL. The first machine was built in 1997 and is named *Hyglac.* It consists of 16 Pentium-Pro 200MHz PCs, each with 128 Mbytes of RAM and it uses 100Base-T Ethernet for communications. Each node contains a 2.5 GB disk. The nodes are interconnected by a 16-port Bay Networks 100Base-T Fast Ethernet switch. *Nimrod*, assembled in 1999, consists of 32 Pentium-III 450MHz PCs, each with 512 Mbytes of RAM and it also uses 100BaseT Ethernet for communications. An 8 GB disk is attached to each node. A 36 port 3-Com SuperStack II 100Base-T switch interconnects the nodes. The third generation machine, assembled in 2001, is named *Pluto* and consists of 31 Pentium-III Dual-CPU 800MHz nodes (a total of 62 processors in all); each node has 2 GBytes of RAM. A 10 GB disk is attached to each node. The nodes are interconnected by the new 32 port Myricom 2000 networking hardware, capable of 240 Mbyte/s bi-directional bandwidth, and greatly reduced latency as compared to the 100Base-T Fast Ethernet switches. One big advantage of these cluster computers is their upgradability. We have for example increased the number of nodes on *Pluto* from the original 20 nodes to 32 nodes in the last half year.

All of the above clusters run the Linux operating system and use MPI for message passing within the applications. A suite of compilers is available as well as math libraries and other associated software. Since the machines are not used by a very large set of users, scheduling software has not been a priority. The Portable Batch System (PBS) for queuing jobs is available on *Pluto* [3]. Besides the compute nodes listed for each machine, a front-end node is also attached to the switch and consists of identical (*Pluto*) or faster CPU hardware as the compute nodes with larger disks, an attached monitor, CD drive and other peripherals.

3. DATA PROCESSING: MAPPING MARS

The Mars Exploration Rovers (MER) to be launched in 2003 rely on detailed panoramic views for their operation. This includes:

- Determination of exact location
- Navigation
- Science target identification
- Mapping

To prepare and test for MER operations, the Field Integrated Development and Operations (FIDO) rovers are being used. These FIDO rover cameras gather individual image frames at a resolution of 480x640 pixels and are stitched together into a larger mosaic. Before the images can be stitched they may have to be warped into the reference frame of the final mosaic because the orientation and the individual images change from one to the next, and because several final mosaics might be assembled from different viewpoints. The algorithm is such that for every pixel in the desired final mosaic a good corresponding point must be found in one or more of the original rover camera image frames. This process depends strongly on a good camera model.

A sequence of software operates within a data pipeline to produce a range of products – both for science visualization and in a mission operations environment. The original image stereo pairs captured by the rover cameras are sent to the data processing center and placed into a database. The image pairs are then sent through the pipeline producing the image products. It was recognized that two key software components that slow the data cycle are the production of a mosaic from a large number of camera image frames, and the process of correlating pixels between an image stereo pair. The following sections summarize the results for these two algorithms; additional results for the mosaic software can be found in [4] along with results for a wide range of applications executing on cluster machines.

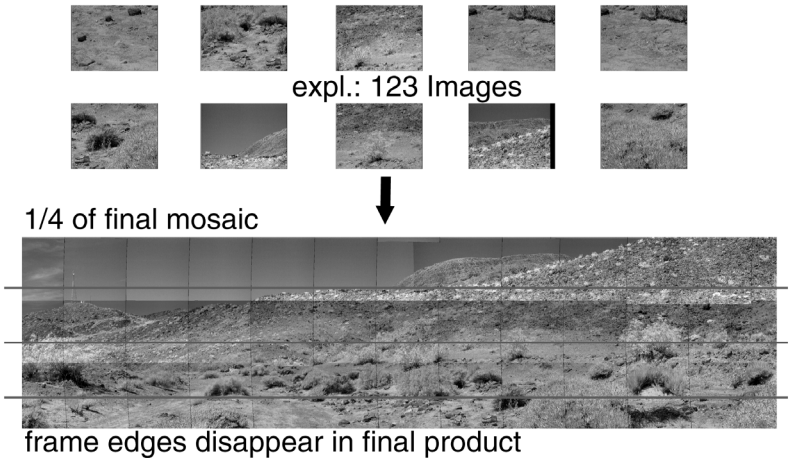

Figure 1: Mosaic generation from individual image frames. The horizontal lines in the panorama (lower image) indicate the strips of the image distributed to the cluster processors.

3.1 MOSAIC PRODUCTION FROM SETS OF IMAGES

The original algorithm executes in about 90 minutes, calculating a complete mosaic from 123 images on a 450MHz Pentium III PC running Linux. It was desired to reduce this processing time by at least an order of magnitude. Initial algorithmic changes to the original software were first performed. Using MPI the modified mosaic software was parallelized and run on the clusters. The processing time was reduced to a range of 1.5 to 6 minutes depending on the specific image set and the processor speed used in the cluster. The images shown in Figure 1 were taken from a FIDO Rover field test in the beginning of May, 2001. The mosaic generation for this particular image (note that only about 1/4 of it is shown) took 3.3 minutes on 16 CPUs of a *Nimrod*-like cluster of 16 CPUs.

The original mosaic algorithm was written for machines that have a limited amount of RAM available, thus restricting the number of individual images that can be kept in memory during the mosaic process. With about 256 MB on a CPU one can safely read in all of the roughly 130 images and keep resident in RAM a copy of the final mosaic. The algorithm was changed to enable this with the aid of dynamic memory allocation. The original algorithm took about 90 minutes on a single 450MHz Pentium III CPU to compose 123 images into a single mosaic. The above algorithm changes as well as others result in a reduction of the required CPU time to about 48 minutes. Running the same algorithm and problem on an 800MHz CPU results in a time reduction to about 28 minutes (Figure 2).

To exploit the parallelism available on a cluster, the parallel mosaic algorithm divides the targeted mosaic into N slices, where N is the number of CPUs (indicated by the horizontal lines in Figure 1). Once each CPU has completed its tasks, it reports the image slice to the manager CPU, which then patches the slices together into one image and saves it to disk. Results of the parallelization of the mosaic algorithm is shown for the 800MHz cluster and for the 450MHz cluster in Figure 2. The dot-dashed line shows the ideal speed-up.

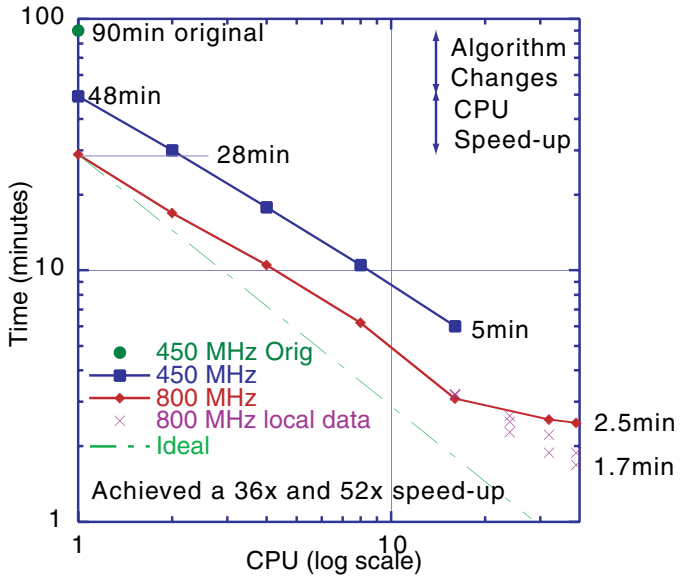

Figure 2: Timing examples for the assembly of 123 individual images into a single mosaic for two different clusters. Differences between the "800 MHz" and "800 MHz local data" indicate changes in how the input images are read from disk (see text).

The actual timings follow a linear scaling with deviations from the ideal attributed to load balancing problems and data staging problems. The 800 MHz curve extends to a larger number of CPUs since the 800 MHz cluster has twice as many CPUs available.

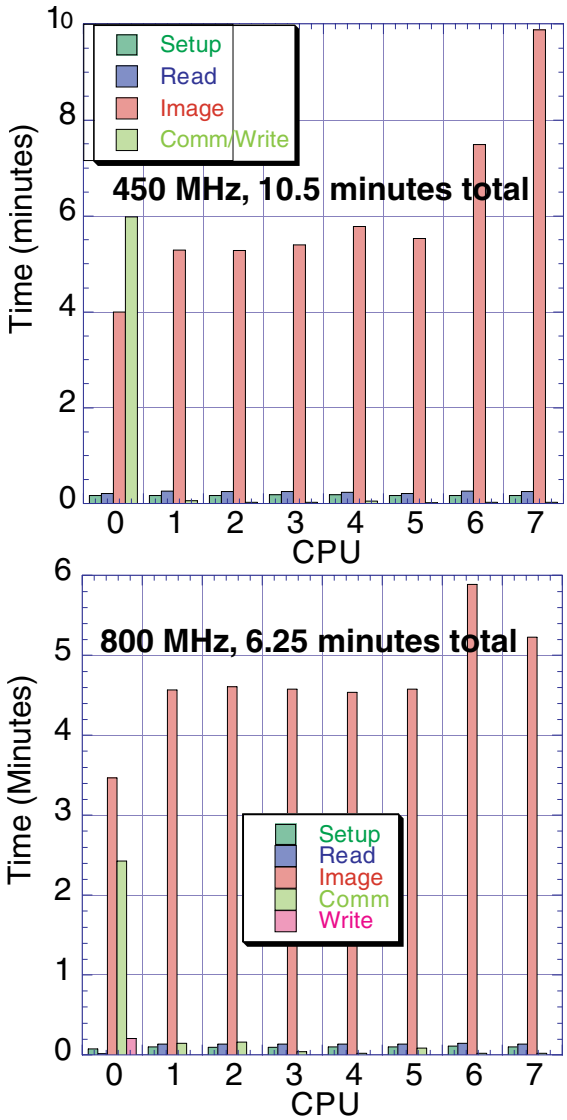

Figure 3: Timing analysis for runs on 8 CPUs on the 450MHz and 800MHz clusters. Times for set-up, image reading, image processing, communication and final image writing are shown.

Figure 3 plots execution time broken into problem set-up, reading of the data, image processing, communication between CPUs and writing data to disk for two different clusters (450MHz and 800MHz). It is seen that most of the time is spent on the actual image processing. What is also noted is that different CPUs work on the problem for differing amounts of time. One solution to the load-balancing problem is a more careful analysis of the image data staging in the search algorithm portion of the processing. An approach that is more independent of the image sequencing and data staging can be a master-slave approach, where the work is dished out to the worker CPUs in smaller chunks in an asynchronous fashion. As some CPU's finish their chunk before others, they can start working on the next chunk. It is interesting to note that the load-balancing problem is, relatively speaking, smaller for the faster CPUs as compared to the slower CPUs. The apparently large communication cost on CPU 0 includes idle time waiting for data from CPUs 6 and 7.

The parallel algorithm deteriorates strongly starting at 24 CPUs. This can be attributed to data staging problems to all the CPUs. If the images are copied to the local disks on each node of the cluster (rather than residing on the front-end disk) the overall performance is significantly improved (crosses in Figure 2). The total processing time is reduced from 2.5 to 1.7 minutes. This comes at the expense of about 7 minutes to copy the data to the local disks via the UNIX rcp process, though in an operational setting a different strategy for data staging may be available. This implies that if the algorithm is to be run on that many CPUs a different method and/or hardware must be found to move data to the local disks.

Figure 4 shows data for the time spent in the set-up, the initial file reading, image processing, communication and final image writing for each individual CPU of *Pluto*. With 39 CPUs, only about 1.5 minutes is spent on the actual image processing. The remaining 0.8 minutes are mostly spent on set-up and reading of the original images. The reading of all the input images by all the CPUs from the front-end disk leads to a dramatic bottleneck in the overall computation.

The heavy disk load on the front-end can be reduced by copying all the images from the front-end machine's single disk to the local /tmp disks of the computation nodes. Figure 4 (bottom) shows the virtual elimination of the previously significant read time. Interestingly, we have also significantly reduced the time declared as set-up. We believe that this is due to the fact that during the set-up time all the input files are probed for their individual size and coordinates, which are needed to define the size of the final mosaic. This double access to the images became apparent to us in the analysis of this data. Additional improvements to the algorithm to only read a file and/or its header once are clearly possible. The copying of all the individual images to the local disks comes with a significant price in performance: an *rcp* shell command takes about 7 minutes to execute. Clearly that is not an efficient

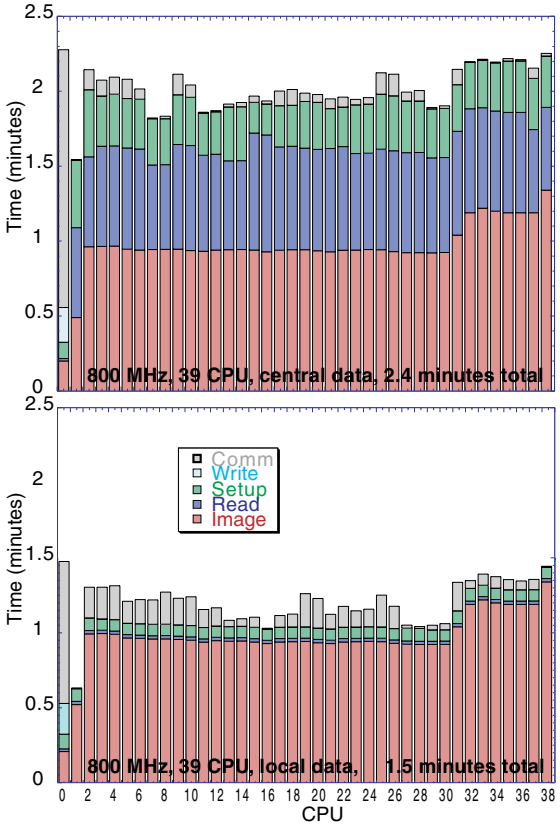

Figure 4: Timing analysis on 39 CPUs running at 800MHz. Top: images are all stored on the front end. Bottom: input images are distributed to local /tmp disks before the processing.

solution. Possibly faster I/O hardware such as a RAID disk or a parallel file system might be solutions to this problem.

Looking at the final performance data with the local data one can see again a load-balancing problem. However, squeezing out the last 10-20% performance by balancing this load and reducing the total time from 1.5 minutes to perhaps 1.3 minutes may prove to be laborious and not necessary as the platform this code will be run on during the Mars Exploration Rover mission is not completely defined at this time.

3.2 CORRELATION OF STEREO-PAIR IMAGES

A second step in the data processing pipeline that exhibited strong opportunities for parallelism is the algorithm for correlating stereo-pair images. This algorithm takes a pair of images from the rover stereo cameras and attempts to correlate pixels in the left and right images. The correlation is necessary for calculation of range and terrain maps. (Figure 5 shows a representative pair of images used in the FIDO tests.) The algorithm starts from a pre-defined seed point defining a pixel in the left image, and attempts to correlate this pixel and a set of surrounding pixels with those in the right image. A camera model that gives information relating the geometry of the two cameras is used to find the starting location in the right image given a pixel in the left image. Starting at the seed point the correlation process spirals outward until new pixels can no longer be correlated with each other. The algorithm then begins anew with a different seed point and continues attempting to correlate pixels that have not been previously processed. In between the main correlation stages, a pass is made along the areas not previously correlated to complete the correlation in those sections of the image pair. This stage is referred to as filling the gores in the image. Generally not all pixels can be correlated, due to the different view angles of the two cameras. For example in Figure 5 the left eye sees an area of ground between the lower ramp and the left of the rover that the right eye does

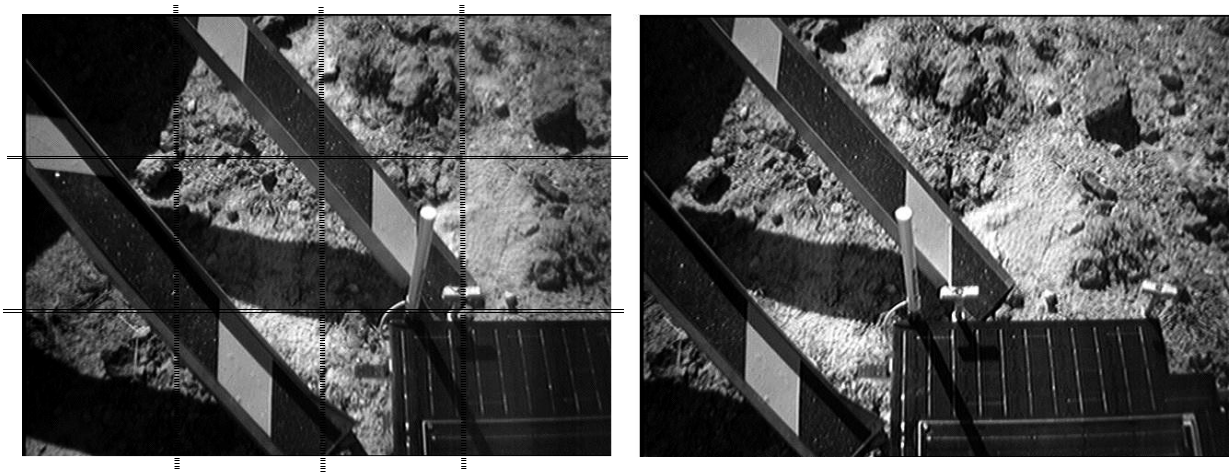

Figure 5. Left and right stereo pair used in correlation example. In this example the lines indicate subframes of the workload for the left image distributed to 12 processors of the cluster.

not see at all.

To exploit the parallelism available on a cluster, the stereo-pair images were divided into subframes depending on the number of processors in use. Figure 5 shows a decomposition for 12 processors–in general the decomposition results in anywhere from equal numbers of column and row subframes to a ratio of 3:1 (columns:rows) subframes for some numbers of processors. E.g., a ratio of 12:4 for 48 processors in use. Though each processor holds the complete pair of images, it is the computational work in the left image that is divided among the processors as indicated by the subframes shown in the left image of Figure 5. For each subframe of the left image in a processor, a seed point is generated for the left frame and the algorithm for correlating pixels between images begins. Since the pixel in the left frame can generally correlate to a pixel located anywhere in the right frame the whole right image is available to to the correlation algorithm. Search window size and correlation window size are input parameters to the correlator. The location of a seed point is chosen randomly in each subframe with the number of seed point passes in the algorithm a variable at runtime. Because the seed point and the sequence of operations in the parallel correlation algorithm is different than those in the sequential algorithm, it is expected that both the number of pixels correlated as well as the quality of correlation in the images will be differ slightly depending on the number of processors in use.

Figure 6 is a plot of the scalability and number of pixels correlated for the correlation algorithm executing on *Pluto*. The timing at one processor indicates the original algorithm and seed points on one processor of the cluster, while the additional points show the timing and number of points correlated for the parallel algorithm and random seed point generator. The overall execution time was reduced from over 38 minutes on 1 processor to 1.6 minutes on 36 processors. The number of pixels considered correlated varied slightly as the number of processors in use was increased but stayed within 5% of the number correlated in the original sequential algorithm. This number could be increased slightly by increasing the number of seed point passes in the algorithm. The quality and number of pixels correlated in the images by the parallel algorithm as referenced to the original sequential algorithm is currently being examined.

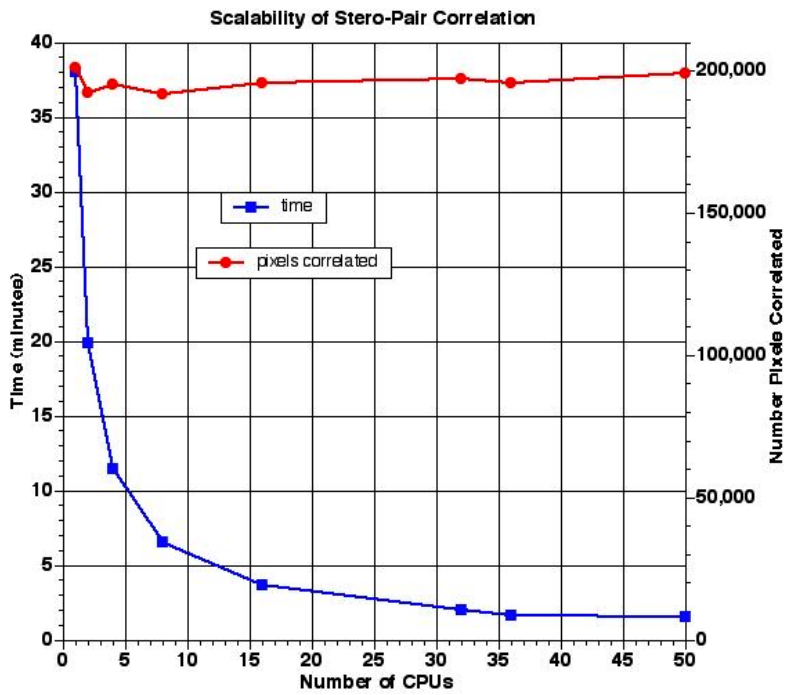

Figure 6. Scalability of stereo-pair correlation algorithm on cluster *Pluto*. Axis on left indicates execution time for the correlation algorithm while axis on right indicates the total number of pixels correlated in the image pair.

Figure 7 is a plot of the timing and number of pixels correlated per processor for 36 processors on Pluto. The timings are broken into the main and gore portions of the algorithm as described above. The number of pixels correlated as well as the time per processor is seen to vary across the processors. It is clear that a refined version of the parallel algorithm can be developed that balances the workload across processors. As in the mosaic generation algorithm, it is unclear if the savings of less than a minute of execution time is necessary until further requirements are defined.

Figure 8 shows an overlay of the left camera image shown in Figure 5 with a raster of 36 subdivisions corresponding to the data in Figure 7. The yellow colors correspond to the original image, the blue overlay indicates a "successful" correlation as returned by the serial code. Almost all the area of the image that is worked on by CPUs 31 through 36 is not even seen in the right image. That is why a load balance problem is evident in Figure 7. It is also interesting

to note that the segment indicated by CPU 34 does show "successful" correlation on the ramp, which is not even shown in the right image. We are currently implementing a left-right and right-left correlation verification algorithm that will eliminate such dramatic errors [5].

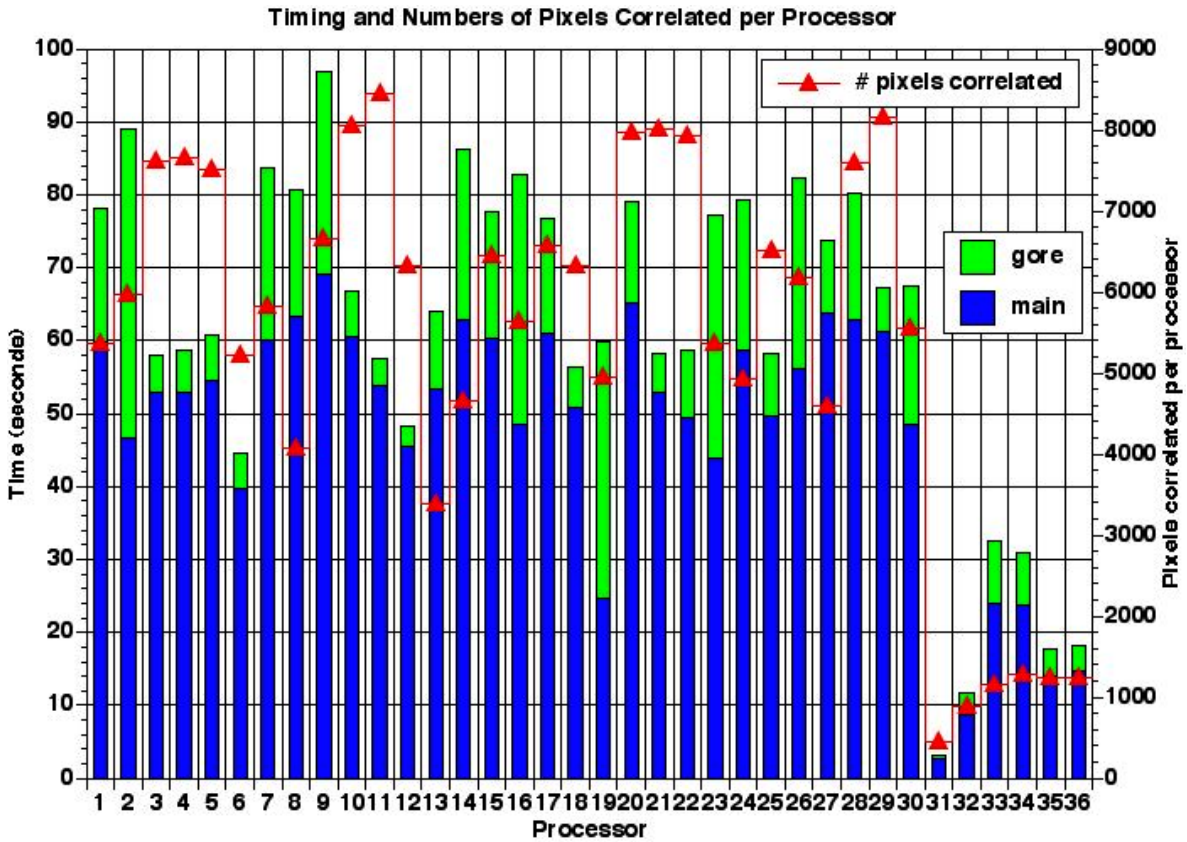

Figure 7. Timing and number of pixels correlated per processor for 36 processors of Pluto. The timing is broken into the main and 'gore' portion of the algorithms as described in the text. All other portions of the code result in less than a second of execution time. The number of pixels correlated on each processor is indicated on the right axis.

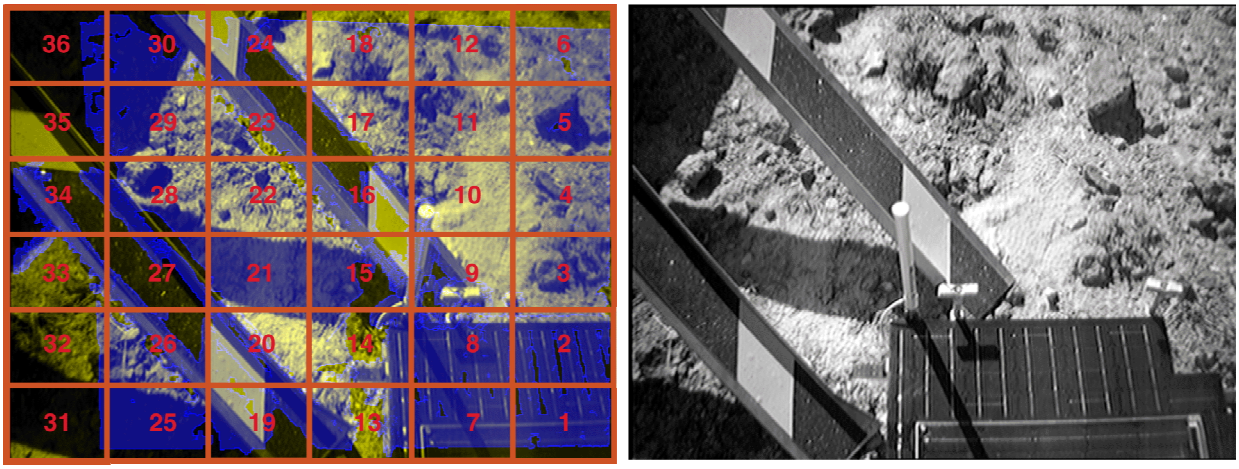

Figure 8. Similar to Figure 5. Left image is subdivided into 36 CPU work loads. Blue overlay is the result of the serial code correlation mask which indicates successfully correlated pixels.

4. CONCLUSION

This paper has summarized the use and performance of cluster computers for processing of stereo-pair camera images to produce science data products and images used for mission operations. The mosaic production software and the correlation software was ported to the cluster computer environment and executed for representative images used in recent field tests to determine performance and fidelity of processing. Due to the inherent parallelism of the algorithms, the reduction of processing time is large. The mosaic production was reduced from about 28 minutes to under 2 minutes using the cluster while the correlation algorithm was reduced from 38 minutes to under 2 minutes. The final mosaic image produced is identical in the original sequential and parallel environment, while the correlations produced in the two environments differ slightly due to variations in the processing order inherent in the parallel algorithm. These differences are currently being examined and will be quantified [5].

5. ACKNOWLEDGMENTS

The TMOD technology program under the Beowulf Application and Networking Environment (BANE) task sponsored the Mars imaging work. The original VICAR based software is maintained in the Multi-mission Image Processing Laboratory (MIPL).

6. REFERENCES

[1] Michael S. Warren, John K. Salmon, Donald J. Becker, M. Patrick Goda, Thomas Sterling, Gregoire S. Winckelmans, Pentium Pro Inside: I. A Treecode at 430 Gigaflops on ASCI Red, II. Price/Performance of $50/Mflop on Loki and Hyglac, SC97 Conference Proceedings, 1997.

[2] D. S. Katz, T. Cwik, B. H. Kwan, J. Z. Lou, P. L. Springer, T. L. Sterling and P. Wang, "An Assessment of a Beowulf System for a Wide Class of Analysis and Design Software," Advances in Engineering Softw, vol. 29, pp. 451-461, 1998.

[3] http://www.OpenPbs.org/.

[4] Tom Cwik, Gerhard Klimeck, Myche McAuley, Charles Norton, Thomas Sterling, Frank Villegas and Ping Wang, The Use Of Cluster Computer Systems For NASA/JPL Applications, AIAA Space 2001 Conference and Exposition, Albuquerque, New Mexico, 28 - 30 Aug 2001.

[5] http://www-hpc.jpl.nasa.gov/PEP/gekco/mars , Parallel Algorithms for Near-Realtime Visualization, Gerhard Klimeck.