

Fermi National Accelerator Laboratory

FERMILAB-Conf-84/18
2300.000

COMPUTING TOOLS FOR ACCELERATOR DESIGN CALCULATIONS*

Mark Fischler and Thomas Nash

January 1984

*Submitted to the Workshop on Accelerator Physics Issues for a Superconducting Super Collider, December 12-17, 1983, University of Michigan, Ann Arbor, Michigan.



COMPUTING TOOLS FOR ACCELERATOR DESIGN CALCULATIONS

Mark Fischler and Thomas Nash
Fermi National Accelerator Laboratory
Batavia, Illinois 60510

Summary

This note is intended as a brief, summary guide for accelerator designers to the new generation of commercial and special processors that allow great increases in computing cost effectiveness. New thinking is required to take best advantage of these computing opportunities, in particular, when moving from analytical approaches to tracking simulations. In this paper, we outline the relevant considerations.

Given the finite level of funding available to solve a particular computing intensive problem, the computing cost effectiveness, defined for example as equivalent floating point operations per second per dollar (EFLOPS/dollar), can become critically important. This figure of merit depends on choices of technology, memory, special purpose hardware, and interconnection architecture. These choices cannot be separated from algorithm decisions. Therefore, it is important that those embarking on highly computing intensive calculations understand the options and trade-offs that they have and the implications of their choices. An example is the very non-linear trade-off between calculational bits required and cost effectiveness: the luxury of not having to worry about computing precision may come at the price of eliminating potentially very powerful table look up techniques.

Table I summarizes potentially relevant types of computing engines. Cost effectiveness numbers are normalized to a VAX 11-780 with moderate peripherals appropriate to accelerator calculations (\$200K). Such numbers (from any source) are normally meaningful only within a factor of two and these are not intended to be better than that. Not the least reason for this is that machines and compilers have different efficiency for different programs even when one focuses, as we are, on a specific class of problems. The cost numbers explicitly do not include development costs (which may be substantial if commercial or laboratory developed systems are not available). The level of development effort required and centers where this is going on is indicated in the fourth column of Table I.

Ease of use is obviously of crucial importance. This means FORTRAN language availability and system user friendliness that meets the popular standards set by the VAX. With the exception of the last case (stand-alone special processors), all the examples we give can meet this requirement. In the first three cases (CYBER, VAX, CRAY), the user interface is delivered with various levels of friendliness as part of the turn-key commercial system. For microprocessor based systems, turn-key systems that take full advantage of the potential cost effectiveness are not commercially available and do not appear to be in planning for marketplace availability.

Bringing the full cost effectiveness of commercially available microprocessors to the high energy physics community in a user-friendly, software and hardware modular package, is one important goal of Fermilab's Advanced Computer Program (ACP). Later, we will briefly outline the ACP's planned system of modules and some of the parameters and design considerations relevant to them. These modules will allow the user to optimize the architectural configuration for a specific problem to get the best possible cost effectiveness.

In Table I, one can see that there are two general approaches to obtaining substantial (orders of magnitude) improvements over commercially available computers: a) use of large arrays of microprocessors, which among Fortran programmable CPUs are the most cost effective; b) use of special purpose devices aimed at the computing intensive kernels of algorithms. The extent to which one can take advantage of the considerable potential of these two approaches depends on the degree of parallelism (and, for the specialized devices, structuring) that specific algorithms will allow and programmers implement. These techniques provide very large amounts of computing power per dollar because of their very low cost per unit and not because individual processor computing power is large. This means that problems with one kind or another of simple parallelism can readily benefit from multiprocessing. A one thousand ray beam calculation, for example, can be trivially divided into up to one thousand parallel, uncoupled, processors. This is clearly not so for a single ray transport.

The issue of intrinsic parallelism is important enough to warrant a small diversion explaining to non-aficionados the different possible types of computer parallelism and the time scale on which they are likely to be relevant. A conventional computer, where single words of data are operated on sequentially (sometimes in pipelines), is referred to as an SISD, single instruction stream, single data stream machine. Computers with instructions that operate simultaneously on vectors are referred to as vector computers or SIMD, single instruction stream, multiple data stream machines. The Cray I and CYBER 205 are two major examples of SIMD machines. Vectorizable problems (particularly those with vector sizes that are matched to the machine width) are handled efficiently by such machines. The transport problem is obviously vectorizable at the five or six dimension level and, therefore, can benefit to some extent from vector processors. The general problem of making efficient use of SIMD computers for the parts of programs in which vectors are not as obviously manifest has been the subject of an intense effort in computer science and is far from being solved.

Fully parallel computers, referred to as MIMD (multiple input stream, multiple data stream), allow each of many processors independent access to any location in memory (Figure 1). These are less constraining and are, therefore, potentially more efficient for the general problem. For this reason, much academic (and some commercial research) is going into the development of general purpose MIMD computers. Although potentially capable of bringing the high cost effectiveness of VLSI microprocessors to the broadest possible range of problems, such machines and the requisite software are not likely to deliver on that potential until at least 1990. (The Denelcor HEP is the only commercially available general purpose MIMD machine. It is of great value for developing MIMD software, but does not have cost effectiveness good enough to include on our Table).

There is a significantly large subclass of problems that are so naturally and intrinsically parallel that they do not require each CPU to have easy access to all of the available memory. Therefore, these problems can take advantage of multiprocessor systems far simpler than the general purpose MIMD concept, specifically arrays of nodes

Table I

TYPE	EXAMPLES	AVAILABILITY ¹	DEVELOPMENT EFFORT REQUIRED ²	POWER/CPU WATTS = 1	TYP. (MAX) # CPUs IN MULTI-PROCESSOR GROUP ³	INST & PERIPHERAL COSTS/GROUP ⁴ (X \$1,000)	TOTAL COST OF TYP. GROUP ⁵ (X \$1,000)	COST EFFECTIVENESS ⁶ WATTS = 1
Conv. mainframe	Cyber 175	Now	Commercial	4	1	1,500	1,500	.5
Super mini	VAX 11/780	Now	Commercial	1	1 (4)	200	200	1
Vector super	Cray 1s	Now	Commercial	16	1	5,000	5,000	.65
Array processor	FPS 164	Now	Commercial	3	1	200	400	1.5
	ST 100	1984	Commercial	25	1	200	500	10
Emulators	3081/E	1985	Major (SLAC/CERN)	5	10	150	350	25
Multi-processor	ACP	1985	Major (Fermilab ACP)	0.5	128 (255)	150	530	25
		1986		1	128 (255)	120	440	60
		1987		1.7	255	60	620	140
Multi-processor called hardware subroutines	Multiplier based	1986	Small - medium	20	100 (255?)	120	470	800
	Memory look up	1986	Small - medium	60 ⁷	50 (255?)	120	530	12,000 ⁷
Stand alone special purpose	Multiplier based	1985	Medium	10	100	250	450	800
		1986		20	100	120	320	2,000
		1986	Medium	350 ⁷	50	120	470	15,000 ⁷

each containing a CPU and its own "local" memory. Such arrays are said to be "loosely coupled" because they have limited (or no) internode communication capability. These problems involve numerical solutions of differential equations on a grid or they are event oriented with each event processed independently of the others before a final statistical evaluation. Pleasantly, the three major computing problems of high energy physics fall into these two categories. The theoretical physics lattice gauge problem is solved (in principle) on a loosely coupled grid of processors with one of several possible nearest neighbor interconnection schemes (Figure 2). The large numbers of data events recorded by many experiments may be reconstructed in a largely parallel, very loosely coupled system (Figure 3). Individual rays in an accelerator transport calculation may also be considered as events in this sense and processed independently.

When accelerator transport problems are limited to fewer rays than the number of nodes available in a multiprocessor system, they can not efficiently utilize that system. Some calculations in progress use a relatively small number of rays (sixteen, for example) to define their phase space. It is not clear whether the limited granularity of the problem is anything more than stylistic. When increased power becomes a reward for finer granularity, calculations may well migrate quickly to approaches involving larger numbers of rays.

Table I cites one approach to competitive cost effectiveness over the next two years that requires somewhat less granularity than commercial microprocessors. The 3081/E emulator is being developed at CERN and SLAC as an answer to experiment reconstruction and high level trigger demands.¹ Emulators are so named because, using very fast logic and bit slice techniques, they are able to "emulate" (in this case) the IBM 3081 instruction set. This means that code written, debugged, and compiled on an IBM mainframe can, with relatively little difficulty, be loaded onto the far cheaper emulator. Since (as Table I shows) the 3081/E unit cost and computing power is higher than microprocessors of the same cost effectiveness, the granularity requirement is correspondingly lower. For the first year of their availability (1985), this possible emulator advantage comes at no sacrifice in cost effectiveness. On that time scale, predictions for the cost effectiveness of the 3081/E and multiprocessor systems are indistinguishable.

Unfortunately, emulators appear to have saturated the technology for the foreseeable future and their cost effectiveness will remain nearly constant with time. Through 1987, improvements that are known to be coming in commercially produced microprocessors will, as Table I shows, at least double the cost effectiveness each year. Beyond that, the crystal ball is cloudier, but one can expect continued improvements in single multiprocessor cost effectiveness through about 1990. At that point, physics limitations on chip speed will force improvements to move in the direction of more processors on a single chip. Thus, in the long run, accelerator algorithm designers clearly must move in the direction of more granularity if they want to take advantage of commercial microprocessor developments.

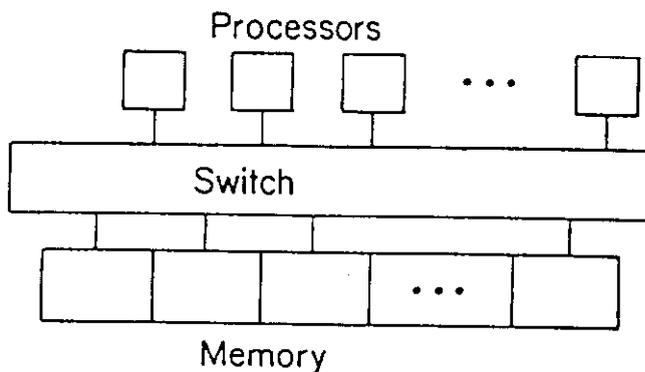


Figure 1: Conceptual diagram of general purpose, closely coupled MIMD computer.

As we noted earlier, the second general approach to the HEP computing problem involves special purpose devices. These may use a variety of techniques including dedicated custom VLSI circuits, commercial VLSI multipliers (which have the highest commercially available potential FLOPS/dollar), and table look up systems that contain rapidly accessible pre-computed functions of arbitrary complexity (and which have no theoretical upperbound to their potential cost effectiveness). These special purpose systems may be completely stand-alone--in which case programmability considerations will preclude their use in environments where flexibility and FORTRAN are prerequisites. Building such devices in the form of modular "hardware

subroutine" coprocessors linked to microprocessors is an ACP concept that will make such systems available in a FORTRAN environment. In general, the lead time for developing specific special purpose systems, either stand-alone or as coprocessors, need not be even as long as a year--once, of course, the algorithm requirements are understood in detail and resources (namely a design group) are available.

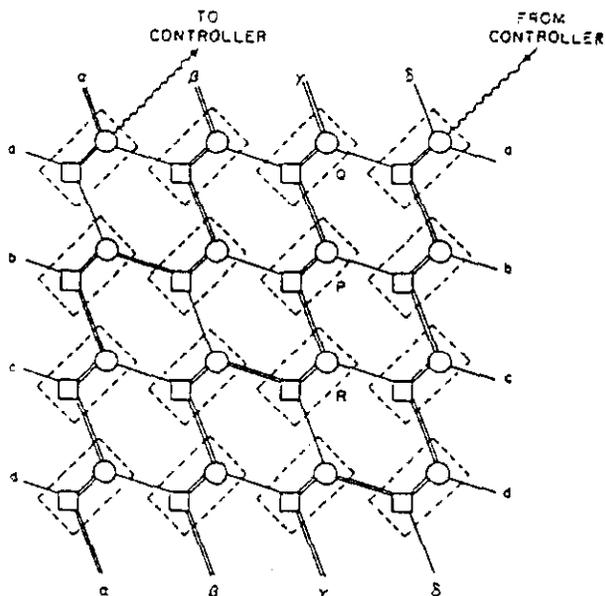


Figure 2: The lattice gauge multiprocessor interconnection scheme of Christ and Terrani (Ref. 7). Circles are processors (and coprocessors); squares are memory.

By the term coprocessor, we mean a hardware device (a copy of which is attached to each node) which enhances the performance of the node by doing a commonly done, time consuming procedure very quickly. An elementary example of such a device is the floating point accelerator found in almost all computers; a third of the processor time is typically spent on floating point adds and multiplies. By appending a faster floating point ALU, one obtains a corresponding gain in speed. Since the process only occupies thirty-three percent of the time, one can at best get a net speedup of a factor of one and one-half. In general, if a coprocessor can take over a part of the calculation that otherwise would take a fraction R of the time, it can improve the cost effectiveness by no more than a factor of $1/(1-R)$. Thus, to exploit this approach, it is important to identify kernels with R greater than 0.9. Here we will present examples of four types of plausible coprocessors.

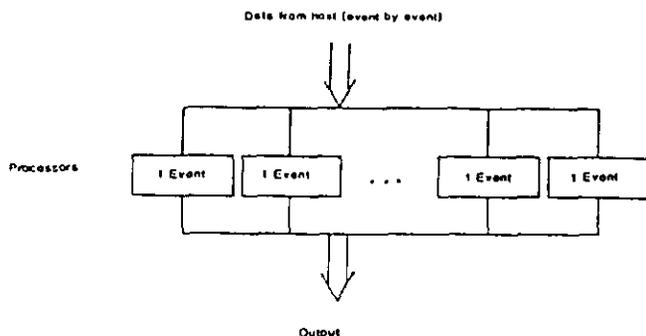


Figure 3: A simple, single rank multiprocessor configuration for event oriented problems. Single events in an accelerator problem are individual rays.

Along similar elementary lines to the floating point accelerator, is the concept of the multiple

precision coprocessor. The standard rule of thumb is that going to double precision increases the total CPU time by as much as a factor of five. Thus, a processor that traps all double precision floating point instructions and sends the data to a special fast sixty-four bit chip set could improve the cost effectiveness quite a bit when it is known in advance that double precision is required. This pattern is quite general--the more that is known in advance about the details of an algorithm, the more the efficiency boost that is obtainable from specialized hardware.

As noted above, coprocessors that can give truly worthwhile improvements over a bare multi-microprocessor system are those which handle some kernel of the algorithm that consumes almost all of the CPU time. For example, the MARYLIE program² spends most of its time in a Newton's algorithm which involves inverting matrices and other steps. A hardware device to do this might easily run fifty times faster than the microprocessor. Another instance of an identifiable kernel is the expansion of high order polynomials in several variables used to find the "kick" from a magnet element field expressed as a multipole expansion.

An alternative to creating a separate device for each such "hardware subroutine" might be to design a single microprogrammable coprocessor, and tailor the coding to the algorithm kernel, picking up a factor of "only" ten to twenty over the microprocessor. Several design versions of this approach have been studied by the ACP group for experiment reconstruction applications. They appear to be potentially equally relevant to the accelerator transport problem.

Another such design is being studied by W. Foster.³ This device also represents an example of a proposed stand-alone specialized processor. The idea is to hook the units directly to the host computer, without using the nodes (thus saving some cost). This approach will be significantly less user friendly, however, than attaching the units to the FORTRAN programmable nodes, where they would be callable as subroutines.

Both these types of specialized microprogrammed devices are very similar in concept to array processors such as those sold by Floating Point Systems, Inc. or Star Technology, Inc. Greater cost effectiveness for the "home built" systems will be obtained by reducing pipeline, memory, and other complexities and avoiding commercial profit and overhead. There may be benefit in experimenting with commercial array processors before committing to a specialized device of this type.

Thus, for problems with a suitable kernel, specialized coprocessors based on commercial high speed VLSI multipliers can get up to a factor of fifty improvement over a bare microprocessor system (which is itself a factor of fifty improvement over the VAX standard). This may not be the end of the road. Using memory look up (MLU) techniques, one can typically compress the calculation of arbitrarily complicated functions into a 50-300 nanosecond look up or about one microsecond if the look up is followed by a hardware interpolation procedure to extend precision.

Linear interpolation gives accuracy to twice as many digits as the table precision, δ , as long as the function tabulated has the property that all of its n -th derivatives are small compared with n/δ^n . This is clearly true for transport "kick" functions, and we can confidently store a k -bit table to get $2k$ -bit accuracy. When many kicks are concatenated (for example using

As a result, the preliminary ACP specifications, which we will describe here very briefly, can be used as a guide to what is practically attainable whether or not ACP modules are eventually used.

The basic ACP node, with examples of several possible adjunct modules attached, is shown in Figure 4. The CPU is a sixteen (soon thirty-two) bit microprocessor with a good FORTRAN-77 compiler. The anticipated relative power for real FORTRAN programs (based on benchmarks and well understood scaling factors) of commercially available microprocessors (as a function of year) is given in Table I. The CPU cost, dominated by "real estate" costs (power, board, interface, bussing, crate, etc.), will be static at approximately \$1,000. Another approximately \$1,500 will be required for each 1.5 megabytes of node memory for program, parameters, and data in 1985; less in later years.

The basic CPU-memory unit sits on a several tiered bus structure. The local bus is connected only to the devices associated with one node and includes commercial floating point or transcendental function coprocessors when required. The global bus is connected to all nodes in a particular group or rank. In principle, the global bus allows communication between nodes as well as with other ranks and the host computer via the rank interconnect module. In practice, it will be wise to avoid arbitration circuitry and limit the nodes to slave-mode communication on the global bus to the host (or other ranks). Special purpose coprocessors may reside on the local bus if they don't require high speed off-board communications (to high speed memory, for example). Otherwise, they will work on a special, short "super bus" connected via a fast memory formatter module to the local bus.

Provision for relatively low transmission rate interconnection to nearest neighbor nodes (for example in grid configurations similar to that shown in Figure 2) could be made through a nearest neighbor interconnect module. Finally, very slow communication to other nodes and large data bases could take advantage of the serial bus capabilities provided by commercial bus systems.

Table II gives typical data rates for the various possible types of communication channels. For purposes of rough system estimates, it is reasonable to assume that the various interconnect modules (rank interconnect, fast memory formatter, nearest neighbor interconnect) will cost about \$1,000 each in production.

Table II

Channel	Speed (Mb/sec.)
Global Bus	20-40
Local Bus	20-50
Serial Bus	1.25
Nearest Neighbor	8
Super Bus	80 +

We must emphasize that although we have catalogued here a complicated set of modules which are relevant in the long term, the ACP focus for at least

the next year is on a simple subset of the full system. This will be based on commercially produced bus equipment for a local and global bus. Specialized coprocessors, nearest neighbor interfacing, etc., are particularly interesting areas for emphasis in 1985 and beyond. There are at least two commercial bus standards recently announced that correspond to the ACP criteria of Table II. These are Intel's MULTIBUS II and Motorola's VME/VMX. Clearly, the basic two tiered bus concept has been widely recognized as important since it keeps local traffic between a CPU and its memory and coprocessors from saturating a global bus connected to all nodes.

Given the potentially available hardware, there are several configurations that are relevant to accelerator modeling. We list a few here that come to mind with some of their advantages and disadvantages.

Figure 3 shows a configuration already referred to which has the important advantage of great simplicity. The disadvantage is that the entire lattice must be stored in each node's memory and for high order calculations, this could prove to be costly.

In another use of the configuration shown in Figure 3, each node calculates a section of lattice; information about particles is passed from node to node over the global bus. A number of rays equal to the number of nodes follow each other through the system in a "systolic" or pipelined fashion. The advantage is again reasonable simplicity, but with smaller memory requirements on each node. The disadvantage is that the amount of processing done in each node must be large enough that the global bus is not saturated. In particular, with one hundred nodes and one thousand elements, this configuration loses in efficiency unless the calculations require at least eighty floating point operations to track a particle through one element. (For six dimensional phase space, even linear kicks satisfy this criterion.) Also, there is some added complexity involving arbitration for the message passing on the global bus.

Figure 5 shows a configuration (first suggested by I. Gaines) that uses nearest neighbor interconnection to pass messages rather than the global bus (here not shown, but implied). This is clearly a natural configuration to represent a ring. It allows memory requirements to be minimized with less restriction on the ratio of processing time to message length. Here the added complexity is buried in the nearest neighbor interconnect module or equivalent.

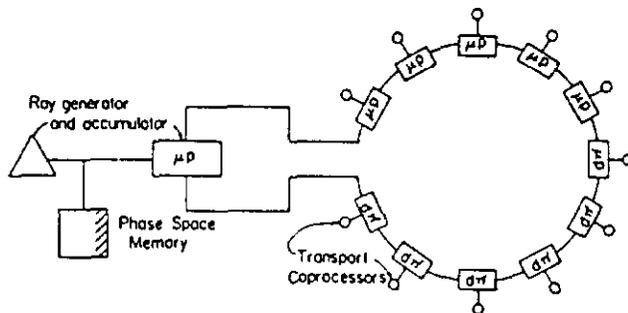


Figure 5: Single ring configuration for accelerator simulation.

Beam-beam interactions can be dealt with (in principle) in any of these configurations. How this is done is conceptually clearest when looking at the ring configuration. As shown in Figure 6, a special device (probably a standard CPU) acts as a phase space accumulator for each ring. Running information on collective phase space parameters is made available to the appropriate node(s) of the other ring. There, software (or a hardware subroutine coprocessor) is used to give the appropriate kick to passing rays.

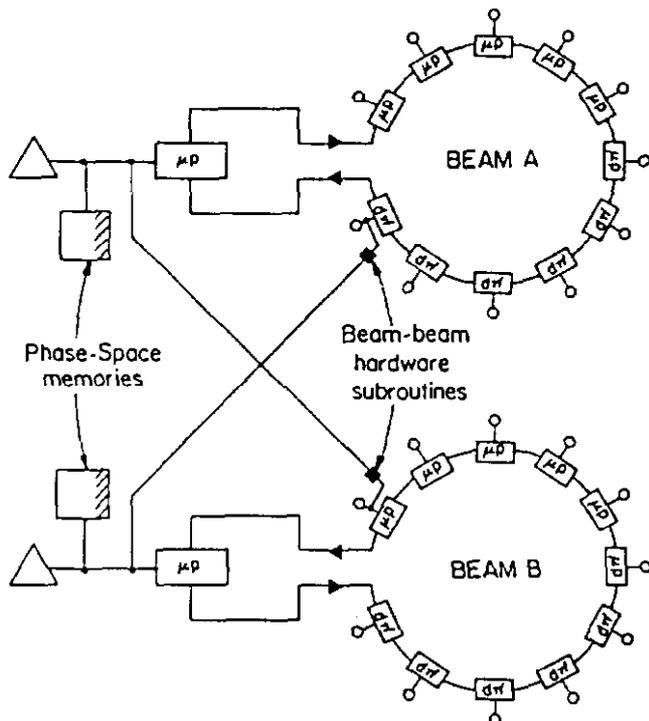


Figure 6: Double ring configuration for beam-beam calculations.

The accelerator transport calculation is, as we hope we have shown, likely to benefit strongly from new approaches to computing intensive problems. For best advantage it is clear that algorithm development will have to be done with a good knowledge of the capabilities and configurability of the hardware. This implies new ways of thinking, which, we hope we have shown, won't be too painful.

Acknowledgment

We would like to acknowledge the many useful discussions with (and suggestions from) other participants at the Ann Arbor SSC workshop, including (particularly) Alex Dragt, Don Edwards, Bill Foster, Jim Niederer, Sho Ohnuma, Ron Ruth, Bob Siemann, and Albin Wrulich. Much of our thinking prior to the workshop, and, of course, the ACP multiprocessor concepts, are due to many discussions with our colleagues, Hari Areti, Ed Barsotti, Don Husby, and Irwin Gaines.

Notes

1. Availability dates indicate that with some optimism and procurement aggressiveness, there is a reasonable probability of obtaining a working system in the first half of the stated year.

2. Major development effort: 5-10 people, 2 years; Medium: 2-3 people, 1 year; Small: 2-3 people, 6 months.
3. Typical size of a multiprocessor group is given here based both on system design issues and on total system costs in the \$300-600K range. In some cases, a design maximum is given based on address space or other known limitations.
4. Host and peripheral costs refer to the computer and its storage that are used in production to feed and gather data. In some future systems such production hosts will not be appropriate for program development which will have to be carried out on another machine. Estimates assume accelerator problems do not require very high speed tape facilities, but do require disk storage of reasonable capacity.
5. Total group cost includes host and peripheral cost and individual node production cost for the typical number of nodes in a group given in Column 6. Development costs are explicitly not included and should be expected to be large if commercial or laboratory developed systems are not available.
6. Total computing power for typical accelerator transport calculations in FORTRAN divided by total group cost, normalized to VAX 11/780.
7. Based on twenty-pole calculation in six dimensions with strong variable coupling only between $(x \text{ and } x')$ and $(y \text{ and } y')$. This example is intended only as an indication of the levels of cost effectiveness obtainable with this method. Actual results will be dependent on the algorithm.

References

1. Paul Kunz, Mike Gravina, Gerard Oxoby et al., "The 3081/E Processor," Proceedings of the Three Day In-Depth Review on the Impact of Specialized Processors in Elementary Particle Physics, Padua, Italy, March 23-25, 1983.
2. Alex J. Dragt and David R. Douglas, "Particle Tracking Using Lie Algebra Methods," University of Maryland Preprint 84-041 (1983).
3. W. Foster, private communication, Ann Arbor Workshop.
4. R. Ruth, private communication.
5. J. Niederer, private communication, Ann Arbor Workshop.
6. A. Dragt and R. Ruth, private communication, Ann Arbor Workshop.
7. N. Christ and A. Terrano, "A Very Fast Parallel Processor," Columbia University Preprint CU-TP-261 (1983).