Terrain Correction of Synthetic Aperture Radar Imagery Using the Cray T3D

Thomas Logan, Alaska SAR Facility, Fairbanks, Alaska

ABSTRACT: *A parallel terrain correction algorithm for ERS1 SAR images was implemented using C and the PVM message passing libraries on a CRAY T3D at the Arctic Region Supercomputing Center. Algorithm optimizations focused on load balancing among individual processors and between I/O and processing times. Scalability and efficiency of the programs were examined, showing 8 processors gives the best efficiency for terrain correction of SAR images at 90 meter pixel resolution, while the programs with large processing kernels show over 90% scalability up to 32 processors. With 8 processors, the parallel implementation executes 40 times faster than the original code, reducing processing time for a full-res SAR image from over 4.5 hours using Sun workstations to under 7 minutes.*

1 Image Distortions

Synthetic Aperture Radar (SAR) images have severe geometric and brightness distortions over elevated and sloping terrain due to the nature of the SAR range mapping and reflectance functions[1]. Before SAR images can be used for accurate surface mapping and change detection, these distortions must be removed through a process called terrain correction. Once distortions are removed, the all-weather capability and fine ground resolution available from satellite SAR instruments provide scientists with a powerful tool. The ability to perform terrain correction on SAR images facilitates a great variety of land applications of satellite SAR imagery, including the areas of land mapping, geology, forestry, hydrology, glaciology, and volcanology.

SAR imaging requires precise determination of the relative position and velocity of the radar platform with respect to its target at all times. However, this information is not available at the time of imaging, since the platform must be moving in order for the azimuth resolution technique to work. The basic solution is to assume that the platform has a uniform velocity over a smooth geoid. This simplifies the image creation considerably, but introduces certain distortions into the results.

When there is significant relief in the area being imaged, a considerable target height error will result from the smooth geoid assumption[2]. This height error, when projected onto the slant range image, gives rise to a slant range error. The slant range error in turn gives rise to a cross-track displacement error in the ground range image, called the absolute location error (see Figure 1). This distorts pixel placement in an image, rendering it unsuitable for quantitative analysis of terrain features. In addition, the smaller the look angle, γ, of the SAR beam, the larger this distortion becomes. For example, in the near range of ERS-1 SAR images, each meter of height estimation error results in nearly 3 meters of cross-track displacement error.

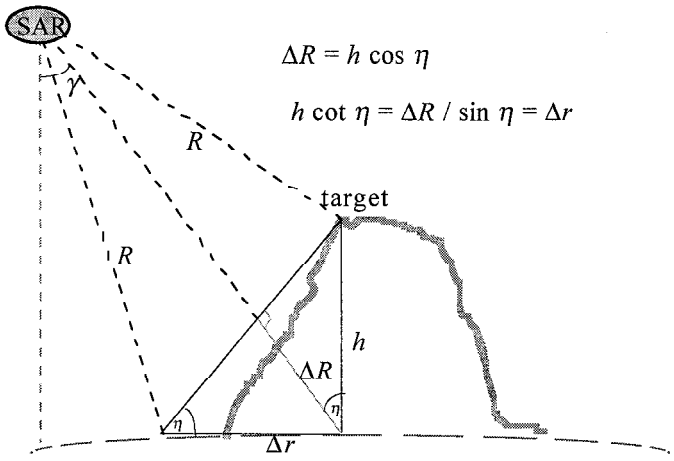

Figure 1: Height Error, $h \Rightarrow$ Slant Range Error, $\Delta R \Rightarrow$ Location Error, ΔR. For the flat earth model, γ is equal to η. Also, the assumption is made that $R >> \Delta R$.

2 Terrain Correction Process -- Theory

To remove the geometric distortions due to terrain relief, this research adapted an algorithm originally developed by Charles Wivell at the USGS Eros Data Center[3]. The algorithm uses a SAR image and a corresponding Digital Elevation Model (DEM) as inputs to create a simulated SAR image based on an imaging radar model. Then, the simulated SAR image is spatially correlated with the actual SAR image. Once the best correlation is achieved, the radiometric values of the actual SAR image are placed by inverting the mapping from the DEM to the SAR image.

The first correction applied is *radiometric calibration*. Ideally, this process will adaptively compensate for all spatial and time dependent variations in the radar system transfer characteristics. Included here are cross-track and image to image intensity inconsistencies due to signal attenuation by distance. Conversion from echo received to mean surface backscatter establishes a common basis for all pixel intensities, giving unique backscatter that is independent of positioning in an image or of the particular image viewed [4].

To correct image orientations, SAR images are *geocoded*. Here, the SAR image is resampled to a spatial representation with known geometric properties. Standard map projections, like the Universal Transverse Mercator (UTM) or longitude/latitude mapping, are used. Processing involves rotation and scaling of an image to properly transform it into the mapping coordinates chosen. Since images are taken at varying pass angles, each *geocoded* image contains an approximately square rotated SAR image inside it, with unused image pixels set to black. An example of this is given in Figure 2.

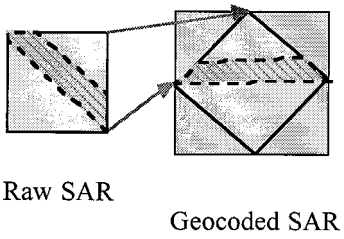

Figure 2: *Geocoded* SAR image

Another correction applied is *geometric rectification*. This process resamples an image to remove the geometric distortions. It is accomplished by applying adjustments that compensate for the difference between actual and estimated terrain height for a particular image. The results are undistorted images in the sense of orthographic maps. Therefore, these images are suitable for use in geographic distance calculations and size measurements. The resulting images appear with correctly spaced and placed land marks, and mountains that have properly sized slopes.

3 Implementation Details

The terrain correction software package prepared at ASF is a standalone set of nine programs and their subroutines. The parallel implementation uses C and the T3D version of the parallel virtual machine (PVM) message passing routines [5]. Programs are called by system calls from a C main program.

A flow graph for the terrain correction process is given in Figure 3. The inputs are a raw SAR image with its metadata and a LAS [6] format DEM file clipped to the area corresponding to the coverage of the SAR image and mapped to the UTM projection. The output is a terrain corrected SAR image.

A project of this magnitude involves many implementation choices. When the goal is to use a parallel machine to speed

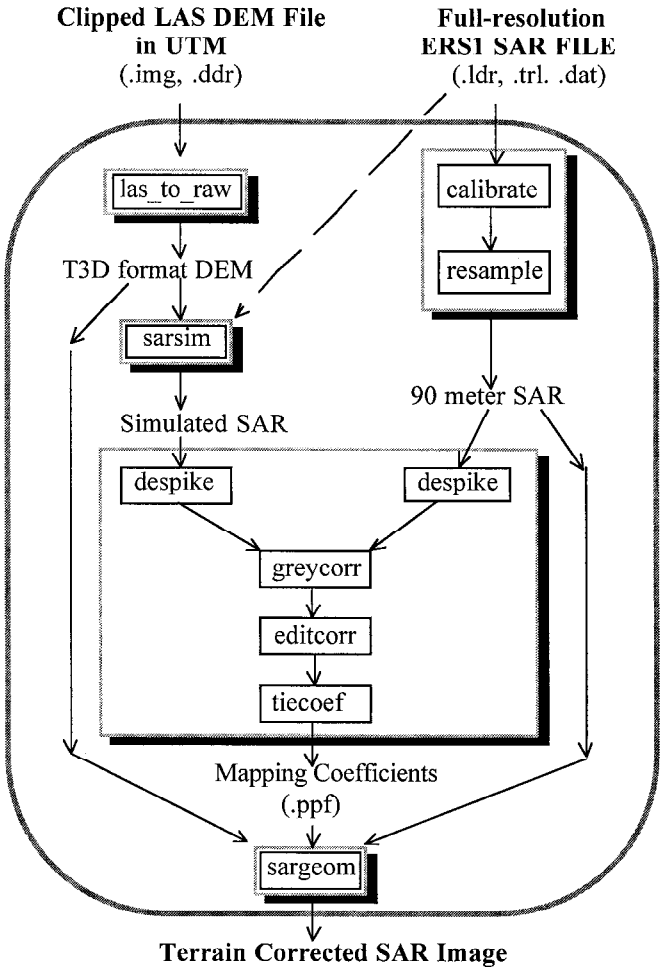

Figure 3: Terrain correction flow graph

processing, one must have suitably parallel problems. Next, an efficient strategy to exploit the parallelism must be established. Finally, correct latency-hiding techniques for the parallel architecture must be applied. A summary of the choices made for this project is given in Figure 4. The terms used in the table to describe the parallel implementation choices are defined below.

Work Sharing. The simplest parallel processing paradigm is sharing, where each processor is an equal partner in a group effort. Data-sharing involves each processor determining its own piece of the input data based on the total number of processors. This leads to work-sharing, where the work is split into equal pieces across the processing elements.

Most of the first draft programs for this project were rapidly prototyped using this combined sharing paradigm. As long as iterations of the main loop are neither output nor control dependent, one can utilize multiple workers simply by distributing these iterations. Independently, the workers determine the number of loops to be accomplished, the total number of workers, and their own starting and ending iteration numbers.

Module	Implementation
calibrate	exclusive input, tightly-coupled synchronization, submission output pre-reads input, buffered output, multiple line sends
resample	shared input, non-coupled synchronization, submission output pre-reads all input
sarsim	exclusive input, loosely-coupled synchronization, cooperative output pre-reads input, multiple line sends, buffered output
despike	work sharing: shared input, non-coupled synch, cooperative output pre-reads all input, buffered output
greycorr	exclusive input, tightly-coupled synchronization, submission output pre-reads all input
sargeom	exclusive input, loosely coupled synchronization, submission output pre-reads input, multiple line sends, buffered output
editcorr	Y-MP
tiecoef	Y-MP
las_to_raw	T3D Serial Code

Figure 4: Implementation choices

Once a processor's iterations are known, the appropriate input data can be read.

As program optimizations progressed, it became obvious that having global control of input or synchronization was necessary in most cases. This was due mainly to the I/O contention that arises when multiple processors try to access a single file.

Master-Slave Processing. To reduce I/O contention and provide synchronization, the master-slave paradigm was used. One master processor controls the distribution of work, distribution of processing parameters, and collection of output data. The remaining processors are the slaves, also called workers. The slave processors get their name from the functions they typically fulfill, which are to wait for work assignments from the master, perform the assignments, and return the results to the master. The amount of control the master has over the slaves varies by algorithm according to the circumstances. Several different input, output, and synchronization strategies were used in this project.

I/O Strategies. Input strategies are termed either exclusive or concurrent. In the former case, the master is the exclusive input agent, performing setup and reading data files as needed. When using concurrent input, each worker is allowed to read its data directly from the input file. Rather than receiving work distributions from the master, inputs are split equally among the workers as in sharing.

In many applications, output from parallel programs must be performed exclusively by a single processor because the output must be ordered in a certain way. In exclusive output, one processor performs all output. However, the means of gathering and ordering output varies. One version is cooperative output. Algorithms in this class allow workers to retain results until processing is completed, then results are combined in an additional processing stage. This reduces the overall message overhead, but may tend to flood the master with messages. To alleviate this bottleneck, a binary tree summation of the outputs may be performed. Another output scheme, submission, requires the master processor to have immediate responses from the workers. If input images are processed from the first line to the last, regular submission of results naturally lends itself to creation of a properly ordered output file. Each time an output line is received by the master, an input line is sent to the slave, keeping a steady work load on the system. This allows the master's output combination time to be hidden in the processing time of workers.

Synchronization. Synchronization strategies are the most varied, being tightly, loosely, or non-coupled. Tightly-coupled algorithms use lock step synchronization on each iteration of a processing loop. The master will only accept output messages from workers in the proper order. Similarly, the input messages are sent in order. This keeps all processors in step at all times. Loosely-coupled synchronization occurs when workers receive assignments from and return outputs to the master, but each proceeds at its own pace. The master receives output lines in any order and sends input lines to the first idle worker. Non-coupled algorithms do not require synchronization in main processing loops. In one version, after the master sets up processing, it merely receives output and writes it to disk. The other allows all processors (including the master) to work on separate input file sections, not synchronizing until the output is coalesced in a final processing stage. This is identical to sharing.

Latency-Hiding Techniques. Buffered reads and writes are incorporated in all of the parallel programs written for this project. Input data are always pre-read before needed, either in large blocks or by an entire file. This decreases the wait time inherent in master-slave processing. When slaves send results to the master, they must wait for their next assignments. If the master reads input data into RAM ahead of time, it can be sent immediately after a completion response is received from a worker. Having one processor ready at all times to send more input reduces the workers' wait times. Similarly, storage of output data until a large amount can be written to disk reduces overall access overhead and decreases total worker wait time.

Exclusive I/O for the master processor is the rule used in the T3D code. The ARSC T3D has only 4 I/O channels, two high speed and two low speed, for communication with the host Y-MP system. The Y-MP must respond to requests from the

T3D as well as from its own processes. This makes file I/O the slowest link in the processing chain. Thus, reduction of file contention represented a significant part of the optimizations performed for this project. Restricting metadata file reads to the master processor allows nearly constant algorithm set-up time. Since the metadata involved in terrain correction comes from three short files, having more than four processors simultaneously performing setup degrades performance to serial execution at disk access speed. Constant start-up time is one thing, but file contention occurs during processing as well. For this reason, exclusive reads are used in all but two of the programs.

Multiple line sends establish a balance between processing time and the overhead involved with processing. Each worker must take as long to process a single input message (receive, do work, return output) as it takes the master to process all other workers (receive and store output and read and send input). Otherwise, workers will have to wait to be serviced. A block processing method that provides scalable input message sizes is used for several of the T3D programs. This method introduces a variable called *block* which controls the number of input lines that are to be sent per message. By adjusting this variable, a balance can be achieved that virtually eliminates worker wait time, up to the maximum number of workers a single master can service.

4 Results

To properly analyze setup, wait, file, work, and output cooperation times in the parallel programs, each of these actions was surrounded by timer calls. Examples of sectioned T3D timings for the master processor and the slowest slave processor are given in Figure 5. The las_to_raw time was not subdivided since it operates in a serial fashion. The Y-MP times for editcorr and tiecoef were not subdivided or analyzed due to the small CPU times involved.

The same full resolution SAR scene and DEM file were used for all timings. All measurements give elapsed CPU times in seconds. IIAS program execution times were extracted from TAE log files produced during the terrain correction run. T3D times were obtained using rtclock() calls, which return a processor's clock counter value. Differences in successive calls divided by the clock rate (150 megahertz) gives CPU seconds. Y-MP times were gathered using the CRAY Hardware Performance Monitor (HPM) utility [7].

It should be noted that timings fluctuate somewhat due to varying response times for I/O requests through the Y-MP and the resulting impact of this variation on algorithm flow. In extreme cases, as much as 20 seconds variation was observed on the longer runs. Figure 6 lists the total time for an average of sample runs of each program. The IIAS calibrate and resample times are from a Sun 4/280 (16 MHz) server, the rest are from a Sun Sparc2 (40 MHz) workstation. The T3D programs were run using from 2 to 64 processors.

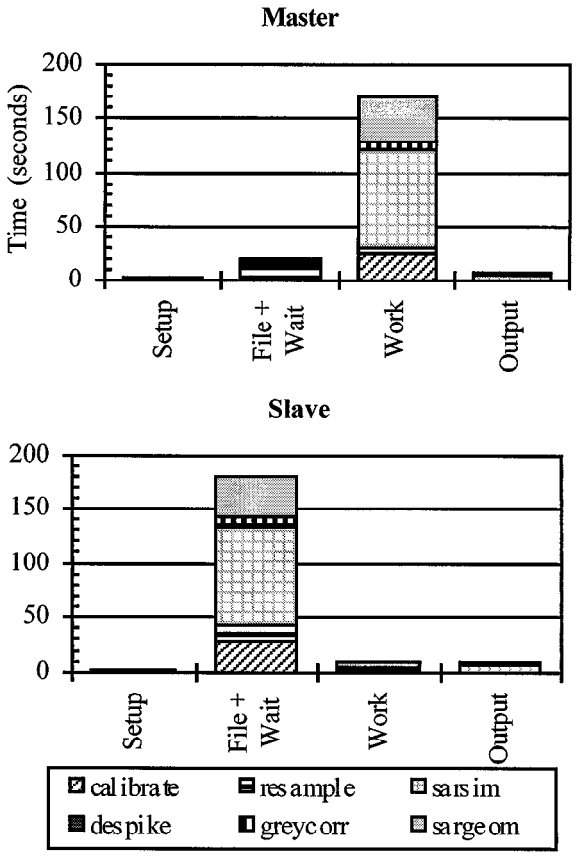

Figure 5: Example of sectioned timings using 16 T3D processors. Differences in master and slave tasks are immediately obvious, as are the large time portions used by calibrate, sarsim and sargeom.

Modules	IIAS	T3D 2	T3D 4	T3D 8	T3D 16	T3D32	T3D64	serial
calibrate	5897*	332	111	47	30	11	6	
resample	3789*	59	20	12	16	16	15	
las_to_raw								12
sarsim	4050*	1479	494	212	101	51	28	
despike	202◊	20	16	12	24	32	64	
greycorr	392◊	120	40	17	8	4	4	
editcorr	8◊							0.8
tiecoef	11◊							1.9
sargeom	1928◊	635	213	92	44	22	13	
TOTALS	16277	2659	908	408	230	151	146	

* Times for Sun 4/280 ◊ Times for Sun Sparc2

Figure 6: CPU Times (seconds). Totals for T3D versions include Y-MP and serial T3D run times.

5 Performance Measures

Several measures for processor performance are discussed in this section. For this code, a count of Millions of Instructions Per Second (MIPS) gives the most accurate performance measure because using Millions of FLoating point OPerationS per second (MFLOPS/sec) ignores the large quantity of integer and byte operations involved with byte image processing.

Either measure indicates the rate at which a single processor is executing, and, when compared with benchmarks, shows how well a program takes advantage of the architecture at hand. Instructions counts and performance measures for the base case runs were obtained from the Y-MP version of each program using the HPM utility (Figure 7). The total operations for a single run of the terrain correction code is estimated to be 105,618 million operations, or 105.6 giga-ops, of which 25 giga-ops are floating point operations.

	Instr ($*10^6$)	MIPS	FP ($*10^6$)	MFLOPS/s
calibrate	20541	62	4948	14.9
resample	5364	91	141	2.4
sarsim	49799	34	12831	8.7
despike*	612	61	78	7.8
greycorr	4815	40	1240	10.3
sargeom	22422	35	5536	8.7
editcorr	40	50	2.6	3.4
tiecoef	91	47	17	9.1
las_to_raw	98	8	2.0	0.2
Totals**	**105,618**	**39.7**	**25,029**	**9.41**

*Rates are halved since 2 PEs work in the base case. **Includes the four runs of despike required per terrain correction.

Figure 7: Instruction counts and base performance measures.

The MIPS and Mflops/sec calculations for each run, shown in Figure 8, were made using the instruction counts divided by the total times given in Figure 7. The parallel code achieves about 700 MIPS and 170 Mflops/sec for both 32 and 64 processors. The MIPS per worker and Mflops/(sec*workers) peak at 39.7 MIPS and 9.4 Mflops/sec on 2 processors, while values bottom out at 12 MIPS and 2.7 Mflops/sec on 64 processors.

The theoretical peak for the DEC alpha processor used in the T3D is 150 MIPS. Achieving 39.7 MIPS per processor shows 26% of this peak, which represents a high percentage of such theoretical numbers. The Linpack rating for a DEC alpha processor is 30 Mflops/sec [8]. However, the Linpack result is for an Alpha workstation with a secondary cache that is not present on the T3D. The result of 9.4 Mflops/sec for a program which consists of less than 25% floating point operations and requires considerable I/O time shows good processor usage.

Slight decreases in processor performance are observed for the 4 to 16 processor cases. This is due to the slowly increasing percentage for the constant serial time compared to the decreasing parallel times. Both of the performance measures drop off rapidly above 16 processors when calculated on a per worker basis. This show the dominance of the non-scalable programs (despike and resample) and the serial code (las_to_raw, editcorr, and tiecoef) when using a large number of workers.

The decrease in performance based on the increase of serial and non-scalable percentage of time is displayed in Figure 9. This shows the relative time used by the serial, non-scalable, and scalable programs, when using from 2 to 64 processors.

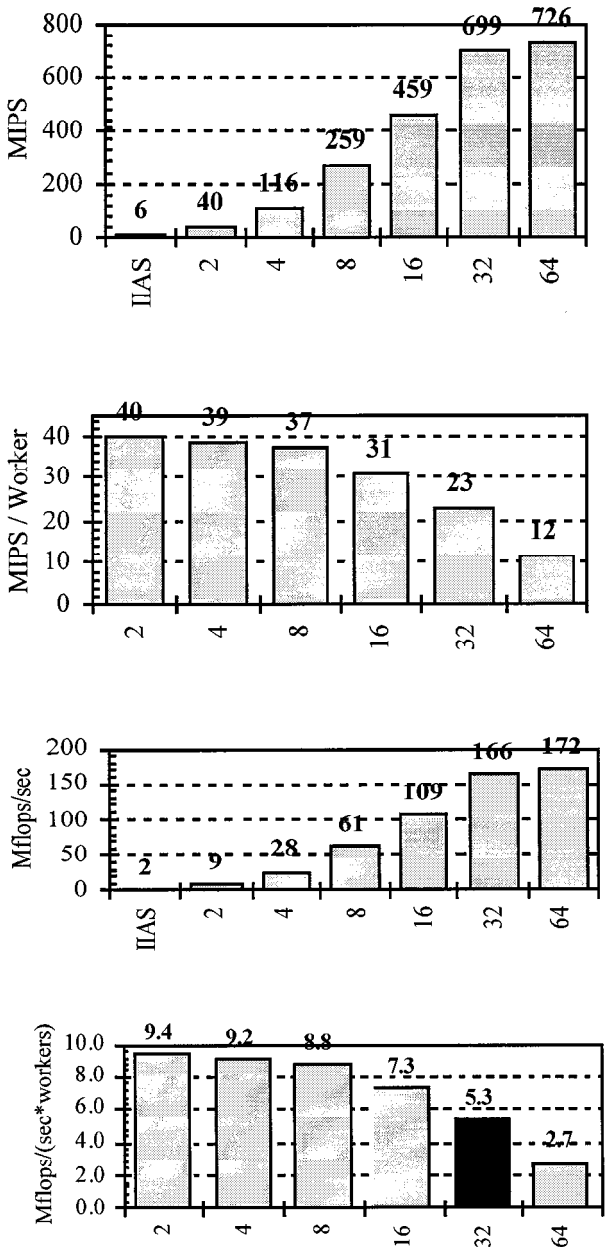

Figure 8: Performance measures for the ASF terrain correction code. A total of 105.6 billion operations are performed each run. The maximum of 726 MIPS is attained on 64 processors, while the maximum of 39.7 MIPS/worker occurs for the 1 worker (2 PE) case. The overhead of the master processor is ignored in these calculations.

Examining the large time slice used by the non-scalable programs in the last two runs, it becomes obvious where the time is being spent and why the performance measures fall off rapidly for these cases. It is interesting to note the that total time used by the non-scalable programs is 79 seconds for both the 2 and 64 processor runs. So, the combination of serial and non-scalable times are constant, while the scalable program

times decreased by a factor of almost 50. As the percent of serial time increases, total scalability decreases, regardless of the scalability of the individual programs.

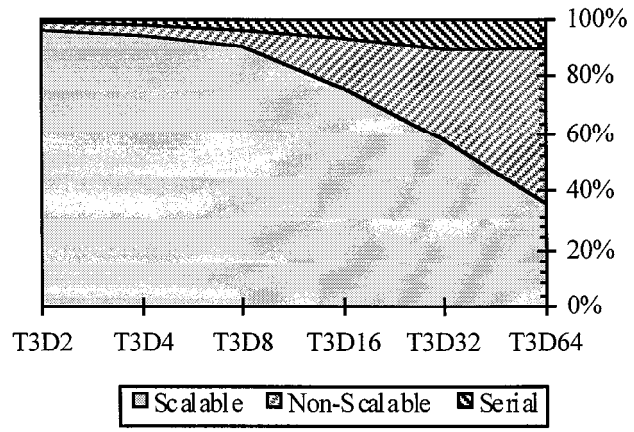

Figure 9: Percentage of time spent in parallel and serial portions of code. As the number of processors increases, the non-scalable parallel algorithms consume a greater portion of the processing time. As the total processing time decreases, the serial code portion of the time increases.

The speedup per processor, or processor efficiency, is graphed in Figure 10. These numbers are speedup / *PE* instead of speedup / *workers*, to include the overhead of having a master processor. Thus, it reveals that for more than 16 processors, the masters are no longer able to keep up with the workers (more than 16 processors) so efficiency is waning. A similar measure, the '128 PE image time,' shows the time per image if N jobs with p = 128/N PEs per job are run simultaneously (Figure 11). This time is found by scaling the time for a single run on p processors by the factor p/128. This gives the average time per result if an entire 128 node T3D were utilized. It points out the large cost of having an idle processor when only two or four total are used. Both Figures 10 and 11 affirm 8 to be the number of processors used most efficiently.

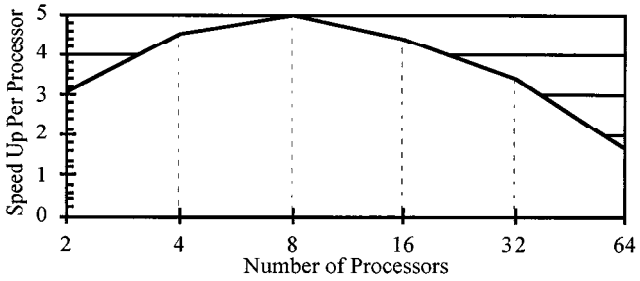

Figure 10: Speedup per processor

6 Conclusions

The final ASF version of the code contains 4 scalable parallel, 2 non-scalable parallel, and 3 serial programs. To see the relative execution times of these programs, total run times are graphed by module in Figure 12. Notice that the serial

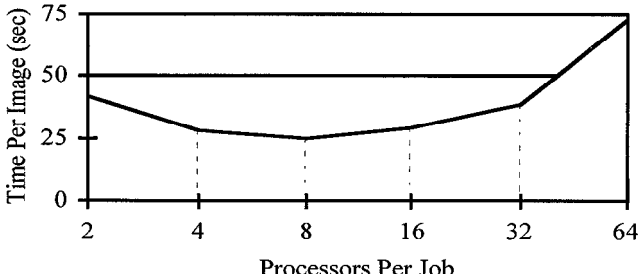

Figure 11: Time per image on 128 PE

programs (las_to_raw, editcorr, tiecoef) are basically invisible, while the times of the non-scalable programs are insignificant until 32 or 64 processors are used. This idea was amplified in Figure 10, where one can see the percentage of total time used by the scalable, non-scalable, and serial portions of the code.

Master-slave paradigm variations dominate the parallel implementation choices for this project. This was done to provide short term scalability for the programs by eliminating file contention and the resulting breakdown into serial execution. The latency hiding techniques of restricted and buffered I/O and block processing were tested and are included to some degree in all of the parallel programs. To reduce I/O overhead and the resulting processor wait time, programs use block reads and always pre-read data. Program output is buffered in a similar manner.

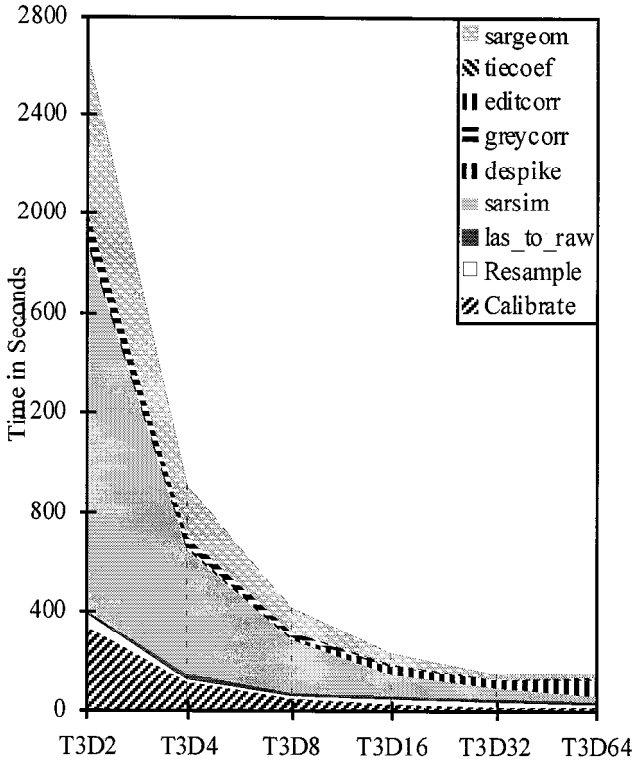

Figure 12: Terrain correction times by module on 2 to 64 processors

For the master-slave programs, interleaved receives and sends are used to reduce worker wait times. Restricting I/O to a single processor improves short term scalability by removing I/O contention. These techniques along with distribution of proper sized work blocks for a target number of processors, reduce worker wait time almost to zero and give over 90% efficiency with 32 processors.

In addition to master-slave processing, two work sharing programs are used. These programs, resample and despike, have small processing kernels but large I/O requirements. Thus, a scalable implementation was not possible. Rather, the parallel implementation that gave the best run time for 8 processors was chosen in order to further strengthen overall performance for that number of PEs.

The scalability performance of the parallel programs is given in Figure 13. The efficiency, which is speedup per worker, shows the ability of the code to use more workers at the same rate as the base case (1 worker, 2 PEs). The loss of efficiency for 16 or more processors results from the increase of non-scalable and serial portions of the total run time when using a large number of processors.

References

[1] Curlander, J. C., and R. N. McDonough, Synthetic Aperture Radar, Systems and Signal Processing. New York : John Wiley and Sons, 1991.

[2] Olmsted, Coert, *Alaska SAR Facility Scientific SAR User's Guide*, ASF-SD-003, July 1993.

[3] Wivell, Charles E., Coert Olmsted, D. Steinwand, and C. Taylor, An Earth Remote Sensing Satellite-1 Synthetic Aperture Radar Mosaic of the Tanana River Basin in Alaska, Photogrammetric Engineering & Remote Sensing, v59, p527-528, April 1993.

[4] Bicknell, T., *User's Guide to Products*, JPL D-9362, January 1992.

[5] CRAY Research Inc., *PVM & HENCE Programmers Manual*, SR-2501, May 1993.

[6] Ailts, B., D. Akkerman, B. Quirk, and D. Steinwand, LAS 5.0 - An image processing system for research and production environments, 1990 ACSM-ASPRS Annual Convention, Technical Papers, Denver, Colorado, Vol. 4, pp 1-12, 1990.

[7] CRAY Research Inc., *UNICOS Performance Utilities Reference Manual*, SR-2040 7.0, 1992.

[8] Dongarra, Jack, *Performance of Various Computers Using Standard Linear Equations Software*, University of Tennessee Computer Science Department Report CS-89-85, June, 1992.

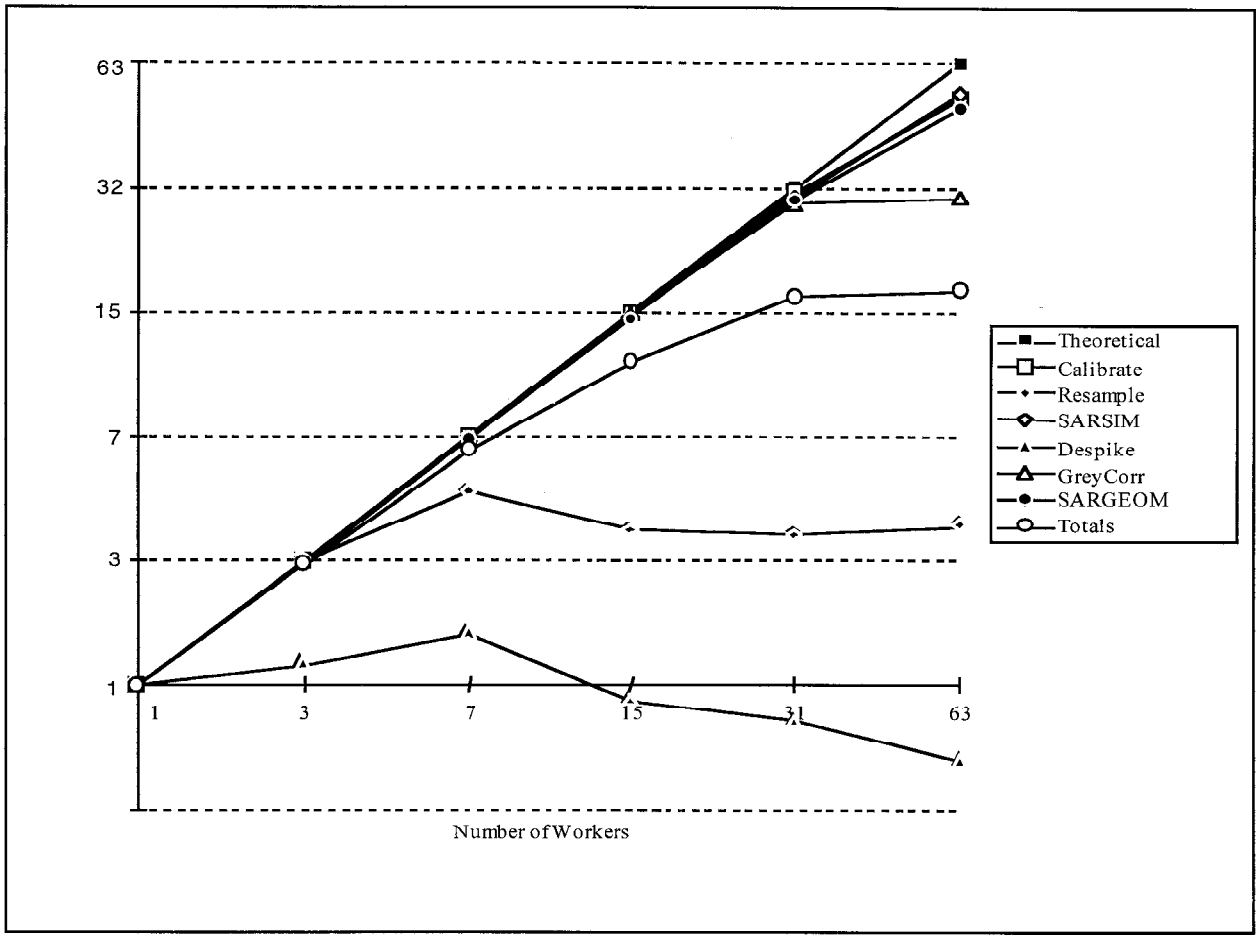

Figure 13: Speedup for parallel programs