Real-time Optical Flow Range-Estimation on the iWarp

Terrence W. Fong* and Raymond E. Suorsa†
NASA Ames Research Center
Moffett Field, CA 94035
terry@ptolemy.arc.nasa.gov
suorsa@windchime.arc.nasa.gov

January 12, 1993

Abstract

This paper describes the implementation of an optical flow range-estimation algorithm, intended to support autonomous helicopter navigation, on the iWarp distributed-memory multicomputer. The implementation exploits numerous features of the iWarp communications and computation architecture to maximize performance and efficiency. The range-estimation algorithm has been previously implemented on both distributed and shared-memory workstations. Experiences and performance figures from the initial iWarp implementation are discussed along with plans for future refinements.

1 INTRODUCTION

There are many proposed vision-based methods to perform real-time obstacle detection and avoidance for autonomous or semi-autonomous vehicles. All methods, however, are computationally stressing and require sustained computation in the gigaflop performance range. To achieve such performance without use of dedicated or specialized VLSI hardware, it is necessary to employ parallel methods. This paper describes the initial results of a parallel optical flow range estimation algorithm, which provides support for autonomous helicopter navigation, as implemented on the iWarp distributed-memory multicomputer.

1.1 Problem definition

The high computational requirement for machine vision is well-recognized and parallel processing presents an approach to achieve the necessary speed for real-time implementation [1]. The estimation of range using electro-optical sensors, for example, involves the processing of large data streams (e.g., 15 MB/sec) with a computation rate of a few gigaflops [2]. In the past, the solution of such signal and image processing tasks required the use of application-specific computing systems, often implemented by custom VLSI.

This approach, however, has significant drawbacks including high development cost, inflexibility, rapid obsolescence and lack of performance scalability [3]. General-purpose processors, by contrast, can provide high levels of flexibility and functionality with much reduced development cost. At present, however, no general-purpose uniprocessor system alone can provide the required computational performance, thus a parallel implementation is required [4].

Parallelism, however, may not provide the required speed increase and the actual performance improvement is often strongly architecture and application dependent [5]. In particular, the communication network plays a significant factor in determining the overall performance and throughput of a parallel system [4].

*Recom Technologies, Inc.
†Flight Systems and Navigation Division of NASA Ames

Moreover, the selection of a parallel processing system for a particular task may be driven by other application specific requirements, such as low energy consumption, weight, and physical size compactness [3].

1.2 Real-time passive range-estimation

This research specifically seeks to provide real-time range estimation from passive imagery to support autonomous helicopter navigation. Several techniques have been proposed for range determination using electro-optical sensors, optical flow and state data from an inertial navigation system [6, 7, 8]. One algorithm of interest can detect, track and estimate range to image features (i.e., regions with common statistics or spatial structure) over time from a multisensor system mounted on a vehicle moving with arbitrary six degrees-of-freedom [9].

Candidate real-time implementations of the algorithm presented in [9], must be adaptable to changes in the estimation algorithm, exhibit effective performance scalability, and have the potential for on-board helicopter use. Moreover, due to the algorithm's complexity, a traditional "image processing" approach will be lacking in flexibility [1].

These constraints emphasize the need for a parallel system utilizing general-purpose processors over application specific computing systems. This does not, however, restrict the use of such devices as processing "front-ends" to the parallel system. Pipelined image processors, for example, may be used to provide a preprocessed frame sequence (e.g., edge-enhanced) to the algorithm running on the parallel machine.

Given existing "off-the-shelf" parallel systems, therefore, the task is to select an architecture which utilizes general-purpose processors and which satisfies other physical criteria. In particular, the candidate system must meet (or have the potential for) the following:

- gigaflop computation range
- low weight
- low power
- high-speed input/output (15 MB/sec or greater)

One architecture which appears to satisfy these requirements is the iWarp.

1.3 iWarp overview

The iWarp is a distributed-memory multicomputer and is the product of a joint development effort between the Intel Corporation and Carnegie Mellon University (CMU). The system supports both tightly and loosely coupled parallel processing and was designed to support high-performance signal processing via balanced communication and computation [10]. The iWarp is distinguished from other distributed-memory multicomputers by three significant features: high-bandwidth internode communications, systolic processing, and multiplexed physical connections [11].

Under the iWarp architecture, processing elements (PEs or "nodes"), are connected in a 2-D toroidal mesh "array". Input/output for the array is handled by special interface nodes. At present, the only interfaces are the "Sun Interface Board" (SIB), which provides communications and proxy services to a Sun workstation, and the "iWarp Serial Interface" (iSIO). The latter provides 60 MB/sec transfer rates and is intended as a general purpose interface to high-speed devices such as frame grabbers, video, and disk [12]. A schematic diagram of the iWarp architecture is shown in Fig. 1.

2 METHODOLOGY

2.1 Multisensor feature-based range-estimation algorithm

A candidate range-estimation algorithm which has shown promise for autonomous helicopter navigation has previously been described in [9, 13, 14]. This algorithm exhibits computational and communication patterns which do not decompose well into standard "image processing" paradigms [2]. In particular, the algorithm is data driven and contains a highly irregular computation structure.

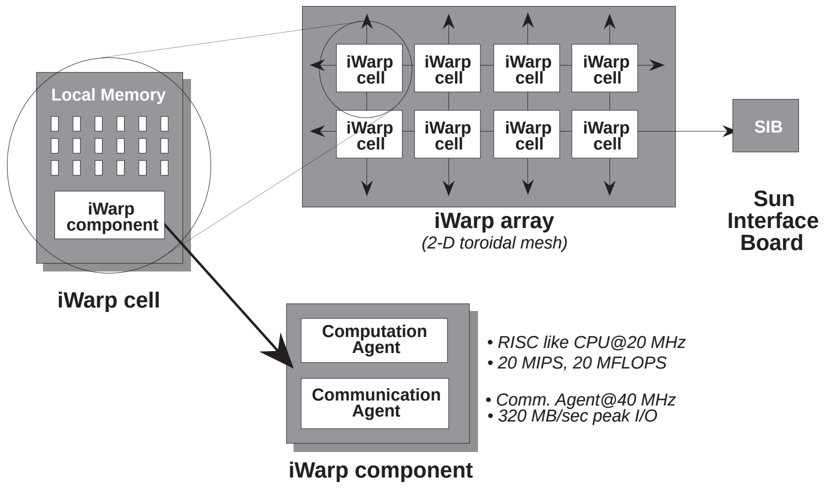


Figure 1: Schematic diagram of the iWarp distributed-memory multicomputer.

The range-estimation algorithm is a modified version of area-based matching which is designed to overcome some of the limitations commonly associated with that method [15]. The algorithm tracks small regions of motion imagery (called "features") over time and uses an extended Kalman filter to estimate the feature's range. The knowledge of sensor velocity, angular rate and the estimated range is used to limit the search area.

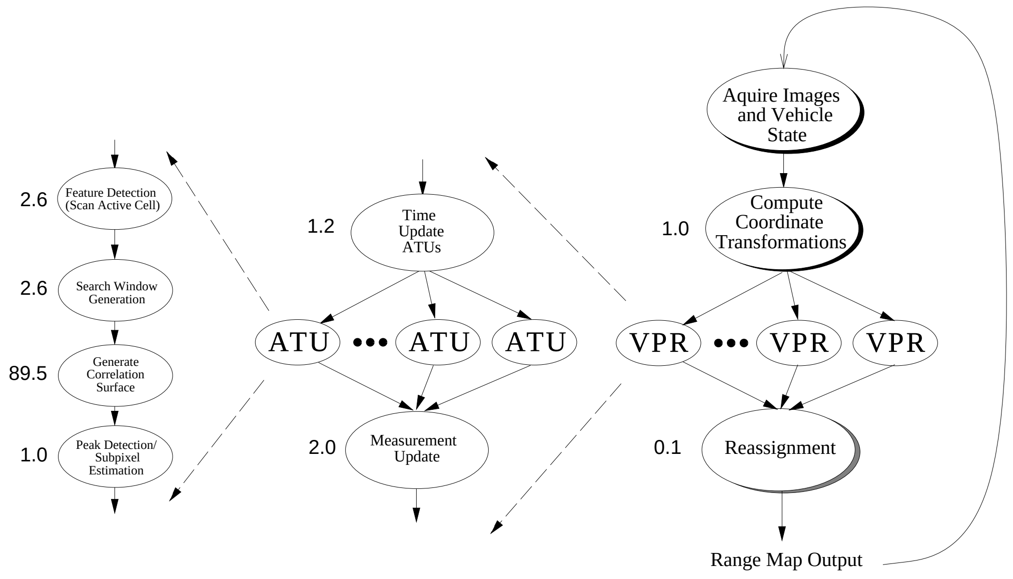
The matching is currently performed using normalized correlation [9]. This method, though computationally intensive, gives accurate results and is immune to linear changes in scene brightness. The major computational portion of the feature-based tracking algorithm is in performing the normalized correlations necessary to form a correlation surface. A correlation surface is generated by correlating the pixel array, which contains a feature, with candidate pixel arrays defined by a search window [2], from an image acquired later in time. The peak of this surface indicates the location of a feature at the new time, and can be used by an extended Kalman filter to estimate the spatial coordinates of the ground object which corresponds to the feature.

2.2 Parallelization

The range-estimation algorithm has been parallelized into a two tiered structure (see Fig. 2) as described in [2]. The lowest level is the autonomous tracking unit (ATU). The tracking unit can track a feature in a single sensor over time. Autonomous tracking units are task parallel in nature. Once a feature is detected an ATU is spawned to track the feature. If a feature leaves the image plane or otherwise becomes untrackable then the ATU dies. As motion imagery evolves, ATUs will track the optical flow within the image. Thus, for foward vehicle motion, an ATU will generally flow from the center of the image towards the edge. The general flow of an ATU, however, cannot be known a priori due to the unknown nature of evolving scene content.

The autonomous nature of ATUs, therefore, will lead to data requirements which will evolve to be nonlocal within the image space. Nonlocal data requirements for each task can lead to very poor performance in a multiprocessor system, due mainly to communication overhead in a distributed-memory system, or memory network contention in a shared memory system [16]. The primary implication of this is that task parallelism alone is not sufficient to efficiently map the algorithm onto parallel hardware. To overcome the data locality requirements, a higher level abstraction, the virtual processor region (VPR), adds spatial locality restrictions to each ATU within the image space.

Each VPR is responsible for maintaining a rectilinear arrangement of grid cells. The VPRs represent separate regions within the image plane that can be allocated to a processing element (PE). Each PE processes the ATUs and performs feature detection in untracked grid cells which are contained within its



2.6
Feature Detection (Scan Active Cell)
2.6
Search Window Generation
89.5
Generate Correlation Surface
1.0
Peak Detection/ Subpixel Estimation
1.2
Time Update ATUs
ATU
ATU
ATU
2.0
Measurement Update
Aquire Images and Vehicle State
1.0
Compute Coordinate Transformations
VPR
VPR
VPR
0.1
Reassignment
Range Map Output

Figure 2: Range-estimation algorithm flowchart.

assigned VPR. Therefore, as an ATU tracks a feature across a VPR boundary, the VPR passes the ATU to the appropriate neighboring VPR before the next image set is acquired.

The feature tracking algorithm based on virtual processing regions exhibits Single Program Multiple Data (SPMD) parallelism. Each VPR can be processed in parallel as soon as its input data has been supplied. The VPR, however, can exploit further parallelism at the ATU level if it contains more than one grid cell. Each ATU/grid cell can be processed in parallel. The data requirements for each ATU are implicitly supplied by the parent VPR. The ATU in turn is composed of a series of serial matrix-like operations which exhibit vector-like parallelism.

2.3 Load balancing

Since each VPR may manage none to many ATUs, a load balancing scheme is necessary to fully utilize all the processors in a distributed system. Several load balancing schemes have been explored for this algorithm [2]. The most promising scheme, static load balancing, has been used for the iWarp implementation.

Given that the master image is divided into M VPRs, static scheduling attempts to distribute all M VPRs from the current frame onto a set of N processors so as to minimize completion time. It is required that $M > N$ to so that the scheduler has the resolution to properly distribute the load. More precisely: Given N processors, a deadline D and an M element set, $\mathcal{X}$, of VPRs each with estimated computation time τ_i, select a disjoint partition of $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2 \cup \cdots \cup \mathcal{X}_N$ such that

$$\max \left\{ \sum_{i \epsilon \mathcal{X}_j} \tau_i : 1 \leq j \leq N \right\} \leq D \tag{1}$$

The estimated time necessary to process all VPRs with a uniprocessor is

$$T = \sum_{i=1}^{M} \tau_i \tag{2}$$

Thus the best case deadline D possible, given N processors, is T/N. This is known as the Multiprocessor Scheduling problem and has been shown to be NP-complete [17]. The challenge of static scheduling is to choose the partitions $\mathcal{X}_j$ in a computationally efficient manner such that the maximum PE compute time approaches T/N. The method used in [2] to choose the partitions $\mathcal{X}_j$ is the efficient heuristic, "*First Fit Decreasing*" (FFD) described in [17] and [2].

3 iWARP IMPLEMENTATION

3.1 Overview

3.1.1 Design assumptions

The range-estimation algorithm was implemented on the iWarp after considering prior parallel instantiations. This previous work, described in [2], yielded two findings upon which the iWarp implementation was based. First, it was determined that the algorithm is amenable to coarse-grained processing by a distributed-memory machine. Second, that the most detrimental aspect of the previous distributed-memory implementation, an Ethernet-based network of UNIX[1] workstations, was the low bandwidth of the node interconnect.

Considering these findings, two design assumptions were made. First, it was assumed that the parallelization described in [2] was effective and that the overall execution time would be reduced by increasing the number of nodes. Second, it was expected that the distributed-memory implementation would become largely compute-bound if the node interconnect bandwidth could be improved.

3.1.2 Constraints

Several constraints impacted the implementation design. Foremost were restrictions resulting from current iWarp systems. At present, there are relatively few installations (i.e., 30-40 sites) and most have 8 (or fewer) nodes. Moreover, almost all machines have limited node memory, typically 512 KB static RAM per node.

Another factor which constrained the design was the input/output interface. During development, the only interface available was the SIB. As discussed previously, this interface provides communications and proxy services between an iWarp and a Sun workstation. The SIB, however, is slow and only provides bandwidths from 0.2 MB/sec (host filesystem access) to 6 MB/sec (node/host process communication).

Finally, the existing iWarp software tools restricted the development process. The standard development environment is limited, consisting of single-node compilers (i.e., C, FORTRAN) and internode communications libraries. A rudimentary facility is also provided for array to host (i.e., Sun) communications. This environment is immature relative to other current parallel systems (e.g., iPCS/860, CM-5) and does not provide abstractions beyond the node level. It should be noted that alternative environments, such as "Express"[2] and the Intel "parallel program generators" (i.e., Adapt, Apply, and Assign) do exist. These alternative environments, however, were considered inappropriate due to inherent limitations or overhead.

3.1.3 Functional description

A distributed-memory implementation of the parallel range-estimation algorithm has been constructed for the Oak Ridge National Laboratory (ORNL) iWarp. The system, shown in Fig. 3, consists of 16 nodes with 2MB of static RAM per node. The SIB supplies the single input/output interface on the system and provides an additional node with 2MB of static RAM. Although the communication bandwidth provided by the SIB is not sufficient for real-time use, it was considered acceptable for development purposes. A high-performance video interface, which will utilize the iSIO, is currently under development and will meet real-time requirements.

For the initial iWarp implementation of the range-estimation algorithm we have chosen to utilize the same coarse-grained, message-passing computation model previously used in [2] for distributed-memory machines. The "Programmed Communications Services" (PCS) tool chain and libraries, developed at CMU, provided static network communications for programmed scatter and gather. An optimizing C compiler was used to generate low-overhead loops and software pipelining.

[1]UNIX is a registered trademark of AT&T.
[2]Express is a trademark of the Parasoft Corporation.

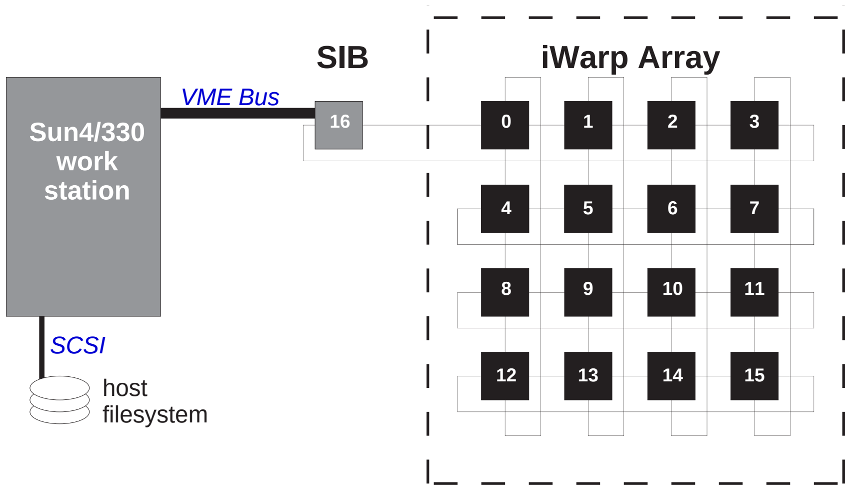


Figure 3: Oak Ridge National Laboratory (ORNL) iWarp system.

3.2 Communications

The communications capability of the iWarp offers tremendous flexibility for implementing parallel algorithms. For example, the channel multiplexing ability of the iWarp communications agent allows for a wide variety of connection topologies, such as meshes, hypercubes and snakes. Additionally, the peak communications bandwidth of 320 MBytes/sec (160 MBytes/sec full-duplex) simplifies the development of efficient parallel software since the communications cost is low. Finally, support for both message passing and systolic communications models provides fine control over the granularity of communication and computation [18].

3.2.1 Communications design

The flexibility of the iWarp communications agent presents the designer with a broad spectrum of options for implementing distributed communications. Since the designer has great control over network topology and message granularity, it is extremely important to understand specific application requirements when selecting a communications method.

Given the design assumptions described in section 3.1.1, it was expected that the greatest benefit would be realized if a high node interconnect bandwidth could be attained. Moreover, since the loosely-coupled, task-parallel model utilizes regular communications patterns (i.e., managed scatter/gather), it is possible to setup a static data distribution network and to ensure efficient resource management. Specifically, by explicitly managing the communications it is possible to ensure that no node becomes deadlocked or causes starvation.

A static data distribution network using PCS was implemented on the iWarp. PCS provides an array-level abstraction of the iWarp as well as direct program access to the iWarp communication hardware [19]. For this implementation, we chose to use a pair of uni-directional, snake networks as shown in Fig. 4. A sending (master) node is located at one end of both networks. The networks traverse the array and make turns at the edges. Receiver (slave) nodes are located at each node and have taps into both networks. One network is then used for sending data from the master node to the slaves (i.e., scatter) and the other for receiving computed results (i.e., gather).

The data scatter/gather technique utilizes a unique feature of the iWarp communication hardware. First, the master node sends an open (start) message marker along the scatter network. This marker is then followed by the data packages for each of the slave nodes, and finally a close (end) message marker. In essence, a single message is sent via the scatter network during each iteration.

As the message proceeds, each slave node in turn receives the open message marker and then takes its data from the data stream. Once the node has received its data, it forwards the open message marker,

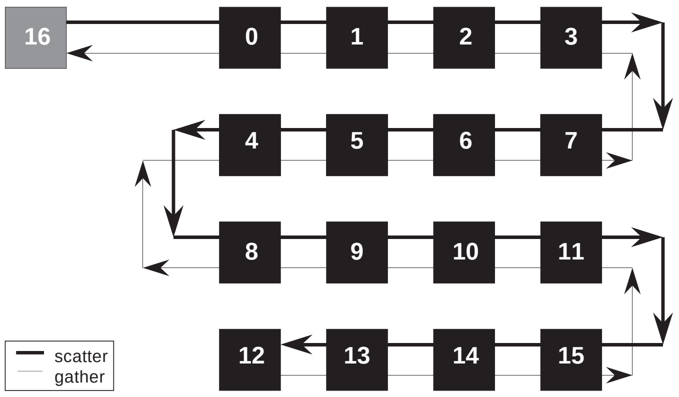


Figure 4: Static data distribution network.

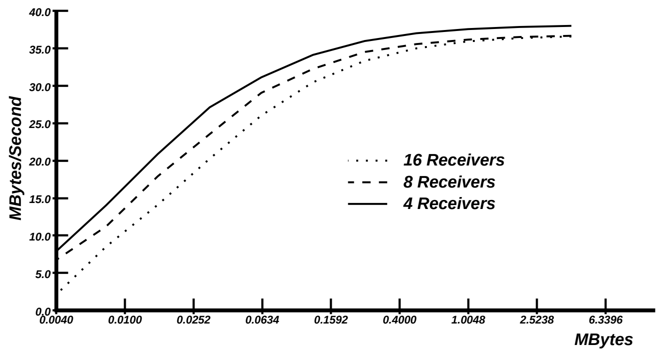


Figure 5: Static data distribution network performance.

commands its communication agent to forward the remainder of the message in "express" mode and begins processing. Once a communication agent is placed in express mode all data is passed through the node without intermediate buffering and without consuming further resources from the computation agent. The last slave node, however, does no forwarding and merely receives the close message marker.

After the slave nodes complete processing, the results are returned in a similar manner to the master node. Due to the low-latency of communications, and because the communications agent is able to operate without taking resources from the computation agent, this technique is extremely efficient for distributing data in a low-latency, high-performance manner.

3.2.2 Communications analysis

The network mapping shown in Fig. 4 was chosen for three reasons. First, it optimally implements a linear network across any number of nodes. Since scalability is a concern for tuning the range-estimation algorithm and because existing iWarp systems have few nodes, efficient network topology mapping is a driving factor. Second, the mapping maximizes communications performance and utilizes a minimum of node resources. Since each node is connected to its neighbors by a single physical bus, multiplexing networks reduces the bandwidth available to each network. Finally, the mapping is able to utilize the express routing capability of the iWarp communications hardware to provide high bandwidth data distribution between the master and the slave nodes.

The performance of the static network was evaluated given varying network lengths over a range of data distribution loads. In each case, a sending (master) node distributed data equally to a number of receiving (slave) nodes. A distribution of 4 MB from the master to 4 slaves, for example, resulted in each slave receiving 1 MB of data. The performance for networks containing 4, 8, and 16 receiver nodes is shown in Fig. 5. As the figure shows, the distribution rate increases asymptotically towards a saturation rate as the data size increases. In all cases, near peak (saturation) transfer rate is achieved for transfer loads above 1 MB. The variance in peak rate between different network lengths may be attributed to delays in switching communications modes and the latencies of communications through a node.

3.3 Computation

The computation agent of the iWarp provides a powerful general-purpose processing engine. The 20 MHz processor operates with an instruction set similar to other 32-bit RISC devices, but additionally provides a 96-bit "Long Instruction Word" (LIW) for superscalar operations. The LIW is extremely flexible and may be used to operate multiple functional units (e.g., integer/logic, floating-point multiply, etc.) in parallel, to perform simultaneous computation and communication, and to provide optimizations such as zero-overhead loops and software pipelining[12].

Most significantly, however, the iWarp computation agent is designed to perform numerical computations while maintaining a high sustained floating-point arithmetic rate. The processor achieves this through the use of nonpipelined floating-point units, and provides high performance for both vectorizable and non-vectorizable codes. The peak performance rating of the computation agent is 20 MFLOPS for single (32-bit) precision operations concurrent with 20 million integer or logical operations per second [10].

3.3.1 Computation design

To enhance computation efficiency and to maximize use of the LIW it was necessary to code the range-estimation algorithm according to particular guidelines. For example, flat iteration with constant bounds and indexing was used wherever possible. Iteration is preferred to recursion since it reduces function call overhead. Constant bounds and indexing allows the optimizer to perform substantially better pipelining and to utilize zero-overhead loops [20].

Since the computation agent has independent, non-pipelined floating-point addition and multiplication units, judicious use of floating-point rather than integer math can yield better performance. Additionally, unlike vector machines, the iWarp does not require that loops be vectorizable to enable full utilization of the floating-point units [10]. A fundamental performance limit, however, is implied by the use of independent units. In particular, if a calculation contains specific data dependencies, additions and multiplications will be serialized and the performance will be marginal.

3.3.2 Computation analysis

As an example of how these computational constraints impact code design, consider the core loop of the correlation surface function. This function dominates the feature tracking processing and accounts for approximately 90 percent of the computational time. Originally, the code was written as shown in Fig. 6, with a nested-loop pair and integer math.

To improve performance, the inner loop was manually unrolled, floating-point used instead of integer math, and full C optimization (including software pipelining) invoked. As a result, a speedup of 8.5 was realized, reducing the execution time by 88%. Although this decrease is substantial, there remains significant potential for further improvements. These issues are discussed in section 4.2.

3.4 iWarp processing

As previously implemented on a distributed-memory machine (i.e., networked workstations), the range-estimation algorithm was structured so that a single master PE would schedule and distribute VPRs via ethernet to the slave PEs. The slave PEs, which contain the bulk of the feature tracking code, then processed a set of VPRs in parallel and returned results to the master PE [2].

For the iWarp implementation, this processing structure was maintained. The master PE was assigned to the SIB node and slave PEs to other iWarp nodes. In place of ethernet communications, the static

```
char left[128][128], right[128][128];
int sum1, sum2, sum3, sum4, sum5, a, b, x, y;
sum1 = sum2 = sum3 = sum4 = sum5 = 0;

for (y = -5; y <= 5; y++) {
  for (x = -5; x <= 5; x++) {
     a = (int) left[j+y][i+x];
     b = (int) right[h+y][t+x];
     sum1 += a*b;
     sum2 += a;
     sum3 += b;
     sum4 += a*a;
     sum5 += b*b;
  }
}
```

Figure 6: Core loop of the correlation function.

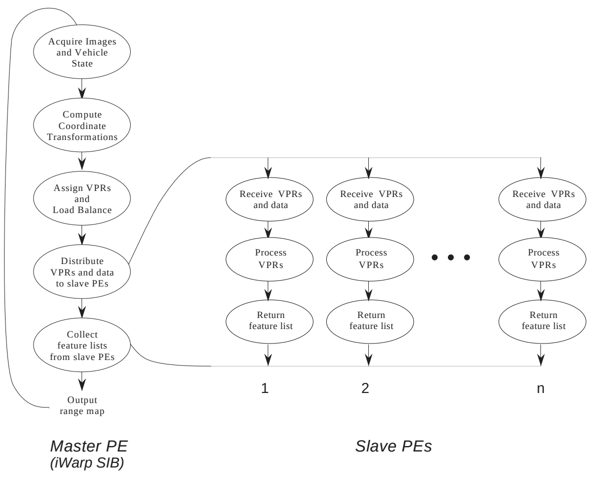

Acquire Images and Vehicle State
Compute Coordinate Transformations
Assign VPRs and Load Balance
Distribute VPRs and data to slave PEs
Collect feature lists from slave PEs
Output range map
Receive VPRs and data
Process VPRs
Return feature list
1
2
n
Master PE (iWarp SIB)
Slave PEs

Figure 7: iWarp processing control and data flow.

data distribution network described in section 3.2.1 was utilized. Due to memory limitations of the ORNL iWarp, however, it was not possible to directly replicate the workstation based processing. Specifically, it was necessary to reduce the processing code and work with quarter-size (i.e., 512x128, 8-bit pixel) images. Currently, the approximate code size for the master and slave programs are 1.5MB and 1MB respectively.

For each processing loop (see Fig. 7), the master PE obtains the image set to be processed, partitions the set based on load-balancing criteria, and determines the data package for each slave PE. The data is then distributed to the slave PEs via the scatter network. Once all the data has been placed onto the scatter network, the master PE spins while computation on the slave PEs proceeds. After computation has been performed, the slave PEs return results via the gather network to the master PE. The master PE performs cleanup activities, and then the loop begins again.

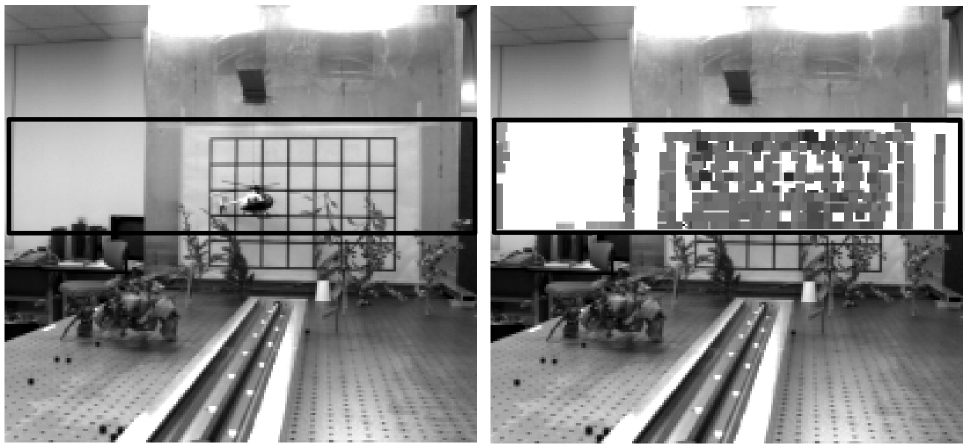


Figure 8: Image with corresponding intensity coded range map.

Compute Time		
# Nodes	Ave	Max
4	1.286002	1.419808
8	0.642997	0.887289
16	0.321598	0.422483

Table 1: Compute node time with load balancing and 110 VPRs.

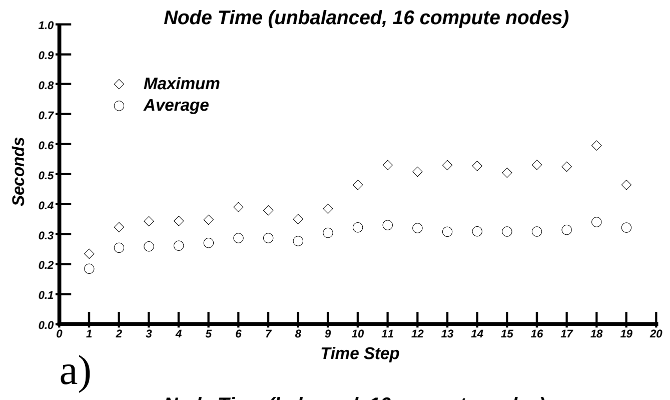
4 DISCUSSION

4.1 Preliminary results

All the results were generated using set of 20 stereo image pairs of pixel dimension 512x128 with 8 bits per pixel. This set is a cropped portion of an image sequence generated using a computer controlled motion table and scaled helicopter dynamics recorded from flight [2]. Fig. 8 shows the full (512x512) 20th input image (left) and the image with range-map overlay (right). The boxed region represents the 512x128 section processed by the iWarp implementation. The range map is intensity coded, with low to high levels representing near to far ranges. To date, two methods have been evaluated for partitioning the input data space.

The first method divided the image into 110 rectangular VPRs. The relatively large number of VPRs was used to provide sufficiently fine computational granularity for the static load balancer. Fig. 9a,b presents the average and maximum node times for 16 compute nodes with and without balancing. As the figure shows, static balancing performs well and reduces load variance considerably. Table 1 lists the average and maximum node times for increasing number of nodes using load balancing. The table shows that the compute time decreases linearly as the number of nodes increases, which is as expected for well-balanced computational loads. Communications testing, however, revealed that this method results in extremely poor communications performance (approximately 0.1 MB/sec). Preliminary analysis indicated that the low bandwidth was attributed to the data partitioning, which required intermediate block-to-block data copies. To verify this hypothesis, the second method, which did not require block copies, was evaluated.

The second data partitioning method divided the image into 11 horizontal strips and assigned a single VPR to each strip. Without the block data copies, a dramatic improvement in communications was realized, with an average bandwidth of 27 MB/sec for data transfers. This method, using 11 slave PEs, was able to process 317 features of the 20th master sensor image (512x128 pixel frame) with a loop time of 1.36 seconds. However, since only a single VPR was assigned to each slave PE, load balancing was not performed and the individual node times varied greatly (i.e., 0.084 to 1.16 seconds).



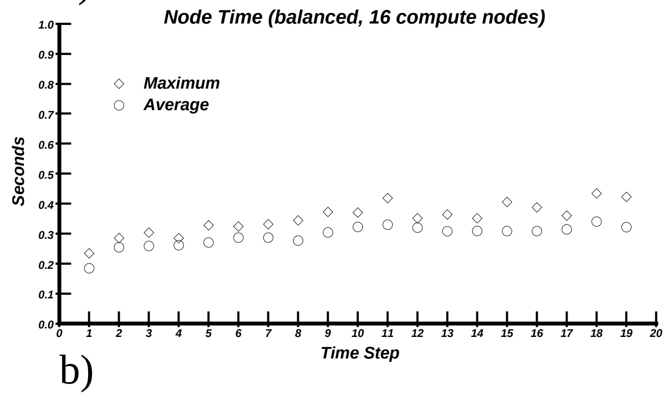
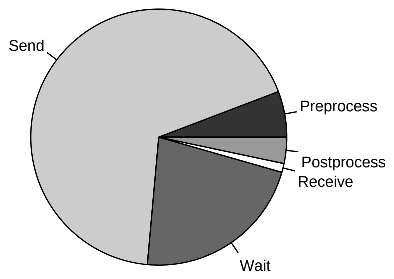


Figure 9: Effect of load balancing.



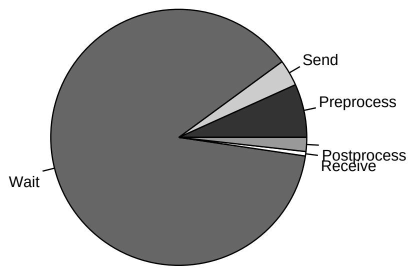


Figure 10: Loop time detail (not including iWarp to Host I/O).

Fig. 10 details the loop time (time to process the 20th image pair) of the master PE for both partitioning methods. These piecharts do not include the host I/O time, which is only indicative of the SIB's performance and not of the parallel process. The graph on the left, corresponding to the first method, reveals that nearly 65% of the loop time is spent in the send (data scatter) portion and that the implementation is communications-bound. This is directly attributable to the low data bandwidth. The graph on the right, representing the second method, illustrates that improved communications has changed the implementation to be compute bound. With the higher bandwidth, the master PE spends 75% of its time waiting for the slave PEs to complete processing.

4.2 Future work

Analysis of the initial iWarp implementation has indicated that performance could be greatly improved if several refinements are made. These include hand-coded assembly for critical sections, the elimination of barrier synchronization calls, spooling for asynchronous communications, and refined network structure for use on larger arrays.

Of these refinements, the greatest benefits would likely result from assembly coding. Since the current implementation is compute-bound, hand-coded assembly of critical sections could greatly increase execution speed. This is particularly significant since the implementation processes character data. Since the iWarp is a 32-bit architecture, the loading of an individual byte requires more cycles than word loads. Assembly coding could utilize word loads and dramatically reduce execution time, perhaps by a factor of 10-20.

5 CONCLUSIONS

The initial implementation of an optical flow range-estimation algorithm on the iWarp distributed-memory multicomputer has been achieved. Two methods for input data partitioning were evaluated. The first method, which partitions image data into rectangular regions, is amenable to static load-balancing and shows linear speedup with up to 16 processors. This method, however, suffers from poor communications performance. The second method, which segments images via horizontal strips, achieves high-bandwidth communications but does not provide sufficiently fine computational granularity for effective load-balancing. This latter method is able to process 317 features of the 20th master sensor image (512x128 pixel frame) with a loop time of 1.36 seconds.

In the next phase of our research, we will refine the implementation to merge the positive aspects of both data partitioning methods and to improve the overall performance. Our goal is to achieve real-time performance and process ten frames per second with a maximum of 350 features. Given streamlined computations and sufficient node memory, this is a realistic goal which we expect will be achieved.

ACKNOWLEDGMENTS

This work was supported in part by Tice DeYoung, DARPA CSTO, under ARPA #7718. The authors would like to express sincere appreciation to Judson Jones and the Oak Ridge National Laboratory for use of the ORNL iWarp machine. Thanks are due to Susan Hinrichs (CMU) for assistance with PCS.

References

[1] A.N. Choudhary and J.H. Patel. *Parallel Architectures and Parallel Algorithms for Integrated Vision Systems*, chapter 6, pages 123–128. Kluwer Academic Publishers, Norwell, MA, 1990.

[2] R.E. Suorsa. A parallel implementation of a multisensor feature-based range-estimation method. Technical memorandum, NASA, Ames Research Center, Moffett Field, CA, February 1993.

[3] B. Baxter, G. Cox, T. Gross, H.T. Kung, D. O Hallaron, C. Peterson, J. Webb, and P. Wiley. Building blocks for a new generation of application specific computing systems. Technical memorandum, Intel Corporation, Hillsboro, OR, 1989.

[4] D. Reed and R. Fujimoto. *Multicomputer Networks Message-Based Parallel Processing.* MIT Press, Cambridge, MA, 1989.

[5] L.S. Haynes, R.L. Lau, D.P. Siewiorek, and D.W. Mizell. A survey of highly parallel computing. *Computer*, 15(1):9–27, January 1982.

[6] B. Sridhar and A.V. Phatak. Simulation and analysis of image-based navigation system for rotorcraft low-altitude flight. In *Proceedings of the AHS Meeting on Automation Application for Rotorcraft*, Atlanta, GA, April 1988.

[7] P. K. Menon, G. B. Chatterji, and B. Sridhar. Vision-based optimal obstacle avoidance guidance for rotorcraft. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, New Orleans, LA, August 1991.

[8] Y. Barniv. Velocity filtering applied to optical flow calculations. *IEEE Trans. on Aerospace and Electronic Systems*, to be published, October 1992.

[9] B. Sridhar, R.E. Suorsa, and B. Hussien. Passive range estimation for rotocraft low altitude flight. *Journal of Machine Vision and Applications*, 1991.

[10] S. Borkar, R. Cohn, G. Cox, T. Gross, H.T. Kung, M. Lam, M. Levine, B. Moore, W. Moore, C. Peterson, J. Susman, J. Sutton, J. Urbanski, and J. Webb. iWarp: An integrated solution to high-speed parallel computing. In *Proceedings of Supercomputing 88*, pages 330–339, November 1988.

[11] T. W. Fong. The iWarp guide: An introduction to the iWarp at NASA Ames. Technical memorandum, NASA, Ames Research Center, Moffett Field, CA, 1993.

[12] B. P. Hine III and T. W. Fong. Evaluation of the Intel iWarp parallel process for space flight applications. In *Proceedings of the 1993 AIAA Aerospace Design Conference*, 1993.

[13] B. Sridhar and R.E. Suorsa. Integration of motion and stereo sensors in passive ranging systems. *IEEE Trans. on Aerospace and Electronic Systems*, 27(4):741–746, 1991.

[14] B. Sridhar, P.N. Smith, R.E. Suorsa, and B. Hussien. Multirate and event driven Kalman filters for helicopter passive ranging. In *Proceedings of the 1st IEEE Conference on Control Applications*, Dayton, Ohio, September 1992.

[15] G. Medioni and R. Nevatia. Segment-based stereo matching. *Computer Vision, Graphics, and Image Processing*, 31(1):2–18, 1985.

[16] J.L. Hennessy and Patterson D.A. *Computer Architecture A Quantitative Approach.* Morgan Kaufmann Publishers, Inc, 1990.

[17] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completness*, chapter 3, pages 63–65. W.H. Freeman and Company, San Francisco, CA, 1979.

[18] S. Borkar, R. Cohn, G. Cox, T. Gross, H.T. Kung, M. Lam, M. Levine, B. Moore, W. Moore, C. Peterson, J. Susman, J. Sutton, J. Urbanski, and J. Webb. Supporting systolic and memory communication in iWarp. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 70–81, May 1990.

[19] D. R. O'Hallaron. iWarp local guide. Carnegie Mellon University, Pittsburgh, PA, 1992.

[20] Intel Corp. iWarp C user's guide. Technical memorandum, Intel Corporation, Hillsboro, OR, 1991.