Final Report:

Processor-In-Memory (PIM) Based Architectures for PetaFlops Potential Massively Parallel Processing

NASA Grant NAG 5-2998

July 15, 1996

Dr. Peter M. Kogge
McCourtney Professor of Computer Science and Engineering
IEEE Fellow, IBM Fellow (retired)
University of Notre Dame
Notre Dame, IN 46556
219-631-6763
fax: 219-631-9260
email: kogge@cse.nd.edu

Final Report:

Processor-In-Memory (PIM) Based Architectures for PetaFlops Potential Massively Parallel Processing

NASA Grant NAG 5-2998

July 15, 1996

Dr. Peter M. Kogge
McCourtney Professor of Computer Science and Engineering
IEEE Fellow, IBM Fellow (retired)
University of Notre Dame
Notre Dame, IN 46556
219-631-6763
fax: 219-631-9260
email: kogge@cse.nd.edu

1. Introduction:

The report summarizes the work performed at the University of Notre Dame under NASA Grant NAG 5-2998 from July 15, 1995 through July 14, 1996. Researchers involved in the work included the PI, Dr. Peter M. Kogge, and three graduate students under his direction in the Computer Science and Engineering Department: Stephen Dartt, Costin Iancu, and Lakshmi Narayanaswany.

The organization of this report is as follows. Section 2 is a summary of the problem addressed by this work. Section 3 is a summary of the project's objectives and approach. Section 4 summarizes PIM technology briefly. Section 5 overviews the main results of the work. Section 6 then discusses the importance of the results and future directions.

Also attached to this report are copies of several technical reports and publications whose contents directly reflect results developed during this study. This includes a paper that will be published in the Frontiers of Massively Parallel Computing Conference in Annapolis, MD in October, 1996. Several other papers are under preparation for submission for formal publication. All these published papers will acknowledge the support received from this grant.

2. The Problem Addressed by this Study:

With the achievement of real *teraflops* (10^{12} floating point operations per second, TF) computers in sight, it is now time to begin focusing on the next major level of processing: a *petaflop* (10^{15} operations PF) per second. Such machines could compute in 15 minutes what would take a TF machine over 10 days, or a high end *gigaflops* (10^{9} operations per second - GF) machine over 30 years. As was shown in the 1994 Cal Tech Workshop on Petaflops Computing, and the 1995 Petaflops Frontiers Workshop (both including sponsorship by NASA, with significant organizational responsibilities by Dr. Thomas Sterling, then of NASA CESDIS), there are a great many applications of significant scientific and economic benefit that would be opened up by such levels of computing. Further, the same technology that would enable these levels of computing would also enable placing today's teraflops machine room in a desk environment, and today's gigaflops rack of computers into handholdable packages. Both of these size reductions have huge potential societal impacts, especially as we move to a more and more mobile society where portable communications and information exchange of all sorts over gigabit networks will become the bedrock of the business environment.

The major problem with achieving such levels of performance today is cost. Even using the best of today's technology would require 10's of millions of chips and literally 10's of billions of dollars to reach a petaflops machine. This is simply not a viable option. However, projecting technology ahead, as was done in the 1994 Workshop, indicates that within approximately 20 years the state of the art in VLSI should make such machines feasible, with at least three different computer architectures reasonable candidates.

Given the above result, why should any attention be paid in 1995 or 1996 to petaflops level machines? The answer is two-fold. First, all the architectures proposed in the 1994 Workshop for petaflops were truly *Massively Parallel Processors (MPP)*, an approach becoming widespread at the very high end of existing machines but proving tough to handle, and virtually unstudied for lower level everyday applications, and not at the level of 10s to 100s of thousands of semi-independent processors (non trivial non SIMD engines). Second, the architecture with the greatest initial potential in terms of minimal chips, and thus cost, is one where multiple entire, **relatively simple but not trivial,** computers are integrated onto single chips that combine the state of the art in memory density with very significant amounts of high speed logic. For obvious reasons the architecture of such chips is becoming known as *Processing-In-Memory* (PIM).

In today's world of superscalar, superpipelined, multiple execution unit, 10 million+ transistor single CPU microprocessor designs, both of these observations are so far out of the mass production norm that without a jump-start, neither will mature to the levels needed by the time the technology is ready for a petaflops. Consequently, what is needed is a believable roadmap projecting no only what appropriate PIM architectures might look like 20 years from now, but **what are the intermediate stops along the way**. This includes what can be done with today's technology, or the technology of 1998 or 2001, for example, to implement such PIM-based MPP machines, and how does this compare with where the conventional trends will take us.

Very preliminary projections [8, 9, 10] based on the first real PIM chips such as EXECUBE [5, 7, 11] indicate that in terms of silicon usage, PIM offers at least an order of magnitude better advantage than other approaches. Given the potential simplicity of PIM designs versus modern microprocessors (EXECUBE cost less than $5M to develop versus hundreds of millions for the very leading edge conventional chips), this means that PIM-based MPPs have the potential to not only affect very high end computing 20 years from now, **but radically change the way we embed very high levels of processing in the mobile world of the late 1990s'!**

3. Study Objectives and Approach:

Given the above observations, the goal of this study was to develop an initial roadmap to achieving petaflop using PIM-based MPPs, with a emphasis on early and continuing design points that would identify both the technologies needed to achieve a petaflop, and intermediate applications, including embedded ones of interest to NASA, that would provide the economic impetus to fund more fully the development of those technologies.

In particular, as described in the original proposal this work was to consist of the following:

1. Interactions with applications experts and the developing petaflops research community (both within and outside of NASA) to validate that architectural choices are not only chip-efficient, but also are reasonable targets for real applications. Such applications will include not only petaflops level ones, but lower performance, especially embedded, applications which may be uniquely enabled by PIMs.

2. Use of existing CMOS VLSI technology trends to identify the design spaces, in terms of amount of memory and logic, possible for PIM chips at regular intervals over the next 20 years. Constraints such as off chip contacts, power dissipation, die size for economic production, etc., were all a part of this characterization.

3. Architectural tradeoffs to identify which type of individual processor node architectures best utilize the silicon and memory bandwidth present on a VLSI chip at each of the above time intervals. This included not only existing architectures (from the very simple to the more complex), but also novel ones that more fully utilize chips with significant memory and logic.

4. Development of candidate MPP system architectures using potential PIM chips identified above, and derivation of key characteristics such as limits and ease of scalability, number of chips, power, needed runtime and support software, etc.

The statement of work also assumed participation at two workshops on petaflops over the year of study.

4. PIM Background

The thesis behind PIM is that many of the problems facing very high performance MPP computers stem from the "traditional" von Neumann bottleneck, and how we have approached it. For technology and cost reasons we have historically separated memory parts and CPU logic parts. With the advent of CMOS microprocessors with very high performance pipelined and superscalar architectures, individual CPU core performance levels have gone through rapid acceleration, requiring ever increasing amounts of bandwidth from the memory subsystem. These rates have far exceeded the bandwidth capabilities of our densest DRAM (Dynamic Random Access Memory) memory parts (needed to control system costs), and the gap will continue to widen over time. The net result is architectural complexity: memory hierarchies are introduced to provide the bandwidth, which in turn drives cost and the software complexity needed to address these hierarchies efficiently, especially in an MPP environment.

The PIM technology is emerging to counter this fundamental defect - namely the combination on one chip of both dense DRAM memory and very significant amounts of logic. This capability permits new architectures to place computing either right next to, or even inside of, the memory macros, where there are huge amounts of raw memory bandwidth.

Many chips today combine logic with some form of memory. At one extreme most modern microprocessors combine millions of transistors of logic with a few tens of KB (Kilo Bytes) of SRAM (for caches). At the other extreme, most conventional DRAMs combine MBs (Mega Bytes) of memory with a few thousands of transistors for address decoding for the internal arrays, latching at the sense amps, and multiplexing to drive the data lines.

PIM chips fall in the middle. They combine large amounts of both memory and logic. However, the key feature that distinguishes them from the conventional chips is that they represent potentially self-contained designs where all the processing functions and all the memory for that processing for one or more nodes are on the same chip. This self-contained characteristic has several key consequences:

- It is possible to conceive of a single part type scaleable MPP design where additional computational resources are added by adding more of the same kind of chips (much as today we add more memory to PCs via plug in SIMMs).
- This in turn permits consideration of novel 3D packaging techniques which both reduce overall system costs and provide shorter chip to chip paths, reducing the other killer of MPP performance - latency.
- Placing the processing logic next to the memory permits a huge increase in the percent of raw memory bandwidth that can be utilized from the memory arrays over today, where at best a few percent of the total bandwidth is presented to the off chip pins. This can reduce or eliminate the need for complex caching and other tricks in the design of the processing logic - again reducing both cost and latency.
- Eliminating from the processing chips the need for often hundreds of pins to support a memory hierarchy means that these same pins can be used to perform something computationally useful - namely communication with other processing nodes.
- Moving multiple processing nodes to a single chip also allows consideration of new architectural techniques such as mixed SIMD and MIMD processing, very high bandwidth memory to memory transfers, and "in the memory array" processing, all of which in turn should be useful to significantly reduce application program complexity.
- A combination of all these techniques offer the potential for greatly reduced power dissipation per unit of performance. Simplicity in logic means that there is less logic to burn power; reduced pin count to support memory hierarchies yields less power, and higher memory bandwidth and lower latency next to the processing logic implies the potential for lower clock rates - again lowering power.

The first such PIM chip to utilize state of the art dense DRAM, EXECUBE [5, 8, 11], integrated onto a 4 Mbit DRAM chip 8 complete CPUs configured in a 3 dimensional binary hypercube, and was used as a one chip type building block for MPPs of both MIMD and SIMD organization. On a chip basis, it was up to 10 times more efficient than traditional techniques in terms of performance per chip over an entire system. At 2.7 watts typical dissipation, it also demonstrated extraordinary performance per watt (again on a system wide basis) for its time frame and implementation technology.

5. Overview of Results

Very significant progress was made on all of the topics listed in Section 3. The following subsections overview the results. In many cases more results are available in the attached reports, with other results to be reported in documents currently under preparation. The order of presentation starts with a brief review of the interactions held with the petaflops community; these meetings will then be referenced in later sections when outputs from them directly affected the results of the study. Following this is a summary of what was learned about petaflops applications requirements, the design space offered by the projected CMOS trends, the potential high level architecture of a strawman PIM chip was investigated, and then how that architecture could evolve into a complete system. This latter section also addresses some issues associated with software: both tools and runtime.

5.1 Interactions with Petaflops community

As part of this effort, we participated in not two, but three different petaflops related workshops:

1.Petaflops Summer Study on Algorithms in Bodega Bay, CA in August 1996. At this workshop we presented some early results from the PIM projections spreadsheet model described later, with emphasis on the significant advantages that PIM held, along with the characteristics that a PIM-based petaflops machine might have, in particular a degree of parallelism in the 160,000+ range. Feedback from this study provided more insight into exactly how much primary and secondary memory, I/O bandwidth, and expected running time were reasonable to assume for petaflops applications.

2.Petaflops Architecture Workshop in Oxnard, CA, in April 1996. The results of this study were used in two separate presentations at this workshop. First, the requirements learned from the Bodega Bay workshop were presented early in the workshop to provide an overview of what was required by <u>any</u> petaflops capable machine, not just the PIM approach. Second, a more detailed presentation was made on the PIM strawman architecture described above, with more realistic CMOS technology constraints taken into account. Outputs to this study were a growth in the architectural design space for PIM chips, and a recognition that PIM could be applied in much more general fashions than had been earlier anticipated.

3.Petaflops System Software Workshop in Bodega Bay in June 1996. Although no formal presentations on PIMs were made at the workshop, it became clear that the PIM approach, especially when coupled with its insertion into a conventional system as "smart memory," held the promise of greatly simplified software (both tools and runtime support) over the more traditional approaches discussed at the conference, and the problems that those approaches faced when scaled to the petaflops region.

Finally, a meeting is planned in early August with interested government agencies as to the next step for the development of petaflops machines in general, and how PIM might play a role in that.

5.2 Applications Requirements

Before one can size a machine of <u>any</u> possible architecture to reach a petaflop, one must have first a set of approximate requirements to work with. The 1995 Summer Petaflops Algorithm Workshop in Bodega Bay, CA provided just such a basis by summarizing from over 30 possible applications that would profit from a petaflop machine a variety of characteristics, including the running time for a typical execution (when on a petaflop per second machine), the primary storage (memory) needed to support the application, the secondary storage (disk) needed, and the I/O rate. Given the cost of memory even in the 2010s, the primary storage figure is most telling. Figure 1 diagrams this as a scatter plot where each dot corresponds to a particular application, positioned on the graph as a function of its execution time for a single run (in seconds, on x-axis) and primary memory (in GB, on y-axis). Three different symbols are used here: one for applications requiring less than 100 seconds (about 2 minutes), one for applications with execution times between 100 and 10,000 seconds (about 3 hours), and one for in excess of that.

As can be seen, the execution time spans eight orders of magnitude, and the memory requirements span six orders of magnitude (1E3 GB is a terabyte TB; 1E6 GB is a petabyte PB). There are significant numbers of applications in each of the time frames, but no clear correlation between running time and primary memory.

Figure 2 perhaps gives better insight into the question of "how much primary memory is enough?" Roughly half the applications could be satisfied by a TB, and 80% by 30 TB. This latter number of 30 TB also corresponds well to an estimate made at the 1994 Petaflops Workshop that

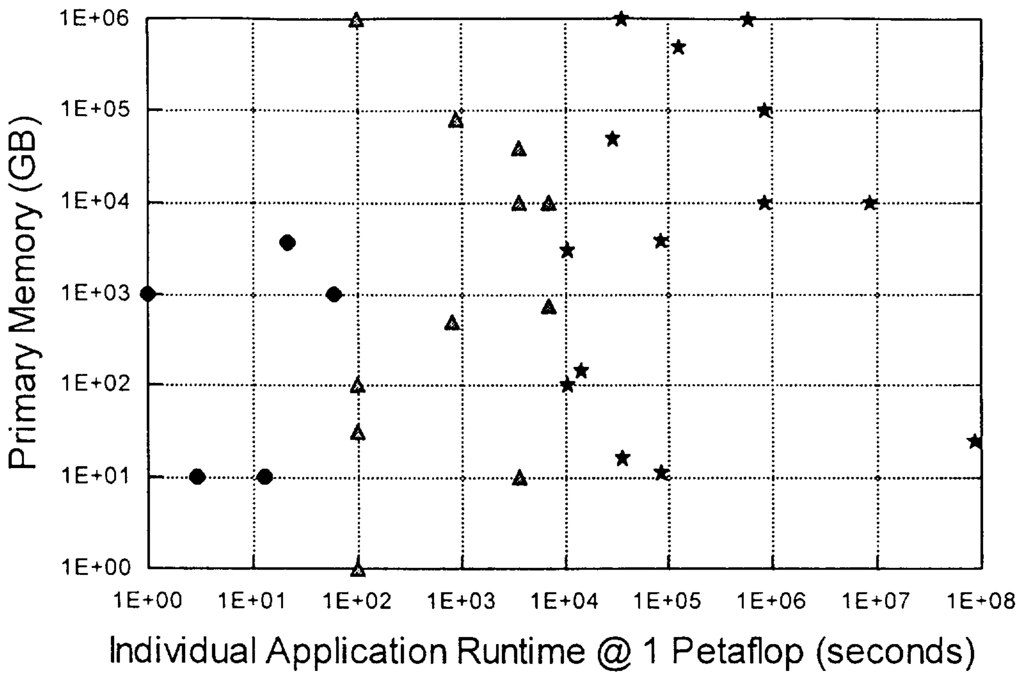

Figure 1. Petaflops Applications: Runtime vs Primary Memory Requirements

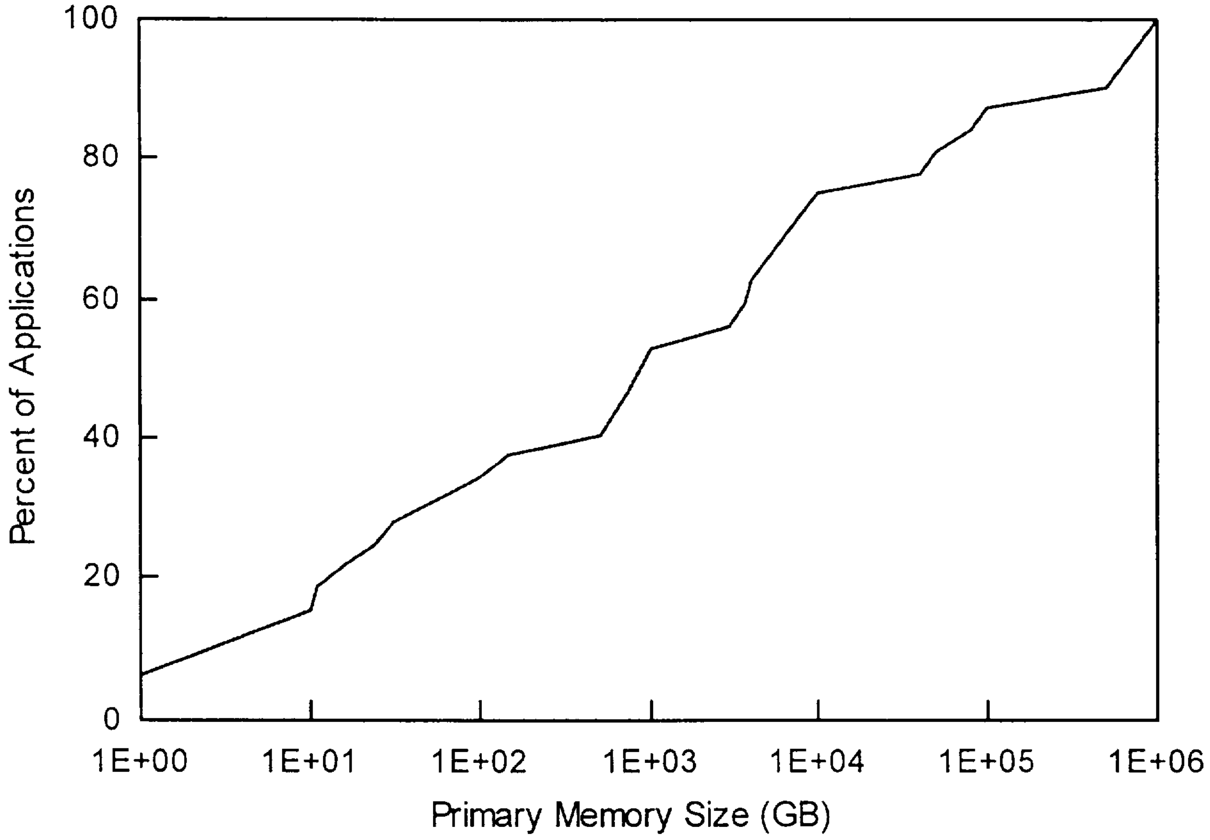

Figure 2. Percent of Bodega Bay Petaflop applications requiring different memory sizes.

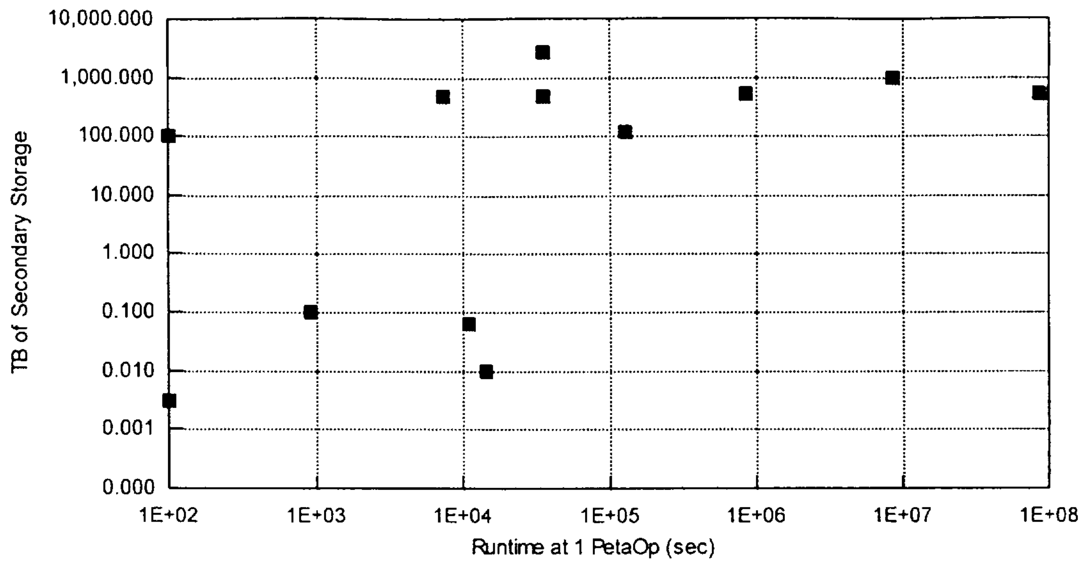

Figure 3. Secondary Storage Needs vs Running Time

there were significant problems (especially 3D + time simulations) where primary memory needs grew as $N^{3/4}$ as performance grew as N (in GF). For 1 million GF (1 PF) this corresponds to 32 TB.

Another system requirement of real interest is the amount of secondary storage needed to support an application. Here the 1995 Bodega Bay workshop was only able to estimate a need for only a subset of the applications studied in Figure 1, with the results summarized in Figure 3, again as a scatter diagram vs running time. A great many of the applications require upwards of a PB.

Figure 4 diagrams one other useful piece of information: I/O rate required by the application. Again this is drawn vs running time. This comes in two parts: an internal I/O rate needed to support the application, and an I/O to reload the memory for the next application execution (labeled IPL in the diagram). The latter number was computed by this study, and assumed that the next application required as much memory as the current one, that all of this memory needed to be loaded, and that it could be loaded concurrently with the current execution. This latter assumption would also force us to approximately double the size of the memory, which may be a costly undertaking.

For the applications for which data was available, an application I/O rate of about 100 GB/sec seemed to be satisfactory, as is an IPL rate of a few ten's of GB/sec.

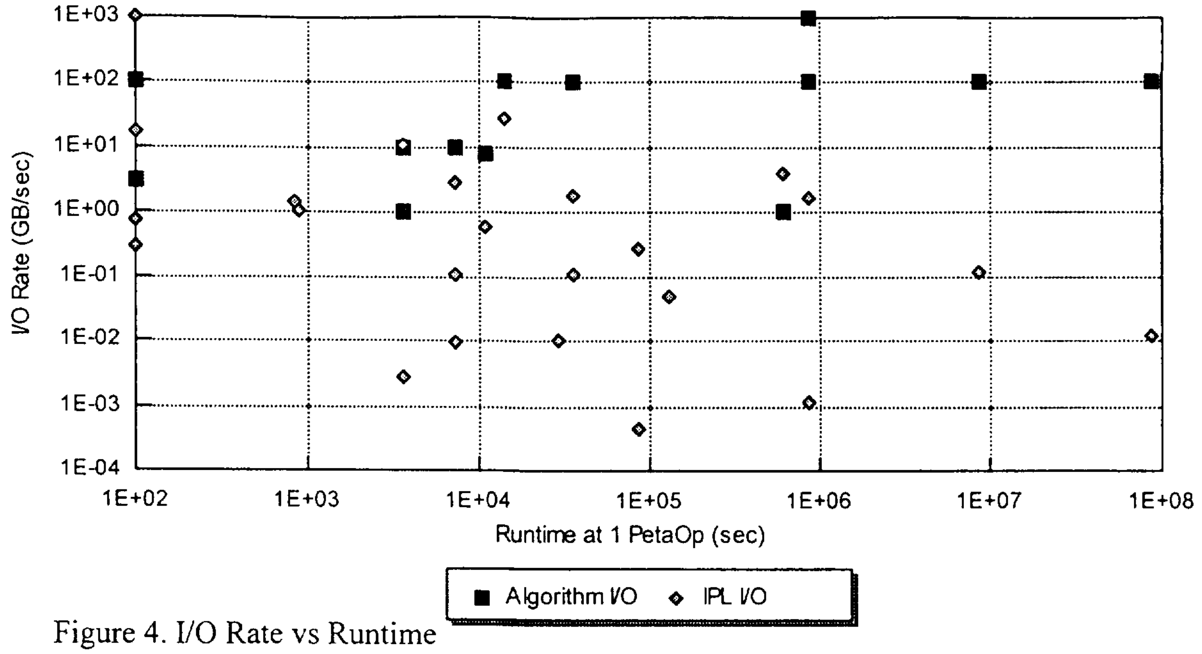

Figure 4. I/O Rate vs Runtime

5.3 PIM Technology Design Space

A major part of the study was to explore the use of CMOS VLSI to identify trends and capabilities in the PIM *Design Space.* This was done primarily by developing several spreadsheets based on the most well respected projection of CMOS technology available at the time, namely the *1994 National Technology Roadmap for Semiconductors* published by the SIA [13]. These spreadsheets utilized the SIA numbers to perform "partitioning" experiments to determine how much logic should be placed on a PIM chip in combination with how much memory.

A fairly complete description of the major worksheet so developed as of April. 1996, is given in Appendix B - a section of the Petaflops Systems Workshop draft proceedings from April, 1996 produced from this study's results.

Rather than repeat the spreadsheet descriptions here, an more instructive summary would be to go through the analysis logic that led to the spreadsheets and their detailed structure. We will start this summary by assuming the following time dependent functions which affect PIM potential designs:

- *M(t)* is the amount of memory (in MB per square centimeter) that can be placed on a chip as a function of time. This comes from the SIA Roadmap.

- *L(t)* is the amount of logic (in millions of transistors per square centimeter) that can likewise be placed on a CMOS chip as a function of time. This also comes from the SIA Roadmap.
- *C(t)* is the clock speed that the above logic can run at, again as a function of time. Again, this comes from the SIA Roadmap.

The SIA roadmap includes many variations for each of these. For memory it can be DRAM or SRAM; for logic it can be full custom or ASIC. For clock speed, the full custom can run at either very high speeds such as might be found in the leading edge microprocessors or in very specialized digital signal processors, at a lower rate more characteristic of commodity microprocessors, or at lower speeds more characteristic of ASIC logic.

Likewise the roadmap discusses off chip pin contacts, and transmission speeds on such pins.

For the purposes of this discussion we will assume DRAM memory and ASIC logic together on the same chip. This is to reflect the fact that at least for the foreseeable future, placing both dense DRAM and the highest density logic on the same chip would require combinations of two very distinct fabrication processes, greatly increasing the cost of the chip. By assuming ASIC style logic, we are focusing on a lower cost technology which can be built on the basic DRAM process, and which, given the volumes of parts needed for a petaflop, is probably necessary to control costs. The spreadsheets developed as part of this study, however, allowed assuming almost any of the above combinations to be specified.

As an approximate reference, for the above choices the SIA Roadmap gives curves which roughly follow the following equations (where t is uniformly the year of technology introduction):

- $M(t) = 4*1.4^{(t-1995)}$ MB per square centimeter
- $L(t) = 2*1.22^{(t-1995)}$ million transistors per square centimeter
- $C(t) = 150*1.1^{(t-1995)}$ MHz clock rate for the logic

To a first order, the PIM chips assumed here will cover a chip with a multiplicity of some standard processing logic (a "CPU" in the following discussion), with some amount of DRAM memory available to each one. Further, although it is likely that the architecture chosen for any particular PIM implementation will evolve with time, we will assume for this analysis that it is fixed, and that to implement one such CPU node will require *P* transistors at any point in time, and that at a clock rate of C_0 such a processor will achieve a "unit" level of performance, in whatever measurement scheme is most appropriate.

Since it is infeasible to build a single chip at any time in the foreseeable future that could contain both the memory and enough logic to support fully any of the applications of Figure 1, we have to assume multiple identical chips arranged in some sort of a parallel system. This assumption in turn requires that if we look at any single chip, the *ratio* of memory to processing power provided by the logic <u>must</u> match that needed by the overall system. Otherwise, a system that minimally meets either the memory or performance needs may be starved in the other direction. (In reality, if we were to err, it should be on the side of extra memory, since parallelizing functions are notorious for requiring extra buffers, array copies, thread contexts, etc. to keep the processors humming).

With the above comments in mind, for this analysis we will assume this ratio of memory to processing is R, in units of MB per unit performance, and that at time t, the fraction of a chip that is implemented as logic rather than DRAM is F(t). For a 32 TB petaflop system such as sized in the prior section, this means that R must correspond to a ratio of 32 MB per GF, when the equivalent performance of a unit CPU is converted into flops.

Given all these variables, at time t the performance of a single CPU in standard units is $C(t)/C_0$, and thus there are $L(t)*F(t)/P$ such CPUs on an average square centimeter, with a peak performance of $F(t)*L(t)*C(t)/[P*C_0]$ units of performance. This means that there are $M(t)*(1-F(t))$ MB of memory on the same square of silicon. The rate of memory to performance is thus $M(t)*(1-F(t))/[(L(t)*F(t)/P)*(C(t)/C_0)]$. If this is to be equal to R we must have:

$$R = P*C_0*M(t)*(1-F(t))/[L(t)*F(t)*C(t)]$$

Solving for F(t) we get:

$$F(t) = P*C_0*M(t)/[R*L(t)*C(t)+P*C_0*M(t)]$$

$$= 1/[1 + (R/P*C_0)*L(t)*C(t)/M(t)]$$ as an alternative representation.

Given this we can compute the peak performance p(t) and amount of memory m(t), per square centimeter of PIM, as:

$$p(t) = F(t)*L(t)*C(t)/[P*C_0]$$

$$= M(t)*L(t)*C(t)/[R*L(t)*C(t)+P*C_0*M(t)]$$

$$m(t) = M(t)*(1\text{-}F(t))$$
$$= R*M(t)*L(t)*C(t)/[R*L(t)*C(t)+P*C_0*M(t)]$$
$$= R*p(t)$$

Inserting the approximations from above:

$$F(t) = P*C_0*4*1.4^{(t-1995)}/[R*2*1.22^{(t-1995)}*150*1.1^{(t-1995)} + P*C_0*4*1.4^{(t-1995)}]$$
$$= P*C_0/[75*R*0.96^{(t-1995)} + P*C_0]$$
$$= 1/[1 + 75*(R/P*C_0)*0.96^{(t-1995)}]$$
$$p(t) = 300*1.34^{(t-1995)}/[75*R*0.96^{(t-1995)} + P*C_0]$$
$$m(t) = R*300*1.34^{(t-1995)}/[75*R*0.96^{(t-1995)} + P*C_0]$$

These fractions can vary all over the map, from almost 1 to astonishingly small, depending on R, P, and C_0, with R the most sensitive of the parameters The spreadsheets developed during the study implemented extensions of them to take into account additional factors. The key results will be given later.

5.4 PIM Architectural Design Space

PIM chips mix memory and logic on the same die, and use the logic to implement processing functions. The F(t) measure defined above indicated what percent of a square of silicon, on the average, was devoted to logic. Given this, plus our desire to keep to a primarily single part type system, there were four potential generic ways that we deemed possible to use this capability:

1. Pure *Single Instruction Multiple Data* (SIMD) PIMs where multiple relatively simple function units nestled very close to the memories on chip, and performed more or less the same operation simultaneously.

2. Single CPU chips where a single microprocessor is married directly to a large chunk of memory on a single die.

3. *Symmetric Multiprocessors* (SMP) chips where multiple CPUs are placed on a single chip, with memory, and with coherence logic to maintain identical internal memory representations.

4. "Tiled" chips where a regular pattern of processing logic with associated separate memory macros is laid down over the chip.

The first was excluded from our consideration because of the perceived difficulty of finding huge amounts of SIMD-only parallelism in real applications.

All of the latter three are real possibilities for potential petaflops machines, but only the last received significant attention in this study. Number 2 in the list, a single CPU, was deemed to be a bad fit, especially as memory densities grew quite large, and as our ability to gain more performance by simple replication than by over designing a "super" CPU. The discussion in the next section will also show a disturbing trend in CPU designs where we do not believe the return from a huge increase in transistors warrants the expense.

The third approach, an SMP chip, is entirely feasible, but was not considered because of concerns about the complexity of the cache coherence logic, and the growing belief during the study that the huge bandwidths and short latencies we were beginning to see as possible would make the need for large cache hierarchies largely obsolete. Further, the approach developed during the study seemed to offer many of the same advantages without complex coherency mechanisms. However, during the April Petaflops Workshop there was sufficient new thoughts about such a configuration, especially if the on chip memory is used to implement a *COMA* (Cache Only Memory Architecture), that some variants here are worthy of future consideration.

5.4.1 Processor Node General Architecture

The bulk of the study effort was targeted at refining an overall PIM architecture that mirrored bullet four in the above list, namely a "tiled" approach to placing logic and processing on a single chip, with the goal of reaching a petaflop. By tiling we mean defining a single arrangement of logic and memory, and then replication that arrangement in some very regular fashion over the surface of the chip. The chip design then devolves to designing relatively small memory and relatively small logic pieces, and then managing how they interconnect when placed next to each other on an overall floorplan.

The PIM chip developed in this study had three separate characteristics worthy of discussion:

1.the processing node macro (memory + logic),

2.the "tiling" of a chip with these macros,

3.and the interfaces provided on the chip for communication with other chips.

Figure 5 diagrams an assumed processing node macro. It consists of a rectangular area of logic transistors surrounded by 4 separate memory macros, with each memory macro having its own decoding, sense amps, and row logic.

These memory macros are the smallest replicable "slices" that are possible with the DRAM technology at the time the chip is assumed fabricated. Today such slices are on the order of 128-256 rows of a few hundred to perhaps a thousand columns, with total densities in the order of 32KB-128KB. For electrical reasons, the sizes of these slices is not expected to change significantly over the coming years.

Even with 4 memories per logic area, there still is a limited amount of memory (perhaps only a MB at most). This is nowhere near enough to balance the performance possible from the CPU logic, so we assumed that just as in ordinary DRAMs, these slices could be "stacked" on top of each other, with either edge sense amp to sense amp, or a second level of metal, used to connect them to the CPU. Such a tiling not only reflects current practice, but allows some nice VLSI design considerations to occur, as will be discussed later.

In total, the positioning of these macro stacks also provides several very key benefits to the design and selection of the processing logic:

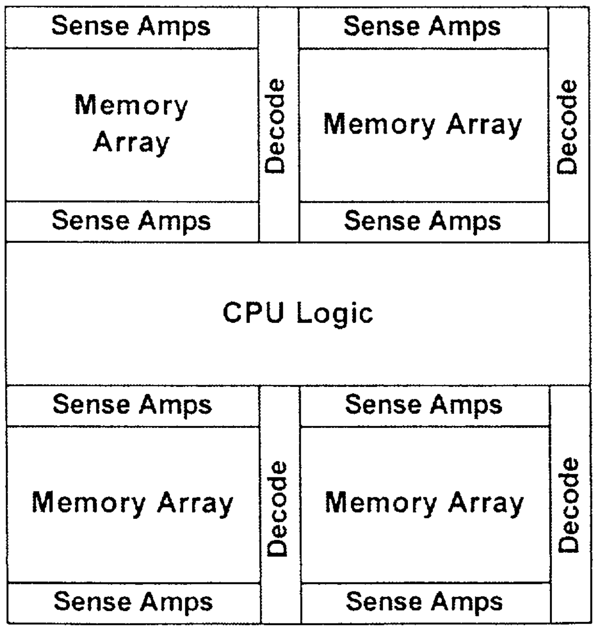

Figure 5. A Single Processing Node Consisting of 4 Memories and 1 CPU

1.There are four of them, providing more memory for the logic, and thus making it easier to provide the proper balance between memory and performance as was discussed before.

2.Turning the DRAM macros "sideways," as shown, with the sense amps facing the logic directly, provides a huge increase in data bits available from each access directly to the CPU logic. It is conceivable that several hundred bits might be so available.

3.Turning the DRAM macros "sideways" also provides a very fast access time, most assuredly under 30 ns. with today's designs (in fact the 1991 era technology in EXECUBE had about a 12.5 ns access time from the address decoders to the sense amps). Some reduction in this time should be expected over time, not only because of technology changes, but also because there is no extra logic in the way to "demultiplex" all the bits from the memory macro down to the handful of bit that go off-chip in a normal DRAM. There is also no need to "split" the address into two parts, RAS and CAS, as is done today. All of these add to improved access latency at the basic macro level.

4.Having four separate stacks of such memory modules provides four times the bandwidth, yielding literally multiple gigabytes of bandwidth per second for each CPU. They also provide the opportunity to have up to four accesses to be concurrent. This could lower the average access time over that if we had only one stack to significantly under 10 ns.

5.If we design the individual memory slices properly, namely place an address latch in each, we could conceivably keep as many concurrent memory accesses as there are slices active within each stack. This can multiply bandwidth and reduce average latency by another huge factor.

5.4.2 Choice for Processing Logic

Nothing in the discussion to date constrains the choice of what the processing logic looks like. If we are going to make the PIMs more or less general purpose, then they should be programmable CPUs. Since logic technology is providing more and more transistors, conventional wisdom is that these CPUs should become more and more complex, i.e. super scalar, super pipelined, multiple issue, with sophisticated branch prediction and out of order execution. All of this requires more bandwidth, so of course this means deep cache support systems.

Conventional wisdom may not, however, be so appropriate in the PIM environment where what we want is to maximize the total amount of performance out of the logic, and we are more than willing to do this with parallelism at the on-chip node level. To investigate this, we surveyed a wide spectrum of modern microprocessors, and accumulated such information as the feature size of the technology used, number of transistors on the chip, the number and complexity of the on chip caches, the clock frequency, the supply voltage, as many benchmark numbers as were available, and as many other factors as were available. For those microprocessors that had a consistent set of benchmark data, namely SPECMARKs, we then translated the data using the following process:

1. To get to the transistor count for the actual CPU core, we took the sizes of all on chip caches and other dense memory hierarchy related structures, and estimated the total number of transistors in them. This was then subtracted from the total chip transistor count to get that for the CPU core alone.

2. To normalize out the effects of technology (at least to a first order approximation), the reported SPECINTs and SPECFLOATs for each chip was divided by the clock rate of the chip. This yields a "SPECS per machine cycle" - analogous to "instructions issued per machine cycle" which is a modern measure of low level microprocessor performance. Note that as technology improved there were multiple cases where the same design would have been implemented in several different clock rates, and each would have a separate set of measurements. We included all such numbers because it often gave insight into second order effects.

3. To get a handle on the "per transistor" effectiveness of the designs, each of the numbers from step 2 above, the clock normalized performance, were divided by the numbers from step 1, the number of transistors in the core of the CPU. The resulting numbers represent an estimate of how much each transistor in the CPU core adds to the overall performance

If one graphs the results from step 2, normalized SPECs, versus the numbers from steps 1, CPU core transistor counts, the graphs are exactly what one would expect. If one adds more transistors to a CPU design, then all else (especially clock) being equal, one ends up with more performance.

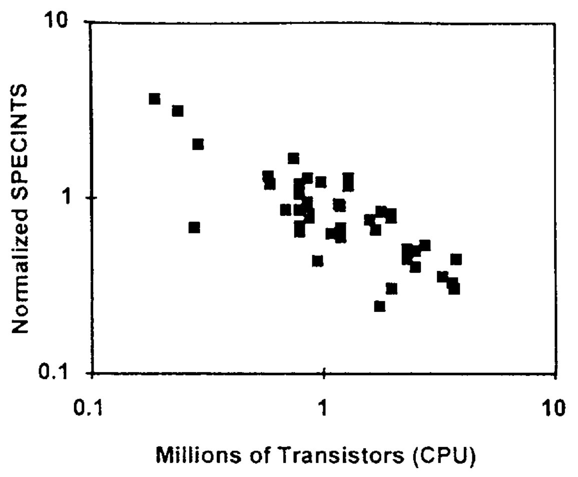

Figure 6. SPECINT Transistor Efficiency.

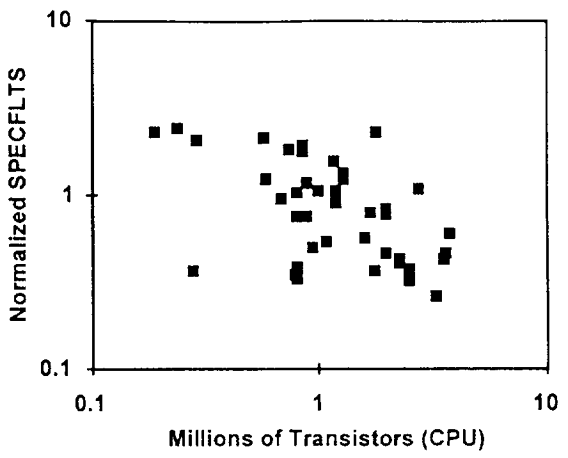

Figure 7. SPECFLOAT Transistor Efficiency

However, the story is different if we graph step 3 versus step 1. Figures 6 and 7 diagram these results for SPECINT 92 and SPECFLOAT 92 respectively (we are still accumulating SPEC'95 data and will publish that shortly). Performance per transistor drops sharply for SPECINTs as the number of transistors added to a CPU increases. This means that if one has a transistor budget to spend on processing, and one is willing to consider parallel CPUs, then by far the most effective designs are not the multi million transistor "leading edge" designs, but the simpler straightforward designs with minimal complexity.

The effect is not so pronounced for SPECFLOATs (because a floating point function unit is approximately the same regardless of the rest of a design), but it is still there, and there is no chip design which is more efficient per transistor than the simplest.

The lesson we took from this result is that the processing logic to assume for our PIM macro should represent a simple CPU design, with a minimum of frills. In fact, for the rest of the analysis presented here we assumed the simplest (upper leftmost) of the designs in Figures 6 and 7, which happened to be the same - a straightforward 5 stage pipelined CPU as is found in a MIPS R3000 or similar chips. Based on the data from the survey, this design incorporates in about 250K transistors both the integer pipeline and a simple but effective floating point unit.

Besides reducing design complexity, this choice has several other key attributes which are particularly advantageous to PIMs:

1.Simpler designs need fewer concurrent accesses to memory to maintain high performance than the more complex ones. In particular there are fewer "speculative" accesses (from branch prediction and the like) which are then never used. This reduces the peak bandwidth needed out of the memory system, meaning that an organization such as Figure 5, with its multiple wide memory macro stacks, can provide the bulk of the bandwidth directly, without additional caches.

2.Fewer transistors (and simpler memory hierarchies) together result in fewer places where power is dissipated, opening up the opportunity for lower power chips. This is critical if we are to build systems with hundreds of thousands of parts.

3.A simpler, pipelined like, design also means the usage of the bulk of these transistors is in very regular structures whose physical placement mirrors the feed forward nature of the organization. This in turn opens up the possibility of still getting good logic packing density with fewer layers of interconnection metal on the chip. In turn, this allows us to consider simpler DRAM-based processes which normally do not offer as many wiring layers as pure logic processes.

5.4.3 Floorplanning by Tiling

Figure 8 now diagrams the way in which this study projected the bulk of a PIM chip ought be floorplanned by the prior CPU and memory stacks. Alternating rows of memory stacks and CPUs fill the chip from the top to the bottom, but with neighboring rows of CPUs offset from each other by the width of one memory stack (i.e. one half their own width). All memory stacks in a single row have the same height, as does all CPUs in the same row.

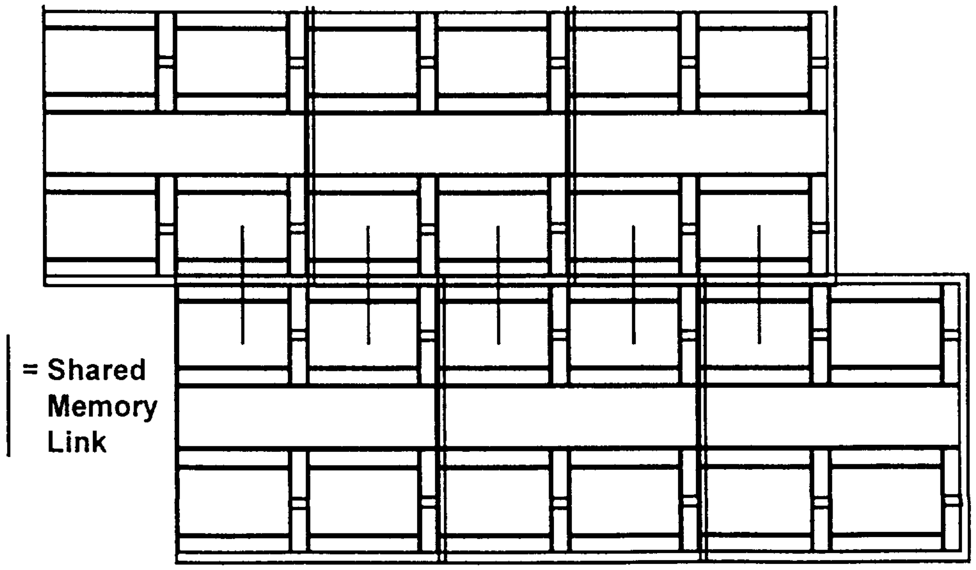

Figure 8. Tiling a PIM Chip with Offset rows of Memory and CPUs

Given that by design each memory stack, regardless of how many slices it is built from, has sense amps at both the top and bottom, each of these rows of CPUs still maintains connectivity to four separate memory stacks, and each stack is connected to two separate CPUs. This results in a true shared memory between two separate CPUs - the best of all possible configurations from a programming perspective (In contrast, chips like EXECUBE did not support shared memory between neighboring CPUs, meaning that all internode communication must be by explicit message passing).

Further, because of the offset between rows of CPUs, the resulting design means that each CPU is sharing memory with not one or two, but four other CPUs. This allows extraordinary amounts of interconnectivity between processing nodes, at the highest possible bandwidths, without complex programming or additional internode communication logic!

Finally, from the standpoint of VLSI chip fabrication, Figure 8 offers a very clear advantage. All the DRAM is separated off into nice, well defined, rows, where it can be placed in its own wells, with its own biases etc., and in general protected from the random signal excursions generated in the logic rows. Further, by using the memory macros themselves for inter CPU communication, there is no need for extensive long distance wires to run around the chip, especially over the memories.

The one apparent drawback from this approach (in fact any PIM approach that uses DRAM in large amounts) is that off chip contacts must be localized around either the periphery of the chip, or in the logic areas. This means that real PIM chips will probably not be able to sustain the same number of off chip contacts that a full blown microprocessor (or other logic intensive chip) of that time might be able to support. It also means, however, that there will be less power dissipation for off chip contacts. Also, given that we do not need contacts to support complex memory hierarchies, the net reduction in contacts useful for inter node communication may not in the end be affected by all that much. Thus what appears to be a problem with a DRAM-based PIM might actually turn out to be either a wash or a slight PIM advantage (lower power).

5.4.4 Adding Off Chip Interfaces

Tiling the surface of a chip with nodes as in Figure 8 not only gives us incredible internal bandwidth for each CPU, but also directly implements a 2D mesh (as in Figure 9), with an

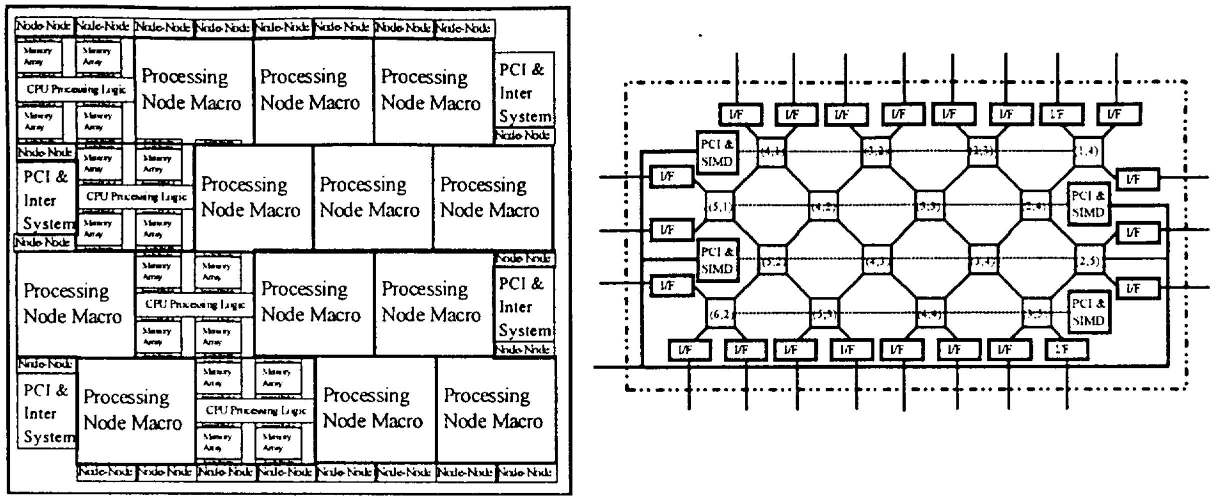

Figure 9. A PIM Chip with its Off Chip Interfaces at the Borders

absolute minimum of interconnection wiring. Further, assuming that the top and bottom rows of the chip are memory stacks (so that each CPU has access to four of them), then assuming that these edge stacks still have dual sense amps as in internal stacks, we find that there are "unused" sense amp connections into the array all around the periphery of the array. This is an ideal place to provide extra logic to provide very high speed I/O between this chip and others (labeled "Node-Node" in Figure 9). This logic has direct access to high bandwidth ports on one side, and the peripheral chip contacts on the other.

There are at least two candidate interfaces that this study considered. First, along the top and bottom we might add a protocol for fast parallel row transfers which permit communication with other chips of the same type. This would be of particular benefit if we were to utilize emerging "chip stacking" technologies where cube of silicon chips are literally "glued" together, with wires running down the sides. Fortuitously, such high density stacking technologies require peripheral chip contacts - exactly what is most appropriate for a DRAM-based PIM as discussed earlier.

The second kind of interface may be a fast "intersystem" protocol, such as Myrinet. This not only provides direct connectivity of the first type, but also permit connection to other arrays of PIM chips (of the same or different organizations) over greater than direct chip-to-chip distances, or to I/O devices. Having multiples of these per chip again provides basic bandwidth firepower that is far beyond that found in modern designs. Again, this can only serve to help simplify the task of configuring real applications.

Finally, also as shown in Figure 9, we also assumed a third type of interface built into each of our PIM chips, namely a simple "memory like" interface such as the PCI bus protocol. This single interface is broadcast to all processing nodes, down the center of the logic rows, and permits a processor outside the chip to directly address the on chip memory as simple memory via loads and stores. Also, this same interface would be used as in EXECUBE as a bus over which global SIMD instructions can be issued. The major difference from EXECUBE is that the address broadcast on this bus would be used to distinguish between direct memory access and SIMD instructions for the processing node. It may also be appropriate to include commands to set up and initiate I/O transfers over the other chip interfaces discussed above. Together, this provides an extraordinary amount of external control and insight into the chip, without having a huge amount of complex routing and wiring to interfere with chip functions.

5.5 System Configurations

Figure 10 illustrates how this design approach might be combined together into an MPP system. Multiple chips might be stacked into 3D cubes and multiple cubes placed on a "memory card." The node-to-node interfaces of the chips would be used for chip - to - chip communication in the stacks. Cube-to-cube, or off-card, transfers would utilize the second style of interface. Further, providing a native memory interface permits insertion of these cards directly into the memory structure of whatever is the "state of the art" microprocessor workstation, making the entire on-chip memory "look like memory" to the microprocessor. Finally, by loading and storing

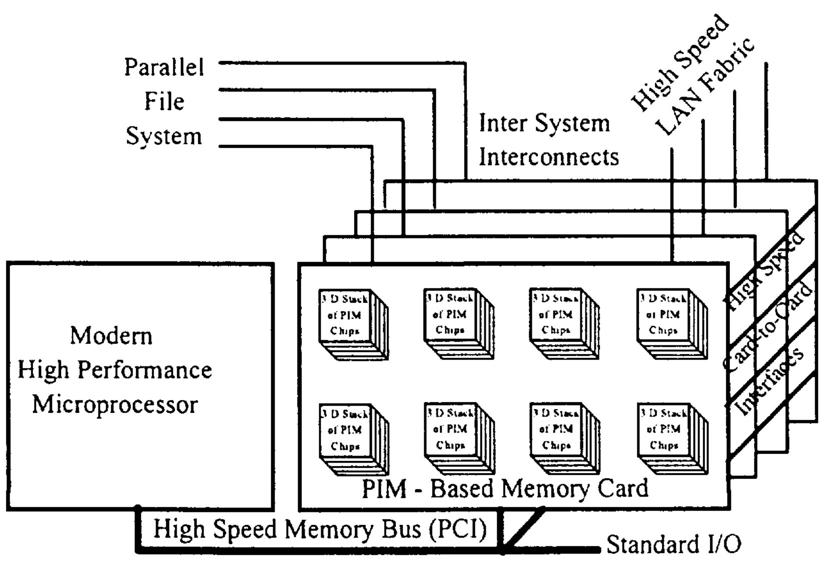

Figure 10. A Potential PIM-based System Architecture

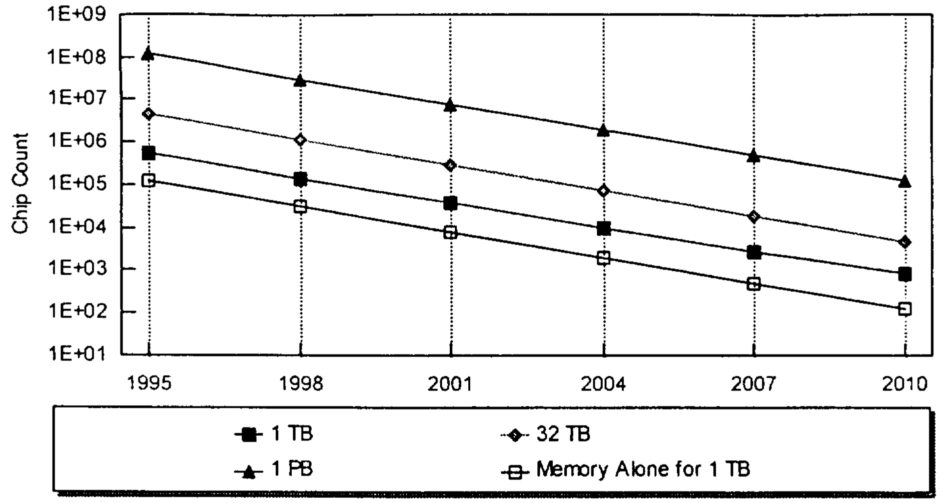

Figure 11. Chip Count for a Petaflop at Different Memory Sizes

to other memory mapped locations this host can initiate SIMD operations in the array as is done with the current EXECUBE chips, which in turn can either start up computational MIMD programs in the array, perform fast global synchronization steps, or initiate transfers between chips over the data interfaces.

5.5.1 Chip Counts for a Petaflop System

A first cut at the total system complexity needed to achieve 1 petaflop peak with configurations such as in Figure 10 can be obtained by utilizing the spreadsheets described earlier. Figure 11 summarizes the results in terms of total chip count for three possible memory sizes: 1 TB, 32 TB, and 1 PB. This data was run assuming a simple 250K transistor CPU as discussed previously. Also as was discussed earlier, a 1 TB system would support perhaps 40% of the applications from Bodega Bay, a 32 TB perhaps 80% (plus match the $N^{3/4}$ 4D simulation problems), and 1 PB representing a fully configured system.

If we assume for the final technology of our petaflop machine the end of the SIA CMOS curve, year 2010, a 1 PB system requires 123,000 chips, the 32 TB system about 4,600 chips, and the 1 TB system around 840 chips. Given the current trends in the state of the art and how this has played out into commercially available silicon, this means that in 2010 the technology would first becomes available in prototype form; but realistically it may be 3 years later before the design and fab processes have been able to design and manufacture enough parts to build a real machine, especially for the larger configuration. This would put real machines in the 2014+ timeframe.

5.5.2 Area and Logic Fraction

How big is this in area? For a 1 PB system, by the year 2001 technology will just reach a sufficient density to reduce the total silicon area down to the equivalent of about one football field. It will be the year 2010 before technology would reduce the smallest of the above systems, the 1 TB memory, to about the area of an office desk.

For both the 32 TB and the 1 PB systems the computation of F(t), the percent of each chip that was logic, yields remarkably low numbers. The 1 PB case really is "all memory," with less than 1% of the area of all chips logic. The 32 TB systems run in the 10 to 15% range. It isn't until we get to "very memory light" systems such as a 1 TB configuration (0.001 MB per MF), that a majority of the chip is logic. As a reference, the lowest curve in Figure 11 is the chip count for DRAM memory chips alone to meet a 1 TB memory. By 2010 (when we have 8 GB memory parts), only 128 parts are needed for 1 TB, while integrating CPUs onto them spreads this out to about an 840 chip system, each about 14 square centimeters in area.

5.5.3 Internal Parallelism

Another key datum that can be gleaned from the spreadsheet is the amount of parallelism present in the system. Assuming a machine that "just peaks" at a petaflop, the number of CPUs is relatively independent of the system architecture or of the memory capacity to performance ratio. Figure 12 gives this degree of parallelism again as a function of time. As can be seen. parallelisms of between 6 million and 2 million CPUs result. Clearly in real life there must be some allowance for the difference between peak and sustainable performance, so if a sustainable petaflop is desired we probably need to consider parallelisms between 3X and 10X these numbers. Given the comments earlier about the relatively low amount of silicon that is actually logic, especially for the memory rich systems, this may have only a minor effect on overall cost.

Clearly, however, programming any machine with two to twenty million separate CPUs is a great stretch in the state of the art and must be considered carefully. As a point in that direction, the same spreadsheet was run with a CPU at the opposite end of the complexity curve from a single issue pipeline, namely a CPU core capable of 4 way issue and multiple floating point function pipes. If such a CPU could be run at 100% efficiency (and this too is pushing the state of the art with modern single CPU compiler technology), we assume that this might give 1 GF flops

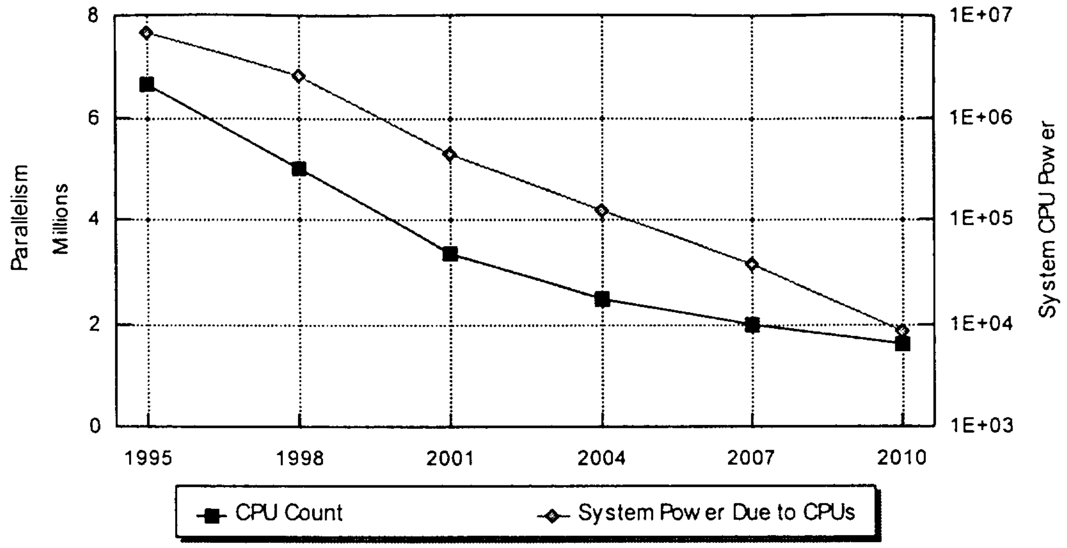

Figure 12. Parallelism and Power Dissipation in a Petaflop System

at 300 MHz, and require about 2 million transistors. While such a configuration does reduce the parallelism to a mere half million CPUs by 2010, paradoxically the more complex logic noticeably increases the chip count, especially at the low memory side where the count more than doubles!

5.5.4 Off Chip Contacts

In terms of off-chip bandwidth, if we assume peripheral contacts only (as was necessary in EXECUBE to avoid disturbing the DRAM), then in 2004, we might assume 600 contacts. Assuming 50 for power and ground, and 100 for the SIMD interface, there are about 450 contacts for the others. Using today's technology, a single Myrinet protocol of 18 pins, would be capable of 160 MB/s. If we had only one such interface for each chip, given the total number of chips, the total I/O bandwidth would greatly exceed the 20 GB/s requirement. This would leave something in excess of 400 pins for chip-to-chip interfaces. Using SIA projections, in 2004, each such pin could run at up to 350 MHz. Assuming a simple parallel transfer protocol, with perhaps 25% of the pins for parity and handshaking, this would provide each chip with up to 13 GB/s of bandwidth to other chips. A square array of chips gives a system inter-chip bisection bandwidth in high fractions of a terabyte per second. The same numbers for 2010 approximately double these rates.

The bottom line is that with PIM technology it appears that off chip bandwidth problems are greatly alleviated.

5.5.5 3D Stacks

Today at least one foundry is building 3D stacks of several dozen memory chips. If the technology continues to improve, and the power dissipation per chip is low enough then in 2010 we could assume stacks of 64 PIM chips for the arrangement of Figure 11. For the smallest memory configurations the means that ***the entire 1 PF PIM system could fit on 14 chip stacks - or one memory card!***

5.5.6 System Power

Finally, using the normal scaling rules of CMOS we can make a crude estimate of power dissipation for such a system. Figure 12 includes a curve which extrapolates this for just the CPU logic. This power is independent of memory density, and drops to about 8 KW with 2010 technology.

Memory power is a bit harder to estimate, but for EXECUBE (1992 technology) a 32 KB memory module on a PIM chip dissipated about 0.1 W. Assuming the same scaling laws as with logic, this yields a power of about 10 KW per TB in 2010. This gives a total on chip power of about 20 KW for a 1 TB system, 330 KW for a 32 TB system, and 10 MW for a 1 PB system.

None of these numbers include power for chip-to-chip transfer.

5.6 Comparisons to Current Practice Extended

Today we have single chip microprocessors with peak performance approaching 1 GF. Assuming that by 2010 we have 16 GF microprocessors, and that to support each in a system one would need 8 L3 cache chips and a bus interface chip, then a 1 PF system would require 60,000 such nodes, for 600,000 chips. Assuming the densest DRAM chips of 0.5 GB each are added to this chip set. For a 1:1 ratio this requires 2 chips per processor, for a total of 720,000 chips! This is 6 times more chips than the PIM approach!

Further, at 2 DRAM chips per processor, there may be extreme problems in obtaining enough bandwidth. Using the run of thumb that each flop requires a data access of a full operand (8 bytes), and that we are issuing 16 instructions per cycle, this means that the CPU core requires a bandwidth of approximately 200 GB/second. Assuming that the cache hierarchy provides 100% of instructions, and 95% of data references, this translates into a bandwidth demand on the two memory parts of about 6 GB/second. This approaches a factor of 10 more than what can be done today.

5.7 Simplifying the Operating System Support

Making the PIM chips "look like memory" to a conventional microprocessor as pictured in Figure 11 has some significant system software simplifying ramifications. Individual data items can be accessed via conventional loads and stores without regard to where in the processor array the data is. Since "memory is memory," the partitioning of data structures to maximize parallelization can be done within the confines of conventional compiler technology by careful memory allocation. No special languages or language extensions are needed to do the mapping, although it may be appropriate to consider preprocessors that, given the desired partitioning arrangement, will develop the data declaration source code to reflect that partitioning. Together with initialization, overall system monitoring, and user interface, this permits the most complex, but usually least computationally stressing, parts of an application to be written as conventional scalar code running in the host microprocessor, utilizing whatever is the current state of the art in tools.

It also means that we need not keep a full blown copy of an operating system in each CPU in the PIM. The host can handle all the complex problems associated with users, file systems, network interfaces, etc., and leave the "runtime" that is resident in each PIM CPU the far simpler job of managing just computations.

While simple, this approach still requires some additional features to be grafted onto the underlying host operating system, with care as "device drivers" of various flavors, including:

1. ability to "page fix" blocks of real memory to virtual memory, so that applications that wish to map data structures to different chips to support parallel computation internal to those chips can do so.
2. support for "broadcasting" sequences of SIMD instructions to different groups of processing nodes within the PIM card set, and for testing for global conditions, such as all processing nodes in group x have reached a common barrier point.

3.support for a parallel file system where the disks are attached to the interfaces which exit both the chips and the PIM cards.

4.support for a closely coupled network of compute nodes as pictured in Figure 11, where the interfaces exiting the PIM cards would form a very high speed LAN interconnect fabric.

Note that the use of these chip interfaces for both parallel file systems and closely coupled network interfaces eliminates a bottleneck that severely constrains current technology. The multiple parallel data ports are interfaced **directly with different memories**, and need not either contend either for some internal memory bus or involve multiple memory-to-memory copies to position the data correctly. This should provide both a substantial reduction in real application complexity, and permit achievement of much higher sustainable levels of performance over an entire application execution.

5.8 Other Key Results

In addition to investigating PIM architectures and there VLSI implementations, this study also looked at a wide variety of related system, benchmarking, and software issues. Following is a brief list of their results:

1.A study of the 5 kernels selected by Geoffrey Fox at the June 1996 Bodega Bay Workshop indicates that they can all benefit from 4 simultaneous memory accesses from the 4 memory macros per CPU, that in most cases the accesses from two separate CPUs to the same memory stack can be partitioned into different slices (permitting concurrency without a performance hit), that some kind of a "test and set" is needed in the PIM CPU ISA to manage synchronism with a neighbor, and that there needs to be two ways of specifying memory addresses: totally local to an individual CPU, and globally.

2.Parallel I/O will be a problem in terms of routing and (de)multiplexing data on the fly, but that internally a message passing paradigm such as MPI should to straightforward to implement.

3.There is a need for parallel global synchronization (not just between two neighbors), and that this needs to be relatively efficient, with multiple simultaneous rendezvous in action at one time.

4.Given the latency and bandwidth inside a CPU, it does not appear that one needs to design an overly complicated run time for each CPU. In particular, a multi threaded kernel may not be as important for PIMs than for conventional machines.

5.A variety of parallel programming language models were studied: data parallel, task parallel, and object parallel. The best of these were the data parallel ones such as HPF and C*. Functional languages such as NESL and Concurrent ML (NESL in particular) had some advantages in their ability to deal with prefix operators which yield "vector-like" operations, but that it looked difficult to unfold recursions in these languages enough to provide the huge amounts of parallelism we have observed in our designs. Other languages such a Lucid (an intensional language) provided multidimensional syntax, but seemed to result in too high a granularity to match PIM's capabilities.

6.A database join function was implemented in parallel on the campus' SP-2 and experiments were run with a variety of topologies of the supposed inter-node interconnects. This included the 3D topology of EXECUBE. Because of limitations in the SP-2, parallelisms of at most 8 real CPUs could be simulated. Results indicate that to get any real performance a very large number of records are probably needed, with well in excess of a 1,000 records per CPU. Even then, only a speedup of half of peak was observed, with little or no variation due to topology.

6. Importance of the Findings and Future Directions

The key result from this work has been that PIM not only represents a viable path to petaflops. but is a key new trend in computer architecture that should be exploited across the board. The importance of the work is shown in the reception it received when used as the technical basis for proposals for several follow-on studies, all of which have been funded, started at about the time this one completed, and will advance the work discussed here in several directions. These projects include:

1.Participation in an NSF-sponsored "Point Design" study whose goal is to look at using PIM technology to perform initial studies to advance to an intermediate level of performance, namely 100 TF, within a shorter time frame that the 20 year outlook for the petaflop study.

2.Design study No. 960587 entitled "Scaleable Spaceborne Computing using PIM Technology" with the Jet Propulsion Laboratory (as a subcontract under their NASA prime contract Task

Order RF-158) to investigate how the increased computing density of PIMs, especially as configured as a memory accelerator as developed under this study, can be applied to a variety of very high performance computing problems for deep space applications.

3.A study entitled "PIM Based Accelerator Technology Infrastructure Development" funded by DARPA through the SPPDG Group at the Mayo Clinic into verifying the ability of commercial VLSI fabricators to build PIM technology chips, the CAD tools needed to built them, and sample applications of interest to DOD. Again the baseline architecture used in these investigations is centered on the strawman PIM chip described above, with DOD applications assuming the "smart memory" based accelerator format.

There are multiple other direct follow-on activities that will be considered in the near future that owe their origins to the work done under this study. These include:

1. working with several research groups around the US to investigate PIM technology as applied to other problems. One example is a group at Cal Tech investigating achieving a petaflop through the use of very high speed superconducting CPU technology, where there is a strong need for a "smart memory" to do operations where the time delays needed to get data to or from the superconducting CPU exceeds greatly the time spend by that CPU to do the work.

2.developing a prototype card using some newly announced PIM-like parts that will permit early exploration of the memory mapped organization for PIMs, and demonstration of prototype software to efficiently drive such an architecture from a conventional workstation.

3.Continued exploration of alternative CPU instruction sets that are both low power and that can thrive on the latency and bandwidth environment presented by a PIM chip, but without the huge complexity found in modern microprocessors.

Finally, at Notre Dame this study has laid the ground work for a rather large number of additional graduate student projects which will push at different aspects of some of the problems we encountered during the study. They should mature into thesis within a year or so.

In summary, this study has achieved some significant results both in achieving a petaflop and in suggesting new directions in computer architecture. It has demonstrated the basic numerology behind PIM technology, and shown that there are some huge architectural gains to be made by reconsidering the way we design systems. Its results have strongly influenced further work both here at Notre Dame and at many other research establishments.

7. Relevant Publications:

1. Kogge, P. M., "Pursing a Petaflop: Architectural Requirements," in preparation.
2. Kogge, P. M., J. B. Brockman, "The PIM Design Spectrum," in preparation.
3. Kogge, P. M., J. B. Brockman, "Pursuing a Petaflop: the Case for PIM," in preparation.
4. Kogge, P. M., et al, "Pursuing a Petaflop: Point designs for 100 TF Computers Using PIM Technologies," to be published in 1996 Frontiers of Massively Parallel Computation Conf., Annapolis, MD, Oct., 1996.
5. Kogge, P. M., T. Sunaga, et al, "A Parallel Processing Chip with Embedded DRAM Macros," to be published in IEEE J. Solid State Circuits, October, 1996.
6. Kogge, P. M., "Computing Component Characterization," Section 8.1, Draft Proceedings, Petaflops Architecture Workshop, April, 1996
7. Kogge, P. M., T. Giambra, et al, "RTAIS: An Embedded Parallel Processor for Real-Time Decision Aiding," 1995 National Aerospace Conference (NAECON), Dayton, OH, May 1995.
8. Kogge, P. M., T. Sunaga, et al, "Combined DRAM & Logic Chip for Massively Parallel Embedded Applications," 1995 Conf. on Advanced Research in VLSI, Raleigh. NC, March 1995.
9. Kogge, P. M., "Processor-In-Memory Chip Architectures for PetaFlops Computing," PetaFlops Frontier Workshop, Feb., 1995
10. Kogge, P. M., Contributor to *Enabling Technologies for PetaFlops Computing*. Cal Tech, 1995.
11. Kogge, P. M., "The EXECUBE Approach to Massively Parallel Processing." Int. Conf. on Parallel Processing, August, 1994.
12. Kogge, P. M., J. Oldfield, et al, "VLSI and Rules Based Systems," *VLSI for AI*, Kluwer Academic Press, 1990.
13. Semiconductor Association of America, *The National Technology Roadmap for Semiconductors,* San Jose, CA

Appendix A

Preprint of

Pursuing a Petaflop: Point Designs for 100 TF Computers Using PIM Technologies

to be published in 1996 Frontiers of Massively Parallel Processing

Annapolis, MD Oct. 1996

Preprint: To be published in Frontiers of Massively Parallel Computation, Oct. 1996

Pursuing a Petaflop: Point Designs for 100 TF Computers Using PIM Technologies

Peter M. Kogge, Steven C. Bass, Jay B. Brockman, Danny Z. Chen, Edwin Sha
Department of Computer Science & Engineering
University of Notre Dame, Notre Dame, IN 46556

Abstract

This paper is a summary of a proposal submitted to the NSF 100 Tera Flops Point Design Study. Its main thesis is that the use of Processing-In-Memory (PIM) technology can provide an extremely dense and highly efficient base on which such computing systems can be constructed. The paper describes a strawman organization of one potential PIM chip, along with how multiple such chips might be organized into a real system, what the software supporting such a system might look like, and several applications which we will be attempting to place onto such a system.

1 Introduction

Massive use of computer based simulation is rapidly becoming a linchpin of science and engineering, largely because of the explosive growth in the performance of modern microprocessors. In terms of peak performance levels, the newest *massively parallel processors (MPPs)*, with *tera(fl)ops* of performance, are now advertised as being capable of attacking even the "Grand Challenges" of the 1980s. There are, however, two major problems with the current state of the art: first, the cost of the very highest end machines lies far beyond the point where widespread deployment is feasible, and second the software environments for these machines are awkward at best, capable of eking out only fractions of the performance of what the hardware is capable of supplying. Together, this signals very severe problems if we are to consider climbing towards the next major level of performance - *peta(fl)ops*.

The thesis of the proposal summarized in this paper is that these problems stem from the "traditional" von Neumann bottleneck, and how we have approached it. For technology and cost reasons we have historically separated memory parts and CPU logic parts. With the advent of CMOS microprocessors with very high performance pipelined and superscalar architectures, individual CPU core performance levels have gone through rapid acceleration, requiring ever increasing amounts of bandwidth from the memory subsystem. These rates have far exceeded the bandwidth capabilities of our densest DRAM memory parts, and the gap will widen over time. The net result is architectural complexity: memory hierarchies are introduced to provide the bandwidth, which in turn drives cost and the software complexity needed to address these hierarchies efficiently, especially in an MPP environment.

A new technology is emerging to counter this fundamental defect - the combination on one chip of both dense DRAM memory and significant amounts of logic. This capability permits new architectures termed *Processing-In-Memory (PIM)* to place computing either right next to, or even inside of, the memory macros, where there are huge amounts of raw memory bandwidth. Projections out to the year 2014 laid the basis for one of three potential peta(fl)ops architectures proposed at the 1994 Pasadena Workshop on Petaflops [15], and was further amplified at followon workshops at the 1995 Frontiers Conference [9], and then at Bodega Bay August 1995 [2].

The objective of the proposal summarized in this paper is thus to utilize PIM technology potentials over the next decade to configure some "point design" MPPs with 100 tera(fl)op (TF) potential for several interesting classes of problems, and chip architectures that should permit greatly simplified programming environments. The applications to be studied include: solutions of nonlinear PDEs, multidisciplinary design problems, and problems in massive image compression. These were chosen both because they exhibit a representative range of granularity, data structures, and need for internal communications, and we have a variety of new computational and parallel program construction techniques for them under current study.

This paper is organized as follows: Section 2 summarizes characteristics of potential 100 TF problems. Section 3 discusses current PIM technology. Section 4 then presents a "strawman" point design. Section 5 discusses different applications, with descriptions in Section 6 on software development. Section 7 then describes our study approach .

2 Hardware Constraints

As the community has learned, a commercially viable MPP is more than just hardware whose functional units can be run at huge clock rates. At the hardware level, true sustainable performance can only be achieved if there is:

1. sufficient bandwidth from the memory system, at low enough latencies, to support the function units,

2. sufficient memory to hold enough data long enough to process it to completion,
3. sufficient bandwidth between different compute nodes of an MPP to avoid delays while waiting for data,
4. sufficient I/O bandwidth for fast initial data load, intermediate results and checkpointing data, and final results, to avoid bottlenecking the main computation.

While all of these are important, perhaps the one with most effect on system replication costs is memory. For example, for "classical" scientific computing a rule of thumb is that a MB of memory is needed for each MF of performance. For a petaflop this translates into a petabyte of memory - which is a huge amount even 20 years from now. The 1994 Pasadena Conference revisited this rule for petaflop level systems, and decided that there was a reasonable set of 4D simulation problems where storage might grow as performance to the 3/4 power (after a GF). The result was that for such problems a ratio of 0.03 MB per MF (30 TB) might be acceptable. The Bodega Bay Workshop carried this analysis one step further and estimated primary memory needs for about 36 different applications with an average memory requirement about 3 TB.

This proposal addresses not a petaflop machine but one of a 100 TF. For the 4D simulation case, this translates to about 5-10 TB. If one assumes that such a machine would not attack problems which ran over a day in length (i.e. around 3 hours = 10^4 seconds on a petaflop machine), then all the Bodega Bay applications that at a petaflop would run in less than 3 hours might take only a terabyte. Further, if one were interested in an "entry level" machine, then 100 GB would permit attacking at least a few applications. We will thus use the three numbers of 10 TB, 1 TB, and 100 GB as initial memory design points.

In terms of secondary storage, for the same subset of applications, there are only two that require over 0.1 PB, with the rest on the order of 100 GB. Since one would not expect this to change with a downsize to 100 TF, we will assume 1 TB of secondary memory - enough for data sets for 10 applications.

For the same application suite, I/O requirements are between 1 and 100 GB per second. We assume that these are for continuous I/O during program execution, so that if the machine is only a 100 TF one, then the I/O would be one tenth - i.e. a max of 10 GB/s. Initial program load, however, has to be estimated separately. Assuming that all of primary memory has to be loaded at startup, and assuming that this is to be done in an overlapped fashion with the prior application, derivation from the Bodega Bay data indicates that an additional sustained rate of 10 GB/s is needed. Thus, a total I/O of 20 GB/s is baselined.

3 PIM State of the Art

Many chips today merge logic with some form of memory. At one extreme most modern microprocessors combine millions of transistors of logic with a few tens of KB of SRAM (for caches). At the other extreme, most conventional DRAMs combine MBs of memory with a few thousands of transistors for address decoding for the internal arrays, latching at the sense amps, and multiplexing to drive the data lines.

PIM chips fall in the middle. They combine large amounts of **both** memory **and** logic. However, their key distinguishing feature is that they represent potentially **self-contained** designs where all the processing functions and all the memory for that processing for one or more nodes are on the same chip. This self-contained characteristic has several key consequences:

1. a single part type scaleable MPP is possible where additional computational resources are added by adding more chips (much as today we add more memory via plug in SIMMs).
2. This permits novel 3D packaging techniques which reduce both overall system costs and chip to chip paths, reducing the other killer of MPP performance - latency.
3. Placing the processing logic next to the memory permits a huge increase in the percent of raw memory bandwidth that can be utilized from the memory arrays over today, where at best a few percent of the total bandwidth is presented to the off-chip pins. This can reduce or eliminate the need for complex caching and other tricks in the design of the processing logic - again reducing both cost and latency.
4. Eliminating from the processing chips the pins to support a memory hierarchy means that these same pins can be used to perform something computationally useful - namely communication with other processing nodes.
5. Moving multiple nodes to a single chip also allows new architectural techniques such as mixed SIMD and MIMD processing, very high bandwidth memory-to-memory transfers, and "in the memory" processing, all of which significantly reduce application program complexity.

3.1 EXECUBE - The First True PIM

EXECUBE [8,9,10] was the first true PIM to be architected as a single part type MPP building block with all the above features. It is based on a 4 Mb DRAM, with a large center circuit block for custom logic. The chip has 16 separate DRAM memory macros, 8 on top and 8 on the bottom. As pictured in Fig. 1, each chip utilizes the logic to implement 8 complete Processing Elements (PEs), each with its own 64 KB memory, 16 bit CPU, and inter PE DMA link support logic. The PEs are arranged on-chip in a 3D binary hypercube, with each PE having a separate full duplex link off-chip. These links can tie directly to links on other chips in almost any topology.

To simplify parallel application programming, the PEs were also designed to run in either SIMD or MIMD mode. A host controller can place on the *SIMD Broadcast Bus* any instruction from the PE ISA to be broadcast to some or all of the PE nodes in the EXECUBE array for execution at the same time. Instructions in the PE ISA include ones to switch from SIMD to independent MIMD mode, permitting a single SIMD instruction to initiate simultaneous MIMD program execution in the array. Other instructions in the PE ISA signal that the PE is to switch back to SIMD mode, and await new instructions from

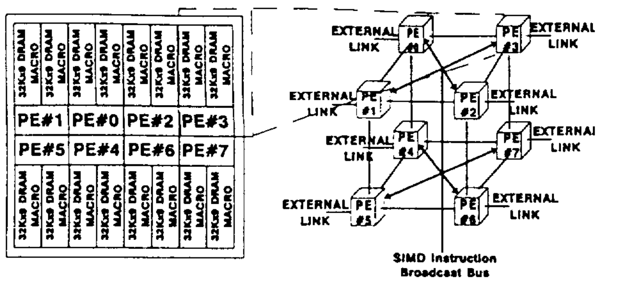

Fig. 1. The EXECUBE PIM chip

the SIMD Broadcast Bus. A simple interlock mechanism on this bus stalls transfer of additional SIMD instructions until all PEs that are designated to respond to a SIMD instruction have actually reached a SIMD mode. This provides an extremely fast and inexpensive global synchronization mechanism.

3.2 Other PIMs

Since EXECUBE, several other chips PIM-like have emerged. Table 1 lists these chips, their peak performance per chip, the number of on chip processors, the overall chip storage capacity, and the ratio of storage to performance. This latter ratio is particularly key because the closer this is to the values needed for real computation, the more viable the overall designs are. What is key from this table is that even today, the only design that come even close to the desired 100 TF ratios are EXECUBE, SHARC, and M32R/D, and that is because they both chose a very dense memory technology.

Note that this table includes neither current microprocessors with on chip caches, or single CPU chips which were designed for MPPs but with limited on chip memory.

Table 1. Current MPP PIM Chips

Chip	Year	# CPUs	Perf.	MB	MB/Perf
EXECUBE	1993	8 s	50 Mips	0.5	0.01
Terasys	1993	256 1b	625 Mbops	0.02	2.6E-5
SHARC	1994	1 32b	120 MF	0.5	0.005
TI MVP	1994	4+1	2K Mops	0.05	2.5E-5
MIT MAP	1996	4	800 MF	0.13	1.6E-3
DAAM	1996	1024 1B	862 Mops	0.5	6E-4
M32R/D	1996	1	51 Mips	2.0	0.02

4 A PIM-Based Generic Point Design

For this proposal we have taken the key ideas from the current crop of PIM chips, especially EXECUBE, married it with a variety of new architectural ideas, and extrapolated the technology ahead 10 years. The resulting "strawman" PIM "point design" system is described here. It will be personalized to specific applications during the study. This design is discussed in several pieces: first, the chip, then configuring the chip into systems, and then system software.

4.1 Strawman PIM Chip Architecture

The PIM chip assumed here has three separate characteristics worthy of discussion: the processing node macro (memory + logic), the "tiling" of a chip with these macros, and the interfaces provided on the chip for communication with other chips. Fig. 2 diagrams an assumed processing node macro. It consists of a CPU surrounded by 4 separate memory macros, with each memory macro having its own decoding, sense amps, and row logic. The choice of 4 macros was deliberate, and reflects the need to have sufficient memory per processing node to be able to sustain the computational throughput.

While the actual choice of CPU architecture is somewhat free, the EXECUBE experience and subsequent studies have indicated that the most efficient choices in terms of units of silicon per unit of performance are today, and will continue to remain, the simpler designs. Today's multi-million transistor superscalar, super pipelined, designs use 5 to 30 times more silicon per unit of performance than a very basic design with 125K transistors for a primarily fixed point engine and 250K transistors for floating point intensive one. With such simpler CPU designs, 4 separate memory macros provide not only the density but also more than sufficient bandwidth, particularly if the latches formed at the memory sense amps are visible to the processor in some way. In practice the nature of these latches could range from simply a fast single line cache, to actually placing programmer visible registers and/or ALUs at the sense amps. In any case, if future DRAM macros merely maintain today's EXECUBE 12 ns page access time, the latency for a memory access as seen by the node's processing logic is measured in a handful of cycles - more than enough to support simple CPUs at near 100% utilization.

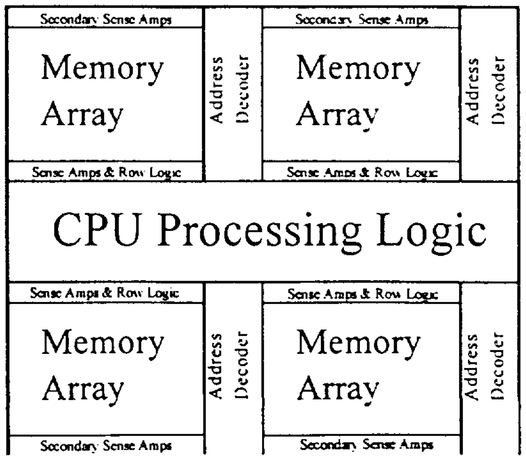

Fig. 2. Strawman PIM Processing Node Macro

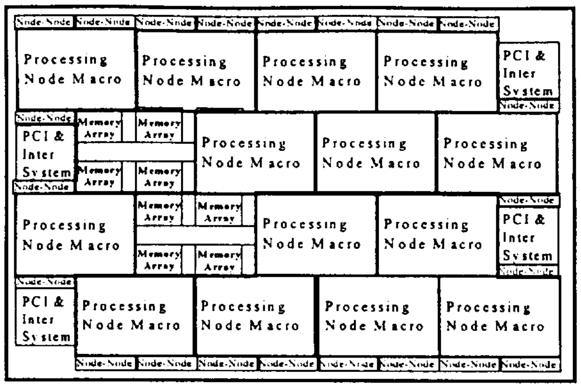

Fig. 3. Generic PIM Chip Floorplan

Finally, one additional feature is a duplicate sets of sense amps at either end of the bit lines (as shown in Fig. 2). This enables the transfer of data on a row-by-row basis between adjacent processors, providing a monumental advantage in bandwidth over today's MPP designs.

Tiling the surface of a chip with nodes as in Fig. 3 not only gives us this incredible bandwidth, but also directly implements a 2D mesh (as in Fig. 4), with an absolute minimum of interconnection wiring. Further, having access to the secondary sense amps on the outside of the processing array permits logic (labeled "Node-Node" in Fig. 2) to be added to provide fast communication off-chip. There are at least two candidate interfaces to consider here. First, along the top and bottom we might add a protocol for fast parallel row transfers which permit communication with other chips of the same type. This would be of particular benefit if we were to utilize emerging "chip stacking" technologies where cube of silicon chips are literally "glued" together, with wires running down the sides.

The second kind of interface may be a fast "intersystem" protocol, such as Myrinet [12]. This not only provides direct connectivity of the first type, but also permit connection to other arrays of PIM chips (of the same or different organizations) over greater than direct chip-to-chip distances, or to I/O devices. Having multiples of these per chip again provides basic bandwidth firepower that is far beyond that found in modern designs. Again, this can only serve to help simplify the task of configuring real applications.

Finally, also as shown in Fig. 3 and 4 we assume a third type of interface built into each of our PIM chips, namely a simple "memory like" interface such as the PCI bus protocol. This single interface is broadcast to all processing nodes, and permits a processor outside the chip to directly address the on chip memory as simple memory via loads and stores. Also, this same interface would be used as in EXECUBE as a bus over which global SIMD instructions can be issued. The major difference from EXECUBE is that the address broadcast on this bus would be used to distinguish between direct memory access and SIMD instructions for the processing node. It may also be appropriate to include commands to set up and initiate I/O transfers over the other chip interfaces discussed above.

4.2 System Configurations

Fig. 5 illustrates how multiple chips would be combined together into an MPP system. Multiple chips might be stacked into 3D cubes and multiple cubes placed on a "memory card." The node-to-node interfaces of the chips would be used for chip - to - chip communication in the stacks. Cube-to-cube, or off-card, transfers would utilize the second style of interface. Further, providing a native memory interface permits insertion of these cards directly into the memory structure of the then current "state of the art" microprocessor, making the entire on-chip memory "look like memory" to the microprocessor. Finally, by loading and storing to other memory mapped locations this host can initiate SIMD operations in the array, which in turn can either start up computational MIMD programs in the array, perform fast global synchronization steps, or initiate transfers between chips over the data interfaces.

4.3 Scaling to 100 Teraflops

A first cut at the total system complexity needed to achieve 100 TF with configurations such as in Fig. 5 can be obtained by interpolating from the 1994 SIA CMOS technology roadmap

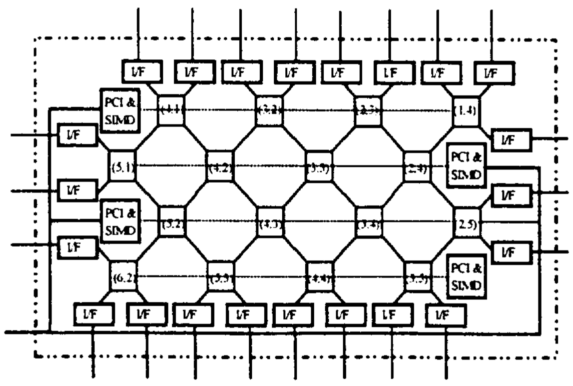

Fig. 4. Single Chip = Mesh MPP

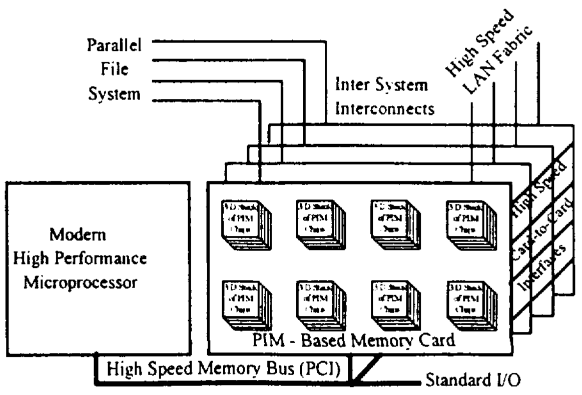
Parallel File System
Inter System Interconnects
High Speed LAN Fabric
Modern High Performance Microprocessor
3D Stack of PIM Chips
PIM - Based Memory Card
High Speed Card-to-Card Interfaces
High Speed Memory Bus (PCI)
Standard I/O

Fig. 5. Assumed System Organization

[1]. Fig. 6 summarizes the results in terms of total silicon area for each of the assumed memory sizes. If 2004 is a technology point for a 2006 machine, then the baseline PIM designs would have the characteristics of Table 2. Besides the basic memory and logic projections, these curves also assumed running the on chip processing logic at the SIA's " High Performance, On Chip Clock" rates. Other more conservative design points might result in 2-3X more chips.

The earlier comment about the dominance of memory on size is shown in the "Total Area" plot of Fig 6, where the lowest curve indicates the amount of silicon needed for the processing logic alone. It isn't until total memory size drops to 100 GB that the logic area dominates.

In terms of off-chip bandwidth, if we assume peripheral contacts only (as was necessary in EXECUBE to avoid disturbing the DRAM), then in 2004, we might assume 600 contacts. Assuming 50 for power and ground, and 100 for the SIMD interface, there are about 450 contacts for the others. Using today's technology, a single Myrinet protocol of 18 pins, would be capable of 160 MB/s. If we had only one such interface for each chip, given the total number of chips, the total I/O bandwidth would greatly exceed the 20 GB/s requirement. This would leave something in excess of 400 pins for chip-to-chip interfaces. Using SIA projections, in 2004, each such pin could run at up to 350 MHz. Assuming a simple parallel transfer protocol, with perhaps 25% of the pins for parity and handshaking, this would provide each chip with up to 13 GB/s of bandwidth to other chips. A square array of chips gives a system inter-chip bisection bandwidth in high fractions of a terabyte per second.

Today at least one foundry is building 3D stacks of several dozen memory chips. If the technology continues to improve, and the power dissipation per chip is low enough then in 2004 we could assume stacks of 64 PIM chips for the arrangement of Fig. 5. For the smaller memory configurations the means that *the entire 100 TF PIM system could fit on 1 to 2 cards!*

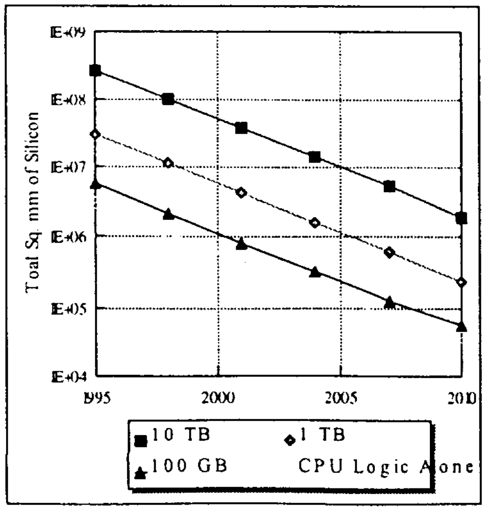

Fig. 6 SIA Projections For a 100 TF

4.4 Comparisons to Current Practice

Today we have single chip microprocessors with peak performance approaching 1 GF. Assuming that by 2004 we have 10 GF microprocessors, and that to support each in a system one would need 8 L3 cache chips and a bus interface chip, then a 100 TF system would require 10,000 such nodes, for 100,000 chips. Assuming the densest DRAM chips of 0.5 GB each, we would need 120,000 total chips for the largest 10 TB configuration, 102,000 for 1 TB, and 100,200 for 100 GB configurations. The PIM approach is superior by up to almost two orders of magnitude!

4.5 Simplifying the Operating System Support

As pictured in Fig. 5, making the PIM chips "look like memory" to a conventional microprocessor has some significant system software simplifying ramifications. Individual data items can be accessed via conventional loads and stores without regard to where in the processor array the data is. Since "memory is memory," the partitioning of data structures to maximize parallelization can be done within the confines of conventional compiler by careful memory allocation. No special languages or language extensions are needed to do the mapping, although it may be appropriate to consider preprocessors that, given the desired partitioning arrangement, will develop the data declaration source code to reflect that partitioning. Together with initialization, overall system monitoring, and user interface, this permits the most complex, but usually least computationally stressing, parts of an application to be written as conventional scalar code running in the host microprocessor, utilizing whatever is the current state of the art in tools.

While simple, this approach still requires some additional features to be grafted onto the underlying operating system, with care as "device drivers" of various flavors, including:

1. ability to "page fix" blocks of real memory to virtual memory, so that applications that wish to map data structures to different chips to support parallel computation internal to those chips can do so.

Table 2. 2004 PIM Design Characteristics

	MB/MF Ratios		
	0.1:=10 TB	0.01 = 1TB	0.001=100GB
Total Chips	33,119	3,686	742
# 3D Stacks	517	58	12
Bisection BW	796	266	119
#Nodes/Chip	5	48	241
MB/Chip	302	271	135
MF/Chip	3,019	27,132	134,713

2. support for "broadcasting" sequences of SIMD instructions to different groups of processing nodes within the PIM card set, and for testing for global conditions, such as all processing nodes in group x have reached a common barrier point.
3. support for a parallel file system where the disks are attached to the interfaces which exit both the chips and the PIM cards.
4. support for a closely coupled network of compute nodes as pictured in Fig. 5, where the interfaces exiting the PIM cards would form a very high speed LAN interconnect fabric.

Note that the use of these chip interfaces for both parallel file systems and closely coupled network interfaces eliminates a bottleneck that severely constrains current technology. The multiple parallel data ports are interfaced **directly with different memories**, and need not either contend either for some internal memory bus or involve multiple memory-to-memory copies to position the data correctly. This should provide both a substantial reduction in real application complexity, and permit achievement of much higher sustainable levels of performance over an entire application execution.

4.6 Minimal Runtime

If the processing logic inside each processing node in the PIM chips is general purpose, then it is probably necessary to provide some sort of runtime system. Unlike current MPPs , this runtime need not approach the complexity of a complete "operating system kernel." There are several reasons for this:

1. use of a host microprocessor with access to the PIM chips means that most system management and user functions can be done in the host, with conventional OS support.
2. the individual units of computation are smaller than the host microprocessor designs, and thus it is nowhere near as critical to keep each and every one busy 100% of the time. This reduces the need for local sophisticated task schedulers.
3. the assumed ability to run in SIMD and MIMD mode off loads a lot of the application code global synchronization and set up to the host microprocessor, where it can be done largely in parallel with on going computations in the array.
4. the huge amount of low latency bandwidth available between nodes on the PIM chips means that very sophisticated messaging protocols may not be needed, and that it ought be possible to graft on simple but high performance "virtual shared memory" protocols onto a systems which is inherently a mix between a physically locally shared and globally distributed memory design.

At this time, it appears that the key features that must be designed into a runtime for the generic PIM chips of Fig. 4 include: a basic library of inter node message passing, support for a virtual shared memory system, a "Remote Procedure Call" like mechanism that permits SIMD code to set up and initiate a MIMD program on some selected set of node, and fault tolerance and rerouting in case a chip goes down.

5 Applications

Three applications were chosen for study as targets for variations of the above point design: solutions of nonlinear partial differential equations (PDEs), multidisciplinary design problems, and some problems in massive image compression. These problems were chosen because: (1) they exhibit a representative range of variation in granularity, data structures, and need for internal communications, and (2) we have a variety of new computational and parallel program construction techniques for them under current study.

5.1 Wave Digital Solution of Nonlinear PDEs

Simulating systems of PDEs is a classical problem which has driven the development of many prior generations of supercomputers. Looking to the future, besides obvious extensions such as more and varying grid points, there will be a real need to attack more complex PDEs, such as ones with non linearities. Given this, one of our chosen applications is the solution utilizing a new finite difference technique for transcribing such systems into discrete space-time representations. This "wave digital" technique was originated by Alfred Fettweis (a recent Visiting Professor at Notre Dame) from earlier work in signal processing, where it was known as "wave digital filtering."

Briefly, the steps involved in transcribing a continuous system of PDE's into a discrete algorithm begin with a special transformation of the coordinate system within which the equations were originally elaborated. This results in the time-domain attributes of causality and passivity being automatically acquired by all the dimensions in the new system, not just time [5]. Next, a multidimensional "reference circuit" is constructed for these equations. Elements in such circuits are typically resistors, transformers, independent sources, as well as so-called *multidimensional* inductors and capacitor [6]. This reference circuit is designed such that if one were to write a certain complete set of equations of motion for the circuit (as Kirchhoff loop and/or node equations). the set of transformed partial differential equations would result. Finally, this reference circuit may be transcribed into a wave digital flowgraph representing the simulation calculations required to be performed at each discrete-space grid point [7].

There are four major properties of wave digital formulations which make them especially good candidates for MPPs such as described here:

1. If the original system is multidimensionally stable (i.e.. passive from an energy conservation view, and most real systems are), then the wave digital formulations are also. This leads to stable finite difference systems even for difficult non linear PDEs operating on the edge of instability.
2. As a consequence of both the stability and the simulation of wave quantities rather than the original problem variables,

there is a potential for reduced dependent variable word lengths and dynamic ranges. This leads to reduced memory requirements and simplified arithmetic units, both of which would minimize total silicon utilization.

3. Communication between computations is strictly "nearest neighbor." In fact, wave digital is the only known second order difference method we know of that has this property. This fits neatly in with the natural structure that grows out of tiling chips with PIM macros as pictured in Fig. 3.
4. Finally, the approach accommodates naturally parameters that vary over space and time, increasing further the range of applicability.

As a simple example, Fig. 7 diagrams the grid structure and inter grid point dependencies for a sample wave digital formulation of the electrical behavior of a pair of parallel plates. While this problem is not exotic by current standards, the resulting solution pictured here is still very characteristic of more complex non linear PDEs. Each grid point involves internal storage of 3 changing wave parameters and 18 grid point-dependent terms, transfers of 6 parameters to and from three neighbors, 29 add/subtracts, 5 constant multiplies, and 10 multiplies by grid point & time-dependent parameters. If each parameter were 32 bits (because of the inherent stability properties of the wave digital approach), this translates into a grid point which needs 84 bytes, and in the time required to do 44 arithmetic operation an inter grid point I/O bandwidth of 24 bytes. If one postulated a problem with 10^4 x 10^4 grid points, 10^6 time steps, where parameter values were changed every 1000 steps from in core tables and wave values sampled every 1000 steps, a 100 TF per second MPP with perhaps 10^5 processing nodes, would need about a minute of execution time, with a primary storage requirement of about 10 TB (for a ratio of 0.1 MB per MF), about 50 MB/sec bandwidth between neighboring processing nodes, and about 20 GB/s of system I/O to archive the wave quantities. This is an ideal match to a PIM based MPP such as described earlier.

In terms of software development for such problems, the process of developing the actual code for each grid point's computation does not appear particularly complex. What is difficult, especially when grids may change dynamically during a computation, or when we wish to group multiple sets of grid points into single computational nodes, is to partition regions of grid points together, and then schedule the computations in an optimal fashion. Much of the ongoing work at Notre Dame discussed below for transformation, partitioning, and scheduling has proven directly applicable.

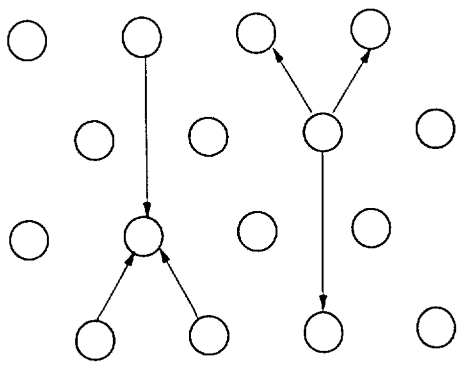
Fig. 7 Wave Digital Solution of a Typical PDE

5.2 Multi Disciplinary Design Problems

Already one of the heaviest consumers of computing cycles, engineering design will motivate the need for increased growth to teraflop levels and beyond over the next decade. A first trend, apparent in many industries, is the drive to reduce the duration of design cycles. This becomes all the more difficult in light of the second trend, which is the need to perform more detailed computer-based analyses during each cycle, allowing engineers to evaluate the quality, reliability, and manufacturability of designs before release to production. This in turn is further complicated by a third trend, which is an increasing focus on simulating and optimizing not only individual components, but also complex, coupled, multidisciplinary systems.

Although the demand for high-performance computing in engineering design is clear, the mechanisms for best providing and utilizing the necessary computing capacity are not. While significant work has been invested in applying MPPs to single-discipline analyses, less is known about its application to multidisciplinary analysis and optimization (MDO). In the research proposed here, we plan to investigate methodologies for massively-parallel MDO on a PIM-based architecture. Specifically, we plan to perform an assessment of the computing requirements of current MDO algorithms on a given large-scale design problem, to identify the bottlenecks and limitations to applying MPPs to MDO, and to develop and size a trial problem. Because of the impact that MDO technology can have on the development of PIMs themselves, we will focus our examples on CAD for VLSI.

One of the major results of the Notre Dame MDO effort has been the development, implementation, and testing of the Concurrent Subspace Optimization (CSSO) algorithm [18]. In this CSSO algorithm, a global optimization problem is divided into subdisciplines, each of which is optimized independently, with tradeoffs negotiated by a system-level coordination procedure. In order for the subspaces to be able to perform their local optimizations in parallel, each must have a means for accurately computing its own system states and performance, as well as a means for approximating the non-local information that is generated in the other subspaces. For computing local system states, each discipline provides a contributing analysis, which may themselves embody highly-complex systems of equations, requiring intensive processing for solution. Non-local approximations are required because in a coupled system, changes in a local design variable not only have a direct impact on the performances of that subsystem, but also have an indirect impact that result from the influence of changes on other parts of the system. During each subspace optimization, the design points that are visited are stored in a design database. At the end of the subspace optimizations, this data is used to approximate a system-level optimization problem that is solved to trade-off the subspace-optimal design moves in order to make a global design move. After the global coordination procedure is performed, a new design point is produced. The CSSO algorithm iterates until convergence is achieved, as measured by relative changes in both the design vector and the system objective function following the global coordination procedure.

A variety of small-scale problems have been used to test the performance of the CSSO algorithm thus far. These test problems have fallen basically into two categories: problems with a moderate number of disciplines (~10) yet simple contributing analyses, and those with only two or three disciplines yet highly-complex contributing analyses. An example of the former is an aircraft concept sizing problem, that considers the geometry of the aircraft, propulsion, aerodynamics, and flight regime [19]. As an example of the latter, in an ongoing project, we are developing methods for concurrent optimization of integrated circuit fabrication processes and cell libraries [11]. This particular technology will be particularly important in the development of PIM chips, where different circuit modules such as memory, logic, and I/O may wish to "push" the process in different directions in order to improve their own local performance. Through the use of the CSSO algorithm, it should be possible to find a process specification that is maximally beneficial to the whole product.

There are several features of multidisciplinary design optimization problems that make them a particularly interesting test case for PIM-based architectures and massively-parallel computing in general. Foremost is the hierarchical nature of the problem that is well-suited to the varying degrees of locality that PIM architectures offer. At the lowest levels, fine-grain parallelism may be used to accelerate computation of individual contributing analyses, while coarser levels of granularity may be applied to the subspace optimizations or system-level coordination. A second issue that will need to be addressed for MDO is how to automate tasks that currently require extensive human intervention. In particular, a great deal of expertise is often required in order to run individual contributing analyses and to interpret their results. Such interpretation is required because the numerical models used in many analyses can and do converge to solutions that are physically infeasible, which will mislead an optimizer. Robust analysis methods will have to be developed for both optimization and analyses in order to take advantage of the available computing bandwidth.

In order to apply massively-parallel MDO techniques to large-scale problems-with many disciplines and complex contributing analyses-it will be necessary to revisit many of the assumptions upon which the CSSO algorithm is founded. In particular, the current CSSO implementation employs a sequential quadratic programming algorithm, using local first- and second-order sensitivity information to guide the search for optimal designs. This approach may prove ineffective for large-scale problems that could produce a highly multi-modal design space. New techniques that use parallel search methods will have to be developed to address such problems.

5.3 Massive Image Data Compression

Another of the proposed applications is compressing and decompressing sequences of image data using new geometric techniques under development at Notre Dame. Problems involving sequences of images occur in many important applications, such as computer animation, satellite data process and analysis, flight simulation, medical data analysis, and data visualization. Further, it is often the case that the volume of image data (e.g., in an image database) is substantially large. For example, NASA Goddard Space Flight Center currently has 60 million images (with about a megapixel per image), spanning about 20 years. As technology advances, by the year 2010, the NASA image repository could hold 25 billion images. Image maintenance and retrieval is therefore a problem of great significance. However, in spite of the fact that data compression research has already produced highly sophisticated techniques for text data retrieval, the study on image data compression and retrieval has lagged far behind.

Our research at Notre Dame on compressing images has been based on the following fact: Image data is often organized and presented in the form of similar image sequences. In a typical sequence of picture frames (e.g., in a movie), often there is only small change from one frame to the next frame, and it is possible to describe the differences between one image and the following in the sequence by several elementary operations (such as adding, deleting, or substituting some new features, translating along a direction, rotating by a small angle, etc.).

Our approach for compressing sequences of similar images, mainly based on data structure and algorithmic techniques from computational geometry, is substantially different from others, with the following advantages:

1. It achieves both fast retrieval time and efficient memory usage. For example, let n be the number of images in a given sequence, and D be the total number of ``differences" between adjacent image pairs over the sequence. Then the retrieval time of our approach is that of generating one image (without having to generate the previous image first) plus an additive O(log D) overhead per pixel column, and the memory for storing the whole image sequence would be that of storing one image plus O(D).
2. While our image sequence compression scheme is virtually lossless, it avoids costly computations needed by performing transforms such as Fourier or Discrete Cosine. The basic operations involved in our compressing and decompressing processes are just usual operations on search trees like data structures, such as search, insertion, deletion, etc. This is especially significant to applications like image query systems (e.g., an image library). Often in such a system,

decompression either takes place very frequently, or takes place at the receiving end of data transmission where computational resources may not be the state of the art.

A key to our image sequence compression technique is the use of persistent data structures [4,14]. Note that ordinary data structures are ephemeral, namely, a change to the structure destroys the old data version (e.g., a deleted item is gone). In contrast, a persistent data structure supports fast access to multiple versions of dynamically changing data. Previously, persistent data structures have only been used to maintain multiple (similar) versions of one-dimensional information. In our case, a sequence of "similar" images can be viewed as a series of varying versions of an image that are generated by performing a sequence of elementary operations. However, to use persistent data structures to compress an image sequence (each image naturally contains two-dimensional data), several extensions must be made, including: (1) Partition the first image into a number of "slim" regions (for example, each such slim region initially may contain pixels in several adjacent columns), with each region being viewed almost as if it contains one-dimensional data. (2) Use a set of closely coupled persistent data structures to maintain subsequent versions of the regions of the image, with one data structure per region. (3) Maintain a "descriptor" data structure for describing certain special operations (e.g., translations and rotations) performed on the contents of the image along the image sequence.

Although our image sequence compression technique appears to be quite useful, we believe the prospects of extending this technique to our PIM based MPP systems is even more interesting. One important reason is that the volumes of image databases are often enormous. For example, a 10-million image library holds 10 terapixels (at 1 megapixel per image). Therefore, a great deal of computing, I/O, and storage power is needed to process compression and retrieval of image data of that kind of size. Our proposed MPP certainly has the potential for this kind of computational capability. Another important reason is that our compression technique seems to be quite readily extensible to the PIM based MPPs. For example, we can store several adjacent regions of the image at one processing node. This processing node then handles the computation of maintaining the persistent data structures for those regions. The interfaces between the processing nodes/chips of the PIM based MPP can then be used to accommodate the coupling/communications between data structures across the boundaries of their represented image regions.

6 Program Development Tools

Even though the systems addressed in this proposal are "point designs," they will exhibit most of the software development problems to be faced in a complete "general purpose" petaflops machine. While some are greatly alleviated by the organization pictured in Fig. 5 others still remain, in particular dealing with the massive amount of parallelism that naturally comes in a PIM design (upwards of a 100,000 processing nodes). Ongoing projects at Notre Dame are addressing a new set of techniques and tools applicable to the point design applications, including transformation of the basic equations or code, optimization of the scheduling of the tasks, and partitioning of the problem to reduce the communication between processors.

Transformation techniques are usually applied to get optimal execution rates in parallel and/or pipelined systems. The retiming (or loop pipelining) technique is a common and valuable tool in one-dimensional problems, represented by data flow graphs, which can maximize the parallelism of a loop body by regrouping the operations in iterations to produce a new iteration structure with higher embedded parallelism [3]. In this study, we will use new techniques to model loop bodies representing greater than one-dimensional problems by *multi-dimensional data flow graphs [13]*. The loop body is transformed, i.e., the existing delays are redistributed, preserving the original data dependencies. This technique has been successfully applied to several problems that have grown out of the wave digital PDEs discussed above

Once large amounts of parallelism has been extracted, such as by the above techniques, an additional problem is the need to map the solution to a fixed number of equal processing elements, such that all of them are able to execute simultaneously, with a minimum communication between processors. To solve this problem, the solution must be partitioned in blocks where the boundaries of such blocks will determine the amount of data to be transmitted. By using multi-dimensional retiming techniques the partitioning and mapping problems can be solved for high parallelism. The data communication, however, requires a data scheduling technique, in order to improve the use of data available locally at each PIM. A method called *carrot hole data scheduling* has been developed at Notre Dame to solve this problem [17]. This research proposes consideration of the incorporation of such a technique into a tool set, with a expected significant performance improvement in complex problems where there are high numbers of data dependencies between tasks.

Finally, even considering the reduction on the data transfer between processors obtained by the carrot-hole data scheduling, a more important optimization is to hide the overhead (or time consumed) caused by the remaining data transmissions. This problem has been target of the preliminary studies, resulting in a technique known as *communication-sensitive rotation scheduling* [16]. This study will extent such a methodology expressly for the candidate PIM configurations, particularly for multiple chip configurations.

7 Proposed Study

The overall objective of this study is to quantify the degree to which PIM-based MPPs can be built **and programmed** over the next 10 years that would provide **effective** performances of the 100+ tera(fl)op range against some real applications. This translates into two specific objectives: refining the generic point design for each application (both PIM chip and system), and developing a coherent view of all the software needed to implement each of the algorithms on its specific point design. In keeping with the objectives of NSF's call for proposals, the emphasis is on the latter, particularly on identifying areas of commonality and/or areas that offer the potential to grow into more "general purpose" software for PIM-based MPPs.

Given thus, the study's approach takes the following steps:

1. For each of the chosen applications a more careful estimation will be made of the overall computational flow, and the expected computational demands (memory, (fl)ops, I/O).
2. These characterizations will be used to define a most appropriate processing logic to fit within the confines of a PIM node macro as diagrammed in Fig. 2. It is expected that there may be three such definitions, one for each application.
3. Each of these macros will then be grown into revised point design chips (as in Fig. 4) and systems (as in Fig. 5), backed up with revised versions of Table 2.
4. For the three systems, a more careful analysis will be made of the modifications needed to be made to the system software running in the host microprocessor. At this point, we expect to see a very high degree of commonality among all three.
5. Each application will then be revisited with its specific point design (chip and system) in mind. The goal here is to construct a scenario for how the application would be transformed into real code, with emphasis on what real development tools would be needed at each step of the way. The emphasis here is on completeness - at each step from application definition to code debug, what needs to be done, what tools are needed to perform them, and do those tools (or the algorithms embedded in the tools) exist.
6. Finally, all three point designs (both hardware and software) will be revisited with the goal of abstracting out those parts which are common to all, and those which represent application-specific needs. For the latter, an initial determination will be made as to how much of an impact on overall effectiveness the lack of that feature or tool would be for that specific application.

The results will be summarized in a final report to be delivered at the Petaflops Workshop at the *Frontiers of Massively Parallel Computation*, Oct. 1996 (this conference).

8 Acknowledgments

The work reported in this paper was partially funded by NASA grants NAG5-2998/S1 and NAG-1-1561, and by the National Science Foundation under grants CCR-9623585, and MIPS-9501006 . Further funding will be from the NSF Point Design Study.

References

1. -, *The National Technology Roadmap for Semiconductors: 1994*, Semiconductor Industry Association, San Jose, CA.
2. -, Workshop on the Application and Algorithmic Challenges for PetaFlops Computing, Bodega Bay, CA, Aug. 1995.
3. Chao, L.F. and E. H.-M. Sha, " Retiming and Unfolding Data-Flow Graphs", *Int. Conf. on Parallel Proc.*, August 1992, pp. II 33-40.
4. Driscoll, J. R., N. Sarnak, D. Sleator, and R. E. Tarjan, "Making Data Structures Persistent," *J. of Computer and System Science*, Vol. 38, pp. 86--124, 1989.
5. Fettweis, A., "Wave Digital Filters: Theory and Practice," *Proc. IEEE*, Vol. 74, No. 2, pp. 270-327, February 1986.
6. Fettweis, A. and G. Nitsche, "Transformation Approach to Numerically Integrating PDE's by Means of WDF Principles." *Multidimensional Sys. and Signal Proc.*, Vol. 2, pp. 127-159, 1991.
7. Fettweis, A. and G. Nitsche, "Numerical Integration of Partial Differential Equations Using Principles of Multidimensional Wave Digital Filters," *J. of VLSI Signal Proc.*, Vol. 3, pp. 7-24, 1991.
8. Kogge, Peter M., "EXECUBE - A New Architecture for Scaleable MPPs," *Int. Conf. on Parallel Proc.*, St. Charles, IL, Penn State Press, Aug. 1994
9. Kogge, Peter M. "Processors-in-Memory (PIM) Chip Architectures for Peta(Fl)ops Computing,"Petaflops Workshop, *1995 Frontiers of Massively Parallel Computation*, McLean, Va., Feb., 1995.
10. Kogge, Peter M., Toshio Sunaga, Histada Miyataka, Koji Kitamura, and Eric Retter, " Combined DRAM and Logic Chip for Massively Parallel Systems," *6th Conf. on Advanced Research in VLSI*, Raleigh, NC, IEEE Press, March 1995, pp. 4-13.
11. Lokanathan, A.N. J. B. Brockman, and J. E. Renaud, "A Methodology for Concurrent Fabrication Process/ Cell Library Optimization", 33rd ACM/IEEE Design Auto. Conf. June 1996.
12. Myricom, http://www.myri.com
13. Passos, N. L. and E. H.-M. Sha " Full Parallelism in Uniform Nested Loops using Multi-Dimensional Retiming", *Proc. Int. Conf. on Parallel Proc.*, Aug. 1994, vol. II, pp. 130-133.
14. Sarnak, N. and R. E. Tarjan, "Planar Point Location Using Persistent Search Trees," *Comm. of the ACM*, Vol. 29, pp. 669--679, 1986.
15. Sterling, T., P. Messina, and P. Smith. *Enabling Technologies for Peta(FL)ops Computing*, MIT Press, Cambridge, MA, 1995.
16. Tongsima, S., N. L. Passos and E. H.-M. Sha " Architecture Dependent Loop Scheduling via Communication Sensitive Remapping", *Int. Conf. on Parallel Proc.*, Aug. 1995, II, pp. 97-104.
17. Wang, J. Q., N. L. Passos and E. H.-M. Sha " Optimal Data Scheduling for Uniform Multi-Dimensional Applications," to appear in *IEEE Trans. on Computers*, 1996.
18. Wujek, B.A., J. E. Renaud, S. M. Batill, and J. B. Brockman, "Concurrent Subspace Optimization Using Design Variable Sharing in a Distributed Computing Environment," Proceedings of the 1995 Design Engineering Technical Conferences, Advances in Design Automation, ASME DE-Vol. 82, pp. 181-188.
19. Wujek, B.A., E. W. Johnson, J. E. Renaud, and J. B. Brockman. "Design Flow Management and Multidisciplinary Design Optimization in Application to Aircraft Concept Sizing," 34th AIAA Aerospace Sciences Meeting and Exhibit, January, 1996.

Appendix B

Computing Component Characterization

a portion of the draft proceedings of

The Petaflops Architecture Workshop

Oxnard, CA, April 1996

Chapter 8

Architecture Working Group

8.1 Computing Component Characterization

The NSF point design studies by their very nature must assume computing and memory components that do not exist today, and will not for the better part of a decade. Given the array of architectural choices available today, and the difficulty for even the expert to fully understand all of them and how they will look in tomorrow's technology, it is clear that without some sort of rational and common projections, it will be difficult to validly compare and contrast the alternative point design approaches as they develop.

To help provide such a common baseline, at least for the mainstream CMOS technology, this chapter will summarize a series of projections based on the Semiconductor Industry Association's (SIA) 1994 National Technology Roadmap. This section is organizedis as follows: a quick summary of the technology trends, a projection of commodity memory components, and then a projection of how those trends may result in high performance Symmetric Multiprocessors (SMP) chips. A spreadsheet program is described that will allow exploration of alternative scenarios. This spreadsheet is used to give some projections for future DEC ALPHA 21164-like chips.

8.1.1 Basic Technology

Table 8.1.1 summarizes some of the key CMOS characteristics listed in the SIA Roadmap (as distributed at the Petaflops Architecture WorkShop, PAWS-96), in three-year intervals from 1995 through 2010. The number in "(#)" after many of the characteristics reference the page in the SIA document where these data originated. The other rows were derived by simple

Table 8.1: Basic SIA Roadmap CMOS Trends

Characteristic (SIA Pg. #)	**Units**	**1995**	**1998**	**2001**	**2004**	**2007**	**2010**
Feature Size(11)	μm	0.35	0.25	0.18	0.13	0.1	0.07
Vdd(14)	volts	3.3	2.5	1.8	1.5	1.2	0.9
DRAM							
Chip Capacity	MB	8	32	128	512	2,048	8.192
Chip Size(12)	mm^2	190	280	420	640	960	1400
Density	MB/cm^2	4	11	30	80	213	585
Chip Cost	Rel. to 1995	1	1.65	2.82	3.76	7.53	12.05
\$/MB	Rel. to 1995	1	0.41	0.18	0.06	0.03	0.01
High Performance Microprocessor Logic Based Chips							
Transistors/Chip(16)	MT	12	28	64	150	350	800
Chip Size(B2)	mm^2	250	300	360	430	520	620
Density	MT/cm^2	5	9	18	35	67	129
Clock: μP(12)	MHz	300	450	600	800	1000	1100
Clock: DSP(46)	MHz	400	600	800	1100	1500	1900
SRAM Cache Density(11)	MB/cm^2	2	6	20	50	100	300
Cost/Transistor (B2)	millicents	1	0.5	0.2	0.1	0.05	0.02
Chip Cost	Rel. to 1995	1	1.17	1.07	1.25	1.46	1.33
ASIC Logic Chips							
Transistors/Chip	MT	9	26	53	108	275	560
Chip Size(B2)	mm^2	450	660	750	900	1100	1400
Logic Density(B2)	MT/cm^2	2	4	7	12	25	40
Clock(B2)	MHz	150	200	300	400	500	625
Minimum Chip Cost	Rel. to 1995 μP	0.75	1.1	0.88	0.9	1.15	0.93
NRE Chip Cost	\$/volume	27	26	26	32	55	56

arithmetic computation from the provided numbers.

Under the "Units" column, "MT" means "Millions of transistors, "MHz" stands for "Megahertz," "NRE" is "Non-recurring Expense,"

For logic, it should be stressed that there are at least three distinct trends that one can assume. First is the full custom, maximum density, very high design cost, but high volume, "Microprocessor" logic family. Two sets of clock rates are given for such designs: a "high performance microprocessor" clock (listed as "μP clock"), and an even higher clock rate "digital signal processor" (DSP) that might be achieved for specialized designs. The third set of logic is Application Specific Integrated Circuit (ASIC), which has lower transistor density and lower clock rate, but is more amenable to relatively inexpensive specialized designs. Note that the largest ASIC chips are almost uniformly twice the size of the custom microprocessor chips, making up in area what they lose in density. Note also that we will assume that SRAM cache density is the same for both ASIC and microprocessor logic, since in either case it would be a custom macro.

Given the difficulty in interpreting the cost numbers in the SIA Roadmap,

Table 8.2: Basic SIA Off-Chip Signaling Trends

Characteristic	Units	1995	1998	2001	2004	2007	2010
μP Package Pins(12)		512	512	512	512	800	1024
μP Signal Pins		410	410	410	410	640	819
ASIC Package Pins (12)		750	1100	1700	2200	3000	4000
ASIC Signal Pins		600	880	1360	1760	2400	3200
Chip-To-Board Rate	MHz	150	200	250	300	375	475
Aggregate μP SIA Bandwidth	GB/s	6.1	8.2	10.2	12.3	24	38.9
Aggregate ASIC SIA Bandwidth	GB/s	9	17.6	34	52.8	90	15.2
Aggregate μP 2-Wire Rate	GB/s	20.5	20.5	20.5	20.5	32	41
Aggregate ASIC 2-Wire Rate	GB/s	30	44	68	88	120	160

the chip costs listed here have all been normalized to the SIA 1995 basis, and assume high volume production. Lower volume specialized chips will cost more. In particular, the "Minimum Chip Cost" for the ASIC chips assumes no allocation for the design NRE. The last row in the table gives a "per chip" estimate of this for volumes from 10,000 to 100,000.

The SIA trends assume two separate types of chips: logic and DRAM. As several recent experimental chips have demonstrated, such as the IBM EXECUBE chip, it should be a possible in the future to mix both on the same die. To do this may require special consideration in several areas such as off-chip contacts and speed of on-chip logic. These are discussed later.

Off-Chip Signaling Potential

Table 8.1.1 totals some basic off-chip bandwidth numbers derivable from the SIA data. It includes the number of available packaged pins, and an estimate of the pins available for data (by allocating 20% of the total to power and ground). Also, two different signaling protocols are assumed: the "Chip-to-Board" rates described in the SIA Roadmap; and a two-pin differential scheme discussed at PAWS-96 that might allow a signaling rate of 1 GHz.

The aggregate signaling rates come from the product of the number of pins on the package, times the signaling rate, divided by 10 to convert to bytes per second (10 was used instead of 8 to allow something for parity and for protocol). Note that in terms of off-chip communications, such as to memories, the real transfer rate must take into account both bandwidth for data and bandwidth for the address or other control information needed to instigate the transfers.

One serious caveat to using these numbers blindly may occur if chips contain very significant amounts of both logic and DRAM. Reaching the large number of pins listed in Table 8.1.1 assumes area contacts on the chip, and when there are significant amounts of DRAM present, it may not be possible to place such contacts over the DRAM cells. In such cases, it may be necessary either to reduce the area pins by the percent of area that is DRAM, or to assume peripheral arrays only.

8.1.2 Commodity Memory

From Table 8.1.1, one can develop a scenario of potential commodity DRAM chips. From that, one might want to assume that for example, for a 2007 time frame machine a 2-GB chip would be available. However, from historical observations of the 64-Mbit (8 MB) chip, the capacities listed here seem to correspond to the year of first limited production. Thus, the real cost savings listed are probably not apparent until perhaps three years later. This means that while one might want to assume in 2007 a 2-GB chip for reducing parts count, the cost per MB is probably more appropriately taken as the 2004 numbers.

In terms of absolute price, in 1995 the street price of packaged and tested DRAM was in the $20 to $40 per megabyte. Given the above discussion, this would translate into 2007 prices of $1 to $2 per MB, or $1M to $2M per terabyte.

SMP CPU Chip Projections Spreadsheet

Taking the above, and related, technology numbers and projecting what commodity microprocessors might be is much more difficult than for DRAM. Technology will have passed the point where entire computers (CPUs, caches, and at least some memory) are on one chip, In fact technology will allow multiple such machines. Each vendor will be making a separate set of design decisions, based on perceived marketplace needs. Consequently, it is irrational to select a single part type as *the* building block for all point designs. Instead a spreadsheet has been constructed to allow specifying a set of architectural parameters, and from that deriving the capabilities of chips that have those parameters, over the same time frame as described above.

The spreadsheet takes the specified architectural characteristics, and assembles an estimate of chip size based on all the SIA numbers discussed above. Table 8.1.2 lists the basic input area for the spreadsheet. The numbers in the larger bold font are the ones input by the user. Those shown in the table are for a trial run assuming the DEC ALPHA 21164 microproces-

Table 8.3: Input Parameters for the SMP Chip Estimator

File:Petatech		SMP Chip Design Projections Worksheet				
05/24/96		...			Enter Design Choices Here	...
ALPHA 21164		L1:Instr	L1:Data	L2	CPU Core(MT)=	2
Block Length	Bytes	32	64	64	Logic Type(a,u,d)=	d
Entries	#	256	256	512	# CPUs/Chip=	4
Associativity	#	1	1	3	On-chip DRAM (MB)=	0
Tag(Adr+Status)	Bits	36	72	36	Reference Clock Rate(MHz)=	300
# Ports	#	1	1	1	Reference Feature Size(u)=	1
					L2 Replicated(r) or Shared(s)	r
Note: L1-Data Cache assumes 2×the tag & 2×the Block to reflect the duplication to get 2 ports						

sor, which is discussed later. They assume numbers for an on-chip two-level cache hierarchy, with and without potential on chip DRAM for memory. The L1 cache is assumed to have two parts, a data cache and an instruction cache.

It is worth discussing the terms in Table 8.1.2 because they would help others use the spreadsheet, and because they represent some of the key characteristics that influence both chip size and performance characteristics.

1. The "Block Length" (in bytes) is the number of bytes read out of the cache on a single access in a single associative set.

2. The "Entries" is the number of cache rows present in the cache array.

3. The "Associativity" is the number of associative sets in the cache (use 1 for a direct mapped cache for a decent approximation).

4. The "Tag" is the number of bits from the address used in the comparison from each set, along with LRU, status, etc. bits. A useful approximation is the number of bits in the virtual address, minus the number needed to represent a page, plus 6 to 8.

5. The "#Ports" is the number of separate ports available from the cache for simultaneous access. Numbers greater than 1 for L1 data caches will become more frequent in the future as microprocessors with multiple load store units become more common. Shared L2 data caches might also have multiple ports.

6. The "CPU CORE" is the number of transistors of logic involved with the CPU itself, exclusive of caches, but including all processing logic, TLBs, write back buffers, etc.

7. The "Logic type" specifies what kind of logic to assume. Three separate values are permitted. "u" selects the custom microprocessor chip size, transistor density, and clock rate. "d" selects the same as for "u," but assumes the higher DSP clock rate. "a" assumes the ASIC size and speed chip.

8. The "#CPUs/Chip" indicates how many of the above CPUs (each with its own L1 cache set) are to be placed on the chip. Any positive number will result in the spreadsheet attempting to fit that number (and no more or less) on the chip of the size specified by the "Logic Type" entry. A "0" here will tell the spreadsheet to jam as many CPU+L1 combinations as possible onto the chip.

9. The "OnChip DRAM" allows specification that a certain number of MB of DRAM are to be attempted to be placed on the chip, along with the CPUs discussed above. Given the current state of technology, it may be appropriate to assume the "a" model of logic in order to get a cost effective chip if DRAM is included. If either "u" or "d" is selected, the cost of the chip might be estimated as some multiple of the sum of the costs of DRAM and logic chips of the same size, since both DRAM and logic fabrication processes might need to be run against the same wafer.

10. The "Reference Clock Rate" is what clock rate the CPU used in the modeling exists at today. This number is used to provide a relative performance multiple for our resulting chip vs known current benchmarks.

11. The "L2 Replicated or Shared" entry allows specification of either "r" or "s." For the former, each CPU+L1 combination on the chip receives in addition its own, separate, L2 cache. For the "s" entry, the spreadsheet assumes exactly one L2 cache for all the on chip CPUs. In the latter case, the user might want to assume multiple L2 ports to reflect the multiple usage.

A snapshot of the output is shown in Table 8.1.2, with the rows defined as follows:

1. "Feature Size" is the number selected by the SIA as representative of the leading edge technology of the time.

2. "Clock Rate" is the clock rate selected by the "Logic Type" entry above.

3. "CPUs/Chip" is either the number of CPUs that the user specified as being on chip, or the maximum number of CPUs that would fit (if the user parameter was 0 as discussed above).

4. "Occupied Area" is the square millimeters taken up by the CPUs, their associated L1s, the L2 (or replicated L2s), and specified on chip DRAM. Again, this is computed by the spreadsheet from the technology densities projected from the SIA numbers.

5. "Spare Area" is area on the chip that is unused. Note that the size of the chip assumed is based on the "Logic Type" entry described above. A negative number here corresponds to an overflow on the chip (the specified number of CPUs won't fit).

6. "If Spare=DRAM" lists how many extra MB of DRAM could be placed on the chip if all the spare area from row 5 was converted into DRAM at the best density of the day.

7. "Relative Perf." is the product of the number of CPUs on the chip and the clock rate specified by the "Logic Type" choice, all divided by the "Reference Clock Rate." Thus it represents a peak performance multiple that might be achieved over the current baseline design. If "Occupied Area" is greater than the assumed chip size, then this multiple is listed as "NoFit" to indicate the chip design is infeasible at this point in time.

8. "Total Transistors" simply represents all the logic, SRAM, and DRAM transistors that went into the area described in item 4, Occupied Area.

9. "-If DRAM filled" adds onto Total Transistors (item 8) one transistor for each DRAM bit that was added to fill up the remaining space specified in item 6, If Spare=DRAM.

Representative Output: the DEC Alpha

As a test case, and one that might actually be representative of a natural design point, the appropriate parameters for the DEC Alpha 21164 chip were placed in the worksheet, with the logic type varied through all three types

Table 8.4: Sample Output of Computed Chip Characteristics

	Units	1995	1998	2001	2004	2007	2010
Feature Size	μm	0.35	0.25	0.18	0.13	0.1	0.07
Clock Rate	MHz	400	600	800	1100	1500	1900
CPUs/Chip	4	4	4	4	4	4	
Occupied Area	mm^2	450	200	89	43	22	11
Spare Area	mm^2	-200	100	271	387	498	609
If Spare=DRAM	MB	(7)	9	66	248	851	2,851
Relative Perf.	Multiple	NoFit	8	11	15	20	25
Total Transistors	MT	37	37	37	37	37	37
-if DRAM filled	MT	(23.3)	120	633	2,265.2	7,692.7	25,697.4

"a,", "u," and "d," and a combination of different numbers of CPUs placed on chip. This chip was a representative choice since it is a performance leader, is well documented, and already has a two level memory hierarchy (L1 and L2) on the current chip. Chip clock rates as of 1995 were 300 MHz (in agreement with the SIA projections), with up to four instructions issued per cycle, of which two could be floating point multiply accumulates—providing a peak performance of 1,200 MF at 300 MHz. We will use these peak numbers as a baseline, knowing that in reality sustainable performance will be several multiples less than this.

As a verification exercise for the known 21164 parameters, the worksheet computed that a chip with a single CPU would contain 9.3 million transistors—exactly what the real one does today. Also, the area at 0.35 micron feature size is 165 sq. mm. The current chip, when built out of 0.5 micron technology, is 299 sq. mm, which if everything scaled according to the linear scaling laws, would convert to $299\times(0.35/0.5)^2=$ 146 sq. mm. This is within 13% agreement, well close enough for estimation. We suspect that the difference may be due to how the SIA computed cache density by including an overhead term from the tag arrays, whereas we computed that term separately from the provided information.

This spreadsheet estimation technique allows computation, as a function of time, of a range of potential SMP chips; their relative performance, on chip DRAM potential, and the ratio between storage (in MB) and performance (in MF). The latter is of particular interest, because if the ratio matches any particular application, then a point design can be made out of only one part type. Tables 8.1.2, 8.1.2,and 8.1.2 are summaries of these three values from the spreadsheet, again assuming an ALPHA 21164 as a baseline. The three CPU counts run include a single CPU per chip, a 16-CPU chip configuration, and one where the chip is packed with as many CPUs as possible, with no

Table 8.5: Peak Performance in MF per Chip

Year	Max DRAM: 1 CPU			16 CPUs			No DRAM: Max CPUs		
	ASIC	μP	DSP	ASIC	μP	DSP	ASIC	μP	DSP
1995	1,200	1,200	1,200				1,200	2,400	3.600
1998	1,200	2,400	2,400				7,200	10,800	14.400
2001	1,200	2,400	3,600	19,200	38,400	51,600	24,000	38,400	51.600
2004	1,200	3,600	4,800	25,200	51,600	70,800	72,000	124.800	171.600
2007	2,400	3,600	6,000	32,400	63,600	96,000	228,000	384.000	576.000
2010	2,400	4,800	7,200	39,600	70,800	121,200	618,000	994.800	1,717,200

extra onboard DRAM. In all cases, we assume that each CPU has its own 96 KB L2, as is present today (it may be reasonable to assume larger L2s). Note that the technology is not dense enough for 16 CPUs until 2001. Note also that in Table 8.1.2, the columns for Max CPUs list the number of CPUs on chip and not the extra DRAM (since there isn't any).

Given these numbers, there is a variety of potential chips that might make for suitable point design basis. For purposes of discussion we will assume 2004 technology for a 2007 point design system.

For a single CPU per chip, but stuffed with DRAM, we can get chips with 1 GF to almost 5 GF of performance, and sufficient DRAM to give a memory ratio of from 0.06 to 0.47—serious numbers that would allow construction of a single part type machine of from 20,000 to 100,000 chips.

For a 16-way SMP on chip, peak performance would run from 25 to 70 GF, with perhaps a 100 MB of DRAM for an L3, meaning that a few thousand CPU chips would be suitable, but with external DRAM a necessity. At 2 GB per chip, this might require upwards of 50,000 DRAMs for a full 100 TF of memory. Bandwidth for this external memory also needs to be investigated.

Stuffing a chip with CPUs only, would reduce the total number of CPU chips to under a thousand, but with no on chip DRAM for L3, and thus more bandwidth needed off chip.

In all cases we are talking about a machine with on the order of 50.000 to 100,000 separate CPUs all running concurrently. System software must be capable of managing this level of parallelism efficiently if any meaningful fraction of the system's peak is to be utilized.

In all cases, power needs to be more carefully estimated to ensure that the chips have not exceeded maximum thermal limits.

Table 8.6: DRAM on Chip at Peak Performance (in MB) per Chip

Year	Max DRAM: 1 CPU			16 CPUs			No DRAM: Max CPUs		
	ASIC	μP	DSP	ASIC	μP	DSP	ASIC	μP	DSP
1995	10	5	5				2	2	2
1998	54	23	23				9	6	6
2001	174	82	82	42	1	1	20	16	16
2004	563	268	268	372	165	165	45	39	39
2007	1861	878	878	1615	740	740	114	96	96
2010	6527	2890	2890	6130	2698	2698	247	226	226

Table 8.7: Ratio of DRAM to MF per Chip

Year	Max DRAM: 1 CPU			16 CPUs			No DRAM: Max CPUs		
	ASIC	μP	DSP	ASIC	μP	DSP	ASIC	μP	DSP
1995	0.01	0.00	0.00						
1998	0.05	0.01	0.01						
2001	0.15	0.03	0.02	0.00	0.00	0.00			
2004	0.47	0.07	0.06	0.01	0.00	0.00			
2007	0.78	0.24	0.15	0.05	0.01	0.01			
2010	2.72	0.60	0.40	0.15	0.04	0.02			