Broad Agency Announcement 97-46

Intelligent Memories

Active Database Technology

Technical Contact:

Thomas F. Knight, Jr.
Phone: + 1 617 253 7807
Email: tk@ai.mit.edu
Fax: +1 617 258 7413

Howard E. Shrobe
Phone: + 1 617 253 7877
Email: hes@ai.mit.edu

M.I.T.
Artificial Intelligence Laboratory
545 Technology Square
Cambridge, MA 02139

Administrative Contact:

Ms. Jeanne DePass
Phone: +1 617 253 3866
Email: jdepass@mit.edu
Fax: + 1 617 253 4734

M.I.T.
Office of Sponsored Programs
Room E19-750
77 Massachusetts Avenue
Cambridge, MA 02139

	Total Cost	Year 1	Year 2	Year 3
Base	$1,723,244	$558,929	$574,261	$590,054
Option	$14,046,037	$4,706,629	$4,629,586	$4,709,822
Total	$15,769,281	$5,265,558	$5,203,847	$5,299,876

“Other Educational”

Active Database Technology

Section II: A. Innovative Claims

Despite rapid improvements in the underlying technology, our systems are still handcuffed by architectures informed by yesterday's constraints. Systems architectures necessary for both crucial applications and important scientific research are unavailable, not because they are impossible, but rather because we have lacked the necessary imagination.

Fluid access to massive amounts of richly structured information is crucial both for a class of DOD applications such as the Dynamic Database, and for important research into the structure of human perceptual and cognitive processing. These are not unrelated goals, since the Dynamic Database requires both perceptual (image processing) and cognitive (image understanding, target recognition) similar to those of humans and animals.

This proposal develops a software and hardware infrastructure capable of handling such rich data at rates three orders of magnitude faster than current technology, while simultaneously addressing software and system organizational issues with modern data representations, memory management, transactional techniques, software layering, and object-oriented approaches.

It is hard to overstate the dramatic effect such processing power could bring to bear on complex problems. Current generation processors find it difficult to handle real-time video, often reducing resolution or frame rate to allow processing. With terabaud memory bandwidth, video data rates become an almost insignificant processing burden.

From a hardware perspective, the enabling technology for this revolution is the integration of dense DRAM with high performance custom logic. The terabaud access to memory enabled by this approach allows construction of commodity reconfigurable processing elements capable of performing parallel signal processing, search, retrieval, and simulation at unprecedented rates. We need no longer manipulate data one piece at a time, but can process it in bulk.

An important strategic approach of our research is to construct a *commodity* component and system: one which will be mass produced, thus riding the commodity—rather than the custom—price, performance, and availability curves. The use of block reconfigurable logic allows the hardware components to be malleable and general purpose, while retaining performance for important military and commercial applications.

Critical software technology includes sophisticated data representations, allowing automatic type analysis and parallel pointer manipulation entirely within the memory system. Bulk data handling approaches, including streams and array-language features, support simple signal-processing tasks. More sophisticated pointer manipulation, complex database retrievals, and automatic garbage collection require excellent interconnection and message delivery mechanisms, and support more complex data accessing mechanisms such as scatter/gather and sparse matrices.

Novel software organizational paradigms, such as transactional execution of subroutines, enforced layering of software structures, a careful analysis of program state, and the role of reversibility in program execution will allow us to deliver the raw hardware performance in a usable form.

B. Technical Rationale

Motivation

Modern computer architectures have been built to optimize SPECmarks rather than to optimize fluid access to massive amounts of richly structured information. Such information access is crucial both for a class of DOD applications such as the Dynamic Database and for important research into the structure of human perceptual and cognitive processing. These two goals are not unrelated, since the Dynamic Database involves both perceptual processing (image processing) and cognitive processing (image understanding,target recognition) similar to those of humans and animals.

The Dynamic Database has the aspiration to maintain a real-time situational and threat assessment over an entire theater of operation. Rough estimates suggest that such a system must manage a representation containing roughly a Petabyte of data which is updated constantly in real time. To maintain the situational and threat assessments, signals from a variety of sensors must be passed through a pipeline of image processing and image understanding systems. These are in turn informed by a high-performance knowledge base with both visual models of potential targets and more symbolic models of military strategy and tactics. Integration of high-level knowledge from auxiliary sources such as newswires, sigint, and commercial broadcast adds another dimension.

Commercial entities, including banks, supermarkets, airlines, and stockbrokers, are already completely dependent upon complicated, high-end systems to manage ever-growing databases of customer information, inventory, reservations, wire transfers, etc. Many are still operating on legacy systems, and even current systems technology is rapidly being outpaced by corporate demands for faster processing of more information. As data-mining systems are increasingly deployed to discover consumer trends, spot illegal transactions, and set flight paths, frequencies, and schedules, the move toward SMP systems is only a stop-gap measure—there is a desperate need for systems that are simply not throttled by the Von Neumann bottleneck between processor and memory.

Scientific motivations also lead us to architect systems capable of processing enormous quantities of information. Intelligent system design requires the building of systems that construct psychologically and physiologically realistic models of human cognitive processes. Such models are not mere scientific curiosities but are significant enablers for the next generation of systems such as the DDB and autonomous robots. Humans (and for that matter most animals) can actively perceive and model their environments with accuracies far above those of our best computational approaches.

Psychological and physiological evidence suggests that the human memory system is actually composed of several semi-independent subsystems such as visual memory (for perception and recognition), episodic memory (memory of event sequences supporting case based reasoning), and semantic memory (supporting taxonomic characterization). Each of these is massive in size and supports real-time processing in spite of the overall slowness of the neural system. For example, visual memory in humans is capable of supporting real-time recognition of any of thousands of specific objects and classification of any of thousands of categories of objects even though the objects are presented in arbitrary poses, with arbitrary lighting and occlusions. Similar impressive performance is observed in each of the other human memory systems. Although AI researchers have developed a computational understanding of many of the processing algorithms used in these tasks, all implementations are currently stymied by their inability to manage and process retrievals in memories comparable in scale to those of humans. The next set of breakthroughs will be enabled by systems that allow us to rapidly build, experiment with, and modify systems that manage realistic scale memories.

Such systems are characterized by:

- Multiple representations optimized for multiple uses (e.g. visual models for target id, some geographic models for map oriented browsing, topological models for road and transportation systems, and still others more symbolic representations of force on force tactics and strategy).
- Object Orientation
- Irregular Data Structures interconnected by diverse pointer structures
- Massive Scale

Each of these characteristics is poorly supported by today's computer architectures.

It is time to bring our architectural, system design, and language technology up to date, to address this critical class of problems. Novel architectures will enable dramatic performance improvements in processing and integrating knowledge in DOD application areas such as SAR, image understanding, target recognition, sonar, and simulation, as well as in a range of important commercial applications such as virtual reality, medical image processing, games, physical simulation, CAD tools, and rasterization engines. Success will enable important research in the development of intelligent, reactive computing systems.

A combined approach, tackling the system organization, language design, algorithm development, data representations, and hardware architecture is required.

Key Design Issues

There is a downside to the rapid progress of computational technology we have been enjoying for the last decade: The good news is that processors are getting faster and memories are getting bigger; the bad news is that these memories do not keep pace with processor speed. Moreover, the bandwidth between components does not keep pace with the volume of data managed. Unless caches can be kept filled, the extra processor speed is useless. Bandwidth between memory and processor in today's designs must be husbanded carefully. In Data Intensive computing systems cache misses in the lower levels of the memory hierarchy loom large, since the data set will not fit in primary memory and the servicing of page misses involves disk and network accesses with essentially infinite latency.

Current architectures recognize this problem and try to deal with it in the case of regular data that can be streamed into the processor in an essentially linear fashion. Thus, SPECmarks continue to improve. However, database system performance has not improved at a similar rate, and object-oriented software systems have fared even worse. Such systems spend much of their time chasing object references and are poorly served by caches, since a referenced object is unlikely to be adjacent to the referencing pointer.

We believe that there are techniques that can be applied to this problem, and that our combined experiences in building architectures, compilers and run-time systems for advanced Dynamic Object-Oriented computer languages provide unique leverage in understanding how to proceed.

Six key ideas drive our design:

- **DRAM/ASIC Technology** Today's VLSI technology allows us to integrate onto a single die conventional (fixed) processing, large memory arrays, interconnection networks, and configurable processing components.
- **Active Memory Interface** On-chip packaging of the memory enables the processor to memory interface to be fundamentally redefined: Instead of sending an address and receiving back an individual datum, the control processor passes a data referencing program and receives back the *stream* of data defined by that program.
- **Manifest Programming Language Structure** Programming languages which express object-oriented computations in the most natural manner, also enable compilers to analyze the computation and to decompose it into a memory intensive component which traverses a complex structure object and a processor intensive component which transforms the information found while traversing the structure.
- **Tagged Data** Data is tagged so that pointers can be distinguished from immediate data and so that Memory Management algorithms can dynamically guarantee locality of reference over computation episodes of modest duration.
- **Read/Write Barriers** Relatively cheap hardware enables these management algorithms to function in real-time by erecting the read and write barriers needed by memory management algorithms.
- **Block Reconfigurable Logic** The reconfigurable logic on-chip can be dynamically reconfigured into elements which at certain times apply local computation to local data (e.g. image filtering operations on a pixel array) and at other times into optimal interfaces with other components of the system.

We next briefly review these insights:

DRAM/ASIC Technology
Accessing data is the most challenging problem in modern computer hardware design. At every level, from the on-chip wiring layers, to the package pins, to the cache design, to the main memory size, to the disk access path, to the network, the latency and bandwidth of the communication is a far more difficult challenge than the actual processing of data. Latency requirements for the kind of sophisticated pointer based manipulations inherent in object-oriented databases place even more extreme requirements on the memory system. Such systems can take little

advantage of bringing large adjacent blocks of data into the processor, since most items are single-address pointer fields, requiring random references.

Modern architectures such as the Digital Alpha are limited primarily by the latency and bandwidth of main-memory accesses, requiring a latency equivalent to the execution time of perhaps 100 instructions, and a bandwidth of about 1 Gigabaud. Even off-chip cache hits provide only 128 bits every two processor cycles, for a cache-access bandwidth of about 20 Gigabaud.

The development of integrated DRAM plus logic components by Toshiba, IBM, and Mitsubishi, among others, promises to dramatically decrease latencies and increase bandwidths for memory access. The Toshiba technology, for example, provides DRAM macros (up to 128 Mbits/die) which are capable of delivering single chip bandwidths of 1 Terabaud between the DRAM array and customized on-chip logic. This is achieved by clocking an extremely wide data path (6,000 bits) at a relatively modest (6ns) clock rate. Latency is approximately 8 cycles (48 ns), also substantially better than off chip memory accesses.

For the first time, this technology provides a means of accessing substantial blocks of memory at data rates which no longer limit our maximum rate of processing. It is instructive to compare this data rate, latency, and capacity with some other parallel architectures. The Thinking Machines CM-2 SIMD processor, for example, had, in its full configuration, 64,000 single bit processors clocked at 8MHz, achieving total bandwidths of 1.5 Terabaud—about the bandwidth achieved in a single die with this technology.

The combination of a factor of 1,000 improvement in memory bandwidth, together with the cost reduction inherent in single-die implementation technology, enables a radical rethinking of processor and memory system architectures.

A hard-learned lesson of the past decade is that, to be successful, a hardware product must achieve commodity status. Products are only successful when produced in volume, by many competitive manufacturers, and with technologically up-to-date processes and designs. The dilemma for computer architects is the absolute requirement to simultaneously radically innovate and to also build with commodity components.

Our approach to this difficulty is to propose a new kind of component—one which we believe will achieve commodity status. The new ecological niche opened by the DRAM-ASIC technology can be occupied by a component which incorporates a large DRAM array, along with high-performance reconfigurable logic arrays. By allowing different applications to configure the logic to their application, we can produce a single component with widespread application.

Specifically, the component we envision will incorporate the following four elements:

- 64Mbit or larger DRAM array with a 2K-wide data-access path and 6ns data clock
- Simple risc processor (e.g. 64 bit wide Strongarm-like processor) with simple pipelining and easy trapping mechanisms
- Interface to (and possibly the implementation of) a high-performance interconnection network
- block-reconfigurable FPGA logic and interconnect

Active-Memory Interface

We must redefine the processor-to-memory interface. A typical object-oriented program consists of operations which start at a set of root nodes, follow specific object references within these objects and then recursively follow those fields of the referenced objects. Data found within the traversed structures are combined, transformed and finally written back. Since there is ample processing near the individual memory elements and very high bandwidth to the memory, it makes sense to bring the traversal of object-oriented structures into the memory system itself, reducing the delay involved in fetching an object and issuing the next address. When the referenced object is non-local the computation is swarmed out to the relevant components, using the high-performance interconnection switch.

Manifest Programming-Language Structure

Dynamic Object-Oriented Languages (e.g. Lisp/Dylan) provide the most natural vehicle for expressing Object-Oriented computations. DARPA/ITO's EDCS program has recognized the importance of such languages and has sponsored the development of new compilation and run-time technology for such languages. The naturalness of these dynamic, higher-order languages for such computations come from their ability to directly express computational patterns. In particular, patterns such as traversing object graphs are directly expressible as a higher-order procedure such as:

```
(define (traverse roots child-producer node-extractor data-action)
  (let ((worklist roots))
    (loop for node = (pop worklist)
          do (set! worklist (append (child-producer node) queue))
             (data-action (node-extractor node))
          until (null? worklist))))
```

Such a high-order procedure is passed a set of roots, a procedure for moving from one node to the next (the child-producer) and a procedure for performing an action at each node. A typical child-producer procedure simply collects a set of object references from each node:

```
(define (tree-traverser node) (list (left node) (right node)))
```

An expression such as `(traverse (list a b) tree-traverser foo my-action)` then expresses a classic binary-tree traversal algorithm, applying `my-action` to the `foo` component of each object. This expression of the computation makes transparent to the compiler exactly the elements we want to be transparent; partial evaluation of this expression with respect to the child-producer procedure directly expresses the traversal component of the computation independent of the action to be performed at the node. It is this component that will be passed to the memory system, resulting in a return stream of the `foo`-components of each object being fed to the processors (fig. 1). Of course, there may be execution-order constraints which result from the action procedure. We believe that extensions of core compilation technology can be developed which will analyze the constraints on execution order of the traversal algorithm. Further, we believe such procedures can ultimately be cast directly into the form of reconfigurable logic pipelines.

Figure 1: Polymorphic graph-search method is parameterized by the action performed on the graph nodes and the links traced through the graph. Partial evaluation of a particular application produces a “residual” computation that can be farmed out to the memory system, mapping a sparse data structure into a dense value stream.

Tagged Data

The type of all data should be manifest; in particular, pointers should be distinguishable from non-pointers. The minimum tagging necessary to support this is obviously 1-bit. In practice, a wider tag has been found to be useful. Tagging is useful for run-time checking and dispatching, which are crucial for Dynamic Object-Oriented systems. However, of greater importance to data intensive applications is the support that tagging provides to garbage collection and other memory-management algorithms. Garbage collection is the process of scanning memory to isolate live from dead data and to make the space occupied by the dead data reusable. In the context of data-intensive systems, it is even more important that garbage-collection algorithms also produce compact memory, dramatically improving locality of reference. The seminal work on these algorithms was done in our laboratory (Minsky, Steele, Baker, Bishop) and important extensions were done at Symbolics (Moon), a commercial spinoff from our laboratory.

Most of the current algorithms for compacting garbage collection use a semi-space framework, in which good data is copied from an old space to a new space. The algorithm begins by copying a set of root nodes (e.g. the contents of the register set) to newspace, leaving behind forwarding pointers, carrying special data tags. The contents of new space are then regarded as a queue: each unprocessed word is successively pulled from the queue, and the data which it references is added to the back of the queue; the reference is adjusted to point to the newspace copy. When the referenced data has already been relocated to newspace (detectable by the forwarding pointer tag on the oldspace copy), then it need not be relocated again, and the reference is simply adjusted to the newspace copy. When the queue is empty, all good data has been moved from oldspace leaving all of oldspace available for reuse. Newspace, furthermore, has been constructed by following referencing patterns, thereby increasing locality. Subtle variation in the queueing discipline can lead to greater improvements in the compactness of the current working set within the larger live set.

Compactness of the working set is crucial for data-intensive systems because it allows the computation to be moved closer to its data. However, monolithic scans of a petabyte data set are obviously intractable. Variants of the above algorithm have been developed in which the memory is broken into sub-areas which can be collected independently, thereby making the process tractable for very large memory systems. Additionally, parallel versions of the algorithms are well known, which are particularly attractive for a system such as the one we propose—systems that are composed from processor-memory building blocks.

Another important variant includes the idea of *generational garbage collection* in which we capitalize on the observation that most garbage is thrown away a short time after its creation. Thus, it makes sense to collect garbage more frequently from newly allocated data. Separating the collection process by generations assists in compacting and quickly reclaiming most garbage.

Read/Write Barriers

Two additional constraints on garbage collection algorithms are that they be able to function in a real-time environment and that they be able to reorganize sub-areas of the memory independently. During the lifetime of the MIT Lisp Machine and its descendant systems, we developed several low-overhead hardware enablers for these two goals. These depend on the erection of read and write barriers.

Real-time and parallel execution of the copying compaction algorithms depend on the erection of a read-barrier, which prevents pointers to oldspace to be read into processor registers; instead the processor is trapped, the oldspace datum is transported to newspace if necessary and the reference forwarded, and then the processor is resumed. To make this cheap, data tagging hardware is provided which examines the datatype tag on the word being loaded and signals that the tag indicates a pointer data type while a processor register contains an old-space vs. new-space flag for each chunk of memory. The trap signal is simply the logical and of these two signals. The trap-relocate cycle can be given a precise overhead bound, thereby allowing real-time processing.

Area-based collection is based on the erection of a write barrier which is used to detect references between memory areas. The write barrier is created by trapping the processor anytime a reference is written to memory and the reference refers to another section of memory. A particular version of this was developed in the Lisp Machine in which certain areas of memory were designated as the areas in which new storage would be created; most of this storage is likely to be short-lived. However, the part of it which does have a longer lifetime is also likely to be the active working set; it is therefore in these areas where compaction and collection have the highest value.

To enable compaction effort to be directed to the ephemeral areas, a processor register marks for each chunk of memory whether or not it is ephemeral. When the processor writes a reference to ephemeral memory into a page which isn't ephemeral, that page is then marked as needing scanning. Such marked pages contain all external references to the ephemeral areas, and are therefore scanned at the beginning of the memory reorganization process. The data referenced by the cross-area references found in these pages are then used as the root nodes of the compaction algorithm.

Ephemeral collection can provide order-of-magnitude speedups in the efficiency of garbage collection. However, the technique has an associated price: on stock hardware, the required write barrier must be implemented either in software or by the MMU with page-table granularity. Both techniques slow down memory writes by one to two orders of magnitude; this has rendered these techniques impractical for a wide class of systems and scientific applications.

Thus, we believe that hardware support for read and write barriers is an architectural feature that provides large payoff for small cost (the chip area dedicated to this logic in the Ivory processor was negligible; tags in the Ivory processor occupied 1 extra byte for each word, a 25% memory overhead for a 32 bit machine and a 12.5% overhead for a 64 bit machine). However, in a 64 bit machine, it is possible to use most of the bits used to encode the exponent of an IEEE floating point number for tagging, so that the actual overhead can be close to zero. We have designs for read/write barriers and general meta-bit support for memory data that will still allow processors to be used with commodity memory systems and communications busses. These novel designs address a long-standing barrier to widespread use of tagged architectures.

We anticipate that logic such as the read/write barriers can be implemented largely or entirely within the reconfigurable logic array on the die, allowing us to experiment with the design of these features over time, while allowing high performance. Moreover, in keeping with the commodity component notion, these features need not confuse more conventional application users of the component.

Block Reconfigurable Logic
A key aspect of our approach is the use of reconfigurable logic as an implementation medium. Two of our previous experimental designs inform our work The DPGA component was a relatively straightforward FPGA design, using conventional small lookup tables and simple gates. Its distinguishing feature was an ability to store multiple configuration personalities, allowing the update of one personality while performing computation with others.

The Matrix component investigated a radically different portion of the design space. Matrix consisted of an array of relatively complex bit sliced arithmetic engines, each containing a multiport register file, ALU, multiplier, and flexible interconnect. The essential idea was that much of the overhead of conventional FPGA technology resulted from attempting to reconfigure at too low a level. Efficient implementations could not be achieved by constructing multiport register files, e.g., from simple lookup tables. By implementing needed logic at a much higher conceptual level, Matrix allowed the implementation of systolic pipelined architectures, as well as SIMD, MIMD, VLIW processor arrays, with performance similar to that achieved by fully customized logic.

We believe that ideas from both the DPGA and Matrix components will form important aspects of the reconfigurable resources of our design. Intimately tying the reconfigurable logic array into the data paths, exception handling, and control mechanisms of the processor enables the compilation of customized "instructions" which perform complex operations, such as parallel vector operations on an entire 72,000 bit memory row; or alternatively (perhaps additionally) to configure an inner loop as a sequence of pipelined operations.

A major technical challenge is to perform this grafting of reconfigurable logic into a conventional processor design without significant performance reductions. We believe that this can be done, because the performance limiting factor in modern integrated circuit processes is largely wire delay, not gate complexity.

Interconnection Technology
An essential component of this active memory is the ability of the memory components to communicate among themselves to pass pointer based messages. This requires a communications router filling the same role as the router in the CM-2 processor. Our previous work on the Cross-Omega Connection Machine, and subsequent work on the Transit/Metro router component led to the development of extremely low latency connection based communications protocols. The central idea of these protocols is the establishment of a half-duplex connection between sender and receiver. Unlike packet based approaches, we construct end-to-end connections, much like the telephone network establishes communications paths.

The advantages of connection, rather than packet, based communications centers around the lack of required buffering, the speed of cut through routing, good fault tolerance, and the ability to acknowledge receipt of messages through turning the channel communication direction around. In combination with the Multibutterfly topology [Maggs, Leighton & Knight], these protocols guarantee near-lossless message routing in a short-haul environment.

Another aspect of our work on high performance routing centered on high speed electrical interconnection technology, producing the first power efficient high speed transmission line drivers [DeHon, Simon, & Knight].

We will use these routing protocols, topologies, and circuit techniques in constructing a fault tolerant high performance interconnection network which provides both processor to memory access, and needed communications between the active memory components themselves.

An aspect of the design still under consideration is the portion of the network interface logic which can be efficiently implemented within the reconfigurable array. Our hope is that details of the routing protocol need not be architecturally specified early in the design process, but, rather, can be experimented with, modified, and updated through the use of reconfigurable resources.

Software technology
Raw bandwidth and hardware-processing capability, of course, is only part of the equation. To deliver useful processing, the impressive processing available must be matched by a careful consideration of the system organization, data structures, and programming language design. This proposal is as much about the methods of organizing this computational power as it is about its creation.

Our starting point is the set of dynamic computer languages, such as Lisp, Scheme, Smalltalk, Dylan, ML, and (to a lesser extent) Java. We choose these languages as a jumping-off point because of our intellectual heritage, our system requirements, and our strong belief that they represent the best in the current state of the art in language design. To be sure, there are defects and omissions in all of these languages; nonetheless, we believe it is easier to modify a language such as Scheme to include new capabilities than many of the more conventional alternatives. Moreover, once we have a solid substrate within such a language, past experience indicates that we can implement more conventional languages on top of such an infrastructure easily. One notable language that will be an early such target is Matlab, due to its popularity in developing signal-processing applications.

The software research we propose is targeted to several short-term requirements, and to a set of longer range, but important, more speculative ideas.

Bulk Data Handling
In the short term, we will add support for primitive operations for handling bulk data. Such operators already exist in languages such as APL and Matlab; we anticipate that these operators can be added quite simply to Scheme. By making these parallel operations explicit, compilation to hardware parallel execution is dramatically simplified. In particular, we envision configuring the reconfigurable arithmetic units into pipelined execution units. Such pipes can, much like the systolic architectures of the early '80s, flow data through them, requiring only that input data be provided, and results stored.

Along with these array, matrix, and vector bulk operations, we will add data-typing support for physical units – which will allow us to catch a whole class of errors in program definition through the use of dimensional analysis. Automatic unit conversion, both within the program, and at user-interface boundaries can thus be gracefully handled.

Bulk Pointer Handling
More complex bulk data operations are also required, for handling structures other than arrays or vectors. Examples include sets, the parallel generalization of lists, as well as indexed or sorted versions of such structures. Additionally, we want to include database manipulation operations as language primitives. Examples include simple database AND and OR operations, as well as some very complex operations such as relational joins. The NYU/Courant development of SETL as a language, and the parallel *Lisp work at Thinking Machines should inform this work. Compiling these pointer-based operations into reconfigurable hardware is less straightforward than the pipelined approaches usable for bulk data operations. Nonetheless, we hope that here, too, we can achieve efficient use of the hardware to parallelize these crucial operations. Excellent low-latency interconnection networks are the key hardware enabler.

An explicit goal of this set of bulk data handling operations is the elimination of program loops. Loops are an artifact of a sequential mind set. Many trivial bugs occur in coding loops. Their elimination, through the use of bulk operations on well defined data objects should dramatically lower the trivial-bug count.

Streams
Stream processing of data is a natural mechanism for much of the needed signal processing. Dynamic languages already provide many of the features required for concisely expressing stream data operations. Recent work by Massalin and Shivers have led to efficient mechanisms for compiling stream code. These novel techniques rely upon abstract semantics, λ-calculus, continuation-passing style, and partial evaluation; we intend to explore using this technology to optimize across the composition of bidirectional stream transducers. Compilation into systolic processing datapaths for the on-chip reconfigurable logic should be a simple extension.

Conventional Garbage Collection
Other relatively simple pointer operations include in-memory garbage collection. Simple forms of such collectors could be built on top of the bulk pointer parallel operations described above—the mark operation is just transitive closure. More complex collector models will rely on the relativistic memory technology described below.

Data Typing is Essential
One important point to note is the absolute requirement in memory systems which are automatically collected, that the data be self-descriptive. It must always be evident from the data

itself that a particular bit string represents a pointer—and thus must be traced through during the mark phase of the collector.

Languages with higher-order procedures and sophisticated, polymorphic type systems are a powerful tool for expressing domain-specific computations. While a procedure written in C can conceivably be specialized with respect to a domain-specific word width for its integer computations, a higher-order procedure written in Scheme can be parameterized and specialized over *procedural* parameters. This ability to parameterize an algorithm over a dynamic computation is not available in first-order languages, and is one of the reasons why higher-order, polymorphic programming languages are more suited to domain-specific programming environments.

With just the set of software concepts described above, an extremely impressive set of capabilities can be delivered. However, we believe there are important additional ideas to be explored.

Software Layering

One central idea is the idea of software layering. We observe that there is an interesting discontinuity in the structure of our hardware/software systems. Typically, instruction execution is viewed as an atomic operation. Software developers never consider that an instruction might be half executed. It is the job of the hardware engineer to present a view of the program execution in which single instructions are either fully completed, or alternatively, not yet started. Such atomic operations enable the software developer to remain ignorant of the details of the hardware implementation.

A logical question to ask is whether this property of atomicity can be extended to higher levels of the software domain. Can we, indeed, develop *layers* of software infrastructure which, viewed from above, appear as atomic operations. Such strong isolation of a client software layer from a provider should allow excellent independent development and evolution of complex software systems. We have built at least one operating system (the PDP-10 ITS system) which delivered this kind of software methodology for operating system calls, utilizing a mechanism called PCSLR. We will investigate whether, and how, such a model can be extended. Particularly important is the question of whether, and how, a multi-layered structure can be built. To the best of our knowledge, no software structure of that kind has yet been constructed.

Layering promises to be a valuable software modularity tool. It encourages modular design, detail hiding, independent development, careful interface specification, sub-module testing, and careful error control.

Program Transactions

A closely related issue is that of *transactions*. We can view the atomic execution of an instruction as a transaction performed on the processor and memory state – with the requirement that the instruction be either atomically completed or completely backed out. Some of our earlier work [Knight 86] developed this idea in the context of automatic parallel subroutine execution on multiprocessors. We believe it is time to look further at these ideas in light of the layering concept described above, and the hardware capabilities now available.

Transactions, of course, are also the building block of a user-visible data base. Thus, building the essential aspects of transactional control directly into the software matrix will enable us to deliver transactional operations to applications much more straightforwardly.

Finally, transactions are the cornerstone of fault tolerant systems. By building transactions into every level of a system, we will be able to deliver fault tolerance with unprecedented simplicity.

Backing Out

Both transactions and layering require that we be able to gracefully back out of partially completed operations. Language features such as Common Lisp "unwind-protect" allow for careful restoration of the state of programs after error traps. Nonetheless, such operations tend to be error prone, and consume design and debug time out of proportion to their size. We are examining several approaches to solving this backout operation. In some of our recent work on fully-reversible computing, we can simply execute a program backwards to unwind a set of badly chosen operations. But for efficient execution, we believe the relativistic memory system ideas, described below are the appropriate solution.

Program State

Another closely related set of ideas is the notion of program state, investigated in the thesis of Alan Bawden at our Lab. Bawden provided a novel, and currently unexploited characterization of "system state," showing that processor state required the presence of both a duplicated reference path (nonlinearity) and execution parallelism. The close relationship of pointer paths, pointer duplication, garbage collection, and the notions of state in transactional systems requires that all of these issues be addressed simultaneously.

Relativistic Memory Systems

The efficient implementation of many of the above concepts relies on an ability to specify to the memory system not just *which* memory item is needed, but also, crucially, *what version* of that item. In other words, the request is an *event* in the relativistic sense. We are accustomed to building memory systems which keep exactly one copy of data at each location. Indeed, many of the difficulties associated with modern processor design revolve around the difficulties associated with keeping multiple copies of such data consistent. But with the advent of speculative execution, backout, transactions, and sophisticated simulation systems such as time warping, we need to think in terms of a relativistic memory design. Such a design specifies memory contents not simply with an address, but also with a timestamp. We believe that, only by building such mechanisms into the core data representations can we enable the next major set of software advances. Such referencing approaches will enable us to do out of order execution in a consistent way, for the first time. Checkpoints, backouts, restarts and dependency checks become straightforward. "What if" scenarios that rely on imaginary data can be easily played out. Time warping [Jefferson 82] approaches to decoupling physical simulation can be easily implemented.

The timestamped memory also directly assists us in building sophisticated generational garbage collectors, since we can directly know *when* a pointer was created.

The set of novel capabilities enabled by a careful re-examination of the core mechanisms for addressing data—such as including the time—will, we believe, enable the next generation of sophisticated language and applications developments.

Applications

In this section we present three motivating applications of the technology we propose to develop. The first of these, the Dynamic DataBase, is a DOD application, involving real-time image understanding as well as complex data modeling. The second, visual memory, is a experimental platform enabling research on high-level visual processing and on the structure of human memory. The third, Raster Memory, is a commercial driver for the components we seek to develop.

Dynamic Database

DARPA has begun development of a "Dynamic Database" capable of maintaining real-time situational awareness over a complete theater of war using all relevant observational assets. The term "Situational Awareness" means knowing the position and velocity of all possible targets within the theater. In current thinking, it is imagined that the situational assessment will be achieved by highly tuned processing pipelines (such as those in SAIP), which will convert sensor data to geographically-oriented representations of the potential targets.

This alone is a daunting task. However, the August 1997 ISAT study on the DDB outlined a far more ambitious goal, namely a total threat assessment. This aggregates and abstracts the situational model into an assessment of which forces dominate the others. Such a model involves multiple, overlapping representations oriented around transportation, battle fronts, logistics pathways, etc. In creating such a model, context from cooperating information sources plays an enormous role in influencing each processing step.

The DDB is a prime candidate for many of the features of our proposed system. The high-level abstraction of Temporal Memory will allow assessments from different times and from different hypothetical bases to coexist without confusion. Because interpretation is both context dependent and organized along many structural dimensions (roads, battle fronts, etc.) the ability of our system to manage locality around use (using automatic storage-management techniques to adapt dynamically to demands of applications) becomes critical. Finally, image-signal flows are channeled through processing pipelines which produce symbolic interpretations. Here the ability to configure logic on-chip, near to the image array, offers orders of magnitude improvement in processing power.

Visual Memory

The Dynamic Database involves rapid classification of imagery (target vs. non-target); this is simply one of many "High-Level Vision" tasks. Other similar tasks involve face recognition, specific object recognition (as opposed to classification), scene description, etc. A robot (or human) navigating through the world needs to perform such tasks routinely and robustly. It is one of the great scientific questions of Artificial Intelligence research to understand how people manage to perform these seemingly difficult tasks so easily and so robustly.

Each of the high-level visual tasks involves both signal processing and semantic matching of the processed image to huge background knowledge bases of models. Shimon Ullman, a research affiliate of our lab, has described in his book on high-level vision a "counter streams" architecture for visual recognition which appears to be both physiologically plausible and highly attractive algorithmically for any image-understanding system. In this architecture, memory-based models flow *downstream,* being transformed at each step, while images derived from the perceptual signals flow *upstream,* again being transformed at each step. (In his scheme 3D models are represented as linear combinations of 2D images taken from a variety of poses; in principal 3 poses are adequate but in practice many more poses are useful). Some of the transformations are purely geometric, while others may be driven by the semantics of the model class, such as variations in facial expression.

This model of visual recognition seems like a natural for implementation in a data-intensive system, since the counter streams can be implemented by passing images from a memory-processor module to several of its nearby neighbors. Since there is processor logic at each node, the various transformations and matching steps can be computed locally. The availability of reconfigurable logic at the node allows the system to configure special-purpose data paths optimized for pixel arrays and for specialized and idiosyncratic transformations dictated by the model class. These are complex tasks which require high-level expression. As we have argued, Dynamic Object Oriented Languages provide the medium for creating domain-specific languages for expressing each of these tasks. And the compiler technology we propose can then transform such expressions into highly efficient configurations of dynamically specialized hardware and the appropriate software to run on them.

Raster Image Memory

We believe that it is necessary to have a commercial driver (a "killer app") for the combined memory/logic component of our system. One such application we will explore is using these chips as a new vehicle for high-end Raster Image Processors (RIP's) such as those used for magazine and advertisement copy. This is a unique niche market within the overall RIP business which is rapidly growing and highly profitable. (As it happens, the leader in this business is Harlequin, a US/UK company with offices in Cambridge Massachusetts and UK. They are also one of the principal vendors of Lisp technology; the MIT AI Lab has a close relationship with Harlequin).

High-end RIPs produce very large pixel arrays both because of the size of the page and the density of the pixels. These systems interpret a source language (e.g. PostScript), convert it to a raw raster image, and then provide sophisticated post-processing to guarantee fidelity in the final printed image. They, therefore, demand the large scale, computationally balanced memory systems that we seek to develop in our research. We imagine building a system in which the top level of interpretation is performed in a sequential (or nearly so) front-end which farms out the details of rendering and correction to the component memory systems. The correction phase of the processing will involve both local execution of complex algorithms and communication between adjacent image fragments. Both of these may make use of the reconfigurable logic.

C. Deliverables

Base Contract:

- A complete specification of the DRAM/ASIC component
- A detailed simulation model of the DRAM/ASIC component
- Design for a high-performance interconnection switch
- Design and prototype implementation of a layered-software methodology
- Design and prototype implementation of a self garbage collecting memory system

Option contract:

- Complete documentation for an implementation of the DRAM/ASIC reconfigurable logic component (Y2).
- Complete documentation for an implementation of a complete microprocessor, switch, and active memory system.
- Complete documentation for an implementation of a significant DOD image-processing task on the architecture.

Note: MIT overhead agreements make actual delivery of prototypes undesirable for cost reasons.

D. Statement of Work

Base:

- MIT will create a system level architecture for a memory system and interconnection network tuned to large database applications.
- MIT will specify a detailed architecture for a reconfigurable computing / DRAM active-memory component, leveraging commercially available on-chip DRAM technology.
- MIT will specify the design of an interconnection network capable of handling inter-chip pointer references.
- MIT will investigate innovative programming technologies, centering on the ideas of transactions and the hard layering of software structures.
- MIT will investigate language features required for high-performance stream data processing, allowing handling of data in bulk.
- MIT will construct simulations for the active-memory component.
- MIT will investigate stream / counterstream visual signal processing approaches.

Option:

- MIT and a corporate partner will perform detailed design and fabrication of the active-memory component.
- MIT and a corporate partner will build a full prototype system.
- MIT and a corporate partner will evaluate the complete system with respect to one or more DOD applications.

E. Schedule and Milestones

Year 1:

- System architecture
- Component architecture
- Language data layouts, types
- Layered-software approaches, transactional techniques
- Option – Component partner choice
- Option – Application partner choice
- Option – detailed component architecture

Year 2:

- Simulate component, architecture
- prototype language implementation
- demonstration of layered-software approaches
- Option – System-implementation partner choice
- Option – Component fabrication
- Option – Performance estimation and optimization for DOD applications

Year 3:

- Implementation of a toy application in simulation
- Evaluation of the effectiveness of software-methodology development
- Option – system evaluation against DOD mission requirement
- Option – Assist partners in technology transfer to commercial and DOD applications

F. Technology Transfer

Our most important effort in technology transfer under this program will be to create and popularize a novel commodity computing component. It will be essential in this effort to identify a VLSI partner who is strongly motivated to commercialize the component designed under this effort. Likewise, it is essential that we locate one or more early adopters of the technology with at least low-volume commercial applications. We have already begun discussion with several potential partners in both of these areas.

- Toshiba and Mitsubishi are vendors of the DRAM/ASIC technology. Our most extensive discussions to date have been with Toshiba, who is currently providing the most aggressive technology. We plan on evaluating IBM as an alternative. Mitsubishi has the most commercially developed technology, delivering a 32 bit microprocessor with 16Mbits of DRAM on a single die. Volume pricing on this component is less than $30.
- Digital, Sun, Lockheed, Alphatech, Harlequin, and EMC have been approached as potential system-level partners, who could assemble and market commercial-scale systems. Our most extensive discussions to date have been with Sun and Harlequin.
- Lockheed, NRO, General Electric, Alphatech, Harlequin, and British Aerospace have been approached as potential users of this technology. General Electric has expressed interest for their medical-imaging applications, and Harlequin has expressed interest for raster image processing in high performance page printers. Alphatech has interest as a systems-level integrator for DOD applications.

Historically, our most effective technology transfer has been through the graduation of students who carry novel ideas with them. These ideas often seed start-up companies, as we have seen in the case of Symbolics Inc., and Thinking Machines Corporation, both Laboratory spinoffs in the computer systems area. The Laboratory has seeded over twenty startup companies, working in diverse areas including industrial computer vision, military-planning technology, Internet databases, and financial services.

Other approaches to technology transfer include the Industrial Liaison program, faculty consulting, as well as extensive participation in conferences and journal publication.

G. Other Research

Our laboratory has a long history of hardware, language, and software development tuned to signal and cognitive processing. A partial list includes:

- (1966) MacLisp for the PDP-10 processor, and the first large memory system for AI applications, a 1 megabyte core memory—the largest in the world at the time.
- (1969) Development of the first real-time garbage collection algorithms (Baker) and the first actor-based object-oriented approaches to software development (Bishop).
- (1973) First hardware proposal for massively parallel AI search and database retrievals (Fahlman).
- (1974) Lisp Machine development—the first workstation specifically designed with hardware for supporting Lisp-like languages, including dynamic type checks, a "safe" programming language, a single, large address space, real-time garbage collection, and sophisticated networking and interface peripherals. This technology was later commercialized by Lisp Machine Inc., Symbolics, and Texas Instruments (Greenblatt & Moon & Knight).
- (1979) Scheme chip—a single-chip interpreter of the Scheme programming language, including multiprocessor-compatible garbage-collection features (Holloway, Sussman & Steele).
- (1981) Connection Machine development—First large-scale SIMD processor with an ability to handle parallel pointer access through a routing network. This technology was later commercialized by Thinking Machines Corporation. Later work with this architecture at the Laboratory led to the development of the Vision Machine, a set of supported subroutines for the CM-2 processor (Hillis & Knight).
- (1983) First fully-integrated silicon retina supports visual processing of images in analog form within the sensor array (Knight).
- (1986) Scheme is popularized as the most important pedagogic language for the advanced university programming community (Abelson & Sussman).
- (1984) Common Lisp is defined, ending competition between MacLisp and Interlisp, with an effective merger. The Laboratory plays a central role in the resolution of differences and implementation of the standard (Steele).
- (1986) The Liquid computer architecture proposes building programming language semantics on a transactional substrate (Knight).
- (1990) Transit/Metro interconnection switch technology delivers lower latency than any previous switch fabric through the use of cut-through routing and connection based bidirectional channels (Minsky & Knight).

- (1991) CAM-8 processor is the first delivered system for high performance cellular-automata simulation. Delivers 50MHz clock rates in a table-lookup based design, expandable to as many as 32 simultaneous operations. Flexible addressing and complete integration with the Sun environment allows widespread use in other laboratories (Margolus).

- (1992) The Supercomputer Toolkit, an array of floating-point processing engines interconnected with a high-performance switching network, constructed in collaboration with Hewlett-Packard Laboratory, is used to integrate the equations of the solar system, proving that the orbit of Pluto is chaotic (Sussman, Wisdom, & Zuras).

- (1992) Abacus SIMD silicon processor component delivers 1K processors and 125MHz clock rates in a single-chip SIMD processor, about 1,000 times the performance of the components in the CM-2 processor (Bolotski & Knight) .

- (1993) Linear logic is used to understand the role of state in programming language design (Bawden).

- (1995) Matrix and DPGA components redirect attention in the reconfigurable logic area away from gate-level design toward higher functionality devices. Matrix is the first reconfigurable component to deliver performance comparable to custom silicon, through the use of bit-slice register files, ALUs and multiplier arrays. (DeHon, Eslick, & Knight).

- (1996) Pendulum, the world's first reversible programming language and architecture is implemented, with important consequences for power savings and language design (Vieri, Frank, & Knight).

- (1997) The amorphous computing effort addresses language and system issues in fault tolerance, massive parallelism, and the emergence of global behavior from local processing and communications (Sussman, Abelson, & Knight).

Of special importance to the hardware aspects of this proposal are our previous efforts in developing the Connection Machine, and its follow-on high-performance silicon implementation, Abacus. Together with our work on the Transit interconnection network, and our innovations in reconfigurable logic, we have previous work in all of the required base technologies addressed in this proposal.

In the software area, we will build on several of our previous accomplishments. With Lisp- and Scheme-based languages, we access a wide variety of symbolic computing techniques. By advancing the parallel variants, such as *Lisp, developed for the Connection Machine, we can add the important features required for bulk data operations. However, the goals of software layering, transactional processing, and automatic garbage collection in memory require us to build on our previous work on the Lisp Machine, and theoretical work on real-time garbage collection, transaction processing, and program state.

There is also a substantial body of related work at other institutions. Of this work, we believe the most directly relevant is the collaborative work between the Supercomputing Research Center and Cray Computer on the Terasys processor in memory array [Gokhale 95] [Cray 94]. This work is unique in that the memory of the SIMD array is accessible to a high-performance conventional processor, unlike previous SIMD array implementations. While the density PIM chip is substantially below that achieved by Abacus in equivalent technology, the integration of the array with a conventional processor provides ready access to a variety of communication and processing capabilities otherwise unavailable. The PIM technology, unfortunately, also omitted support for two other system aspects which we believe are crucial: data tags on processor data, and pointer-based message delivery. Both of these are of great utility in sophisticated object-oriented database technology, as a way of making the data self-descriptive, and to enable the parallel processing of pointer-based data structures, such as complex intelligent knowledge databases.

Previous work on other SIMD processors (MPP, GAPP, DAP, MasPar, Wavetracer, Illiac-IV) is well reviewed elsewhere. Notable, perhaps, is the early work on embedded DRAM plus processing done at the IBM Federal Systems Division.

Other hardware of note includes the reconfigurable processing array work at Digital Paris research laboratory. While building on conventional FPGA technology, this work achieved dramatically better performance at important tasks such as searching genomic databases and brute-force solution of cryptographic problems.

The ongoing work by Patterson at Berkeley on embedding risc processors into DRAMs relies on similar technology as does our proposed work, but is otherwise quite distinct, in that the array comprises a MIMD processor, rather than a conventional uniprocessor with active memory.

Very large, reliable, disk arrays are the bread and butter technology for EMC, a local mainframe storage provider. Single cabinets containing tens of terabytes of memory are available now; future version containing more than a petabyte are already under design. Collaboration with EMC will provide access to this raw storage capacity, and integration into existing mainframe systems in way other difficult for us.

In the software area and system-organization areas, there are several related efforts. Wilson, at the University of Texas, Austin, is developing sophisticated techniques for handling pointers in very large, garbage collected, distributed databases. David Weiss, at Indiana, has developed a memory system for a conventional processor which can automatically track reference counts in the memory, and perform garbage reclamation, within the limitations of that technique. Work at Harlequin on the Dylan language, and continued work in ML, Smalltalk, and Java, all pushes forward the development of the sort of dynamic languages which we believe are essential for the kind of dynamic object-oriented processing required by large knowledge databases.

H. Key Personnel

Dr. Thomas F. Knight, Jr. is a Senior Research Scientist in the MIT Artificial Intelligence Laboratory and the MIT Department of Electrical Engineering and Computer Science. His work focuses on the development of novel computing structures at the implementation, architectural, and programming levels. An accomplished computer architect, he was a founder and technical director at Symbolics, Inc., Exa Corporation, and Polychip, Inc. At MIT he teaches a senior level computer design and architecture course and a graduate VLSI digital circuit design course. He received the MIT Graduate Student Council teaching award in 1991. His recent architectural work has focused on low latency interconnection networks, SIMD processors, reversible machines, and reconfigurable logic arrays. Current research topics include developing novel computing structures from living biological cells, physical limits to computation, and advanced human/computer interface technology.

Dr. Howard E. Shrobe is Associate Director and a Principal Research Scientist of the MIT Artificial Intelligence Laboratory. During 1994–1997 he served as an IPA at DARPA where he served as Chief Scientist of the Information Technology Office. His work has spanned the areas of Artificial Intelligence, Software Engineering, Advanced Computer Languages, VLSI design, and Computer Architectures to serve the special needs of these advanced computing technologies. He received his MS and PhD degrees from the Artificial Intelligence Laboratory at MIT where he was a co-founder of the Programmer's Apprentice project. In 1979 he joined the staff of the MIT Artificial Intelligence Lab as a Principal Research Scientist and in that role was one of the main designers of the Scheme-81 microprocessor (a LISP interpreter on a chip) and the DPL/Daedalus Integrated Circuit Design. Dr. Shrobe also served as Chief Technical Officer at Symbolics Inc. At Symbolics, Dr. Shrobe was one of the architects of the Ivory microprocessor (a Lisp chip with novel garbage collection hardware) and of the NS CAD system used to design it. He also supervised the development of Statice, the first commercial object-oriented data base.

Dr. Olin Shivers is a Research Scientist at the Massachusetts Institute of Technology. Dr. Shivers is a recognized expert in the field of compiler optimizations, semantics-based static analysis, and software systems. His doctoral dissertation on the static analysis of higher-order programming languages was nominated for the ACM Distinguished Dissertation award, and has defined the terms and methodology of subsequent research in the field.

I. Facilities

The Artificial Intelligence Laboratory is an interdepartmental Laboratory of the Massachusetts Institute of Technology. It houses approximately 180 faculty, students, and staff and performs research in intelligent systems, human interfaces, cognitive processing, computer vision, robotics, computing technology, and software systems. Our facilities include high-end workstations from Sun, Digital, Intel, Hewlett-Packard, and Silicon Graphics. In addition, we construct and utilize unique, in-house hardware and software systems, including robots, custom user-interface displays and kinesthetic interfaces, computer-vision systems, and high-performance architectures.

Perhaps our most important assets are our students, who consistently provide implementation, inspiration, and effective technology transfer.

In addition to computing resources, we also have available a complete machine shop, extensive high-performance digital electronics construction and debugging facilities, and industrial grade CAD tools, including complete Cadence VLSI tools, Code-V optical design tools, and Pro-Engineer mechanical tools.

Access to other ongoing research programs at the Laboratory, especially in image processing, image understanding, knowledge representation, user interface, and text understanding, will directly assist us in this program.

J. Cost by Task Please see attached budgets.

Section III: Bibliography and papers

- Gokhale, M, Holmes, W, and Iobst, K, "Processor in Memory: The Terasys Massively Parallel PIM Array," IEEE Computer, April 1995, pp. 23-31.
- Cray, S, Keynote address, PetaFlops Workshop, Pasadena, 1994 (Proceedings published by MIT Press).
- Baker, Henry, "The Treadmill: Real-Time Garbage Collection Without Motion Sickness", ACM Sigplan Notices, Vol 27 (3), pp. 66-70, 1992.
- Baker, Henry, "List Processing in Real Time on a Serial Computer", Communications of the ACM, vol 21(4), pp 280-294, 1978
- Becker, M, and Knight, TF, "Fast Arbitration in Dilated Routers," Parallel Computer Routing and Communications Workshop, University of Washington, Seattle, May, 1994.
- Bishop, Peter. B, Computer Systems With A Very Large Address Space and Garbage Collection, MIT Laboratory for Computer Science MIT/LCS/TR-178, 1977.
- Bolotski, M, Amirtharajah, R, Chen, W, Kutscha, T, Simon, T, and Knight, TF, "Abacus: A High Performance Architecture for Vision," Int. Conference on Pattern Recognition, October, 1994.
- Chong, FT, and Knight, TF, "Randomized Multipath MIN Architectures," 4th Annual ACM Symposium on Parallel Algorithms and Architectures – SPAA '92, San Diego, p. 286-295, June 1992.
- Chong, FT, Brewer, E, and Knight, TF, "Building a Better Butterfly: The Multiplexed Butterfly," Parallel Computer Routing and Communications Workshop, University of Washington, Seattle, May, 1994.
- Chong, FT, Brewer, E, Leighton, T, and Knight, TF, "Building a Better Butterfly: The Multiplexed Metabutterfly," International Symposium on Parallel Architectures, Algorithms, and Networks, Kanazawa, Japan, December, 1994.
- DeHon, A, Bolotski, M, and Knight, TF, "Unifying FPGAs and SIMD Arrays," ACM/Sigda workshop on FGPAs Feb. 1994.
- DeHon, A, Drenkhahn, F, Knight, TF, and Minsky, H, "3-Dimensional Packaging for High Peformance Interconnect in Large Scale VLSI Systems," International Journal of High Speed Electronics and Systems, Vol. 6 No. 4 (1995) pp. 613-630.
- DeHon, A, Knight, T, and Minsky, H, "Fault Tolerant Design of Multistage Routing Networks," Int. Symposium on Shared Memory Multiprocessing, p. 60-71, Inf. Proc. Soc. of Japan, Tokyo, April 1991.
- DeHon, Andre, "Metro: A Router Architecture for High Performance Short Haul Routing Networks," Int. Symposium on Computer Architecture, May 1994.

- Knight, TF, “An Architecture for Mostly Functional Languages,” ACM Lisp and Functional Programming Conference, August, 1986.
- Knight, TF, “Technologies for Low Latency Interconnection Networks,” ACM Symposium on Parallel Algorithms and Architectures, June, 1989.
- Knight, TF, “Technologies for Low Latency Interconnection Switches,” Computer Architecture News, v. 19, n. 1, pp 61-8, March, 1991.
- Knight, TF, “Transit: Technology for Reliable, Low Latency Interconnection Networks,” Invited keynote paper, International Symposium on Parallel Architectures, Algorithms, and Networks, Kanazawa, Japan, December, 1994.
- Ladin, R and Liskov, B, “Garbage Collection of a Distributed Heap” Proceedings of 12th International Conference on DCS, 1992.
- Lieberman, H and Hewitt, C, “A Real–Time GarbageCollector Based on the Lifetimes of Objects.” Communications of the ACM, June, 1983.
- Minsky, Henry, Knight, TF, and DeHon, A, “An Enhanced Crossbar Chip for a Shared Memory Multiprocessor Router,” Hot Chips III Conference, Stanford, Aug. 1991.
- Minsky, Marvin L., “A LISP Garbage Collector Algorithm Using Serial Secondary Storage”, MIT Artificial Intelligence Laboratory AI Memo 58, 1963.
- Moon, David A., “Symbolics Architecture”, IEEE Computer vol 20 (1), pp. 43-52, 1987.
- Moon, David A., “Garbage Collection in a Large Lisp System”, Conference Record of the 1984 ACM Symposium on Lisp and Functional Programming, pp. 235-246, 1984.
- Shivers, Olin, *Control-Flow Analysis of Higher-Order Languages.* Ph.D. dissertation, Carnegie Mellon University, May 1991. Technical Report CMU-CS-91-145, School of Computer Science.
- Ungar, David, “Generational Scavenging: A Non–Disruptive High Performance Storage Reclamation Algorithm”, ACM Software Eng. Notes/SIGPLAN Notices Software Engineering Symposiumon on Practical Software Development Environments, 1984.
- Vieri, C, Frank, MP, and Knight, TF, “A Reversible Computer for Low Energy Computing,” submitted, International Symposium on Low Power Electronics, October, 1996.