High-Resolution IRAS Maps Parallelised

Tj.R. Bontekoe[1,2] G.D. van Albada[3]
W. Hoffmann[3] D.J.M. Kester[4]
L.B.F.M. Waters[1]

1: Dept. of Astronomy, Univ. of Amsterdam, romke@astro.uva.nl
2: Bontekoe Data Consultancy, Herengracht 47, 2312 LC Leiden
3: Computational Science Group, Univ. of Amsterdam, dick@wins.uva.nl
4: Space Research Organisation Groningen, Postbus 800, 9700 AV Groningen

Keywords: Maximum Entropy, Image Processing, Parallelisation, Peano Curves, Sparse Matrices

Abstract

The strip-scanned data obtained by the IRAS satellite are being used to produce maps of the infra red emission from large areas of the sky. The resolution is enhanced to nearly the telescope diffraction limit using Maximum Entropy Processing. The memory and processing requirements of this algorithm necessitate the use of parallel computers for large fields. We describe a number of techniques used to accelerate the MaxEnt code and describe some preliminary results obtained with these techniques.

1 Background

Astrophysical processes are best studied from images which cover a large range in scale lengths. The IRAS (Infra Red Astronomical Satellite) data constitute a unique source of information from the largest possible scale, viz. the entire sky, down to the scale of the optical diffraction limits of its telescope. The IRAS spacecraft and mission were optimised for the detection of point sources, as the sky was scanned in narrow, overlapping strip scans. Consequently, the data are not in the form of images but consist of a set of time-ordered, position-tagged samples. Standard processing of the data results in maps having a spatial resolution of about the size of the IRAS detectors. Maximum Entropy (MaxEnt) processing improves this (linearly) by a factor 3–5, and yields a spatial resolution approaching the telescope diffraction limits (1'–2', depending on wavelength). The algorithm used (MemSys5) gives astrophysically reliable images (Bontekoe et al. 1994). However, it is quite CPU intensive and therefore is usually applied to relatively small areas of the sky (say, 1° by 1°). The aim is to accelerate the process by about two orders of magnitude.

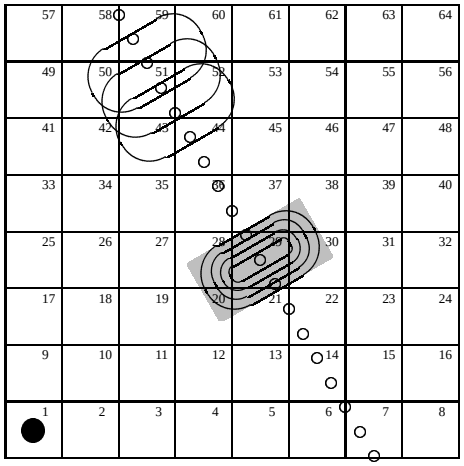

Figure 1: Illustration of IRAS detector scan crossing the sky.

Figure 1 illustrates the geometry of a single detector scan projected onto the map pixel grid. The open circles represent the centre of the detector at the sampling instants. The ovals in the grey rectangle indicate the contours of the sensitivity profile of the detector surface (the detector Response Function, or RF). The grey rectangle is the "support", or array size, for the RF. The three ovals in the upper-left corner indicate the way consecutive samples partially overlap. The filled circle indicates the IRAS telescope diffraction circle at the same scale as the detector RF. The pixels as shown here are about a factor ten larger than in actual reconstructions.

The single detector readout value, when positioned as the grey rectangle, is a weighted sum of the (unknown) intensities in pixels 20, 21, 28, 29, 30, 37, and 38. The weights are the integrals over the Response Function over the pixel area. (The total integral over

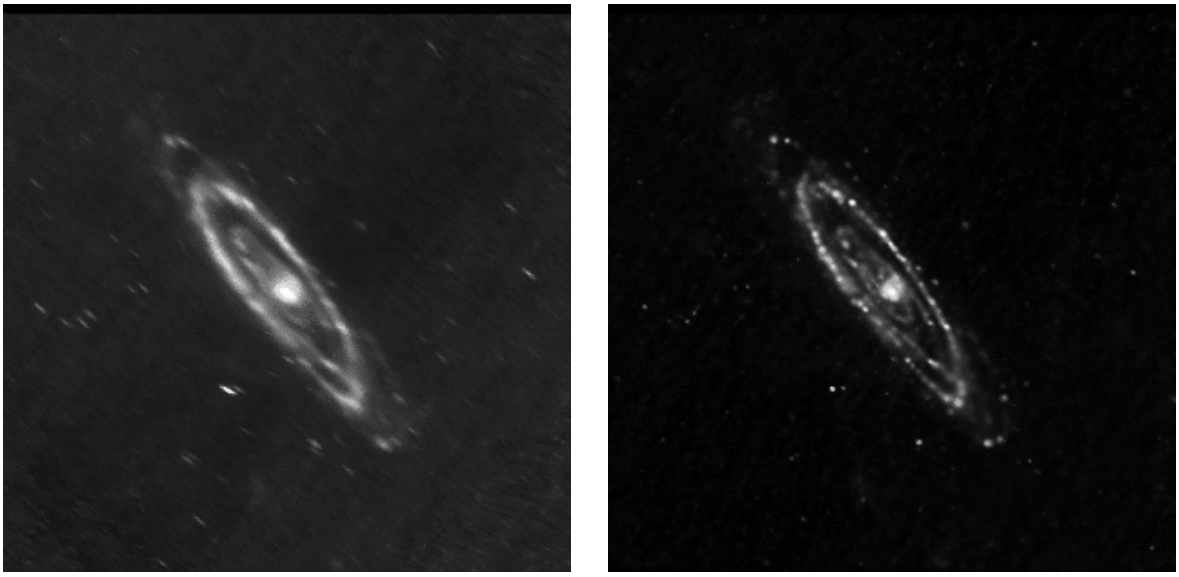

Figure 2: The Andromeda nebula standard processing (l) and Maximum Entropy (r) from IRAS data.

the grey area is normalised to unity, and thus the sum of the weights will be unity when the detector does not overlap the map boundary.) If one takes the weights of all pixels not covered by the detector as zero, the datum value can be written as a vector inner product of the image vector f (here 64 pixels) and a row vector with mostly zeros except for the positions enumerated above where the weights are stored. This can be extended to all detector data, yielding the imaging equation:

$$d = R\,f, \tag{1}$$

where d is the data vector, f the image vector, and R the response matrix with all the weights. This matrix equation gives a complete description of the IRAS sampling geometry on the sky.

Taking the image reconstruction of the Andromeda nebula (at 60 μm wavelength) as an example of a medium size field (Fig. 2). The image has a size of 3.4° square, and is computed on 1000^2 pixels. The number of data samples taken in this area is about 7 10^5. The 7 10^{11} elements of R are only sparsely filled with about 10^9 non-zero values. The fact that this system of equations is under-determined is not relevant for our method of image reconstruction, though more data give better reconstructions.

2 MaxEnt reconstruction

Practical data carry uncertainties, sometimes (incorrectly) called noise. The image reconstruction depends on the quality of the data, and therefore the imaging equation is modified to

$$d = R\,f \pm \sigma. \tag{2}$$

Here the vector σ represents the uncertainty in the data. The detailed statistics of IRAS data is unknown; we always choose normal distributions. This formalism allows each datum to have a different uncertainty, a property we use extensively to model various instrumental effects.

Since the entropy requires independence of its elements and since f *must* be spatially correlated, the MaxEnt solution requires an extra *Intrinsic Correlation Function* (ICF). The ICF transforms an uncorrelated *Hidden Space Image* h into the *Visible Space Image* f:

$$f = C\,h. \tag{3}$$

The entropy $S(h)$ is maximised as function of h, and not of f. We apply the Pyramid Maximum Entropy scheme, which induces all relevant spatial scale lengths simultaneously in the image via the ICF (Bontekoe et al. 1994). This scheme performs a local blurring of the hidden space solution, with a range of scale lengths. The transform $f = Ch$ allows a faster algorithm than matrix-vector multiplication by using a window function for this blurring.

The MemSys5 package (Gull and Skilling, 1991) solves h from

$$d = RC\,h \pm \sigma, \tag{4}$$

from which the "astronomical image" f is easily calculated. The MemSys5 package iterates between forward transforms and transpose transforms; the latter require the transposed matrices R^T and C^T also to be available. Note that a pseudo inverse matrix is never used.

After convergence of MemSys5 the IRAS data requires a new self-calibration step. The original scan data are updated for a better zero point, systematic drift effects with time, and detector gains. Then the whole process starts again until the entire process has

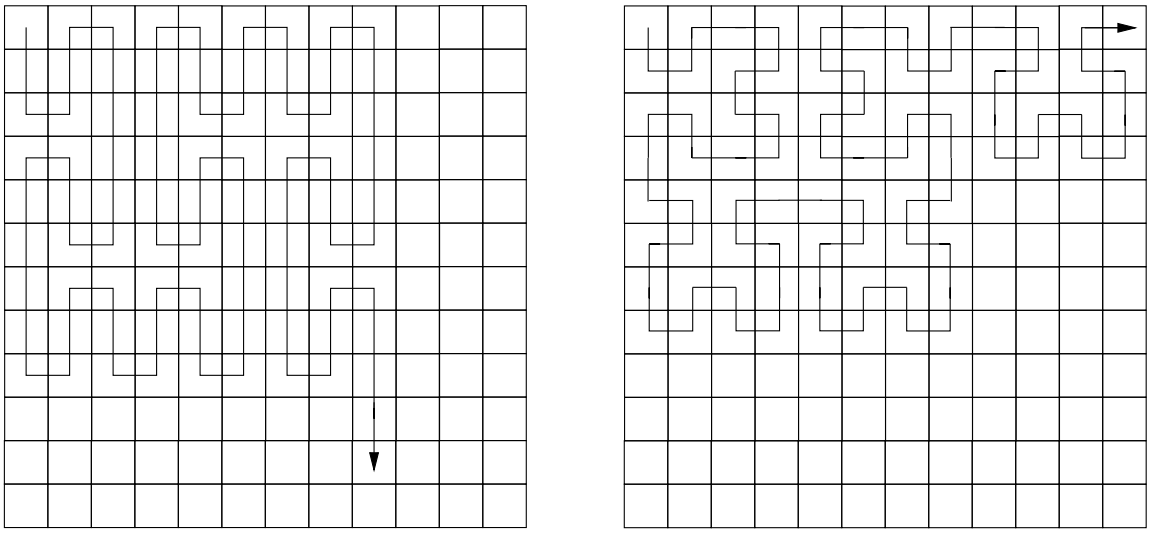

Figure 3: Two examples of Peano curves.

converged. At this point typically 10^4 to 10^5 transform operations have been performed. The main computational bottleneck lies in the amount of intermediate data constituting the R matrix, which are repeatedly needed.

Currently on standard workstations the (compressed) matrix R for an image like the Andromeda nebula, and its transpose, cannot be stored in memory. However, the entire matrix can be computed from relatively few input values. Currently, we compute the non-zero parts of each row of R when needed for the multiplication with $\hat{f}$. Then this row is overwritten by the next row, and has to be re-computed for the next iteration. Similarly for the other transforms. About 90 percent of the computing effort is in the four transform operations (C, R, R^T, and C^T). Though the ICF constitutes a significant fraction of the computing time, we concentrate in this paper on the speedup of the response matrix transform.

3 Sparse matrix storage

As stated before, R is a sparse matrix, each row describing the contribution of a point on the sky to a given data sample. Adjacent points on the sky will contribute to the same data sample, and a single sky position will be included in a number of consecutive observations. Referring to Fig. 1 the storage of the image $\hat{f}$ is row-wise, as indicated by the pixel numbers. Only the pixels (partly) covered by the grey rectangle are taken in consideration. The consecutive "runs" in memory are pixels 20–21, 28–30, 37–38. (In practice the average run length is about fifteen pixels.) For each datum we store the number of runs (here 3). Then for each run we store the starting addresses (here 20, 28, and 37), and the run lengths (resp. 2, 3, and 2). Using the standard pixel numbering in images, each row of R will thus contain some twenty runs of consecutive non-zero values, each of about fifteen elements. The R-matrix can thus be stored efficiently using a run-length type encoding; both access speed and memory requirements can be improved in this way. No suitable standard data structure for this type of sparse array was found in (Saad, 1994), so one was designed and implemented.

4 Peano curves

The efficiency is further improved if there are fewer runs; each run then necessarily becomes longer. Essentially this implies that nearby pixels must have nearby numbers. This can, to a degree, be realised by using a suitable space-filling Peano curve to number the pixels. The curve used in a first experiment is shown in the left-hand panel of Fig. 3. It has the advantage that the conversion from image coordinates to position on the curve and vice versa can be calculated very simply. It has the disadvantage of being based on powers of three. The standard power-of-two pattern (the Peano-Hilbert curve) depicted in the adjacent frame is more difficult to program as it involves both mirroring and rotations of the basic pattern, but was found to perform better.

We found that for both numbering schemes the number of runs in R was only slightly reduced (as was to be expected) [1] , but the statistics of the run-lengths

[1]The number of runs in a "patch" is dependent on the number of edge pixels in the patch, which is constant, and the probability that the next pixel on the run lies outside the patch. It can easily

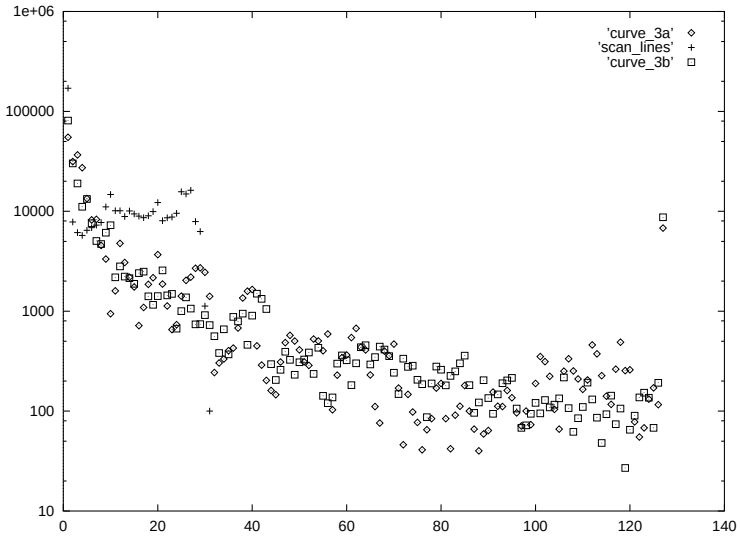

Figure 4: Histogram of the run-lengths for the three different pixel-numbering schemes. The standard row-wise pixel numbering (scan-line) has no runs longer than 31 pixels; both other schemes have more very short and very long runs. The bin at 127 counts all runs of 127 and more pixels.

changed dramatically (Fig 4). We now had a number of long runs containing the majority of the non-zero values, and a larger number of short runs, mostly containing just a few insignificantly small values. If, as we expect, the non-inclusion of these small values does not significantly affect the restoration process, we find we can delete up to 40% of all runs, together containing about 5% of the non-zero elements of R.

The optimisations have not yet been implemented in the code. The attainable performance gain should be between 5% and 10% at least. Improved locality in the accesses to $\hat{f}$ and R and a better loop optimisation may further improve the performance.

5 Parallelisation

The MemSys5 algorithm exhibits coarse-grain parallelism, since only vector–matrix multiplications are involved. To be more specific, in the forward transform ($d = Rf$) essentially only vector inner products are used, viz. the vector $\hat{f}$ with a row of R to give one element of the "mock data" vector $\hat{d}$. In the transpose operation, only scalar-vector products are required. The order of which these multiplications are performed is immaterial. This allows a master–slave parallel paradigm in which the matrix R and the vector d are subdivided into parts of several rows, and distributed over the available processors. The image vector $\hat{f}$, which is not subdivided, is broadcast to all processors and multiplied by these sub-matrices of R, yielding sections of $\hat{d}$. These $\hat{d}$ sections are then returned to the master program, where they are combined into the full mock data vector $\hat{d}$. The transpose transform is done similarly.

This paradigm requires the sub-matrices to be communicated once to the slave processes. The original computational scheme, where each row of R is computed when needed and overwritten after multiplication with $\hat{f}$, appears quite wasteful in terms of CPU use, though it is very efficient in terms of memory use. In the current parallel systems each processor has about as much memory as a standard workstation. Therefore it was decided to write out the matrix R in a compressed format and distribute its sub-matrices over the processors.

This scheme is being implemented. A first test shows an acceleration of at least a factor two in the R and R^T transforms.

6 Conclusions

The use of the Maximum Entropy method makes it possible to produce reliable maps of the infra red emission of the sky from the strip-scanned IRAS observations. The procedure is computationally intensive, but can be significantly accelerated by retaining an intermediate matrix, the response matrix R. The size of this matrix is such that parallelisation not only helps to further accelerate the code, but is also necessary to store the matrix for large fields. The storage requirements and data locality are further helped by the use of

be seen that this probability will not change dramatically between numbering schemes using space-filling curves. The standard row-wise pixel numbering scheme already behaves very much like a space-filling curve in this sense.

a suitable sparse matrix storage scheme and by renumbering the pixels in the image.

References

Bontekoe Tj.R., Koper E., Kester D.J.M., 1994, A&A 284, 1037.

Gull, S.F., Skilling, J., 1991, MemSys5 Users Manual, Maximum Entropy Data Consultants Ltd.

Saad, Y, 1994, *SPARSKIT: a basic tool kit for sparse matrix computations*, http://www.cs.umn.edu/Research/arpa/SPARSKIT/paper.ps