Computational Architectures for Responsive Vision: the Vision Engine

James J. Little, Rod Barman,
Stewart Kingdon and Jiping Lu

Technical Report 91-25

November 1991

Laboratory for Computational Vision
Department of Computer Science
University of British Columbia
Vancouver, British Columbia, CANADA V6T 1W5
email: little@cs.ubc.ca

Abstract

To respond actively to a dynamic environment, a vision system must process perceptual data in real time, and in multiple modalities. The structure of the computational load varies across the levels of vision, requiring multiple architectures. We describe the Vision Engine, a system with a pipelined early vision architecture, Datacube image processors, connected to a MIMD intermediate vision system, a set of Transputers. The system uses a controllable eye/head for tasks involving motion, stereo and tracking.

A simple pipeline model describes image transformation through multiple functional stages in early vision. Later processing (e.g., segmentation, edge linking, perceptual organization) cannot easily proceed on a pipeline architecture. A MIMD architecture is more appropriate for the irregular data and functional parallelism of later visual processing.

The Vision Engine is designed for general vision tasks. Early vision processing, both optical flow and stereo, is implemented in near real-time using the Datacube, producing dense vector fields with confidence measures, transferred at near video rates to the Transputer subsystem. We describe a simple implementation combining, in the Transputer system, stereo and motion information from the Datacube.[1]

[1]This research was supported by the Natural Sciences and Engineering Research Council of Canada and the Networks of Centres of Excellence Institute for Robotics and Intelligent Systems, Project A-1.

1 Introduction

Responsive vision is vision responding to the environment: a response is "an answer, an action, stimulus, movement, or change elicited by a stimulus or influence", responsive is "reacting readily or favourably". The characteristics of our responsive system are: active response in dynamic environment, real time computation, and using multiple modalities in a multi-purpose system.

The system is designed to be general purpose and support computation for all levels of vision. The levels of vision processing include early vision, which is spatially homogeneous, involving dense processing, such as filtering; middle vision, which is spatially distributed, regular but sparse, such as line following, aggregation; and late vision, which is symbolic, such as matching[LBC89, TFF88].

Our system, the Vision Engine, consists of multiple architectures, each commonly available. They include pipelined: Datacube MaxVideo-20 image processor with a Digicolor color image digitizer, a MIMD multicomputer: 16 T800 Transputers operating at 25 MHz, each with 2 MB memory, with programmable interconnections through a crossbar. These subsystems are connected by a Maxtran board, a bidirectional interface[2] operating at video rates. The Maxtran board maps data from the video bus of the Datacube system into video RAM attached to a Transputer, whence image data can be partitioned and sent to other processors in the system. The effective data rate on the video bus is 7.03MB/s which matches well the data rate of 6.72MB/s across the four unidirectional links exiting the Transputer on the Maxtran. The communication is bidirectional so that data can be returned to the Datacube system for further processing or display. The connections between the Maxtran and the Transputer subsystem pass through a crossbar so that the data can be sent to independent Transputers for each module. Figure 1 depicts the organization of our system. The entire system resides on the VME bus connected to a Sun SparcStation 2 host. The system controls a CRS-460 six degree-of-freedom robot arm and other actuators such as a pan/tilt platform from either a Transputer (specialized for I/O) or from the Sun host. As well, the Transputer system has direct image digitizing and image display. System tasks include model-based tracking [Low90] and robotic control.

2 Multistage Vision

In our system, data flows through stages of early processing such as convolution or edge-detection, then communication from Datacube to Transputer, then middle processing,

[2]Built by Microsystem Services Ltd., from a design by British Aerospace.

e.g., grouping, and, finally, late processing: model matching. The last two stages occur on the Transputer subsystem.

These two architectures enforce very different processing and communication disciplines. The image processors form a pipeline of processing elements, through which images flow and new images are created. Each image processing module processes data independently, performing one complete function, such as convolution or table lookup. Synchronization occurs on complete images, either fields or frames.

The Transputer subsystem allows asynchronous computation with message-passing. Data dependent processing allows different operations to be applied to different elements depending on its value or its context. The Transputer system gives us increased flexibility in data and task distribution.

2.1 Process Models

A variety of models of parallelism are applicable; we survey several and indicate their relevance to our architecture. Functional parallelism allocates separate computational functions to separate processors. This is the natural model for the pipeline architecture where the processors are specialized for a particular function. Data parallelism [HS86] distributes collections of data over processors; it capitalizes on the fact that in many applications large amounts of data pass through only a few processing operations. Systolic processing is attractive both as an architecture [Kun82] and as a processing model [Sha87], since it combines elements of both functional and data parallelism: there are multiple processing elements performing each function.

The multicomputer architecture can support each of these modes of parallelism, and permits the multiple levels of vision to operate simultaneously; any sensing to control system must have all functional stages active in parallel, and so must contain functional parallelism. Communication in our system is organized by data flowing through multiple computational stages. Each node in the graph generated by the data flow corresponds to an independent component.

It is useful to treat data structures are distributed objects, aggregates, produced and destroyed as data flows through the architecture. Aggregates can both act as a single element at one level of computation, and as multiple objects at a finer level. The vector model [Ble89] and Paralation model [Sab88] are instances of this type of simplified model for parallel programming. The elegance of the expression of programs is balanced by the care needed to implement such models on coarse-grain machines. Aggregate models hide details of data distribution and communication, which makes implementation simpler, at the cost of developing special interfaces. Template-based programming [AWG91] is an example of such an interface: it implements common pro-

gramming paradigms, e.g., divide-and conquer, efficiently on multicomputers, hiding data movement. Each worker process need only be programmed to handle a small section of the data – the communication tasks are handled by the "template", which passes the data to the workers and retrieves the sub-results that are then joined into the final result. Similarly, we can implement templates for the data distribution methods we describe later. Dataflow models have also been used in vision programming. Shapiro [SHG87, Sha89] developed the INSIGHT language for describing vision computation directly

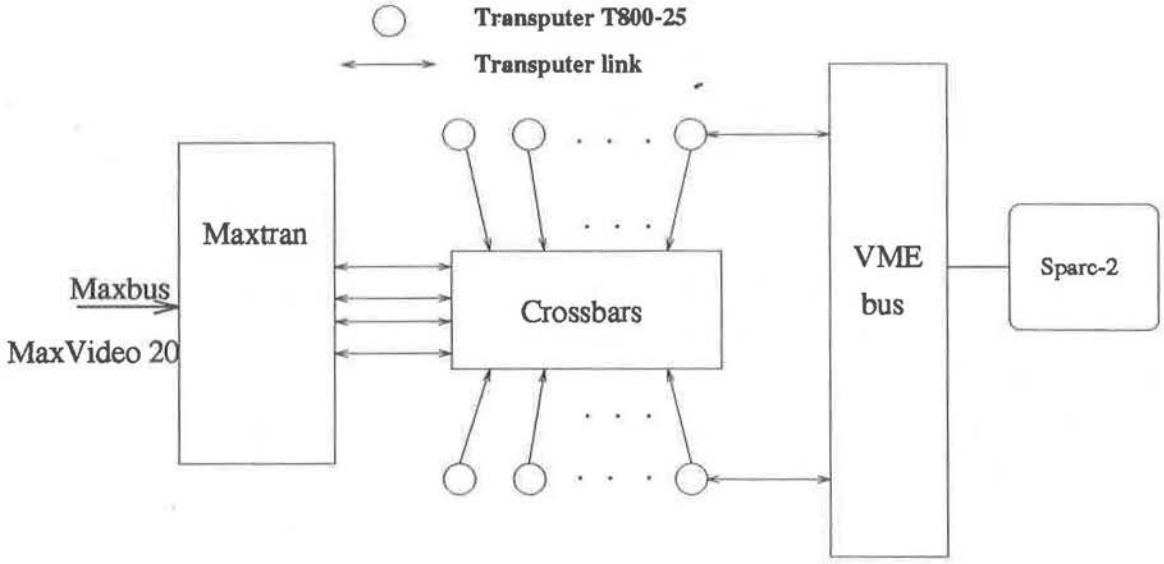
In order to be responsive, each component of our vision system needs to be able to provide results to successor components, at any time. The interface between successive layers must provide for this ability, either by tailoring the algorithm to this need, or by explicit storage of the results of the previous step. This kind of interaction is common in robotic applications. Computations that can be interrupted any time and provide approximate results have been termed "anytime", "interruptible" [Pai89] or "imprecise"[Liu91]; solution by iterative methods, often used in early vision, provide just such "anytime" results.

3 Mapping Vision onto the Engine

Allocation of tasks onto components of the Vision Engine mirrors the categorization of vision processing as early, middle and late. Early processes are mapped, when possible, to the Datacube subsystem; others are put on the Transputer subsystem. In the Transputer subsystem, the organization of processing depends critically on the computation/communication ratio of the task load and of the subsystem. When communication time for a subtask description dominates the subtask computation time, direct solution rather than task sharing becomes preferable. This issue becomes complex because Transputers can overlap communication and computation. Moreover, the volume of data decreases markedly over the levels, so that much less data is transmitted to high-level processes.

Early vision processes involve many data-independent operations, but others, such as Markov Random field computations[PGL88], may require data dependent operation. Any data dependent computation can be mapped into a data independent model, however, the result of such a mapping may be inefficient. For example, full simulation of a data-dependent operation can be implemented at each processor in a Connection Machine[Hil85]. The pipelined component of our Vision Engine evaluates a suitable fraction of the tedious filtering operations before data enter the Transputer subsystem. However, any data dependent operations can only be implemented on the Datacube by

multiple passes. Therefore, all data dependent processing is handled in the Transputer subsystem.



A cube of transputers for intermediate vision

Figure 1: Schematic of Vision Engine

The distribution of data varies, depending on the type of computation. The crucial element is whether computation at a point in the image (or any distributed structure) is supported by elements at bounded distance or unbounded distance. The Apply language [KW86] is an example of a language for image processing that hides details of data partitioning from the programmer. Later, Webb extended the Apply to the Adapt language to permit more global operations [Web90].

For iterative schemes, with no inherent limit on the distance, direct subdivision, with no overlap from neighboring images, is effective. During each iteration, adjacent sections of the image communicate border elements. Provided the communication time for the border elements is small relative to the computation time for the interior, a data partitioning scheme is well suited for a coarse grain system such as our Transputer subsystem. The ratio of border size to interior is what limits the granularity. There is a tradeoff – when communication is greater than computation, no further subdivision is needed. For computation with bounded support, such as filtering and simple edge detection, data can be partitioned, copying overlapping elements. During each iteration, then, no interprocessor communication is required. This method was used to implement the mean-field method for edge detection on the Transputer subsystem

[Sik90].

Early vision modules create many maps of image-form data, depth motion, color, edges; these compete with communication buffers for the available memory. But, as data flow through the levels of vision, data reduce in volume. Middle vision often transforms two-dimensional data into more compact representations, such as boundary curves. This both reduces the amount of data and removes the data from the image format. This compression destroys the regularity of the data mapping to processors. Data distribution and load-balancing of irregular structures for middle and late vision can become problematic in this approach.

Partial load balancing can be achieved by allocating a process master within each processor, usurping the function of the scheduler. It can partition the data into small cells, processing each cell in turn. When the master at a neighboring processor becomes idle, it can request data from the master, which will return the data provided the communication overhead does not make this inefficient. The cell size must be determined so that the decision to hand over data is simple.

For late vision, matching models can use replicated data, multiple copies of the model, as used in SIMD vision [LBC89, TFF88]. This method absorbs the reduction of data from the image, but replicates the information from the model.

4 An Example

We implement a complex, integrated system not only as an demonstration of our Vision Engine in action, but also as an example of the core of a responsive vision system, which needs dense results, but perhaps not detailed results, to provide quick response to changing situations. Coverage by the vision system should be broad – one function of a vision system is to monitor the environment for threats, albeit as simple as an obstacle during locomotion. Moreover, it should be able to support higher functions like recognition and so should provide general, not task-based results.

The system is designed so that each component operates independently: when algorithms are implemented so that they can provide results "anytime", they interface easily with control systems. Robustness can be maintained even though shared state is produced; this allows the system to derive complex structures to facilitate recognition, in contrast to the simple parallel sensing-to-action stages in the subsumption architecture [Bro87].

We demonstrate a multistage vision system utilizing

- SAD optical flow on the Datacube
- SAD stereo on the Datacube

- edge detection on the Datacube
- data fusion on the Transputers

The example uses several paths from image to output. All modules use bandpass filtering: for edge detection, it is central; optical flow[BLP89], which minimizes the sum of absolute values of differences (SAD) and SAD stereo[DP86, HLLJ91] use bandpass processing as an initial stage. The bandpass image is buffered in the Datacube system as well as passed on the Transputer subsystem as edge detection output.

The Datacube implements the optical flow computation to produce dense flow vectors as well as confidence measures from two frames. The images are taken at 30 frames per second; we only use one field of the frame. The images are 512 by 240 – our Sony cameras average successive odd and even pairs so that we do not have missing data in the vertical direction. We smooth with a Gaussian before subsampling to 128 by 120. Optical flow [BLP89] takes $(2d+1)^2$ passes through the images to compute a maximum displacement of magnitude d. Subsampling has several benefits: it reduces the number of pixels, making the operations of the Datacube faster; it also reduces the maximum displacement so that there are fewer iterations. We run optical flow at maximum displacement of 2 pixels in any direction. Then we run stereo at approximately 20 displacements. The stereo algorithm is essentially identical to the optical flow algorithm, restricted to horizontal displacements. This proceeds at approximately 15 frames per second. The stereo, optical flow, edges (zero-crossings of the Laplacian of Gaussian) and Laplacian of Gaussian are composed into an image that is then sent to the Transputer network. The Datacube output is only 64K pixels, and can be transferred at 1.44 MB/s (through one crossbar) or approximately 11 ms on four links. Our system uses 16 workers Transputers so the image actually must be broken into small chunks requiring more transfers, so the data rate from the Maxtran interface is effectively slower.

Let us contrast various data distribution strategies. One effective strategy is to drive the decomposition in the Transputer subsystem by the early tasks: each of n processors receives a fraction $(1/n)$ of the image. The advantages are that the dominant computational cost is in the early/middle stages, and later stages require less. The disadvantage is that each processor requires a full copy of the vision code. Alternatively, one can use functional parallelism and implement each component of the intermediate and high-level vision in separate processor groups in the Transputer subsystem. The advantages are several: first, the previously mentioned load-balancing strategy can be used within each processor group. Second, the individual modules, say, components found by edge linking, can finish and be available to answer requests from the control subsystem, without waiting for completion of other subtasks. The disadvantage is

that data are distributed by module and must be communicated to achieve fusion. The communication costs in fusion are high; put another way, the coupling between functional modules is high. Therefore, data partitioning is appropriate for fusion. Functional parallelism becomes possible when the stages that receive image data from the Datacube can compress the data into a more compact format.

We have implemented a simple fusion scheme to demonstrate that all stages do interact. We use the disparity data and the optical flow data as filters on the Laplacian of Gaussian (LOG) images. Only if the disparity data is in a specific range and optical flow is non-zero is the output of the LOG image non-zero. Data are distributed among the 16 transputers, which do the filtering. The results pass to the other edge of the 4 by 4 mesh and pass into a Transputer frame display that produces RGB images via lookup table. The resulting output occurs at approximately 10 frames per second.

5 Discussion

We have described a hybrid architecture that permits a useful compromise between effectiveness and economy. Its structure is specialized to the particular hardware; nevertheless, the process model is general and relies on insights common to many vision applications. The pipeline system (over 1.2 Gop in 8x8 convolution) exceeds the power of a large number of Transputers. Functional parallelism, moreover, permits timely processing of data from sensors, while later processors performing high-level tasks access image maps in a task-dependent fashion.

The architecture is not be biased toward any one technique; it supports massively parallel methods, such as Markov Random Fields and a variety of transform techniques. Current implementations are at a variety of levels in vision. The computational system contains a significant state component; we plan to use previous fields projected to next time step to influence the next decision (boundaries from previous image constrain the next). The computational model also supports distributed methods, including cooperating expert subsystems, symbolic processing, and geometric computation.

References

[AWG91] H.V. Sreekantaswamy A. Wagner and N. Goldstein. A template-based approach to programming multicomputers. Technical report, Univ. of British Columbia, Dept. of Computer Science, 1991.

[Ble89] Guy E. Blelloch. Scans as primitive parallel operations. *IEEE Transactions on Computers*, 38(11):1526–1538, November 1989.

[BLP89] H. Bulthoff, J. J. Little, and T. Poggio. Parallel computation of motion: computation, psychophysics and physiology. In *IEEE Workshop on Vision and Motion*, pages 165–172, Irvine, CA, 1989.

[Bro87] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Transactions on Robotics and Automation*, 2:14–23, 1987.

[DP86] Michael Drumheller and Tomaso Poggio. On parallel stereo. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 1439–1448, Washington, DC, 1986. Proceedings of the IEEE.

[Hil85] W. Daniel Hillis. *The Connection Machine.* MIT Press, Cambridge, Mass., 1985.

[HLLJ91] J. Mikko Hakkarainen, James J. Little, H.S. Lee, and J.L. Wyatt Jr. Interaction of algorithm and implementation for analog vlsi stereo vision. In *Proceedings of 1991 SPIE Conference on Visual Information Processing: From Neurons to Chips,*, April 1991.

[HS86] W. D. Hillis and G. L. Steele Jr. Data parallel algorithms. *Communications of the ACM*, 29(12):1170–1183, December 1986.

[Kun82] H. T. Kung. Why systolic architectures. *IEEE Transactions on Computers*, 15(1):37–46, 1982.

[KW86] H.T. Kung and Jon A. Webb. Mapping image processing operations onto a linear systolic machine. *Distributed Computing*, 1:246–257, 1986.

[LBC89] James J. Little, Guy E. Blelloch, and Todd Cass. Algorithmic techniques for vision on a fine-grained parallel machine. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(3):244–257, March 1989.

[Liu91] J. Liu. Algorithms for scheduling imprecise computations. *IEEE Computer*, pages 58–68, May 1991.

[Low90] D. G. Lowe. Integrated treatment of matching and measurement errors for robust model-based motion tracking. In *Proc. 3rd International Conference on Computer Vision*, pages 436–440, Osaka, Japan, 1990.

[Pai89] D. K. Pai. Programming parallel distributed control for complex systems. In *Proceedings of 1989 IEEE Symp. on Intelligent Control*, pages 426–432, 1989.

[PGL88] T. Poggio, E. Gamble Jr., and J. J. Little. Parallel integration of vision modules. *Science*, 242(4877):436–440, October 1988.

[Sab88] Gary W. Sabot. *The Paralation Model: Architecture-Independent Parallel Programming.* The MIT Press, Cambridge, Massachusetts, 1988.

[Sha87] Ehud Shapiro. Systolic programming: A paradigm of parallel processing. In *Concurrent Prolog: Collected Papers.* MIT Press, Cambridge, Mass., 1987.

[Sha89] Linda Shapiro. Programming parallel vision algorithms: A dataflow language approach. *International Journal of Supercomputing Applications*, 2(4):29–44, Winter 1989.

[SHG87] Linda Shapiro, Robert Haralick, and M. Goulish. INSIGHT: a dataflow language for programming vision algorithms in a reconfigurable computational network. *Internat. J. Pattern Recognition and Artificial Intelligence*, pages 335–350, 1987.

[Sik90] O. Siksik. Markov random fields in visual reconstruction. TR-90–40, UBC Dept. of Computer Science, Vancouver, BC, 1990.

[TFF88] Lewis W. Tucker, Carl R. Feynman, and Donna M. Fritzsche. Object recognition using the Connection Machine. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, Ann Arbor, MI, June 1988.

[Web90] Jon A. Webb. Architecture–independent global image processing. In *Proc. 10th Int. Conf. on Pattern Recognition*, 1990.