ParalleX: A Study of A New Parallel Computation Model

Guang R. Gao[1], Thomas Sterling[2,3], Rick Stevens[4], Mark Hereld[4], and Weirong Zhu[1]

[1]Department of Electrical and Computer Engineering
University of Delaware
{ggao,weirong}@capsl.udel.edu

[2]Center for Advanced Computing Research
California Institute of Technology
tron@cacr.caltech.edu

[3]Department of Computer Science
Louisiana State University
tron@cct.lsu.edu

[4]Mathematics and Computer Science Division
Argonne National Laboratory
{stevens,hereld}@mcs.anl.gov

Abstract

This paper proposes the study of a new computation model that attempts to address the underlying sources of performance degradation (e.g. latency, overhead, and starvation) and the difficulties of programmer productivity (e.g. explicit locality management and scheduling, performance tuning, fragmented memory, and synchronous global barriers) to dramatically enhance the broad effectiveness of parallel processing for high end computing. In this paper, we present the progress of our research on a parallel programming and execution model - mainly, ParalleX. We describe the functional elements of ParalleX, one such model being explored as part of this project. We also report our progress on the development and study of a subset of ParalleX - the LITL-X at University of Delaware. We then present a novel architecture model - Gilgamesh II - as a ParalleX processing architecture. A design point study of Gilgamesh II and the architecture concept strategy are presented.

1 Introduction

Historically as technology has advanced, computer architecture has changed to exploit the new opportunities offered and to compensate for the exposed weaknesses. Alternative strategies for organizing the computation as well as the architectural structures and system software have been devised to provide governing semantic principles upon which such architecture and programming models are based. For more than a decade, the dominant model of computation has been the communication sequential process or more commonly the “message passing model” represented by various implementations of MPI (e.g. MPICH-2, OpenMPI) adopted because of its applicability to microprocessor based MPPs and commodity clusters. Other models used are multiple threads (e.g. OpenMP), vector, and SIMD, but message passing is dominant. However, as semiconductor technology has continued to evolve, both new opportunities and new challenges have emerged demanding corresponding improvements to architecture. Most pronounced is the move to multicore components and the re-emergence of heterogeneous computing elements such as GPUs and Clearspeed SIMD [2] attached processors. The IBM Cell architecture embodies both heterogeneous and multicore elements [6]. These exemplify but do not fully reflect the increasing need for new programming methods and architecture structures to continue to fully benefit from Moore’s Law. An important objective of this research project has been the exploration and development of a possible new execution model that will more readily employ the potential capabilities of near term semiconductor technologies while easing the programmer burden. Such an execution model could lead to new programming models and languages, new parallel computer architecture, and new supporting system software.

In this paper, we present the progress of our research on a parallel programming and execution model - mainly, *ParalleX*. We describes the functional elements of ParalleX, one such model being explored as part of this project. ParalleX principal elements are discussed, including locality, global name space, multithreading, parcels, local control objects, percolation, echo, and parallel processes. We also report our progress on the development and study of a subset of ParalleX - the LITL-X at University of Delaware. We then present a novel architecture model - Gilgamesh II - as

1-4244-0910-1/07/$20.00 ©2007 IEEE.

a ParalleX processing architecture. A design point study of Gilgamesh II is presented and the architecture concept strategy is presented.

2 Parallel Programming Execution Model: ParalleX and LITL-X

This section discusses the specific requirements for a new execution model and describes the functional elements of *ParalleX*, one such model being explored by Sterling's group at LSU as part of this project. We also describe *LITL-X* a subset of ParalleX that is being developed and studied at Gao's group at Delaware.

2.1 Requirements for a New Model of Computation

A new model of computation is required to meet the challenges of emerging technologies. Such a model must serve as a discipline to govern future scalable system architectures, programming methods, and runtime and operating system software. Among its most important requirements is to address the dominant factors impeding significant improvements to efficiency. Among these are latency, overhead, starvation, resource contention, and programmability. Latency is the time delay, measured in processor clock cycles required to access remote data or services such as local main memory or remote nodes. Overhead is the critical path work required to manage parallel physical resources and concurrent abstract tasks. Overhead can determine the scalability of a system and the minimum granularity of program tasks that can be effectively exploited. Starvation is the lack of work and therefore the idle cycles experienced by an execution site (e.g. processor core) caused either due to inadequate program parallelism or due to poor load balancing. Contention for shared resources causes delays while one requesting execution site is blocked by another accessing the same needed resource. These can include shared communication channels, common memory banks, or other mutually accessible resource. Programmability relates to programmer productivity. Systems that require direct and explicit resource management by the programmer to achieve optimized performance can be much more difficult to use than those machines with mechanisms more naturally suited to effectively exploit the function elements thus reducing programmer intervention. All of these efficiency factors can benefit from improved programming models and parallel architecture requiring a model of computation that incorporates the means and facilitates the methods to make effective use of the critical resources.

Other demands on the execution model include such practical concerns as power consumption, reliability, and the size of the large scale machines. While these would appear to be properties of the bulk hardware, the manner of its use is dictated by the execution model and therefore for a given sustained performance can be significantly affected by it. A new computational model is also needed to move towards a new balance of resources. To devise structures and coordinate computing around the realistic precepts demands a model of computing quite distinct from those conventionally employed by the mainstream today. This will enable the exploitation of the true performance opportunities yielded by technology and support architecture advances.

The following are considered essential characteristics of a future model of computation:

- System wide global name spaces both for data and active tasks with efficient address translation and communication routing in the presence of dynamic object distribution,
- Rich parallelism semantics and granularity to exploit a diversity of forms to make available the million to billion way parallelism that will be required of near nanoscale systems by the end of the next decade,
- Direct support for lightweight processing of irregular time-varying sparse data structure parallelism such as that for trees (N-body codes), directed graphs (adaptive mesh refinement, semantic nets), and particle in cell (magneto hydro dynamics),
- Intrinsic mechanisms for automatic latency hiding, to mitigate this major source of performance degradation due to blocking on completion of remote actions and the ensuing idle times experienced,
- Incorporation of low overhead mechanisms for managing global system parallelism including synchronization, scheduling, data movement, and load balancing, and
- Affinity semantics to establish relationships that would lead to locality opportunities through both compile time and runtime techniques.

Addressing these challenges through new strategies and structures of computation would remove the reliance on manual, pains taking, and error prone direct user intervention including direct control of hardware mechanisms, direct management and allocation of hardware resources, and direct choreographing of physical data and task locality. As a consequence, efficiency and performance scaling could be dramatically higher, systems smaller and lower power, and programming dramatically easier. Towards this end, ParalleX is being devised to provide an alternative framework from conventional methods and currently addresses many (but not all) of these requirements.

2.2 Principal Elements of ParalleX

An execution model is neither a programming model (although it influences it) nor a virtual machine (although it is often treated as such) but rather a set of guiding principles that organize the computation and govern the relationships between the vertical layers of the system stack (i.e. application, language, compiler, runtime system, operating system, system architecture, core architecture, and hardware mechanisms). An execution model specifies referents, their interrelationships and actions that can be performed on them. In so doing, it defines the semantics of state objects, functions, parallel flow control, and distributed interactions. Curiously, an execution model intentionally leaves some implementation policies unspecified although defining their boundary conditions. Technology, structure, and mechanism are all left open to permit multiple realizations of the model depending on engineering constraints. Even policies of scheduling and dispatch are incomplete to enable optimizations through disparate means. Such a model should have efficient realizations on a range of system classes from conventional MPPs to unique new designs of truly parallel computer architectures. Such a model enables reasoning about the decision chain through the vertical striation for resource scheduling and action performing. It influences design decisions for programming languages and computer architecture as well as the intervening system software. ParalleX is an experimental execution model, synthesizing a number of important concepts with prior art in a unique semantic ensemble to address many of the critical challenges expressed above.

ParalleX is an asynchronous parallel computing model with a partitioned global address space. It exploits a split-phase multithreaded transaction distributed computing methodology that decouples the computation and communication for overlapping and moves the work to the data when this is preferable to just moving the data to the work as is conventionally done. The message driven paradigm combined with multithreading and an advanced prestaging technique referred to as "percolation" provides intrinsic latency hiding at multiple levels within the system. Lightweight synchronization constructs, when implemented efficiently, deliver a diversity of forms and scale of parallelism. The global name space permits efficient direct manipulation of data and task objects for dynamic control of abstract and physical resources as well as ease of user programming. The combination of message driven multi threaded execution with lightweight synchronization offers a powerful semantic execution framework and an alternative to the conventional communicating sequential processes method that has dominated much of the last two decades. The basic semantic mechanisms of ParalleX are briefly described below:

Locality: Like many models dealing with distributed computing, ParalleX recognizes a local physical domain. In the case of ParalleX, it is the locus of resources that can be guaranteed to operate synchronously and for which hardware can guarantee compound atomic operations on local data elements. While a typical example may be a conventional MPP node, it could just as easily be a small part of a single chip as suggested by the Gilgamesh-II architecture discussed in the next section. Within a locality, all functionality is bounded in space and time permitting scheduling strategies to be applied and certain pathological cases like race conditions to be precluded.

Global name space: Similar to the emerging class of PGAS programming models (e.g. CAF [11], Titanium [14], UPC [1]) and certain parallel architectures (e.g. Cray T3E, SGI Origin), it allows any first class object to be remotely identified efficiently through a hierarchical naming structure. In ParalleX, actions as well as data are first class entities to permit direct control of the computation by the computation. Also, hardware resources have their own names (typed) and therefore can be referenced to a limited degree by the software, again to support direct manipulation of the relationship between abstract and physical resources both by the runtime system and the hardware architecture when supported.

Multithreaded: A thread is a partially ordered set of interrelated basic operations. In ParalleX, a thread is ephemeral and serves a single locality. More than one thread may be active concurrently within the domain of a given locality. Threads can be near fine grain and are not limited to an SPMD model with the same thread operating on different localities. Internal operations of threads are represented as static dataflow ordering but may be mapped in to a single sequential instruction stream through the compiler back end. Threads can suspend or terminate when a remote access is required. If suspending, a local control object (see below) is created from its state. If terminating, a parcel (see below) is constructed and dispatched to the destination remote data where a new thread is invoked thus moving the work, in essence, to the data.

Parcels: ParalleX employs a message driven paradigm for asynchronous distributed operation. A parcel includes a destination virtual address of a remote target object and an action specifier defining a task to be applied to that object. Additional argument values can be carried by the parcel to move prior state to the site of the invoked thread execution. Parcels differ from other such constructs such as active messages in that it also carries a "continuation specifier" that defines what happens after the specified action is completed. This allows the locus of control to migrate across the distributed system. Message-driven computing through parcels allows physical resources (execution locality) to operate via a work queue model. It largely circumvents idle cycles due to blocking on remote access delays.

Local Control Objects (LCO): A rich set of synchronization primitives is provided to facilitate lightweight control and exploit a diversity of parallelism. LCOs eliminate most uses of global barriers greatly freeing the dynamic adaptive flexibility of parallel processing and relaxing the over constraining operation imposed by barriers. Dataflow synchronization, futures, and meta-threads are examples of the kind of LCOs that can be employed. They can standalone or be incorporated in larger data structures to support coordinated multi access to shared global data structures like directed graphs for knowledge management. Dataflow constructs allow true asynchronous value oriented flow control determined at compile time. Futures permit anonymous producer-consumer computing. "Depleted threads" (our term) provide a kind of temporary state storage for suspended threads.

Percolation: ParalleX provides a mechanism for moving work (both state and task descriptions) to unused parts of the system through a mechanism referred to (by us) as "Percolation" which was devised as a latency hiding mechanism as well. For a precious resource, overhead and latency can greatly degrade system efficiency. Percolation (developed by Sterling and Gao as part of the HTMT Project) is a workflow strategy that employs ancillary mechanisms to prestage data and tasks in high speed memory near the high cost compute elements when a task is to be performed. This is a variation of parcels but used with hardware as the target rather than abstract data objects. Prefetching is also a form of prestaging but performed by the compute element itself, thus imposing the overhead burden, and possibly the impact of latency, on it as well.

Echo: ParalleX does not assume cache coherency outside of the domain of the locality even though it has a global name space. When a writable variable is to be used by many separate execution points during the same temporal interval, ParalleX may assert a copy semantics called (again by us) "echo". This construct identifies the tree of equivalent locations all of which are to be operated upon as if a single value. It is inspired by location consistency (developed by Gao), although it employs a different model. Echo is a split phase operation. Using it requires that a thread defer committing side effects until it gets an acknowledgement that the value it used is the current one. This permits overlap between coherency verification and continued computation with the latest known value, thus reducing the apparent latency and increasing the available parallelism.

Parallel Processes: ParalleX differs from conventional distributed computing languages in that the notion of parallel processes is not just that there may be multiple processes being performed concurrently, but rather that each process may have many parts, either subprocesses or threads, running concurrently (or in parallel) as well and distributed across many execution sites. Parallel Processes can be object oriented in that once instantiated they can have additional messages incident upon them invoking methods to create new instances in the form of threads (single locality) or processes (multiple localities).

2.3 LITL-X: A Subset of Parallel-X On A Cellular Parallel Computer Architecture

As a part of our effort in studying the principles outlined in ParalleX in the previous section, we have moved on studying LITL-X - a subset of ParalleX - through a prototype design and implementation effort at University of Delaware. To be more concrete, We are working on a prototype programming API, LITL-X (pronounced "little-X") (previously under the nick name Latency Intrinsic-Tolerant Language), which provides the application programmers with a powerful set of semantic constructs to organize parallel computations in a way that hides/manages latency and limits the effects of overhead. This is quite different from locality management, although the intent of both strategies is to minimize the effect of latency on the efficiency of computation. Locality management attempts to *avoid* latency events by aggregating data for local computation and reducing large message communications. Latency management attempts to *hide* latency by overlapping communications with computation. A number of experimental languages exhibiting these attributes, at least to some degree, have been developed (e.g. Split-C [9], EARTH-C [7], SISAL [10], and UPC [1]).

A version of LITL-X will be developed by extending the TNT - a courase-grain thread layer as described elsewhere [3]. Under LITL-X, we are adding the following classes of parallel constructs to TNT for latency tolerance and data movement overhead management:

- Extend the thread model with the ability to launch and manage *asynchronous calls*, a feature that has been studied in the past at Delaware (under the EARTH model [13]) or elsewhere such as Cilk [4].

- *Percolation* [8] of program instruction blocks and data at the site of the intended computation, to eliminate waiting for remote accesses, which are determined at run time prior to actual block execution.
- *Synchronization constructs* for data-flow style operations, leveraging our past studies on EARTH [13].
- *Atomic sections* [12], a parallel programming construct that can simplify the use of fine-grained synchronization, while delivering scalable parallelism by using a weak memory consistency model, such as *location consistency* [5].

LITL-X is not intended as a final programming language for end users, but rather a logical testbed to prototype a set of promising concepts and to test their impact on system performance and efficiency.

3 Gilgamesh II: Towards a New ParalleX Processing Architecture

While ParalleX may be able to support computing on conventional platforms with superior operation for some classes of problems, one of its strengths is that it suggests innovative parallel computer architectures that may make exceptionally good use of future semiconductor technology through the implementation of execution elements designed explicitly for low power and parallel cooperative execution. As part of this and other ongoing projects, a point design, Gilgamesh II, is being devised and evaluated, in part, to validate the ParalleX execution model. Here the basic concept of the Gilgamesh parallel architecture point design is described.

3.1 Design Point Technology

It is unlikely that any truly innovative ideas will find their way in to supercomputers of the next few years for the simple reason that the design cycle of systems can be as long as 7 years and routinely a major vendor will have two design generations underway and overlapped at the same time. For this reason, a longer timeline is assumed and a technology target date of 2020 is selected.

3.2 Architecture Concept Strategy

The concepts of Gilgamesh II architecture reflect the requirements described above and the ParalleX model of computation. It strives to align supercomputer architecture with the cost imperatives implicit in the technology evolution. ALUs should become ubiquitous and operated at high availability rather than utilization to support other more precious resources. Memory access latency should be minimized and then hidden to achieve the highest memory throughput possible for a given application. System performance is likely to be limited by memory capacity and system wide communications bandwidth and latency. Architecture should address these key challenges.

ParalleX is the execution model around which the Gilgamesh architecture design point is derived. The architecture incorporates hardware structures to support most of the mechanisms described before to minimize overhead, maximize parallelism, and provide scalability while bounding power consumption. It uses the message-driven split-phase multithreaded transaction processing paradigm for latency hiding and exploiting of near fine grain parallelism through the use of in-memory synchronization objects. As shown in Figure 1, the architecture is heterogeneous with two computing structures designed to operate best at the two modalities of operation determined by degree of temporal locality. At high temporal locality (where cache hit rates would be highest on conventional processors) a streaming architecture based on dataflow control concentrates many ALUs interconnected via local registers and 4-way multiplexers to provide high operation count for modest data rates. At low (or no) temporal locality (where cache hit rates would be very poor) an advanced Processor in Memory architecture called "MIND" has been developed to provide short latencies and very high memory bandwidth with in-memory threads. Like ParalleX, the architecture provides hardware support including address translation for a global name space without cache coherence and in support of the Echo copy semantics. A single building block element is used to build up this highly parallel system. A peak performance in excess of 1 Exaflops is achievable with 100K chips. Each Gilgamesh chip is a heterogeneous multicore subsystem with a dataflow accelerator and 16 PIM modules, each with 32 MIND nodes. Each chip is capable of approximately 10 Teraflops although the theoretical peak is substantially higher. While the main memory of the system is provided by the MIND modules, a DRAM backing store referred to (by us) as the "Penultimate Store" is included on an additional 100K chips for a total memory storage of 4 Petabytes. The system is assumed to be connected by the innovative Data Vortex network (invented by Coke Reed, Interactics Holding). This design point is under investigation as one possible future architecture exploiting the advanced methods provided by the ParalleX execution model.

4 Acknowledgments

We acknowledge the support from the National Science Foundation (CNS-0509332) and the U.S. Dept. of Energy under Contract DE- AC02-06CH11357, IBM, ETI, and other government sponsors. We would also like to acknowl-

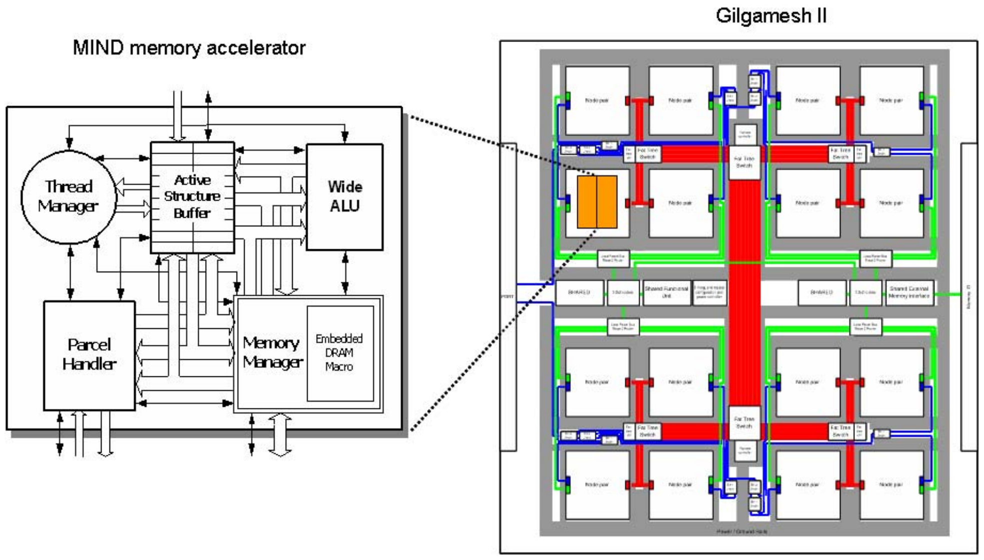

Figure 1. Gilgamesh II: A New ParalleX Processing Architecture

edge other members at the CAPSL group, who provide a stimulus environment for scientific discussions and collaborations.

References

[1] W. W. Carlson, J. M. Draper, D. E. Culler, K. Yelick, E. Brooks, and K. Warren. Introduction to upc and language specification. *CCS-TR-99-157*, May 13 1999.

[2] ClearSpeed. ClearSpeed CSX600. `http://www.clearspeed.com/`.

[3] J. del Cuvillo, W. Zhu, Z. Hu, and G. R. Gao. TiNy Threads: A thread virtual machine for the Cyclops64 cellular architecture. In *Fifth Workshop on Massively Parallel Processing*, Denver, Colorado, USA, April 2005.

[4] M. Frigo, C. E. Leiserson, and K. H. Randall. The implementation of the Cilk-5 multithreaded language. In *Proceedings of the ACM SIGPLAN 1998 Conference on Programming Language Design and Implementation*, 1998.

[5] G. R. Gao and V. Sarkar. Location consistency, a new memory model and cache consistency protocol. *IEEE Transactions on Computers*, 49(8):798–813, August 2000.

[6] M. Gschwind, H. P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, and T. Yamazaki. Synergistic processing in Cell's multicore architecture. *IEEE Micro*, 26(2):10–24, 2006.

[7] L. J. Hendren, X. Tang, Y. Zhu, G. R. Gao, X. Xue, H. Cai, and P. Ouellet. Compiling C for the EARTH multithreaded architecture. In *Proceedings of the 1996 Conference on Parallel Architectures and Compilation Techniques (PACT '96)*, pages 12–23, Boston, Massachusetts, October 1996.

[8] A. Jacquet, V. Janot, R. Govindarajan, C. Leung, G. Gao, and T. Sterling. Executable performance model and evaluation of high performance architectures with percolation. Technical Report 43, Newark, DE, Nov. 2002.

[9] A. Krishnamoorthy, D. Culler, A. Dusseau, S. Goldstein, S. Lumetta, T. von Eicken, and K. Yelick. Parallel Programming in Split-C. In *Supercomputing '93 Proceedings*, pages 262–273. IEEE Computer Society Press, November 1993.

[10] J. McGraw, S. Skedzielewski, S. Allan, O. Oldehoeft, J. Glauert, C. Kirkham, B. Noyce, and R. Thomas. *SISAL: Streams and iteration in a single assignment language, language reference manual version 1.2*. Lawrence-Livermore-National-Laboratory, Mar. 1985.

[11] J. Reid. Co-array Fortran for full and sparse matrices. *Lecture Notes in Computer Science*, 2367, 2002.

[12] V. Sarkar and G. R. Gao. Analyzable atomic sections: Integrating fine-grained synchronization and weak consistency models for scalable parallelism. *CAPSL Technical Memo 52*, Feb 9th 2004.

[13] K. B. Theobald. *EARTH: An Efficient Architecture for Running Threads*. PhD thesis, May 1999.

[14] K. Yelick, L. Semenzato, G. Pike, C. Miyamoto, B. Liblit, A. Krishnamurthy, P. Hilfinger, S. Graham, D. Gay, P. Colella, et al. Titanium: A high-performance Java dialect. *Concurrency Practice and Experience*, 10(11-13):825–836, 1998.