Preliminary Design Examination of the ParalleX System from a Software and Hardware Perspective

Alexandre Tabbal
Department of Electrical Engineering
LSU, Baton Rouge, LA, USA
atabbal@ece.lsu.edu

Matthew Anderson
Department of Physics and Astronomy
LSU, Baton Rouge, LA, USA
matt@phys.lsu.edu

Maciej Brodowicz
Hartmut Kaiser
Thomas Sterling
Center for Computation and Technology
LSU, Baton Rouge, LA, USA
{maciek,hkaiser,tron}@cct.lsu.edu

ABSTRACT

ExaScale systems, expected to emerge by the end of the next decade, will require the exploitation of billion-way parallelism at multiple hierarchical levels in order to achieve the desired sustained performance. While traditional approaches to performance evaluation involve measurements of existing applications on the available platforms, such a methodology is obviously unsuitable for architectures still at the brainstorming stage. The prediction of the future machine performance is an important factor driving the design of both the execution hardware and software environment. A good way to start assessing the performance is to identify the factors challenging the scalability of parallel applications. We believe the root cause of these challenges is the incoherent coupling between the current enabling technologies, such as Non-Uniform Memory Access of present multicore nodes equipped with optional hardware accelerators and the decades older execution model, i.e., Communicating Sequential Processes (CSP). Supercomputing is in the midst of a much needed phase change and the High-Performance Computing (HPC) community is slowly realizing the necessity for a new design dogma, as affirmed in the preliminary ExaScale studies. In this paper, we present an overview of the ParalleX execution model and its complementary design efforts at the software and hardware levels, while including power draw of the system as the resource of utmost importance. Since the interplay of hardware and software environment is quickly becoming one of the dominant factors in the design of well integrated, energy efficient, large-scale systems, we also explore the implications of the ParalleX model on the organization of parallel computing architectures. We also present scaling and performance results for an adaptive mesh refinement application developed using a ParalleX-compliant runtime system implementation, HPX.

Keywords

Execution Model, ParalleX, Model of Computation

1. INTRODUCTION

An entire class of parallel applications is emerging as scaling-impaired. These are simulations that consume extensive execution time, sometimes exceeding a month, but which are not able to use effectively more than a few hundred processors. Some examples are modeling colliding neutron stars to simulate gamma ray bursts in numerical relativity and simultaneously identifying the gravitational wave signature for detection with such massive instruments as LIGO (Laser Interferometer Gravitational Observatory). These codes are based on Adaptive Mesh Refinement (AMR) algorithms to concentrate processing effort at the most dynamic parts of the computation space at any one time. However, today's conventional parallel programming methods such as MPI [12] and systems such as distributed memory MPPs and Linux clusters exhibit poor efficiency and constrained scalability for this class of applications. This severely hinders scientific advancement. Many other classes of applications exhibit similar properties. A new execution model and programming methodology is required [11] to achieve dramatic improvements for such problems and prepare them for efficient exploitation of Petaflops systems comprising millions of cores. This paper briefly presents such a model, ParalleX (PX), and provides early results from an experimental implementation of an AMR application simulating a semilinear wave. The AMR implementation is based on the HPX runtime system that suggests the future promise of the PX strategy.

1.1 Challenging Performance Factors

1.1.1 Sources of performance degradation

The bottlenecks to the effective use of new generation HPC systems include: (**S**)tarvation due to lack of usable application parallelism and means of managing it; (**L**)atency from remote access across the system or to local memories; (**O**)verheads damaging strong scalability, efficiency, and hampering effective resource management; (**W**)aiting for contention resolution due to multicore chip I/O pins, memory banks, and system interconnects ($SLOW$ [16]). ParalleX model addresses these issues by providing explicit mechanisms that offset, mask or counteract the effects of these phenomena.

1.1.2 Power

As explained in [5], power usage is the main factor limiting scalability of computer systems. Part of the problem has its origin in the laws of physics, yet another major part is due to the design *legacy* methodology of computer systems. Complex hardware and software designs burn too much extra energy not directly related to the computation (e.g., caches, speculative execution, standard software stacks). Simplicity is fundamental to the execution model philosophy we are embracing; simpler designs are expected to conserve power through elimination of expensive implementations with diminishing returns on the investment in silicon area and its related power usage. In addition, reduction of complexity of the computing system stack is not only welcomed, but necessary to the design of feasible future execution platforms. To better assess the power consumption of the PX mechanisms, we augmented a reconfigurable cluster (multicore SMP nodes with FPGAs) with AC and DC power meters that allow us to estimate the energy distribution in the system.

1.1.3 Productivity

Productivity is a major concern for the HPC community not only as the throughput of the machine in terms of completed jobs, but also the ability of users in utilizing the computing systems. This is primarily related to the exposed application programming interface (API) that systems and applications programmers use to realize the work they are trying to accomplish.

The PX model [9] has been developed to address these challenges by enabling a new way of computation based on message-driven flow control in a global address space coordinated by lightweight synchronization elements. This paper describes the PX model and presents a runtime system architecture that delivers the mechanisms required to support the parallel execution, synchronization, resource allocation, and name space management.

2. THE PARALLEX EXECUTION MODEL

An execution model [of computation] is a set of governing principles that guide the co-design, function, and interoperability of all layers of the system structure from the programming model through the system software to the hardware architecture. The trajectory of HPC systems over the last six decades has been a sequence of phases of continued incremental improvement demarcated by abrupt transitions in fundamental paradigms. Such phases include vector computing, SIMD arrays, and CSP, each a result of new technology-enabled opportunities driving new architecture in combination with an associated programming model. In each case, the new class of systems reflected a fundamental execution model. In the last few years, changes in technology and limits of design complexity have forced multicore structures while future system projections suggest ExaScale systems by the end of this decade comprising hundreds of millions of cores and multi-billion way parallelism. To coordinate the cooperative operation, their design, and their programming of computing components comprising such massive structures, a new model of computation is required inaugurating a 6^{th} phase of HPC.

PX is motivated by: (1) the long term objective of enabling Exaflops scale computing by the end of the decade in an era of flattening clock rates and processor core design complexity resulting in the expected integration of up to a billion cores by the beginning of the next decade; (2) the more immediate scaling concerns of a diverse set of what we will call *scaling-challenged* problems that do not scale well beyond a small cores number, and takes a long time to complete.

PX is being developed to explore computing paradigms capable of exposing and exploiting substantially greater parallelism through new forms and finer granularity, while eliminating the over-constraining properties of such typical synchronization constructs as barriers. PX is also devised to incorporate intrinsic latency hiding and facilitate dynamic adaptive methods of resource and task management to respond to contention for shared physical and logical resources, including runtime load-balancing. PX, like any true model of computation, transcends any single element of a high performance computer to represent the holistic system structure, interdependencies, and cooperative operation of all system component layers.

The form and function of the current experimental PX model are briefly described:

Active Global Address Space (AGAS) provides a system-wide context of a shared name space. While avoiding the constraining overhead of cache coherence, it extends the PGAS models (UPC [18], GASNet [3]) by permitting the dynamic migration of first class objects across the physical system without requiring transposition corresponding virtual names. This is critical for dynamic objects, load-balancing, and fault tolerance in the context of reconfigurable graceful degradation.

Parallel Processes are the hierarchical context in which all computation of a given parallel application is performed. Unlike conventional models, PX processes may span multiple nodes and may share nodes as well. Such a process provides part of the global name space for its internal active entities, which include other chips' processes, threads, data, methods, and physical allocation mappings.

Threads provide local control flow and data usage within a single node serving as the primary modality for specifying (thread method) and performing (instantiated thread) most of the computational work to be performed by an application program. Threads represent a partially ordered set of operations on an infinite set of single-assignment registers as well as on relatively global data (allowing partial functional programming). Threads (and processes) are ephemeral and as first class objects can migrate (easier fault tolerance and power management).

Local Control Objects (LCOs) are a unification of the semantics of synchronization enabling a new methodology of representing, managing, and exploiting distributed parallel control state. Using conventional object-oriented techniques limited to small objects within a single node, these dynamic objects realize synchronization constructs from simple semaphores to complex dataflow templates and*actor future*.

Parcels are messages that convey action as well as data asynchronously between physically disjoint system nodes. Parcels are a class of Active Messages with destinations either logical objects or physical entities, actions representing either primitive hardware functionality or subroutine methods, payload of a few argument values or alternatively large blocks of data, and a specifier (continuation) of follow-on activity and/or change of global control state.

Percolation is a special technique for using precious resources or new resources by moving the work to the resource while both hiding the latency of such action and eliminating the overhead of such action from the target resource. This is of value for efficient use of GPU accelerators or for new previously unallocated nodes for dynamic resource allocation.

While PX suggests changes or entirely new forms of system elements including computer architecture, near-term

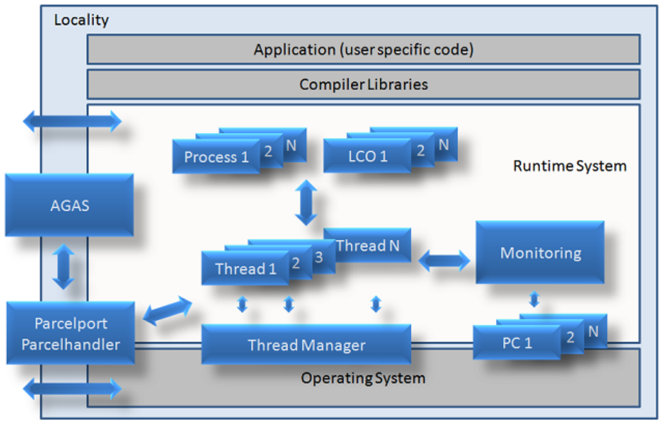


Figure 1: Modular structure of HPX implementation.

work has been focused on best practices for conventional systems to provide early value for problems that challenge scaling.

What follows describes such an experimental runtime system and its application to an AMR numerical relativity code.

2.1 A Runtime System Implementation

The high-performance PX (HPX) [1] runtime system –a C++ implementation, represents a first attempt to develop a comprehensive API for a parallel runtime system supporting the PX model. Among the key features of HPX we discuss:

- It is a modular, feature-complete, and performance oriented representation of the PX model targeted at conventional architectures and, currently, Linux based systems, such as SMP nodes and conventional clusters.
- Its modular architecture allows for easy compile time customization and minimizes the runtime memory footprint.
- It enables dynamically loaded application-specific modules to extend the available functionality, at runtime. Static pre-binding at link time is also supported.
- Its strict adherence to Standard-C++ [17] and the utilization of Boost [4] enable it to combine powerful compile time optimization techniques and optimal code generation with excellent portability.

We designed HPX as a runtime system providing an alternative to conventional computation models, such as MPI, while attempting to overcome their limitations such as: global barriers, insufficient and too coarse-grained parallelism, and poor latency hiding capabilities due to the difficultly in orchestrating the overlap of computation and communication.

2.2 General Design

The implementation requirements of the HPX library as described in the previous section directly motivate a number of design objectives. Our most important objective was to design a state-of-the-art parallel runtime system providing a solid foundation for PX applications while remaining as efficient, as portable, and as modular as possible. This efficiency and modularity of the implementation is central to the design, and dominates the overall architecture of the library. Figure 1 shows a block diagram of the architecture of our HPX implementation. It exposes the necessary modules and an API to create, manage, connect, and delete any PX parts from an application; it is generally responsible for resource management. The current implementation of HPX provides the infrastructure for the following PX concepts.

2.3 AGAS – The Active Global Address Space

The requirements for dynamic load-balancing and the support for dynamic AMR related problems define the necessity for a single global address space across the system. This not only simplifies application writing, as it removes the dependency of codes on static data distribution, but enables seamless load-balancing of application and system data. The abstraction of localities is introduced as a means of defining a border between controlled synchronous (intra-locality) and fully asynchronous (inter-locality) operations. A locality is a contiguous physical domain, managing intra-locality latencies, while guaranteeing compound atomic operations on local state. Different localities may expose entirely different temporal locality properties. Our implementation interprets a locality to be equivalent to a node in a conventional system. Intra-locality data access means access to the local memory (or disk), while inter-locality data access and data movement depend on the system network. In PX, accessing first class object is decoupled from its locality.

2.4 Threads and their Management

The HPX-thread manager implements a work queue based execution model. PX-threads are first class objects with immutable global names, enabling even-remote management. We avoid threads crossing localities (expensive operation); instead, work migrates via *continuation* by sending a PX-*parcel* that might cause the instantiation of a thread at the remote locality. PX-threads are cooperatively (non-preemptively) scheduled in user mode by a thread manager on top of an OS-thread per core. The threads can be scheduled without a kernel transition, which provides a performance boost. Additionally the full use of the OS's time quantum per OS-thread can be achieved even if a PX-thread blocks for any reason.

2.5 Parcel Transport and Parcel Management

In PX, *parcels* are an extended form of active messages [19] for inter-locality communication. Parcels are the remote semantic equivalent to creating a local PX-thread. If a function is to be applied locally, a PX-thread is created; if it has to be applied remotely, a parcel is generated and sent which will create a PX-thread at the remote site. Parcels are either used to move the work to the data (by applying an operation on a remote entity) or to gather small pieces of data back to the caller. Parcels enable message passing for distributed control flow and for dynamic resource management, featuring a split phase transaction based execution model. While the current implementation relies on TCP/IP, we will be moving to existing high performance messaging libraries, such as GASNet [3] and Converse [10].

2.6 LCOs – Local Control Objects

An LCO is an abstraction of different functionalities for event-driven PX-thread creation, protection of data structures from race conditions and automatic event driven on-the-fly scheduling of work with the goal of letting every single function proceed as far as possible. Every object which

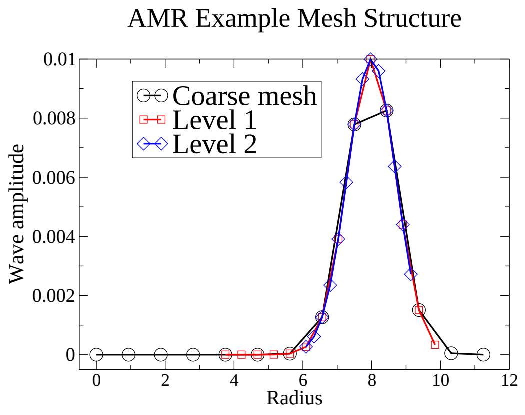


Figure 2: Two levels of AMR.

may create (or reactivate) a PX-thread as a result of some action exposes the necessary functionality of an LCO. A well know and prominent example of an LCO is a Future. It refers to an object that acts as a proxy for a result that is initially not known, usually because the computation of its value has not yet completed. The *future* synchronizes the access to this value by optionally suspending the requesting thread until the value is available. HPX provides specialized implementations of a full set of synchronization primitives (mutexes, conditions, semaphores, full-empty bits, etc.) usable to cooperatively block a PX-thread, while informing the thread manager that other work can be run on the OS-thread (core). The thread manager can then make a scheduling decision to execute other work. In general, LCOs are used to organize flow control.

The next section discussed the AMR physics application we implemented in HPX.

3. AMR-BASED APPLICATION

Modern finite difference based simulations require adequately resolving many physical scales which often vary over several orders of magnitude in the computational domain. Many high performance computing toolkits have been developed to address this need by providing distributed AMR based on the MPI libraries [15, 7]. These toolkits serve all types of applications ranging from astrophysics and numerical relativity to Navier-Stokes solvers. Adaptive mesh refinement, introduced by Berger-Oliger [2], employs multiple computational grids of different resolution and places finer-resolution meshes where needed in the computational domain in order to adequately resolve phenomena at increasingly smaller physical and temporal scales.

3-D AMR simulations are typically 10^4–10^5 times faster than performing a computation using a single resolution mesh. A sample initial AMR mesh structure of the test application we explore here is illustrated in Fig. 2; the initial data supplied is a wave pulse. As the wave pulse moves, the higher resolution meshes adjust accordingly in order to keep the local error criterion below threshold. Other crucial components to a typical finite difference based AMR simulation include the clustering algorithm, which decides how to best arrange and break up the many overlapping finer meshes, and the load-balancing scheme, which decides how to distribute the workload across the available processors.

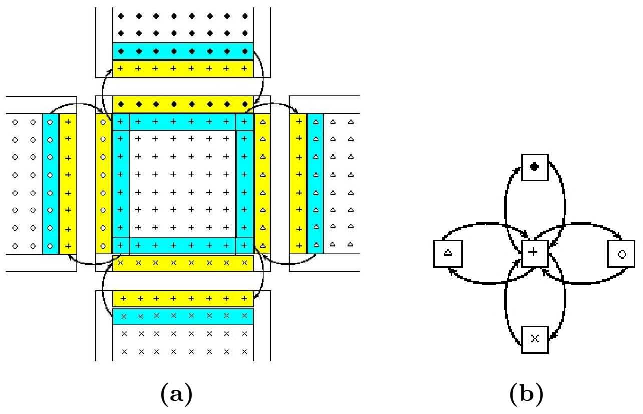


Figure 3: Two different approaches to structured mesh based communication: (a) 2-D representation of a typical communication pattern for a finite difference based AMR code, and (b) the PX-based AMR.

The clustering algorithm, load-balancing scheme, and the specific application workload all impact the scaling characteristics of the AMR application. We differentiate scaling characteristics into two types: strong and weak scaling. In strong scaling, the test application problem size is kept constant while the number of processors devoted to computing the simulation is increased. In weak scaling, the problem size is increased as the number of processors is increased so that the local processor workload on the system is kept constant.

While some AMR applications have successfully demonstrated weak scaling to many thousands of processors, strong scaling to such numbers has not been demonstrated among available alternatives.

MPI based AMR applications suffer from a global barrier every timestep: no point in the computation can proceed to the next timestep unless all other points have reached the same point in the computation. Depending on the granularity characteristics of the physical problem being simulated, this global barrier can cause a large number of processors to sit idle while waiting for the rest of the grid to catch up.

The PX based AMR presented here removes all global barriers to computation, including the timestep barrier, and enables autonomous adaptive refinement of regions as small as a single point without requiring knowledge of the refinement characteristics of the rest of the grid. A key component of this is the ability to implement the finest granularity possible in a simulation. Finite difference AMR codes normally pass large memory blocks of grid points to the user defined code; only the boundaries of these blocks are communicated between processors as illustrated in Fig. 3(a). While the PX based AMR code explored here is capable of this paradigm, it is also capable of the extreme limit of computational granularity – a single point – as shown in Fig. 3(b). This provides the most amount of independence in computing different regions of the computational domain and thereby gives more flexibility in order to better load balance the AMR problem. The ability of the PX based AMR code to change granularity as a runtime parameter allows the user to adapt to the optimal granularity for a specific architecture and a specific number of processors.

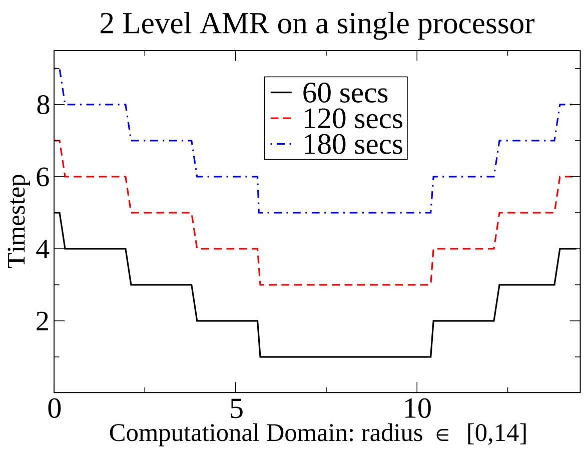
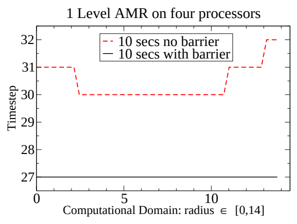


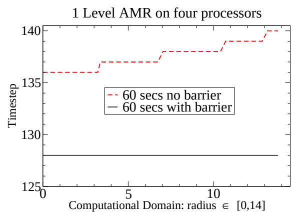
Figure 4: Snapshots at various wall clock time intervals of the timestep each point in the computational domain has reached; when global barriers are removed, some points in the computational domain can proceed to compute more timesteps than others in a fixed amount of wall clock time. The futures construction ensures that causality is still respected in the computation. Consequently the timestep curve takes on an upward facing cone shape.

4. ANALYSIS

In this section we explore three main implications of key concepts in the PX based AMR implementation. First, we find that one of the principal concepts of PX, message-driven work-queue execution, results in implicit load-balancing for parallel AMR simulations. Second, we demonstrate that PX is capable of implementing highly nontrivial AMR problems and successfully run in parallel across an order of magnitude of processors using the smallest computational granularity possible. Third, we demonstrate the functionality of parallel memory management for asynchronously accessed dynamically refined mesh objects. Global timestep barriers are ubiquitous in MPI based AMR codes. The message-driven work-queue execution model of PX allows us to eliminate this global timestep barrier using *futures* enabling a much more efficient overlap of computation and communication as illustrated in Fig. 4 and 5. Without a global timestep barrier, some domain points in an AMR simulation compute faster than others while still respecting the causality of the problem as ensured by the *futures* construction. In Fig. 4 this is demonstrated for a 2 level AMR simulation, or a simulation with three different resolution meshes – a coarse mesh, a finer mesh, and a finest mesh. The semilinear wave application was run for 60, 120, and 180 seconds of wall clock time. The timestep each point in the computational domain had reached by the end of the 60, 120, or 180 seconds is plotted. Unlike in MPI simulations, where each point has to be at the same timestep, some points in the computational domain are able to compute more timesteps faster than others. Futures ensure that causality is respected by requiring that the immediate neighbors of a point being updated be at the same timestep as the point being updated. Thus the resulting timestep curve in Figs. 4 and 5 resembles an upward facing cone where the tip of the cone is located in the region of highest spatial resolution.



(a)



(b)

Figure 5: Illustration of the impact of implicit load balancing. This plot compares the timestep reached by every point in the computational domain after either 10 or 60 seconds of wall clock time for an AMR simulation with 1 level of refinement. The refinement criterion was scalar field amplitude. (a) and (b) show results performed on four processors. Removing the global timestep barrier gives more flexibility to load balance; consequently the parallel cases which don't enforce a global barrier are able to compute faster than those cases in (a) and (b) which do enforce a global timestep barrier.

On multiple cores/OS-threads, load-balancing across the processors substantially improves using the message-driven work-queue execution model. The independence in computing different regions of the computational domain gives more flexibility in order to better load-balance the AMR problem. In Figure 5, we compare AMR simulations with 1 level of refinement running with and without a global timestep barrier on four processors. AMR simulations were run for either 10 or 60 seconds of wall clock time and the timestep reached by each point in the computational domain was plotted. Cases without the global barrier were able to compute more timesteps than cases with the global barrier in the same amount of time. This is a natural consequence of the message-driven work-queue execution model: processors are able to overlap communication and computation more efficiently than algorithms which enforce a global barrier every timestep.

Strong scaling results for several PX based AMR simulations are found in Figure 6. In MPI based AMR simulations, the strong scaling performance steadily decreases as more levels of refinement are added. We see the opposite behavior for PX based AMR simulations. The strong scaling stays essentially the same as levels are added. When more processors are used, the strong scaling actually improves as more levels are added. This behavior is a natural consequence of the implicit load balancing capability of PX based AMR simulations.

5. PARALLEX CORE ARCHITECTURE

Learning from the software implementation and AMR's results, we suggest a parallel core architecture to support the PX model. We discuss some details of a notional architecture for a lightweight core that would be closely associated with the memory banks (vaults) for cases of poor temporal locality. This Embedded Memory Processor (EMP) incorporates the mechanisms that would be needed to minimize the overhead, latency, and in some cases contention while exposing parallelism in an energy efficient structure. Complex scientific workloads (memory-intensive and poor

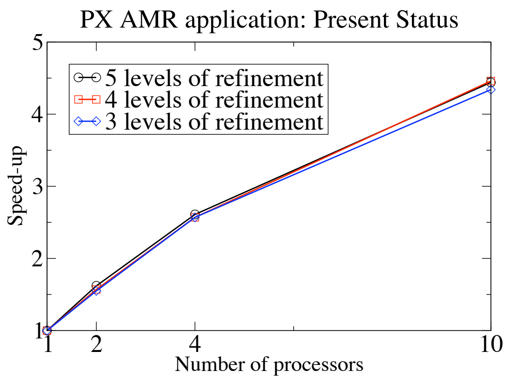


Figure 6: The current strong scaling performance of the PX based AMR application with various levels of refinement. The scaling stays essentially the same except for a slight *improvement* as more levels of refinement are added and the problem is run on more processors. This is completely the opposite behavior from what is observed with the MPI based comparison AMR code where strong scaling steadily degrades as more refinement levels are added.

locality) exhibit very different operational properties than structures primarily dedicated to compute-intensive threads with high temporal locality or many operations per input operand. Structures optimized for one or the other modality will perform better, prove more efficient in terms of utilization and power, and be less costly in die real estate than a more generalized core designed to serve both operational domains. EMPs have to deal effectively with both: an innovative lightweight core to serve memory intensive threads in the context of a highly scalable system capable of the low end real-time embedded applications to Petaflops capability within a rack, and beyond to ExaScale systems comprising 1K such racks deployed before the end of this decade. This lightweight processor will be closely associated with a memory vault comprising one or more dedicated memory banks with tight coupling for low latency, high bandwidth access to data resident in the related memory vault, and will provide direct support for most of PX's functional elements. The principal objectives of the EMP design and operation are:

- Maximize throughput of associated memory vault and related operations,
- Provide low power operation with best energy efficiency,
- Serve in an ensemble of as many as 10^9 cores,
- Support message-driven computation,
- Participate in an efficient AGAS context,
- Enable compound atomic operation sequences, important for realization of LCOs,
- Incorporate fault responsive mechanisms for continued processing in the presence of failures,
- Enforce protection of logical domains and physical resources,
- Minimize cost and complexity of design, fabrication, compilation, and operation.

The following describes key conceptual aspects of an innovative strategy to achieve these objectives for a memory-oriented lightweight processor that will serve as a key highly replicated component of a possible PX-enabled ExaScale computing system.

5.1 Simplicity of Design

Even as performance per processor has increased for some applications, performance per transistor and per Joule has declined steadily over the same period. Additionally, design costs have increased such that a modern general purpose processing chip can cost hundreds of millions or a couple of billion dollars from concept to fabrication and packaging. This limits new processor designs to mass markets such as PCs, embedded, mobile, or games. An alternative design strategy that better suits PX methodology and, in turn, AMR applications is that of maximum simplicity. The following are properties and advantages that may be derived from the strategy of design simplicity:

- Low cost of design (few $100s of thousands) and usage of open-source simulation and verification tools
- High efficiency of energy per operation
- Small core size (low transistor count) provides higher processor density per die and increases yield
- Elimination of wasteful speculative mechanisms

When combined with co-design of all system hardware and software layers, minimizing processor complexity has a number of important advantages that may prove significantly superior to traditional solutions.

5.2 Embedded Memory Processing

The EMP design, building on Processor-In-Memory (PIM) concepts investigated in [8] and implemented in [6] a decade later, will be optimized around the requirements of memory access and operation. It has the following objectives: minimize memory access latency and maximize bandwidth (feasible by the enabling technology, special interconnection to and physical placement with respect to memory banks), support AGAS and enable message-driven computation for asynchronous management and latency mitigation, offer logical mechanisms to support ensemble data processing in conjunction with a billion like cores, and achieve highest energy efficiency for memory access and processing.

5.3 Multithreading

Although the emergence of multithreaded architecture has taken literally decades, its likely role in the future of computing, especially in such architecture structures as the EMP, is significant. Multithreaded structures can be employed in one simple and uniform organization to replace a number of conventional mechanisms that are complicated in design and profligate in their use of power. A multithreaded architecture will incorporate register state for multiple threads of execution that are active and operating concurrently with single cycle context switching for minimum overhead.

5.4 Dataflow Instruction-Level Parallelism

Fine grain parallelism is exposed at the instruction level; the EMP takes a radical departure from conventional practices in this regard to employ an alternative strategy for the exploitation of instruction level parallelism for simplicity of design and superior energy efficiency. A variant of the static dataflow model of computation [14] is to be supported by the

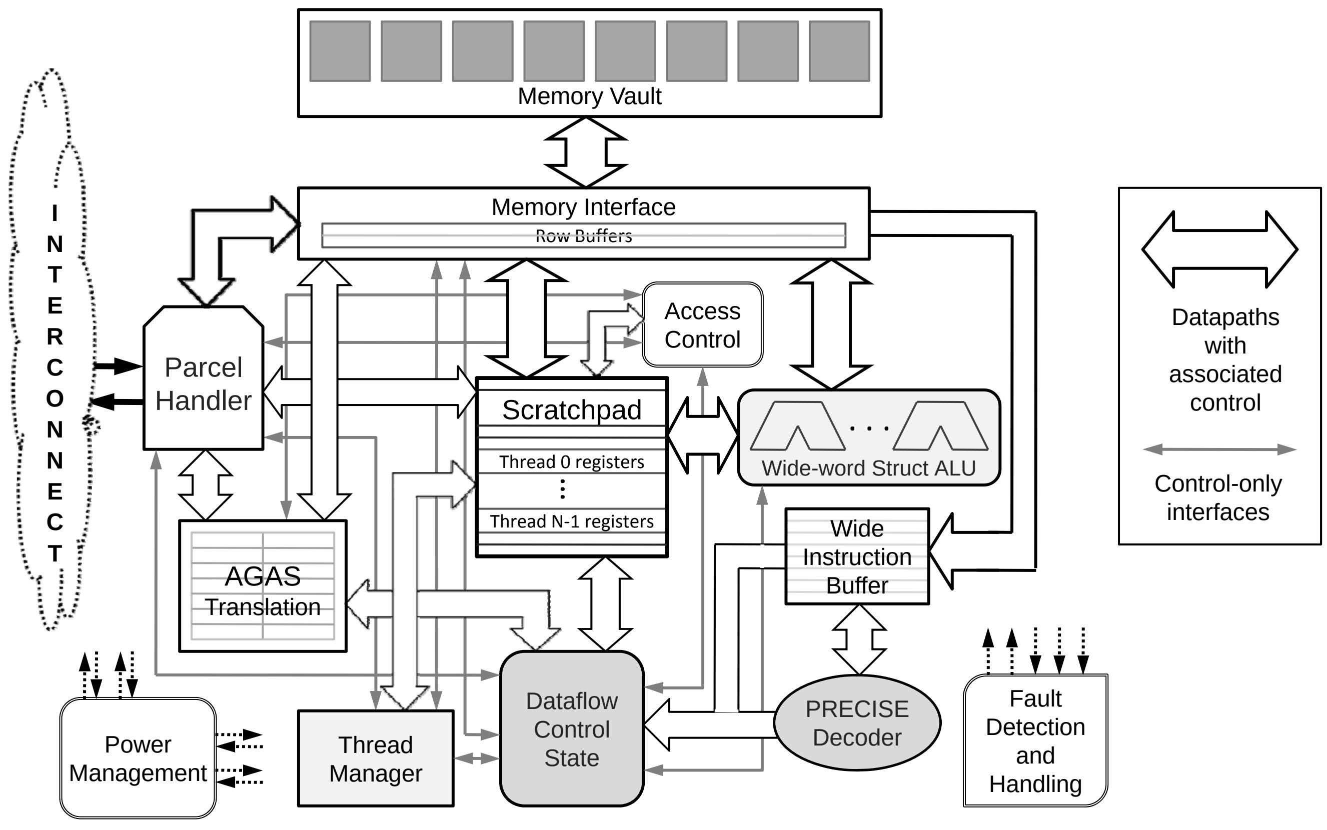


Figure 7: Architecture of the Embedded Memory Processor.

EMP, in which a compiler-derived representation of expressions computed by a thread is encoded in instruction stream and utilized by the processor with almost no overhead. A new control state that replaces the program counter with the program control frame to manage out-of-order execution of fine-grain operations will be investigated to reduce the general O(n^2) control state complexity to O(n), while retaining availability of sufficient instruction level parallelism for local latency hiding.

5.5 Message-Driven (Parcel) Computation

An innovative research class of message-driven computing mechanisms is the *parcel* (PARallel Control ELement) which includes first class objects as destinations and specifies threads to be instantiated as well as other types of actions to be performed (e.g., atomic memory operations). In support of parcel-driven computation the EMP architecture will incorporate hardware mechanisms for the acquisition, buffering, and verification of parcels, as well as aiding their transformation into thread instances or support more primitive actions, such as hardware state examination and modification for fault detection and handling. The architecture will also provide hardware assist for conversion of data structures representing remote actions to their representative parcels that will carry them out.

5.6 AGAS Translation

For purposes of efficiency and productivity, including system support in the core processor for some form of dynamic GAS management is required. Pure distributed memory systems require application software to manage all inter-process communication and hardware support for cache coherence, both time and energy cost heavy. PGAS provides an environment for put/get semantics not requiring cache coherence, while being sufficiently constrained in terms of static allocation to permit easy routing decisions. However, this comes at the cost of a wealth of applications and system operational aspects that demand more dynamic behavior, specifically the ability to migrate virtual objects across physical space without altering the objects' virtual names.

5.7 PRECISE: Variable-Length Binary ISA

A concept of unproven but potential value for reducing bandwidth requirements of instruction fetch and as a consequence power requirements with added benefits to storage capacity at core-buffer, instruction cache, and main memory levels will be pursued based on an advanced form of variable length binary instruction set architectures. Processor register extensions for collapsed instruction set encoding (PRECISE) are described in [13].

5.8 Wide-Word Processing

Close proximity of logic to memory banks affords opportunity for optimized datapaths for higher bandwidth with lower access latency offering best memory access performance for low temporal locality access patterns and lowest energy per access. Actions operate on heterogeneous data structures (as opposed to scalars or SIMD-like operations), where the action to be performed on one element of the structure is dependent on the state of that of another within the structure. Graph processing where vertices are represented as such structures may be greatly accelerated with wide-word accesses and processing.

The EMP designed in accordance with these principles shows promise of practical implementation of billion-core machines in this decade.

6. CONCLUSIONS

We have presented the ParalleX model of computation, a distributed parallel AMR application framework, and performance results using this framework for a real-world physics application. We note that the HPX implementation of PX is available (upon request) under an open source license. The performance and scalability of the new model of computation can be further improved through co-design of the underlying execution hardware. We have delineated a feasible and novel processing architecture, which provides a direct support for most of the PX primitives. While it rivals many of the existing solutions in simplicity, it also offers a potential for increased power efficiency, robustness and security, paving a viable path to the ExaScale systems in the future. As part of future work, we will concentrate on improving scaling for multidimensional AMR applications on both multicore and distributed machines. In the hardware domain, we will test the elements of the EMP on programmable logic (FPGA) boards plugged into traditional computing nodes. Promising results of these early experiments may result in custom silicon implementation.

7. REFERENCES

[1] Alex Tabbal. High-Performance ParalleX. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. ACM/IEEE, 2010. Poster.

[2] M. J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *J. Comp. Phys.*, 53:484, 1984.

[3] D. Bonachea. GASNet Specification, Tech Report (UCB/CSD-02-1207), 2002.

[4] Boost: A Collection of Free Peer-Reviewed Portable C++ Source Libraries, 2010.

[5] S. Borkar. The ExaScale Challenge. In *VLSI Design Automation and Test (VLSI-DAT), 2010 International Symposium on*, pages 2 –3, apr. 2010.

[6] J. Draper et al. The Architecture of the DIVA Processing-In-Memory Chip. In *ACM International Conference on Supercomputing (ICS'02)*, June 2002.

[7] R. Hornung et al. SAMRAI home page. http://www.llnl.gov /CASC/SAMRAI/, 2010.

[8] K. Iobst, M. Gokhale, and B. Holmes. Processing in Memory: The Terasys Massively Parallel PIM Array. *IEEE Computer*, 28(4):23, Apr. 1995.

[9] H. Kaiser, M. Brodowicz, and T. Sterling. ParalleX: An Advanced Parallel Execution Model for Scaling-Impaired Applications. *Parallel Processing Workshops, International Conference on Parallel Processing*, pages 394–401, 2009.

[10] L. Kalé, J. Phillips, and K. Varadarajan. *Parallel and Distributed Processing*, chapter Application performance of a linux cluster using converse, pages 483–495. Springer Berlin / Heidelberg, 1999.

[11] P. Kogge et al. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems, 2008.

[12] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard, V2.1, 2008.

[13] C. Michael, M. Brodowicz, and T. Sterling. Improving code compression using clustered modalities. In *Proceedings of the 46^th ACM Southeast Conference*, March 2008.

[14] G. Papadopoulos and D. Culler. Monsoon: An Explicit Token-Store Architecture. In *17th International Symposium on Computer Architecture*, number 18(2) in ACM SIGARCH Computer Architecture News, pages 82–91, Seattle, Washington, May 28–31, June 1990.

[15] E. Schnetter. Carpet homepage. http://www.carpetcode.org/, 2010.

[16] T. Sterling, C. Dekate, and M. Brodowicz. Introduction to High-Performance Computing; Models, Methods and Means (*Graduate Computer Science Class, LSU*), Spring 2006, 2007, 2008, 2009.

[17] The C++ Standards Committee. Working Draft, Standard for Programming Language C++, 2008.

[18] UPC Consortium. UPC Language Specifications, v1.2, Tech Report LBNL-59208, 2005.

[19] T. von Eicken et al. Active Messages: A Mechanism for Integrated Communication and Computation. In *Proceedings the 19th Annual International Symposium on Computer Architecture*, pages 256–266, Gold Coast, Australia, May 1992.