PARALLEL PROCESSORS FOR MILITARY TRAINING SYSTEMS - A LABORATORY PROOF-OF-CONCEPT MODEL

W. J. Rowan, D. M. Kotick,
M. W. Layton, C. E. Ruiz and C. N. Pope
Naval Training Systems Center
12350 Research Parkway
Orlando, FL 32826-3224

ABSTRACT

Military simulators require very large amounts of high speed computational power. Traditionally, the need has been met by assemblages of minicomputers. The laboratory project described was undertaken to explore the feasibility of employing low-cost microcomputers instead. The paper details the considerations dictating the architectural design, describes the partitioning tool used to assign modules to processors, and discusses methods employed to overcome real-time synchronization problems. Recommendations are presented for system features which would result in the flexibility and expandability required for this application.

INTRODUCTION

Sophisticated military training systems have a seemingly insatiable appetite for computing power. While today's procurement requirements can be met by aggregates of speedy minicomputers, demands for lower life-cycle expenditures and for greater realism are forcing a look at computational alternatives. Parallel microprocessors offer the promise of abundant processing power necessary for greater realism in increasingly complex warfare scenarios. Lower initial costs, easy expandability, and the opportunity for fault tolerance should contribute significantly to meeting the budgetary constraints. The project described in this paper was undertaken to examine the feasibility of employing arrays of microcomputers to meet the computational demands of such real-time systems and to examine the suitability of alternative architectural features in this application.

Before large scale use is made of this technology in military trainers, the potential advantages need to be confirmed and, to the degree possible, quantified. Similarly, disadvantages need to be identified and characterized. The most obvious question to be addressed is that of multiplicity of processing power as a function of number of elements. On the other side of the ledger is the issue of software development times which, intuitively, must be more extensive. Other concerns in the simulation environment are synchronization methods and real-time input/output capabilities.

LABORATORY PROJECT DESCRIPTION

To examine these issues, a real-time operational flight trainer (OFT) was designed using Motorola 68020-based microcomputers in place of more conventional minicomputers. The OFT is shown in Figure 1. Since the goal of the project was to investigate the feasibility of multi-micro architectures for training systems, a complete software development effort was deemed inappropriate. Instead, it was decided to use proven software from an existing T2-C aircraft simulation and modify it as required. The use of existing simulation software offered the added advantage of providing a basis of comparison for the project results. The aircraft model is a 6 degree-of-freedom, rigid body aircraft simulation using Euler explicit techniques for mathematical integration. Linear interpolation of functions of one and two variables is used extensively in both the aerodynamics and engine modules. The software package was originally developed as a mix of SEL (Systems Engineering Labs) Fortran and assembler languages. Complete conversion was made to FORTRAN using a VAX 11/780.

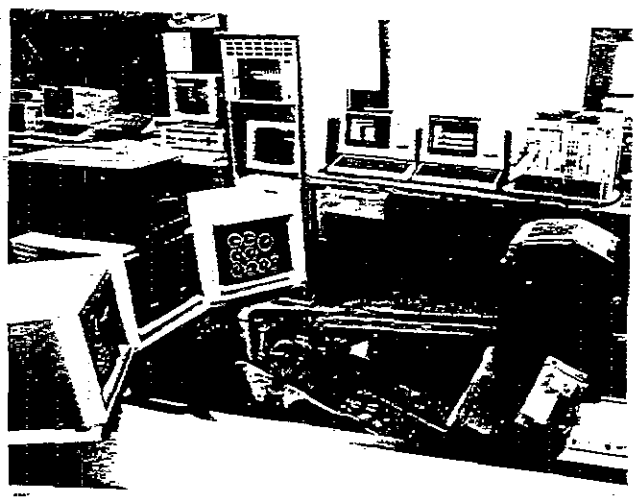

FIGURE 1. LABORATORY PROOF-OF-CONCEPT OPERATIONAL FLIGHT TRAINER

The architectural features were dictated by real-time requirements of the flight trainer which mandated a minimum computational task to be computed 30 times per second. The need for an efficient means of communications among computational elements led to the selection of the 32-bit, non-proprietary VME bus which also accommodated the system-wide real-time interrupt. Four Motorola MVME-130 boards provided the required processing power with a fifth serving as the controller which is responsible for maintaining the time base, handling exception processing, and interfacing with the instructor/operator. To realize the high computational power required, local access to memory-resident instructions was considered a necessity. VMX bus interfaces provided on the MVME-130's were used for local access to 1-megabyte dynamic ram boards. These memory boards are dual-ported, with the other port used by VME bus for interprocessor data.

Synchronization had to be addressed at two levels, the first of which involves the running of programs such as engines and weights-and-balances in a coordinated fashion so that differential times used in mathematical equations are accurate and that the results of each module accurately reflect dependencies on other modules. This requirement is met through the interaction of the control program and kernels resident in each of the application processors which execute one of six lists of modules upon receipt of the real time interrupt. The list to be executed is dependent upon a "frame" number passed to all applications processors in a control word accompanying the interrupt. An interesting problem arises in generating one interrupt which must be serviced by several VME-resident processors in that all would respond with an interrupt acknowledge cycle in response to the one interrupting device. The interrupting device (in this case the control processor) would have to be tailored to complete a number of interrupt acknowledge cycles exactly equal to the number of interrupt handlers on the bus. There would also be some adverse time skewing. To overcome these problems, use was made of the Multiprocessor Interrupt Request (MPIRQ*) line of the MVME-130 which permits the handling of IRQ1* via a Zilog 8036 timer port set up for input, resulting in the appearance of a local board interrupt (level 6) free of the VME handshake requirement. The same port, configured for output, is used by the interrupter to toggle IRQ1*.

The other synchronization issue is that of data coherency, a term used here to denote the requirement that elements of a definitive set be calculated in the same time frame. The term is applicable in flight simulation to transformation matrices, vectors, etc. In sequential simulations, coherency is assured whereas in concurrent processors, the user of a particular vector, for example, could access the complete vector in the middle of an update by another processor. Where coherency is vital, validity flags attached to the vectors are negated during the update process and checked before access.

A similar problem is that of data currency, in which there exists a mandatory sequence of calculations being performed by multiple processors. In this project, a semaphore mechanism was employed to prevent the usage of stale data. Careful attention had to be paid to idle times attributable to "computational skew." It was necessary in most cases to make empirical observations of module execution times and adjust module run orders accordingly. Predictive methods would, of course, be more desirable, but the use of a higher order language coupled with the pipelined 68020 makes the method hard to come by.

For the actual exchange of data, two potential methods were inherent in the chosen architecture. The first of these resulted from the accessibility of VME bus to all memory, which readily supported a tightly coupled shared memory approach. In fact, with each processor having dedicated VMX access to a subset of this total memory, parameters could be updated without the incurrence of VME bus penalties. Using processors would read via VME bus, however. In this scheme, then, each time a processor required a parameter, it would have to access VME with a possible arbitration cycle involved. To avoid the adverse impacts of this method on module execution times, especially as the number of processors increased, a message passing mechanism was employed. All required transfers were accomplished by the control processor during the interval between the completion of all applications software and the end of the time frame as signaled by the occurrence of the real time interrupt. Applications processors flagged the control processor upon completion of all required modules. Although two global bus accesses are required for each parameter, this method offers the advantages of efficient block moves and elimination of multiple VME accesses by each processor. With two applications processors, the required transfers consumed 1.87 milliseconds. This time increased to 2.20 ms for three processors and 2.9 ms for four processors. Although no other processors were available to enlarge this data base, it would seem that for this particular application, where 33 ms time frames must be honored, seven or eight processors would be a practicable limit.

PARTITIONING TOOL

Obviously, with the static partitioning scheme employed, a determination has to be made of the parameters which must be passed between processors. To provide automation of this function as well as the identification of critical timing relationships, a VAX-resident off-line partitioning tool was developed as part of this project. The tool was designed to result in a rather course grained parallelism in which the basic unit was a functional module of the operational flight trainer software. A major exception to the rule was the consigning of linear function interpolations to separate processors. For the architecture adopted, it was felt that finer grained parallelism would have overwhelmed the data transfer capabilities. Once the modules are identified and coded, the remaining tasks are to identify the input/output parameters and establish mandatory timing relationships.

The partitioning tool algorithm uses the VAX 11/780 Fortran compiler's cross reference information to automate the determination of module input and output parameters. To avoid oversights by the algorithm, it is first necessary to convert subroutines to in-line programming. The result of this step is a table listing each simulation parameter and the modules referencing it together with flags denoting input or output. The algorithm then determines the source of each module's input parameters and, conversely, all destinations of output parameters. The physical locations of parameters were ordered for efficient block transfers between processors. This ordering was accomplished using Fortran Common statements.

Establishment of timing relationships involves the examination of every pair of modules in the system with a data interface. Timing requirements are identified in terms of time frames (33 ms periods for this 30 iteration per second simulation) with the most critical data

element timing requirement determining the necessary relationship between the two modules. For example, if it is sufficient to pass one data element from Module A to Module B every tenth frame while another must be passed every frame, both are then transferred every frame. This analysis is unavoidable in training system applications if the fidelity of the simulation is to be maintained.

These tasks having been accomplished, modules must be assigned to the available applications processors in a manner which will not exceed the local memory capacity of any processor, will allow all processors to complete execution within the frame time, and accommodate the identified critical data timing relationships. The optimization of module assignments is an iterative process of assigning and reassigning until acceptable results are achieved. To facilitate this process, a program was written to grade the anticipated performance of a given partitioning scheme. This program generates a parameter and time interrelationship list for each module which serves as feedback for determining more optimal assignments.

SYSTEM PERFORMANCE

It was found in the performance of this project that the computer system constructed did not execute the simulation software as quickly as originally estimated. Estimates had been based on articles in technical periodicals suggesting processing power of at least 1.5 million instructions per second (MIPS) for the 68020. In the design process, proper consideration was not given to all the factors which go into the derivation of such figures. Actual performance measurements in the environment of this particular project point to an execution rate of approximately .75 MIPS. The most significant factor in this discrepancy is, perhaps, the use in this project of dynamic ram boards which introduced memory access wait states of 2 for VMX (1 clock cycle) and 4 for VME (2 clock cycles). Another contributing factor was the instruction mix which is rich in floating point operations for which the on-board Motorola 68881 was used. As an aggregate, neglecting end-of-frame interprocessor communications, the computing system was able to execute the simulation program in 62 ms, 38 ms, 26 ms, and 23 ms for one, two, three, and four processors, respectively. Computing power increased, then, from .75 MIPS for one processor to 1.22 for two, 1.79 for three, and 2.0 for four. The failure to realize a linear function is attributable to the coarse grain parallelism preventing perfect load balancing, the largest module consuming 23 ms. It is felt that with more work, this module and others could be subdivided to achieve somewhat better results. Of more significance are those figures which include interprocessor communications times and, as such, represent usable computing power. This real power increased from .75 MIPS for one processor to 1.16 for two, 1.65 for three, and 1.8 for three processors. These results are plotted in Figure 2.

The static partitioning/scheduling scheme was designed primarily to minimize associated overhead so that real time frame boundaries would not be exceeded. Major drawbacks are the lack of flexibility in moving modules among processors and the sheer effort involved in producing a working load. Another problem is the requirement to maintain two separate source programs, one representing the actual software and the other, with subroutines brought in-line, used for identification of module inputs and outputs. It can be seen that a major modification to a fielded training system could result in a very significant repartitioning effort. It would be very desirable in developing and debugging such a modification to have the flexibility to reassign modules on-line and be provided performance figures by the operating system. The implementation of such a capability would be very difficult in a system employing message passing for data sharing. In a system employing shared memory for this purpose, however, implementation problems should be manageable since reassignment of modules would not affect the physical locations of common variables.

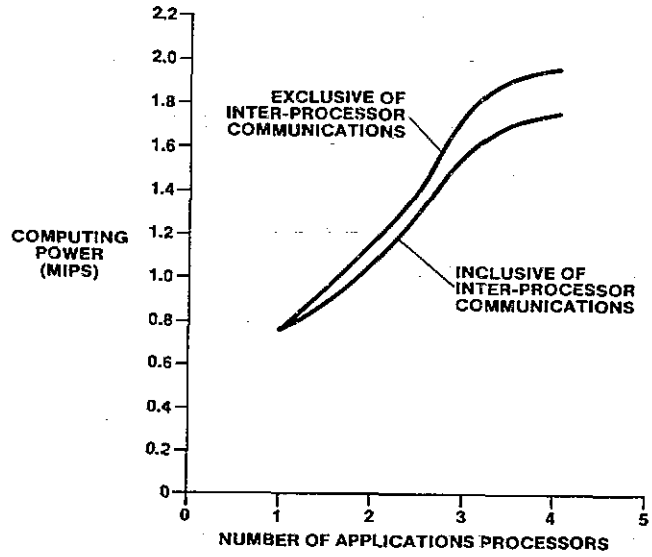

FIGURE 2. AGGREGATE COMPUTING POWER

AN ALTERNATIVE TO STATIC PARTITIONING

A more desirable method of assigning modules to processors in the simulation environment would be on the basis of availability. Here, as a processor becomes available, it executes the next task in the queue provided that all precedence conditions have been satisfied. Such conditions would be the execution of tasks which must precede the queue task. This method can be implemented by having all processors execute out of a single load in common memory which would also contain all data. Obviously, results would be completely processor-independent. Provision to all processors of fast instruction caches could reduce bus traffic to acceptable levels. Data caches could also contribute to this goal, but maintaining currency in a multiprocessor environment could be tricky. With the availability of very inexpensive memory, it becomes possible to provide a complete copy of all software to each processor's local memory.

Thus, execution of accepted tasks would not put any load on common memory busses which could be kept free for data accesses. In order to avoid loss of data as frame-to-frame execution of modules takes place on different processors, it would be necessary to COMMON all local variables. Retention of local data could also be assured by complete context switching, but in this real-time environment, the overhead penalties incurred could probably not be tolerated.

As in other real-time environments, downtime in training systems is very disruptive and needs to be kept to an absolute minimum. The dynamic task allocation scheme discussed offers the added advantage of inherent fault tolerance. After executing all simulation tasks, each processor could then perform confidence checks, the failure of which would be conveyed to the control processor which would take appropriate action. The other result of the failure would be that no further tasks would be accepted from the queue. Provided spare processing power is available, the failure would be transparent. A mechanism would have to be incorporated into the system for an "orderly transition of power" should the control processor fail.

SUMMARY

This project truly demonstrated the feasibility of using parallel microprocessors to satisfy the computational requirements of complex real-time training systems. The hardware architecture and software partitioning algorithms employed, while not optimal for achieving such advantages as flexibility in module reassignments, computing power expandability, and fault tolerance, did permit studies of these issues, thereby bringing into focus characteristics which should be possessed by any such system used in government training systems.

RECOMMENDATION

Perhaps the most important characteristic of a parallel microcomputer system used for simulation is its degree of expandability. There are numerous examples of undersizing of computer systems in simulator designs. Easily expanded systems would eliminate such problems as well as reduce the impact of major modifications to fielded systems. Keys to such expandability are the avoidance in the architectural design of bottlenecks such as shared-memory busses, the ability of the control algorithm to accommodate reassignments of software modules, and the inherent granularity of the simulation software. The achievable granularity is actually very dependent on the interprocessor communications capacity of the system since this requirement increases as modules are divided into ever smaller pieces. In reality, simulation does not lend itself to very fine grain parallelism (although certain aspects of visual simulation and digital radar landmass simulation might). Consequently, it is doubtful that massively parallel architectures would need to be employed. With this premise, the need to avoid bottlenecks seems to favor architectures employing dedicated communications paths between all pairs of microprocessors. Each microprocessor would operate out of local memory. The optimum operating system would facilitate module reassignments, possibly dynamically, and would incorporate fault tolerant features. This latter requirement would probably dictate a need for complete copies of all modules in each microcomputer's memory. Real-time I/O could be assigned to one or more processors which would communicate with applicable applications processors. Employing such an ideal machine in military training systems simulators would result in the realization of lower life-cycle costs.

ABOUT THE AUTHORS

MR. WILLIAM J. ROWAN is an Electronics Engineer with the Naval Training Systems Center's Research and Development Department. He is currently assigned to the Systems and Computer Technology Division working on the application of new computer architectures to training systems' computational requirements. He holds a B.S. degree in Engineering Science from Hofstra University. He has previously held engineering positions with the Boeing Company, Singer's Link Division and the Harris Corporation.

MR. DAVID M. KOTICK is an Electronics Design Engineer with the Systems and Computer Technology Division of the Naval Training Systems Center. His principal responsibilities include the integration, evaluation and design of microprocessor-based board-level products. He has received both a B.S.E. and M.S.E. in Electrical Engineering from the University of Central Florida. He has worked in the area of real-time multi-microprocessing at NTSC since 1983. Mr. Kotick is a member of the IEEE and IEEE Computer Society.

MR. MARK W. LAYTON is currently participating in the Undergraduate Air Force Pilot Training Program. Prior to his selection for this program, he worked as a Computer Scientist with the Systems and Computer Technology Division of the Naval Training Systems Center. He has applied operating systems and graphics expertise to the area of microcomputer simulation design. He received his B.S. in Computer Science from the University of Central Florida in 1983.

MS. CARIDAD E. RUIZ is a Computer Scientist with the Systems and Computer Technology Division of the Naval Training Systems Center. She has applied graphics expertise to the area of microcomputer simulation design. She received her B.S. in Computer Science from the University of Central Florida in 1984 and is currently pursuing her M.S. in Computer Engineering specializing in Artificial Intelligence.

MR. CHARLES N. POPE is an Electronics Engineer currently working in the field of visual technology research at the Naval Training Systems Center. He received the B.S.E. degree with a major in Electrical Engineering from the University of Central Florida. He has worked extensively with the design of microcomputer-based simulators, particularly in the area of software partitioning tool development.