Fermi National Accelerator Laboratory

FERMILAB-Conf-87/22
2380.000

Software for the ACP Multiprocessor System*

J. Biel, H. Areti, R. Atac, A. Cook, M. Fischler, I. Gaines
C. Kaliher, R. Hance, D. Husby, T. Nash, and T. Zmuda

Advanced Computer Program
Fermi National Accelerator Laboratory
P.O. Box 500, Batavia, Illinois 60510

February 2, 1987

*(Presented at the Computing in High Energy Physics Conference, Asilomar, California, February 2-6, 1987)

Operated by Universities Research Association Inc. under contract with the United States Department of Energy

FERMILAB-Conf-87/22

SOFTWARE FOR THE ACP MULTIPROCESSOR SYSTEM

J. Biel, H. Areti, R. Atac, A. Cook, M. Fischler, I. Gaines
C. Kaliher, R. Hance, D. Husby, T. Nash, T. Zmuda

Advanced Computer Program
Fermi National Accelerator Laboratory
P.O. Box 500 Batavia, IL 60510, USA

Abstract

Software has been developed for use with the Fermilab ACP multiprocessor system (described in an accompanying paper). The software was designed to make a system of a hundred independent node processors as easy to use as a single, powerful CPU. Subroutines have been developed by which a user's host program can send data to and get results from the program running in each of his ACP node processors. Utility programs make it easy to compile and link host and node programs, to debug a node program on an ACP development system, and to submit a debugged program to an ACP production system.

1. Introduction

The Fermilab Advanced Computer Program (ACP) has developed a multimicroprocessor system for use in experimental (and in the future, theoretical) high energy physics. A companion paper describes the hardware that has been developed. In this paper, the ACP multimicroprocessor software is described.

In high energy physics experiments, each experimental event is statistically independent of all others, and can be analyzed independently. This 'trivial parallelism' allows the analysis to be divided among an arbitrary number of computer processors. Each processor is loaded with a copy of an analysis program and is then sent a series of events to process. This approach to multiprocessing eliminates any need for communication between processors and is appropriate for experimental reconstruction of data as well as Monte Carlo simulations. We will first describe this mode of operation which is presently the primary application of the system. In section 6 we will discuss new software capabilities that will support internode communication required by problems like lattice gauge theory. The ACP software provides tools that ease the transition from the conventional uniprocessor environment to the multiprocessor world. This allows a user to enjoy the full power of a parallel processing system without getting bogged down in architectural or hardware details.

2. General Approach

The hardware for the ACP system consists of a host computer (a MicroVAX) connected over a high speed data link to a large number of (over 100) node computers (32-bit microprocessors). In order to use the ACP system for experimental analysis, the user must divide the analysis program into two parts: a host program and a node program. The host program runs on the host processor. It reads events from magnetic tape, sends the data for each event to one of the node processors,

gets the results from any node processor that has completed its task, and writes the results to another magnetic tape. The node program analyzes the event it has been sent and then signals the host computer that it is done. A single host processor can keep n node processors busy as long as the time required by the host program to read, send, get, and write an event is less than $1/n$ of the time taken by a node processor to analyze one event.

Dividing an existing single processor program into node and host programs is generally an easy task because the part of the program that performs the event analysis is usually contained in a well defined set of Fortran subroutines. The node microprocessors are required to have a full Fortran 77 compiler so these subroutines can generally, with only minor modifications, be used to construct the node program. The host program runs on a VAX computer under the VMS operating system and it may, therefore, use any of the VAX extensions to Fortran 77 that the user desires. The user's node program is called as a subroutine on the node processor under a tiny ACP operating system called the "tight loop monitor". This consists of only a few hundred lines of Fortran and is, therefore, very easy to adapt to any future ACP node designs.

3. Components of ACP User Support Software

There are five basic parts to ACP user support software: a set of user callable subroutines for the host program to transfer events to and get results from the node processors, a compilation and linking utility (MULTICOMP), a simulator of an ACP multiprocessor which runs on a single VAX, a mechanism for detecting and processing node program errors, and a batch submission process for production runs.

Host Program Subroutines

The set of Fortran callable subroutines by which the user's host program communicates with the node processors has been designed to be very easy and flexible. All host program subroutines use node COMMON blocks as the destination of data being sent to a node and as the source of data being retrieved.

A typical user host program is structured as follows.

1. **Initialize the ACP multimicroprocessor system (subroutine ACP_INIT)**
   Subroutine ACP_INIT initializes the ACP multimicroprocessor system. It reads a user parameter file (UPF, described further below) in which the user defines the desired node configuration, downloads the nodes with copies of the user's node program, and starts each node running. When each node is started, it enters a program loop waiting to receive notification that it has been sent an event from the host program. It then calls the user's event analysis subroutine.

2. **Transmit calibration constants to each node (subroutine ACP_BROADCAST)**
   Next the typical host program calls subroutine ACP_BROADCAST to transmit a set of calibration constants to each of the nodes. ACP_BROADCAST automatically converts data (such as floating point values) from host format to node format.

3. **Begin event processing loop**
   The event event processing loop, which comprises the greatest part of the required computing, consists of steps 4 through 8.

4. **Read the next event to be processed (subroutine ACP_READTAPE)**
   Subroutine ACP_READTAPE reads the next event from the input magnetic tape. If the host program has not yet processed the last event read, this step is skipped. If there are no more events on the input tape, the processing loop is exited by proceeding to step 9.

5. **Send the event to a node for processing (subroutine ACP_SENDEVENT)**
   Subroutine ACP_SENDEVENT tries to send the event to a node for processing. The attempt is not always successful because most of the time all of the node processors are already processing events. The success or failure of the send operation is indicated by the value of a logical variable returned by ACP_SENDEVENT. The user does not need to specify a particular node as the destination because all node processors are running the same program. The user's node program is automatically started running immediately after the node has been sent the event.

6. **Get the results from any node that has finished (subroutine ACP_GETEVENT)**
   Subroutine ACP_GETEVENT checks to see if any node has finished its processing. If so, the results are placed into a user Fortran array. The success or failure of the get operation is indicated by the value of a logical variable returned by ACP_GETEVENT. If the program has accumulated enough results to fill an output tape record, then step 7 is executed, otherwise step 7 is skipped.

7. **Write results to the output magnetic tape (subroutine ACP_WRITETAPE)**

8. **Loop back to step 1**

9. **Histogram accumulation (subroutine ACP_SUMNODE)**
   After the event processing loop has been exited, a typical host program calls subroutine ACP_SUMNODE to combine the histograms that have been accumulated in each of the node programs into a single summary set of histograms.

This typical program outline illustrates the general features available from the ACP host subroutines. There are additional routines that provide more general features. For example, by using subroutine ACP_SENDBLOCK instead of ACP_SENDEVENT, the user can both specify the particular node to which data is to be sent and send data to more than one node common block before the node program processes the event. Subroutine ACP_GETBLOCK provides similar generality in getting data from a node. The user can have more than one event analysis subroutine in the node program. By using subroutine ACP_RUNNODE, the particular node program that is to be used on an event can be specified at the time each event is sent to a node. Currently in test are subroutines that support the CERN ZEBRA data management package on the ACP multimicroprocessor system. This is to provide ACP support for the latest version of CERN software packages such as HBOOK for histograms and GEANT for Monte Carlos.

ACP Compilation and Linking Utility Program (MULTICOMP)

The MULTICOMP utility program was developed to make it very easy for a user to prepare a program to run on an ACP multimicroprocessor system. By typing a single MULTICOMP command to the VAX operating system, a user's source code for the host and node programs is turned into host and node executable programs. MULTICOMP shields the user from a myriad of details that must be taken care of in creating the host and node programs. It only compiles or links those routines or libraries that have changed from previous compile or linking passes. After using MULTICOMP, a single RUN command will start execution of the program in interactive mode, or the batch submission system described below may be used.

For example, MULTICOMP eliminates the need to learn how to compile and link programs for various types of node microprocessors. There is, in fact, a considerable difference in the way programs are compiled and linked for the two node microprocessors currently supported by ACP. Compilation and linking for the Motorola 68020 microprocessor takes place on one of the ACP node processors via a UNIX-like operating system (called LUNI) that was also developed by the ACP. The Absoft version 2.2 compiler is presently used. Compilation and linking for the AT&T 32100 microprocessor takes place on an AT&T 3B2 computer connected to a VAX 11/780 via a high speed serial link.

The input to MULTICOMP consists of a user created file (the UPF) that lists the number and type of nodes that the user wishes to use, the host program source files, and the node program source files as well as several other user-selectable options such as which ACP Multiprocessor system (or the simulator) is to be used or whether a node debugger is to be used. The UPF file is a standard text file that can be created by any VAX text editor. At a minimum, the UPF file must specify one source file for the host program and one source file for each type of node microprocessor that is to be used. Optionally, the user may give a list of source files, object files, and library files to be used for the host program and for each type of node processor program. The information in the UPF file is then used by MULTICOMP to perform the necessary compilation and linking steps. The output of MULTICOMP is an executable file for the host program and an executable program for each type of node processor.

If the user needs to make any changes in the host or node program (or in any of the other options that may be selected in the UPF file), it is only necessary to make the changes and again invoke MULTICOMP. MULTICOMP is very sophisticated in interpreting the changes that the user has made and performing the minimum number of operations needed to update the host and node programs. For example, if only a user object module library has been updated, MULTICOMP will relink the program that uses the library and do nothing else.

ACP Multiprocessor Simulator

Before testing a program with actual ACP node processors, it is strongly recomended that a user test the program with the ACP multiprocessor simulator. The node simulator supports all the ACP features, simulating physical node processors as VAX subprocesses. The primary reason for the node simulator is to allow a user to test that the original single processor program has been properly split into host and node parts without having to cope with any problems due to an

unfamiliar node compiler, linker, and processor. The ACP multiprocessor simulator can be run on any VAX.

A user requests the node simulator with a single command line in the UPF file. When MULTICOMP is invoked, the node program executable program that is produced will have been compiled and linked for running on a VAX. The program can then be run by issuing the VMS "RUN" command with the name of the host program executable file. The VAX subprocesses used to simulate the nodes are automatically created when subroutine ACP_INIT is called in the host program.

Node Program Errors

A large multiprocessor computer system presents new challenges as well as opportunities in the detection, interpretation, and handling of program errors. The ACP system provides automatic exception reporting, an event history file, and event verification.

If a problem occurs in a node program, the ACP system automatically prints an informative message with the node number, the node program counter address, the type of error that has occurred, and (if possible) the Fortran line number. In the UPF file, the ACP user can select three possible courses of action to take after an error has occurred: ignore the error and proceed as if nothing has happened, kill (i.e. send no further events to) the node but continue processing events with the remaining nodes, or abort the entire job.

An event history file is automatically produced whenever a user program is run. This file contains, for each node, a list of the sequence number of each event that was sent to the node. If a not understood error occurs in a node, a separate test run can be made with just the event that triggered the error message. If this run does not reproduce the error, then the problem may be caused by a particular series of events and not just the last event. The information in the event history file can then be used to recreate the same series of events that caused the error in the node program. In actual practice, this feature has not yet proven to be necessary.

Event verification is a powerful bug detection tool that is only available on a multiprocessor system. The ACP user can, optionally, request in the UPF file that some of the node processors be declared "verify nodes." Whenever the user sends an event to a node, a check is made to see if one of the verify nodes is available. If so, the event is also sent to the verify node (refered to as the node's "verify partner") and both nodes are started running. When both nodes have finished, a check is made to see if they contain the same results in a node common block the user has selected in the UPF file. An error is declared if the results are not identical. Because the event history of the node and its verify partner are different, this technique allows the detection of subtle, incipient errors, seen in many physics codes, that are dependent on the order in which events are processed by a node and which may contaminate results without generating hard error messages.

Batch Submission Process

Once the user has compiled and linked the program with MULTICOMP, the program can be run in one of two ways. A user can run the program directly (by simply issuing the VMS "RUN" command with the name of the host program executable file) or the program can be submitted as

a batch job. The ACP has developed a batch submission mechanism based on the standard batch queues available in VMS. The MULTICOMP process and batch queue submission take place on what is called the "development host" (at Fermilab, the VAX Cluster) which is not itself necessarily connected to any ACP node processors. When a user job is submitted, all the necessary files to run the job are concatenated (into a VMS "save set") and a job is placed into a special batch queue. When each job reaches the top of the batch queue, the save set is copied to the ACP production VAX (over DECnet) and a job is submitted into the production VAX batch queue. The production VAX is limited to one user at a time, so when a job enters its batch queue, the job immediately begins execution. After the user production job has run, the results files are copied back to the VAX from which the user originally submitted the job. The batch system includes tape management features which warn the computer operators which magnetic tapes are scheduled for use in upcoming jobs, and issue VMS tape mount requests for each ACP job as it enters execution.

4. Bringing up a program on an ACP Multiprocessor system

Based on experience with a number of users, the ACP now recommends the following sequence as the most effective in bringing up a program on the ACP system:

1. Be sure the program runs satisfactorily as a single processor program on a VAX. For Fermilab experimenters with old analysis programs on CDC computers, this step can be the most difficult because of the difference between the 60-bit word size of the Cyber and the 32-bit word size of the VAX. A program that runs on a VAX is much easier to convert to a program that will run on nodes with 32-bit words.

2. Split the program into a host program and a node program and run the program on the ACP multiprocessor simulator with one simulated node. This checks that the single processor program has been successfully split while postponing any problems that may be caused when more than one node is used.

3. Run the program on the simulator with several simulated nodes.

4. Run the program with one real node processor. This checks for problems that are caused by differences between the node and VAX compiler/linker/hardware. By far, the largest number of problems that occur in this step are due to uninitialized variables. The VAX initializes all variables to zero. The user may usually specify that this also be done in the node program. Problems encountered when testing with a single node are often solved by using an interactive node debugger program which allows a user to debug his program at the level of Fortran source lines and Fortran variable names.

5. Run the program with more than one real node processor. This is the final check that everything is working properly.

5. Multiprocessor Utilization

The most important measure of any multiprocessor system (other than the cost effectiveness of its available processing power) is what percentage of the nodes processors are processing data.

In actual practice, the performance of the ACP system has been excellent. The Fermilab E691 reconstruction program kept approximately 95% of the nodes busy in the 53 node system then available. A typical reconstruction program will have no difficulty achieving similar results on a system with many more than 100 nodes.

The two fundamental limits on multiprocesor utilization in the ACP system are events-per-second and bytes-per-second. The events-per-second limitation is a measure of the minimum time required to send an event to a node and get the result in the limit where the event is zero bytes in length and the node program finishes instantaneously. The bytes-per-second limitation is a measure of the maximum rate at which the data for a very long event can be transfered between the host computer and a node.

Both of these fundamental limits have been measured for an ACP multiprocessor whose host computer was a single MicroVAX and whose host computer was a pair of MicroVAXes. When two MicroVAXes are used as the host computer, the second MicroVAX acts as an intelligent peripheral device to unburden the first MicroVAX of the details of host-node communication. For a single host MicroVAX, the limits were 49 events per second and 504K bytes per second. For a two MicroVAX host, the limits were 100 events per second and 520K bytes per second. In the two MicroVAX system there is much more CPU time available for user host programs without degrading system performance. The events-per-second limit is primarily due to the software overhead of the ACP multiprocessor system software. The bytes-per-second limit is due to the maximum transfer rate supported by the MicroVAX QBBC data link to the nodes (which depends on a DEC DRV11 parallel block transfer device). With the use of the VBBC presently in design, this number will increase so that the tape drive limits of ~800KB/sec will dominate. Higher speed mass storage devices may also be used with the VBBC up to the 20MB/sec bandwith of the Branch Bus.

As long as each node takes many seconds to analyze an event, the events-per-second limit will not prevent the host program from keeping all of 100 or more nodes busy. For users with node programs that analyze events too quickly, the effective events-per-second limit can be easily increased by processing groups of events at a time in each node, sending and retrieving them in blocks.

6. Support of Theory Computations

At present, the ACP multimicroprocessor system at Fermilab is being used primarily for the analysis of experimental events. In a collaboration with the theory group at Fermilab, the ACP is developing hardware and software resources for lattice gauge calculations and similar theory problems. Nodes with array-processor-like support and very high floating point performance are in development. Extensions to the ACP software have been specified to allow easy communication between node processors for problems like lattice gauge.

As part of the initialization of the host and node programs, the user calls ACP subroutines that define the communication links that each node is to have with the other nodes. Examples are (in all cases, periodic boundary conditions are assumed):

1. Ring connectivity. Each node has two links, one to the node on its left, the other to the node on its right.

2. Grid connectivity. Each node has four links, a link to the left, a link to the right, a link up, and a link down.

3. Hypercube connectivity. The node communication links are configured as lines drawn between the verticies of an n dimensional hypercube.

From then on, the node program can transfer data between nodes by making calls to ACP node subroutines. The location of the other node is always specified relative to the node making the transfer request, with parameters indicating, for example, "two nodes to the left", or "one node up".

Two important features of this communications approach are that the program for all the nodes can be identical, and that the program is completely independant of the physical mechanism for internode data transfer. Although the program results will not depend on the physical node communication, the program efficiency will decrease if a communications-intensive application is run on a system which is a poor match to the defined connectivity.

7. Conclusions

The ACP multiprocessor system software provides easy access to the powerful multiprocessor hardware. Additions that support a variety of internode communication are being developed. The software has already proved successful and easy to use for experimental event reconstruction and is now being applied to lattice gauge calculations.