# A parallel implementation of speckle image reconstruction

## Toward parallel speckle reconstruction for the Dutch Open Telescope

A. G. de Wijn

Astronomical Institute, P.O. box 80 000, 3508 TA Utrecht, The Netherlands
e-mail: A.G.deWijn@astro.uu.nl

**Abstract.** In this report, I present a parallelized speckle-reconstruction code for use with the Dutch Open Telescope. The code implements the speckle-masking algorithm. The theory of speckle masking is explained in some detail. We discuss the differences and similarities between the new implementation in C and the old implementation in IDL. I present measurements on the performance of both implementations. The parallelization model of the C implementation is discussed. The model and the measurements are compared qualitatively and quantitatively. Finally, I make some recommendations for a reconstruction farm for the Dutch Open Telescope.

## 1. Introduction

The general belief of stellar activity is that stellar coronae and chromospheres are heated magnetically, but it remains an open question how this comes about. Magnetism lies at the heart of most solar and heliospheric physics. The intricate structure of the solar field, the activity cycle and the influence of the field on the heliosphere represent major quests of astrophysics bearing directly on the human environment. Solar activity modulation affects satellite orbits, influences jet stream patterns, and contributes to the causes of ice ages.

In order to study solar magnetic fields, observations at high spatial resolution are required over periods of time in the order of hours. Magnetic structures in the photosphere have typical diameters of less than 100 km. Although a telescope with a primary mirror of about 50-cm diameter is theoretically able to resolve such features, observations with these resolution are very rare in practice.

The reason is that the Earth's atmosphere distorts the image. This well-known effect is due to local variations in temperature, which affect the index of refraction. This effect is commonly referred to as *seeing*. It can also be observed above the flame of a candle, or above a road on a sunny day. Figure 1 illustrates the effect. The intensity distribution of the object is distorted by the atmosphere. Rays that were initially parallel are affected differently, and arrive at the telescope via different paths.

The severity of the distortion is usually expressed in the so-called *Fried parameter*. The Fried parameter describes the effective diameter with respect to the resolution of the telescope, i.e., making the primary mirror larger than the Fried parameter does not allow one to see smaller structures. The major advantage of a larger mirror is that it collects more light, so that fainter objects become visible.

There are several ways to reduce the effect of seeing. One way is to reduce the amount of atmosphere between the sun and the telescope. Some of the first attempts at this involved telescopes mounted on balloons, which were flown to high altitudes. Although the results were not disappointing, the size of the telescope was severely limited by the weight limit imposed. Furthermore, the duration of a flight was relatively short. There are several new projects in development for balloon-mounted telescopes, claiming long-duration observations with large telescopes. Of course, the best way of reducing the effect of seeing is by leaving the Earth's atmosphere altogether, but building a space-based telescope is costly. Also, once such a telescope has been launched, servicing it is difficult at best, and very expensive.

Ground-based observatories are usually built on elevated terrain selected for low atmospheric turbulence and high seeing quality. Many telescopes are built on extinct volcanoes at altitudes over 2 km. For example, the Dutch Open Telescope (DOT), together with a large number of other telescopes, both solar and night-time, is built on the Canary Island La Palma at a height of 2350 m. Many more telescopes are situated on Tenerife at a similar altitude.

Solar telescopes suffer from the high heat flux of the sun. When a conventional night-time telescope would be used to observe the sun, the interior would quickly heat up and produce large amounts of turbulence inside the telescope itself, distorting the image to unacceptable levels. Classic solar telescopes therefore have most of the optics in a vacuum. The DOT pioneered an open design, allowing the strong trade winds on La Palma to continually flush the primary mirror. In this way the formation of internal

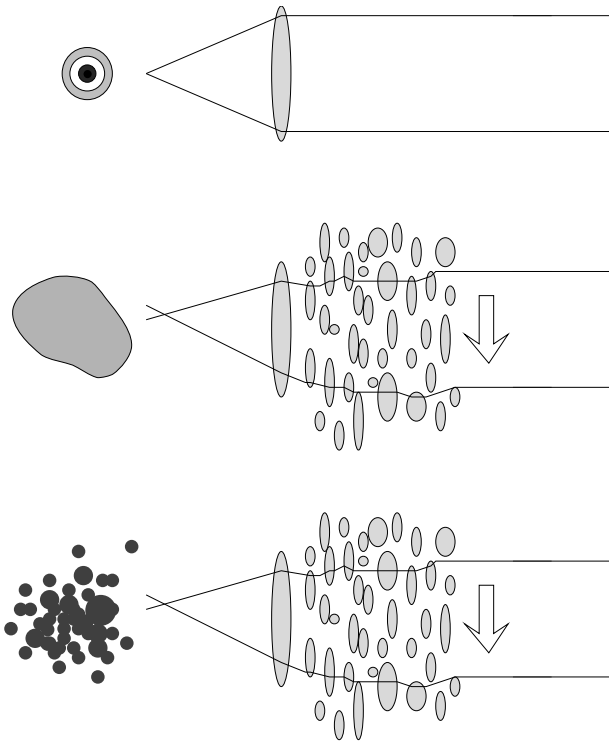

**Fig. 1.** The effects of seeing. In an ideal situation (left), a telescope records a point source as the diffraction pattern of the aperture. The atmosphere distorts the image. Bubbles of air with varying refractive index float through the field of view. In a long-exposure image (middle), the image of a point source is a blurry shape much larger than the diffraction limit of even small telescopes. Short exposures (right) show "speckles," the instantaneous result of the effect of the atmosphere on the image.

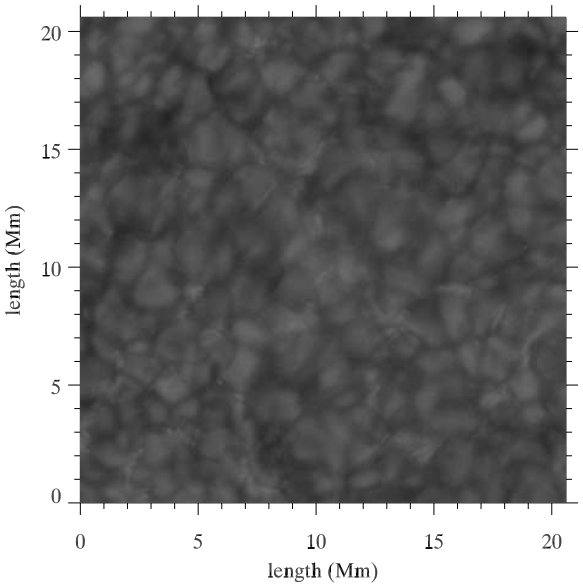


**Fig. 2.** The single best frame of a burst of 100 images.

turbulence is strongly reduced. This design is now being adopted for several new solar telescopes in development.

Also, there are techniques to reduce the effect of seeing. A well-known method is *adaptive optics*, the correction of the distorted image in real-time by adjustable optical components, e.g., a deformable mirror. Because this technology is relatively new, few telescopes have implemented it. There are several disadvantages to adaptive optics. The fully corrected image region cannot be larger than the area over which the atmosphere is coherent, which is usually in the order of a few arcseconds. This is solved with so-called multi-conjugate adaptive optics, but as yet such systems are not used in production environments.

It is also possible to correct an image after it has been recorded. Extra information on the object recorded is needed to remove the atmospheric distortions. Phase diverse reconstruction requires the recording of two simultaneous images of the object with a known phase difference. One camera is in focus, while another is slightly out of focus, with the distance between the two cameras known. Provided the seeing quality is not too bad, it is then possible to reconstruct the image of the original object in the focal plane. The Swedish Vacuum Solar Telescope and the German Vacuum Tower Telescope, among others, have used this method to produce high-quality images of the sun.

Speckle masking of an image is a stochastic process to remove seeing effects from observations. In order to reconstruct one image, one needs a large number of images, called a *burst*, recorded in a period of time significantly less than the time in which the recorded object changes. If the atmospheric conditions are not too bad, it is possible to find a "most likely original" image, with a resolution near or at the theoretic limit of the telescope. This form of image restoration has been used on the DOT since 1999.

It is widely believed that both phase diversity and speckle masking are valid methods of improving observation quality. Both methods, though different, produce similar results. Comparison with observations made during superseeing show very similar structures. The ultimate test will be when Solar-B has been launched and reconstructed earth observations can be compared with its results.

Speckle reconstruction is a computationally very intensive process. On the DOT, the reconstruction of a 90-minutes time series at 30-seconds cadence with bursts of 100 images currently takes about two months on six dual 600-MHz Intel Pentium III machines. The algorithm is now implemented in IDL. IDL is an interactive data-manipulation language. It is an excellent tool for what it was designed to do, and is widely used in astrophysics. Its performance is far from poor, but it is not well suited for situations where performance is paramount, where the implementation does not change rapidly, or where the same program is used to process large quantities of different data sets.

The speckle reconstruction as used on the DOT allows for parallelization in several ways. The most simple

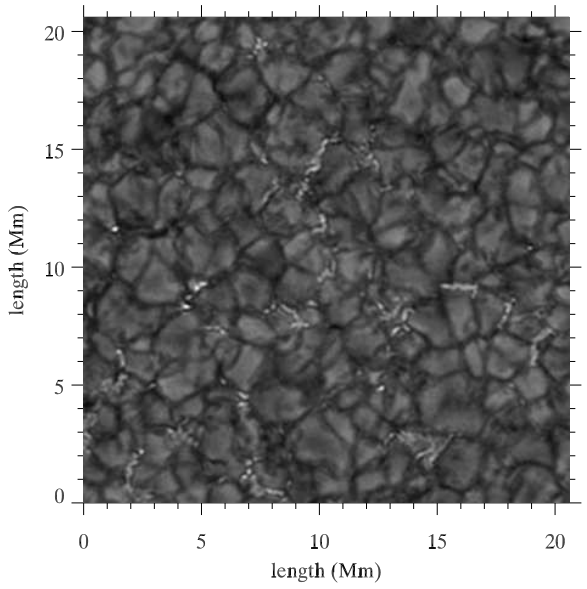


**Fig. 3.** The speckle-reconstructed image of the burst of which Fig. 2 is the single best frame. The improvement in quality is obvious.

is of course the processing of several bursts at a time. However, one might desire a quick preview of the area being observed. Since the reconstruction of one burst now takes in the order of a few hours, the current setup cannot provide such a preview. Furthermore, the hardware requirements are substantial, especially with respect to memory usage. If the memory usage can be spread over several machines, a sufficiently powerful system would be cheaper, while the processing speed would not be reduced significantly. Therefore, I have sought to parallelize the reconstruction of a single burst.

## 2. History

In 2001, to celebrate the 365-th anniversary of Utrecht University, the Utrecht University Linux Users Group (UULUG) built a temporary cluster supercomputer from existing university hardware. While scouting for possible users of the cluster, the speckle-reconstruction algorithm was identified to be an excellent application to run on such a cluster. The reconstruction loop itself requires very little input data, and produces even less output, while it requires a large amount of calculations.

Since the DOT team had no one able or willing to spend time on such a project, I took it onto myself to rewrite the IDL implementation in C and to parallelize it. Though not completely finished, the result of my efforts was run during the demonstration of the cluster on July 9, 2001.

There have been a large number of improvements since then. The implementation of the preprocessing and the speckle reconstruction is now finished. The implementation has been better optimized for use in a parallel environment and several improvements over the original IDL code have been made.

## 3. Speckle Reconstruction

Any implementation of speckle reconstruction will consist of at least two parts: the preprocessing and the reconstruction. Since the reconstruction can only work on an isoplanatic patch, i.e., a region of the image over which the atmosphere is coherent, in most cases a third part is needed to reconnect the reconstructed subimages the size of an isoplanatic patch.

### *3.1. Preprocessing*

During the preprocessing, the data is prepared for the reconstruction loop. This means that it is corrected for specs of dust on the optical components near the focal plane, such as the CCD itself, and dark current from the CCD. This is commonly referred to as flatfielding and darkfielding, respectively. Errors such as due to telescope guiding and atmospheric distortions causing large scale displacement must be removed. Also the seeing quality has to be computed. We now discuss this in more detail.

First, the data must be corrected for CCD and telescope errors. Thermal excitations, e.g., may increase photon counts on the CCD. To correct for these virtual photons, a burst is recorded while the CCD shutter is closed, so no photons from the object of interest reach the CCD. This is called a *darkfield* burst. The average image of this burst is computed and is subtracted from the data.

A speck of dust on the CCD for example will cause a localized decrease in the intensity whose position on the CCD will not move. The CCD itself may have different sensitivity at different positions. In order to correct this, every recording run, which may last for a few hours, must include a *flatfield* burst. A flatfield burst is very similar to a normal burst, except the telescope is intentionally out of focus. As a result, all features on the sun are blurred to such an extent that no structure can be detected. This means the CCD records an essentially flat image, but with all telescope errors present. Something like a speck of dust on the CCD will be clearly visible. The flatfield burst is summed to produce one flatfield image, and the data is divided by this image.

Next, residual telescope guiding errors, which are negligible for the DOT due to its excellent mechanical stability, and large scale motions caused by atmospheric distortions must be removed. This involves feature tracking to align the images in the burst. Once that is done, the images in the burst have to be split up into parts no larger than an isoplanatic patch. Since distortions on the size of an isoplanatic patch are not removed by the alignment of the whole field, these sets of subimages must also be co-aligned.

From these sets of subimages the seeing quality can be computed. When comparing each subimage in a stack to the corresponding average subimage, low frequencies in the spatial domain will be approximately the same,

while high frequencies will be much reduced in the average. When the seeing quality becomes better, more high frequencies will remain. Thus by comparing the ratio between the two power spectra, and using a model of the atmosphere, a number indicating the seeing quality, the *modified Fried parameter*, can be estimated. The modified Fried parameter is the Fried parameter divided by the telescope aperture. From this, the *speckle transfer function* (STF) can be computed. This function is also dependent on the telescope aperture. The calculation of the STF would require the numerical evaluation of a complex four-dimensional integral, and is not done during the preprocessing. STFs have been precomputed for several values of the modified Fried parameter, and these are interpolated at the computed value.

The noise in the measurements must also be computed. What is computed is the power spectrum of photon noise. This is done by subtracting the darkfield from the flatfield burst and dividing the result by the average flatfield image. The result should only contain noise, and no signal. This burst is split into subimages and the average power spectrum for each stack is computed. The result is called the *noise power*. This quantity can be different in different parts of the image, but not by much. Therefore, just like the average flat- and darkfield images, the noise power can be computed in advance, and has to be computed only once per observation run.

### 3.2. Speckle Masking

At this point all variables needed during the reconstruction loop are known, and all data required have been prepared. The reconstruction requires one stack of subimages, the noise power corresponding to that subimage, and the STF. There is no correlation between subimages, meaning every stack of subimages can be reconstructed separately.

The fundamental assumption in the speckle-masking method is that the *recorded* intensity distribution $f$ of an object is a convolution of the *real, time-independent* intensity distribution $f_0$ with a *point-spread function* $PSF$. That is,

$$f(\mathbf{x},t) = f_0(\mathbf{x}) \otimes PSF(\mathbf{x},t)\,, \tag{1}$$

where $\otimes$ denotes the convolution. This equation in the spatial domain has an equivalent in the Fourier domain,

$$F(\mathbf{q},t) = F_0(\mathbf{q})\,S(\mathbf{q},t)\,. \tag{2}$$

For a large number of observations (a burst) made at times $t_1, t_2, \ldots$, we have

$$\left\langle |F(\mathbf{q},t)|^2 \right\rangle = \left\langle |F_0(\mathbf{q})\,S(\mathbf{q},t)|^2 \right\rangle = |F_0(\mathbf{q})|^2 \left\langle |S(\mathbf{q},t)|^2 \right\rangle\,, \tag{3}$$

where $\langle \ldots \rangle$ has been used to indicate the average over time. Here $\langle |S(\mathbf{q},t)|^2 \rangle$ is known STF (cf. Sect. 3.1). Transformation back to the spatial domain will yield the autocorrelation function of $f_0(\mathbf{x})$, but not the original object. This is known as the Labeyrie method.

In practice, $f_0$ gradually changes with time. This imposes a restriction on the time between the first and the last observation: the time in which the object observed does not change noticeably, and $f_0$ is approximately time-invariant.

In order to be able to recreate the original image $f_0(\mathbf{x})$, it is not enough to know the absolute value of the intensity distribution in the Fourier domain $|F_0(\mathbf{q})|$. The phases $\phi(\mathbf{q})$ in

$$F_0(\mathbf{q}) = |F_0(\mathbf{q})|\,\mathrm{e}^{i\phi(\mathbf{q})}, \tag{4}$$

must also be computed. We will rewrite Eq. 2 in such a way that the left hand side is known from measurements, and the right hand side is made up of two factors, one of which is depends only on the object, and the another one is real. If this second factor can be computed from other known quantities such as the telescope aperture and the seeing quality, the original image can be reconstructed. It turns out to be useful to multiply the left hand side of Eq. 2 with $F(\mathbf{p},t)\,F^*(\mathbf{q}+\mathbf{p},t)$ and the right hand side with $F_0(\mathbf{p})\,F_0^*(\mathbf{q}+\mathbf{p})\,S(\mathbf{p},t)\,S^*(\mathbf{q}+\mathbf{p},t)$. From Eq. 2 it can easily be seen that these two expressions are equal. Again averaging over time, we then find

$$\langle F(\mathbf{p},t)F(\mathbf{q},t)F^*(\mathbf{q}+\mathbf{p},t)\rangle = F_0(\mathbf{p})F_0(\mathbf{q})F_0^*(\mathbf{q}+\mathbf{p}) \times \langle S(\mathbf{p},t)S(\mathbf{q},t)S^*(\mathbf{q}+\mathbf{p},t)\rangle\,. \tag{5}$$

The *speckle-masking bispectrum* is now defined as

$$F^3(\mathbf{q},\mathbf{p},t) = F(\mathbf{q},t)F(\mathbf{p},t)F^*(\mathbf{q}+\mathbf{p},t)\,. \tag{6}$$

Inserting this into Eq. 5, we find

$$\left\langle F^3(\mathbf{q},\mathbf{p},t)\right\rangle = F_0^3(\mathbf{q},\mathbf{p})\,\left\langle S^3(\mathbf{q},\mathbf{p},t)\right\rangle\,, \tag{7}$$

where the last term $\left\langle S^3(\mathbf{q},\mathbf{p})\right\rangle$ is called the *speckle-masking transfer function*. Detailed analysis by von der Lühe showed it to be real-valued (von der Lühe 1985), and thus does not contribute to the phases of $F_0^3(\mathbf{q},\mathbf{p})$.

We now turn our attention to the phases. Expanding $F_0^3(\mathbf{q},\mathbf{p})$ with Eq. 4, one finds

$$F_0^3(\mathbf{q},\mathbf{p}) = A_0(\mathbf{q})\,A_0(\mathbf{p})\,A_0(\mathbf{q}+\mathbf{p}) \times \mathrm{e}^{i[\phi_0(\mathbf{q})+\phi_0(\mathbf{p})-\phi_0(\mathbf{q}+\mathbf{p})]}\,. \tag{8}$$

Thus, the phases can be computed by solving

$$\mathrm{e}^{i\phi(\mathbf{q}+\mathbf{p})} = \mathrm{e}^{i\phi(\mathbf{q})}\mathrm{e}^{i\phi(\mathbf{p})}\mathrm{e}^{i\Phi(\mathbf{q},\mathbf{p})}, \tag{9}$$

where $\Phi(\mathbf{q},\mathbf{p})$ is the phase of the average bispectrum at position $(\mathbf{q},\mathbf{p})$. This can be solved recursively, starting at $\phi(\mathbf{0}) = \mathbf{0}$, by making only one assumption about the coordinate $\mathbf{q}'$ closest to $\mathbf{0}$. We assume the atmosphere has no effect on $\mathbf{q}'$. In this case, the phase can be estimated by averaging the phases of the individual images in the burst at that point.

Summarizing, one can find the Fourier phases of the object one by one. Together with the Labeyrie method, this completely describes the Fourier transform of the object, and simple back transformation will give the reconstructed image of the object in the burst.

### 3.3. Postprocessing

Once all the subimages have been reconstructed, they must be joined together to form a single large image. The subimages must be aligned with respect to each other and to the average recorded image. The average recorded image is obviously quite blurred, but it does represent a long-exposure image of the recorded object.

## 4. The Implementations

Currently, the data recorded by the Dutch Open Telescope in La Palma is reconstructed using an implementation of the speckle-masking algorithm in IDL, as has been noted in the Introduction. This implementation was written by P. Sütterlin, and is based on an implementation by C. de Boer.

I have developed an implementation of the speckle-masking algorithm in C. Large parts of this implementation are heavily based on the IDL implementation discussed above. The code was then parallelized using the MPI message-passing library. The only part not yet implemented in C is the reconnection. The reconnection is the most fragile part of the whole process. Logical arguments are not enough to position reconstructed subimages. The regions of overlap between two subimages, though very much alike, do not have to be exactly identical. This introduces small errors, in the order of one pixel, in the positioning of the subimages. The error will most likely not be noticed in a single image, but in a movie it may be painfully visible. The reconnection algorithm used for DOT data is still under development.

The implementation thus only covers the preprocessing and the speckle masking. Starting the preprocessing, images from the burst are distributed as evenly as possible over all CPUs, while keeping the lowest possible number of images on CPU 0, since some calculations are only performed on that CPU. During the preprocessing, the data is split into subimages. Since the speckle masking requires a full stack for every subimage, the data is shuffled to distribute the subimage stacks evenly over all CPUs, again minimizing the number of subimages on CPU 0. In an example, assume a burst of 100 images of 256 pixels square, and 8 CPUs. First, CPUs 1 through 7 will receive $\lceil 100/8 \rceil = 13$ images from the burst, while CPU 0 will receive 9 images. Every image will consist of 36 subimage stacks, assuming the standard configuration. This means that during the splitting, CPUs 1 through 7 will get $\lceil 36/8 \rceil = 5$ subimage stacks, while CPU 0 will receive only one subimage stack.

The vast majority of the time is spent in the reconstruction loop. Reconstruction requires a negligible amount of communication, and scales linearly with the number of subimages, and thus with the surface size of the input image. The total time is expected to increase approximately linearly with the input data size. Also, since the number of subimages is large (around 280 for the data that was used in the tests and 1000 for data recorded with the current cameras), inverse linear scaling is also expected with respect to the number of CPUs.

The requirements for the speckle masking and the preprocessing are very different. The preprocessing operates on the whole burst at once, using an amount of memory up to seven times the size of the burst, while the speckle masking operates only on one stack of subimages, requiring in the order of 10 MB of memory. However, the speckle masking is much more computationally intensive, and requires much less communication between CPUs. It is possible that in the future, the DOT will use a (multiprocessor) system to do the preprocessing, while a cluster consisting of fast machines without disk storage and with minimal amounts of memory will handle the reconstruction, retrieving the data needed from the preprocessing machine, and returning the reconstructed image. I will investigate both parts separately.

Figure 4 shows the schematic layout of the implementation together with estimates on the memory requirements. The code assumes the average flat- and darkfield have been precomputed, together with the noise power, and have been stored on disk. The program largely follows the explanation of preprocessing and speckle masking from Sect. 3.

The implementations introduce one nontrivial parameter, $N_{\mathrm{md}}$, the *masking depth*. Usually, it is just too much work to compute the phases for all possible combinations of $\mathbf{p}$ and $\mathbf{q}$ in Eq. 9. The bispectrum and the phases are computed only for those $\mathbf{p}$ and $\mathbf{q}$ where $||\mathbf{p} - \mathbf{q}|| \leq N_{\mathrm{md}}$. Increasing the masking depth produces more redundant information, which results in a better estimates for the phases being computed. Thus increasing the masking depth should result in a better reconstruction.

In order to make quantitative and qualitative predictions about the behavior of the implementation, I estimate the number of operations required and the time required for communication between CPUs. The number of non-floating-point operations is negligible. Operations are split between "simple" and "complex" operations. Floating-point additions, subtractions, and multiplications are classified as simple operations, and one such operation is assumed to take $s$ seconds. All other operations on floating point numbers, most frequently divisions or square roots, are classified as complex operations, taking $c$ seconds to complete. The bandwidth between CPUs is $b$ floating point numbers per second, $l$ is the network latency in seconds. $N_{\mathrm{bi}}$ is the number of images in the burst, $N_{\mathrm{b}}$ is the one-directional size of the data, $N_{\mathrm{si}}$ is the number of subimages in one full-field image, $N_{\mathrm{s}}$ is the one-directional size of a subimage, $N_{\mathrm{CPU}}$ is the number of CPUs used, and $N_{\mathrm{md}}$ is the masking depth. Obviously, since all subimages together cover the full-field image, $N_{\mathrm{b}}^2 = \alpha N_{\mathrm{si}} N_{\mathrm{s}}^2$, with $\alpha$ depending on the subimage overlap. For a reasonable overlap, $\alpha \approx 2$.

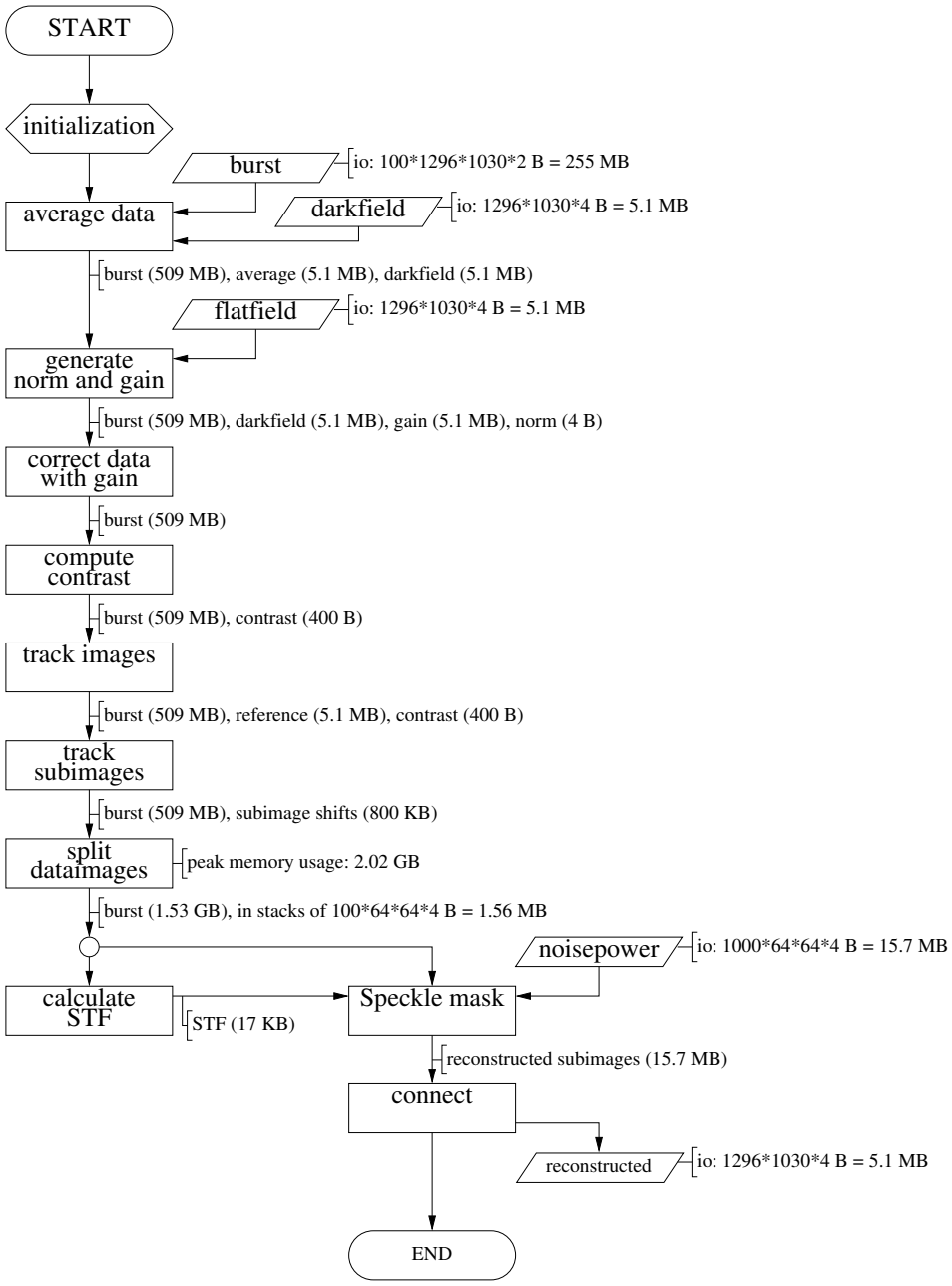


**Fig. 4.** The flowchart of the speckle-reconstruction implementation, also indicating the memory requirements, assuming a burst of 100 images of $1296 \times 1030$ pixels.

For the preprocessing, by analyzing the code by hand, I find

$$\begin{aligned} t_{\mathrm{calc,pre}} &= (N_{\mathrm{bi}} + 5\log_2 N_{\mathrm{b}} + 16)sN_{\mathrm{b}}^2 \\ &+ [(5.5 + 8N_{\mathrm{bi}})s + N_{\mathrm{bi}}c + 15N_{\mathrm{s}}^2 s \log_2 N_{\mathrm{s}}]N_{\mathrm{si}} \\ &+ (19.5s + 2.5c)N_{\mathrm{s}}^2 \\ &+ [(15N_{\mathrm{bi}}s\log_2 N_{\mathrm{b}} + 17.5N_{\mathrm{bi}}s + 2c)N_{\mathrm{b}}^2 \\ &+ (35.5N_{\mathrm{bi}}s + 3N_{\mathrm{bi}}c)N_{\mathrm{s}}^2 N_{\mathrm{si}} \\ &+ 15N_{\mathrm{bi}}N_{\mathrm{s}}^2 \log_2 N_{\mathrm{s}} s]N_{\mathrm{CPU}}^{-1}\,, \end{aligned} \tag{10}$$

$$\begin{aligned} t_{\mathrm{comm,pre}} &= \{N_{\mathrm{bi}}N_{\mathrm{b}}^2 + N_{\mathrm{s}}^2 N_{\mathrm{CPU}} \\ &+ [2(2 + N_{\mathrm{si}} + N_{\mathrm{si}}N_{\mathrm{s}}^2)N_{\mathrm{bi}}N_{\mathrm{CPU}}^{-1} \\ &+ 2N_{\mathrm{s}}^2 + N_{\mathrm{b}}^2]\log_2 N_{\mathrm{CPU}}\}b^{-1} \\ &+ (N_{\mathrm{bi}} + N_{\mathrm{si}} + N_{\mathrm{CPU}} + 8)l\,. \end{aligned} \tag{11}$$

A similar evaluation of the speckle-masking loop, again analyzing the code by hand, yields

$$\begin{aligned} t_{\mathrm{calc,msk}} &= (9.5s + 1.5c)N_{\mathrm{s}}^2 \\ &+ \{(11.75s + 10s\log_2 N_{\mathrm{s}} + 2.25c)N_{\mathrm{s}}^2 \\ &+ (14N_{\mathrm{md}}^2 + 3N_{\mathrm{md}} + 10\log_2 N_{\mathrm{s}} + 6)sN_{\mathrm{s}}^2 N_{\mathrm{bi}} \\ &+ (16s + 2c)N_{\mathrm{bi}}^2 \\ &+ \tfrac{\pi}{8}[(8s + c)N_{\mathrm{md}}^4 \\ &+ (53.5s + 19c)N_{\mathrm{md}}^2]N_{\mathrm{s}}^2\}N_{\mathrm{si}}N_{\mathrm{CPU}}^{-1}\,, \end{aligned} \tag{12}$$

$$t_{\mathrm{comm,msk}} = N_{\mathrm{si}}(N_{\mathrm{s}}^2 b^{-1}\log_2 N_{\mathrm{CPU}} + l)\,. \tag{13}$$

I have assumed that a two-dimensional Fourier transform of $N^2$ complex data points requires $10N^2 s\log_2 N$ seconds to complete, and all interprocessor (collective) communication is implemented ideally.

Assuming reasonable values for the variables this can be simplified. Typical order-of-magnitude values for the various parameters are $N_{\mathrm{bi}} \sim 100$, $N_{\mathrm{si}} \sim 10^3$, $N_{\mathrm{b}}^2 \sim 10^6$, $N_{\mathrm{s}}^2 \sim 10^3$, and $N_{\mathrm{md}} \sim 10$. It is also reasonable to assume that $s$ and $c$ do not differ more than an order of magnitude. Using these order-of magnitude estimates, the above equations reduce to

$$\begin{aligned} t_{\mathrm{calc,pre}} &= (N_{\mathrm{bi}} + 5\log_2 N_{\mathrm{b}})sN_{\mathrm{b}}^2 + 15N_{\mathrm{s}}^2 N_{\mathrm{si}}s\log_2 N_{\mathrm{s}} \\ &+ (15N_{\mathrm{bi}}\log_2 N_{\mathrm{b}}sN_{\mathrm{b}}^2 + 35.5N_{\mathrm{bi}}sN_{\mathrm{s}}^2 N_{\mathrm{si}})N_{\mathrm{CPU}}^{-1}\,, \end{aligned} \tag{14}$$

$$\begin{aligned} t_{\mathrm{comm,pre}} &= (N_{\mathrm{bi}}N_{\mathrm{b}}^2 + 2N_{\mathrm{si}}N_{\mathrm{s}}^2 N_{\mathrm{bi}}N_{\mathrm{CPU}}^{-1}\log_2 N_{\mathrm{CPU}})b^{-1} \\ &+ N_{\mathrm{si}}l\,, \end{aligned} \tag{15}$$

$$\begin{aligned} t_{\mathrm{calc,msk}} &= (9.5s + 1.5c)N_{\mathrm{s}}^2 \\ &+ [14sN_{\mathrm{bi}} + \tfrac{\pi}{8}(8s + c)N_{\mathrm{md}}^2]N_{\mathrm{md}}^2 N_{\mathrm{s}}^2 N_{\mathrm{si}}N_{\mathrm{CPU}}^{-1}\,, \end{aligned} \tag{16}$$

$$t_{\mathrm{comm,msk}} = N_{\mathrm{si}}(N_{\mathrm{s}}^2 b^{-1}\log_2 N_{\mathrm{CPU}} + l)\,. \tag{17}$$

## 5. Test Results

It is of interest to investigate the performance of the implementation presented here, also in comparison with the IDL implementation. We will present data regarding speed for both implementations and the scaling of the parallel C implementation will be investigated on various platforms. In order for the numbers presented here to be representative for the production environment, machines similar to those in use on La Palma were used wherever possible.

In all these cases $N_{\mathrm{md}} = 9$, $N_{\mathrm{bi}} = 100$, and $N_{\mathrm{s}} = 64$.

### 5.1. IDL

The benchmarks presented here were performed on a dual 450-MHz Pentium II. This machine resembles the machines used on La Palma. The IDL implementation uses a lot of intermediate storage. All data files are stored in XDR format to allow for cross-platform compatibility. During the tests it was discovered that this introduced quite some overhead, and the code was promptly changed to allow for non-XDR format intermediate storage.

For a burst of 100 images of $768 \times 572$ pixels, the preprocessing took roughly 600 s when not using the XDR format. Writing in XDR results in 135 s of additional overhead when storing in little-endian format and 170 s when storing in big-endian format.

Only ten of the 280 subimages were reconstructed. This took 558 s. If all subimages were to be reconstructed, it would take 15612 s. This means that 96% of the time is spent in the main reconstruction loop.

### 5.2. Sequential C

In order to compare the C code with the IDL code, a complete run was timed on the same machine as the IDL code was timed on. In total, the reconstruction took 7025 s. Thus, overall, the C code is 2.3 times as fast as the IDL code when running on a single CPU.

Table 1 shows a comparisons between the various functions in the preprocessing in IDL and C. Most functions show a speed increase by a factor of 2, roughly. The only large difference is in the image-tracking routine, which shows a speed increase by a factor of 30. This probably is a result of the highly sub-optimal Fourier transforms in IDL in combination with a smoothing function to remove large structures. The latter has been replaced with a simple and cheap Fourier filter in the C implementation.

More tests were run on a 1200-MHz Athlon. The implementation uses the Fourier transform library FFTW, of which a special optimized version exists for the Athlon CPU. I have run the program with this optimized library, timing the preprocessing and the reconstruction. The times for startup and initialization are not included. Figure 5 shows the behavior of the code. The curve in the figure represents linear scaling with respect to the data surface size, and it seems to fit well. The points are not exactly on the curve, since the number of subimages is discrete. This can most clearly be seen from the diamond at 400 pixels and the circle at 448 pixels. These input data sizes produce the same number of subimages. This effect introduces larger errors as the input data size goes up. When going from, for instance, six to seven subimages in both directions, the relative increase is larger than when going from eight to nine, but the absolute difference is smaller. Note that the diamonds and circles are on the same curve, indicating that Fourier transforms of arrays whose size is not a power of two do not slow down the computation much. This is in agreement with the model, which shows that the contribution of Fourier transforms is small.

The dashed curve in Fig. 5 is the scaling as predicted by the model. Though it shows the same behavior as the measurements, for large images its predictions are 50% off. This is contributed to cache effects. The CPU benchmarks on the Athlon show that the time required for one operation increases by a factor of 4 if the data

| function | C | IDL | factor |
|---|---|---|---|
| data correction, computation of the contrast, norm, and gain | 20.9 s | 32.8 s | 1.6 |
| image tracking | 4.3 s | 126.0 s | 29.3 |
| subimage tracking, splitting and noise power computation | 160.5 s | 356.8 s | 2.2 |
| computation of the STF | 37.3 s | 77.3 s | 2.1 |

**Table 1.** The comparison of specific parts of the preprocessing in C and IDL. Most functions show a speed increase by a factor of roughly 2, with the notable exception of the image tracking. The discrepancy is attributed to much faster Fourier transforms and the replacement of a smoothing function with a Fourier filter.

size increases somewhat beyond the cache size. I find $s \approx c \approx 2.4 \times 10^{-8}$ s for data sizes much larger than the cache size, and $s \approx c \approx 7.1 \times 10^{-9}$ s for data sizes smaller than the cache size. The dashed curve was drawn assuming that during the preprocessing the data is not in cache, and that it is during the masking. Filling in the known quantities in Eqs. 10 and 12 I find

$$t = 0.92 \text{ s} + 1.6 \times 10^{-3} N_\text{b}^2 \text{ s} + 3.8 \times 10^{-5} N_\text{b}^2 \log_2 N_\text{b} \text{ s}. \quad (18)$$

This also assumes a ratio of 2.25 between $N_\text{s}^2 N_\text{si}$ and $N_\text{b}^2$. It must be noted that the last term only becomes significant when $\log_2 N_\text{b} \approx 45$. This implies an image size of some $10^{13}$ pixels square, and thus the last term can be safely disregarded.

The two dotted curves are similar predictions as the dashed curve. The top curve assumes both the masking and the preprocessing are not in cache, while the bottom curve assumes both are in cache. It is difficult to say how much of the masking loop is in cache and how much isn't. Closer inspection of the code indicates arrays that don't fit in the Athlon's cache are used. This means the expected time spent is somewhere between the top dotted curve and the dashed curve. The measurements are indeed within these boundaries.

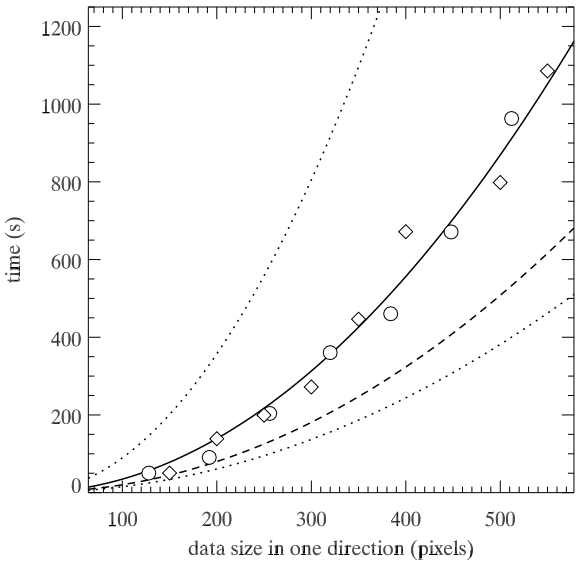


**Fig. 5.** The behavior of the code as a function of the input data size. The size of the sides of the input data is plotted horizontally. The data is a square of the indicated size. Circles are with the input data a multiple of 64 pixels, while diamonds are at multiples of 50 pixels. The curve drawn through the points indicates linear scaling with the data surface size, $y = \alpha x^2$. The dashed curve represents the predictions of the parallelization model, Eq. 18. The dotted curves are also predictions of the model, making different assumptions about cache effects than the dashed curve.

### 5.3. Parallel C

The parallel code was tested on a Beowulf cluster consisting of sixteen 180-MHz Pentium Pro based machines, connected with a 100-Mbit Ethernet. These machines were used for educational purposes before they were installed as cluster computer. Though not very powerful, this system was a very useful tool during the development of the code. This system is, however, not a good representation of the setup currently on La Palma, nor will it ever be. But benchmarks on this machine will still show the strengths and weaknesses of the implementation.

In order to compare the measurements with the model, the values of $s$, $c$, $b$ and $l$ must be known. I have timed simple and complex operations on data similar in size to the data used in these experiments. I find $s \approx 7.8 \times 10^{-8}$ s, $c \approx 3.6 \times 10^{-7}$ s. I find $b \approx 10^6$ floats/s with a latency of $l \approx 0.6$ ms. Benchmarks of the MPI implementation MPICH, which was used, showed non-$\log_2 N_\text{CPU}$ behavior for `gather` and `allgather` collective communication. It is expected that the preprocessing will scale to a larger number of CPUs if this is remedied.

To be able to test the implementation on a large range of CPUs with the same data set, the input data size was limited to 256 pixels square because of memory requirements. Figure 6 shows the behavior. The dashed line is the scaling as predicted by the model. With Eqs. 10, 11, 12 and 13 the predicted relation is found to be the sum of

$$t_\text{pre} = 7.48 \text{ s} + 126 N_\text{CPU}^{-1} \text{ s} + 4.70 \times 10^{-3} N_\text{CPU} \text{ s} + 7.37 \times 10^{-2} \log_2 N_\text{CPU} \text{ s} + 14.8 N_\text{CPU}^{-1} \log_2 N_\text{CPU} \text{ s}, \quad (19)$$

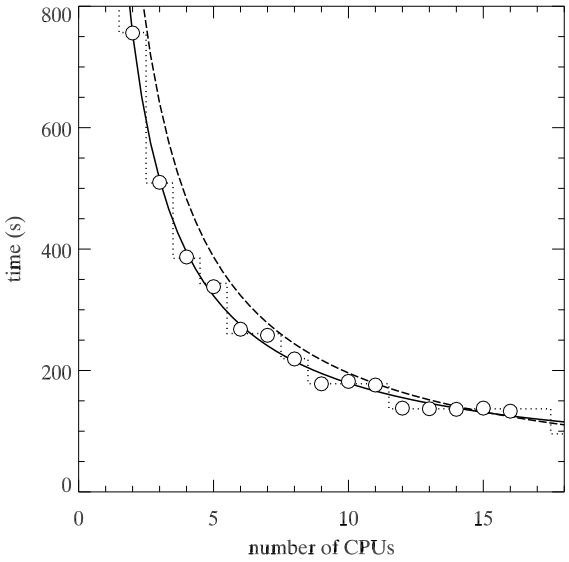


**Fig. 6.** The behavior of the code on a cluster computer consisting of sixteen 180-MHz Pentium Pro machines with switched 100-Mbit Ethernet interconnects, using 256-pixel square input data. The solid curve indicates inverse linear scaling with respect to the number of CPUs, and seems to fit quite well in the cases where the number of subimages can be spread almost evenly over all CPUs. The dashed curve indicates what is expected from the parallelization model, Eqs. 19 and 20. The dotted stepfunction is given by Eq. 21.

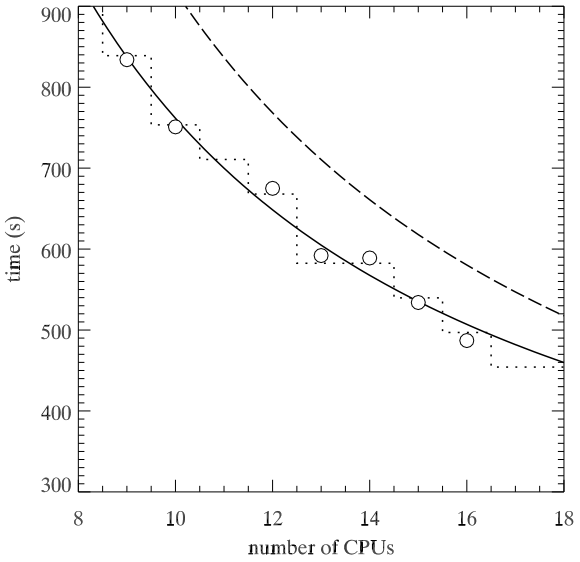


**Fig. 7.** The behavior of the code on a cluster computer consisting of sixteen 180-MHz Pentium Pro machines with switched 100-Mbit Ethernet interconnects, using 512-pixel square input data, Eqs. 22 and 23. The dotted stepfunction is given by Eq. 24.

and

$$t_{\mathrm{msk}} = 2.68 \times 10^{-2}\ \mathrm{s} + 1767 N_{\mathrm{CPU}}^{-1}\ \mathrm{s} + 0.15 \log_2 N_{\mathrm{CPU}}\ \mathrm{s}\,. \quad (20)$$

The stepping behavior can easily be explained by the fixed number of subimages. In this case, there are 36 subimages. At 12 CPUs, there are $\lceil 36/12 \rceil = 3$ subimages per CPU. At 16 CPUs, there are still $\lceil 36/16 \rceil = 3$ subimages on several, but not all, CPUs. This means that the reconstruction loop takes the same amount of time. Since the contribution of the preprocessing is minimal, the speed difference between 12–16 CPUs is not measurable. In fact, since the preprocessing and communication do not much contribute to the total time, a stepfunction should be able to fit the data quite well. The dotted line in the figure is the figure is the curve

$$t = 12.9\ \mathrm{s} + 41.3 \lceil 36/N_{\mathrm{CPU}} \rceil\ \mathrm{s}\,. \quad (21)$$

It must be noted that the effect is very pronounced here because of the limited memory capacity of the machines. As the ratio between the number of subimages and the number of CPUs increases, the effect will become less noticeable. Since the number of subimages is around $10^3$ for an image recorded with the current cameras, this effect should remain hardly noticeable while the number of CPUs is under a few hundred.

These measurements indicate the preprocessing on this machine with this data size is not expected to scale beyond 20 CPUs. The model indicates that the speckle-masking loop is expected to scale to $10^5$ CPUs, assuming there are sufficient subimage stacks.

For larger input data sizes, the model overestimates the time required on this machine. Figure 7 shows the comparison between model and measurements for input data of 512 pixels square. The dashed line is again the model, and it is given by the sum of

$$t_{\mathrm{pre}} = 29.8\ \mathrm{s} + 577 N_{\mathrm{CPU}}^{-1}\ \mathrm{s} + 4.70 \times 10^{-3} N_{\mathrm{CPU}}\ \mathrm{s} + 0.27 \log_2 N_{\mathrm{CPU}}\ \mathrm{s} + 69.3 N_{\mathrm{CPU}}^{-1} \log_2 N_{\mathrm{CPU}}\ \mathrm{s}\,, \quad (22)$$

and

$$t_{\mathrm{msk}} = 0.11\ \mathrm{s} + 8293 N_{\mathrm{CPU}}^{-1}\ \mathrm{s} + 0.96 \log_2 N_{\mathrm{CPU}}\ \mathrm{s}\,. \quad (23)$$

The stepping effect is again clearly visible. The dotted line is given by

$$t = 26.7\ \mathrm{s} + 42.8 \lceil 169/N_{\mathrm{CPU}} \rceil\ \mathrm{s}\,, \quad (24)$$

where 169 is the number of subimages for this image size.

For this input data size, the preprocessing again does not scale well beyond 20 CPUs. This is not unexpected, since both the computation and the bulk of the communication scale linearly with the input data size. Again, the speckle masking is expected to scale well up to the limit imposed by the number of subimage stacks.

## 6. Conclusions

The C implementation shows a definite speed increase as compared to the IDL implementation. On the same hardware, the C version delivers roughly twice the performance. The implementation also shows that the speckle masking itself can be run in parallel on a large number of CPUs. The preprocessing of the data does not scale as well. Both parts of the reconstruction do behave similar to what the theoretical model derived from the code predicts.

Measurements on the Pentium Pro cluster indicate that on that cluster the preprocessing does not scale beyond 20 CPUs. Assuming the network performance remains the same, on a cluster of Athlons such as the one benchmarked the preprocessing will not scale much beyond 2 or 3 CPUs. Using conservative estimates, the model indicates the masking loop will scale well on a cluster of Athlons up to 100 CPUs.

Since the preprocessing does not scale as well and the hardware requirements for the preprocessing and the masking are very different, a large cluster of homogeneous hardware is obviously not the correct choice of reconstruction farm for the DOT. Most likely a system consisting of several sets of a multiprocessor machine for the preprocessing in conjunction with several possibly diskless, fast machines with little memory would be the system of choice. The number of masking nodes per preprocessing node is hardware dependent and should be investigated on a case-by-case basis. For the Pentium Pro cluster, the ratio would be approximately one preprocessing machine for twenty masking nodes. For use on this kind of system the code will have to be adapted to a master-slave system, but this should not be very difficult. The number of such sets of machines desired depends on the amount of input data and the time constraints on its processing. Several of these systems would provide a fast masking pipeline that would greatly increase the scientific potential of the DOT.

*Acknowledgements.* I would like to thank the DOT team, R. Hammerschlag, F. Bettonvil, R.J. Rutten, and P. Sütterlin, for the support they have given this project. P. Sütterlin in particular deserves mention for many useful discussions regarding speckle masking. I have benefited much from A. van der Steen's guidance and extensive knowledge on parallel processing.

## References

de Boer, C. R. 1993, Ph.D. Thesis
Lohmann, A. W., Weigelt, G., & Wirnitzer, B. 1983, Appl. Opt., 22, 4028
von der Lühe, O. 1985, A&A, 150, 229
Weigelt, G. P. 1977, Optics Communications, 21, 55