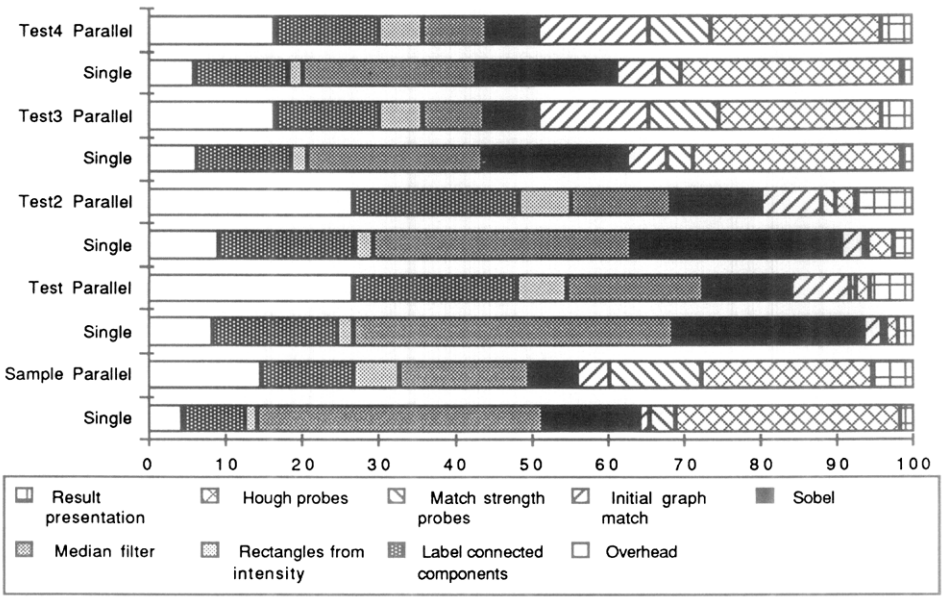


FIG. 8. Alliant FX-80 percentage of effort for each major subtask.

on configurations from 1 to 11 processors, with the optimum time being obtained with 8 or 9 processors. Additional processors resulted in a reduction in performance, due to a combination of factors. As the data were divided among more processors, the ratio of processing time to task creation overhead decreased so that the latter came to dominate the time on some tasks. We also believe that some of the tasks reached the saturation point of the bus, since one run that was observed on a B-model cache system showed performance improving with more processors. The table shows the performance for a single processor running the sequential version, to provide a comparison baseline, and the performance on the optimum number of processors for each data set.

Figure 11 compares the relative time for a single processor versus that for the optimum number of processors, for each data set. As with the Alliant, it can be seen that the fixed overhead cost grows in proportion as the total time decreases. The best performance increase was obtained for the Sobel and median filter portions, because they involve sufficient processing to keep all of the processors busy. Labeling connected components, on the other hand, is less computationally intensive, and thus the process creation overhead is a significant portion of the time. The Test and Test2 data sets also show less than the average performance increase, since the models are assigned to separate processors for the matching process and all but one of the models quickly drops out for those data sets. In the other sets, where matching can continue on multiple models in parallel, the improvement in performance is closer to the average. Result presentation suffers from the same problem as the overhead component:

TABLE XIII

Image Understanding Architecture Simulator Times for Major Subtasks

| Data set | Sample | Test | Test2 | Test3 | Test4 |
|---|---|---|---|---|---|
| Total | 0.0844445 | 0.0455559 | 0.0455088 | 0.4180890 | 0.3978859 |
| Overhead | 0.0139435 | 0.0139435 | 0.0139435 | 0.0139435 | 0.0139435 |
| Label connected components | 0.0000596 | 0.0000596 | 0.0000596 | 0.0000596 | 0.0000596 |
| Rectangles from intensity | 0.0161694 | 0.0125489 | 0.0134704 | 0.0131378 | 0.0129635 |
| Median filter | 0.0005625 | 0.0005625 | 0.0005625 | 0.0005625 | 0.0005625 |
| Sobel | 0.0026919 | 0.0026919 | 0.0026919 | 0.0026919 | 0.0026919 |
| Initial graph match | 0.0155662 | 0.0153462 | 0.0135538 | 0.2953212 | 0.2356714 |
| Match extension | 0.0300650 | 0.0017674 | 0.0024856 | 0.0899214 | 0.1277396 |
| Match-strength probe (average) | 0.0026500 | 0.0001146 | 0.0004095 | 0.0543250 | 0.0071766 |
| Hough probe (average) | 0.0068430 | 0.0003251 | 0.0005092 | 0.0084591 | 0.0109868 |
| Result presentation | 0.0022826 | 0.0009452 | 0.0011944 | 0.0029768 | 0.0029766 |

TABLE XIV

Image Understanding Architecture Validation Output

| Statistics | Sample | Test | Test2 | Test3 | Test4 |
|---|---|---|---|---|---|
| Connected components | 134 | 35 | 34 | 114 | 100 |
| Right angles extracted | 163 | 106 | 100 | 262 | 250 |
| Rectangles detected | 31 | 23 | 19 | 60 | 55 |
| Depth pixels > threshold | 23,185 | 13,598 | 14,065 | 19,730 | 19,753 |
| Elements on initial probe list | | | | | |
| Hough probes | 44 | 5 | 8 | 84 | 100 |
| Initial match-strength probes | 24 | 20 | 15 | 81 | 80 |
| Extension match-strength probes | 20 | 1 | 3 | 41 | 54 |
| Models remaining | 3 | 1 | 1 | 2 | 1 |
| Model selected | 10 | 1 | 5 | 7 | 8 |
| Average match strength | 0.45 | 0.86 | 0.84 | 0.81 | 0.84 |
| Translated to | 151,240 | 256,256 | 257,255 | 257,255 | 257,255 |
| Rotated by (degrees) | 85 | 359 | 113 | 23 | 23 |

being dependent on the sending of an image to an external device.

*Warp*

The CMU Warp is a systolic array consisting of 10 high-speed floating-point cells in a linear configuration [5]. Processing in the Warp is directed by a host processor, such as the Sun-3/60 workstation used in executing the benchmark. The implementation was programmed by one person in 2 weeks, using a combination of the original C implementation and subroutines written in Apply and W2. The objective was to obtain the best overall time, rather than the best time for each task. While it would seem that the latter guarantees the former, consider that the Warp and its host can work in parallel. Even though the Warp could perform in 1 s a step that requires 4 s on the host, it is better to let the host do the processing if it would otherwise sit idle while the Warp is computing. Thus the Warp implementation exploits both the tightly coupled parallelism of the Warp array and the loosely coupled task parallelism present in the benchmark.

Table XVIII lists the major subtask times for the Warp. Note that sums of the times for the individual steps will not equal the total time because of the task parallelism. Table XIX presents the validation data supplied for the Warp implementation.

Figure 12 shows the relative effort of the Warp on each major subtask, for each data set. Because task parallelism

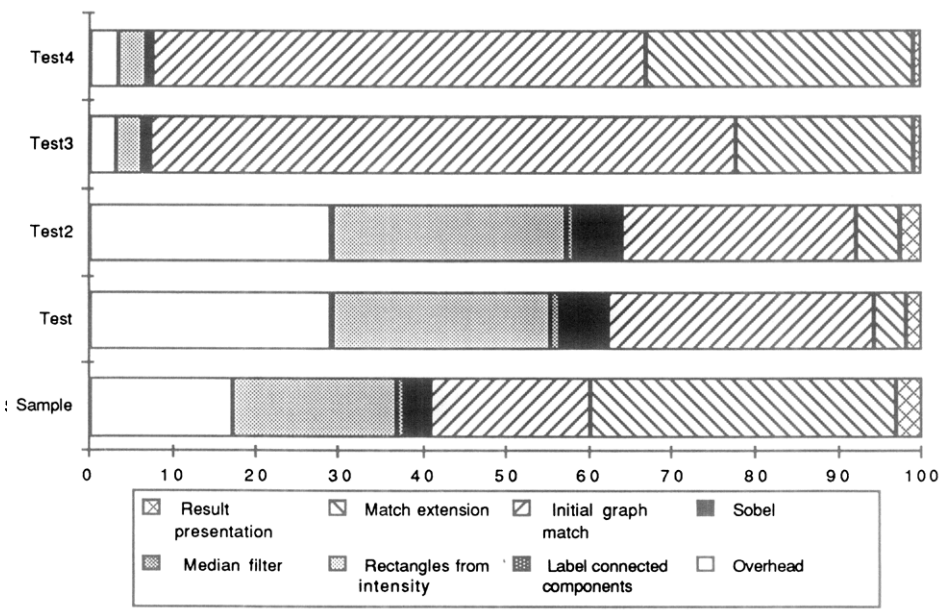


FIG. 9. IUA percentage of effort for each major subtask.

TABLE XV

ASP Simulator Times for Major Subtasks

| Data set | Sample | Test | Test2 | Test3 | Test4 |
|---|---|---|---|---|---|
| Total | 0.130720 | 0.035960 | 0.039810 | 0.113070 | 0.118820 |
| Overhead | 0.000820 | 0.000820 | 0.000800 | 0.000800 | 0.000800 |
| Label connected components | 0.039200 | 0.022800 | 0.022800 | 0.034800 | 0.031300 |
| Rectangles from intensity | 0.003310 | 0.002920 | 0.002880 | 0.003190 | 0.003350 |
| Median filter | 0.000720 | 0.000720 | 0.000510 | 0.000610 | 0.000510 |
| Sobel | 0.000624 | 0.000624 | 0.000624 | 0.000680 | 0.000624 |
| Initial graph match | 0.000009 | 0.000009 | 0.000009 | 0.000009 | 0.000008 |
| Match-strength probes | 0.003300 | 0.000429 | 0.000388 | 0.005640 | 0.006430 |
| Hough probes | 0.080274 | 0.005497 | 0.010545 | 0.063354 | 0.071726 |
| Result presentation | 0.000850 | 0.000440 | 0.000470 | 0.000470 | 0.001030 |

was used, the sum of the individual times exceeds 100% of the total wall-clock time. Thus, Fig. 12 shows percentages of the sum of the major subtask timings. The overhead for the Warp includes initialization and downloading of code for the Warp array and the overhead for the Sun-3/60 host, which performed I/O, extracted the strong cues, and controlled all of the top-down portions of the benchmark. It is clear that the least amount of time was required for the Sobel and median filter operations, which took advantage of Warp's floating-point capabilities. The connected components labeling operation was also performed by the array, as was the *K*-curvature portion of extracting the rectangles from the intensity image. The host performed all of the model matching, but called on the Warp to do the match-strength and Hough probes.

*Connection Machine*

The Thinking Machines Connection Machine model CM-2 is a data-parallel array of bit-serial processors linked by a hypercube router network [4]. In addition, for every 32 processors, a 32-bit floating-point coprocessor is provided. Connection Machines are available in configurations of 4096, 8192, 16,384, 32,768, and 65,536 processors. Results were provided for execution on the three configurations in the middle of the range and extrapolated to the largest configuration. The team at Thinking Machines spent about 3 programmer months converting the low-level portion of the benchmark into 2600 lines of *LISP, a data-parallel extension to Common LISP. There was not enough time to implement the top-down portion of the benchmark before the workshop. However, the implementors also questioned whether the Connection Machine would be the best vehicle for this portion, which is more concerned with task parallelism and would greatly underutilize the machine's potential parallelism. They suggested that if the data base included several thousand models, a method might be found to take advantage of the Connection Machine's capabilities.

Table XX summarizes the results for the Connection Machine, with times rounded to two significant digits (as provided by Thinking Machines). A 32K-processor CM-2 with

TABLE XVI

Aspex ASP Validation Output

| Statistics | Sample | Test | Test2 | Test3 | Test4 |
|---|---|---|---|---|---|
| Connected components | 133 | 34 | 33 | 113 | 99 |
| Right angles extracted | 126 | 99 | 92 | 210 | 197 |
| Rectangles detected | 25 | 21 | 16 | 42 | 39 |
| Depth pixels > threshold | 21,255 | 14,533 | 12,891 | 18,582 | 18,817 |
| Elements on initial probe list | | | | | |
| Hough probes | 55 | 3 | 5 | 97 | 93 |
| Initial match-strength probes | 28 | 20 | 15 | 142 | 142 |
| Extension match-strength probes | 60 | 3 | 5 | 110 | 97 |
| Models remaining | 2 | 1 | 1 | 2 | 1 |
| Model selected | 10 | 1 | 5 | 7 | 8 |
| Average match strength | 0.64 | 0.96 | 0.93 | 0.84 | 0.87 |
| Translated to | 151,240 | 256,256 | 257,255 | 257,255 | 257,255 |
| Rotated by (degrees) | 85 | 359 | 114 | 22 | 22 |

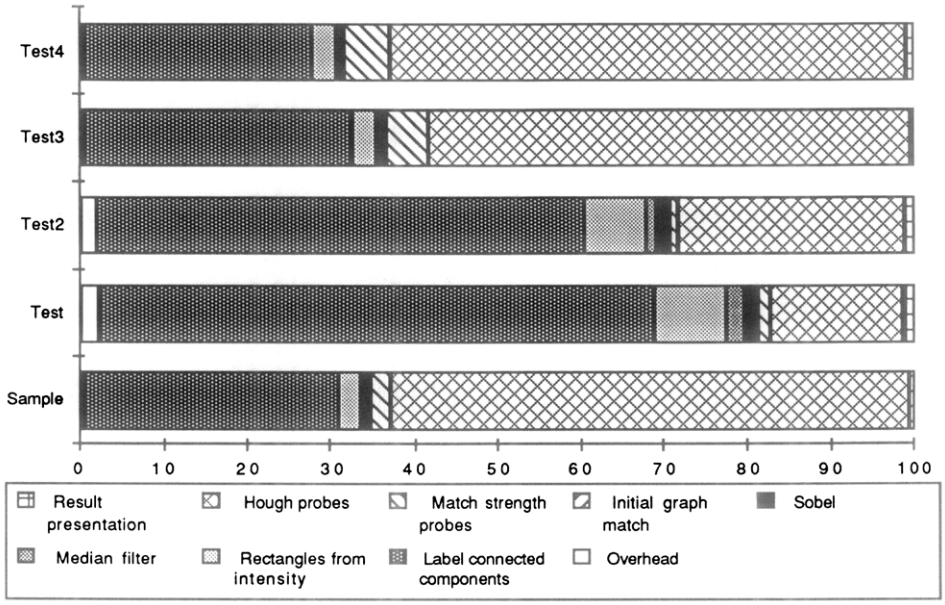


FIG. 10. ASP percentage of effort for each major subtask.

a Data Vault disk system and a Sun-4 host processor was used to obtain the results. Results were supplied for only one data set and did not indicate which one was used. It is interesting to note that several tasks saw little speedup with the larger configurations of the Connection Machine. Those tasks involved a collection of contour values that are mapped into 16K virtual processors, which is enough to operate on all values in parallel, and so there was no advantage in using more physical processors. It was suggested that the Connection Machine might thus be used to process contours for several images at once to make use of the larger number of processors. For those tasks that are pixel-oriented, 256K virtual processors were used and therefore a proportional speedup can be observed as the number of physical processors increases.

*Intel iPSC-2*

The Intel Scientific Computers' iPSC-2 is a distributed memory multiprocessor that consists of up to 128 Intel 80386 processors linked by a virtual cut-through routing network which simulates point-to-point communication. Each processor can have up to 8 Mbytes of local memory and an 80387 math coprocessor. The implementation for the iPSC-2 was developed by the University of Illinois at Urbana–Champaign using C with a library that supports multiprocessing. The group had only enough time to implement the median filter and Sobel steps. However, they did run those portions on five different machine configurations, with 1, 2, 4, 8, and 16 processors, and on four of the five data sets. Table XXI presents the results, which are divided into user

TABLE XVII

Sequent Symmetry 81 Times for Major Subtasks

| Data set | Sample | | Test | | Test2 | | Test3 | | Test4 | |
|---|---|---|---|---|---|---|---|---|---|---|
| Processors: | One | Eight | One | Eight | One | Nine | One | Eight | One | Nine |
| Total | 889.66 | 251.33 | 300.34 | 73.88 | 282.71 | 77.87 | 562.15 | 174.96 | 578.14 | 139.72 |
| Overhead | 5.84 | 6.00 | 5.57 | 5.93 | 5.62 | 5.87 | 5.75 | 5.86 | 5.65 | 5.90 |
| System time | 3.60 | 9.40 | 2.00 | 5.40 | 2.10 | 6.40 | 2.80 | 7.60 | 2.90 | 8.80 |
| Label connected components | 19.27 | 12.68 | 19.34 | 15.83 | 19.29 | 16.01 | 19.60 | 16.84 | 19.58 | 16.89 |
| Rectangles from intensity | 4.18 | 1.45 | 2.62 | 0.92 | 2.74 | 1.92 | 3.42 | 1.42 | 3.38 | 1.89 |
| Median filter | 239.24 | 31.00 | 114.12 | 15.25 | 85.81 | 11.08 | 85.83 | 11.45 | 85.79 | 11.11 |
| Sobel | 110.89 | 15.00 | 113.21 | 15.46 | 110.80 | 14.83 | 110.84 | 15.20 | 110.81 | 14.73 |
| Initial graph match | 18.52 | 3.08 | 18.53 | 3.76 | 19.90 | 4.35 | 52.53 | 7.21 | 51.63 | 7.17 |
| Match extension | 470.90 | 161.34 | 16.16 | 5.97 | 24.08 | 9.38 | 271.07 | 103.99 | 288.21 | 69.10 |
| Result presentation | 20.82 | 20.78 | 10.80 | 10.76 | 14.47 | 14.43 | 13.11 | 12.99 | 13.09 | 12.93 |

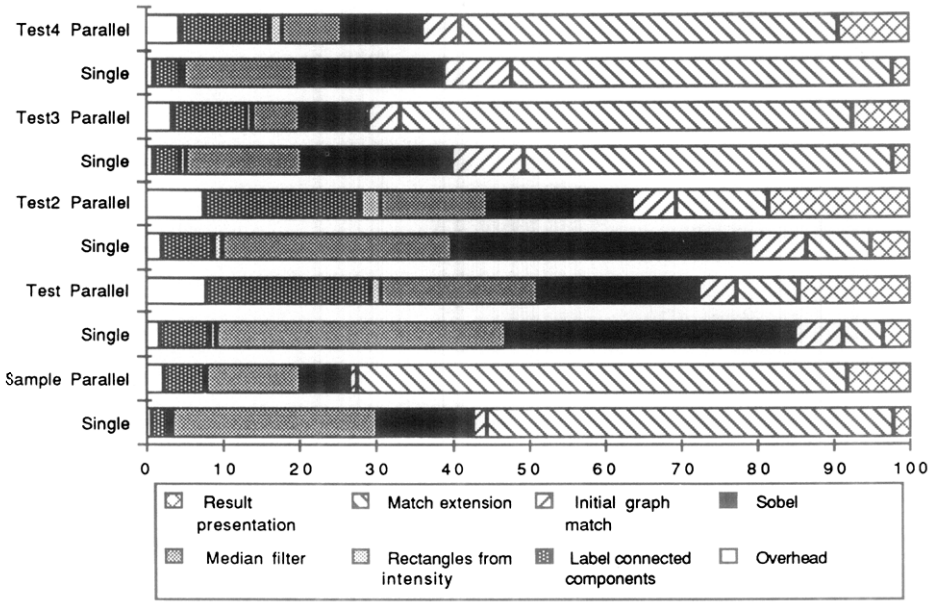


FIG. 11. Sequent symmetry 81 percentage of effort for each major subtask.

time and system time (including data and program load time and output time). Note that no system overhead was counted in the single-processor configuration because uniprocessing is done on the cube server, so no time was required for downloading code and data to the cube processors.

## COMPARATIVE PERFORMANCE SUMMARY

As mentioned above, a direct comparison of raw timings is not especially useful. We leave it to the reader to make informed and intelligent comparisons of the results in the context of his or her own applications. For example, a valid comparison of architectural features should take into account the technology, instruction rate, and scalability of the processors used to obtain the results. An example of such a comparison is given below. The authors hope to develop a reasonably broad set of scaling functions for future versions of the benchmark, so that it will be possible to directly compare architectures that are not too dissimilar.

In the meantime, Figs. 13 through 17 compare the percentage graphs for the different architectures, for the five data sets. Only the complete implementations are shown, since a total time is required to compute the charts. It should again be noted that the results for the IUA and the ASP are from simulations, and the other results are based on actual execution.

The figures make it possible to compare the architectures on the basis of their individual strengths and weaknesses, with technology factored out. Some general observations can be made from these figures. First, some dramatic speed increases are possible in the bottom-up portions of the benchmark, where the communication and control requirements are simple enough to allow data-parallel processing. Second, the multiprocessors show a performance pattern similar to that of the uniprocessors, and in contrast, each of the data-parallel arrays manages to nearly eliminate the cost of one or more subtasks, so that the remaining subtasks are greatly emphasized. Lastly, fixed overhead costs, such as I/O, grow to dominate the processing time, especially when the task itself is small.

### *Issues in Comparing Performance*

The following discussion is intended to show how some pitfalls can be avoided and to point out others that should be kept in mind when comparisons are made.

There are two types of comparisons that can be made: the

TABLE XVIII

Warp Times for Major Subtasks

| Data set | Sample | Test | Test2 | Test3 | Test4 |
|---|---|---|---|---|---|
| Total | 43.60 | 20.30 | 22.30 | 58.10 | 55.30 |
| Overhead | 14.14 | 13.18 | 14.68 | 17.82 | 19.70 |
| Label connected components | 3.98 | 4.04 | 4.60 | 4.54 | 4.56 |
| Rectangles from intensity | 5.50 | 3.30 | 3.60 | 4.13 | 4.44 |
| Median filter | 10.70 | 8.70 | 1.38 | 1.40 | 2.00 |
| Sobel | 0.48 | 0.48 | 0.72 | 0.94 | 0.92 |
| Initial graph match | 0.42 | 0.24 | 0.22 | 1.22 | 1.38 |
| Match-strength probes | 9.10 | 2.64 | 2.86 | 13.60 | 13.50 |
| Hough probes | 15.30 | 0.96 | 1.68 | 23.30 | 25.80 |
| Result presentation | 2.60 | 2.26 | 2.52 | 2.24 | 2.26 |

TABLE XIX

Warp Validation Output

| Statistics | Sample | Test | Test2 | Test3 | Test4 |
|---|---|---|---|---|---|
| Total match-strength probes | 91 | 23 | 20 | 247 | 239 |
| Hough probes | 58 | 3 | 5 | 97 | 95 |

TABLE XX

Results for the Connection Machine on the Low-Level Portion

| Configuration | 8K | 16K | 32K | 64K |
|---|---|---|---|---|
| Total (low-level tasks only) | 1.26 | 0.91 | 0.71 | 0.63 |
| Overhead | 0.255 | 0.255 | 0.255 | 0.255 |
| Label connected components | 0.34 | 0.21 | 0.14 | 0.10 |
| Rectangles from intensity | 0.52999 | 0.38437 | 0.31506 | 0.25336 |
| Median filter | 0.082 | 0.041 | 0.025 | 0.015 |
| Sobel | 0.052 | 0.026 | 0.014 | 0.008 |

implementation comparison and the architectural comparison. (Since we are concerned here with architectures, we ignore the third type of comparison, software support, even though software tools can have an effect on performance. It can be assumed that each implementation was built with the most efficient tools available, and that the difference is therefore small.) An implementation comparison considers only the raw timings for running a particular software implementation of a benchmark on a particular hardware implementation of an architecture. Such a comparison is useful if one is looking to buy a machine in today's marketplace. However, it does not reveal which architecture is conceptually a better choice. When an architecture is constrained to a hardware implementation, numerous design decisions that affect its performance are made. Any change in any of those constraints, such as improvements in technology, increases in budget, and different size, weight, and power restrictions, will impact performance. Thus, one architecture may perform better than another simply because it has been committed to hardware at a later date, with more modern technology. Given equal technology, the other architecture might actually be superior.

An architectural comparison seeks to adjust benchmark timings for differences in implementation constraints. However, it is not fair to simply divide by the clock rate, list price, volume, shipping weight, and power consumption. When constraints change, there are many design choices that can be made. For example, moving to a denser integration technology makes it possible to add more processors or to make the existing processors more powerful. Furthermore, performance does not always scale directly with changes. Consider that adding processors may increase performance only on tasks with the greatest parallelism, so that performance increases just in proportion to the processor utilization of an application. This is precisely why simulation results and actual execution results should not be compared. An architecture that is still under simulation has yet to be fully constrained.

When a constraint is changed to be equal that of another system, the many possible outcomes cannot be captured by

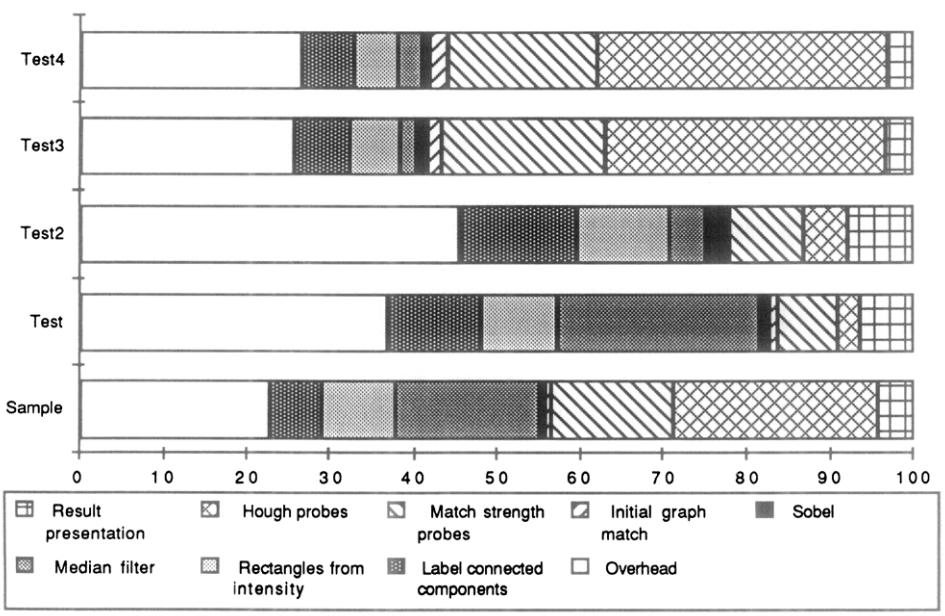


FIG. 12. Warp percentage of effort for each major subtask.

TABLE XXI

iPSC-2 Results for Median Filter and Sobel Steps

| | 1 | | 2 | | 4 | | 8 | | 16 | |
|---|---|---|---|---|---|---|---|---|---|---|
| Configuration | User | System | User | System | User | System | User | System | User | System |
| | | | | | Median Filter | | | | | |
| Sample | 176.47 | 0.00 | 87.93 | 11.52 | 43.46 | 11.23 | 22.27 | 3.10 | 11.14 | 3.82 |
| Test | 75.45 | 0.00 | 37.72 | 10.88 | 18.99 | 10.84 | 9.66 | 3.15 | 4.84 | 3.87 |
| Test2 | 60.84 | 0.00 | 30.36 | 11.48 | 15.25 | 11.45 | 7.63 | 3.73 | 3.81 | 4.19 |
| Test3 | 60.83 | 0.00 | 30.36 | 11.12 | 15.25 | 11.23 | 7.63 | 3.49 | 3.82 | 4.03 |
| | | | | | Sobel | | | | | |
| Sample | 78.63 | 0.00 | 39.32 | 3.53 | 19.68 | 3.00 | 9.84 | 2.37 | 4.92 | 2.91 |
| Test | 80.82 | 0.00 | 40.42 | 3.47 | 20.25 | 2.89 | 10.15 | 2.43 | 5.10 | 2.82 |
| Test2 | 80.82 | 0.00 | 40.42 | 1.46 | 20.25 | 1.99 | 10.15 | 1.87 | 5.10 | 2.50 |
| Test3 | 78.63 | 0.00 | 39.31 | 2.62 | 19.68 | 2.51 | 9.84 | 2.17 | 4.92 | 2.69 |

a single formula. In making any such adjustment, one is assuming that the system designers would make a specific design choice given a change in constraints. In some cases the assumption is unjustified. Even if the assumption is justified, one must be careful in estimating the effects. The process of adjusting a constraint and making an assumption about the resulting effects on a real architecture essentially turns it into a new, hypothetical architecture. Thus, without detailed knowledge of the hardware, a simple formulaic adjustment can reduce the validity of benchmark timings to just slightly better than back-of-the-envelope estimates.

As an example, we consider an architectural comparison of the Connection Machine, ASP, and IUA running the benchmark on data set Sample (it is assumed that this is the data set used to obtain the Connection Machine timings). For the sake of this example, we also make the invalid assumption that the simulated timings are as accurate as those for actual execution. These three architectures were chosen because, for the low-level portions of the benchmark, they have enough architectural similarity to be compared without too many distracting complications.

Table XXII compares the total times for the three architectures. The line labeled "Raw total" would be an implementation comparison. If all three machines actually existed, the Connection Machine would be the slowest. However, each of these machines has a different clock rate. For the

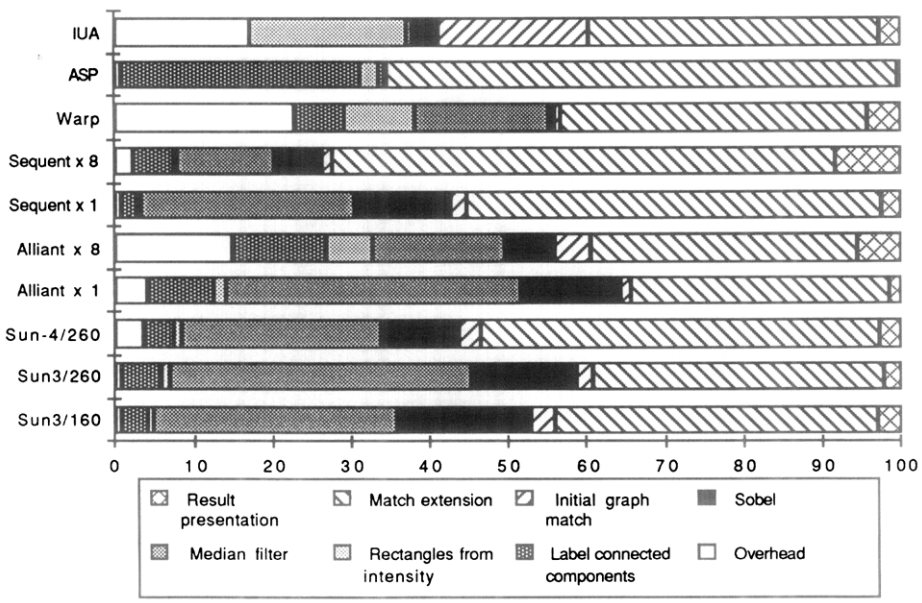


FIG. 13. Distribution of processing time for data set Sample.

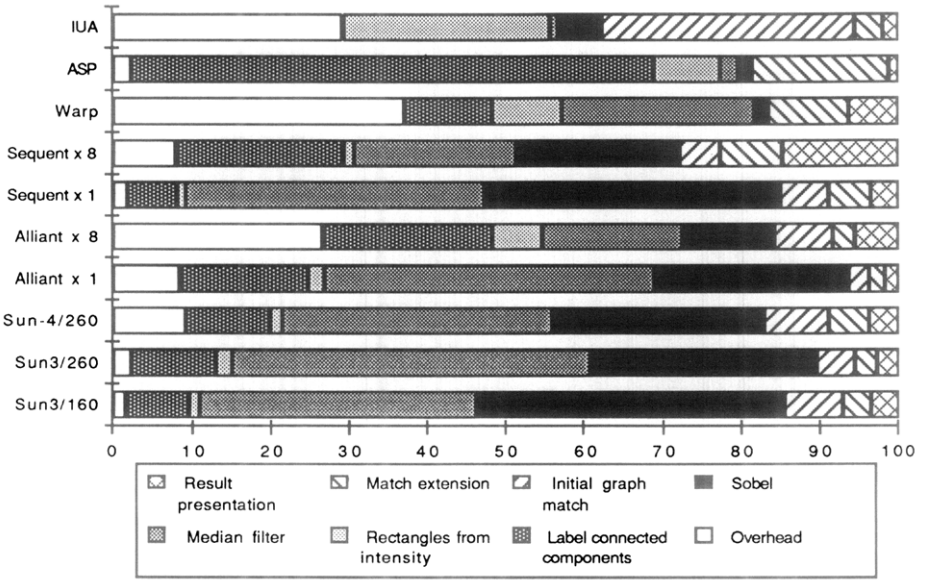


FIG. 14. Distribution of processing time for data set Test.

Connection Machine, the rate is 4 MHz; for the ASP, 20 MHz is assumed; and for the IUA, 10 MHz is assumed. The second line of the table shows the times normalized to a 10-MHz clock rate. This is perhaps the simplest adjustment that can be made to these data. It assumes only that, with improved technology, there is no reason that the Connection Machine clock cannot speed up by a factor of 2.5. The time shown for the Connection Machine is for 64K processors, while the other two machines have four times that number. We could simply multiply by four, but from Table XX, it is clear that the total time for the Connection Machine does not scale up linearly with the number of processors. The third line of Table XXII extrapolates the 256K-processor performance from the data in Table XX, again with a normalized clock.

The last row of figures in the table may seem like a reasonable comparison, but consider that the Connection Machine has 32-bit floating-point coprocessors, while the ASP and IUA do not. What would the performance of the ASP and IUA be if coprocessors were added to them? Also, the

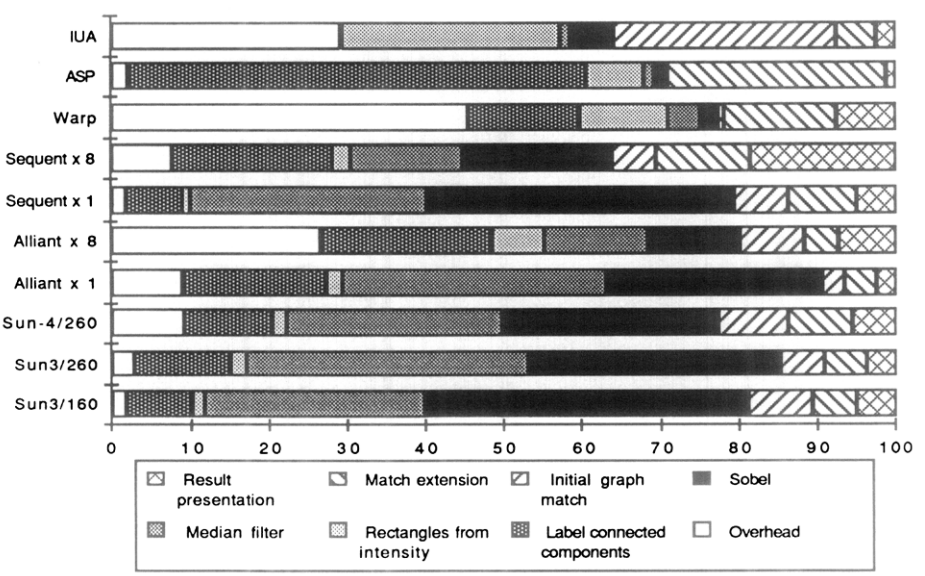


FIG. 15. Distribution of processing time for data set Test2.

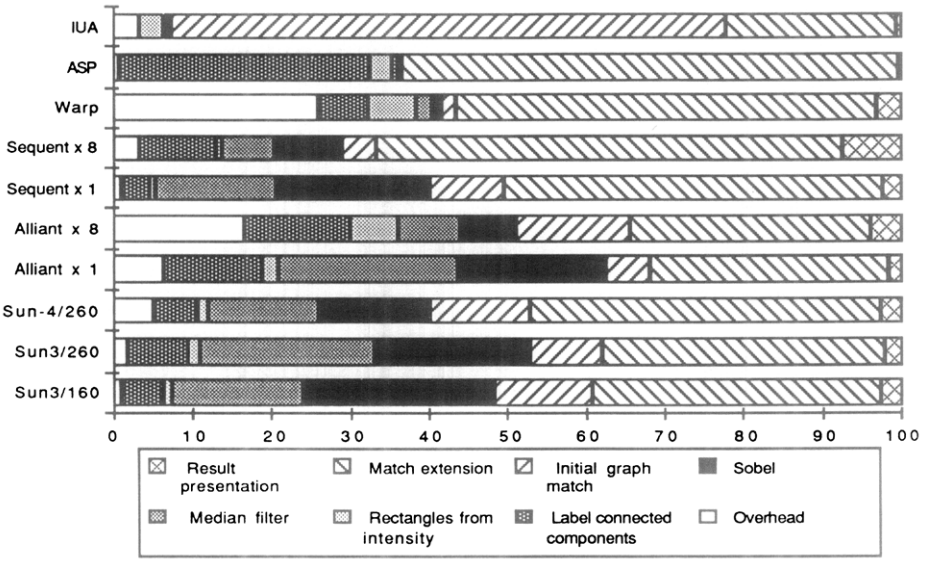


FIG. 16. Distribution of processing time for data set Test3.

ASP is to be built in custom wafer-scale integration, the IUA in custom VLSI, and the Connection Machine in gate array technology. What design changes would occur if their technologies were made equivalent?

Architectural comparisons cannot be taken very far, even for similar architectures, before the hypothetical questions begin to dominate the analysis. Consider the difficulty in comparing radically different architectures such as the Alliant FX-80 and the Connection Machine. However, the more data there are available, the further the analysis can be taken. For example, running the benchmark on a CM-2 without the floating-point option would answer one of the preceding questions. The example considers only the total time for the low-level processing, but a detailed examination of subtask times can reveal patterns that would be more informative. Thus, the greater the variety of processing in a benchmark, and the more extensive its instrumentation requirements, the farther the architectural comparisons can be taken.

## RECOMMENDATIONS FOR FUTURE BENCHMARKS

At the conclusion of the Avon workshop, a panel session was held to discuss the benchmark, the ways it could be

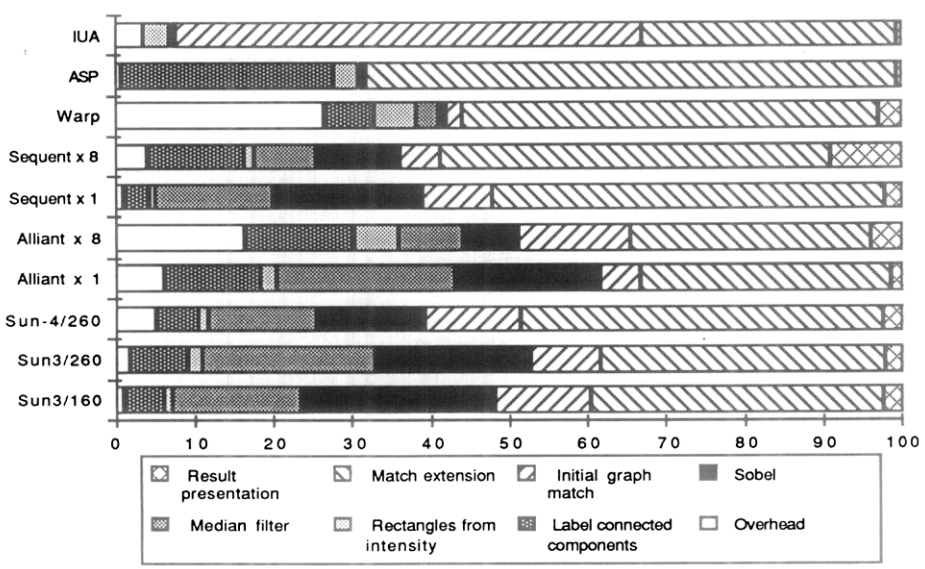


FIG. 17. Distribution of processing time for data set Test4.

TABLE XXII

Example of an Architectural Comparison

| Architecture (Low-level tasks only) | CM-2 | ASP | IUA |
|---|---|---|---|
| Raw total | 0.630 | 0.04489 | 0.03343 |
| Normalized clock | 0.252 | 0.08977 | 0.03343 |
| Normalized clock and processor count | 0.228 | 0.08977 | 0.03343 |

*Note.* Based on invalid assumptions; not for actual comparison purposes.

improved, and future efforts. The general conclusion of the participants was that the benchmark is a significant improvement over past efforts, but there is still work to be done.

One complaint concerned the size and complexity of the benchmark solution. The sample solutions help, but much work is still required to transport them to parallel architectures. Several people felt that a FORTRAN version should be made available so that the benchmark would be taken up by the traditional supercomputing community. Another comment was that most groups do not have the resources to implement such a complex benchmark, and it would be almost impossible to tune its performance as is done with smaller benchmarks. A counterargument that most vision applications are not highly tuned and that the benchmark might therefore give a more realistic indication of the performance that could be expected was voiced. Suggestions for reducing the size of the benchmark included removing either the match-strength or the Hough probe (although there was no consensus on which one to remove) and simplifying the graph-matching code through increased generality.

Several people complained that the benchmark data sets were too small. The groups that had benchmarked data-parallel systems all indicated that they would like to see data sets involving thousands of models so that they could exploit more data parallelism, rather than being forced into a task-parallel model. Of course, those who had benchmarked multitasking systems took the opposite view. It was suggested that the benchmark should provide a range of data sets with model bases ranging through several orders of magnitude. Such data sets would provide another dimension to the performance analysis and thus some insight into the range of applications for which an architecture is appropriate. Beyond simply increasing the size of the model base, several of the vision researchers expressed a desire to see a broader range of vision tasks in the benchmark. For example, motion analysis over a succession of frames would test an architecture's ability to deal with real-time image input and would help to identify those architectures with a special ability to pipeline the stages of an interpretation. However, there was an immediate outcry from the implementors that the benchmark is already too complex. It was then suggested that an optional second level of the benchmark that would be based on the basic task, but extended to include image sequences and motion processing, could be specified.

An important observation was that the complexity of the benchmark was not the issue; rather it was the cost of implementation. It was suggested that the benchmark might be more palatable if it was reorganized into a standard set of general-purpose vision subroutines. Even though a group might have to implement all of those routines, it would then have a library that could be used for other applications, over which it could amortize the cost. The benchmark specification would then be a framework for applying the library to solve a problem and could involve separate tests for evaluating the performance and accuracy of the individual subroutines.

Part of the discussion focused on the fact that the benchmark does not truly address high-level processing. However, as the benchmark designers were quick to point out, there is no consensus among the vision research community as to what constitutes high-level processing. Until agreement can be reached on what types of processing are essential at that level, it will be pointless to try to design a benchmark that includes it. The current benchmark has a well-defined task of model-directed processing as its high-level component. It was also noted that the current top-down direction of low-level processing by the benchmark has some of the flavor of the high-level control of intermediate- and low-level processing, which many researchers feel is necessary. It was decided that the community is not yet ready to define high-level processing to the degree necessary for a benchmark to be built around it.

Another point was that a standard reporting form should be developed, and that the sequential solution should output its results to match that form. Although the benchmark specification included a section on reporting requirements, the sequential solution did not precisely conform to it (partly because the reporting requirements included aspects of the implementation beyond the timings and statistics that were to be output). In fact, most of the groups followed the example of the reporting format for the sequential solution, rather than what was requested in the specification. It was also noted that because the benchmark allows alternate methods to be used whenever dictated by architectural considerations, the reporting format cannot be completely rigid.

The conclusion of the panel session was to let the benchmark stand as specified for some period of time, to allow more groups to complete their implementations. Then a new version should be developed with the following features: It should be a reorganization of the current problem into a library of useful subroutines and an application framework. A set of individual problems should be developed to test each of the subroutines. A broader range of data sets should be provided, with the size of the model base scaling over

several orders of magnitude and perhaps a set of images of different sizes. The graph-matching code should be simplified and made more general purpose. A standard reporting format should be provided, with the sample solutions generating as much of the information as possible. Lastly a second level of the benchmark that extends the current problem to a sequence of images with motion analysis might be specified. The second level would be an optional exercise that could be built on top of the current problem to demonstrate specific real-time capabilities of certain architectures.

## CONCLUSIONS

The DARPA Integrated Image Understanding Benchmark is another step in the direction of providing a standard exercise for testing and demonstrating the performance of parallel architectures on a vision-like task. While not perfect, it is a significant improvement over previous efforts in that it tests performance on a wide variety of operations within the unifying framework of an overall task. The benchmark also helps to eliminate programmer knowledge and cleverness as a factor in the performance results, while providing sufficient flexibility to allow implementors to take advantage of special architectural features.

Complete implementations have only been developed for a handful of architectures. In the meantime, it is possible to draw a few general conclusions from the data that have been gathered. Tremendous speedup is possible for the data-parallel portions of the interpretation task. However, almost every architecture in this sample devoted the majority of its overall time to the model-matching portion of the benchmark on data sets involving complex models. One conclusion might be that this portion of the task simply does not permit the exploitation of much parallelism. However, when the model-matching step is viewed at an abstract level, it appears to be quite rich with potential parallelism, but in the form of task-parallel direction of limited data-parallel processing. While this style of processing can be sidestepped by increasing the size of the model base so that the entire task becomes data parallel in nature, the inclusion of more complex and realistic high-level processing brings us back to dealing with this processing model. Thus, one potential area for research that the benchmark points out is the development of architectures, hardware, and programming models to support task parallelism which can direct data-parallel processing in a tightly coupled manner.

A benchmark can be used to make either implementation or architectural comparisons. Implementation comparisons can be made for any benchmark data and are primarily useful for making purchasing decisions regarding contemporary machines. Architectural comparisons require that a benchmark include a wide variety of processing and a rich set of instrumentation. Even then, architectural comparisons must be made with great care and an understanding of the potential for misleading results.

## ACKNOWLEDGMENTS

We thank Claire Bono, Chris Brown, C. H. Chien, Larry Davis, Todd Kushner, Ram Nevatia, Keith Price, George Reynolds, Lew Tucker, and Jon Webb for their many helpful comments and suggestions in response to the draft benchmark specification. For their efforts in the designing, programming, and debugging of the sequential and Sequent Symmetry solutions to the benchmark, we thank Poornima Balasubramaniam, Sunit Bhalla, Chris Connolly, John Dolan, Martin Herbordt, Michael Scudder, Lance Williams, and especially Jim Burrill. For providing benchmark results on other architectures, we also thank Alok Choudhary, R. M. Lea, A. Krikelis, I. Kossioris, Lew Brown, Dan Mezynski, Jon Webb, Lew Tucker, Steven Levitan, Mary Jane Irwin, Sunit Bhalla, Martin Herbordt, Michael Scudder, Michael Rudenko, and Jim Burrill. And, for their many suggestions for improvements to the benchmark, we again thank all of the above, plus Jake Aggrawal, Thomas Binford, Martin Fischler, Prasanna Kumar, Daryl Lawton, Randall Nelson, Karen Olin, Dennis Parkinson, Tomaso Poggio, Tony Reeves, Arnold Rosenberg, and Myung Sunwoo.

## REFERENCES

1. Carpenter, R. J. Performance measurement instrumentation for multiprocessor computers. Report NBSIR 87-3627, U.S. Department of Commerce, National Bureau of Standards, Institute for Computer Sciences and Technology, Gaithersburg, MD, Aug. 1987.
2. Choudhary, A. N. Parallel architectures and parallel algorithms for integrated vision systems. Ph.D. Thesis, University of Illinois, Urbana–Champaign, Aug. 1989.
3. Duff, M. J. B. How not to benchmark image processors. *In* Uhr, L., Preston, K., Levialdi, S., and Duff, M. J. B., (Eds.). *Evaluation of Multicomputers for Image Processing.* Academic Press, Orlando, FL, 1986, pp. 3–12.
4. Hillis, D. W. *The Connection Machine.* MIT Press, Cambridge, MA, 1986.
5. Kung, H. T., and Menzilcioglu, O. Warp: A programmable systolic array processor. *Proc. SPIE Symposium* Vol. 495, *Real-Time Signal Processing VII,* Aug. 1984.
6. Lea, R. M. ASP: A cost-effective parallel microcomputer. *IEEE Micro.* **8,** 10 (Oct. 1988), 10–29.
7. Preston, K. Benchmark results: The Abingdon Corss. *In* Uhr, L., Preston, K., Levialdi, S., and Duff, M. J. B., (Eds.). *Evaluation of Multicomputers for Image Processing.* Academic Press, Orlando, FL, 1986, pp. 23–54.
8. Preston, K. The Abingdon Cross. *IEEE Comput.* **22,** 7 (July 1989), 9–18.
9. Rosenfeld, A. R. A report on the DARPA image understanding architectures workshop, *Proc. 1987 DARPA Image Understanding Workshop.* Morgan Kaufmann, Los Altos, CA, 1987, pp. 298–302.
10. Sunwoo, M. H., and Aggarwal, J. K. VisTA: An image understanding architecture. *In* Prassana Kumar, V. K., (Ed.). *Parallel Architectures and Algorithms for Image Understanding.* Academic Press, San Diego, CA, 1990.
11. Weems, C. C., Riseman, E. M., Hanson, A. R., and Rosenfeld, A. An integrated image understanding benchmark, recognition of a $2\frac{1}{2}$ D "Mobile." *Proc. 1988 DARPA Image Understanding Workshop.* Morgan Kaufmann, Los Altos, CA, 1988, pp. 111–126.
12. Weems, C. C., Riseman, E. M., Hanson, A. R., and Rosenfeld, A. A

computer vision benchmark for parallel processing systems. *Proc. Third International Conference on Supercomputing.* International Supercomputing Institute, St. Petersburg, FL, 1988, Vol. III, pp. 79–94,

13. Weems, C. C., Hanson, A. R., Riseman, E. M., and Rosenfeld, A. A parallel processing benchmark for vision. *Proc. 1988 IEEE Conference on Computer Vision and Pattern Recognition,* 1988, pp. 673–688.
14. Weems, C. C., Levitan, S. P., Hanson, A. R., Riseman E. M., Shu, D. B., and Nash, J. G. The image understanding architecture. *Internat. J. Comput. Vision* **2** (1989), 251–282.
15. Weems, C. C., Hanson, A. R., Riseman, E. M., and Rosenfeld, A. A report on the DARPA integrated image understanding benchmark exercise. *Proc. 1989 DARPA Image Understanding Workshop.* Morgan Kaufmann, Los Altos, CA, 1989, pp. 165–192.

---

CHARLES WEEMS received the B.S. and M.A. degrees from Oregon State University in 1977 and 1979, respectively, and the Ph.D. from the University of Massachusetts at Amherst in 1984. All degrees are in computer science. Since 1984 he has been a research assistant professor and director of the Parallel Image Understanding Architectures Research Group at the University of Massachusetts. His research interests include parallel architectures to support low-, intermediate-, and high-level computer vision, benchmarks for vision, parallel programming languages, and parallel vision algorithms. He is also the coauthor of a widely used introductory computer science text.

EDWARD RISEMAN received the B.S. degree from Clarkson College of Technology in 1964 and the M.S. and Ph.D. degrees in electrical engineering from Cornell University in 1966 and 1969, respectively. He joined the Computer and Information Science Department as assistant professor in 1969, has been a full professor since 1978, and served as chairman of the department from 1981–1985. Professor Riseman has conducted research in computer vision, artificial intelligence, learning, and pattern recognition. He is the director of the Laboratory for Computer Vision Research and directs projects in knowledge-based scene interpretation, motion analysis, mobile robot navigation, and parallel architectures for real-time vision. The analysis of static and dynamic images is being applied to a variety of domains including natural outdoor scenes, biomedical images, industrial robotic environments, aerial images, and satellite images.

ALLEN HANSON received the B.S. degree from Clarkson College of Technology, Potsdam, New York, in 1964, and the M.S. and Ph.D. degrees from Cornell University in 1966 and 1969, respectively, all in electrical engineering. He is a full professor in the Computer and Information Science Department and associate director of the Computer Vision Laboratory at the University of Massachusetts at Amherst. For the past 15 years, his research efforts have been in artificial intelligence, computer vision and image understanding, and pattern recognition. He is coeditor of Computer Vision Systems (Academic Press, 1978) and Vision, Brain, and Cooperative Computation (MIT Press, 1987), coauthor of Fundamentals of the Computing Sciences (Prentice-Hall, 1978), and an editorial board member of several journals. He is a founder of Amerinex Artificial Intelligence Corp. and VI Corp.

AZRIEL ROSENFELD is a tenured research professor and director of the Center for Automation Research, a department-level unit of the University of Maryland in College Park. He also holds affiliate professorships in the Departments of Computer Science and Psychology and in the College of Engineering. He holds a Ph.D. in mathematics from Columbia University (1957), a rabbinic ordination (1952), a Doctor of Hebrew Literature degree (1955) from Yeshiva University, and an honorary Doctor of Technology degree from Linkoping University, Sweden (1980) and is a certified Manufacturing Engineer (1988). He is a widely known researcher in the field of computer image analysis. He wrote the first textbook in the field (1969), was a founding editor of its first journal (1972), and was cochairman of its first international conference (1987). He has published over 20 books and over 400 book chapters and journal articles and has directed over 30 Ph.D. dissertations.

Received November 6, 1989; revised July 25, 1990; accepted July 25, 1990