ROTORCRAFT FLIGHT SIMULATION IN THE PARALLEL MICROPROCESSOR ENVIRONMENT

Jeff McFarland*
Don Monroe*
Cindy Maher*
Engineering and Training Simulation Department
McDonnell Douglas Helicopter Company
5000 East McDowell Road
Mesa, Arizona 85205

ABSTRACT

Our multiple microprocessor approach to application processing provides an open architecture that permits computing resources to be expanded and customized to meet simulator/training requirements Migrating computing resources from the minicomputer to the microprocessor environment results in an enormous savings in simulation costs. These savings are realized in the initial hardware investment and again in the reduction of facility requirements, including power, space and cooling. This was particularly evident in our parallel microprocessor implementation of the rotorcraft flight model at McDonnell Douglas Helicopter Company, Mesa, AZ, for engineering simulation.

Our rotorcraft simulation was migrated from the GOULD 32/9780 to the Motorola 68020 microprocessor environment. The required frame rate exceeded the capabilities of a single microprocessor. Therefore, the rotorcraft simulation, previously running in serial, was divided into three small models running in parallel on three microprocessors: the main rotor/tail rotor and equations of motion, the stabilization systems, and the engines. The VME bus was selected as the backplane for the system because of high speed data transfer capabilities of this bus and the large selection of devices available for VME. The Motorola 68020 CPU was selected as the target microprocessor. Development was done on the VAX/VMS system using FORTRAN 77 and C cross-compilers for the MC68020 target. Although it was initially uncertain whether the flight model could remain stable running on parallel processors, the parallel portage resulted in insignificant variance from the serial model. Timing results were easily within the required frame rate. This paper discusses the advantages of the microprocessor approach over the minicomputer approach for our rotorcraft flight simulation, the migration of the serial flight model to parallel processing, and how this approach can further enhance rotorcraft flight simulation.

ADVANTAGES OF MICROPROCESSOR OVER MINICOMPUTER

The migration from the minicomputer to the microprocessor at MDHC was driven by the capital cost per MIP advantage of the microprocessor and the modularity of the microprocessor. The development of denser and faster microprocessors and the introduction of a wider range of support tools has complemented the migration to microprocessor at MDHC and provides a bright outlook for our future systems.

Cost Per MIP

The cost advantage of the microprocessor over the minicomputer was defined by looking at three areas: hardware capital investment, overhead associated with the equipment investment, and software tools capital investment.

The hardware investment advantage was determined by studies done at MDHC comparing the processing speed, memory capacity, and hardware investment between the mainframe and the microprocessor. These studies concluded that "when comparing mainframe and microprocessor costs along a common metric of 'millions of instructions per second' (MIPS), the ratio of costs is in the neighborhood of 16:1 in favor of the microprocessor."**

The microprocessor holds an even larger advantage in reduced overhead. A fully loaded 21 slot VME chassis housing 16 CPU boards and 5 peripheral boards (approximately the equivalent computing power as the mainframe) has a power requirement of only 960 watts and a space requirement of a mere 2.25 cubic feet of simulation real estate. This is a power advantage of 100:1 and a real estate advantage of 80:1 in favor of the microprocessing system.

The cost advantage of software development tools for the microprocessor varied based on the development machine. Microprocessor software development tools were available for many different software environments – VAX/VMS, SUN/UNIX, PC/MS-DOS, etc. For a large development effort, the cost of software tools for either system was competitive. For small projects, the cost was much less for the microprocessor tools because they could be housed in a smaller development environment. Over the past two years, a wide selection of support tools have been introduced to the market.

*Member Technical Staff
**MDHC ETSD 5-year plan

MDHC was able to customize our microprocessor development environment to our requirements by selecting tools from a variety of sources and housing these tools on our centralized development systems. Although the cost of the software tools for the microprocessor and the cost of software tools for the mainframe are competitive, the flexibility of the microprocessor software tools provided the advantage in this area.

Modularity

Modularity plays a key role in the design and development of training and simulation devices at MDHC. The modular cockpit has been a much praised approach. A large part of the modular cockpit has been achievable through the use of microprocessors. The microprocessor provided modularity of the computing hardware by allowing hardware to be dedicated to an application without having to archive a large and scarce resource. MDHC has seen the advantage of this approach when reconfiguring the simulation arena for simulations of unique rotorcraft.

Future Development

The microprocessor industry is producing faster and denser microprocessors. In the two years MDHC ETSD has been involved with the Motorola microprocessors, the speed of the 68020/68881 VME CPU board has doubled. This board is expected to double again with the introduction of the MC68030/68882. Also, there are many products being released this year that boost 2-10 MIP speeds. Cross-compilers and cross-debuggers are now available for most languages including Ada. The only negative in the microprocessor market has been the lag of quality software tools by 6-12 months from the introduction of the microprocessor.

MDHC'S EXPERIMENTAL PROJECT

A major obstacle in the migration from the mainframe to the microprocessor for the ETSD simulation was the portage of the MDHC rotorcraft flight model. The flight model was the largest of several tasks residing on a GOULD 32/9780 minicomputer. During runtime, the IPU (one of the two GOULD 32/9780 processors) was dedicated solely to the flight model. The rotorcraft handling qualities required by the simulation dictated that the rotorcraft flight model complete a frame 60 times a second. Our design goal was to move the MDHC rotorcraft flight model from the GOULD to the microprocessing environment with no degradation in performance. The success of this design goal would reduce the cost of processing hardware for the rotorcraft flight model by a factor of 16. Also, it would provide proof of concept for the migration from minicomputer to microprocessor in simulation.

Parallel Implementation

A MDHC study determined that the current rotorcraft flight model would run at a 38 Hz frame rate on a single Motorola 68020/68881 microprocessor. Therefore, the 60 Hz frame rate requirement dictated the use of parallel processing. Parallel processing allows a modular system to be expanded to meet the computational requirements of the project. The modularity of microprocessors has made parallel processing accessible.

Moving to a parallel processing environment involved porting the distribution of the flight model into multiple CPUs, hardware integration of the CPUs into the system, software integration with the cross-compiler and target download, and developing a real-time operating system to handle system calls, synchronize tasks and monitor communications between processors.

The Flight Model

A significant challenge in porting the MDHC flight model to the microprocessor environment was the distribution of the model across multiple processors. The MDHC flight model for the AH-64A is a serial rotor-map model developed in the early 1980s for use by the Research and Engineering division at MDHC. This model is integral to several unique systems throughout the company for test and evaluation of the AH-64A flight performance, subsystem interaction evaluation, simulation, and training. The portage to a parallel environment was in support of simulation and training.

The rotorcraft flight model consists of components modeling the airframe, rotor systems, the engines and the control systems. The model's main components and their associated dynamic inputs and outputs are shown in Table 1. These modules are listed in their calling order within the serial model.

The dynamic I/O table illustrates that the flight model is heavily tied into a serial structure. Many of the modules listed in Table 1 required the outputs of preceding modules as inputs. Initially, we were concerned with modules receiving a combination of past and current data in the same frame. We could maintain a serial structure within each processor which alleviated some concerns about the stability of the model. Intuitively, we felt that if the model could run fast enough, any remaining stability problems would dissolve. We were able to run at 60 Hz with no apparent stability problems.

Parallel Distribution

Timing studies indicated that the flight model would need to be distributed across three processors to meet the 60 Hz frame rate requirement. As a side note, due to a system I/O requirement of 2 msecs per frame, the flight model actually needed to run at over 70 Hz. The module distribution displayed in Table 2 describes the final distribution of the parallel flight model. In this configuration, the flight model, executed at 80 Hz. When executed with a 60 Hz system sync, 4.3 msecs of dead time are available for system I/O.

Processor 1. Top consideration was given to the equations of motion module and integration module since their outputs drive the simulator. The Main Rotor module, Tail Rotor module, and Vertical Stabilizer module have the largest impact on the Equations of Motion module. We wanted to maintain the serial structure of the serial model with these modules and also insure they ran within the same frame. Therefore, these modules were grouped on processor 1.

TABLE 1 Flight Model Components Dynamic I/O – Listed in Order of Serial Calling Sequence

Dynamic Inputs	Module	Dynamic Outputs
Rates, Velocities, Main Rotor Swashplate Angles, Engine Torque	Main Rotor	Forces and Moments, Main Rotor Wake Skew Angle and Downwash, Torque Required
Rates, Velocities, Main Rotor Wake Skew Angle and Downwash	Fuselage	Forces and Moments
Rates, Velocities, Tail Rotor Swashplate Angle	Vertical Stabilizer	Forces and Moments
Rates, Velocities, Tail Rotor Swashplate Angle, Engine Torque	Tail Rotor	Forces and Moments, Torque Required
Rates, Velocities, Main Rotor Wake Skew Angle and Downwash	Horizontal Stabilator	Forces and Moments
Rates, Velocities, Ground	Landing Gear	Forces and Moments
Forces and Moments	Equations of Motion	Angular and Translational Accelerations
Main Rotor Torque Req, Tail Rotor Torque Req, Collective Swashplate Angle	Engines	Engine Torque
Pilot's Inputs, Rates, Velocities, Aircraft Attitude	Control Laws	Main Rotor Swashplate Angles, Tail Rotor Swashplate Angle
Accelerations	Integration	Rates, Velocities, Euler Angles

Processor 3. Processor 3 was dedicated to the engines. The engines were allotted their own processor since their execution time was approximately equal to the execution time of processor 1. We were not concerned with the stability of the engines since their inputs are fairly static under normal flying conditions.

Processor 2. The grouping in Processor 2 was derived by default. The parallel flight model executive and all remaining modules were placed on Processor 2. The frame time of this grouping was close to the frame time of the other two groupings.

System Integration: Hardware

The Motorola 20 MHz 68020/68881 VMEbus CPU board was chosen as the MDHC rotorcraft flight model target environment. Of the high speed CPUs, the MC68020/68881 was selected because of the availability of VAX base cross-compilers, the department familiarity with the microprocessor, and the department familiarity with the VMEbus.

Table 2 Parallel Flight Model Module Distribution

Processor 1	Processor 2	Processor 3
Main Rotor	Executive Control	Engines
Tail Rotor	Control Laws	
Vertical Stabilizer	Fuselage	
Equations of Motion	Horizontal Stabilator	
Integration	Landing Gear	

The hardware integration of the multiple CPUs in the VMEbus system involved jumper/switch configuration on the CPU board for the VMEbus memory map address of the board and the VMEbus access priority level. Our boards were 1M boards mapped into memory for 32 bit addressing and 32 bit data words. Our boards were configured as VMEbus slaves daisy chained together at the same bus priority level. Peripherals were added as additional boards in the system. The MDHC parallel processing rotorcraft flight model chassis is configured in Fig. 1.

Systems Integration: Software

Porting the application program to a microprocessing environment was essentially a port to the microprocessor compiler, a scheme to download the target microprocessor memory, and implementation of the runtime operating system. Porting the application program to a multiprocessing environment involved modifications for multitasking and multiple processor communication.

The Compiler

The original MDHC rotorcraft flight model was developed under VAX/VMS in VAX FORTRAN. The Motorola 68020/68881 cross-compiler used by MDHC for the parallel processing flight model was the Microtec FORTRAN 68K under VAX/VMS. ETSD was among the first customers for the FORTRAN 68K and the only customer using it for a floating point intensive application. Microtec had a difficult time delivering an acceptable product – particularly in support of the 68881 Math coprocessor. After about six months, and many revisions, most problems had been solved.

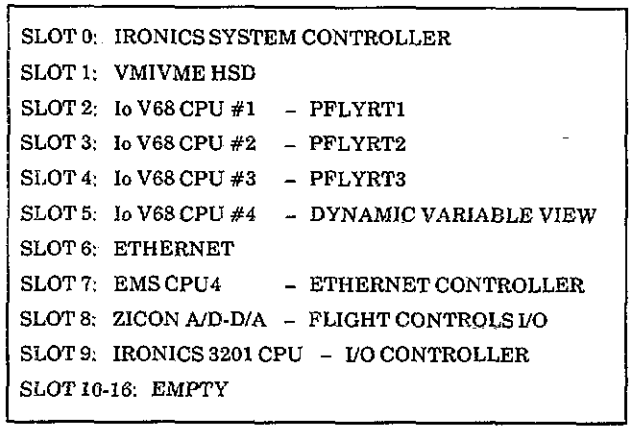

Fig. 1 MDHC Parallel Flight Model VMEbus Configuration

The compiler dictates the allowable extensions to a language. A highly portable program would contain no extensions to the language. The "portability" of a application is a significant issue. Two major portability areas were addressed for the MDHC flight model portage from VAX FORTRAN to the Microtec FORTRAN 68020 cross-compiler. The first was the inability of Microtec FORTRAN to support local static variables. The flight model takes advantage of the fact that on the VAX and on the SEL, local variables remain static. We created an equivalence to local static variables by using a global data block with access to only the local module. A VAX/VMS resident utility was developed to search for the local variables required to remain static and to group them into one common block for each module. This utility greatly reduced the difficulty and chance for errors in this port. The second issue was with uninitialized data. The 68020 runtime operating system does not clear all of the memory that has been allocated for the application before the program is downloaded - primarily because the operating system is downloaded with the application. This causes a discrepancy with the way VAX/VMS executes a program. To alleviate this discrepancy we call a FORTRAN subroutine that initializes all data at the start of the application.

As a final word about the compiler portage, the decision to port the original flight model to Ada was rescinded after a MDHC study found the ETSD standard Ada compiler to be four (4) times slower than the ETSD standard FORTRAN compiler for the MC68020/68881. As Ada becomes faster through maturity, the flight model portage to Ada will become feasible.

Target Download

At MDHC, the target load of the application was done using the Ethernet bus between the DEC VAX CLUSTER and the VME chassis. The VME Ethernet control was customized to our requirements based on the TCP command protocol. The VME Ethernet controller accepted Motorola S Records. Basically, each S record contained an address and a data segment. During a valid communication, the address was modified to a VME 32-bit address within the target board's memory and the data segment was written to that address. The last S record contained the start address of the application. With a little bit of handshaking between the Ethernet controller and the CPU monitor, the application would autoboot upon download completion.

Runtime Operating System

The operating system is a single tasking environment whose duties include: terminal I/O, system clock, task exit, and memory management. The operating system was linked to each application as a runtime environment. The CPU monitor (or MDHC ETSD Ethernet Downloader) controlled the start of the application. Upon the start of the application, the operating system initialized hardware and defined the memory segments. During the run the runtime environment handled any system calls as direct calls from the application. Finally, upon normal completion, the operating system halted the CPU.

The multiple processor issues of communications and tasking were handled by the application. Mailbox interrupts, polled I/O, and shared memory were the three methods of processor communication. Tasking was hardcoded as discussed in the parallel flight model module distribution.

Communications

Mailbox interrupts are a highly useful feature on most 68020 CPU boards on the VMEbus. A hardware interrupt is generated when a preassigned address (mailbox) is written to from the VMEbus. This is a local interrupt and costs only 1 bus cycle (write). Upon interrupt, the CPU can read the mail - learning sender and message. In our parallel processing flight model, the synchronization of all processors is maintained using mailbox interrupts. Some of the functions controlled by mailbox interrupt were GO, HALT, INITIALIZE, TRIM, RUN, RESET, and ABORT.

Shared memory can be accessed through the VMEbus or through the CPU local bus. A data structure of all data common to the processors was located in VMEbus memory for access by all processors. To meet speed requirements it was important to keep this data structure small to minimize bus access cycles. Polled I/O of shared memory was another method used to communicate between processors. To minimize possible bus clashes, we never polled on a VMEbus memory location. Our rule was to write to VME memory and read from local memory.

Data was communicated to the world outside of the VME backplane through a HSD bus. The HSD bus is a parallel high speed data bus. This bus performs DMA transfers between processing environments.

Debugging

The final phase in the software port was debugging in the microprocessor environment. The CPU monitor

provided memory view, register view, start instruction, breakpoints, an assembler, and a disassembler. During the rotorcraft flight model portage to the microprocessor, MDHC developed a FORTRAN common block view routine that allows the display and modification of any common variable during runtime. This routine is housed on a separate CPU in the VME chassis and has proven to be an invaluable debug tool. At the time of procurement for the MDHC microprocessor FORTRAN cross-compiler, a source level debugger was not available. MDHC has since implemented a source level debugger for Ada applications but has no plans for a FORTRAN debugger.

Results

The parallel processing rotorcraft flight model maintained the 60 Hz frame rate with 4.3 msecs per frame of dead time for system overhead. MDHC is currently involved in a qualitative evaluation of the parallel flight model. Preliminary testing in the batch mode compared values between the serial model and the parallel model and the results looked very good. Also, already several hours of flight time has been logged on the parallel flight model. Pilot opinion has been optimistic. The study should be concluded by late 1988.

SIGNIFICANT ACCOMPLISHMENTS AND FINDINGS

The MDHC rotorcraft flight model portage to the multiprocessing microprocessor environment provided significant accomplishments and findings for training and simulation. There is parallel processing capability in rotorcraft flight modeling, even in an originally non-parallel design, that was used to our advantage to reduce the "cost per MIP" of a real-time rotorcraft flight model. This portage has proved the validity of the migration from minicomputer to microprocessor. Flight model development of a blade-element model in the multiprocessing microprocessor environment is feasible using similar hardware and methodology discussed in this paper. Hardware performance is advancing at a rapid rate. For modeling work, the major hardware performance improvement to come will be in the performance of floating point intensive applications. The parallel flight model also has made available the evaluation and implementation of the new high speed multiprocessing hardware, such as the transputer and other array processors. This hardware can be evaluated and integrated upon introduction.

ABOUT THE AUTHORS

Mr. McFarland holds a Bachelor of Science in Mechanical Engineering and is currently pursuing a Master of Science degree. Mr. McFarland has 5 years experience in realtime system software – including system communications and electro-mechanical controls. Since joining McDonnell Douglas Helicopter Company in 1985, his primary responsibilities have been with the M68020/68881 processor and the VMEbus environment.

Ms. Maher is currently finishing an MS degree in Mathematics and Computer Science. Since joining MDHC in the spring of 1985, she has been a integral member of the Flight Dynamics group in the Training and Simulation department. Her main contributions have been in the areas of control laws and flight model development and implementation.

Mr. Monroe holds a BS in Aeronautical Engineering and has 18 years experience in the aerospace industry, mainly in aerodynamic performance and preliminary design. He joined the MDHC Simulation department one year ago, and has worked largely on the implementation of the parallel flight model.