High-Performance Computing Systems for Autonomous Spaceborne Missions

Thomas Sterling
Daniel S. Katz
Larry Bergman

NASA Jet Propulsion Laboratory
California Institute of Technology

Abstract

Future generation space missions across the solar system to the planets, moons, asteroids, and comets may someday incorporate supercomputers both to expand the range of missions being conducted and to significantly reduce their cost. By performing science computation directly on the spacecraft itself, the amount of data required to be down-linked may be reduced by many orders of magnitude, thus greatly reducing the mass of the resources needed for communication while increasing the quality and quantity of the science achieved. By performing the mission planning in real time directly on the spacecraft, complex and highly responsive missions can be conducted out of range of direct human intervention and the cost of mission management can be reduced. Through highly replicated computing structures, continued operation can be maintained in the presence of faults by means of graceful degradation. Two classes of systems, reflecting very different strategies of computer system architecture, are actively being pursued by the NASA Jet Propulsion Laboratory to take advantage of the opportunity of embedded high performance computing on spacecraft for deep space missions. Commodity off-the-shelf (COTS) Clusters may permit the direct application of commercial computing hardware in loosely coupled ensembles to benefit from the enormous investment of industry in mass-market components. New Processor-in-Memory (PIM) architectures combine multiple nodes on a single chip of processor-memory pairs exposing the full memory bandwidth. This paper examines the driving issues motivating the use of supercomputing for future deep space missions and describes two active research projects at NASA JPL that are pursuing both the COTS and PIM strategies for next generation spaceborne computing.

1. Introduction

The field of embedded computing is rapidly expanding beyond its historical boundaries of small single-chip programmable controllers in automobiles, medical equipment, and toasters. For real time control assimilating real time image information and for intelligent real-time planning of action sequences, embedded computers are demanding performance and storage capacities once reserved for the domain of high-performance computer systems. No application area will stress the demands of embedded computing more than the requirements of future spacecraft employed for deep space missions throughout the solar system and beyond. Past missions have used relatively modest performance computers for control, data assimilation, and data downlink, emphasizing low power and reliability. Future missions, while maintaining these important characteristics, will expand their capabilities in order to reduce spacecraft cost through bandwidth reduction, reduced power consumption, and improved reliability through parallelism. In addition, new classes of deep space missions will be enabled through the merger of high performance computing and autonomous real time planning that will include onboard symbolic processing.

The challenge of combining high reliability, low power, and high performance at reasonable cost for future generation spacecraft is being addressed by two strategies at the NASA Jet Propulsion Laboratory. Representing an important break with the past, researchers pursuing one approach are exploring the potential of employing commercial off-the-shelf components in clusters to manage costs and track technology advances while achieving reliability through software fault tolerance techniques and node redundancy. In a second approach, the recent fabrication technology permitting the merger of both DRAM cells and CMOS logic on the same chip die, referred to as processor in memory or PIM, has created new opportunities for innovative computer architecture that may provide superior processing capability for spaceborne systems within a scalable framework. The COTS strategy may yield new systems in the near term while missions may employ PIM at the end of the decade. In the long term, PIM may become COTS and the two approaches merge. This paper briefly presents the potential for high-performance computing in spacecraft to support low cost, long duration, deep space science missions. The two differing strategies being investigated by NASA are described with both the benefits and difficulties considered. The COTS strategy is being undertaken by the Remote Exploration and Experimentation (REE) project. The PIM strategy is being pursued by the Gilgamesh project. This paper describes the major opportunities for high performance computing in space and provides details of these two very different concepts in meeting the needs of future deep space missions.

2. Role of Computing in Future Deep Space Missions

During the last half of the 20th century, NASA has completed a preliminary survey of all but one of the planets of the solar system with unmanned spacecraft. The wealth of information returned has far exceeded that which could be obtained from Earth bound observations with the largest telescopes and has greatly reshaped our thinking on how the solar system and individual planets were formed, and more importantly, has given us vital clues on the birth and lifecycle of planetary bodies.

As mankind embarks into the 21st century, important questions remain: How did life begin? Does it or did it exist on other bodies within the solar system? How do life-bearing planets form? Do Earth class planets and intelligent life exist elsewhere in the Universe? The answers to these questions (and/others) will require the development of far more sophisticated unmanned spacecraft than have been flown to-date. The requirements will be daunting: they must venture into far more hostile environments such as under the frozen oceans of Europa, gather comet dust, or navigate asteroid fields. This will require far more on-board autonomy than has ever been flown. These spacecraft must be able to *think* and behave on their own, as human-in-the-loop control will be difficult or impossible because of the delays due to long distances. Examples include docking and rendezvous, entry-descent-landing, and planning and optimizing the operation of a collection of instruments during a short observation window. Most ambitious of all would be giving a rover or fleet of rovers a science goal or set of goals, and permitting them to gather specimens independent of low-level human control. This last application may soon be reality. MISUS (Multi-Rover Science Understanding System,) now being developed at the NASA Jet Propulsion Laboratory by the Machine Learning Systems Group, will allow multiple rovers to locally analyze instrument data and make decisions on which rock samples to measure and even return to a mother ship (if part of a sample/return mission). More advanced generations of MISUS may someday even imbue emotions; that is, be able to manage risk versus science payoff, or be able to look for the unexpected, such as erupting volcanoes or geysers.

Another unique problem associated with deep space exploration is that the communication link capacity between the spacecraft and Earth has not been growing at the same rate as the instrument resolution and generated data volume. The net result is that the science return is greatly diminished from its potential, and will not improve unless some form of on-board processing can be brought to bear to analyze and compress/downlink only the data of most interest.

For these reasons (the trend toward increased on-board intelligence and data analysis capability,) next generation spacecraft computer architecture designs are driving toward higher performance systems that are:

- Scalable
- Low power and mass
- Able to be embedded in real-time hardware in loop control systems
- Able to Support robust fault management methods
- Capable of high I/O and floating point performance to support large image and data analysis applications
- Capable of high performance in accessing random memory blocks to support autonomy applications

The first generation approach that leverages heavily off of commodity off-the-shelf (COTS) components is REE, which combines robust fault management middleware with COTS microprocessor clusters to increase the fault tolerance to a sufficiently acceptable level for space systems. REE provides high floating point performance essential for image processing space borne applications, is constrained by current COTS technology in: non-optimal size, weight, and power consumption (there is a great deal of silicon real estate on each die that is not utilized for many space applications), and I/O. These are distributed memory clusters, rather than shared memory machines. Clusters (such as a Beowulf) can perform well on applications even when not embarrassingly parallel. Newer approaches, based on processor in memory (PIM) and RISC like designs (typical of Gilgamesh) offer higher I/O, much lower latency across global memory, better scalability, and the ability to degrade with finer granularity in the presence of either permanent or soft faults (e.g., single event effects or SEEs). But the COTS approach leverages industrial investment in mass-market components while that of Gilgamesh demands custom development for a small market. For this reason, both system strategies are being pursued in support of next generation space mission applications requiring high performance and high reliability. REE promises to deliver one to two orders of magnitude improvement in power-performance over current spacecraft in the next few years, while Gilgamesh will exceed this, in a longer time frame.

3. COTS Clusters for Spaceborne Computing

The goal of the REE Project [1] is to move ground-based supercomputing into space in a cost effective manner and to allow the use of inexpensive, state of the art, commercial-off-the-shelf (COTS) components and subsystems in these space-based supercomputers. The motivation for the project is the lack of bandwidth and long round trip communication delays that severely constrain current space science missions. Unlike typical radiation-hardened space-based systems, the use of COTS hardware will require the REE system to withstand relatively high rates of single event upset (SEU) induced errors. Depending on mission environments and component technologies, an REE system will be required to withstand average fault rates of between 1 and 100 SEU-induced soft errors per CPU-MB-day with occasional peaks of up to 1000 soft errors per CPU-MB-day [2]. Unlike traditional fault tolerant computer systems, the REE computer need not provide 100% reliability but instead, as with many sampled data or convergent-computation systems, is allowed to occasionally fail in a computation. Periodic resets to flush latent errors, and other techniques which provide less than 100% availability, are also permissible. Further, the REE computer need not initially support hard real time or mission critical computation, as these tasks can be off-loaded to the spacecraft control computer in the first REE systems.

The flexibility afforded by the above requirements allows the system to be optimized for high-performance, low-power, supercomputing rather than for “hard” fault tolerance. Thus, REE seeks to maximize simplex operation and minimize resource replication, redundant executions and other high-overhead strategies. Software-implemented triple-modular redundancy (TMR) and other high-overhead techniques are integrated into a suite of operational options for flexible fault tolerance, but these are not the primary operating modes of the system. This permits a small subset of nodes to operate in a highly reliable and real time manner.

Another project goal is to allow scientists to develop science applications in their laboratories and to easily port the resulting software to the REE computer with minimal or no re-engineering for fault tolerance or for the spacecraft computing environment. In addition to COTS hardware, the project thus seeks to utilize a commercial operating system and to support standard commercial application development tools (compilers, debuggers, etc.) and methods to the maximum extent practical.

The REE computer architecture is a Beowulf-type[1] [3] parallel processing supercomputer comprising a multiplicity of processing nodes interconnected by a high-speed, multiply redundant communication fabric. In the current instantiation of the system, dual Power PC 750 based computational nodes containing 128MB of main memory and dual redundant Myrinet [4] interfaces are interconnected via a redundant Myrinet fabric. The node level operating system is Lynx [5] Operating System (OS), to which multiple versions of MPI [6] have been ported. The current system may contain up to 20 nodes (40 processors) and is extensible to at least 50 nodes with a power-performance of better than 40 MOPS/Watt. The applications are written so that they may be automatically configured to execute on up to 50 processors with the system being informed, by the application, of the optimal number of processors for maximum throughput and the system assigning the number of processors available based on system status and operational constraints such as available power, spares availability and mission phase.

REE is designed with multiple layers of fault-tolerance with the detection and correction of errors accomplished at as low a level as possible. Analysis of application fault tolerance (AFT) requirements and determination of the applications' native error tolerance is being conducted, in conjunction with the development of generalized software-implemented fault-tolerance (SIFT) mechanisms. To aid application developers, a library of fault-detection-enabled scientific subroutines for linear algebra and Fast-Fourier Transform (FFT) routines has been developed. Three system software layers have thus far been defined to aid in achieving the required fault tolerance:

- A middleware layer which, conceptually, resides between the OS and the application,
- A reliable communications layer which ensures that all system level communications are either error free or error-noted and which, conceptually, is viewed as a series of driver level enhancements to the node OS, and
- A global coordination system that manages the overall system.

The combination of node operating system, reliable communications software, middleware, and global coordination layers are simply referred to as the REE System Software. Some of the responsibilities of the REE system software include:

[1] Beowulf-class computers were originally defined as parallel clusters of commodity hardware and open-source operating systems and tools. This definition has grown to include most clusters composed of personal computer central processing units (CPUs) and commodity operating systems and tools.

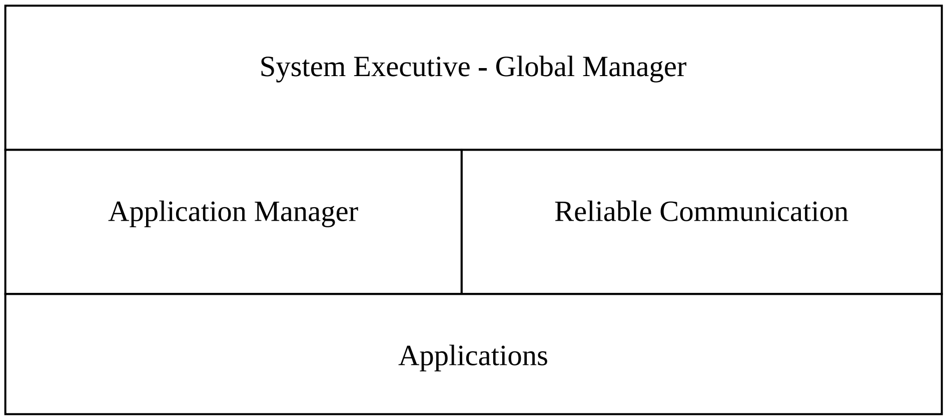


Figure 1. Software Layers in the REE architecture.

1. Managing system resources (maintaining state information about each node and about the global system, performing system resource diagnostics, etc.).
2. Job scheduling (globally scheduling jobs across the system, local job scheduling within the node, allocation of resources to jobs, etc.).
3. Managing the scientific applications (launching the applications, monitoring the applications for failure, initiating recovery for applications, etc.).

Part of the SIFT middleware layer is an Application Manager, which is intended to oversee the execution of the scientific applications. As the applications represent the ultimate "customer" of the REE environment, efficiently supporting their required dependability level is paramount. The Applications Manager monitors the science application for externally visible signs of faulty behavior as well as for messages generated internally by the applications requesting fault tolerance services.

Fault tolerance concerns for the REE System Software must also be addressed since these components ultimately ensure the correct operation of the REE environment. Several of its operations, such as scheduling and resource allocation, are considered to be critical and therefore must be protected at all costs. We currently envision that these operations will therefore be run under the software-implemented TMR system previously discussed. Another module that must be protected is the Applications Manager. This software module, which is resident on each node engaged in applications processing, must be self-checking to ensure correct operation of this "middleware" layer.

3.1 Application Manager

The scientific applications executing on an REE platform are programmed using MPI [6], a standardized messaging interface used to implement parallel applications. These are typically computationally intensive programs that perform such actions as on-board image filtering and signal processing.

Core routines within each application, such as matrix multiplication, employ algorithm-based fault tolerance (ABFT) [7] [8] to help protect against data faults. Internal ABFT techniques, however, do not mitigate the need for an external entity controlling the applications. Capabilities such as launching application processes, terminating rogue application processes, detecting failures in application processes, and migrating failed processes to functioning nodes are some responsibilities that must be relegated to an external controlling entity. The Application Manager fulfills this role.

Because the target MPI applications often consist of several processes and because these applications cannot sacrifice performance, replication is not viewed as an acceptable approach for ensuring fault tolerance in an environment with constrained processing resources. Effective reporting and detection of errors is considered most important, as the target MPI applications can tolerate occasional restarts and rollbacks to previous checkpoints.

For tolerating non-crash failures, the application manager will provide an API (application programming interface) to the MPI application. The MPI application will be lightly instrumented with API calls; no fundamental redesign of the application will be necessary. Through this interface, the application can communicate vital information to the Application Manager so that it can measure the health of the application. Examples include the reporting of correctable and uncorrectable ABFT errors directly detected by the application, as well as periodic updates concerning the application's progress. REE will also investigate applying some of the error detection techniques such as control flow signature checking on the process's execution. The ultimate goal of these techniques is to improve a process's self-checking capabilities, and the MPI applications outfitted with these techniques would work in tandem with the application manager to accomplish this goal.

Recovery of an MPI application is complicated by the fact that current MPI implementations do not easily allow single MPI processes to be restarted; instead, all processes must be launched again by the Application Manager to restart the application. The application manager is also responsible for handling node failures that affect one or more MPI applications. Whenever the application manager detects a node failure through its heart-beating mechanism, it must migrate all affected processes to another node. Selecting a spare node is done with the assistance of Resource Manager, an REE System Software component that is external to the Application Manager. Again, the MPI applications themselves are oblivious to the exact recovery actions taken by the SIFT layer.

3.2 Algorithm-Based Fault Tolerance

The main idea behind Algorithm-Based Fault Tolerance (ABFT) [7] is that the data being used in an algorithm may be augmented with a small amount of additional checksum-like information. At the completion of the algorithm, this additional data may be checked to see if the algorithm transformed it correctly. If this method is used literally, an ABFT-enhanced library routine that multiplies two *n* 5 *n* matrices would have to copy the matrices to *n+1* 5 *n+1* matrices, calculate row and column checksums into the extra rows and columns, do the multiplication, check to see if the checksums on the resultant *n+1* 5 *n+1* matrix hold, then copy the *n* 5 *n* part of the matrix into the library's output. An alternative which is less computationally intensive is to find a post-condition on the multiply which may be checked. For the matrix problem **A** * **B** = **C**, one might select a random vector of length *n* **r**, and check to see if **A** * (**B** * **r**) = **C** * **r** . This check involves 3 operations of $O(n^2)$ to verify an operation of $O(n^3)$. This idea is known as result checking (RC) [9].

Both ABFT and RC must take care with round-off error. In general, it appears that RC can catch as many errors as ABFT, and at lower computational cost. The REE Project has implemented result checking in multiple contexts. In parallel, it has been applied to matrix-matrix multiplication, LU decomposition, singular value decomposition, and fast Fourier transform (FFT). The matrix operations are based on PLAPACK [10], the FFT on FFTW [11] [8]. Sequentially, result checking has been applied to Level 3 BLAS [12] [13].

3.3 Applications

The REE project initially selected a set of first round application teams, consisting of:

- Gamma-ray Large Area Space Telescope (GLAST): This team, led by Prof. Peter Michelson (Stanford) and Prof. Toby Burnett (U. of Washington) will examine detection of gamma rays in a sea of background cosmic rays (about 1 in 10,000 events will be a gamma ray), and reconstruction of the gamma-ray trajectory.
- Mars Rover Science: Dr. R. Steven Saunders (JPL) leads this team, which has two applications. First, texture analysis and image segmentation are used to identify various materials on Mars for further scientific analysis. Second, images obtained from a stereo camera are analyzed for use in autonomous navigation.
- Next Generation Space Telescope (NGST): Led by Dr. John Mather (Goddard Space Flight Center – GSFC), this team also has two applications. The first is to perform multiple fast reads of the charge coupled devices (CCDs) which take the telescope images in order to eliminate or reduce the effect of cosmic rays which hit these CCDs during an exposure. The second is to perform fine optical control by using a wave front sensing algorithm to control a deformable mirror.
- Orbiting Thermal Imaging Spectrometer (OTIS): This team is led by Prof. Alan Gillespie (U. of Washington). They are designing an application to take hyperspectral imaging data and retrieve temperature and emissivity, as well as performing spectral matching and unmixing, then image classification.
- Solar Terrestrial Probe Project (STP): This team, led by Dr. Steven Curtis (GSFC), is examining using fleets of spacecraft for two applications: radio astronomical imaging and plasma moment analysis.

These applications take advantage of large amounts of computing, as well as performance/power ratios that are at least an order of magnitude above those available in today's spacecraft. They are attempting to implement and test new approaches to science data processing and autonomy. They have all delivered parallel code to the REE project, and currently 8 of the 9 codes have been successfully run on an embedded cluster.

The initial applications development aims at running the applications on the testbed without fault-detection or fault-recovery. Once the applications run successfully, these topics will be addressed through the development of an applications programming guide. This will define an interface between the application and the middleware, including progress messages, error reporting, application-specified check pointing, ABFT calls, and other tools that will be developed. This is intended to be a living document, because the REE Project plans to iteratively test applications using both random and focused fault-injection, and to use the results of these experiments to modify or add to the set of tools available to the application. REE is eager to collaborate with others in examining, developing and testing these tools. The overall iterative process should drive down the number of undetected and therefore uncorrected faults to a sufficiently low number for the environment in which each mission will operate to satisfy the mission scientist and to be similar in scale to other errors that are commonly accepted, including instrument noise and transmission errors.

4. Processor in Memory Architecture for Spaceborne Computing

A second distinct path to future spaceborne computing has emerged, also as a consequence of rapid advances in semiconductor technology. While the COTS strategy leverages investment in conventional microprocessor architecture and manufacturing, a new method based on custom design may deliver improved efficiency, performance, scalability, power consumption, and reliability. The key advance, enabling this approach and justifying the implicit development cycle, is the recent semiconductor fabrication process that makes possible the integration of both DRAM cells and CMOS logic devices on the same silicon die. The possibility of a single chip incorporating both logic and memory opens up an entirely new space of computer system architecture including Symmetric Multiprocessor (SMP) on a chip, system on a chip (SOC), and processor-in-memory (PIM) [17]. SMP on a chip integrates multiple processor cores (e.g. 4) on a single die each with their associated L1 and L2 caches. An L3 cache with necessary data paths and protocol control logic provides a fully cache-coherent shared memory structure across the on-chip processors and the external interface to the separate main memory and I/O interconnect systems. SOC delivers a single chip computer with a processor core, its associated L1 and L2 caches, and a large block of DRAM main memory as well as external I/O interfaces all on a single semiconductor die. PIM tightly couples logic with the DRAM memory at the row buffer and sense amps. While advantages can be derived from all three approaches, PIM exposes the greatest opportunity for superior computing structures for future spaceborne processing. The *Gilgamesh* project, conducted by the NASA Jet Propulsion Laboratory, is developing a new generation family of parallel architectures called "*MIND*" (Memory-Intelligence-Network Devices) based on PIM for both spaceborne and ground-based high-performance computing systems [18][19]. By leveraging this new fabrication opportunity and incorporating innovative concepts in processor architecture design based on dynamic adaptive resource management methods, an important new component is anticipated that will accelerate the application of high performance in space and its use to a broader range of computational challenges on the ground.

4.1 PIM Provides Smart Memory

PIM provides smart memory as well as a means of exposing the entire internal memory bandwidth of the DRAM cell array to the processing logic. Typically a current generation DRAM acquires data in blocks of 2048 bits in a single read cycle that may take between 60 and 100 nanoseconds. PIM applies logic across the memory output row buffer to perform basic operations on all (if appropriate) data accessed. Because the logic speed is usually faster than the DRAM access time, such wide operations can be performed at the memory's maximum throughput making best use of the available bandwidth. Where such operations can be performed on-chip, without resorting to data movement off-chip through external I/O drivers, both latency and power consumption is much less than with conventional systems. Many tasks are ideal for in-memory processing and far more efficient than migrating data to external processors through their multi-layer cache hierarchy. Vector and stream processing through large blocks of memory, compound atomic operations such as synchronization primitives, and pointer chasing for manipulation of irregular data structures as well as gather-scatter operations may all be accomplished with greater efficiency by means of PIM. But there are many different approaches to exploiting PIM technology through variations in chip architecture. The Gilgamesh project is devising an advanced PIM-based architecture particularly well suited to high-performance computing with properties important to both ground-based and spaceborne data processing. Among the important advances of this architecture is the incorporation of distributed address translation mechanisms in support of virtual memory.

4.2 SIA Roadmap

The Semiconductor Industry Association tracks technology trends and projects future advances, thereby guiding systems developers as they prepare next generation product development and marketing plans. Over the next decade, it is anticipated that DRAM density will expand by two orders of magnitude while CMOS logic gate density will increase approximately by a factor of 30 to 40 in the same time frame. However, CMOS logic speeds will grow at a much more modest rate over the next ten years. Chip-wide microprocessor clock rates are expected to increase by only a factor of 3 to 4, although local clocks will be increasingly incorporated that within ten years may be as much as 3 times faster than the chip-wide clock rate.

The implication of these projections combined with the capability to couple DRAM and CMOS logic suggest that performance gains will be derived primarily through parallelism and the concurrent use of large amounts of resources rather than relying on increases in logic speed. Achieving equivalent sustained performance requires architectures that incorporate sufficient bandwidth to all resources. In addition, single processor architecture is unlikely to be able to incorporate 100 times the logic and deliver 100 times the performance. This leads to the likely conclusion that even single-chip systems of the future will embrace multiple processors. PIM architecture is one class of advanced structure that responds to anticipate technology trends and provides a scalable means of tracking those trends delivering comparable improvements in sustained performance.

4.3 The Opportunity of PIM

PIM provides a new balance in the relationship of computing resources whether used in conjunction with external conventional microprocessors or in standalone configurations. The opportunities afforded by PIM are derived principally from the possibility of close proximity of computing logic to memory row buffer on the same die. The implications of this single shift in system structure can have profound effects on the operational behavior of future systems. Specifically, the low latency, low power, and high degree of parallelism resulting from the direct connection between logic and memory buffers yields the following comparative advantages:

- Short access time to main memory through physical proximity and elimination of layers of memory hierarchy,
- Parallel computing by simultaneous access to a couple of hundred bytes and very wide ALU for concurrent operations,
- Low power consumption by greatly reducing off-chip transfers and employing very simple processor designs,
- Increased memory bandwidth by partitioning memory on-chip into multiple separately accessible blocks operating simultaneously,
- Enhanced reliability with multiple processor/memory blocks per chip allowing graceful degradation, and
- Reduced system resource contention such as memory bus and cache lines.

The change in balance of resources opens the way to new designs of processors optimized for different tradeoffs than traditional systems. Where floating point ALU throughput or pipelined execution unit utilization are emphasized in conventional processor design, PIM processor design should be optimized for effective memory bandwidth as well as for off-chip communications. Ironically, superior performance will lead to lower ALU utilization, instead emphasizing higher availability. Processors become much more simple because much of the conventional memory management hardware can be eliminated including caches in some cases. Such new designs will reverse current trends that show a continuous reduction in the effective computation per transistor.

4.4 Scalable PIM System for Spaceborne Computing

PIM technology and architecture are particularly well suited for spaceborne applications and capable of supporting space mission critical computing tasks, even on long duration flights. Several factors contribute to this. PIM offers a significant departure from conventional practices in space today as well as to the COTS cluster strategy being explored by the REE project. It provides innovation in two critical aspects of spaceborne data processing: reliability and power consumption. In addition, it provides a foundation for advanced scalable computing to support mission capabilities beyond those supported today. In particular, the ability to produce science products from raw sensor data by application of high performance computing in situ can reduce by orders of magnitude the amount of data and therefore downlink bandwidth required of spacecraft, resulting in significant reduction in overall spacecraft weight and mission cost.

PIM permits new approaches to reliability through graceful degradation of its medium and fine grain structures. Unlike typical spaceborne systems, PIM systems comprise of many processor/memory nodes even on a single chip. As failures occur, the effected nodes may be damaged and inoperable, but the remainder of the resources in the system and even the same chip can continue to perform their tasks. Thus the overall system continues to operate, albeit at a slightly reduced aggregate performance. And because the processor designs are custom, hardware mechanisms for fault detection and isolation will yield significant responsiveness in the presence of failures while retaining significant operational resources separate from the fault sites. An important consequence is the prospect for longer duration mission profiles.

Among the most important attributes of PIM-based spaceborne scalable computing is the power efficiency achieved through effective use of transistors, reduction of off-chip data transfers, and the use of all data acquired across the row buffer (when appropriate) in each single memory access cycle. But effective use of power also involves active power management strategies. The mediums to fine grain structures of PIM lend themselves to providing highly redundant resources. Such repetitive organization can permit much of the overall system to be turned off without losing operational functionality. When computing demands are small, all but a few of the processor/memory nodes comprising the total system can be deactivated, thus reducing total power consumption during these periods. For peak computing requirements, all nodes can be brought online.

4.5 Gilgamesh MIND chip

Gilgamesh (billions (GIga) of Logic Gates Assembled in a MESH) is a system incorporating MIND chips. MIND is an advanced PIM-based architecture that extends PIM functionality beyond current practices while exploiting the advantages of the basic PIM concept. The result is a physically simple architecture with a hardware-supported dynamic adaptive resource management execution model.

4.5.1 MIND Physical Architecture

The MIND chip incorporates four or more nodes, each of which constitutes a processor and memory pair. The node memory is a block of DRAM, of at least 1K rows each of 1K or more bits. A typical DRAM block employing today's fabrication technology would have a row width of 2048 bits and 1K rows. But the number of rows is likely to grow along with the number of nodes as feature size diminishes in the near future. An inter-node, intra-chip communication channel gives direct access to all node memory blocks by all node processors.

The MIND chip incorporates additional functional units that are shared among all nodes and accessed through the intra-chip communication channel. These include one or more pipelined floating-point units, a row-wide permutation network, and shared interfaces to off-chip resources.

Communication between separate MIND chips is supported by an active message protocol [15] called Parcels (PARallel Communication Element). A MIND chip has one or more parcel transport layer interfaces that accept and transmit packets between the chip and the network substrate integrating the multiple MIND chips comprising a Gilgamesh system. The network topology may vary depending on system scale and requirements. But the interfaces support both chip-to-chip direct communications performing intermediate routing of packets and chip-to-switch communications for high capability network structures and technology. A second parallel interface shared among the MIND nodes supports master/slave relationships with external intelligent devices. It is this interface that permits MIND chips to be employed within conventional systems replacing conventional *dumb* memory chips with MIND smart memory chips. Thus, a high-speed off-the-shelf microprocessor or ensemble of microprocessors can execute applications with the assistance of direct in-memory hardware support assuming libraries or compilers capable of directing the MIND chip functionality are applied. Another external parallel interface permits data streaming in to or out of the MIND chip memory blocks. This interface can be used for such high bandwidth sensors as real time CCD imagers or for streaming between chip memory and directly attached secondary storage. Finally, a set of signal lines is available for real time control of external sensors, actuators, and I/O devices that are also shared among MIND nodes.

4.5.2 The MIND Node Processor Architecture

The processor of a MIND node has a very different structure than that found with conventional microprocessors. This is due in part to its distinct relationship with its associated closely coupled memory and due to a different optimization metric. While typical microprocessors are optimized for high utilization of their integer and floating-point arithmetic units, the MIND processor architecture is devised primarily to maximize the effective throughput of its associated memory block, and secondarily to maximize the chip external interfaces. Because the processors themselves incorporate a relatively modest proportion of the overall chip area (typically less than 20%) with the memory area exceeding half of the chip, availability rather than utilization is an important operational property of the processors.

Each processor comprises four major subsystems:

1. memory controller interface
2. row-wide ALU
3. wide register bank
4. parcel handler

The memory controller queues up memory requests, performs accesses to addressed memory rows, conducts simple compound atomic operations, and achieves memory cell refresh and error detection and correction. The row-wide ALU is as wide as the row buffer or some subintegral length that permits the equivalent of parallel operations on potentially all data acquired in a single memory access cycle. It includes Boolean and binary integer logic as well as a permutation network for data displacement and alignment within the row wide buffers. The row-wide ALU includes instructions which when performed in sequence efficiently perform floating-point operations in a few logic cycles.

The wide register bank provides a bank of registers, each of which is row-wide in length, or a block of registers subintegral in length, the total sum of which is equal to a full row buffer. These wide registers are managed by means of dynamic naming and are employed for myriad tasks more diverse than those of conventional microprocessors. For example, wide registers can be used as conventional banks of word width registers, instruction caches, and translation look aside buffers, message packet buffers, vector registers, and temporary memory row buffers.

The parcel handler is the logical interface between the MIND node and other nodes on remote chips. It accepts from and delivers to the global Gilgamesh system parcel packets that carry information about actions to be taken and data upon which to perform them. The parcel handler may fully satisfy the requirements of an incident parcel or pass it on to higher functionality elements of the node such as the memory controller or the thread execution scheduler for further processing. It also accepts a parcel from the register bank and wraps it in a transport packet for transmission.

4.5.3 MIND Execution Model

The MIND architecture, although simple in physical design, provides a sophisticated execution model. Key characteristics of the supported low-level computation methodology are:

- virtual memory
- multithreaded
- message driven
- object based
- dynamic resource allocation

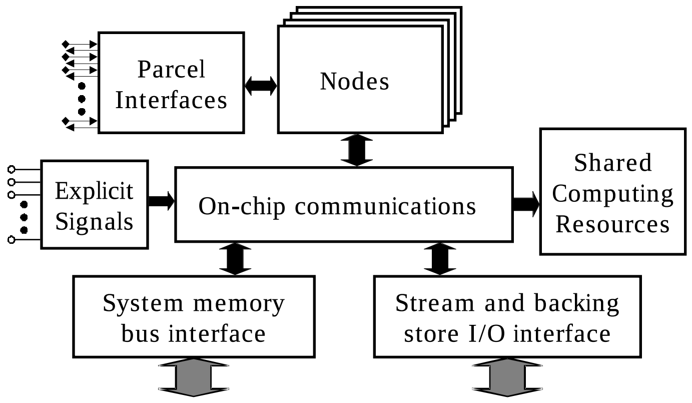
All information is organized in virtual pages [16] which may be allocated anywhere in the Gilgamesh system and may be moved as memory requirements and load balancing demands which evolve over time. The MIND architecture provides a distributed memory mapping and address translation scheme with supporting mechanisms. Virtual memory, which includes the virtual naming of scheduled tasks, enables the implementation of a protective abstract interface between changes in operational structure of the physical system due to faults and reconfiguration for power management and the parallel program execution, eliminating static resource allocation for data and tasks. But equally importantly, virtual memory handling allows the management of irregular data structures. In such applications, the data structures include entries for pointers or links to other parts of the structure (such as a sparse matrix or semantic net) that are virtual addresses. The distributed virtual to physical address translation scheme involves several stages and distinct support mechanisms. The supervisor virtual address space includes an active node table and the address translation directory table. Every physical node that is active is responsible for part of the directory table as defined by the active node table. Preferential placement of pages at the same site as the relevant directory table entries allows direct one-hop access to the destination virtual page. However, a single level of indirection from the respective directory table entry supports random placement of pages. Within a particular MIND chip incorporating several nodes, pages may be placed anywhere and on-chip mapping tables provide specific assignment information. Wide-registers in combination with the wide-ALU are used as generalized TLBs for rapid lookup (within a single logic cycle) for fast address translation. Because MIND handles virtual pointers, it can perform many irregular structure manipulations including tree walking, parallel prefix, and gather-scatter operations.

Multithreading permits separate threads of execution to be active concurrently, sharing logical, communications, and memory resources with efficient context switching mechanisms [14]. When ordinarily multithreading is in the architecture of a processor, it is used to hide memory access latency. Gilgamesh uses multithreading as a unifying resource management mechanism to enable simultaneous operation of the execution unit, memory unit, functional units, and external communications. It also provides a cycle-by-cycle task prioritization for real time response. Multithreading hides the on-chip latencies for accesses to the shared functional units and the memory blocks of other nodes on the same MIND chip.

To mitigate the potential deleterious affects of latencies between MIND chips, a split transaction parcel protocol for message driven computing is supported. Node computing resources are not allocated until a parcel arrives. Therefore, they are employed for other local tasks and are not wasted waiting for some round-trip reply. This approach may be employed to support object-based local execution where state exists between successive parcels. As such state is stored in memory this does not consume critical (e.g. register banks) physical resources.

Resources are allocated dynamically by the runtime system. By means of parcels, tasks are generally moved to the data rather than moving the data to the tasks. The data itself may be distributed for uniformity across the system or may be aligned with other data structures shared by the tasks. This is possible because both data pages and task identifiers are virtual and can be assigned to any physical resource (node). Dynamic allocation supports power management by reconfiguration and fault tolerance through graceful degradation, as previously discussed.

MIND Chip Architecture



Parcel Interfaces
Nodes
Explicit Signals
On-chip communications
Shared Computing Resources
System memory bus interface
Stream and backing store I/O interface

5. Conclusions

A new domain of opportunities and challenges for spaceborne computing is opening, with the potential to greatly extend the scope of science conducted by unmanned spacecraft throughout the solar system. The application of high-performance computers exhibiting exceptional size, weight, and power will revolutionize the nature of deep space missions resulting in greatly reduced cost and opening the way to new classes of missions unattainable through today’s conventional practices. Critical to the success of any new strategy is the ability to operate over long duration in the presence of faults. NASA JPL has undertaken two advanced development projects to provide the necessary hardware and software technologies to enable such future missions. The REE project is exploring the application of Beowulf-like clusters of embedded COTS devices, benefiting from the cost and reliability advantages derived from the mass market computing industry. The Gilgamesh project is developing custom processor-in-memory architectures to exploit new technology opportunities for unprecedented power and performance efficiency in a scalable architecture. These strategies complement each other and together ensure that NASA will have the needed computing capability in space throughout the coming century.

The REE project requires that dependability be provided through software. Extensive error detection and recovery services are provided to the science applications through a variety of mechanisms including check pointing, TMR, and ABFT-enabled scientific subroutine libraries. The applications are built and tested without fault-tolerance features, and then modified to use substantially off-the-shelf fault-tolerance components. An initial application manager is being used by REE, controlled by the prototype REE system software. Through the application manager and system software, the application is able to tolerate process failures, process hangs, and node failures. The ABFT-enabled libraries are essentially transparent to the application and provide high fault coverage of the mathematical routines themselves, though not of logical or arithmetic codes outside the library routines. Additional fault detection strategies will be required to protect the remainder of the application codes, the node operating system and system software. The REE Project will culminate in the development of a final flight-capable prototype hardware/software system during the 2005 time frame.

The Gilgamesh architecture brings virtual memory, multithreading, and message driven computing to the realm of PIM technology to provide an innovative class of computing structures uniquely suited to the challenges of long-duration deep-space science missions. Power management and fault tolerance are supported through this fine grain architecture where multiple processor-memory nodes reside, even on a single chip, and many such chips operate cooperatively through an advanced high-level active message communication protocol. The Gilgamesh project is implementing an FPGA prototype in 2001 and has targeted a second generation fully operation integrated circuit incorporating sixteen processor-memory nodes to be delivered for validation and testing in 2004.

6. Acknowledgements

This research was carried out at the Jet Propulsion Laboratory (JPL), California Institute of Technology, and was sponsored by the National Security Agency (NSA) and the National Aeronautics and Space Administration (NASA). The Defense Advanced Research Projects Agency (DARPA) and NSA provided additional funding for early research on PIM. Important contributors to related work include the University of Delaware, University of Southern California Information Sciences Institute, and the University of Notre Dame as well as the California Institute of Technology Center for Advanced Computing Research.

7. References

[1] R. Ferraro, "NASA Remote Exploration and Experimentation Project," (http://www-ree.jpl.nasa.gov/).

[2] R. Ferraro, R. R. Some, J. Beahan, A. Johnston, and D S. Katz, "Detailed Radiation Fault Modeling of the REE First Generation Testbed Architecture," *Proc. of 2000 IEEE Aerospace Conf.*

[3] T. Sterling, J. Salmon, D. Becker, and D. Savarese, *How to Build a Beowulf*, The MIT Press, 1999.

[4] Myrinet is a class of products of Myricom, Inc. (http://www.myricom.com/).

[5] Lynx OS is a product of LynuxWorks (http://www.lynuxworks.com/).

[6] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra, *MPI: The Complete Reference*, The MIT Press, 1996.

[7] K.-H. Huang and J. A. Abraham, "Algorithm-Based Fault Tolerance for Matrix Operations," *IEEE Trans. on Comp.*, 33(6):518-528, 1984.

[8] M. Turmon and R. Granat, "Software-Implemented Fault Detection for High-Performance Space Applications," submitted to *IEEE Trans. on Comp.*, 2000.

[9] M. Blum and S. Kannan, "Designing programs that Check their Work," *J. of the Assoc. of Comp. Mach.*, 42(1):269-291, 1995.

[10] R. A. van de Geijn, P. Alpatov, G. Baker, and C. Edwards, *Using PLAPACK: Parallel Linear Algebra Package*, The MIT Press, 1997.

[11] M. Frigo and S. G. Johnson, *FFTW*, http://www.fftw.org/.

[12] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. Duff, "A Set of Level 3 Basic Linear Algebra Subprograms," *ACM Trans. Math. Soft.*, 16(1):1-17, March 1990.

[13] J. A. Gunnels, D. S. Katz, E. S. Quintana-Ortí, and R. A. van de Geijn, "Fault-Tolerant High-Performance Matrix Multiplication: Theory and Practice," to appear in *Proc. of the IEEE Int. Conf. on Dependable Systems and Networks*, 2001.

[14] J.B. Brockman, P.M. Kogge, V.W. Freeh, S.K. Kuntz, and T.L. Sterling. Microservers: A New Memory Semantics for Massively Parallel Computing. *Proc. ACM Int. Conf. on Supercomputing* (*ICS'99*), June 1999.

[15] W.J. Dally, *et al*. The Message Driven Processor: A Multicomputer Processing Node with Efficient Mechanisms. *IEEE Micro*, 23-28, 1992.

[16] M. Hall, J. Koller, P. Diniz, J. Chame, J. Draper, J. LaCoss, J. Granacki, J. Brockman, A. Srivastava, W. Athas, V. Freeh, J. Shin, and J. Park. Mapping Irregular Applications to DIVA, a PIM-Based Data Intensive Architecture. *Proceedings* (*SC'99*), November 1999.

[17] P.M. Kogge, T.Sunaga, H. Miyataka, K. Kitamura, and E. Retter. Combined DRAM and Logic Chip for Massively Parallel Systems. *16th Conf. on Adv. Res. in VLSI*, 1995.

[18] P.M. Kogge, S.C. Bass, J.B. Brockman, D.Z. Chen, and E. Sha. Pursuing a Petaflop: Point Designs for 100 TF Computers Using PIM Technologies. *Frontiers of Massively Parallel Computation*, October 1996.

[19] L. Bergman and T. Sterling, "A Design Analysis of a Hybrid Technology Multithreaded Architecture for Petaflops Scale Computation." International Conference on Supercomputing, Rhodes, Greece, June 1999.