Evaluating Optical-Flow Algorithms on a Parallel Machine

M. Fleury, A. F. Clark, and A. C. Downton

Department of Electronic Systems Engineering, University of Essex

Wivenhoe Park, Colchester, CO4 3SQ, U.K

tel: +44 - 1206 - 872817

fax: +44 - 1206 - 872900

e-mail: fleum@essex.ac.uk

keywords: optical-flow, parallelisation, algorithmic testing

Abstract

Algorithmic development of optical-flow routines is hampered by slow turnaround times (to iterate over testing, evaluation, and adjustment of the algorithm). To ease the problem, parallel implementation on a convenient general-purpose parallel machine is possible. A generic parallel pipeline structure, suitable for distributed-memory machines, has enabled parallelisation to be quickly achieved. Gradient, correlation, and phase-based methods of optical-flow detection have been constructed to demonstrate the approach. The prototypes enabled comparisons to be made between the speed when parallelised and (already known) accuracy of the three methods when parallelised, on balance favouring the correlation method.

1 Introduction

Optical-flow (OF) methods intended for pixel-level, video stream inter-frame motion estimation are complex, multi-component algorithms with lengthy execution times if processed on a workstation or PC. The purpose of this paper is to examine whether parallelisation would enable algorithmic designers to speed-up the development cycle, thus encouraging further

activity in this promising field. To that end, three representative OF methods have been parallelised. This paper contains: details of algorithmic structures; implementation considerations; timings; and assessments of the suitability for parallelisation of each of the chosen OF methods.

Despite their complex structures, OF methods can be parallelised by mapping onto a 'parallel pipeline'. The developer will want to use a variety of test sequences in order to adjust parameters, which suggests a continuous flow of test sequences through a pipeline. A parallel pipeline consists of one or more stages each of which may employ processors working in parallel within a data farm. Component algorithmic parts of an OF method are partitioned across the pipeline stages. Provided there is a continuous flow, a pipeline will reduce the throughput of test sequences, while internal parallelism will reduce the time to test an individual sequence. After the OF field has been found, further processing stages may be needed. For example, motion segmentation or frame in-betweening are possible further steps, which can easily be added to the generic pipeline structure. No one current OF method is suitable for all purposes. Therefore, a number of OF prototype parallel pipelines have been implemented on a multicomputer to form an algorithmic testbed. A generic method of parallel pipeline construction eased the burden of repeated parallelisations.

The paper is organised as follows. Section 2 takes a closer look at the nature of the problem faced by developers of optical flow applications. Section 3 is background material on parallel implementation addressing the issue of how to quickly decompose an existing sequential code. Section 4 is a detailed description of the three implementations. Section 5 is an analysis and evaluation of the results of the parallelisation. Finally, Section 6 is a summary.

2 Optical Flow Methods

In this Section: the three OF methods are introduced, Section 2.1; problems faced in algorithmic development are considered in Sections 2.2 & 2.3; and sequential processing times are given in Section 2.4.

A 2-D OF dense vector field, akin to a disparity field, can be extracted from a sequence of video frames by processing the image irradiance over time, without the difficulty of tracking corresponding features in the 3-D scene structure [1]. However, due to the well-known aperture problem only a single component of motion may be available by such processing, unless there is further amalgamation of results. Amorphous imagery, as found in weather images, [2] can be tackled. Our OF implementations were parameterized for small pixel motions (three or fewer pixels per-frame) suitable for applications such as monitoring heart motion [3] or plant growth [4]. Sub-pixel estimates of motion are required for applications such as plant growth. Inaccuracies, typically blurring at motion boundaries, are particularly visible in motion-adjusted film sequences, such as in frame-rate conversion.

2.1 Existing sequential implementation

Three OF routines, chosen for their representative value, have been parallelised: a gradient-based method [5, 6] (LK); a correlation method using a Laplacian pyramid [7, 8] (AN); and a phase-based method [9, 10] (FJ). Public-domain sequential code first written for a comparative study [11], available from `ftp://csd.uwo.ca/pub/vision/`, served as the starting point for the parallelisations. All the original sequential code was written in the 'C' programming language.

Because the issue of the optimal accuracy available has already been addressed in [11], we do not pursue this issue except to refer to established results. An authenticated, standard set of video sequences created by synthetic means[1] is also available at the same `ftp` site. The video sequences range up to approximately common intermediate format (CIF) in size (352×288 pixel luminance image at 30 frames/s). [12] is a complementary survey of OF techniques, while [13] refers to the same imagery and algorithms.

[1] Standard natural scene sequences are also available.

2.2 The problem of spatiotemporal filtering

In [11], attention was drawn to the importance of implementational techniques: pre-processing by temporal smoothing over a large number of frames, as well as Gaussian smoothing and possibly demodulation over larger spatial neighbourhoods; choice of window size in search areas and parameters in relaxation routines; and in post-processing through choice of threshold or confidence measure, which should be correlated with known errors found from synthetic image sequences. Unfortunately, increased spatiotemporal smoothing has the potential to slow processing further.

The need for temporal smoothing occurs because of the trade-off between allowed velocity, $\mathbf{v}$, temporal frequency (fixed), ω_t, and spatial frequency, $\omega_{\mathbf{x}}$ given by $\mathbf{v} = \omega_t/\omega_{\mathbf{x}}$, which is a consequence of the temporal sampling theory. In cinematography, for reasons of audience response, temporal aliasing is not suppressed, requiring temporal smoothing to disambiguate motions. Artificial motions such as those produced in 'stop-and-shoot' test sequences can also give rise to temporal aliasing. Considerable numbers of frames, twenty-one or more [9], are deemed necessary to smooth a single frame estimate. Recursive methods of temporal smoothing have been experimented with [14], though with some loss in accuracy as full smoothing requires non-causal processing. Recursive methods by definition require repeated synchronization which make them unsuitable for a scalable parallel solution catering for increased image sizes or faster throughput.

2.3 Algorithmic Variation

A survey of OF methods reveals that no one method is suitable for all imagery and all purposes. However, building several bespoke parallelisations is much less appealing than developing a generic method which can be easily adapted to each algorithm.

Energy methods [15] will cope with randomly-patterned surfaces, i.e. too much image structure, in a way that methods relying on conservation of image irradiance over the interlude of a frame will not. Phase-based methods are immune to contrast variance in a way that

energy-based methods are not. Both energy- and phase-based methods will fail if the complex-valued Gabor detection filters are not tuned to the spatiotemporal frequency of interest. Localised gradient and correlation methods cannot support local area deformation between frames as might occur from scene object rotation, and camera pan or zoom, that is if the scene capture rate precludes piecewise linear approximation of these motions. The viewing model is further restricted to orthogonal scene projection as perspective distortion also changes the area of projected regions. Localized methods also assume diffuse Lambertian surface reflectance, strictly supporting only camera motion as scene illumination should be constant.

Systematic analysis of the algorithmic variants should include: the method of implementation in support of the basic algorithm; the number of frames available; the accuracy of an algorithm; the density of measurements required by the application; the camera viewing arrangements; the type of motion of objects in an imaged scene; and the image structure present, including noise.

2.4 Processing difficulties

There has been extensive OF algorithmic development over many decades [16]. However, reported processing times remain slow. For example, in [17] a robust algorithm on a 256×256 pixel image sequence took about $3\frac{1}{2}$ hours on a Sparc-10 workstation, while other algorithms ranged between 30 minutes [8] and 5 minutes to perform. [18] took 61.4 s per frame for their algorithm on a SPARCstation 20, but this time was for an 128×128 sequence. [19] report also on a 128×128 pixel sequence[2], 4 minutes per frame on a SUN workstation and interestingly 10s per frame on a 128 processor CM5, which should be compared with the timings in Section 5.[3] Alas, few users will have access on a regular basis to the equivalent of a CM5, whereas access to a less powerful parallel machine, such as the one described in

[2]For comparison, broadcast quality images are 720×576 pixels in size.

[3]The multicomputer timings, however, include I/O. The CM5 has a 'fat' tree, constant bandwidth, network with single instruction multiple data streams (SIMD) support though usually operating in multiple instruction multiple data streams (MIMD) mode. I/O provision is considerable.

Section 5, is more likely. The machine used may be a legacy system, no longer state-of-the-art due to the inexorable effect of Moore's law but still able to serve a purpose. The eventual target hardware may be at VLSI level. For example, the TI 'C80 or MVP is a good match to distributed-memory architectures having four RISC processors and a control processor with floating-point unit on the same die.

3 Developing a parallel prototype

Our procedure for producing the parallel prototype is as follows. The first stage is a semi-manual partition of an existing sequential code. A static analysis is made through a high-quality counter-based program profiler, such as `Quantify` [20]. The end product is timing ratios of functional elements of the method alongside a call graph. To a certain extent, timing ratios (not the absolute times) are a stable feature of a software system despite rapidly evolving hardware (in terms of clock speed) on which that system might be implemented.

The second stage is to employ pre-written reusable software components to aid construction of the pipeline. A data-farm template makes available communication primitives, buffering modules, and a message-passing interface. (As our templates have built-in instrumentation, event tracing is subsequently supported to identify how dynamic effects alter an initial performance estimate.) The third and final stage is to slot the sequential code sections into the programmer's template with limited disturbance of the code integrity. Message data which must fit a generic structure are also identified at this stage. The template adds all parallel features by means of library calls. Employing library calls implies that an alternative compiler can be used (and indeed was) if the code is transferred to another processor. Alternative implementations of the library calls can be written for the new processor. A description of a software toolkit to further automate this process can be found in [21].

Each stage of our parallel pipeline is a data farm. Data farming requires a centralised process, the farmer, to distribute work to a number of active server processes, which are dubbed workers. Data decomposition is typically arranged in OF by some geometric de-

composition across the frame sequence. For a row-major memory organisation horizontal sub-image strips can be used extending across a cross-section in time. After a start-up phase of static work scheduling, load imbalance is subsequently avoided by demand-based scheduling provided the number of image strips significantly exceeds the number of processes. On medium-grained multicomputers nested loops involving time-consuming calculations in the inner loop are sought to achieve an effective parallelisation. [22] is a detailed study of data farming.

4 OF prototypes

4.1 Gradient-based routine

The LK algorithm assumes: that image-patch brightness is conserved over a small interval in time (normalised to a frame interlude) leading to the well-known optical-flow equation (OFE) [1]; and that the spatial gradient is also constant. The linear least-squares error is found for an over-determined set of equations resulting from these assumptions for each pixel from weighted values taken across a neighbourhood patch. As the LK routine relies on numerical differentiation, it is essential first to remove high-frequency components, which would otherwise amplify any high-frequency components. Typically, Gaussian smoothing across 5×5 spatial neighbourhoods as well as temporal smoothing across fifteen frames is performed.

The processing cycle for the LK algorithm is summarized in Fig. 1. All processing is with real-valued numbers. Stage one of data processing is Gaussian temporal-spatial filtering, with a reduction in per-frame computational complexity from $O(N^2m^2T)$ to $O(N^2(T+2m))$ multiply/accumulates (MACs) through separability, where N is the size of a square image, T is the filter's temporal extent, and m is the spatial region of support. Initially, the temporal filter (larger than shown) is applied, centred on a middle target frame. Spatial filtering can then take place on that target frame. Stage two in Fig. 1 is numerical differentiation which is applied to the frames output from stage one. A tagged array allows frames to be

manipulated without regard to storage order in a circular (finite) buffer. Differentiation is separately applied in each dimension to a stage two target frame, giving rise to three images, Fig. 1, with complexity $O(N^2m)$ MACs. For each pixel in a target frame, a normalised 2×2 measurement matrix is formed from a weighted 5×5 neighbourhood. A weighted 1×2 time measurement matrix is likewise formed. Per-frame computational complexity in this stage is at least $O(N^2m^2(M^2+B))$ MACs where M is the size of the spatial measurement matrix and B is the number of columns in the time matrix. Finally in stage four, the eigenvectors of the spatial measurement matrix are first calculated in order to determine confidence thresholds for acceptance of velocity vectors. The complexity is $O(N^2M^3)$ multiplications. Calculation of full velocity vectors for each pixel is by matrix multiplication of the spatial measurement matrix and the time measurement matrix.

The application structure is revealed through a call graph output by `Quantify`, Fig. 2 showing the top twenty calls stemming from the `main` function. The graph can be expanded, and further detail still is available from an annotated listing. A function list will show which functions are candidates for optimization, but more helpful to the parallel programmer is the function detail display showing the percentage call time taken up by a function and its descendants. 47% of time is taken up in successively calculating derivatives and velocities, for an image of size 252×316 pixels. Of the remaining functions, 29% of time is taken up by a candidate for data-farming, Gaussian smoothing. About 5% variation in this ratio occurs if a smaller image is employed, because the ratios also include load time.[4]

To load balance, the same number of processors were used in each stage of the pipeline, Fig. 3. Because of the size of each inter-stage message (at least 57 Kbytes for the 252×316 frame sequence) the disparity in processing time between the two stages is taken up by message transfer time. This is convenient as the size of buffers would be prohibitive on the machine available.

Data farming of image strips in the first stage of the pipeline was straightforward, requiring

[4]System call time can alternatively be turned off. A number of other variable overheads are ignored such as paging and window traps where present.

limited modification of the data-farm template. Demand-based scheduling in stage one will result in out-of-order row arrivals to stage two. If the two pipeline stages can be decoupled the LK algorithm becomes an asynchronous pipeline with potential gains in frame traversal latency. In the algorithm for input to stage two, Fig. 4, a linked-list data structure enables a message to be assembled in the correct order ready to transmit to a stage two worker. The linked list enables storage nodes to be discarded once a message is assembled. The messages are not immediately farmed out but are transferred to a circular message buffer to preserve order. Worker requests from farm two are 'fairly' demultiplexed, though in alternation with the servicing of stage one arrivals.

To combine numerical differentiation and calculation of output motion vectors in the second stage an intermediate data structure was needed. In Fig. 5, the input message, sufficient in extent for 3D numerical differentiation, is transformed to an intermediate set of image strips, now reduced in size and ready for calculation of measurement matrices. Border regions are extended by zeroisation rather than using wrap-around. Edge velocities are discarded because of possible sensor error. Seven rows across five frames were needed to perform four-point weighted central differencing[5] for any one output row. A linear least-squares estimate was made across five rows of the differentiated data, finally resulting in a single row of output.[6]

4.2 Multiresolution routine

The AN method, again assuming brightness conservation, correlates image patches across two frames through a coarse-to-fine strategy. In forming Laplacian pyramids (one for each frame), band-pass smoothing is performed but no temporal smoothing occurs. At the coarsest level, direct neighbourhood correlation takes place but at higher levels four parent estimates of the flow vector, when suitably projected, are employed to guide the search at each pixel site. The overlapped search avoids one erroneous estimate passing up several levels of the tree.

[5] See [14] on the accuracy of this numerical differentiation method.

[6] Partial velocity is velocity in the normal direction, perpendicular to a patch edge; [1] raw velocity is before thresholding from the eigenvector spread; and the residual is an error term from the OFE [6].

Within the neighbourhood the search is exhaustive, unlike corresponding motion estimation algorithms in image compression. The minimum of the estimates, using a sum-of-squared-differences (SSD) measure for ease of computation, forms a correlation or error surface. The principal curvatures of the surface along with the amplitude at each pixel site are the basis of a confidence measure. The confidence measure guides a global relaxation which propagates firm to weak estimates. The Gauss-Seidel relaxation method was used in this version of the AN routine.

The call graph output by `Quantify` for the AN method, Fig. 6, shows that there is just one main processing stream (through `compute-flow`, shown qualitatively by the width of the connections). For small pixel motions, a two step resolution pyramid is needed. Construction of a two-step Laplacian pyramid is a relatively small overhead so parallelisation was not implemented even though spatial smoothing is often parallelised. The coarse flow estimate formed a first data-farm stage. As the subsequent relaxation phase is over a decimated image, it is short-lived. Therefore, for ease of implementation, the second matching phase was also performed on the same data farm, Fig. 7. Finally, the fine level relaxation could be performed on a single processor pipeline stage.

The AN parallelisation utilised two vertically-sliced image pyramids to enable data farming for the correlation search. Each row of flow vectors has four parents originating in two successive rows at a coarser level. In Fig. 8, the four parent estimate sources for one particular fine-level pixel site are shown. An example projected flow vector (labeled 'a') for just one of the parents is included in Fig. 8. Projected vectors, previously adjusted for sub-pixel accuracy, have a maximum extension of three pixels in either direction. The fine-level search involves a further maximum one pixel extent offset from the projected flow vector, for example the vector labeled 'b' in Fig. 8. At each potential match point a 3×3 convolution patch was formed, for example the dashed-line rectangle in Fig. 8. At each search point, the convolution patch, which was previously smoothed by a Guassian filter, was correlated with a fixed correlation area in the second image; in the example, shown outlined by a bold line in Fig. 8. In a two-level pyramid, with a 3×3 search to source one row of flow vectors, two rows at the

coarse level were sent and eleven rows from each of the two image frames were needed at the fine level (Fig. 8 shows part of this strip for one image). Indexing was required to send the correct two coarse-level rows in any message as parity differed with the fine-level row index.

For pyramids greater than two levels and larger images, the data-farm processing can be replicated across several pipeline stages. The data transfer burden was reduced over the LK algorithm, making the AN algorithm a more efficient parallelisation. However, its accuracy over small motions was limited due to the quantised nature of the search. Though correlation is normally used if temporal differentiation is not possible because of lack of frames, paradoxically an improvement may be to pre-process with temporal smoothing if such frames were available.

4.3 Phase-based routine

The FJ method consists of multiple stages of processing, Fig. 9, before full velocity estimates are reached.[7] Timing tests on the sequential version revealed that 80% of computation time is taken up by filtering with Gabor pseudo-quadrature filters (to extract phase), which therefore formed the basis of the data-farming stage. Gabor filters are designed in 3D frequency space to match the features of interest but are implemented in the spatiotemporal domain.[8] Separate filters are needed for each spatial orientation, speed, and scale. Greater resolution is possible for filters corresponding to higher frequencies. Despite the use of the Heeger algorithm [15] based on a trigonometric identity for making separable 3D Gabor filtering, multiple passes through the data results in a computational load which is significantly greater than other methods implemented. If each of orientation, speed, and scale were to be sampled n times, with a region of support m constant in all dimensions with separable filters, and square image size N, and T frames then the time complexity is $O(n^3 m N^2 T)$ multiplications per frame.

[7] Phase component and full velocities are alternative estimates. The component estimate requires differentiating the phase.

[8] In frequency space, the amplitude spectrum of an image patch moving at a constant velocity has the normal component of this velocity spectrum concentrated in a line around the origin, which feature is matched by a Gabor filter.

It will be seen from Fig. 10 that there is also a significant data storage problem as typically twenty-two sets (one for each filter) of five frames each are needed after filtering. Five frames are needed later to perform numerical differentiation which is a prelude to finding the phase. Had a greater range of velocities been expected then further sets of filters would be needed.

The different filter estimates enable the detection of multiple motions in the same locality as might occur at occlusion boundaries, through transparency, shadowing, and virtual motion arising from specularities (highlights). However, clustering of velocities at a single location was not performed. If one row is farmed out at any one time, as implemented, then parallel processing can proceed as far as component calculation of the OF vectors.[9] If five rows were to be processed at any one time then all processing can be performed in parallel without the need for the temporary storage of intermediate files, but with a corresponding increase in data communication.

5 Analysis of results

Experiments were made on an eight-module MIMD Transtech Paramid [23] with a distributed memory organization. Communication between processors is by message passing. Each module consists of an i860 superscalar processor, with a five-stage instruction pipeline and possible four-way vectorisation, giving a manufacturer's peak of 200 MFLOP/s at 50 MHz. The i860 is serviced via overlapped-memory by a transputer communication coprocessor, with raw per-link bandwidth of 20 Mbps (and valency four). The interconnect between modules can be software reconfigured at compile time through an electronic switch. A virtual channel system is present as a micro-kernel on each module: messages are transparently packetized and routed by link sentinel processes. Architectural comparisons between this machine using an i860 and a DEC alpha solution [24] indicate that the key issue is the memory hierarchy and not primarily the processor speed.

[9] Only the normal component of phase, i.e. the component normal to the phase contour not to an image edge, is unambiguously resolved.

5.1 The LK results

Initial work concentrated on the LK method because in [11] the method is reported to be amongst the most accurate, though for image sequences without many of the problems listed in Section 2.3. Of the standard synthetic sequences, when the error is measurable, the Yosemite valley scene is usually selected as most representative of natural images [13]. First tests took place on a single i860. We wondered what would be the effect of relaxing the accuracy. Table 1 shows, for frame 9 of the sequence, no advantage in accuracy is to be gained from temporal smoothing over 15 frames and hence these results are not recorded, but dropping below 11 frames introduces significant error.[10] It appears that the large temporal extent is needed which brings a communication overhead on parallel machines. Notice that a general processing model that includes OF must account for non-causal filtering [25], (i.e. sufficient previous frames must be available).

Some researchers [26, 27] have not used temporal smoothing arguing that it is unnecessary for small pixel motions. Table 1 shows the error when no temporal smoothing was applied. The final row, parameter = 0.0, records the effect of turning off spatial smoothing as well.

Paradoxically, the timings on the single i860, which exclude file I/O, record lower times for apparently more processing. When the smoothing loop is switched off transfer of spatial results to the target array still occurs implying that a memory interaction was responsible for the delay. In fact, in all sequential runs a swap file was stored in the host machine's local disc requiring transfers over a SCSI link. For filter parameter 1.5 a SPARCstation 1+ takes 154.9s and 60.3 s with compiler optimisation.

Timings were taken, without I/O, for the LK method running with two active processors on each of the two pipeline stages. The message size was varied between pipeline stages by means of the data structures described in Section 4.1. The data size needed for each stage of the LK routine, Table 2, is somewhat different even if the number of image lines

[10] For fine work (not this study), [16] has found that the angular accuracy measure, which is non-linear with angle size [9], can return asymmetric results. We found a small discrepancy with tests on a SPARC machine resulting from a divergence from the IEEE floating point specification to gain speed on the i860.

remained the same. However, the optimum times occurred, Table 3, when the message lines size were balanced with little effect from changing between fifteen frames, $\sigma = 1.5$, and eleven frames, $\sigma = 0.5$. However, processing was not balanced between the two stages as only one of the processors in stage two was in constant use. This result was quickly determined by examining the event trace. The almost linear speed-up, compared to Table 1 timings, for four active processors is misleading as reduction in per-processor program size meant that no intermediate disc access was required. Importantly, increasing the number of processors in each pipeline stage did not result in any improvement in times, because the message-passing overhead escalated.

5.2 The other methods

Table 4 records timings for the AN method as additional stages of processing were incrementally introduced into a single farm pipeline. Considerable savings over workstation timings are possible (see also Section 2.4). The two fixed overhead timings, from relaxation and from construction of the Laplacian pyramid, are also recorded. Pyramid construction precedes the parallel stage and in an implementation would be linked to I/O (not included as I/O is a system dependent overhead) on a single processor. The relaxation routine timing represents processing after the first and second stages of the coarse-to-fine processing, but the processing after the second stage dominates. The timings suggest that a balanced pipeline is possible for the AN method.

FJ timings, Table 5, were taken for a full run on the workstation and for the parallelized Gabor filtering stage on the Paramid machine. However, the Paramid system proved inadequate to the task of processing larger images, although remaining timings are unaffected by swap-file activity. From the timings taken, which indicate scaling, given an effective memory sub-system, using a parallel machine would pay off if simply to reduce the inordinately lengthy computation time on a workstation.

5.3 Evaluation

Prototyping the three methods allowed a tentative evaluation to be reached.

Other considerations being equal, the LK method is appealing to implement because it has the shortest processing time and because the complete structure can be parallelised. Given the regular nature of the processing, there is also the possibility of transfer to VLSI. However, further reductions in speed could not be achieved on our general-purpose machine because of the data-transfer cost. The peak speed-up was 3.69 for 4 active processors processing a 252×316 image in 10.7s with full accuracy. If detection of larger pixel motions were to be required either a larger local patch would be needed, which would add to the data communication burden, or a multiresolution pyramid would need to be constructed. The correct method of extending the OF field to 100% density is not established. These difficulties are probably resolvable but would reduce the speed advantage that the LK method holds.

Correlation methods, especially without a search guided through an image pyramid, are prone to distortion from multi-modal results. There is also a question mark over how to best achieve sub-pixel accuracy. Conversely, correlation has a higher immunity to noise than methods involving numerical differentiation. However, because of the reduction in data transfer this class of algorithm is worth persevering with. The per-row computational complexity of the AN method is $O(d^4)$ while message sizes are $O(d)$ implying that increasing the window size, d, will quickly bring a benign parallel regime. The peak speed-up for one stage using four processors was 2.95 at 65.2s/frame for the AN method.

The FJ method is presently simply too burdensome to consider in a production environment, probably being most suitable for research using small-scale images. Another difficulty are low densities reported as being below 50% [11]. Parallelisation has some value in reducing job throughput (reducing the time for a 150×150 pixel image from 339.7s to 114.9s, a speed-up of 3.1 for 4 processors). It is not clear that simplified filters will significantly reduce the computation and storage overhead [28]. [29] advocate locating orientation by finding the eigenstructure of just one scatter matrix for a local patch in space-time (by analogy with

forming an inertia tensor). However, the method shares a similar computational structure to the LK method. A link between the detection of spectral energy (and possibly phase) and spatial differentiation, via the Fourier derivative theorem has long been noticed [30].

The data intensive nature of OF methods causes many of the processing problems, as fast I/O is no longer available as it was on bit-serial SIMD machines. A data farm can in some circumstances be run without physical transfer of data between farmer and workers, whereby the farmer acts as a work scheduler and tokens are passed if work is completed. Local I/O is sometimes available on general-purpose multicomputers but as a costly alternative, which is a deterrent to research budgets. One idea [31] is to provide VRAM on one edge of an array of medium-grained processors in a manner reminiscent of earlier SIMD arrays. Local image data is passed along each row of the array. This arrangement may be adaptable to a pipeline structure.

6 Summary

OF methods represent a form of motion detection potentially applicable to a wide range of imagery. Unfortunately, processing rates if full accuracy is preserved are far from video rate. This is the more so if realistic image sizes are considered. A further problem is that recent algorithmic implementations entail significant data communication and storage of intermediate results. In our implementations which were on a modestly parallel machine, the OF pipelines achieved speed-ups which justified the parallelisation. In one case speed-up arose from decomposition of the algorithm and would not scale. Most gains came from parallelizing correlation methods. Therefore, parallelisation of OF methods by means of a parallel pipeline, the purpose of this study, is in all cases worth considering as it will speed-up subsequent development work. Future work on OF may focus on using heterogeneous hardware within the prototyping pipeline.

Acknowledgement

This work was carried out under EPSRC research contract GR/K40277 'Portable software tools for embedded signal processing applications' as part of the EPSRC Portable Software Tools for Parallel Architectures directed programme.

References

[1] B. K. P. Horn and B. G. Schunk, "Determining optical flow," *Artificial Intelligence*, vol. 17, pp. 185–203, 1981.

[2] Q. X. Wu, "A correlation-relaxation-labeling framework for computing optical flow - template matching from a new perspective," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, pp. 843–853, September 1995.

[3] P. Baraldi, A. Sarti, C. Lamberti, A. Prandini, and F. Sgalli, "Evaluation of differential optical flow techniques on synthesized echo images," *IEEE Transactions on Biomedical Engineering*, vol. 43, pp. 259–271, March 1996.

[4] J. L. Barron and A. Liptay, "Optic flow to detect minute increments in plant growth," *BioImaging*, vol. 2, pp. 57–61, 1994.

[5] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in $7^{\underline{th}}$ *International Joint Conference on Artificial Intelligence*, pp. 674–679, 1981.

[6] E. P. Simoncelli, E. H. Adelson, and D. J. Heeger, "Probability distributions of optical flow," in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 310–315, 1991.

[7] F. Glazer, G. Reynolds, and P. Anandan, "Scene matching by hierarchical correlation," in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 432–441, 1983.

[8] P. Anandan, "A computational framework and an algorithm for the measurement of visual motion," *International Journal of Computer Vision*, vol. 2, pp. 283–310, 1989.

[9] D. J. Fleet and A. D. Jepson, "Computation of component image velocity from local phase information," *International Journal of Computer Vision*, vol. 5, no. 1, pp. 77–104, 1990.

[10] D. J. Fleet, *Measurement of Image Velocity*. Kluwer Academic, Norwell, MA, 1992.

[11] J. L. Barron, D. J. Fleet, and S. S. Beauchemin, "Performance of optical flow techniques," *International Journal of Computer Vision*, vol. 12, no. 1, pp. 43–77, 1994.

[12] S. S. Beauchemin and J. L. Barron, "The computation of optical flow," *ACM Computing Surveys*, vol. 27, pp. 433–467, September 1995.

[13] H. Liu, T.-H. Hong, and M. Herman, "A general motion model and spatio-temporal filters for computing optical flow," *International Journal of Computer Vision*, vol. 22, no. 2, pp. 141–172, 1997.

[14] D. J. Fleet and K. Langley, "Recursive filters for optical flow," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, pp. 61–67, January 1995.

[15] D. J. Heeger, "Model for the extraction of image flow," *Journal of the Optical Society of America A*, vol. 4, pp. 1455–1470, August 1987.

[16] M. Otte and H. Nagel, "Optical flow estimation: Advances and comparisons," in *Computer Vision - ECCV'94* (J.-O. Elkundh, ed.), vol. 1, pp. 51–60, Springer, Berlin, 1994. Lecture Notes in Computer Science No. 800.

[17] E. P. Ong and M. Spann, "Robust multiresolution computation of optical flow," in *International Conference on Acoustics, Speech, and Signal Processing*, vol. 4, pp. 1938–1941, 1996.

[18] D. P. Elias and N. G. Kingsbury, "An efficient block segmentation algorithm for true motion segmentation," in $6^{\underline{th}}$ *International Conference on Image Processing and its Applications*, vol. 1, pp. 209–213, 1997. IEE Conference Publication No. 443.

[19] J. Weber and J. Malik, "Robust computation of optical flow in a multi-scale differential framework," in $4^{\underline{th}}$ *International Conference on Computer Vision*, pp. 12–18, 1993.

[20] Pure Software Inc., 1309 South Mary Ave., Sunnyval. CA., *Quantify User's Guide*, 1992.

[21] M. Fleury, N. Sarvan, A. C. Downton, and A. F. Clark, "Methodology and tools for system analysis of parallel pipelines," *Concurrency: Practice and Experience*, vol. 11, no. 11, pp. 655–670, 1999.

[22] A. S. Wagner, H. V. Skreekantaswamy, and S. T. Chanson, "Performance models for the processor farm paradigm," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, pp. 475–489, May 1997.

[23] Transtech Parallel Systems Ltd., 17-19 Manor Court Yard, Hughenden Ave., High Wycombe, Bucks., UK, *The Paramid User's Guide*, 1993.

[24] N. Sarvan, R. Durrant, M. Fleury, A. C. Downton, and A. F. Clark, "Analysis prediction template toolkit (aptt) for real-time image processing," in *IEE Image Processing and its Applications, IPA99*, vol. 1, pp. 116–121, 1999.

[25] S. E. Moore, J. Sztipanovitz, G. Karsai, and J. Nichols, "A model-integrated program synthesis environment for parallel/real-time image processing," in *Parallel and Distributed Methods for Image Processing*, pp. 31–41, 1997. SPIE Volume 3166.

[26] O. Tretiak and L. Pastor, "Velocity estimation from image sequences with second order differential operators," in *IEEE International Conference on Pattern Recognition*, vol. 1, pp. 16–19, 1984.

[27] A. Verri, F. Girosi, and V. Torre, "Differential techniques for optical flow," *Journal of the Optical Society of America A*, vol. 7, no. 5, pp. 912–922, 1990.

[28] D. J. Fleet and A. D. Jepson, "Hierarchical construction of orientation and velocity selective filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, pp. 315–325, March 1989.

[29] J. Bigün, G. H. Granlund, and J. Wiklund, "Multidimensional orientation estimation with applications to texture analysis and optical flow," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, pp. 775–790, August 1991.

[30] E. H. Adelson and J. R. Bergen, "The extraction of spatio-temporal energy in human and machine vision," in *IEEE Workshop on Motion: Representation and Analysis*, pp. 151–155, 1986.

[31] R. S. Cok and J. S. Gerstenberger, "A T9000-based parallel image processor," in *NATUG'93 Transputer Research and Applications 6* (S. Atkin and A. S. Wagner, eds.), pp. 142–153, IOS, Amsterdam, 1993.

Filter parameter	No. of frames	Angular error	Standard Deviation	Density %	Min. error	Max. error	Time (s)
1.5	15	4.26	10.14	39.79	0.00	111.62	39.53
1.0	11	5.28	10.89	54.07	0.00	121.04	36.27
0.5	9	18.74	23.09	71.90	0.02	140.23	35.02
Without temporal smoothing							
1.5	15	7.68	15.42	54.92	0.00	125.40	39.94
1.0	11	11.09	19.26	62.75	0.00	131.18	36.64
0.5	9	22.31	25.87	75.01	0.00	136.88	35.53
0.0	5	28.78	28.01	79.74	0.00	152.99	27.40

Table 1: LK Method: Angular Error (degrees) for Yosemite Sequence, frame 9, processing times for one processor

Lines per	Stage 1		Stage 2	
Message	Out	Return	Out	Return
1	52144	6324	56888	10140
2	56884	12644	63208	20240
3	61624	18964	69528	30364
4	66364	25284	78848	40476
5	71104	31604	82168	50588

Table 2: LK Method: Message Sizes (bytes) with Filter Parameter = 1.5, two stages 2 processors per stage

Stage one	Filter parameter $\sigma = 1.5$ 15 frames temp. smoothing Stage two					Filter parameter $\sigma = 0.5$ 11 frames temp. smoothing Stage two				
	1	2	3	4	5	1	2	3	4	5
1	18.05	17.42	17.66	17.47	17.55	18.69	12.30	11.41	12.04	12.78
2	18.02	12.12	12.66	12.33	13.40	18.78	12.47	11.36	10.72	11.90
3	17.88	12.56	10.70	11.36	11.91	18.50	12.70	10.94	11.26	11.58
4	17.92	12.10	11.22	10.12	11.61	18.82	12.70	11.36	10.62	11.25
5	17.01	12.56	11.50	11.19	13.87	17.76	12.55	10.85	11.24	13.07

Table 3: Per-frame LK Timings (s) with Various No. of Rows per Message for a 256×316 Sized Image., 2 stages, 2 processors per stage

	SPARC station 1+		No. of Worker i860s								
Size	`gcc`	opt. 3	1	2(a)	2(b)	2(c)	4(a)	4(b)	4(c)	4(d)	4(e)
100×100	171.3	103.6	23.5	15.2	13.2	12.3	12.2	9.4	8.2	3.8	0.3
150×150	458.8	282.5	60.0	34.3	29.7	27.8	27.5	20.9	18.2	8.4	0.8
252×316	1445.5	895.3	192.2	122.7	107.2	100.1	98.1	75.6	65.2	29.8	2.9

Table 4: Per-frame AN Method Timings (s) Parallelizing: (a) flow vectors (b) flow vectors and confidence measures (c) with coarse-level calculations, and Sequential overheads: (d) relaxation (e) pyramid construction.

	SPARCstation 1+		Worker i860s		
Size	`gcc`	opt. 3	1	2	4
100×100	834.1	257.9	152.7	105.8	51.4
150×150	2476.7	775.3	353.2	165.5	114.9
252×316	10500.1	3392.6	-	-	-

Table 5: Per-frame FJ Method Timings (s)

List of Figures

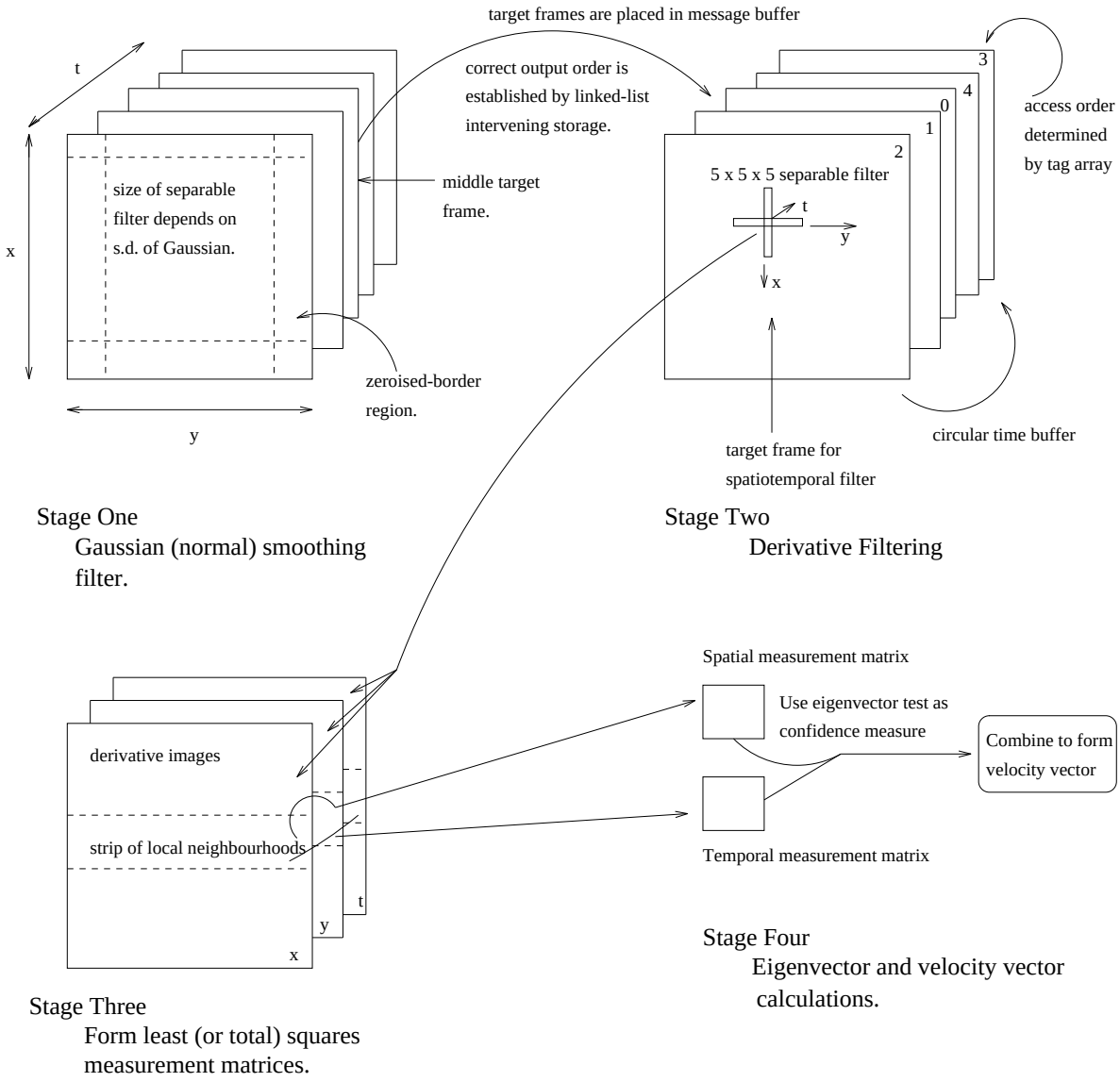

Figure 1: Data Processing in the LK Method

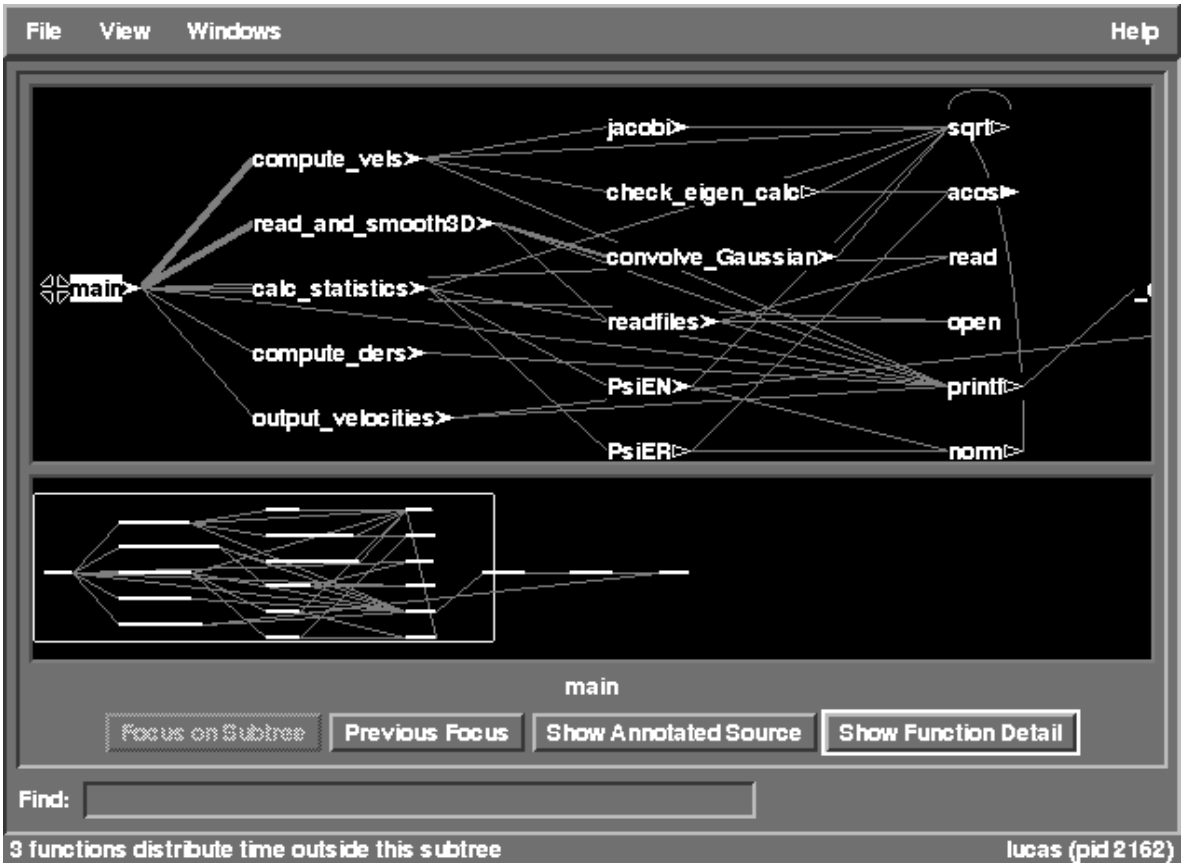

Figure 2: Call Graph for the LK Sequential Code

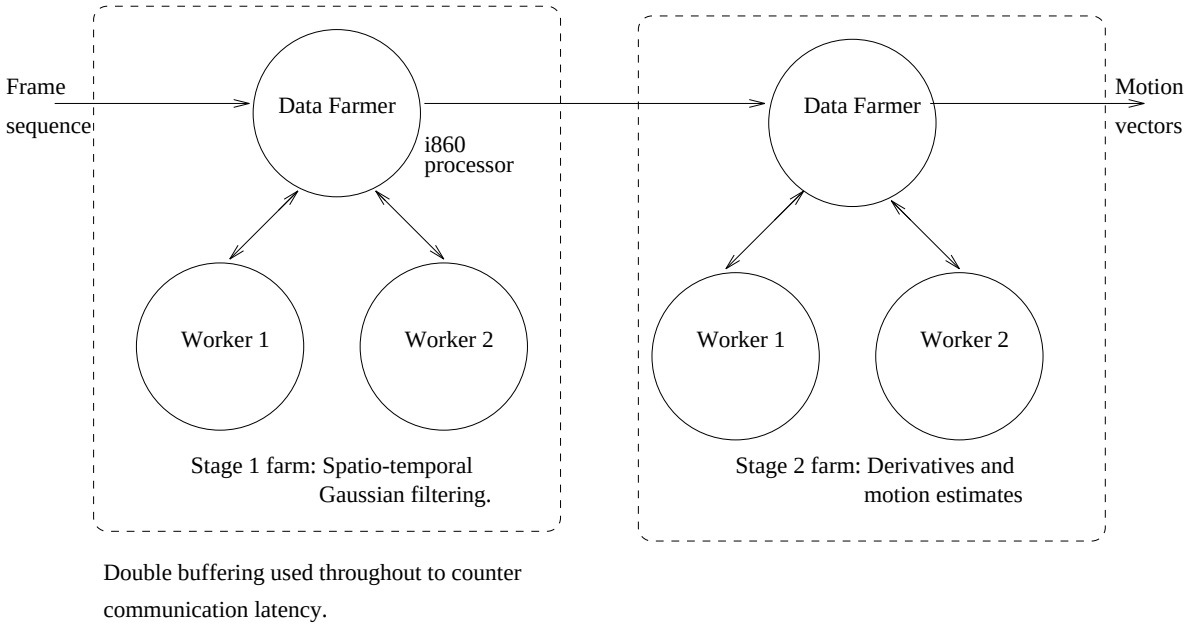

Figure 3: LK Farm Layout

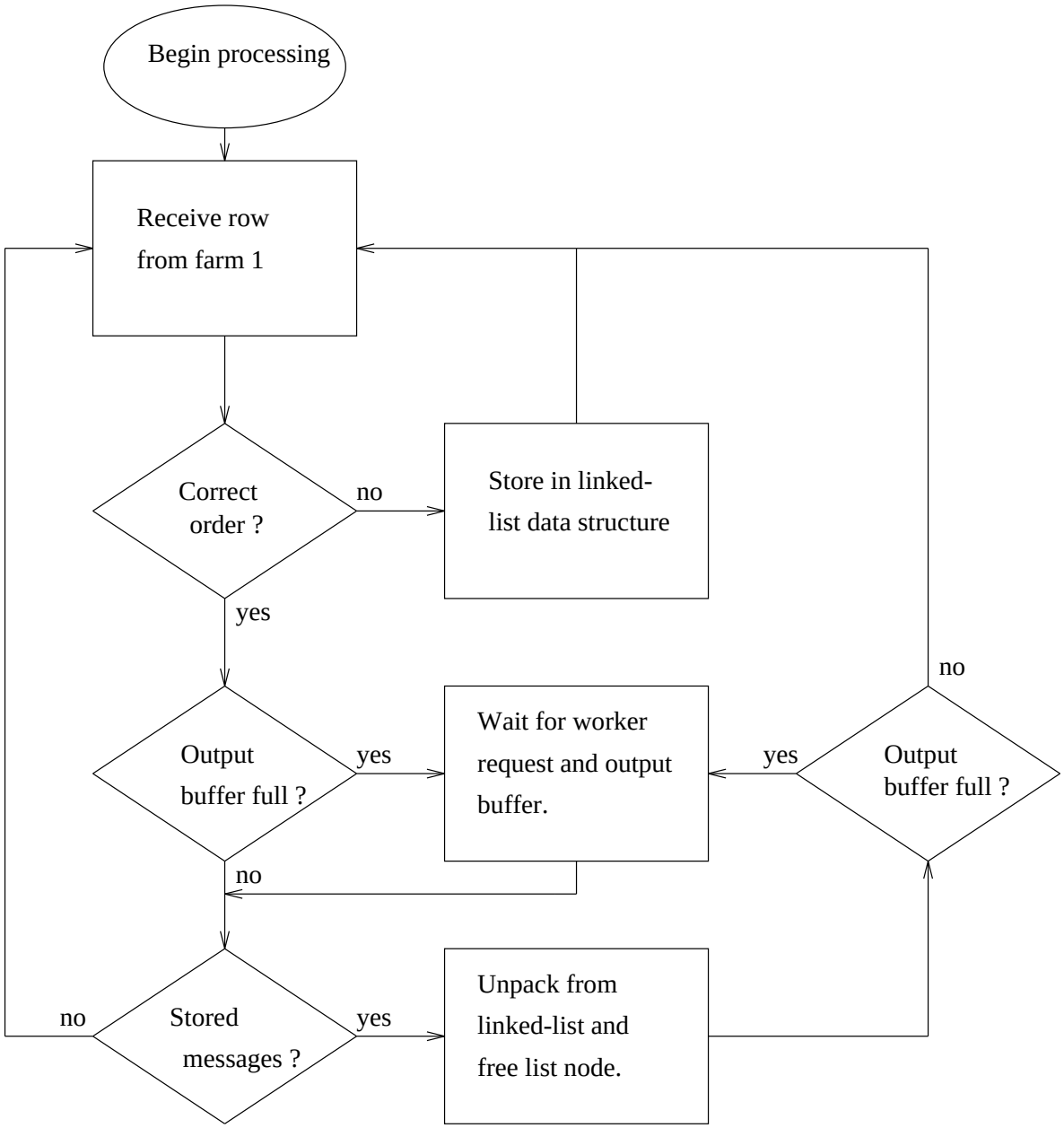

Figure 4: Buffering Out-of-Order Arrivals

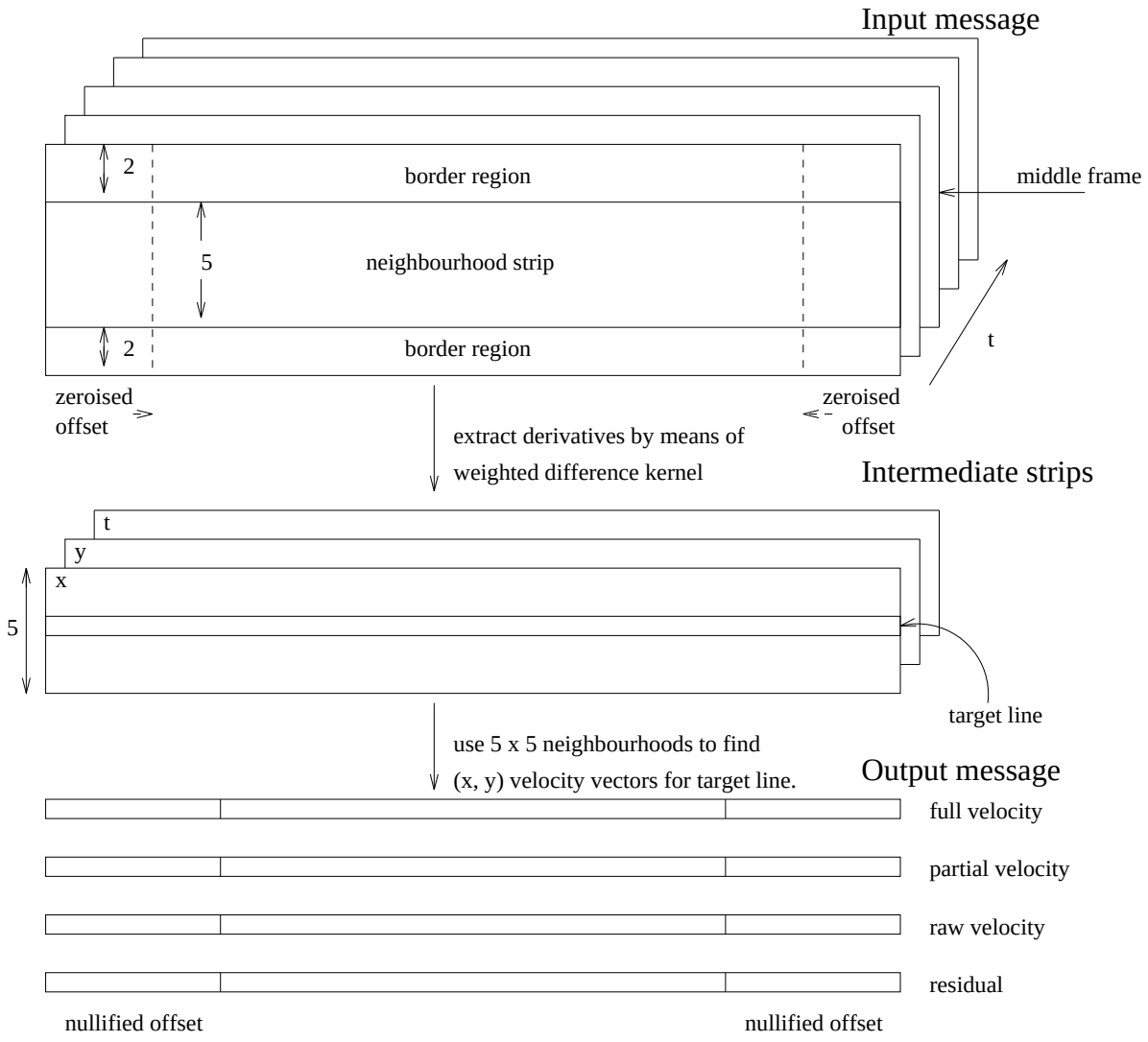

Figure 5: Pipeline Second-Stage Message Structure

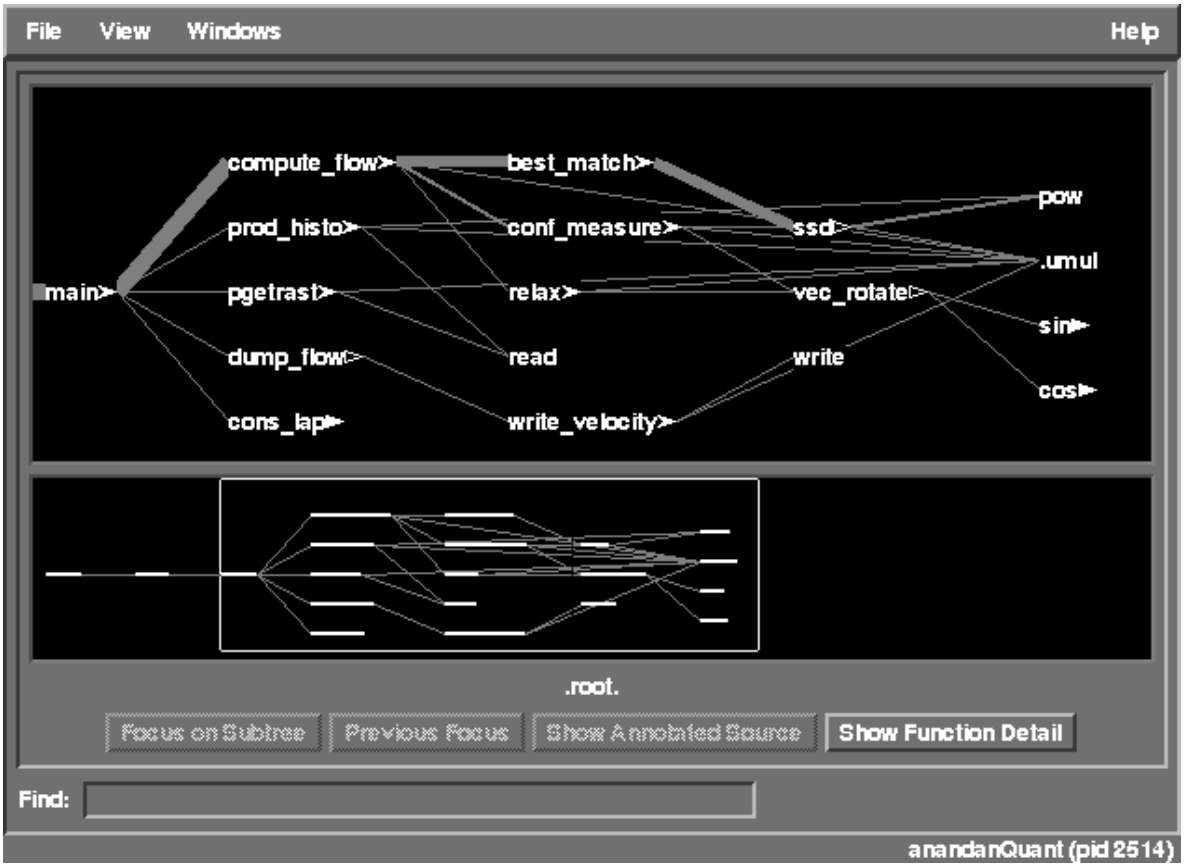

Figure 6: Call Graph for the AN Sequential Code

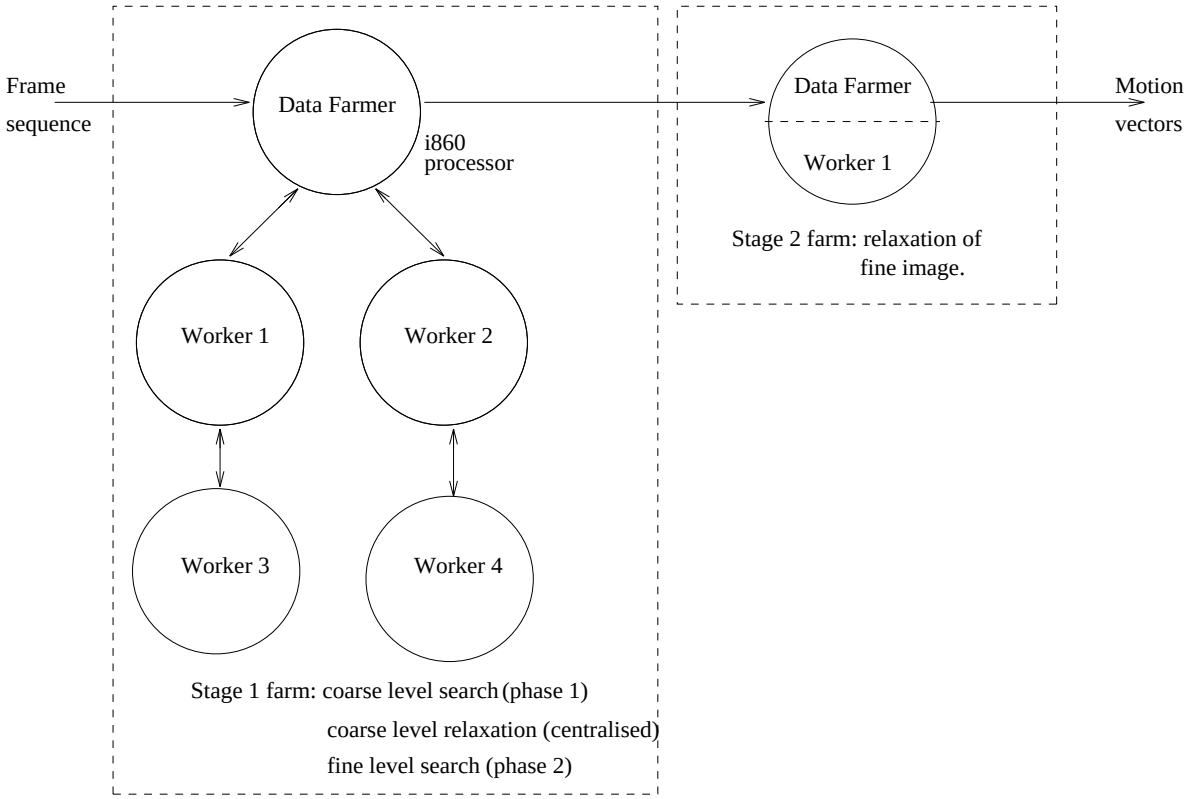

Figure 7: AN Farm Layout

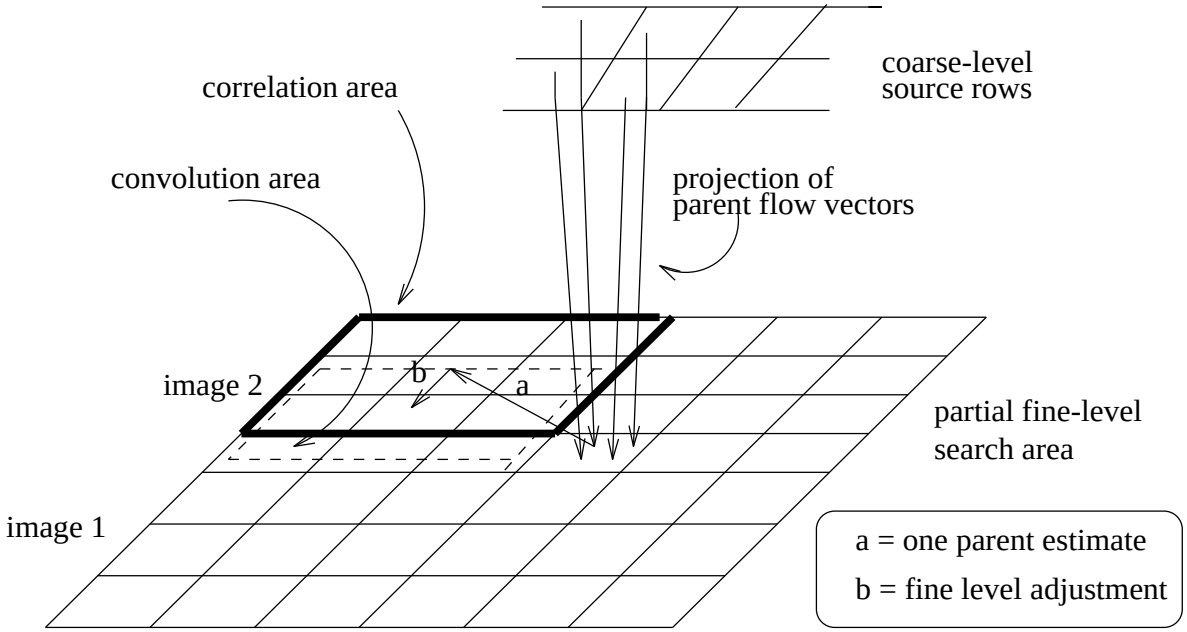

Figure 8: Example AN Search Area

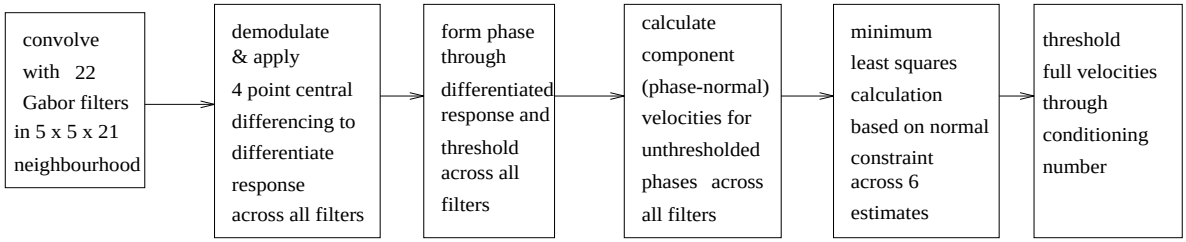

Figure 9: FJ Processing Pipeline

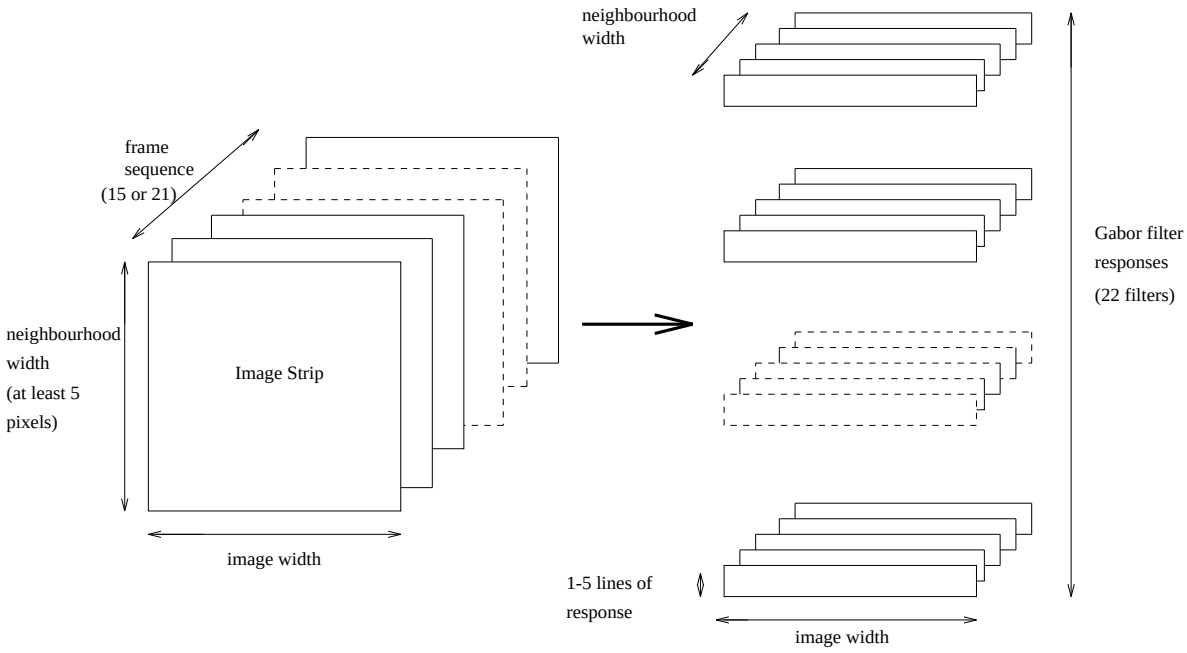

Figure 10: Simplified FJ Filtering Data Structure