SLAC – PUB – 3736
July 1985
(T/E/I/A)

RESUME ON VECTOR AND PARALLEL PROCESSING IN HEP*

Paul F. Kunz

Stanford Linear Accelerator Center
Stanford University, Stanford Ca, 94305

A Bit of History

Processing data in HEP has come a long ways as this short historical summary shows. In the 1960's, HEP detectors were counters, spark chambers, and bubble chambers. The data acquisition systems, by today's standards were slow and produced a small amount of data. The fastest computers of that era were sequential machines. HEP was using the supercomputers of the day and they were quite adequate for our needs.

In the 1970's, detectors moved to entirely electronic readout with first proportional chambers, then drift chambers. Data acquisition systems became modest in size as well as the amount of data produced. The fastest computers split into two paths, bigger sequential machines and vector processors, i.e. the supercomputers. HEP found that *productivity tools* were more important than raw CPU speed and our codes didn't vectorize very well, thus followed the route of the sequential machines and losing our expertise on the supercomputers.

In the 1980's, detectors became massive data generators with data acquisition systems that can easily saturate tape writing speeds. The large computers are similar, but faster, then the ones in the 1970's but there are new options for processing; namely parallel processing. Experimental HEP now has more needs for computing then we can afford to acquire by traditional means. To these needs, we also have to add the new needs of the accelerator physicist and those of the theorist. Thus, at conferences like this one, we discussion how the computing problems are going to be solved.

HEP Computer Needs

HEP computing needs have been dominated by experimentalist's event processing, Monte Carlo generation, and analysis requirements which have become very large with the advent of large 4π detectors. However, HEP needs is no longer completely dictated by the experimentalists, the needs of theory and accelerator design must also be considered. We must also look, therefore, at a wider range of solutions then those that have been traditional considered *best* for HEP. Namely, we must look at vector and parallel processors as well as the standard sequential machines.

In fact, HEP's event processing needs are rather unique in the field of large scale processing and frequently misunderstood by industry, computer scientists, and others. The short description of the event processing needs that follows is intended to help those outside of HEP to understand them.

For the HEP experimentalist, about 1/3 of the CPU cycles are needed for event processing. An event is a snapshot of a nuclear interaction as seen by the detector elements. Processing an event is the reconstruction of the physics parameters from this *crude* snapshot. I emphasis the word *crude* because the detector elements don't have the granularity one would expect from a photograph, nor is their performance perfect. The snapshot can be 50K to 200K bytes of data and the processing may take 10 to 100 seconds of CPU time on a mainframe.

* Work supported by the Department of Energy, contract DE-AC03-76SF00515.
Invited talk presented at the Conference on Computing in High Energy Physics
Amsterdam, The Netherlands, June 25-28, 1985

The problem for the experimentalist is not finding a computer with sufficient power to process one event. Rather the problem is that 1-2 events per second are collected while the detector is running, thus millions of events are collected per year. Event processing must keep up with event collection or a backlog develops, there would not be feed back to the running detector and physics results will not be presented in a timely fashion. To analyze the events processed, comparison is made with a set of events which have been generated by Monte Carlo simulation techniques. This set of events will be at least equal in number to the real events and will also take a considerable amount of CPU cycles to generate and process. The processing of real and Monte Carlo events typically use up about 2/3 of the CPU cycles available at an HEP laboratory.

The event processing is not the whole problem. The development of the programs, which can easily reach 500K lines of FORTRAN, is a problem of considerable magnitude. This development is done by 10-50 physicists working over a number of years without real events to judge their progress on, then when real data is available from an imperfect detector they must work rapidly to correct, tune, and otherwise make the programs work properly on a short time scale so that physics results can be extracted. To this, there is of course, many other development jobs such as calibration and alignment work, and analysis of the event samples. This development and analysis work dictates a computer with good *productivity tools* which may not be the one with the fastest CPU per unit cost.

The needs of the HEP theorist are of the more traditional type known well to industry and computer scientist as *scientific and engineering* computing. That is, they consist of a model of a physical system on a lattice of points where one needs to do a large number of floating point calculations on each point and many iterations must be done for the system to stabilize into some state. The needs of the HEP accelerator designers is one of tracking a large number of particles through a grid of dipole, quadruple, and sextupole magnets to understand the stability of a particular design. Both of these kinds of problems are similar in nature to problems in other fields of science. They both have a relatively small amount of code and they vectorize well, unlike the event processing problem of the experimentalist.

VECTOR PROCESSING

The term *vector processor* and the allied term *array processor* are somewhat misnomers for a style of computer architecture which is based on a simple fact: there is no way with a given technology that floating point arithmetic is going to be as fast as a binary add. Also there is no practical way that random memory access time is going to be as fast as a binary add. Thus in a computer a single floating point operation is going to take multiple CPU cycles. For example, a floating point add may be divided into a number cycles show in Fig. 1. The operations performed in each cycle are as follows:

1. Fetch operands from memory and/or register files.
2. Prenormalize the mantissa with the smallest exponent.
3. Add the mantissi.
4. Postnormalize the resulting mantissa and correct the exponent if necessary.
5. Store the results in memory or register file.

With an appropriate computer architecture, this operation over a number of operand pairs can be made faster by overlapping the steps or pipelining them. Thus as shown in Fig. 2 one can do 3 floating point adds in only 7 cycles instead of the 15 it would take if done sequentially.

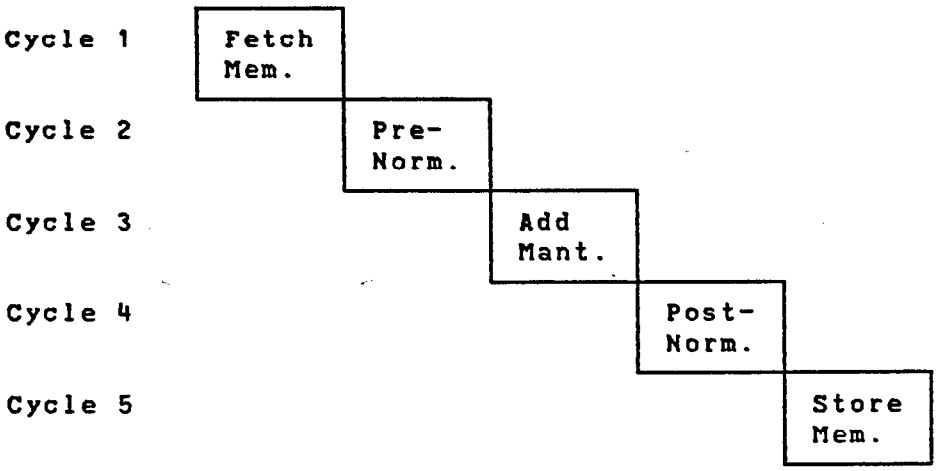

Fig. 1. Example of multiple cycles of floating point add.

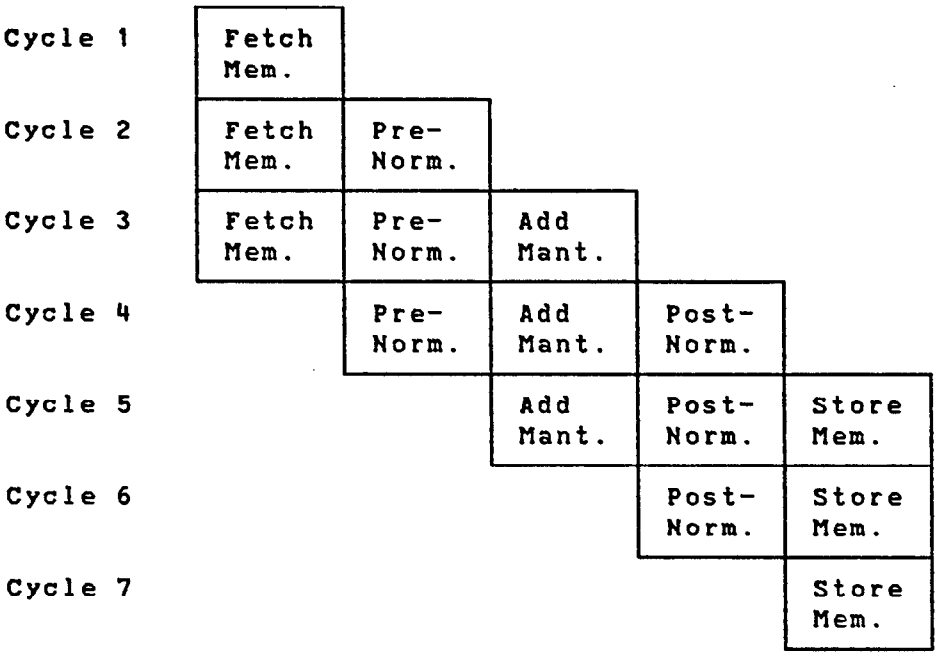

Fig. 2. Example of pipelined cycles of floating point add.

This pipelining only works well if the data operands are in an orderly pattern, which the FORTRAN programmer knows as a vector or an array, thus the terms vector or array processor is used for one that can perform in this manner.

Since the introduction of the Cray-1 supercomputer in the mid-1970's, there has been renewed interest in this type of architecture. At this conference, Tor Bloch reviewed the two newest entries in the supercomputer arena in a paper entitled "Supercomputers". The first was the NEC SX-2 which has a 6 nanosecond cycle time. With each memory cycle it fetches 8 words but with an elapsed time of 216 nsec. It also has very large register files but with a 40 nsec access time. The other is the Cray-2 with a 4.1 nsec clock. It fetches 1 word with each memory cycle which has a 242-260 nsec access time. Both machines have significantly larger memories than previous supercomputers: 256 Mbytes for the SX-2 and 256 MWords for the Cray-2.

To the HEP experimentalist, these machines look impressive but even more inappropriate than previous supercomputers. That is, it is difficult for the experimentalist to understand how to use these machines effectively. To the theorist and accelerator physicist, however, these new

machines look good as their codes do vectorize. The trend towards very large memory sizes in supercomputers is very much needed and the CPU speed must also be very fast to be able handle the very large lattices one can now put into memory.

The supercomputer to a theorist is what an accelerator is to an experimentalist. That is, once a model of the physical world is running on the supercomputer the theorist can perform his *experiments* to study the validity of the model or study the states the model can produce. The HEP community is used to the idea of spending money to build accelerators for the experimentalist. However, they are not used to the idea of spending money on supercomputers for theorist, nor are the theorist used to getting together to make proposals to obtain the money they need to buy equipment. Good theory results can be obtained from supercomputers, but the HEP community as a whole needs to decide how much money they are worth and make the appropriate investments.

In spite of the experimentalist's pessimism on supercomputers, one paper at this conference showed some interesting results with HEP code. Given by Kenichi Miura of Fujitsu Limited and entitled "Vectorization of Monte Carlo Codes on the FACOM VP-200 for High Energy Physics Applications", this paper showed a speed up factor of 3.5 on a well know event generation Monte Carlo code from CERN. One could ask the following questions about this work.

- The author worked alone in improving the code for his vector processor, understanding the workings of the code only from comment cards. Could the author of the code have done an even better job in speeding up the code?
- After the conversion of the code to a vector processor, is it still maintainable and/or can be further developed in the sense we have come accustomed to or has it become very difficult to understand or modified?
- Are the techniques applied to obtain the speed up general, or was the author *lucky* with this one code? Can the techniques be applied to real event processing with all the exceptions the code has for a real detector (as opposed to an idealized detector)?

Nevertheless, this paper show some validity to what many are beginning to suspect, that vector processors should be looked at more carefully by the HEP community. It would be a shame if a vendor of one of these machines had to show us the techniques to make them effective in HEP, if they are there.

PARALLEL PROCESSING

The other method of getting more performance is to try to exploit the inherent parallelism of a program and have these parts run in parallel on separate processors. These processors can be either tightly coupled or loosely coupled, in many cases it doesn't matter. They don't even have to be complete computers as long as they are cost-effective processors.

For the HEP event processing, it is clear that events can be processed in parallel. That is, the method of having one event processed by one processor *works* as was clearly demonstrated at this conference by papers from Richard Mount ("Alternatives in High Volume HEP Computing") and Martin Pohl ("Loosely and Tightly Coupled Parallel Processors for High Energy Physics"). There is nothing new here, it has been exploited by users of the 168/E since 1979. These authors have also shown that the method is not sensitive to the method of coupling, it's working equally well with tightly coupled processors such as the Elxsi computers or loosely coupled processors such as the FPS-164. Trying to exploit parallelism within one event, however, has so far been less effective as was shown by these same authors along with a paper by Jalby and Maillard ("Use of SIMD-SPMD machine for Simulation in Particle Physics"). The

result is that the overall execution time can easily be slowed down by the non-parallel part of the program, even with tightly coupled processors.

Joe Ballam in his paper ("Future Plans for HEP Computing in the U.S.A.") raises the question of whether parallel processing technique will continue to be valid in the SSC era where a single detector will require 1000-2000 VAX 11/780 equivalent processing power. The answer seems to be affirmative and can be understood from the following simple arguments:

- A data acquisition computer with a certain I/O bandwidth recorded the data at the detector.
- What ever the power of the parallel processors (as long as they can run the complete program), one will add enough of them to obtain the required total CPU power.
- As long as rate of event processing is not greatly different from the original data acquisition rate, then the host computer with I/O capacity at least equal to the data acquisition computer will be sufficient to run a processor *farm*.

There were a number of papers on processors with an orientation towards parallel event processing presented at this conference. It should be emphasized these processors which are designed and built within the HEP community are not real computers. By stripping the hardware down to the bare processing essentials, they are very cost effective means to obtain the large scale computing that is needed.

In considering using one of these processor systems, there are a number of issues that should be examined by potential users. Some of which were addressed by users and implementers:

- The user friendliness of the interface between the processors and the host.
- The difficulty of porting the user code to the processors, or writing special code for it.
- The importance of uniformity between the processors, the host computer, and the computer where the code was originally developed.
- The speed versus cost of processor.
- The procedures for checking that the processing system is working correctly.
- The real costs and time scale of manufacturing and maintenance of the system.

An ideal system would address all these issues well, but as things go each system excells on some issues but not on all of them.

Since on-line triggering and filtering systems have become more sophisticated, the distinction between them and off-line processing systems has become blurred. There was a number of papers presented with an orientation towards on-line systems that could be considered as candidates for off-line systems as well and vice-versa. The same set of issues are discussed but with different emphasis for the on-line environment.

Another area that has been successfully using parallel processors is the theory calculations. Here one has learned that one can make efficient use of parallel processors provided that:

- The amount of time one processor works on its data is much longer than the time it takes to communicate with other processors.
- The amount of time spent on the parallel part of the algorithm is much larger than the sequential part.

Both conditions can be easily met by HEP QCD lattice guage codes. Papers on both loosely coupled and tightly coupled systems were presented. The tightly coupled systems have an advantage when the communications times can not be made small, but for many of the HEP theory codes this advantage is small.

It is interesting to note the emphasis put into parallel processing by the computer industry. Besides the papers presented by industry at this conference, there are many other companies, both new startups and our traditional suppliers, producing products with parallel processors. One can ask the question to as whether industry is (finally) learning that the users can and will modify existing programs or develop new ones that run efficiently on parallel processors and that not all scientists have *dusty decks* that they must run without change.

At this conference, there was fairly large number of papers presented on processors and/or systems with an orientation towards doing theory calculations. As with the off-line and on-line processors, the same issues are being addressed, but the emphasis is again different. The program length being very much smaller than for event processing, one can be much more lenient on the issue of porting the code to the processor. The main emphasizes are on speed of the processor and the system that interconnects the processors to themselves.

CONCLUSION

HEP physicists are quite active in the field of vector and parallel processors, and this activity extends over off-line event and Monte Carlo processing, on-line trigger/filtering, theoretical calculations, and accelerator design. Although only HEP's theoretical work seems appropriate for vector processors, there are some signs they could be used for experimental work as well and their use is worthy of further study. Use of parallel processors in HEP is well established in all areas with many systems in use and still more being planned.

The next question to the experimentalist, is how are they going to analysis all the events that get processed? That is, it is less clear how vector or parallel processing is going to help with the histogramming, cutting, fitting, etc. These processes one would like to be interactive procedures and yet can consume a lot of I/O resources, not just CPU cycles. At future conferences, we should begin to see the answers to this question.