CONF-811040--189

DE83 001888

Data System Design Considerations for a Pseudo-Steady-State Device

W. R. Wing

ORNL,* Oak Ridge, Tennessee 37830

The Advanced Toroidal Facility is being designed to run in a steady state. This places stringent requirements on a data system, since it must provide steady-state support that is equivalent to the support users are accustomed to from pulsed experiments; i.e., enough capacity to reduce diagnostic data for live presentation. Parameters such as density, position, and temperature must be presented live (i.e., within 0.1 s). Quantities such as plasma shape or internal structure should be available with a minimum of delay. The traditional solution to providing such capabilities is to use distributed processing to off-load data acquisition from the analysis computers. However, this suffers in a real-time environment because of the necessity of moving large quantities of data from acquisition to analysis. We expect to solve the problem by using a pipelined design that will acquire data directly into shared memory, where any one of four CPU's (VAX 11/780's) can proceed with analysis.

Last spring, when I agreed to give this talk, I assumed that at this time we would have finished the design of the data system for ISX-C and would be well into its implementation. However, a funny thing happened on the way to construction. Our tokamak has turned into a stellarator. ISX-C has now become ATF-1. Everybody knows how to build data systems for tokamaks; now we have to learn what needs to be done for stellarators. The ATF project will have a data system with some interesting features due to the nature of ATF itself. ATF will feature steady-state operation. It will also attempt to be simultaneously a research device and a technology development device. Because of this and some other reasons we will get to in a minute, the data system will be responsible for both the physics data acquisition and analysis and will also be responsible for machine monitoring and control.

Let's take a look in more detail at what these points imply. First, steady-state operation implies that startup and shutdown become exceptional conditions; the normal mode of operation is running. Thus, there is no mode change between data acquisition and analysis. In particular, there is no way to use time between shots or pulses for data analysis. Because the system will need to acquire some burst-mode data in response to both anticipated and unanticipated events, there is a blurring of the distinction between scan-driven and interrupt-driven operations. Thus, design emphasis is on bottlenecks in data flow, not on data size or program size. Second, a combined physics-monitor/control system implies that physics codes (elephants) must avoid control codes (rabbits). Input and output devices should be compatible across the system, and there should be no distinction between physics and engineering diagnostics. Similarly, files and file access should be compatible. And finally, the system must be fail-soft. In other words, no single failure should be capable of halting the system. Failures should, at worst, degrade operation.

*Operated by Union Carbide Corporation under contract W-7405-eng-26 with the U.S. Department of Energy, OFE.

At first glance, this looks like a classic situation in which to use distributed computing; that is, to use a large number of CPU's running in parallel and communicating over a network. For physics one might assign one CPU per diagnostic and use CPU's sized to the calculation load. For monitoring and control one would use one CPU per major function, i.e., state control, safety, discharge control, and logging.

We don't think this is the solution. The trouble is that there is no logical distinction between types of data. This is the "other reason" alluded to earlier that drives us to a unified system in the first place. Not only is there no clear distinction between physics and operations data, the vast majority of data are needed by both systems. There is a complete spectrum of both data and applications in which the reduced data are needed. Figure 1 illustrates this point by sketching some of the data presently produced (and used) by the ISX-B data system. Even in the case of nominally pure physics data, any given input signal is likely to be needed by several different analysis tasks (running in other CPU's). This forces the network to be totally general-purpose, supporting bidirectional data traffic, remote disk file access, and connecting each CPU to all others in the network. Networks like this are NOT efficient. Let's look at two examples: The first example comes directly from DEC corporate headquarters at Marlborough, where a pair of VAX 11/780's supporting DECNET phase III, are linked together via DMR-11's. The DMR-11 hardware speed is 1 Mbit/s. Using this hardware DECNET/VAX realizes 25 kbytes/s, AND it uses one fourth of the entire CPU doing it. The other example is presently running in FED at ORNL. A PDP-11/45 is linked to a PDP-10 via a DL-10 memory window and supports codes optimized for file transport only. The hardware speed is over 1.5 Mbytes/s, while the realized speed is only 12 kbytes/s.

So what would happen in our hypothetical distributed computer data system? Assume an 11/34 is measuring electron density along five chords (running a multichord interferometer) at a 1-kHz sample rate. It must provide continuously updated readings to:

State Control for Auxiliary Heating
State Control for the Machine
Diagnostic CPU for X ray
Diagnostic CPU for Radiometer
Diagnostic CPU for Electron Temperature

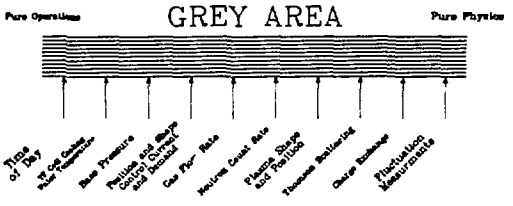

Fig. 1. The spectrum of uses of acquired and reduced data in a unified data system.

Diagnostic CPU for Ion Temperature
Diagnostic CPU for Spectroscopy
•
•
•

The point is that after the first three, the network traffic alone would use all available CPU resources.

So just what are we proposing? We propose to eliminate the network and instead analyze the data flow needed, then organize the system to support that flow; i.e., to design a pipelined system. The conceptual data flow can be divided into three independent paths.

Input → Data Acquisition → Data Pool
Data Pool → Data Analysis → Data Pool
Data Pool → Data Disposition → Output

Note that all these processes are simply special cases of a generalized "Move Data" from point A to point B. Thus, they can all be specialized versions of a single database-driven scheduler.

Let's examine the three stages separately. We will assume that all input will be via CAMAC. Then for simplicity, we will try to reduce the computational requirements to millions of floating point operations per second. Finally, we will direct all output to Logical Instruments. To see just what is required, let's examine the needed scan rates:

500 points scanned at 1 Hz
200 points scanned at 10 Hz
100 points scanned at 1 kHz.

We also need roughly 6 Mbytes of on-demand transient recorder-style data acquisition arranged as;

200 points sampled at 1 kHz for 1 s., and
200 points sampled at 150 kHz for 0.1 s.

This on-demand data acquisition is similar to the sort of data handling now used on tokamaks and other pulsed experiments. In a steady-state situation, the demand corresponds to a significant event, either a deliberate experimental modification of the plasma or an unexpected instability or dump. We will assume that such experiments occur no more often than once every five minutes. Thus, the on-demand data load averages out to 20 kbytes/s and the ≈300 kbytes/s load of steady-state data handling will dominate the system.

We have already demonstrated a sustained 1 Mbyte/s throughput in a CAMAC system using the ISX-B Data System P. E. 8-32 computer, a Jorway Model 432 Branch Driver, and PCAM (Perkin-Elmer CAMAC Access Method) software. PCAM supports 3 mechanisms for CAMAC access:

Experiment Mode; Precompiled, 24 µs per call
  No Branches
  No Loopes
  No LAM's
  Fixed Data Arrays
Linked-List Mode; Pre-evaluated, 240 µs/call
  Branches
  No Loops
  LAM's
  Fixed Data Arrays
Immediate Mode; Evaluate at run time 2.4ms/call
  Branches
  Loops
  LAM's
  Variable Data Arrays

A CAMAC Branch will easily support burst transfers at a 1.5 Mbyte/s rate. Thus, if it can be kept busy 20% of the time, it will meet the 300 kbyte/s throughput requirement. Now if the scheduler is no better than the PCAM Linked-List mode, it is only necessary to block the CAMAC input instruments so that transfers can take place in block reads of 48 or more data points, and the requirements will be met. Note that this will always be true for transient recorders, but would require clustering instruments like scanning ADC's. However, there is still a better way. In fact, there are two. They are a K.S. Model 2053 Branch Driver or a K.S. Model 2053 Branch Driver plus a smart front end.

A 2053 supports five modes of operation; one of them represents programmed I/O, and the rest are all DMA modes.

DMA Mode 4 is known as the "Execute List" mode and has the ability to automatically execute lists of read and write operations with no software overhead. Using this mode still requires the CPU to handle branching, looping, LAM's, and errors. For a general case, this means that CAMAC data would be handled at roughly the same rate as the PCAM Linked-List mode. This would be adequate to support the needed throughput, but it would demand the clustering of CAMAC modules as noted and would tie up a lot of CPU resources.

However, it can be extended using a smart front end and the VAX "secret port." The UBA has two Unibus ports, the standard port and the NPR priority arbitration card slot. If we remove the NPR-PA card, insert a PDP-11 Unibus, some memory, and a CPU, and move the PA card to the CPU, we have a smart front end. Figure 2 illustrates just how this all fits together. Note the PDP-11 memory, which would contain the actual code used to drive the front end and which could be PROM. Note that this is not an easy system to program. In fact, because of several problems, we will use it only if we have to. To take full advantage of the front end, it must be able to execute data acquisition code in the VAX memory and write data to VAX memory without involving the VAX CPU. This implies being able to read the VAX system page tables and modify the UBA mapping registers accordingly. In fact, in a big system it might be necessary to modify the mapping registers just to see all the system page tables. This can only be done by loading the first mapping register at bootstrap time so that it wraps back around and points the first field of addresses at the rest of the mapping registers. Thus, the smart front end can be used to

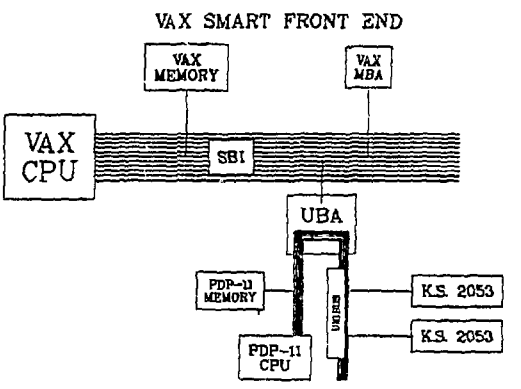

Fig. 2. The block diagram of a VAX smart front end using the VAX "secret port."

handle the looping, branching, LAM's, and errors while the 2053's Execute List mode handles straight linear code. However, there is poor error recovery (i.e., none). The trouble is that the microprocessor in the UBA doesn't have an internal data path that bypasses the SBI. Because of a quirk in the timing of read and write operations, the front end CPU can write the mapping registers, but it cannot read them. A UBA used with such a CAMAC B.D. can't be used with any other peripherals. Finally, it's not clear that this double port will ever appear on the UBA for a future '790. In fact, it doesn't appear on the UBA for the '750 right now.

Now, what about analysis? Let's assume a simple model for general output value calculation:

$$OV_j = (IV_1 \times IV_2)K_1 + IV_3K_2 + K_3.$$

So each output value can depend on its own input and a linear combination of two other inputs. This yields five floating point calculations per output value. At 100k data points/s, this means a total throughput of 0.5 Mflops.
In theory, a VAX CPU can reach 1.0 Mflops In practice, in a general case it actually achieves a bit less than 0.25. Thus, to sustain this throughput, we expect to need two CPU's just for analysis.

Finally, how do we expect to handle the required output? We need three different forms of output: output to archive, output to control, and output to display. Archive output is the easiest. At the simplest level, we can simply stream data out on a pair of 6250 BPI tape drives. At the next higher level (if enough throughput is left to handle it), we can write data to pre-defined data files. Finally, the highest level would involve defining files in real time as we go, and letting the OS do the work. Control output is the next most complicated. In fact, it gets us into the idea of "Logical Instruments". We expect to need to define Logical Instruments for both data acquisition and feedback control. As far as the host CPU is concerned, a logical instrument would be simply another CAMAC location (or block of locations) at which it could read or write data values. In fact, of course, we expect to use this as a means for supporting autonomous input and output devices. They might be simply microprocessor-driven CAMAC modules or fully independent computer-based systems. Logical Instruments let us isolate pure preanalysis, set up stand-alone control CPU's, and isolate safety functions to PC's. In this context, it is worth noting the existence of the new Jorway A2 CAMAC Crate Controller. It is a standard A2, but in addition, it will not only accept an auxiliary controller, it is capable of acting as an auxiliary controller. Thus, it is possible to have a CAMAC Crate with two completely independent host computers. The Jorway A2 has a mailbox memory and a fairly complete set of LAM functions for interlocking the two host processors.

As an example of the problems of unified input and output, let's consider displays. I've noted that the system must support a unified file structure and unified input/output devices. This implies a set of unified displays for both physics and operations. The problem is that an operations display is almost certainly a raster-scan device. It needs to support color, area fill, and possibly touch-sensitive interaction, all of which imply raster-scan technology. On the other hand, a physics display to have high resolution (4096 x 4096), and very high speed selective refresh (it should handle vector end points at DMA speed). It turns out that there are systems that, in effect, do both. That is, the file structures, driver software, and interfaces are independent of the final output display. The actual display device can be chosen to be either raster or stroke-vector to suit the need at hand. We are looking at systems made by Sanders Associates, Megatek, and a British firm, Sension Ltd. So far, at least, there is no obvious winner.

So, where is all this leading? The system I have, in effect, described is illustrated in Fig. 3. The symmetry for input and output is obvious. What should also be obvious is that the system is set up to handle processing as a pipeline. That is, data are acquired directly into shared memory (the pool). Here, analysis tasks running in the separate CPU's can access and reduce data and return reduced results to the pool and can access the results of prior or parallel computations. It may be worth noting that since a VAX multiport memory can only be shared among four CPU's, the system topology has to change slightly for expansion beyond the four CPU's shown. One possible route to expansion is shown in Fig. 4. The pipelined nature of the system is much more obvious here, but at the same time there is now less flexibility as to where any particular analysis task could run.

Obviously, the problems of providing support for steady-state devices are just beginning to be appreciated, and at Oak Ridge we are just beginning to understand what will be required. Nonetheless, I believe it is a fascinating problem, and I hope you have enjoyed thinking about it with me.

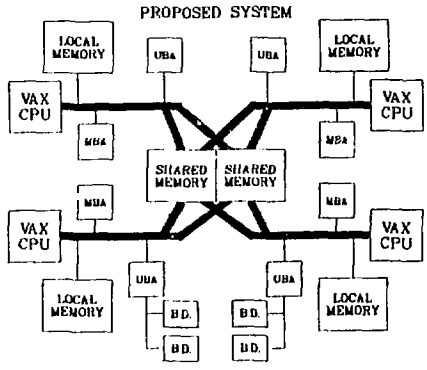

Fig. 3. A block diagram of the pipelined system. Note the symmetry for input and output and the location of all data in a common (shared memory) pool.

SYSTEM EXPANSION

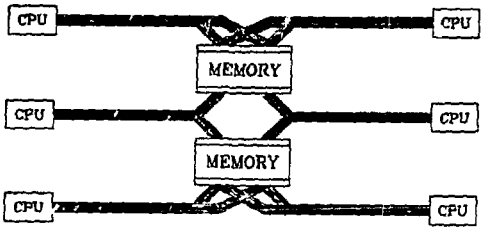

Fig. 4. A block diagram of an expanded version of the pipelined system, with six instead of four CPU's.