A non von Neumann Continuum Computer Architecture for Scalability Beyond Moore's Law

Maciej Brodowicz
Center for Research in Extreme Scale Technologies
Indiana University
420 N. Walnut St.
Bloomington, IN 47404, USA
mbrodowi@indiana.edu

Thomas Sterling
Center for Research in Extreme Scale Technologies
Indiana University
420 N. Walnut St.
Bloomington, IN 47404, USA
tron@indiana.edu

ABSTRACT

A strategic challenge confronting the continued advance of high performance computing (HPC) to extreme scale is the approaching near-nanoscale semiconductor technology and the end of Moore's Law. This paper introduces the foundations of an innovative class of parallel architecture reversing many of the conventional architecture directions, but benefiting from substantial prior art of previous decades. The Continuum Computer Architecture, or CCA, eschews traditional von Neumann-derived processing logic, instead employing structures composed of fine-grain cells (*fontons*) that combine functional units, memory, and network. The paper describes how CCA systems of various scales may be organized and implemented using currently available technology. As programming of such systems substantially differs from established practices, a still experimental ParalleX execution model is introduced to be used as a guide for the implementation of related software stack layers, ranging from the operating system to application level constructs. Finally, the HPX-5 runtime system, an advanced implementation of ParalleX core components, is presented as an intermediate software methodology for CCA system computation resource management.

CCS Concepts

•**Computer systems organization** → **Cellular architectures;**
•**Hardware** → **Emerging architectures;**
•**Software and its engineering** → *Ultra-large-scale systems;*

Keywords

computer architecture; parallel computing; extreme scale

ACM ISBN 978-1-4503-4128-8/16/05.

DOI: http://dx.doi.org/10.1145/2903150.2903486

1. INTRODUCTION

With device technologies approaching their asymptotes, opportunities exist in alternative, perhaps radically different computer architectures. All mainstream and conventional supercomputer architectures are von Neumann derivatives, themselves derived at a time of extremely different technology and architecture tradeoffs. Looking forward it is recognized that the early priorities still assumed in today's design no longer apply and that the forms of abstract parallelism being employed can no longer take adequate advantage of the inherent capabilities of the emerging semiconductor logic. Specifically, the driving emphasis of floating point ALU utilization, while justified in the 1950's and 1960's is now completely wrong with these components among the lowest cost. Yet the entire memory hierarchy (cache structure) is based on this assumption. In this paper, an alternative HPC architecture class based on uniform arrays of very lightweight operational units and an advanced execution model (ParalleX) is investigated that will guide the distributed computation and runtime software (HPX-5) to provide control and programming interface to the system.

A class of cellular architectures is envisioned to take advantage of future near-nanoscale technologies, address the key challenges of starvation, latency, overhead, and contention, respond to asynchrony of interaction within a global name space, and dynamically adapt to changing resource demands and availability. The full system (*Simultac*) is an n-dimensional array of very small operational units (*fontons*) with nearest neighbor interfaces, single cycle functionality including functional data operations, tagged data storage, and data migration to nearest neighbor. As will be seen, it is as if three distributed global systems were mapped on top of each other including memory, communication, and array of functional units. Control is provided by the abstraction of the ParalleX execution model to delineate tightly integrated local sets of dataflow operations and intermediate tasks called *compute complexes*, to designate blocks of data and their virtual addressing and physical translation, and a hierarchy of *ParalleX processes* mapped across large fonton sub-arrays. Global synchronization and continuations are realized through *dataflow* and *futures* manifest as *local control objects* that establish and manipulate asynchronous precedent constraints and balance eager and lazy evaluations. HPX-5 is an experimental runtime software system that is a reduction to practice of the ParalleX execution model, and has been ported to a number of medium

Table 1: Properties of CCA compared with other prevalent technologies.

Property	CCA	Multi-core	GPU
Granularity of execution	fine	medium (disjoint cores)	fine to medium (determined by SIMD)
On-die local store bandwidth	very high (aggregate intra-fonton)	medium (registers, caches)	high (registers, caches)
Attached memory bandwidth	medium to high (optional external DRAM ensemble)	low (shared DRAM banks)	medium to high (GDDR, HBM); low (host DRAM)
Memory addressing	global, associative	node local	local only; node local with HSA
Memory coherence	software controlled with hardware support	node scope (NUMA)	GPU scope
On-die communication bandwidth	very high	medium (HT, QPI)	medium-high (shared storage)
I/O latency	low to medium, variable	medium (PCIe)	medium (PCIe)
I/O bandwidth	medium to high	medium (PCIe)	PCIe and host dependent
Recovery from faults	likely (replication and dense routing)	highly limited	limited

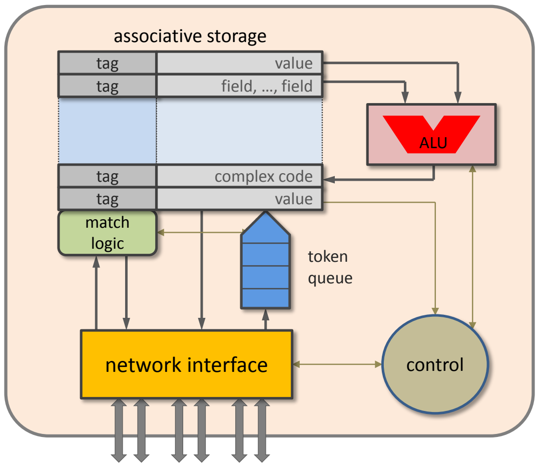

Figure 1: Internal structure of the fonton.

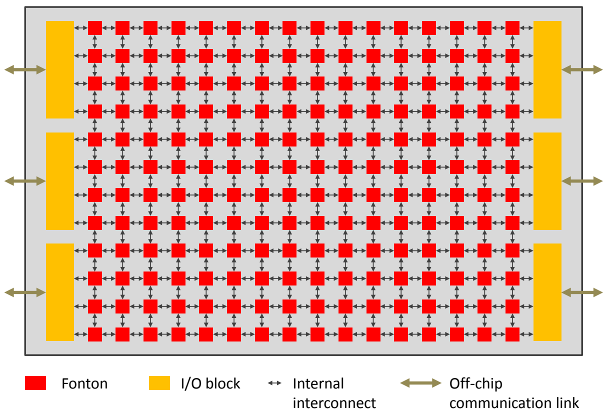

Figure 2: Example fonton arrangement of the on-die logic (2-D mesh interconnect shown for simplicity).

and large scale systems around the country.

2. COMPONENTS OF CONTINUUM COMPUTER ARCHITECTURE

The high-level architecture of a fonton, the highly replicated elementary building block of a CCA system, is shown in Figure 1. It combines hardware components responsible for processing, storage, and network access into a single unit that may be replicated to form structures of arbitrary complexity. Each of these units is matched to the others in terms of access latency and data bus bandwidths to maximize the work performed in each clock cycle and avoid unnecessary buffering of intermediate request data.

The processing within a fonton is controlled by a local state machine that arbitrates and coordinates the flow of requests that are generated internally as well as those arriving from the network. There is no notion of explicit program counter. The operands necessary to execute a sequence of micro-operations are bundled with the relevant instructions to form a token, an atomic execution unit. Token execution may affect the state of a local fonton as well as state of a remote fonton or even fonton groups. The first happens when the operations modify the contents of local storage registers, for example, when writing the result of ALU operation or upon explicit initialization. The latter is carried out through generation of tokens targeting the state of remote fontons.

The fonton is equipped with a minimal ALU that supports integer arithmetic and logical operations, along with pattern matching and bitwise permutation of data values. The complexity of this unit is much lower than that of traditional FPU; floating-point and extended integer arithmetic are higher-level functions that are synthesized using a combination of multi-cycle processing within a single fonton or using closely connected fonton groups in adjacent locations. Although potentially slower, this provides unparalleled flexibility of matching the precision (and therefore resource footprint and energy) to specific application.

Since physical memory is distributed across the system, traditional addressing schemes cannot be applied directly. Instead, fontons explicitly store associative tags along with the memory contents. The tags are unique for each entry, but also permit matching of storage cell sets, implementing a form of wildcard addressing. As the physical location of

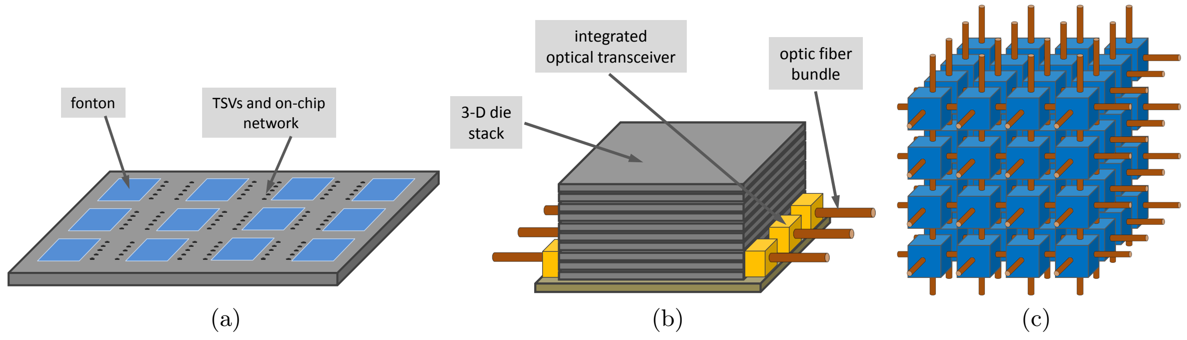

Figure 3: Possible implementation of a high-performance CCA system using currently available technology, showing (a) single die, (b) 3-D die stack with optical I/O, and (c) full CCA system (simultac). High fonton density and bandwidth required for data movement are achieved through 3-D stacking, through-silicon vias, and integrated photonics connecting logic instantiated on conventional VLSI CMOS silicon.

fonton containing arbitrary address tag is not known a priori (unless it happens to be local to the requesting fonton), part of the system is configured as a distributed, redundant address resolution service for non-local memory accesses. To support this mode of operation, register files of predetermined fontons store routing information for the subset of memory contents.

The network interface consists of several bidirectional links connecting the nearest neighboring fontons creating a physical array. A token packet, if traffic conditions permit, is sent over a link either in a single cycle for very short messages or by wormhole routing for extended messages. The network interface can perform associative lookup on register tags to identify whether token's destination target is local or if the related register contains routing information. In the first case the token is absorbed by the fonton for processing; in the second, the routing information is interpreted to modify the token's direction. Otherwise, the token continues to move in a direction determined by the applicable default routing rules.

The first step involved in scaling the CCA platform beyond the single cell level is instantiation of multiple fontons on a die similarly to replicating multiple processing cores in a multicore processor. The difference between these two approaches, however, is that due to small fonton size a much larger number of them may be accommodated per die; layout issues are also simplified due to high level of repetition and regularity compared to the complexity of modern processor cores (especially taking into account various layout limitations required to provide high clock rates). Another difference is usage of denser interconnect with higher degree of local connections. While conventional many-core implementations use relatively sparse intra-chip networks such as QPI, HyperTransport, or custom ring networks (such as used by IBM Cell processor or Intel MIC variants) to permit communication between individual cores, the CCA emphasizes efficient nearest-neighbor communication between fontons with triangular, rectangular or hexagonal perimeter.

3. EXECUTION AND PROGRAMMING ENVIRONMENT

3.1 ParalleX Execution Model

The ParalleX execution model [4] describes a hierarchy of processing tasks, the structured objects upon which they operate, and the organization and synchronization of concurrency as well as the means of managing the uncertainty of operational uncertainty. It serves as scaffolding for introspection: the ongoing monitoring of system behavior and the policies for adjusting priorities of task and data placement in response to the system and application state. It also reflects local relationships, both physical and abstract. Together the distribution of parallel work and resources and the relative affinity of spatial and temporal locality guide dynamic load balancing to maximize concurrency of execution while minimizing overheads and latencies. Principal constituents of ParalleX can be briefly described and justified in terms of the SLOWER [5] performance model. Primary semantic constructs of ParalleX, depicted in Figure 4, are:

- **Global Name Space** provides means of referencing first class objects across the entire data and task set of an executing application and system.
- **Locality** is a physical resource integrating data storage, logical functionality, and control and data movement with guaranteed bounded response time.
- **ParalleX Process** is a named hierarchical context for data and activities that spans multiple localities (unlike the conventional MPI process).
- **Compute Complex** describes the principal task structure that embodies medium and fine grain parallelism and means of performing work in the ParalleX model. It serves the purpose of the "thread" in more conventional systems and a usual thread could be a limited implementation of the complex.
- **Local Control Object (LCO)** is a class of synchronization semantics that in the ParalleX execution model are organized in dynamic graphs across the distributed system. Two most important LCO forms include dataflow and futures [2]. An LCO establishes the events required to instigate a new action, the conditions associated with these events that have to be satisfied (predicate), and the resulting action.
- **Parcel** is a form of active message. It supports message-driven computation to move work to data for reduced latency effects. A parcel can move small or large amounts of data between virtual objects anywhere in the application program across applied resources. It can also invoke

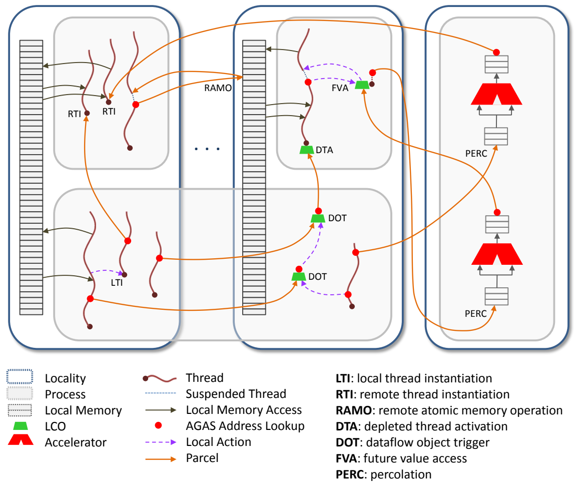

Figure 4: Semantic components of ParalleX and their interactions.

an action at the site of a first class object. The action can be a full compute complex, or a lightweight primitive atomic operation that does not engage the total runtime environment.

3.2 HPX Runtime System

The HPX runtime system software is a proof of concept and first reduction to practice of the ParalleX execution model. Multiple versions of HPX have been developed to emphasize different aspects of the model, different goals for the system, or alternative implementation methods. For example, LSU has developed the HPX-3 runtime system library based on C++ and the Boost Library. Indiana University has developed and is refining a C-based version, HPX-5 [1] for performance optimization, application scaling, and architecture evaluation. Both currently target existing HPC systems and Unix-like operating systems presenting a set of library calls to the application programmer. HPX has been used to implement a number of scientific codes or enhance their execution environment [3, 6] for improved scalability.

The HPX runtime system software incorporates a set of coupled services to support the ParalleX execution model as described in detail above. HPX has a set of major functions that include:

- Global address space management and address translation,
- Thread queuing and scheduling (complexes on conventional cores),
- Parcel communications protocol and message packet transfer,
- Synchronization objects, structures, and state transition, and
- Introspection, measurement, and dynamic policies.

The existing HPX-5 libraries may be already used to enable porting of application to CCA while minimizing the associated development effort. However, optimized runtime algorithms for fine-grain spatial management of computations are still absent from current implementation. Such mechanisms will have to be developed to permit allocation of simultac resources taking into account not only the footprint occupied by each executable component, but also the

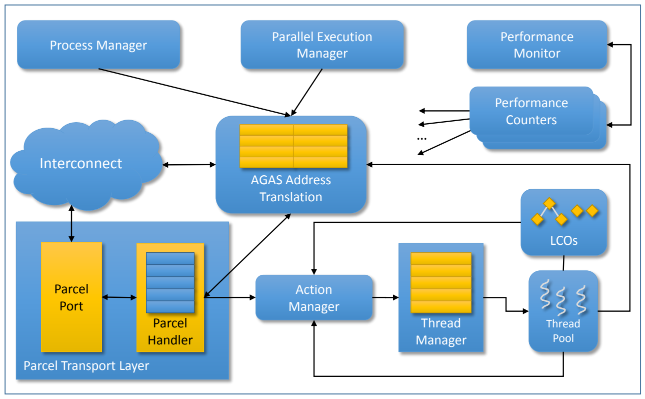

Figure 5: Components of the HPX runtime system.

topological proximity to other computations that component may interact with. The developed solution must be robust enough to accommodate the changes in resource requirements over application runtime. Significant modifications of the HPX scheduler are also expected in order to take advantage of dense execution resources. For example, it is no longer necessary to suspend a newly created thread because all cores of the processor are busy. Instead, a decision must be made where in the unassigned portion of the execution medium should the new complex be started, and if running out of space, how to route the trigger event (possibly emitted when some of the earlier started processes finish or when data dependencies are satisfied) to the fonton containing such a dormant complex. The scheduler will also shift from purely temporal management of computations to directing their spatial arrangement, including selection of the most appropriate execution regime if several implementations are available (e.g., pipelining vs. in-place data processing).

4. REFERENCES

[1] HPX-5 web page. http://hpx.crest.iu.edu/.
[2] H. C. Baker and C. Hewitt. The incremental garbage collection of processes. In *SIGART Bull.*, pages 55–59, New York, NY, USA, August 1977. ACM.
[3] C. Dekate et al. Improving the scalability of parallel N-body applications with an event driven constraint based execution model. *The International Journal of High Performance Computing Applications*, 2012.
[4] H. Kaiser, M. Brodowicz, and T. Sterling. ParalleX: An Advanced Parallel Execution Model for Scaling-Impaired Applications. In *Parallel Processing Workshops*, pages 394–401. IEEE Computer Society, 2009.
[5] T. Sterling, D. Kogler, M. Anderson, and M. Brodowicz. SLOWER: A performance model for exascale computing. *Supercomputing Frontiers and Innovations*, 1(2), 2014.
[6] S. Yang, M. Brodowicz, H. Kaiser, and W. Ligon. PXFS: A persistent storage model for extreme scale. In *Proceedings of the Workshop on Computing, Networking, and Communications, ICNC'14*, 2014.