APPLYING ADVANCED PARALLEL PROCESSING CONCEPTS TO RADAR SIMULATION AND IMAGE GENERATION

Edward W. Drew and Ron Matusof
CAE-Link Corporation
Binghamton, New York

ABSTRACT

This paper discusses advanced parallel processing concepts and their use for radar simulation and image processing. It describes both the advantages and disadvantages of a number of architectures and illustrates these with actual implementations. It discusses issues relevant to real-time image generation, including latency, synchronization, and scheduling dispersion. It also discusses the problems inherent in designing state-of-the-art systems in a research and development environment, and then applying that product to an evolving market. Finally, it makes recommendations concerning future directions in parallel processing and simulation.

ABOUT THE AUTHORS

Edward W. Drew is a senior staff engineer with the CAE-Link Corporation, Binghamton, New York, and has 25 years of experience in tactical and radar simulation. He is currently assigned as systems engineer for the pDRLMS product line, with responsibility for radar simulation for several flight training devices. Mr. Drew holds a MS in Physics from the State University of New York at Binghamton, a D.C.Ae in Aviation Electronics from the Cranfield Institute of Technology, England, and a B.Sc in Physics from the University of Bristol, England.

Ron Matusof is a staff engineer with CAE-Link Corporation, Binghamton, New York. He has over ten years of experience in tactical, radar, and image simulation and simulator networking. He is currently acting as Principal Investigator for CAE-Link's Transputer Modular Processing System. Mr., Matusof holds a Bachelor of Science in Electrical Engineering from the University of Pittsburgh. He has published several papers on the subjects of Interoperability, mission rehearsal, and cue correlation.

APPLYING ADVANCED PARALLEL PROCESSING CONCEPTS TO RADAR SIMULATION AND IMAGE GENERATION

Edward W. Drew and Ron Matusof
CAE-Link Corporation
Binghamton, New York

INTRODUCTION

Throughout the history of simulation, one of the most complex problems has involved the synthesis and generation of imagery that represents the simulated terrain. This imagery, whether it is for out-the-window viewing, simulation of infrared (IR) for night vision operations, or simulation of radar imagery, has usually required large processing capability and expensive, custom designed hardware and software.

In the last few years, three conflicting forces have radically changed the approach to the simulation of imagery. First, real-world equipment (such as IR sensors and radars) has become significantly more complex. This has tended to drive up the complexity of image simulations. At the same time, competitive pressure has forced a decline in the price of image simulations by at least one order of magnitude, and we can expect this trend to continue in the future. Finally, as defense budgets continue to decline globally, systems originally designed for military use are being converted into more commercial applications, which tends to both add competitive cost pressure and at the same time increase performance requirements. The net result is that the simulation of imagery during the mid 1990's will have to perform more detailed computations across a wider number of applications, and at a significantly lower cost than systems designed just ten years ago.

In 1988, we began work on a data flow architecture for multi-mode radar simulation.[1] This work was performed for four years under research and development funding with the intended goal of producing a radar simulation system with increased processing capability and an order of magnitude reduction in recurring cost.

We were interested in applying parallel processing techniques to lower the cost and increase the performance of our simulation. Parallel processing allows for increased computational performance at significantly lower recurring costs, but carries with it a set of unique design methodologies and constraints. The architecture we describe in this paper has been applied to a variety of Navy, Air Force, and International Digital Radar Land Mass Simulation (DRLMS) programs. Additionally, the same architecture (and, in fact, the same hardware design) supports US Army helicopter combat training by providing environmental feedback information.

In this paper, we discuss our approach to parallel processing, including both the advantages and disadvantages we have discovered after six years of product development, testing, and fielding.

PROBLEM DEFINITION

The basic ideas behind most radar simulations involve simulating the signal at various stages of its life, including: its emission by the radar, its propagation, effects from reflection off terrain and cultural surfaces, the signal's return propagation, and the radar internal signal processing characteristics. The degree to which the signal is simulated at each stage determines the overall fidelity of the radar simulation.

High fidelity radar simulation is an extremely complex undertaking. The equations that govern the radar signal propagation (namely Maxwell's equations) are not well suited to operate in discrete time steps, like those found in most simulations. On the other hand, the effects of the signal propagation are either totally independent (as when terrain is illuminated) or additive (for example, when the received signal is processed). The independent and additive nature of radar makes it a primary candidate for parallel processing.

Similarly, the problems faced by the designer of image generators are highly complex and require large processing capabilities (many estimates range from one billion to one trillion floating point operations per second). One subset of the image generator is the function of environmental feedback. Environmental feedback refers to the

processing that provides information concerning the interactions between any two points in the simulated environment (such as line of sight, collision detection, surface attributes, etc.). These interactions are usually independent and we find that they are ideally suited to parallel processing.

Our intent was develop a parallel processing architecture that could apply to a wide variety of complex applications including radar simulation, image generation, acoustic simulation, acoustic analysis, and the like. Although we have been generally successful in our system development, we have learned some interesting lessons concerning parallel processing, radar simulation, data base manipulation, and product development using state-of-the-art techniques and hardware.

PARALLEL PROCESSING

Conventional computer processing is a sequential task where program execution occurs in a pre-defined order and operates under the control of a central processor. Although the speed of conventional computer processors continues to improve dramatically, the highest processing throughput attainable is still limited by the speed of the central processor, its ability to access memory, and the ability to move data between the processor and input/output ports.

Recent enhancements to serial processors have included a super-scalar architecture, which, in theory, allows several instructions to be executed simultaneously. In practice, there are only particular combinations of instructions that execute simultaneously, such as operand and instruction fetch, so performance improvements vary greatly between applications.

Parallel processing uses a different approach to improve the processing power of a computational architecture. Rather than using the conventional von Neumann model of a computational machine, parallel processing divides the problem into a number of individual tasks that are processed independently. In many applications, the problem is divided across multiple processors where the individual processor is still a von Nuemann machine operating in serial fashion. Although this approach provides significantly greater overall processing power, it is still constrained by the bandwidth of the interconnection between the individual processors.

Another approach to parallel processing divides the tasks into a number of software processes and interconnects these processes through a data flow path known as a link. This approach separates the processing architecture (a software function) from the physical architecture (a hardware function). The software design does not constrain itself to a von Nuemann architecture (even if housed on von Nuemann computational platforms) and significant processing improvements are theoretically possible. The overall system performance, however, is limited by the choice of computational processors, the mapping of software processes and links to these processors, and the physical interconnection of the processors.

ADVANCED CONCEPTS

After a great deal of research, we chose to implement a prototype programmable DRLMS (pDRLMS) on an interconnected network of Inmos (now SGS-Thompson) T-800 transputers. Transputers are powerful 32 bit processors that support high level language development, parallel on-chip processing, and include four high speed data link connections to other transputers. In 1988, when we started this program, these processors were among the fastest processors on the market. They were also relatively new to the market and very few applications had been implemented on them.

The pDRLMS uses a data-flow architecture and its design attempts to separate the software implementation from the hardware architecture. Data flow architectures are those in which the messages that flow between nodes provide control and synchronization of the system. Software tasks are divided into processes that communicate with each other via virtual links. A virtual link does not necessarily have a corresponding physical link, and the mapping of processes (software tasks) to processors (hardware nodes) is usually not one-to-one.

Traditional DRLMS applications involve some form of pipeline, where each process acts as a worker on an assembly line. Data comes as input from the previous worker, processed, and then output to the next worker. Many attempts to design parallel

DRLMS implementations have involved making parallel pipelines, so that numerous processes occur concurrently. Unfortunately, the start of the pipeline (data transfer from the host computer) and the end of the pipeline (display of imagery to the crew) can not be broken into multiple parallel tasks, and these become bottlenecks in the system.

A different problem occurs for applications that involve large data base processing capabilities, such as environmental feedback calculations. In these cases, the problem can be decomposed until there is a one-to-one mapping between data base polygons and software processes. In this extreme case, the benefits of parallel implementations are lost to the overhead of communicating between a large number of processes.

The degree to which a problem space is made parallel is known as its granularity. Coarse grain parallelism occurs when the problem is broken into very large pieces. Fine grain parallelism occurs when the problem is broken into very small pieces (such as a small block of sequential code). When a problem is too coarsely parallel, throughput suffers since there is a large amount of sequential processing in each software process. When the problem is too finely parallel, system performance degrades due to an increase in message traffic between the processes. We therefore chose a methodology of decomposition that optimized the granularity for the most efficient processing architecture. This method is iterative, and attempts to decompose the problem in four ways:

1. Functional Decomposition. During functional decomposition, large functions that operate independently are identified. For example, the major functions in a radar simulation might be data base manipulation, radar illumination, radar effects, and radar image generation.

2. Domain Decomposition. During domain decomposition, the data which is to be processed is examined to determine if there are subsets (domains) which can be conveniently grouped to provide increased processing efficiency. The domains required by each major function are then identified. Domain decomposition is very useful in reducing the amount of processing a single node must perform.

3. Farming. A farm is an architectural concept where each processor performs the whole task on a portion of the domain. This scheme allows scaling of the processing. The processing power can be increased by increasing the size of the farm.

4. Pipelining. A pipeline is the antithesis of a farm in that each processor performs a portion of the task on the whole domain. In many applications, there is great efficiency gained by breaking a single serial task into a set of smaller tasks operating in a pipeline.

Once the problem has been fully decomposed, the required replication of tasks is determined. The appropriate number of times a process, farm, or pipeline is replicated is a function of the desired throughput of the system, the available resources, the desired traffic on the network, and the degree of parallelism inherent in the problem space. This represents a delicate balance between system performance, system cost, and system reliability and these decisions are better made on a case-by-case basis.

The decomposition of a simple DRLMS is shown in Figure 1. Although the decomposition in the previous paragraphs describes functional decomposition, the methodology described works equally well for other decompositions, such as object-based or data-driven decompositions.

Once the software processes have been identified, they are mapped into a software architecture known as a *virtual network*. The virtual network describes the interconnection of individual processes and the communication and data path between them. There is no requirement for the virtual network to map one-to-one with a *physical network*. It is because of this separation of virtual and physical networks that it is possible for a process to have large numbers of virtual connections, while the host processor has only four physical connections.

Our architecture is based upon a virtual intercommunication scheme which make

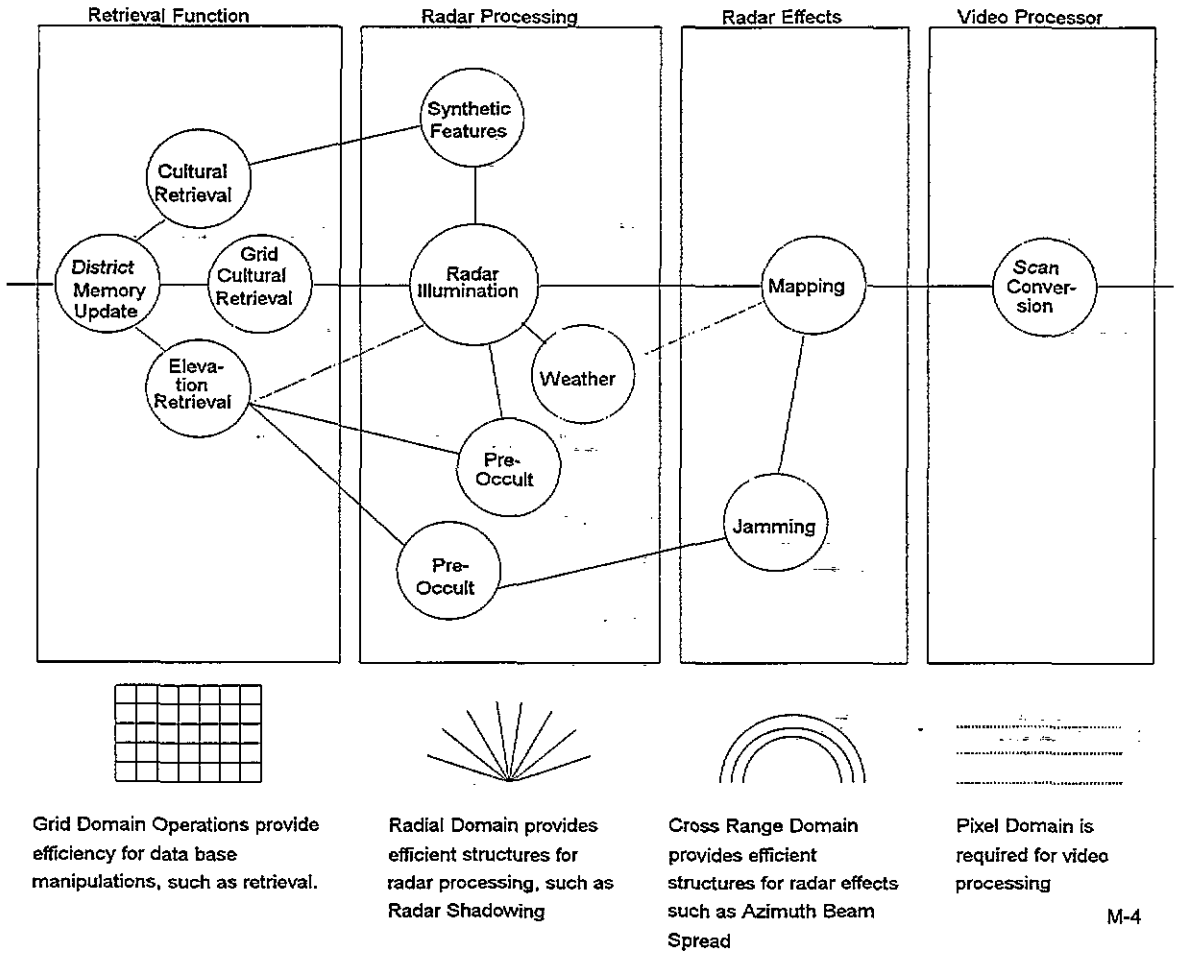

Figure 1 Simple DRLMS Decomposition
Processing is divided into functions and domains.

the physical implementation of the network transparent to the application software. This capability is provided by a software package known as the message handler. The message handler resides on each physical node and it acts as a postman, sending messages to the correct destination, and receiving and buffering incoming messages. Messages are received via any of the physical links and buffered. Once messages are received, the message handler re-transmits those messages destined for other nodes (a process referred to as ``through-routing'') and organizes the messages intended for this node. Based on the incoming message traffic, the message handler ``wakes up'' processes which have turned themselves off while waiting for data and assigns them to execute a particular task. The message handler also transmits messages created on this node by resident processes.

APPLICATIONS

Our first application of this architecture was in support of the AH-64 Combat Mission Simulator for the US Army. The architecture was used to implement a Terrain Information System (TIS). The TIS is an environmental feedback system which provides high-fidelity line-of-sight, elevation, and occultation calculations which are fully correlated to the simulator's image generator data base.

The TIS is a classic example of domain decomposition and farming. All information passed between the host computer and the TIS travels through a single process known as an Interface Manager. This process determines what information is required and which part of the network will provide it. The Interface Manager then requests that environmental feedback information be provided by one of the four farms comprising the TIS. Each farm is composed of one Executor, which is responsible for coordinating the activity of the farm, and fifteen model processors. Each model processor has a unique domain, consisting of a subset of the data base. The Model Processor operates upon its domain, with the Executor correlating the results and sending the information back to the Interface Manager for distribution to the host. As such, the problem of environmental feedback has been domain decomposed at the Model Processor level and farmed at the Executor level. The TIS architecture is shown in Figure 2.

Our second venture was a production version of our research and development pDRLMS. The pDRLMS is an example of all four methods of problem decomposition. The DRLMS was first decomposed into major functional processes of data base retrieval, illumination, radar characteristics, and network control.

Several domains became apparent at the outset. Three types of data base are required for the pDRLMS, namely gridded terrain, list-based culture, and weather descriptors. Although these could be viewed as separate domains, we opted to consider them as a single domain consisting of information which is defined in spatial terms (i.e., absolute position and attitude). Radar effects, on the other hand, tend to happen in a series of sectors emanating radially from the radar transmitter and coincident with the sweep pattern. We defined this domain to consist of information which is defined in radial terms (rotation and range) from the transmitter. Beamspread effects are a function of range and have very little contribution from rotation. We opted to define a third domain for beamspread which is based solely on range from the transmitter. A final domain was reserved for video generation, which requires information to be described in terms of pixel space.

With our major functions defined and our domains identified, we proceeded to determine where farming and pipelining would be beneficial. The result is shown in Figure 3.

ISSUES AND LESSONS LEARNED

Our architecture is asynchronous, and it brings with it certain concerns inherent with the asynchronous generation and use of data. The system is synchronized to ``real-time'' in at least one place: the interface with the host computer. In the case of pDRLMS applications, it is also synchronized to the frame rate of the radar display. Between these points, the system is synchronized only by the flow of messages. It is therefore possible to envision situations where data consistency across the network is not achieved. We had feared in certain applications that it might be possible to generate erroneous data due to data inconsistency, but thorough testing of the system

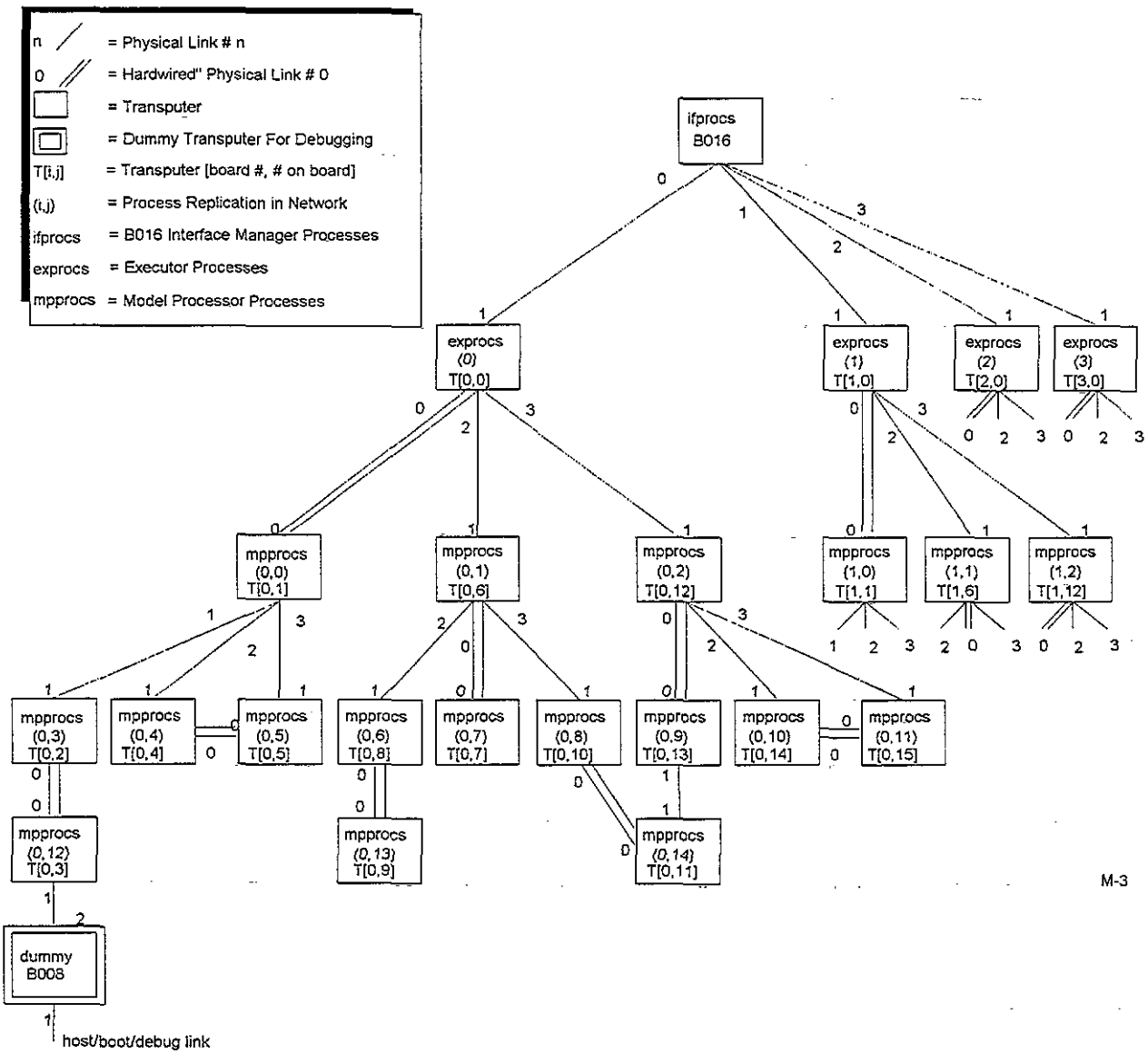

Figure 2 TIS Physical Network

Each Executor Processor Coordinates the activities on an individual farm of model processors.

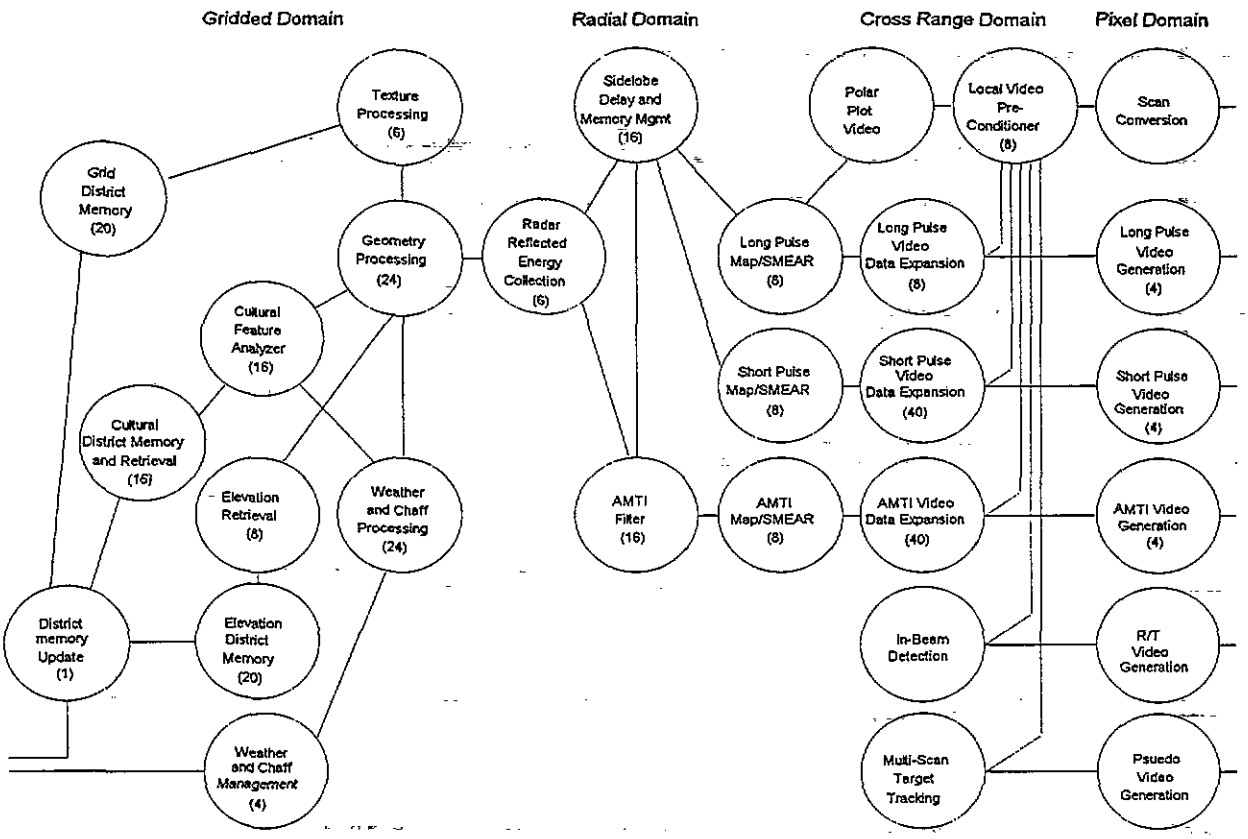

Figure 3 Simple DRLMS Decomposition

System has been decomposed into major functions and domains (see Figure 1) and is further decomposed into pipelines and farms. Numbers in parenthesis indicate the size of the processing farm (number of parallel processes).

by both CAE-Link engineers and our customers has failed to detect this problem.

Similarly, we had been worried that in very large networks, scheduling dispersion could potentially cause unsynchronized updates which would be perceivable by the crew, and again this has not been the case.

This is not to say that our experience with this development has been without problems. One of the first problems we encountered was the support for high level languages. Although compilers were available for both FORTRAN and C, the only truly supported compiler was provided for OCCAM. OCCAM was the first language to be based upon the concept of parallel and sequential execution, allowing for communication and synchronization between concurrent processes. Although the language is quite efficient and provides significant advantages for parallel processing, we have found that it is equally hard to find qualified OCCAM programmers. As the compilers for C and C++ have improved, and libraries have been added to support parallel execution, we have begun to reduce the amount of OCCAM in the system with the eventual goal of eliminating OCCAM entirely.

There were few applications hosted on T800 Transputers when we began this design effort in 1988. The size of our networks (between 64 and 400 physical nodes thus far) required special considerations in our design. We developed our message handling software largely because there was no commercially available alternative from which to choose. We therefore have invented a proprietary, closed architecture in a situation where we had desired a fully open architecture. As the use of transputers grows, commercial alternatives are becoming available and it is our goal to migrate to a commercial message handler system in the near future.

We also tended to find quirks in the prototype hardware with which we were working. To be fair, we were generally using pre-qualified hardware and beta releases of software tools and often found ourselves in the unenviable position of working around unanticipated hardware shortfalls.

Our method of decomposition appears to have worked quite well, although it still leaves a bottleneck at the interface to the host and at the output to a crew display. However, we have successfully reduced the cost and complexity of a typical radar application by over an order of magnitude and have implemented the TIS with only one board type and the pDRLMS with only four.

CONCLUSIONS

In general, we have found that our architecture, and most other parallel architectures, offer the following advantages over traditional approaches:

1. Scalability. Our system has been scaled to a factor of 12-to-1 increase in performance simply by adding processors, and we believe that a 32-to-1 increase is readily achievable.

2. Reconfigurability. Because the hardware, the communication software, and the application software are designed independently, a wide variety of applications can be hosted on this architecture.

3. Optimization. This architecture allows processes to be moved from processor to processor without affecting the hardware interface. This allows for easy load balancing, additional problem space decomposition even after the system has been built, and the ability to optimize by adding additional processes as needed.

We are firm believers in the concepts of parallel processing. There are many fine parallel processing architectures on the market, and we are not attempting to imply that this architecture is inherently better than any other. We believe that parallelism can be successfully exploited as a general purpose architecture to solve a variety of problems, and we propose our four-step method of problem decomposition as an attractive method of determining the appropriate level of granularity for a given problem space.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the contributions of the design engineers who created and developed the CAE-Link parallel processing architecture over its six-year life, and in particular, Curt Carlson, Gary Daniel, Peter Hunt, and Karl Lindberg. The authors also thank Harry Johnson and Ed Rotthoff for their comments.

REFERENCES

1 Hunt, P., and Carlson, C., "A Data Flow Architecture for Multi-Mode Radar Simulation", Interservice/Industry Training Systems Conference, Orlando, FL, November, 1988.