

# PIM Architectures to Support Petaflops Level Computation in the HTMT Machine

Peter M. Kogge, Jay B. Brockman, Vincent W. Freeh  
CSE Dept., Univ. of Notre Dame, Notre Dame, IN 46556

## Abstract

*The HTMT project is an ambitious attempt to combine a variety of emerging technologies into a petaflops-level computing system available many years before an equivalent machine can be built from current technologies. One of the key problems in such an architecture is overcoming latencies between the main memory and the high performance CPUs, which can grow to literally tens of thousands of cycles. In HTMT the approach taken to overcoming this is a multi-level memory system, with most of the levels to be fabricated using Processing-In-Memory (PIM) technologies in architectures which actively manage the flow of data without centralized CPU control. This paper overviews the current architecture for such chips within the context of the HTMT system, and how this architecture supports the expected execution model.*

## 1. The HTMT Program

The HTMT project (*Hybrid Technology Multi Threaded Architecture*)[1] [19] is a first step in an attempt to combine multiple unconventional technologies into a design for a world class supercomputer. It is an outgrowth of a series of workshops [18] and an NSF-sponsored program to investigate novel point designs to achieve a hundred teraflops (the latter documented in a series of papers in [2]). HTMT as a formal multi-agency funded, multi-institution project began in 1997, and completed an initial feasibility study phase in the summer of 1999. A current phase is focused on further refining the architecture and risk reduction in preparation for potential hand-off to a full scale development.

The overall goals of the HTMT project are fourfold:

- explore and characterize a synthesis of exotic technologies, innovative architectures, and aggressive latency management techniques in a way that could dramatically accelerate availability of petaflops scale comput-

- ing systems, where a petaflop ( $10^{15}$  flops) is several orders of magnitude times more compute power than any computer system demonstrated to date, and lies beyond the envelope of current programs such as ASCI,
- do so for applications which are both of significant value and “non-trivial” (i.e. requiring significant amounts of storage and/or communication),
  - develop components, architecture, and design data to minimize future design risk and establish confidence in the conclusions, and
  - determine the feasibility and effectiveness of the HTMT strategy for executing real-world computations at revolutionary performance levels.

There are projects other than HTMT which are also currently targeted at a petaflop, such as IBM's *Blue Gene* system [3]. Such designs, however, focus on specific classes of applications (protein folding in the case of *Blue Gene*) for which limitations in some aspect of design are acceptable (for example: limited memory, connectivity, or application development tools).

It is projected that without the HTMT project, or something similar, it would be the 2010s before conventional technology will reach a level where such systems are generally available and useful for a broad class of applications, and that when they do, parallel programs to execute on them will require finding opportunities for literally multi-million-way parallelism in real applications. A successful HTMT program will thus lay the groundwork for prototypes in the 2005-06 time frame that achieves petaflop-level performance, on multiple real problems with much less required application-derived parallelism. In addition, if successful, HTMT should spin off a wide variety of technologies and architectural techniques that would have a broader range of impact.

A key to the successful operation of HTMT is a memory system into which much of the data management functions can be migrated. This provides both a sophisticated “prefetching” capability which reduces effective latency, and also moves many computations out to a highly parallel

memory system where they can be performed without data movement at all. This paper focuses on this memory system. Section 2 introduces enough of the HTMT architecture to motivate the need for such a memory. Section 3 then briefly reviews the PIM technology assumed for the memory's construction, and the architecture of the resulting chips. Section 5 then discusses the proposed model of execution, which is centered on having the PIM chips perform the lion's share of the data management functions in a program.

More detail on the HTMT PIM design and related technology trade-offs as of summer 1999 can be found in a project Technical Report [14]. It is expected that as the current phase of HTMT progresses, future papers will go into depth on many of the topics introduced here.

## 2. The HTMT Architecture

The baseline HTMT architecture, Figure 1, consists of the following components (followed by the corresponding research organizations):

- 4,096 extremely fast 256 GF CPUs (termed *SPELLs*) and local memories (termed Cryogenic RAMs, or *CRAMs*) along with a high speed interconnection network, (*CNET*), all built out of a superconducting logic technology (SUNY Stonybrook),
- Two levels of "smart" SRAM and DRAM memories based on PIM technologies, grouped into clusters (University of Notre Dame),
- A very high bandwidth optical network, termed the *Data Vortex*, to interconnect the clusters of memory (Princeton University),
- 3-dimensional Holographic memory RAMs (*HRAMs*) for extremely dense on-line backing storage (California Institute of Technology),
- Large external arrays of disks and tape silos,

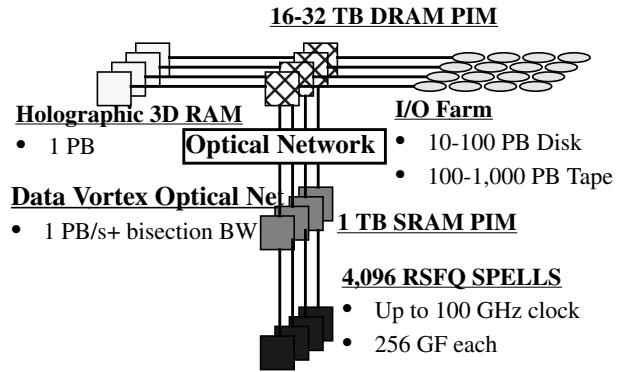


Figure 1. HTMT Block Diagram

- An overall system execution architecture based on a combination of an active message-like paradigm (Notre Dame) and a multi threaded data percolation model (Univ. of Delaware).

In addition, personnel from JPL, IDA, Argonne National Labs, NASA Ames, Tera Computer Corp., TRW, Hypres, TI, San Diego Supercomputer Center, UC Berkeley, and UC Santa Barbara actively participate in a variety of roles. Overall program management is out of JPL.

Figure 2 (courtesy Larry Bergman, Cal Tech) is a rendition of a potential HTMT machine room, with labels indicating the location of the major structures. Table 1 summarizes the capacities, latencies, and bandwidths between different subsystems within this design, as seen from a program running in a *SPELL*. Units "W" are 64-bit quantities - a double precision floating point number. Also included are ratios per flop.

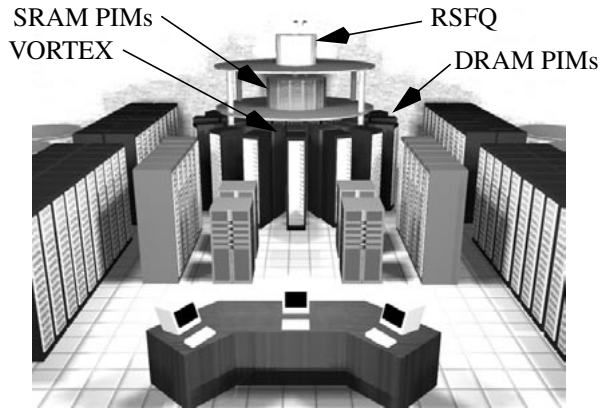
The parts of the HTMT system architecture of relevance to this paper are outlined below. The HTMT web site has more detail on both these and the subsystems skipped here.

### 2.1. The Superconducting Region

Today's best design point for a petaflop is exhibited in

Table 1: The View from a *SPELL*

	Capacity	Capacity	Round-trip Latency	Read Bandwidth	Write Bandwidth
<b>Units</b>	1 W = 8B	W/Flop	cycles	W/Flop	W/Flop
<b>Local CRAM</b>	256KW	0.00001	70	1	1
<b>Local SRAM</b>	32MW	0.000125	240+	0.25	0.25
<b>Remote CRAM</b>	512MW	0.002	270	0.08	0.08
<b>DRAM Cluster</b>	512MW	0.002	16,000	0.04	0.04
<b>HRAM on 1 Cluster</b>	32GW	0.125	67,000	0.04	0.04
<b>I/O Farm</b>	10-100 PB	10s	600,000,000	TBD	TBD



**Figure 2. HTMT Machine Room**

IBM's Blue Gene machine, where multiple multi-threaded 1 GF CPUs are implemented on a single chip with relatively small amounts of memory. A million of these nodes is needed for a petaflop, and requires many millions of concurrent threads in order to run at sustained petaflops.

If one were to scale CPU designs forward a few years, perhaps the best one might imagine is a superscalar design capable of perhaps 8-way instruction issue with 4 floating point function units capable of a multiply-accumulate launch every cycle. Assuming a 2005 CMOS technology [17] perhaps 3.5 GHz might be achievable, yielding a theoretical peak of perhaps 25 GF. 40,000 of these CPUs would be needed to peak at a petaflop. Realistically, latency concerns in getting results from a larger memory hierarchy would throttle this performance tremendously. More probably, upwards of a million such CPUs, or aggressive multi-threading, or both, would prove necessary to get anywhere near a sustainable petaflop. Further, to get sufficient bandwidth for general applications requires a huge number of interconnects.

This vicious cycle of a huge number of threads and very expensive inter-node connections drove the HTMT team to consider radically different technologies that would allow individual CPUs with far higher clock rates, greatly reducing the number of such devices needed for a petaflop (thus simplifying the interconnect tremendously), and greatly increasing the performance of individual threads (thus greatly reducing the number of concurrently active threads needed to be extracted from the program).

The technology baseline for HTMT's main computation engines is *Rapid Single Flux Quantum* (RSFQ), a superconducting-based technology which appears at this time to permit clock rates of perhaps eventually as much as a 100 GHz - more than an order of magnitude better than that with CMOS. A two-level multi-threaded pipelined CPU, termed a *SPELL*, has been designed with the capability of supporting up to 16 active threads, each of which could support 8 concurrent smaller threads termed strands [8].

In the current design only 4,096 such CPUs are needed for a realistic petaflop. Each such CPU has a large register file and a few MB of very high speed static local memory (termed CRAM) also built out of RSFQ devices, and a connection to the CNET - an RSFQ based high speed fabric between the nodes. All of these RSFQ-based subsystems are kept at 4° K to maintain their superconductivity.

## 2.2. CMOS Memory Hierarchy and Model

To achieve a sustained petaflop by 2006-2007 using 2005 technology, HTMT must have a system memory with three properties: high capacity, low effective latency, and high bandwidth.

To achieve enough memory capacity requires using the densest available memory for the bulk of the high-speed random access storage - DRAM. A common rule of thumb for scientific computing is that 1 byte per flop is balanced, but for a petaflop, implementing a petabyte out of any foreseeable CMOS technology is unaffordable. (In HTMT we do have a petabyte of relatively random access memory, but it is made out of HRAMs). There is, however, a wide class of HTMT-relevant problems for which a "3/4 rule" may be applicable - for N GF, only  $N^{3/4}$  GB or DRAM-like memory is needed. This reduces DRAM requirements to perhaps 16-32 TB. Even in 2005, this is 32K-64K chips.

If one compares CRAM and DRAM from Table 1, it is obvious that the latency gap—70 SPELL cycles to access CRAM versus 16,000 SPELL cycles to access DRAM—is too large. Further, the storage per flop available from the CRAM is woefully inadequate for a conventional main memory-cache hierarchy. Consequently, another level of memory, this time made from SRAM, is inserted between the two. In 2005 technology, a TB of such SRAM memory provides about 256 times as much storage as CRAM, at a latency about 3 to 5 times as long, and requires about 16K chips - 4 per SPELL.

Looking at the latency ratios between a SPELL and SRAM versus SPELL to DRAM, there is still a large gap which by conventional wisdom ought to be covered by insertion of yet another memory level. For HTMT no such intermediate technology has emerged, and even if it had, the cost of an additional set of interconnects would have been prohibitive. Thus the hierarchy is CRAM-SRAM-DRAM-HRAM-Disk-Tape.

Finally, two other questions needed to be answered about this memory hierarchy. First, are intermediate levels (i.e. at least CRAM and SRAM) to be treated as caches (with tags, replacement policies, etc) or as addressable memory. They were left as directly addressable memory partially because of the cost of managing the tags, partially because of the cost of any cache coherency protocol which would have to be applied over up to 4,096 CPUs, but more

importantly because as described later, the SRAMs and DRAMs have significant amounts of programmable logic, and leaving the storage as addressable allows independent programs in the RAMs to manage data transfers directly.

The second major question is whether or not there ought to be a virtual memory mechanism in place as seen by a program running in the SPELL. Initial thoughts were that this would introduce an excessive overhead, particularly in the RSFQ regions, but considerations of variable interleaving for complex object access and reliability<sup>1</sup> led to a relatively simple mechanism handled primarily in the SRAM and DRAMs.

### 2.3. Interconnect

There are four significant areas of interconnection in HTMT: SPELL to SPELL, SPELL to SRAM, SRAM to DRAM, and DRAM to HRAM. For the first, an RSFQ-based very high speed CNET couples SPELL/CRAM clusters within the cryostat. Between the cryo area and the SRAM layer are a huge, but manageable, number of wire interconnects. At this time this interconnect assumes an extension of the point-to-point half duplex equalized differential signalling as described in [7], where each pair of wires is assumed to support 10 Gbps. The interconnects between a DRAM chip and a string of HRAM chips uses a similar technology, as does interconnections between neighboring SRAM or DRAM chips in their clusters.

The interconnect of most interest from the PIM perspective is the Data Vortex - an all optical network between the SRAM and DRAM layers [4]. The Data Vortex consists of several tens of thousand optical fibers which lead between PIM clusters through a multi-layer optical switch. These fibers are uni-directional: some are driven as input ports by the PIMs; others deliver data to the PIMs. Each fiber supports a combined time-division-multiplexed (TDM) and a wave-division-multiplex (WDM) scheme where multiple optical frequencies (colors) are carried simultaneously on the fiber. The current design accepts single packets of information on one fiber, and transmits them, as packets, to some other fiber. Each packet is TDM split into a dead space (to allow for jitter, etc in the system), a header (to control routing through the optical network), and a payload of data. The header and payload are both WDM, meaning that up to 64 "bits" of information can be on the fiber at the same time.

The WDM in the header provides one bit at each of the multiple simultaneous color bands, with the presence or absence of energy at that color indicating a desired switch-

---

1. Assuming that even if individual memory chips had MTBFs in the 100,000s of hours, the whole system would see chips dropping out regularly at a few hour intervals. Even a simple virtual mechanism would allow programs to function around such faults.

ing action inside of the Vortex. The current topology of the Vortex allows optical data entering on any one input fiber to exit out of any other output fiber, without any conversion back into an electrical format. A maximum latency of about 100 ns is forecast.

The WDM in the payload also supports 64 colors. Each of the optical frequencies is modulated as a bit stream at 10 Gbps, providing an aggregate 640 Gbps (= 80 GBps = 10 GWps) data rate. The total data size carried in one payload is thus the length of the payload in ns, times the data rate possible in one ns (80 bytes). While the exact length of the payload is still a system parameter, longer payloads clearly result in increased effective data bandwidth on the fiber. Current design parameters have focused on a payload of around 2K-4K total bits. Given these numbers, the bisection bandwidth of the Vortex is on the order of petabytes per second.

### 2.4. Packaging PIMs into Clusters

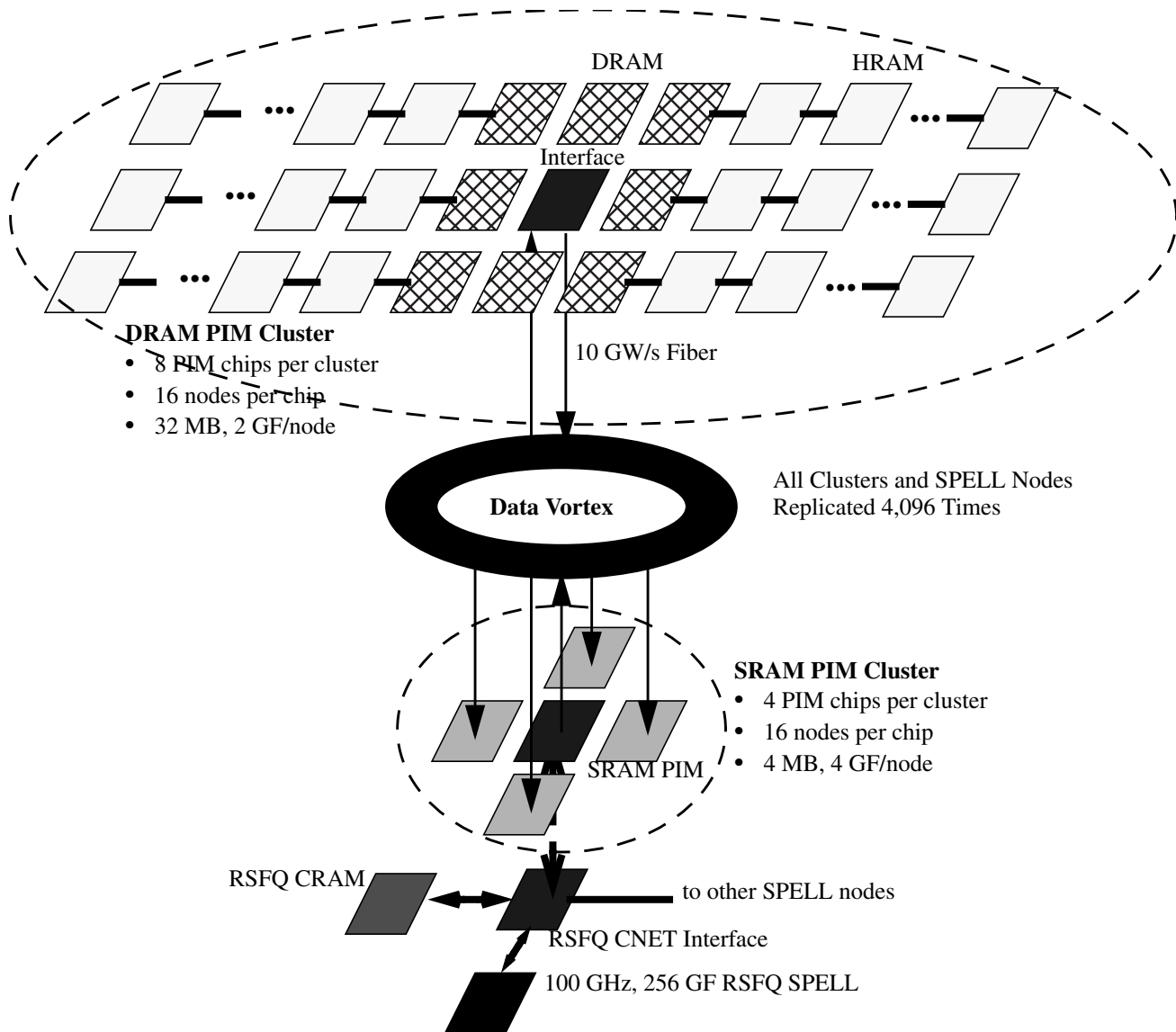
To simplify the packaging and scalability of the HTMT system, the SRAM and DRAM/HRAM memory levels are split into a large number of relatively small numbers of chips which are compatible with MCMs or other dense packages. Each of these partitions is termed a *cluster*, and at the end of Phase 2 each of the SRAM and DRAM levels were partitioned into 4096 clusters, as pictured in Figure 3.

There is one SRAM cluster directly coupled to each SPELL/CRAM complex, with 4 SRAM chips and a Vortex Interface chip. Four Vortex fibers bring data into the SRAM cluster, while one fiber may be driven with outgoing data. This imbalance in fibers reflects the expectation that there typically will be more incoming data to an SRAM complex than results leaving it. A Phase 3 activity will include investigating if the optical to electrical conversions for each of the four incoming fibers can be integrated onto the SRAM chips themselves. The savings in interconnect, power, and latency may be substantial.

The current DRAM cluster configuration consists of 8 DRAM chips, a fiber interface chip, and 32 HRAMs - 4 connected to each DRAM. Because of the cost of the fibers, only one pair of Vortex fibers connect to each such cluster, one for incoming data and one for outgoing. Clearly, if cost permits, more outgoing fibers would better match the SRAM bandwidths.

## 3. PIM Technology

As hinted above, the SRAM and DRAM chips of HTMT will perform a significant amount of processing and data management, independent of the SPELLs. Implementing this in an efficient way requires placing a significant



**Figure 3. HTMT Cluster Diagram**

amount of logic on the same die as the memory. While this is a common practice with SRAM designs (consider any microprocessor with dense caches), it is only recently that such technology has become available for DRAM.

*Processing-In-Memory (PIM)* VLSI technology (also known as *Intelligent RAM* [16], *embedded RAM*, or *merged logic and memory*) allows the fabrication of multiple dense DRAM memory macros interconnected to dense conventional logic, on a single chip. Each of the memory macros includes all necessary decoding, timing, refresh, and redundancy logic to allow it to function as a totally independent memory subsystem. Storage sizes can be selected in multiples of a few million bits, with current macros in the 2MB range not atypical. To date, random access times as seen by nearby logic have been in the sub 15 ns range,

less than 1/2 of the access time of a conventional DRAM chip as seen from off-chip. Further, each such access fetches up to several thousand bits, with several hundred of these presented to the outputs of the macro. For this paper we will term these several hundred bits a *wide-word*, and the several thousand bits latched internally as an *open row*. Very fast mechanisms allow external logic to select which wide-word from the currently open row is visible to the macro's outputs. Pipelined "page mode" rates as low as 5 ns per wide-word are currently possible, resulting in peak bandwidth rates per macro in the multiple GBps range. Both access time and transfer rates are projected to increase over the years. With multiple nodes on a chip, internal bandwidths can be enormous, and the HTMT

design points have used this to advantage to match the Vortex and other interconnect requirements.

The other half of PIM technology is the ability to fabricate millions of transistors of logic next to these memory macros. Until very recently, such logic was fabricated using the same transistors that were used in the memory arrays. While this provided logic density rivaling that on pure logic chips in the same feature size, the use of these higher threshold transistors did mean that logic clock rates were around 1/2 of their pure logic counterparts. Recently, however, the deep trench technology used to make the DRAM capacitor has been migrated to a pure logic process [11], giving us the best of very high speed CMOS logic with perhaps 60% of the storage density of a pure DRAM process. However, as much as this speed is desirable, the bulk of the typical HTMT DRAM chip will be memory, not logic, and sacrificing density for speed may not be the best design point. Consequently, for now the HTMT projections have assumed the slower but more dense version.

An HTMT project report [13] documents in more detail the technology projections used to make the PIM chip sizing projections for the final machine.

#### 4. PIM Node Architecture

Both the SRAM and DRAM chips are organized similarly. The bulk of the chip consists of multiple identical nodes, each of which contains a memory macro and local logic designed into a processing unit. Other regions of the chip support the high speed wire interconnect to a Vortex interface chip, other PIMs in the same cluster, and either the HRAMs (in the case of the DRAM PIMs) or the SPELL/CRAM (for the SRAM PIMs).

The processing logic is termed an ASAP (*At-the-Sense-Amps-Processor*), and is positioned literally at the digital outputs of the macro (see [5], [10], and [12]) for more discussion on ASAPs). The current HTMT ASAP configuration is designed to maximize the use of the hundreds of bits that are available at each wide-word access from the macro (256 for the current design point), and perform the local processing functions described below. The organization of an ASAP (Figure 4) consists of:

- several multiported wide-word register files,
- a linear array of function units (both fixed and floating) to perform computation on the wide words from the register file,
- status logic which allows each function unit to record its own conditions and conditionally participate in an operation,
- a permutation network which allows computable reorganization of data within a wide-word, and a variety of special purpose wide-word registers.

Some amount of SRAM and ROM may also be present to provide fast access to certain data structures such as routing and translation tables.

The multiple function units in an ASAP can operate in a variety of modes, starting with simple scalar combinations of pairs of “registers” taken as fields from the wide registers. Next are SIMD-like operations on wide-words from the register files. The status logic allows both conventional conditional participation and more elaborate cascading of conditional tests. A specialized wide register called the *local instruction register* (LIR) contains a sub-field for each ALU which provides an independent operation and status update specifier. Together, the contents of the LIR is termed a *wide ALU control instruction* (waci), and permits execution of a VLIW-like operation in a single instruction time. This is crucial for several of the PIM functions discussed below.

The wide register files are designed to support multi-threading at the hardware level. There is one global group of registers that can be shared among threads, and another for individual threads. For the latter there are 16 sets of 8 registers, called *thread register contexts* (TRC). Within a TRC, separate wide registers hold thread status (PSW, PC,.), scalar registers, an instruction buffer, and LIRs for

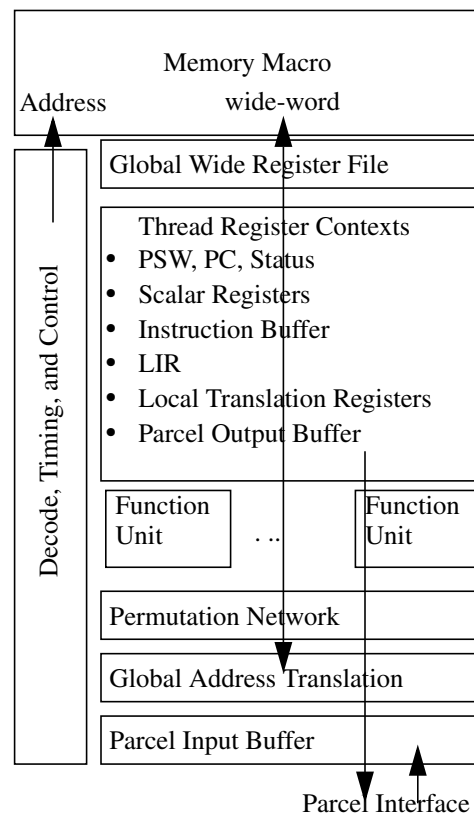


Figure 4. Node Organization and Floorplan

both wacis and permutation control.

The memory model assumed by the node ISA divides node memory into two classes of storage. A local area holds local stacks, heaps, persistent node control information, and code. It is not accessible outside of the node. A global area contains all data which is visible to the rest of the HTMT system. For simplicity, this area may be divided into 8 segments which may be mapped into any of up to 256 different system-wide address spaces. We term these segments *panes*, because unlike a conventional virtual memory page or segment, not only can the external logic “look in” to the memory, but logic behind the memory can “look out.”

The ISA of the node is a RISC-like combination of vector-vector, vector-scalar, scalar-scalar, and thread management. Options on many of these utilize the LIR, permitting multiple different operations to be performed to different parts of a wide register simultaneously.

There are two sets of loads and stores, one that access local data (32 bit addresses) and one for global data (64 bit addresses). The address translation mechanisms behind these are also subtly different. Local address translation is for relocation and protection; there are no hooks for off chip references. Global address translation, however, not only checks if a particular system datum is local on the node, but also, if not, serves as a source for a hint as to where in the system the data may be found. This latter hint can be used to generate the routing for a message-like communication (termed a *parcel*) to the appropriate destination node. One very important difference between this scenario and a conventional system is that in HTMT all memory is “smart,” i.e. it is capable of determining whether an incoming request is really for data that is present on the node, or if not, forwarding the message to somewhere where it can be handled. In the current HTMT design, this same set of global translation logic is also used to determine if the address accompanying an incoming parcel is in fact local.

The number and size of panes, and of potential address spaces, was chosen to allow all translation information to be stored within a small number of wide-words which can be swapped in and out quickly on a thread swap. Early simulations indicate that 8 such panes, coupled with relatively small number of such TLB-like entries may very well be highly efficient for typical node programs [15].

In terms of current sizing projections, it is estimated that in 2005 a DRAM node with an ASAP as defined above and 32 MB of RAM would occupy about 40 mm<sup>2</sup> of 0.01 micron silicon, about the same as a 4 MB SRAM node.

## 5. The Execution Model

There are two components to the execution model presumed for HTMT application programs. First is a *percolation model* where data is transported up and down the memory hierarchy independently of the SPELL programs, so that ideally the SPELLs never have to leave the cryo regions to find their data. Second is the communication mechanism by which the data transfers required by the percolation model are actually initiated. Nearly all of the functionality to support this is provided by the PIMs, including:

- initializing data structures and perform simple reorderings on memory objects that would be too inefficient to perform in the SPELLs (such as scale a matrix, or perform a transpose),
- "gather/scatter" indexing operations to transfer data segments between memory levels,
- computations on data flowing between the HRAM, the I/O, and the memory, such as compression/decompression, spectral transforms, or ECC/redundancy computations,
- task synchronization activities “in the memory” in response to SPELL instructions, and
- active “closure pushing” where the PIM memories will on their own assemble all data and code needed by a SPELL to do some work, push the composite down to the SPELLs, manage CRAM storage allocation, signal the SPELL of waiting work, and extract the results.

In a very real sense, this functionality results in a model of execution for the HTMT where an application program's execution is split in half—the data management parts executed by code in the PIMs, and the bulk of the computation intensive operations performed in the SPELLs. Likewise the traditional functions of a computer system's “runtime,” such as task scheduling and temporary memory management, migrate into the SRAM PIMs, while more global “operating system” functions such as I/O, file systems, and application control fall naturally to the DRAM levels. All of these functions were considered during the design of the PIMs.

### 5.1. The Percolation Model

The percolation model used in HTMT has two major objectives. First, it tries to avoid latency delays by very aggressive and programmable “prefetching” of data. Second, it addresses the relative lack of storage at the lower levels of the memory hierarchy near the SPELL by partitioning the application programs into relatively small and self-contained fragments, each of which represents one or more threads of execution derived from the original program. Data flow-like synchronization mechanisms are used

to indicate when such threads should be initialized, and once initialized, run to completion on a SPELL.

When mapped onto the PIM layers, this approach results in some fairly specific execution characteristics, as pictured in Figure 5. First, there is a pool of buffers, which when filled with data are called contexts here, in both the SRAMs associated with a SPELL and the SPELL's CRAM. Each such context matches to execution of one of the SPELL threads mentioned above.

Both the CRAM and SRAM buffers are managed by a combination of the SRAM's runtime (*PIMOS*) and code derived from the application program. The actual process of gathering data into, or scattering data out of, these contexts is managed by yet other threads that run initially in the SRAM, are sent to the DRAM via parcels, and execute as threads there.

## 5.2. Parcel Communication Mechanism

To communicate between the nodes on each PIM, the concept of a parcel (*PARAllel Communication Element*) was developed as a descendant of a combination of active messages [9] and object-oriented architectures such as the J-machine [6]. A parcel is addressed to an object in memory, consists of an address space identifier, a command, and arguments/data (up to several thousand bits), and corresponds to a single Data Vortex packet. Three kinds of parcel-triggered functions have been identified: traditional "dumb" memory operations, SPELL parcels to implement some special SPELL support functions, and more sophisticated application-level parcels. Examples are given below.

In terms of comparisons to more traditional paradigms, parcels are different from messages in a conventional message passing environment because they are addressed to an object, not to a process or a node, and are interpreted more

like a continuation than a simple data transfer.

Also, unlike an active message, a parcel does not need a program-specified target node - this is derived from the target object address by the global translation mechanism outlined earlier. Also unlike many active message definitions, the synchronization mechanism between the thread triggered by the arrival of a parcel at a node and an ongoing thread at the node may be much richer than a simple semaphore. For example, consider the synchronization needed to detect when all the data needed for a context in Figure 5 have arrival. As each parcel arrives, the thread it initializes may both store the data in the parcel in the designated region of the target data object and update a count of the number of datums so far received. When this count reaches zero, a thread suspended for this event may be restarted to transfer the context in bulk to the matching CRAM buffer and start the SPELL. If organized properly, a single wide-word read, coupled with a wide ALU control word that updates several fields in different fashions concurrently, can perform the above functions in a very few number of cycles. Given that each SRAM cycle is liable to be on the order of a hundred SPELL cycles, minimizing this time is important.

Given that objects in HTMT are likely to be huge in relation to what can be stored on a single PIM node (TBs in size vs. a few MB), when a parcel is sent to an object, it usually goes to a node (termed a microserver - see [5]) holding code to describe and manage that object. The command in the parcel is then treated akin to a method name: an index to code for an access function defined for that object. This code uses local object description information and parameters from the parcel's arguments to determine exactly where the information needed by the parcel resides, and to relaunch, if necessary, the parcel to the correct data node. This may happen recursively. An example of where this might be of value is in managing a TB-sized object

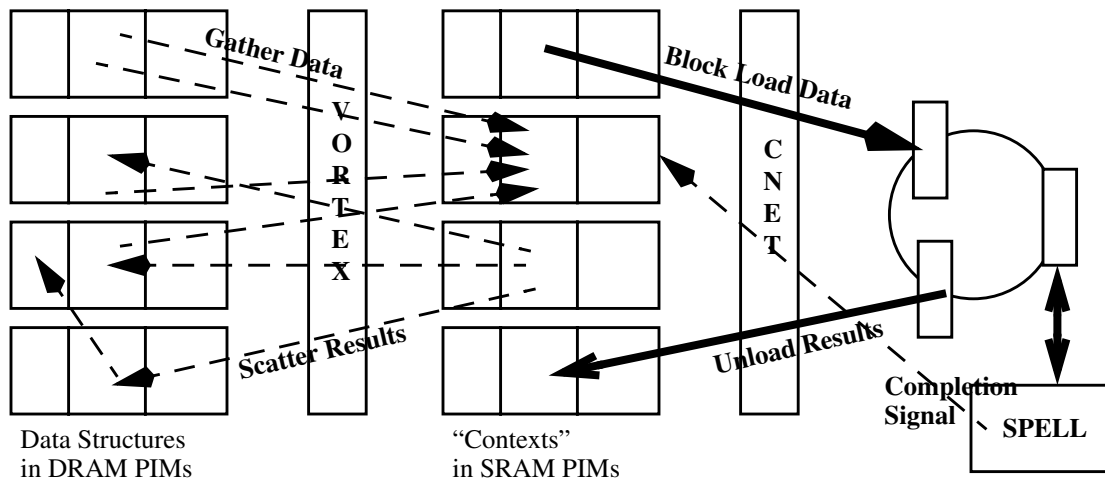


Figure 5. Percolation Buffer Flow Diagram

where the data may consume hundreds to thousands of nodes of memory. The object's home microserver may be coded so that its access method determines which other microserver(s) hold further information on where the specific data being requested really is. Using something like an extended quad tree may thus cause a single parcel request for a row, for example, to be broken in many concurrent parcels, each of which returns part of the requested row - all without the source of the original parcel having the slightest clue as to the current data layout across nodes.

### 5.3. The PIM OS

An array of PIM chips creates a very powerful computer in its own right; the current projections are that both the SRAM and DRAM layers have very nearly a peak petaflop of performance by themselves. But this configuration is quite unconventional. Foremost, the PIM array is at heart a memory part. Even though it can be a very powerful computer, it must not sacrifice memory performance for computation. The runtime that manages each node (*PIMOS*) must support PIM arrays that are stand-alone "distributed, embedded" computers, along with those that are intelligent memory "accelerators." The PIMOS is also the lowest level of software executing on a multi-chip PIM system. It creates a single abstract machine out of the many PIM nodes. It is a "co-operating system" in both senses of the word. First, it creates a single (cooperating) image of the machine. Second, it is made up of individual operating systems that are in most respects equals.

The uniqueness of the PIM architecture and environment have given rise to the following initial design objectives for the PIMOS. First, the PIMOS is event-driven. A memory is a "reactive" part--it responds to asynchronous events. Parcels carry asynchronous events between memories. Second, the PIMOS is memory-oriented. The processors are associated with the memories, rather than the other (traditional) way around. Messages are sent to objects in memory, not to processes on processors. Computation is performed on data in memory, not in a process on a processor. This is more than a semantic distinction. It is a fundamentally different way to view computation.

The third objective in the PIMOS is scalability. HTMT could contain upwards of one-half million PIM nodes. A logarithmic algorithm in such a system has nearly 20 steps. It is imperative that the PIMOS be scalable, which in this context means that all decisions can be made locally. There are no global decisions or agreements among memories. (There can be global information, though.) Therefore, local decisions are based on rules that create the desired "globally emergent" behavior of the system.

Lastly, the PIMOS must provide a single, unified machine. The memories and processors are not explicitly programmed, as in "put this on PIM 6," or "send M to PIM 12." Rather, memory objects are created in the PIM array. Messages are given to the object ID, which is essentially a virtual address, and when PIM hardware cannot find the object locally, the PIMOS must manage the process of finding it, where-ever it is in memory. This may very well include managing whatever coherency and consistency policies are appropriate for that object.

## 6. Summary and Future Work

This paper has summarized the architecture of HTMT, with emphasis on the memory hierarchy. By integrating a full suite of emerging technologies, HTMT has forced a major re-examination of the way high performance computers should be built. This is particularly true in the memory area, where the introduction of PIM technology has radically changed the semantics of memory and the models used to invoke memory operations. Although nothing from the work described here has as yet been built, we have reached a point where some good engineering can fully explore the design space. Much of this work will be performed in the current phase of HTMT, including:

- refinement of the PIM runtimes, both in support of the percolation model and in overall management of the storage part of HTMT.
- extension of the parcel concept to become an entire continuation, including code.
- development of a functional simulator which will allow prototyping of the above runtime functions in an environment where data transfers from typical applications can be modelled.
- further refinement of the PIM ISA and more detailed design of the ASAP logic.
- explorations of alternative memory macro designs which bury additional PIM-relevant features within them, such as use of multiple open rows as free "cache lines" [20].
- investigation of key applications which may tend to be highly PIM resident, with a minimum of SPELL interaction needed.
- development of programming models which can extract from an application the code needed to be placed on the PIMs, along with generation of the parcels necessary to manage them.

We fully expect to back up the design choices described here with much more detailed simulation in the coming year.

## 7. Acknowledgements

This work was performed in part for the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the Defense Advanced Research Projects Agency (DARPA) and the National Security Agency (NSA) through an agreement with the National Aeronautics and Space Administration.

Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government, the (name of the contractor), or the Jet Propulsion Laboratory, California Institute of Technology.

## 8. References

- [1] -, HTMT project web site, <http://htmt.cacr.caltech.edu>
- [2] -, 1996 Symposium on the Frontiers of Massively Parallel Computation, Annapolis, MD, Oct. 27-31, 1996.
- [3] -, "IBM Unveils \$100 million research initiative to build world's fastest supercomputer," [www.ibm.com/news/1999/12/06.phtml](http://www.ibm.com/news/1999/12/06.phtml), Dec. 1999.
- [4] M. Arend, C. Reed, K. Bergman, "Physical Design and Specifications for the Data Vortex Network," HTMT Tech Note 33, Cal Tech (see the HTMT web site).
- [5] J. Brockman, P. Kogge, V. Freeh, S. Kuntz, T. Sterling. "Microservers: A New Memory Semantics for Massively Parallel Computing", In *Proc. of the ACM International Conference on Supercomputing*, June, 1999, pp. 454-463.
- [6] W. Dally, et al, "The Message Driven Processor: A Multicomputer Processing Node with Efficient Mechanisms," *IEEE Micro*, April 1992, pp. 23-39.
- [7] W. Dally, et al, "High Performance Electrical Signalling," MPPOI'98.
- [8] M. Dorojevets, P. Bunyk, D. Zinoviev, and K. Likharev, "Superconductor Electronic Device for Petaflops Computing," accepted to FED Journal.
- [9] T. von Eicken, D. Culler, S. C. Goldstein, and K. Schauer, "Active Messages: a Mechanism for Integrated Communication and Computation", In *Proc. of the 19th International Symposium on Computer Architecture*, May 1992.
- [10] M. Hall, P. Kogge, J. Koller, P. Diniz, J. Chame, J. Draper, J. LaCoss, J. Granacki, A. Srivastava, W. Athas, J. Brockman, V. Freeh, J Park, J Shin. "Mapping Irregular Applications to DIVA, A PIM-based Data-Intensive Architecture," Supercomputing, Portland, OR, Nov. 1999.
- [11] IBM Microelectronics, "IBM chip advances spur "system-on-a-chip" products," [www.chips.ibm.com/news/1999/sa27e/](http://www.chips.ibm.com/news/1999/sa27e/), Feb. 22, 1999.
- [12] P. Kogge, J.B. Brockman, V. Freeh, "Processing-In-Memory Based Systems: Performance Evaluation Considerations", Workshop on Performance Analysis and its Impact on Design, held in conjunction with ISCA '98, June 27-28, 1998.
- [13] P. Kogge. *PIM Technology Projections for the HTMT Project*, CSE Dept. Technical Report, Univ. of Notre Dame, Sept. 13, 1999.
- [14] P. Kogge, et al. *Final Report: PIM Architecture Design and Supporting Trade Studies for the HTMT Project*, CSE Dept. Technical Report, Univ. of Notre Dame, Sept. 13, 1999.
- [15] R. Murphy, *Design Parameters for Distributed PIM Memory Systems*, M.S. Thesis, CSE Dept., Univ. of Notre Dame, April 2000.
- [16] D. Patterson et al., "A Case for Intelligent DRAM: IRAM," *IEEE Micro*, April 1997.
- [17] Semiconductor Industry Association, *International Technology Roadmap for Semiconductors - 1998 Update*, <http://notes.sematech.org/ntrs/Rdmpmem.nsf>.
- [18] T. Sterling, P. Messina, and P. H. Smith, *Enabling Technologies for Petaflops Computing*, MIT Press, Cambridge, MA, 1995.
- [19] T. Sterling, L. Bergman. "A Design Analysis of a Hybrid Technology Multithreaded Architecture for Petaflops Scale Computation," Int. Conf. on Supercomputing, Rhodes, Greece, June 20-25, 1999.
- [20] J. Zawodny, P. Kogge, J. Brockman, E. Johnson, "Cache-In-Memory: A Lower Power Alternative," Workshop on Power-Driven Microarchitecture, ISCA., Barcelona, Spain, June 27-28, 1998.