Parallel Image Processing Applied to Radar Shape-from-Shading

A. Goller, M. Gelautz, and F. Leberl

Abstract

Various widely used radar image processing algorithms require considerable computing resources but can take advantage of a parallel implementation. We focus on the shape-from-shading (SfS) algorithm in its application to radar images. A given serial version of the SfS algorithm was parallelized and improved to handle large images. We experimented with parallelization techniques such as data decomposition, the manager/worker method, and dynamic load balancing with double buffering. The parallel version of SfS was ported to two supercomputers: Meiko's CS-2HA and Intel's PARAGON XP/S-A4 distributed memory machines, and to a cluster of workstations (CoW) made up of Silicon Graphics' Indies. Important results concerning the performance of the parallel SfS implementation with those architectures are presented and compared to each other, showing that 14 processors can speed up SfS by up to 13 times over the use of a single processor.

Introduction

Raw radar data are subject to *radar signal processing* (Curlander and McDonough, 1991) and will result in conventional image pixel arrays. Due to the large quantity and high rate of raw radar signals, it is customary to configure parallel processing systems for radar signal processing. *Radar image processing* then is applied to improve these images, or to extract information about the imaged surface, such as topographic shape, surface roughness, types of materials on the surface, or changes which might occur in the surface over time. Such information extraction often requires the use of multiple images and of elaborate algorithms.

While radar *signal* processing has traditionally been implemented on special purpose parallel processing hardware, radar *image* processing has scarcely been a topic of parallel processing research. An exception is early work with radar images from the Space Shuttle (Ramapriyan *et al.*, 1986) and recent work in noise despeckling (Addison *et al.*, 1996). However, parallel radar image processing is becoming a strategy of increasing importance and is motivated not only by complex algorithms, but also by the need to cope with extraordinary quantities of image data, and by increased availability of affordable open parallel computing platforms. This need is being demonstrated with the radar image coverage of planet Venus, which was obtained by NASA in its Magellan mission from 1990 through 1992 (Leberl *et al.*, 1992). Processing of the raw data signals acquired during the mission resulted in 400 Gbytes of image pixels.

Our interest in parallel radar image processing derives from the desire to process these images in order to obtain a detailed topographic surface model or digital elevation model (DEM), and to use this model for precise terrain corrections of the images (orthorectification, geocoding). NASA created a so-called Magellan Stereo Toolkit as a collection of sequential algorithms and functions that an individual data user could employ to process small subsets of the Magellan images (Curlander and Maurice, 1992). We briefly review four of the most frequently used algorithms.

- *Image Matching* establishes a set of corresponding points in two overlapping images in order to measure stereo parallaxes, mosaic images, or compare multiple images of a given terrain. Various appoaches exist, which for radar images are mainly based on correlation of pixel arrays, yet must cope with the radiometric differences due to illumination differences. Therefore, some algorithms use additional information derived from edge filters and local image statistics (Gelautz *et al.*, 1996). Some of the algorithms are quite time-consuming and would therefore benefit from a parallel implementation. However, the structural complexity of available serial implementations is high so that we decided that these algorithms should first be theoretically studied before parallel code gets written.
- *Resampling and Gridding* is used to resize DEMs and geocode images. The implemented resampling and interpolation algorithms are not very time consuming compared to other radar image processing algorithms. Thus, we are not concerned with parallelizing this code unless new time consuming algorithms or real-time needs exist.
- *Shape-from-Shading (SfS)* is based on the idea that the variation of brightness, or *shading*, is a result of the terrain shape responding to the illumination by the radar sensor. A DEM becomes locally refined by applying SfS. We have chosen this algorithm due to its time complexity and its algorithmic structure.
- *Visualization and Perspective Rendering* employs the 3D DEM, adds the corresponding 2D terrain-corrected images, and produces a perspective view of the combined data set. Ideally, this visualization provides the illusion that the human viewer is located on the terrain surface or experiences a fly-over in a fictitious aircraft. It will be useful to perform the computation in a parallel manner representing a classical computer graphics application. Clearly, this makes "rendering" a parallelization topic, which will be covered at a later time.

Shape-from-Shading

Principle

Normally, in radar image SfS, the assumption is made that the amount of light reflected by a particular part of the terrain surface is only a function of its orientation and its reflecting properties σ_0: i.e.,

$$I_{\text{pixel}} = R(\theta, \sigma_0(\theta)). \quad (1)$$

Photogrammetric Engineering & Remote Sensing,
Vol. 65, No. 3, March 1999, pp. 259–267.

0099-1112/99/6503–259$3.00/0

Institute for Computer Graphics and Vision, Technical University Graz, Münzgrabenstrasse 11, A-8010 Graz, Austria (goller@icg.tu-graz.ac.at).

If we know the pixel's brightness, I_{pixel}, and its reflective behavior, σ_0, then we should be able to compute the slope θ of the surface patch with respect to the radar's antenna position. Slope values θ_{ij} in each pixel ij need to be integrated into a continuous terrain surface in such manner that they are consistent with the observed slant ranges as well as with the gray values $I_{ij}(1), I_{ij}(2), \ldots, I_{ij}(n)$ in n input images (1), (2), ..., (n) (see Figure 1).

Horn and Brooks (1989) collected ideas about SfS in general as they existed in 1989. Whereas some early solutions to the SfS problem relied on solving the resulting differential equations directly, most recent approaches are based on the formulation of SfS as a minimization problem which is mathematically treated by calculus of variations techniques.

Generally, the reconstruction of topography from image gray values faces the following two problems: (a) The reflected energy is not only a function of the imaging geometry (e.g., local incidence angle), but is also influenced by the reflectance properties. In the particular case of terrain reconstruction in planetary sciences, the reflectivity of the surface materials is normally not known. (b) Even if the reflectance behavior is known, SfS still constitutes mathematically an underdetermined problem: A particular image gray value may have been generated by a variety of surface orientations. In the example of SAR images, a given pixel intensity imposes only a cone constraint on the corresponding local surface facet, with the axis being the radar look direction and the half-angle being the local incidence angle. In some applications, insufficient knowledge of the illumination direction poses further problems, which, hoewever, are not of concern when dealing with radar images.

Problem (a) is often circumvented by assuming the albedo to be constant in a first approximation. Another technique, which is adopted in the algorithm we employ, is the use of multiple images acquired with different look angles. The uniqueness problem (b) is generally tackled by adding either additional constraints, or *a priori* knowledge obtained from other sources. A frequently used constraint is the requirement of integrability, as defined by

$$Z_{XY}(X, Y) = Z_{YX}(X, Y) \tag{2}$$

with $Z(X, Y)$ being the surface height above the (X, Y) plane. In other words, the second-order partial derivatives are independent of the order of differentiation. If the integrability constraint is fulfilled, the surface height Z obtained by integration over surface slopes is independent of the path of integration. A second constraint is the requirement of smoothness, which regulates the amount of allowable oscillations in the reconstructed terrain surface.

A notable approach to the SfS problem which does not use the smoothness nor the integrability constraint was presented by Wei and Hirzinger (1994). Their idea is to use a multilayer neural network to represent the terrain surface analytically. In this formulation, the problem is converted into the task of training the network so that a given cost function is minimized with respect to the network weights. The obtained surface Z is solved directly, and is therefore automatically smooth and integrable, which avoids the problems arising in most other solutions due to the smoothness and integrability constraint.

The application of SfS to radar images, also denoted as radarclinometry, was pioneered by Wildey (1986) in the context of recovering the shape of the surfaces of other planets. In this work, uniqueness was enforced by assuming the terrain surface to be locally cylindrical. Further research on SAR imagery was carried out by Kirk (1987), Frankot and Chellappa (1989), Guindon (1990), and Thomas *et al.* (1991).

Kirk (1987) used finite elements instead of the variational approach, and points out the computational efficiency of his method. Guindon (1990) proposes the integration of individual SAR range lines into terrain elevation profiles independently of each other. This one-dimensional approach is motivated by the observation that SAR image gray values are mainly indicative of the range component of terrain slope, rather than the full 3D surface orientation.

Frankot and Chellappa (1989) presented a new solution to enforce strict integrability in an iterative SfS algorithm. Their idea is to project the (generally) non-integrable surface estimates obtained in each iteration step onto a subspace which contains only integrable solutions. This leads to "nearest" integrable surface slopes which are then input to the next iteration step. The advantage of this algorithm over other approaches which incorporate integrability by use of a penalty function is the enforcement of *strict* integrability,

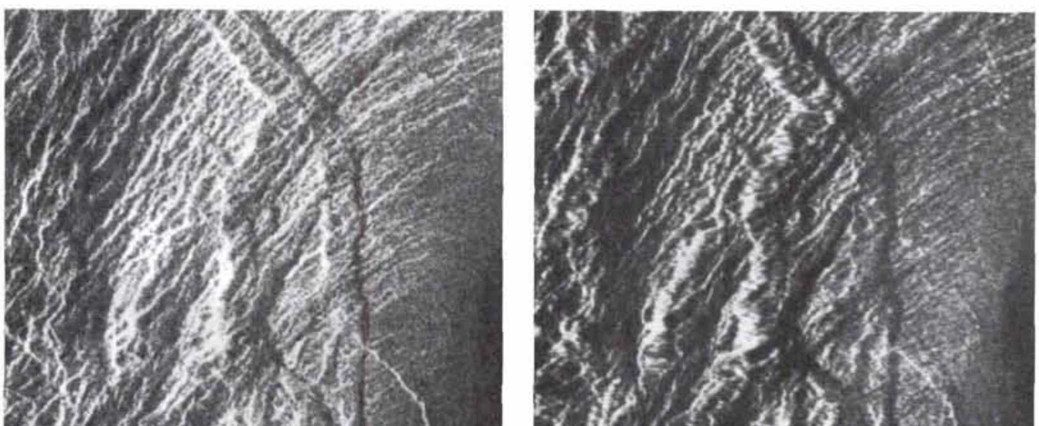

Figure 1. Images from the planet Venus, at 2° South, 73° East, geocoded using a coarse DEM obtained from stereoscopy. The area is about 40 km by 40 km. Look angles off nadir are 42° (left image) and 23° (right image). Pixels are about 75 m by 75 m in size.

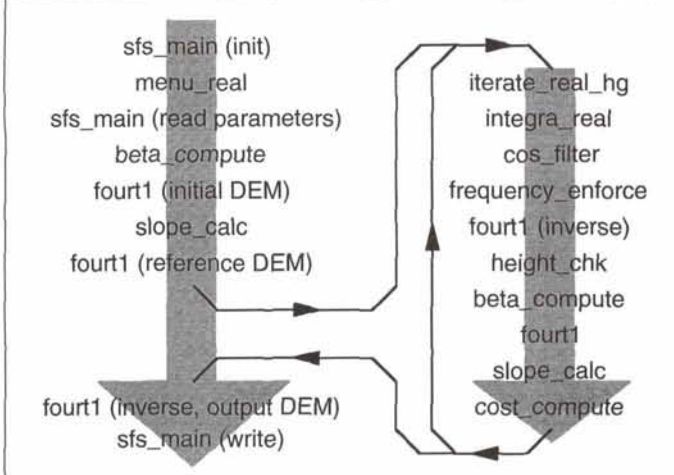

Figure 2. Description of shape-from-shading in terms of called subroutines.

whereas the penalty term pulls the solution only "close" to integrability.

Chosen Implementation

For our parallelization experiments, we have chosen the algorithm presented by Thomas *et al.* (1991), which is an extension of the work by Frankot and Chellappa (1989). Due to the use of multiple images, the simplified assumption of constant reflectance properties is no longer necessary. Furthermore, this technique has proven to be more robust to noise. Although the algorithm has been generally formulated for a set of n images, we have currently considered only cases with $n = 2$. This is consistent with the idea of using two overlapping images for a stereoscopic surface measurement, which is then followed by SfS to "refine" the solution.

In this algorithm, the calculus of variations problem derives from minimizing the following cost function (superscripts refer to image number, assuming two images):

$$\varepsilon = \iint (I^{(1)}(X,Y) - R^{(1)}(Z_X,Z_Y))^2 + (I^{(2)}(X,Y) - R^{(2)}(Z_X,Z_Y))^2 + \lambda(Z_{XX}^2 + 2Z_{XY} + Z_{YY}^2)\, dXdY \qquad (3)$$

where I is the actual image gray value, R is the predicted image gray value, Z is the terrain height, Z_X is the slope in the X (range) direction, Z_Y is the slope in the Y (azimuth) direction, Z_{XX} is the second-order partial derivative in the X direction, Z_{XY} is the second-order partial derivative with respect to X and Y, Z_{YY} is the second-order partial derivative in the Y direction, and λ is the regularization parameter.

The first term in Equation 3 is a measure of the difference between the pixel gray values I in one of the real SAR input images and the gray values R predicted by simulation using the current estimated terrain model. The second term refers to the other input image. The third term serves for regularization. It acts as a penalty function that limits the amount of terrain oscillations. The solution to the minimization problem is obtained iteratively (see Figure 2, cost_compute). At each iteration step, the resulting estimates of the terrain slopes are integrated to calculate heights (integra_real). At this step, integrability is enforced according to the method by Frankot and Chellappa (1989). The original solution for the surface slopes is projected onto a subspace of surfaces which can be represented by a set of Fourier basis functions, and thus fulfills automatically the integrability constraint. This frequency domain formulation of the problem leads to the massive use of Fast Fourier Transforms (FFTs) in the code (see Figure 2, fourt1), which we will discuss later in more detail. The spectral domain representation also facilitates the incorporation of low frequency information obtained from other sources, such as, e.g., stereoscopic analysis. The algorithm itself is fairly complex, but extensively described in the literature (Leberl, 1989; Thomas *et al.*, 1991). We therefore abstain from repeating its full description here.

The stereo process carried out in a preparatory step outputs two geocoded images and a preliminary DEM with elevations at each surface point XY. These surface elevations are combined with ephemeris data and assumptions about $\sigma_0(\theta)$ (e.g., the Hagfors (1964) reflection model) to serve as input to the SfS computation, as illustrated in Figure 3. Experience has shown that SfS should be applied to refine shape information obtained from other techniques. These might be altimetry, stereoscopy, or inaccurate DEMs from previous satellite missions. However, SfS should not be the sole source of shape information, because it produces large ambiguities in the surface shape, particularly at low frequencies (Leberl *et al.*, 1992). As already mentioned before, the spectral representation of the DEM lends itself well to merge low frequency stereo or altimetry elevations with high-frequency SfS elevations (see Figure 2, cos_filter and frequency_enforce).

Software Engineering Issues

It is important to analyze the algorithm and especially the data flow carefully, because this point shows if and how efficient the algorithm can be parallelized. As illustrated in Figure 2, several subroutines are called within the iteration loop. Logically, the loop begins at beta_compute to derive the incidence angles, which are then passed to slope_calc to calculate the slopes. To process the steps, Frankot and Chellappa (1989) proposed within the routines iterate_real_hg and integra_real, the Fourier transform (fourt1) has to be called in advance. Then, frequency domain constraints are applied for low (frequency_enforce) and high frequencies (cos_filter). After the inverse FFT, the refined DEM is compared to the reference DEM in height_chk and the cost function (Thomas *et al.*, 1991) is calculated in cost_compute as a means for iteration control.

Each call to the subroutines in this loop results in an additional refinement step, requiring iteration R_k to be finished prior to the start of iteration R_{k+1}. All data items used are either read-only as the initial DEM and images, or they are rewritten in each iteration, meaning that there is nothing during iteration R_k that may be calculated in advance. Thus,

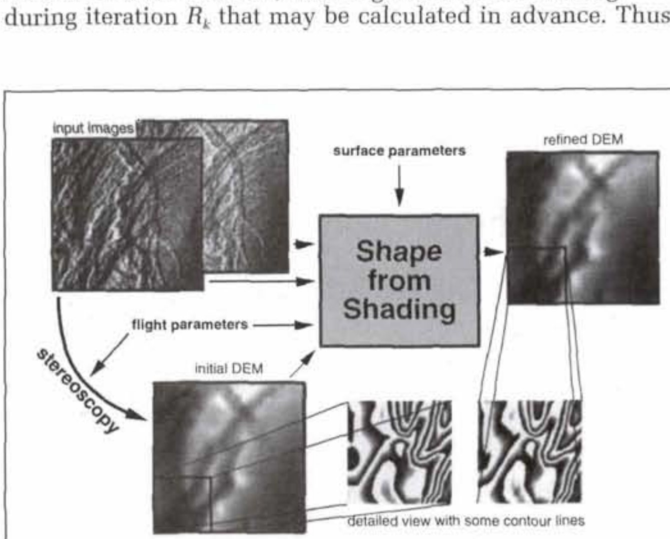

Figure 3. Shape-from-shading data flow.

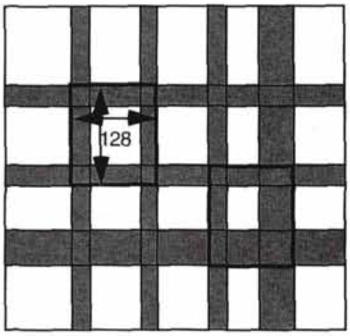

Figure 4. Data decomposition. The input data set is divided into equally large, overlapping patches which then are refined independently of each other.

trying to parallelize this loop according to the program decomposition paradigm (e.g., Zomaya *et al.*, 1996) leads to a dead end.

Nevertheless, all subroutines calculate one or several values for each pixel. For simplicity, let us assume that all images are square having n lines, each containing n pixels. If only one or a few pixels in the near neighborhood are necessary to compute the new one, the time complexity is $O(n^2)$, and one out of several processors may compute just a small piece of the whole data set.

We found that the near-neighborhood constraint is true for all subroutines except the FFT which has a complexity of $O(n^2 * \log(n))$ (Gonzalez and Woods, 1992). Thus, the overall complexity calculates to $O(i * n^2 * \log(n))$ where i represents the number of iterations. This estimate is mathematically correct, but says little about real-time behavior. Consequently, not using $O()$-notation leads to a better estimate for computing time T where the parameter c_1 is considerably larger than c_2: i.e.,

$$T \sim i * (c_1 * n^2 + c_2 * n^2 * \log(n)),\ c_1 \gg c_2. \quad (4)$$

The term $c_1 * n^2$ represents the time needed by all $O(n^2)$-routines during one iteration. The second term represents the time needed by the FFT, which is about 10 percent of the time described by the first term if $n \approx 1000$, which reflects a common image size. Time increases linearly by the number of iterations as can be seen from the leading factor i.

Handling Large Data Sets

While the FFT works well for small images, the time needed by Fourier and Inverse Fourier Transforms rises according to their time complexity of $O(n^2 * \log(n))$. Calculating the FFT for an entire image would last too long and is not necessary because SfS just refines the slope and terrain locally.

Data Decomposition is the method being applied to divide the input images and the initial DEM into smaller parts. Figure 4 shows the partition scheme. The subimages or patches, each 128 by 128 pixels in size, overlap one another. This is necessary due to erroneous effects at the edges of the patches.

Data decomposition of course influences the algorithmic behavior. On the one hand, the algorithm becomes very suitable for parallelization. On the other hand, using just local FFTs has two further impacts. First, execution time is reduced to $O(n^2)$, because the FFT is applied to equally sized, small parts, regardless of the total problem size. Second, the lower frequencies, which are generated by the FFT in the spectral domain, must be replaced by the low frequencies of the initial reference DEM. This guarantees that the small terrain patches which can then be calculated independently fit together after the refinement operation.

Putting the Patches Together is nevertheless difficult due to SfS's behavior at the seams. The low-frequency enforcement guarantees that the patches do not differ with respect to medium height and average slope. However, local disparities and oscillations occurring at the seams must be removed by an extra procedure, called "feather." To obtain satisfying results in conjunction with an acceptable amount of overhead calculation, the overlapping region was found to be about one quarter of the patch size. Due to the fixed size of all patches, the overlapping region is larger at the rightmost and bottom patches (see Figure 4).

Within the overlapping region, about ten pixels are cut off and replaced by the neighboring patch, as illustrated in Figure 5. This is necessary because SfS tends to produce high-frequency oscillations in the range direction which are caused by the many operations in the frequency domain which are susceptible to typical "ringing" effects. In order to provide a smooth transition between the patches, interpolation is done in the middle part of the overlapping region, where the terrain is linearly interpolated from one patch to the next patch. However, this interpolation is not always satisfying and will be subject to further research.

Parallelization Approaches

Because the data set has been split into smaller parts, a means for distributing these parts across all computing nodes must be found. Additionally, all nodes should be equally loaded. To reach these goals, we rely on well established paradigms. However, parallelization is meaningful only if the code on a single node is optimized as well.

Single CPU Optimization therefore is necessary to later obtain a good parallel code. It is also much easier to optimize the code before parallelization. Modern computing nodes are equipped with RISC processors, most of them with a super scalar architecture. To facilitate all available integer and floating-point units (FPUs), as well as to advance the usage of CPU registers and cache memory, a variety of compiler switches may be set. Sometimes, the source code must be slightly rewritten to enable the compiler's features. In order to help the compilers, code changes such as loop splitting, loop reversal, and loop blocking as well as some scalar optimizations were encoded by hand.

The Manager/Worker Method – a centralized approach –

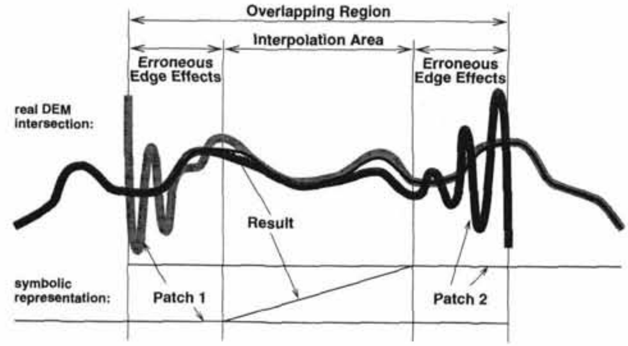

Figure 5. Interpolation strategy to merge adjacent DEM patches by means of a gradual transition weighing one patch less as one moves into its neighbor.

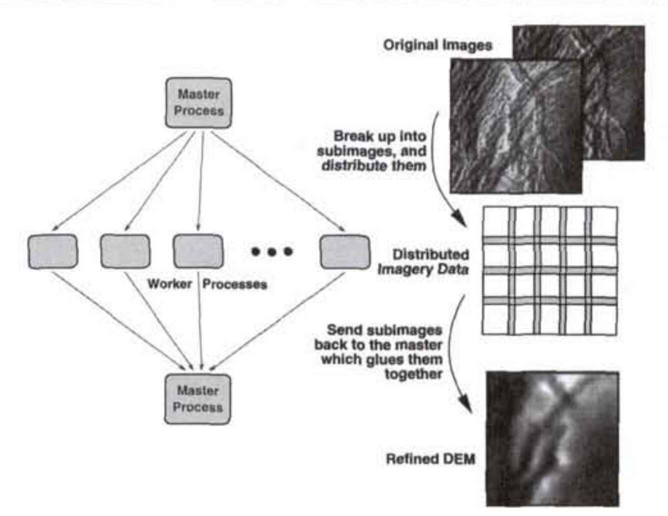

Figure 6. Manager/Worker method (left) combined with data decomposition (right).

was taken to distribute the patches to all available processors. One manager process does the I/O and controls all other processors, the workers. This scheme is illustrated in the left part of Figure 6. The manager reads the whole data set, partitions it, and sends the subimages to the workers. The workers themselves perform the real refinement calculations and then send the data back. The manager collects the results, postprocesses, and stores the refined and fitting patches back to disk. Because the manager performs no "real" computation, it can control many workers concurrently. However, the manager remains the bottleneck in both, file I/O and communication to the workers.

Through our experiments, we found some advantages in the Manager/Worker method. The simple communication structure and the well-defined tasks each processor has to perform led to an easy implementation. The work of porting the code onto other platforms was small as well, because the manager easily can be tested with dummy workers that only send back the received DEM patch.

Load Balancing appears to be a problem because the subimages are rather large, and we have to cope with problems when applying "coarse grain size" parallelization (Fox *et al.*, 1988; Wenzel, 1991). Static load balancing, which creates a schedule prior to the distribution of the subimages, cannot be used because the tasks the workers are charged to perform are slightly unequal in time, and not all CoW (Cluster of Workstations) workers are equally powerful.

In contrast, we used a simple paradigm for dynamic load balancing. The manager decides which subimages to send next while the workers perform computation. An easy and sufficient method is to take the patches in their natural order. When a worker sends back its result, this can be seen as a request for new work. Then, the manager sends the next available and not yet calculated subimage to this worker. At the beginning, or if some workers request new tasks nearly at the same time, they are served in a "round-robin" fashion. We found that the manager/worker paradigm as a basic communication structure is well suited to implement dynamic load balancing on top of this method.

However, this simple way of dynamic load balancing suffers at three points. The tasks are not sorted with respect to the time they are expected to be needed. Thus, it is possible that one of the last tasks submitted is one of the largest ones. This leads to the second problem. All workers must wait for completion of the preceding task. The remaining work cannot be redistributed. These two problems are not really serious if there are many more subimages than available workers and if the amount of work per task is about the same. While the first restriction can be met only when processing large data sets, the second one is fulfilled implicitly because all patches are of the same size and the computations for each patch do not differ significantly from one another.

Third, if many workers request tasks concurrently, the manager is overloaded for a moment. In this case, the workers are again served in a round-robin fashion, and some workers are forced to wait until the manager has time for them. It happens routinely that in this time other workers request new tasks, too. Thus, the workers are virtually synchronized. This problem also occurs at the very beginning. Assuming tasks with an equal size, this problem cannot be solved just in time, causing the manager to be overloaded periodically. These circumstances sometimes lead to dramatic losses in efficiency, especially if processors are interconnected via a bus system, or if there are other restrictions limiting the number of concurrent communications.

To solve this problem, we extended the dynamic load balancing with double buffering. To explain this mechanism, in Figure 7 a manager is shown with only two workers for simplicity. Initially, every worker gets one piece of work. Instead of requesting a new task after the first task was fin-

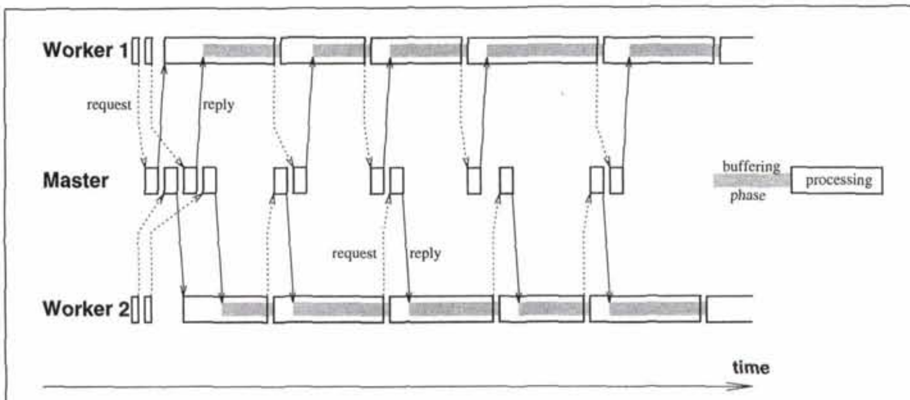

Figure 7. Communications behavior of the implemented dynamic load balancing technique introducing double buffering. For simplicity, only one manager and two workers are drawn.

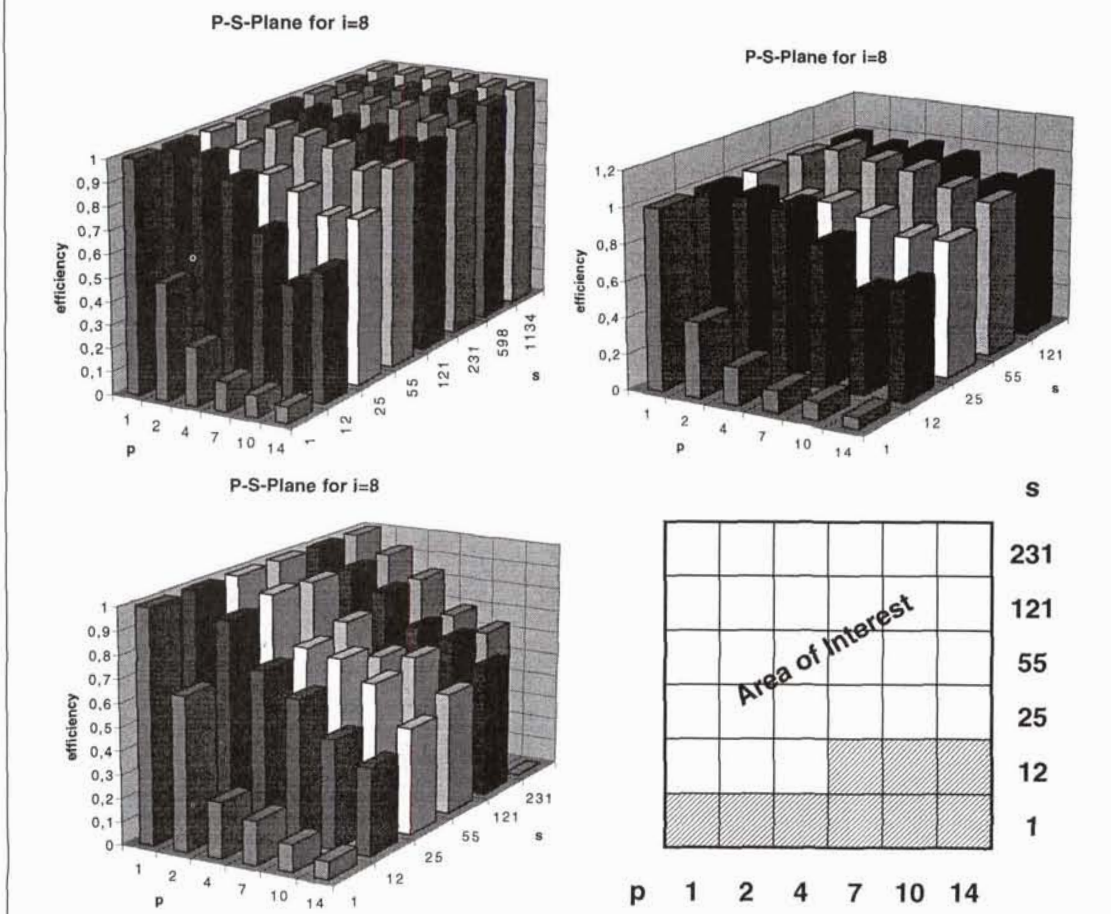

Figure 8. Efficiency with a certain number of processors (p) versus problem size (s). Paragon (top left) and Meiko (top right) perform well in all areas of interest (bottom right, shaded area has $p \leq s$). The efficiency of the COW (bottom left) decreases when several processors are used (i represents the number of iterations).

ished, the worker immediately requests a second task. This overloads the manager heavily at first, but the workers do not care because they already have tasks to compute. The manager now has plenty of time to submit all requested tasks while the workers are calculating the first problem. Thus, all workers can start their next task immediately after they have finished their preceding task, and they can request a new task in advance. As can easily be concluded, this method theoretically reduces the idle time of workers down to zero.

Hardware Description

For this project, we had access to three different computing platforms, located in Vienna and Graz. All platforms are equipped with a set of general purpose microprocessors. Thus, they all can be classified as MIMD computers (Flynn, 1972). All nodes run an extended UNIX operating system. Communication is according to the message passing paradigm, because the memory is distributed across all nodes.

Meiko CS2-HA – Computing Surface 2-High Availability (CS2-HA) from Meiko is located at a European Union facility, the European Centre for Parallel Computing at Vienna (VCPC). This supercomputer is equipped with 128 compute nodes, each performing 100 MFLOPs/second. Disk capacity is about 40 Gbytes, and each node is equipped with 64 Mbytes of RAM. The network is scalable and thus can be easily extended by adding 8 * 8 crosspoint switches (ELITE chips) (http://www.vcpc.univie.ac.at/meiko/overview/MeikoOverview.html).

Paragon XP/S-A4 is the second computer to which we had access. This supercomputer from Intel contains 56 compute nodes, each equipped with 16 Mbytes of RAM, and working at 75 MFLOPs/second. The Paragon is equipped with 15 Gbytes of disk space and the processors are interconnected via a 2D mesh. The Paragon is located at the University Computing and Information Services Center (EDVZ) at the Technical University Graz.

SGI Indy-CoW is an Indy-workstation cluster from Silicon Graphics, Inc (SGI). Each Indy performs about 100 MFLOPs/second and is interconnected with a 10 Mbits/second Ethernet line. A Power Challenge computer managing 47 Gbytes of disc space is also connected to the Ethernet and serves as the master processor. However, this cluster is by no means a standalone facility. Although performance was measured mainly during weekends and over night, results depend on the usage of the various workstations at that time.

Performance Assessment

Figure 8 shows the efficiency obtained on each computing system, measured for various values of p and s. p represents the number of processors involved, and s is the number of patches into which the whole data set was decomposed.

Utilizing more than half of all processors is commonly agreed to be an acceptable efficiency for parallel applications. In all three diagrams, the efficiency never drops below this 50 percent level in the area of interest (see Figure 8, bot-

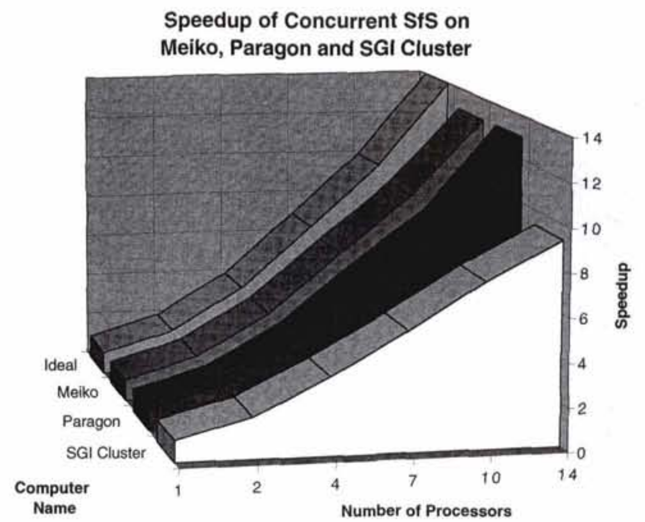

Figure 9. Performance comparison of all platforms relative to an ideal parallelization.

tom right). This area excludes parts where $s < p$, because then the number of patches to distribute is smaller than the available number of processors. Consequently, some processors must be idle. Areas where $s \approx p$ are also out of interest because load balancing cannot work properly under such circumstances. Due to the coarse grain parallelization applied, we have to exclude the first row ($s = 1$) and half of the second row ($s = 12$, $p \geq 7$).

The upper left diagram in Figure 8 shows the efficiencies measured for the Paragon supercomputer. Efficiency is very close to 1 whenever s is large enough to keep the load balancing mechanism working. This holds for the region having $s > 4 \cdot p$. Paragon can thus be said to be the best and most stable system examined.

Next best is the Meiko supercomputer (upper right diagram). Execution times varied; repeatedly running SfS with the same input data led to significantly different timings due to the operating system's way of invoking the parallel processes. We therefore took the average of five runs to obtain the one-processor timings. However, we measured the time just once for all other parameter settings, resulting in a rather spiky diagram, even showing efficiencies larger than 1.

The efficiency of the CoW already decreases at a low number of processors, which can be seen in the lower left diagram. Mostly independent of the problem size s, the efficiency decreases linearly with the number of processors p, indicating a communication bottleneck. It is not only the poor communications bandwith of Ethernet but also the bus topology that allows only one process to send data any time. In particular, the latter fact causes many bus collisions when several workers try to request new work nearly at the same time.

To obtain a better impression of the achieved performance gain, all three platforms were compared to an ideal parallelization, which only can be acchieved theoretically. A 1024- by 1024-pixel DEM, or 121 patches, was therefore refined on up to 14 processors. Execution time on one processor is about 112 seconds on a Meiko, 128.5 seconds on a Paragon, and 500 seconds on an Indy workstation.

In Figure 9 the speedup is shown, pointing out again the very good utilization of the computing nodes on Meiko and Paragon. Supercomputers are likely to perform well in this area of image processing due to their high bandwith communication links, but also the CoW may be used as an inexpensive alternative.

SfS Results

Fast execution is valuable, but carefully comparing the results to those from the sequential version are mandatory to prove the correctness of the parallelized program. Because we had already modified the sequential algorithm to handle large data sets, correctness must be checked by comparing (a) the original and the modified sequential algorithms and (b) the sequential and parallel versions.

The outputs for comparison (b) are identical. However, because it was impossible to refine large images before this project, there were no data available for comparison (a). We therefore had to validate the output of the modified sequential program for large images by a human quality control process. It is important that the transition from one refined patch to its neighbors is smooth, and that oscillations at the rim of each patch are removed completely.

In Figure 10, we compared the input DEM as obtained from stereo processing and subsequent resampling, and the SfS-refined output DEM in a rather analytical way. Each immediate jump from white to black corresponds to a contour line. Contour lines are set every 250 m. To illustrate the terrain's behavior within these lines, fractions of the steps between the contour lines are shaded linearly as different gray values.

A better visual impression can be received looking at the DEMs from a perspective view point near the lower left corner, as shown in Figure 11. Triangulation artefacts and plain, artificially looking slopes are replaced by a more realistic terrain shape.

Conclusion and Outlook

An existing serial version of a time-consuming radar image shape-from-shading algorithm was first extended to handle large data sets using the principle of data decomposition, and then parallelized by employing a manager/worker algorithm with load balancing based on double-buffering. The parallel implementation was ported to a Meiko CS-2HA, an Intel Paragon XP/S-A4, and a cluster of SGI Indy workstations (CoW). Performance measurements on these platforms have shown that the Meiko and Paragon perform very well with an efficiency of better than 98 percent when using up to 16 processors, which was the maximum number of computing nodes available for our experiment. The efficiency of the CoW was found to decrease significantly when less than ten processors were used, reflecting mainly a communication bottleneck on the Ethernet. However, due to the good performance in smaller configurations, the CoW might be used as an inexpensive alternative to supercomputers if only a few computing nodes are available.

Based on the knowledge obtained from this implementation, we intend to proceed with the parallelization of other computationally intensive algorithms used in image processing. Work will be focused on radar image processing algorithms such as matching, gridding, and resampling in order to process Magellan's massive 400 Gbytes of image data and to obtain a DEM from these images. Prior to that step, signal processing code must be improved to compute more accurate images from raw radar echoes. Specifically, this elaborate task also has to be parallelized, exploiting insight and results from this ongoing work.

The experiences gained in this work also will be valuable for processing data from Earth-orbiting satellites equipped with either electro-optical or radar sensors. Concurrent algorithms will become mandatory for processing high resolution data from future satellites.

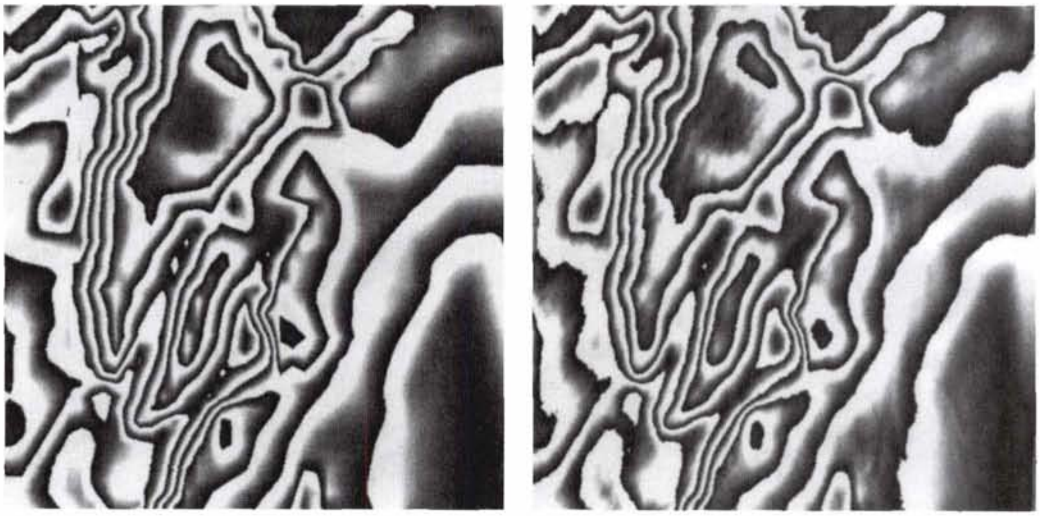

Figure 10. Contour plots of both the initial DEM (left) and the refined DEM (right) from the surface of Venus. One transition from black to white corresponds to 250 m. The original images are shown in Figure 1.

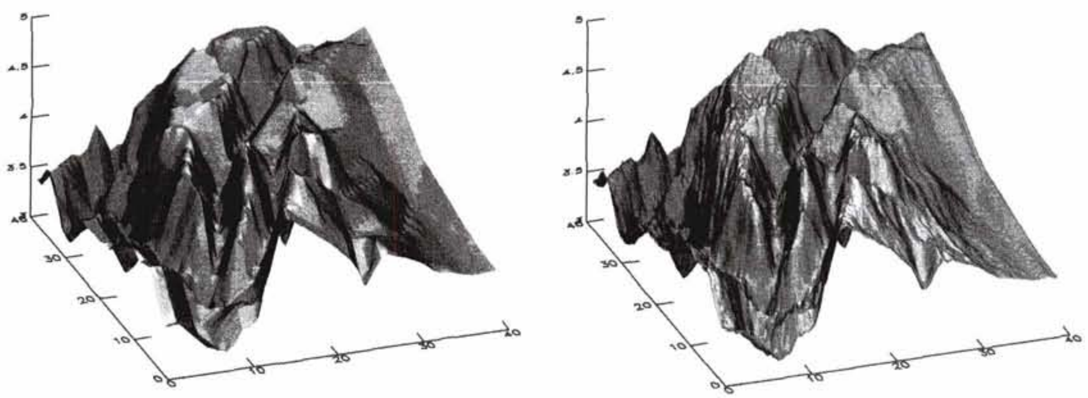

Figure 11. Perspective visualization of both the initial DEM (left) and the refined DEM (right) of Figure 10. All scales are in km.

Acknowledgments

This work was financed by the Austrian Research Funds, "Fonds zur Förderung der wissenschaftlichen Forschung (FWF)," within the Research Program, "Theory and Applications of Digital Image Processing and Pattern Recognition," Project 7001, Task1.4 "Mathematical and Algorithmic Tools for Digital Image Processing."

References

Addison, C., E. Appiani, R. Cook, M. Corvi, J. Harms, G. Howard, and B. Stephens, 1996. PULSAR: Parallel Noise Despeckling of SAR Images, *Proceedings, High-Performance Computing and Networking (HPCN96) Conference*, Lecture Notes in Computer Science 1067, Springer Verlag, pp. 177–182.

Curlander, J., and R. McDonough, 1991. *Synthetic Aperture Radar: Systems and Signal Processing*, John Wiley & Sons, 647 p.

Curlander, J., and K. Maurice, 1992. *Magellan Stereo Toolkit User Manual*, Vexcel Corporation, Boulder, Colorado.

Fox, G., M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, 1988. *Solving Problems on Concurrent Processors, Vol. 1, General Techniques and Regular Problems*, Prentice-Hall International, London, 592 p.

Flynn, M.J., 1972. Some Computer Organizations and Their Effectiveness, *IEEE Transactions on Computing*, C-21(9):948–960.

Frankot, R., and R. Chellappa, 1989. A Method for Enforcing Integrability in Shape-from-Shading Algorithms, *Shape-from-Shading* (B. Horn and M. Brooks, editors), MIT Press, Cambridge, pp. 89–122.

Gelautz, M., G. Jakob, G. Paar, S. Hensley, and F. Leberl, 1996. Automated Matching Experiments with Different Kinds of SAR Imagery, *Proc. of IGARSS 96*, Lincoln, Nebraska, pp. 31–33.

Gonzalez, R.C., and R.E. Woods, 1992. *Digital Image Processing*, Addison-Wesley, pp. 119–127.

Guindon, B., 1990. Development of a Shape-from-Shading Technique for the Extraction of Topographic Models from Individual Spaceborne SAR Image, *IEEE Trans. Geoscience and Remote Sensing*, 28(4):654–661.

Hagfors, T., 1964. Backscattering from an Undulating Surface with Applications to Radar Returns from the Moon, *Journal of Geophysical Research*, 69(1):3779–3784.

Horn, B., and M. Brooks, 1989. *Shape-from-Shading*, MIT Press, Cambridge, Massachusetts, 569 p.

Kirk, R., 1987. *A Fast Finite-Element Algorithm for Two-Dimensional Photoclinometry*, Thesis, California Institute of Technology, Pasadena, California.

Leberl, F., 1989. *Radargrammetric Image Processing*, Artech House, Norwood, Massachusetts.

Leberl, F.W., J.K. Thomas, and K.E. Maurice, 1992. Initial Results from the Magellan Stereo-Experiment, *Journal of Geophysical Research*, 97(E8):13675–13687.

Ramapriyan, H., J.P. Strong, Y. Hung, and C.W. Murray, Jr., 1986. Automated Matching of SIR-B Images for Elevation Mapping, *IEEE Trans. Geoscience and Remote Sensing*, GE–24(4):462–472.

Thomas, J., W. Kober, and F. Leberl, 1991. Multiple Image SAR Shape-from-Shading, *Photogrammetric Engineering & Remote Sensing*, 57(1):51–59.

Wei, G., and G. Hirzinger, 1994. Learning Shape-from-Shading by Neural Networks, *Proceedings, Mustererkennung 1994* (W.G. Kropatsch and H. Bischof, editors), Informatik Xpress, Technical University Vienna, pp. 135–144.

Wenzel, L., 1991. *Parallele Programmierkonzepte*, Franzis Verlag, München, Germany.

Wildey, R., 1986. Radarclinometry for the Venus Radar Mapper, *Photogrammetric Engineering & Remote Sensing*, 52(1):489–511.

Zomaya, A.Y. (editor), 1996. *Parallel and Distributed Computing Handbook*, McGraw-Hill, Series on Computer Engineering, Chap. 9.

Received 18 August 1997; accepted 03 March 1998; revised 01 May 1998)