Continuum Computer Architecture for Nano-scale and Ultra-high Clock Rate Technologies

Thomas Sterling
Maciej Brodowicz

California Institute of Technology

1. Introduction and Motivation

The anticipated advent of practical nanoscale technology sometime in the next decade with likely experimental technologies nearer term presents enormous opportunities for the realization of future high performance computing potentially in the pan-Exaflops performance domain (10^{18} to 10^{21} flops), but imposes substantial, albeit exciting, technical challenges as well. With device density (basic components per unit area) at nanoscale predicted at least 1000X today's commercial feature size and local clock rates expected to be at least 10X that of current generation semiconductor technology, advanced technologies will perform in an operational regime dramatically different from conventional CMOS-based microprocessors and DRAM at present. The dominant factors that will determine the structures and behavior of future nanoscale computing elements are:

1. Clock rate – the logic speed of advanced technologies will be significantly faster than current generation,
2. Device density – linear feature size reductions of close to two orders of magnitude resulting in three to four orders of magnitude gates per unit area,
3. Parallelism – systems implemented in nanoscale technology will comprise orders of magnitude greater number of local execution sites and therefore will need to exploit much higher degrees of algorithm concurrency.
4. Latency – time of signal propagation (measured in clock cycle time) through a sequence of gates or equivalent physical constructs across an entire component is dramatically greater than that of conventional system devices,
5. Reliability – much smaller devices may break more easily; moreover, the probability of single-point failures grows with the scale of the system

The consequence of these interrelated factors alone and in combination is that:

- Advanced components approaching the nanoscale will exhibit strong bias towards locality of action far more extreme than today, and
- Architecture defined to be implemented using nano-scale technologies will have to respond to this.

At one time, processors could employ a single clock such that any logic within the processor was accessible in a single clock cycle for a round trip signal. Even today, this is not really feasible and a combination of slower global clocks with faster local clocks is used while super-pipelined logic structures make the on-chip latency problem manageable. But with nanoscale devices at super high clock speeds and many more gates of propagation possible per unit distance, only a tiny fraction of the total on-chip devices will be accessible by a round trip signal in a single clock cycle. Thus, any nanoscale chip will be inherently balkanized into myriad sub domains of local

action; this is largely independent of the architecture devised. To exploit all these domains, an extremely high degree of application parallelism will have to be available. In all likelihood, these will require effective use of fine-grain algorithmic parallelism which in turn requires very lightweight mechanisms to minimize temporal overhead in the management of these concurrent tasks and parallel resources (again, within these local domains). Coordination through synchronization must now be very lightweight as well. Further, slight variations in structure can cause signal skew. It will become very difficult to achieve true synchronous operation. And none of this takes into consideration main memory which, if classical architectures were to be used, would be thousands of clock cycles away. Waiting for remote responses will dominate any use of conventional architecture techniques.

To appreciate this, we suggest a new parameter, τ (tau), that quantitatively reflects the disparate properties of future new nanoscale devices. τ is the ratio of the number of gates on a chip to the number of gates through which a signal can propagate round trip in a single clock cycle. This does not assume a linear sequence of gates but a two dimensional set with a radial distribution. Thus, the parameter τ may be as small as 1 where all gates on a chip (or module) can be accessed. For a modern CMOS microprocessor, τ can be approximately 10 or slightly greater. But even today, experimental superconductor Rapid Single Flux Quantum (RSFQ) logic fabricated with niobium technology with clock rates > 100 GHz exhibit a τ of approximately 1000. Future nanoscale technologies when fully realized will deliver a τ rating of between ten thousand and a million depending on a number of implementation specific details. Architectural ideas that work for processors in the 1 to 10 τ range will fail when τ soars to five orders of magnitude or more. The purpose of the proposed research is to develop radical architecture structures that will enable nanoscale and other advanced technologies to effectively perform general purpose applications. It is expected that should this research be successful, the proposed architecture concepts in synergy with future nanoscale device technology will yield a new generation of computing systems capable of multiple Exaflops of sustained performance.

A set of architecture concepts collectively referred to as “Continuum Computer Architecture” (CCA) so named because it approximates an ideal continuous space, continuous time medium of execution, but in discretized form has been derived to address the challenges of nano-scale technologies and the end of Moore’s Law. CCA is both a parallel model of computation and an implicit highly parallel hardware structure incorporating local mechanisms invented to enable efficient performance of the CCA execution model. CCA is intended to operate effectively in the high τ regime. Its structures reflect the locality of action in two ways. A site of instruction execution has very few resources but all that is necessary for a single instruction to be performed. CCA dispenses with the concept of “the processor” and instead merges logic, communication, and state storage into a single physical element. The second method that addresses the expected operational regime is that all such elements are message-driven and perform split-transaction execution. An element performs an action upon the incidence of a packet called a parcel that wants to access and potentially modify the element’s local state using the element’s own local logic. For those familiar with classic cellular automata, CCA would appear to be superficially similar. Both exploit locality of action based on local rules and local or nearest neighbor state. Both achieve an emergent global behavior from the effective synergy of myriad simple interacting local elements. But where the effect of cellular automata is often the special purpose mimicking of some physical phenomenon like thermal diffusion through a gaseous medium or some more abstract effect like the game of life, the effect of CCA through

the symbiosis of its interacting elements is a global general purpose parallel computing discipline to govern the execution of any general problem.

A more detailed description of a possible CCA model of execution and logical structure that supports it is presented later. But here we identify some of the critical benefits anticipated of Continuum Computer Architecture for nanoscale technology:

- Organizes all computing actions in local domains of action
- Exposes mammoth memory bandwidth to overcome memory wall
- Permeates the system with arithmetic-logic units to eliminate sources of potential contention and latency increases inherent in typical centralized approaches
- Exploits and exposes the full potential of nanoscale processing capability
- Employs an asynchronous message-driven computing model for split-transaction execution and latency hiding
- Direct implementation of *futures* synchronization construct for powerful efficient fine grain parallel flow control
- Enables lightweight objects for efficient dataflow synchronization
- Permits fine-grain parallelism of large irregular non-ergodic sparse data structures to be efficiently exploited
- Achieves enormous system bi-section bandwidth and adaptive routing for contention avoidance.
- Provides the basis for a new class of fault tolerant devices through reconfiguration and graceful degradation (these issues will not be discussed within the scope of this paper)
- Provides automatic resource management through a “diffusion” methodology of data migration.

2. An Overview of the Principles of Continuum Computer Architecture

We have introduced the driving requirements of nanoscale technology to favor a computer architecture class that emphasizes locality of action, very high parallelism, automatic latency hiding, lightweight synchronization mechanisms, and fully distributed asynchronous control. In this section, an overview is provided of the radically new concept, Continuum Computer Architecture that is invented for the purpose of exploiting nanoscale technologies of the future. While this brief description is at a high level rather than comprehensive in all details, the basic opportunity and technical strategy of the CCA model is presented to sufficient degree to establish it as a new class of computer architecture.

2.1 An Abstraction

Imagine an ether-like medium in N dimensions (assume N = 3 for the sake of discussion) the physical properties of which sustain computing. Admittedly bizarre, such a “continuum” would have attributes of 1) data state, 2) information propagation, and 3) the ability to modify the local state. The degree (or amount) of any one of these characteristics at a local site within this continuum is a product of its density function and the bounded contiguous volume over which it is integrated.

This is easy to understand for state capacity, but is more difficult for the other two properties. The amount of state that can be stored at a site in the continuum is proportional to the local contiguous volume of the storage. A property of the medium, the storage density, determines the actual bit content of the space being considered.

Ordinarily, the rate of movement of data is described as bandwidth. In a continuum, data movement must be treated as a vector. It has direction but what of its magnitude? In mechanics it would be its rate of traversal or perhaps its momentum. For the computing continuum it will be asserted that the distance covered in unit time along the direction of the vector orientation is a constant property of the computing medium. Instead, the magnitude of the communication vector is the product of the integral of the communication density that is a property of the computing medium and the normal area upon which the vector is incident. The time for a communication vector to transit such a cut is the ratio of this bandwidth and the capacity of the vector which may be equated in conventional terms to the total information content (measured in bits).

The maximum rate of state modification or peak performance is similarly derived. This is the maximum numbers of operations that can be accomplished in unit time within a bounded contiguous volume of the computing continuum. It is proportional to this volume and a product of the performance density coefficient property of the medium.

Sustained performance, even over a short time window and relatively small volume, is much more complicated as it must be parallel algorithm-driven. Actual computational actions, or operations, that contribute to the sustained performance occur at any single point (actually tiny volume) when a key criterion is satisfied: we will call this the “condition of coincidence”. Coincidence is the state of locality in time and space of the logical (or abstract) arguments, the physical resources, and the task description information. All of these elements, logical and physical, need to be at the same place at the same time in order to enable a designated operation to take place. As self-evident as such an observation may appear, achieving a coincidence event is the foundation of all computer architecture which can be distinguished by the methods and mechanisms employed to accomplish it.

2.2 Semantic Elements of CCA

CCA employs a message-driven strategy of split-transaction execution. While the final semantic definition of CCA will depend on specific details of a given architecture, key aspects of the logical operation can already be established. The semantics of CCA differ somewhat from conventional microprocessor architectures as they embody both the logical functionality of the local cell and the logical operational relationship among cells. These we represent as the cell instruction set architecture (ISA), and the protocol of the global message-driven computing, respectively. The semantic or logical abstraction of the CCA concept comprises several classes of functions and relationships including: 1) data values, structures, and naming, 2) basic atomic (simple and compound) operations, 3) local and global flow control, and 4) synchronization and task management. Due to space considerations, the entire semantic model as it is understood, although incompletely, is too large to represent here. Instead, we touch briefly on the use of message driven computation and global system synchronization, two essential parts of the entire model. We note that all data variables have virtual names and the hardware mechanisms of the CCA system are able to determine the physical location of any data object anywhere within the system. This “magic” is based on the concept of reference trees and a new strategy of partially

constrained relative data placement, the details of which are too complicated to include here. We also assert that processes are first class objects and as such exist within the application name space and can be manipulated or managed while running. This permits the computation to reason about itself and to control its own execution. This is particularly valuable when performing problems with exponential growth and employing heuristics to govern the suspension of certain tasks in favor of more valuable tasks.

The basic unit of reaction is a quantum (or atomic) message (we call it a “burton”) that targets a particular named variable, data structure, or object (it can be a control object). The burton identifies the destination object and specifies the action to be performed at its site. Therefore, a burton is one of the two basic types of continuations supported by CCA. It contains all the information needed to carry out part of the computation and the logical means to continue the computation after the action has been completed. It incorporates references to critical parts of the name space including the global state, lexically scoped relatively local state, source of thread instruction sequences, and some temporary values. A burton is like a traveling thread. It can carry a short sequence of operation specifiers like a thread and state upon which the burton can operate. Upon incidence at the target site within the continuum computer and completion of the specified action or actions, the burton can terminate, continue, or create one or more new burtons.

Synchronization semantics provide the means of governing the parallel control of program execution. The semantics of synchronization are based on in-memory synchronization data types and atomic operations performed on them. Among the several forms to be provided, the two most powerful are the *futures* construct and the dataflow object. The *futures* construct is a dynamic data structure devised in the late 1970s by Hewitt [1] to provide an important n-way producer-consumer synchronization mechanism. The future is important because it provides a fully distributed means of synchronizing actions on data products which are generated by sources unknown to the accessing task. Requesting burtons arriving at the site of a futures variable for whom the value has yet to be produced can be queued efficiently until the result is available. This action is performed locally and is made efficient because linked data structures like lists, trees, directed graphs, etc. can be created very quickly because data structures are rapidly copied and moved in CCA. The dataflow construct is also a lightweight object that accepts multiple burtons and when a criterion of continued execution is satisfied, the action is performed on local state and possibly one or more burtons are created to continue the execution elsewhere in the system. The dataflow construct may be a satellite data structure associated with another data element within a large linked data structure.

3. Related Research in the Field

Architecture for Nanoscale technology and some other advanced technologies is in its infancy with few concrete examples. This is understandable as the technologies themselves are incipient and there is little motivation on the part of mainstream architects to invest time and resources in a future with so much uncertainty in the likely success or operational properties. Nonetheless, there are a couple of instances to be considered. Also, some of the constructs and mechanisms that comprise the CCA strategy have prior art and can be identified. These will be briefly mentioned here.

Peter Kogge of the University of Notre Dame has led an investigation to examine micro-architectural issues related to quantum dots [2] under development by Craig Lent and his team, also at Notre Dame. This work applied conventional structures derived from the discipline of RISC micro architecture and techniques. This work, while demonstrating that it was feasible to realize such a processor using QCA technology, latency considerations and other issues made it far from effective. Through private communications, it was suggested that something like CCA might be one possible path to overcome those deficiencies identified.

Mikhail Dorojevets at the State University of New York (SUNY) at Stonybrook has explored micro architecture structures for superconductor RSFQ [3] as part of the Flux project sponsored by the NSA and conducted in collaboration with Northrop-Grumman. A niobium chip [4] was fabricated incorporating some of these mechanisms and is currently under test. Earlier, the HTMT project [5, 6] including these institutions and others explored a super-pipelined multithreaded microarchitecture assuming availability of submicron technology operating at over 100 GHz [7] to demonstrate latency hiding within the execution pipeline.

Message driven computation has a long history. In the late 1970s, Hewitt developed the Actor model [8], an object oriented computing model employing message-driven computation. Daly, Keckler, and Noakes at MIT developed the J-Machine [9], a highly parallel architecture with individual processors that were message-driven. Yelick and Culler at UC Berkeley developed the active message model [10] and split-C language [11] for message driven computation for distributed memory machines through software. The DIVA PIM [12] architecture developed by Draper, Hall, and others incorporated a variant of the parcels message driven protocol initially devised for the HTMT architecture. Parcels have continued to be used as the basis for the Gilgamesh MIND [13] architecture developed by Sterling and the Cascade architecture under development by Cray Inc.

Halstead at MIT in the late 1970s developed reference trees for management of distributed virtual address spaces, possibly with copies, and later incorporated this in the early 1980s in the MultiLisp language [14] and multiprocessor implementation. Sterling has developed a variant of reference trees for address management and translation for the MIND architecture. These techniques with important advances will be employed for CCA.

The futures synchronization construct was also developed by Hewitt as part of the Actors model and employed very successfully by Halstead in his implementation of MultiLisp. A variant of futures was devised by Arvind in the 1980s initially at UC Irvine and then at MIT as part of the dataflow language Id Nouveau [15]. Burton Smith of Tera (now Cray) incorporated hardware mechanisms in support of futures in the MTA architecture [16] for efficient producer-consumer computation.

Multithreaded computation has a long tradition with an early implementation by Smith at Denelcor in the HEP computer [17] and then at Tera in the MTA. Gao at McGill (now University of Delaware) developed the Earth system [18] (no relation to the Japanese Earth Simulator) which was a software implementation of a multithreaded execution model. Culler at Berkeley developed the treaded abstract machine or TAM [19] for conventional multiple-processor systems. There are many other examples of multithreading including the MIND PIM architecture under development by Sterling at Caltech.

Cellular structures that employ fine grain devices interconnected to nearest neighbors have been employed for more than two decades in support of a number of paradigms [20]. Two of the most widely used are cellular automata invented by von Neumann [21] and systolic arrays invented by H.T. Kung [22] now at Harvard. Simple pipeline structures are trivial cases of this but support important computations such as gene mapping or string matching.

CCA can be viewed as an extreme extension of processor in memory taken to its limits. Under the DOE Office of Science Advanced Programming Models project led by Argonne National Laboratories, Sterling has explored a message-driven multithreaded futures-synchronized advanced programming model called “ParalleX” and is developing a intrinsic latency tolerant parallel programming language called “Agincourt”. ParalleX will provide the execution model for CCA and Agincourt will be suitable as a high level language for eventual programming of CCA systems. A project that has just started under the new DOE-sponsored Fast-OS program is developing an ultra lightweight kernel with services that are distributed among many fine grain computing elements such as multi-core processor chips, processor in memory chips, or sub arrays of CCA cells. Finally, a small two year project funded by DARPA starting in 1997 created the original concepts upon which CCA is based.

4. A Replicated Cell Architecture for CCA

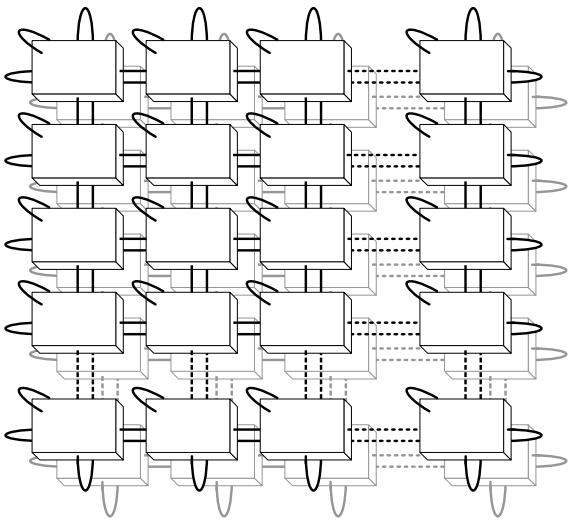

Figure 1. CCA System

An actual physical system will approximate the ideal CCA by discretizing the N-dimensional space into local cells. Each cell integrates the capabilities of state storage and addressing, nearest neighbor data transfer and routing, and state modification under program control. It also supports task creation and termination in the form of continuations, either as lightweight objects which are sedentary (stay in roughly the same place) or burtons which are nomadic (move from place to place). A cell is not a processor. There is no program counter in the classical sense, no dedicated register bank, no stack pointer. Yet, all of these may be emulated effectively with the resources of one or a few contiguous cells in cooperation. An idealized view of the CCA system for N = 3 is shown in Figure 1.

The major components of a cell, depicted in Figure 2, are the 1) associative tagged data blocks, 2) basic arithmetic/logic functional unit, 3) the burton decoder and dispatch controller, 4) the burton router, and 5) the trans-cell data paths and switch. Together, these provide the necessary mechanisms to support basic operations in response to incident burtons. A block of data comprises some number of words. Each block has associated with it an associative tag that establishes its virtual name and a type identifier which may either be a hardwired type or a programmer defined type. In addition, every byte has a set of status bits that can be used for conditions and synchronization. There are multiple data blocks in each cell; but the number of these is an optimization parameter and is to be determined. A block looks a little like a register bank and/or a cache line of classical systems. But it is neither. It is not a register bank because it is in the user global name space through its associative tag. Its not a cache line because it is not caching anything. That is where a particular data lives and there is no assumption of coherence with respect to some remote data. While a block will usually hold program data, it can also hold sequences of basic operations to provide thread instruction streams, global routing information, and synchronization state such as futures.

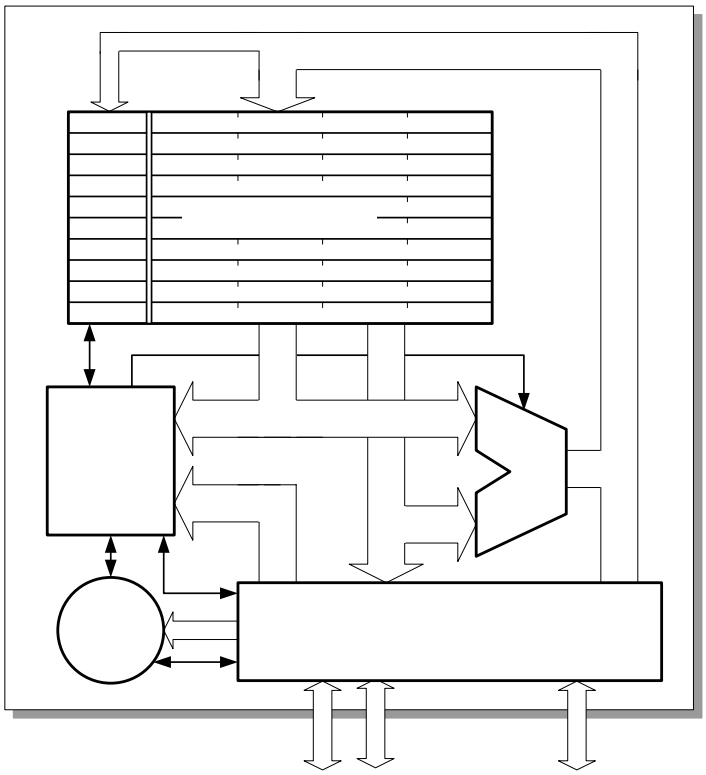

Figure 2. Structure of the CCA Cell

At the lowest level, the functional units are largely conventional with some minor adjustments. Their total capability is determined by the optimal space they can take up, balanced with that of the state storage blocks and the communication data paths. A functional unit may require multiple micro-cycles to perform an operation. The functional unit will directly support basic operations such as logical, integer adds, and permutation operations. It can also provide partial operations in support of more complex functions such as floating point arithmetic. A floating point operation may be distributed across a sequence of cells in a pipelined manner, each cell doing part of the total function. A burton carries the opcode(s) to be performed at its destination

cell on its target data and the control interprets and dispatches on these instructions in response to the arriving burton, where appropriate.

Most burtons just pass through the cell on their way to some destination determined by the location of a target user variable or data structure. The cell structure of CCA serves as a massive mesh communication fabric. Burtons are wormhole routed through the sequence of cells. The head of the burton indicates its type and the logical entity for which it is searching as well as the physical destination (or direction) it is moving towards/in. As the burton header enters a cell, the cell logic decides what action to take and may do an associative search (probably a single cycle) on the small stack of tagged data blocks to determine if the target variable is local. If not, it will pass the burton in the direction it is going with minimum delay. However, it may cause some rerouting to occur due to either implicit traffic control (avoiding conflicts using adaptive routing strategies) or explicit routing updates (through "crumbs" – intermediate points of redirection for search) for a given variable or block of data which may have moved from an assumed location. The cell contains sufficient control logic and data paths to carry out these communication actions.

The cell also supports more sophisticated operations on data for purposes of fine grain synchronization and task management. This is to realize the semantics of the futures and dataflow constructs which may involve multiple contiguous cells. These actions operate on the synchronization information embedded as part of the cell data blocks. Other functions enable garbage collection, burton creation, and advanced address management. Each cell is constantly "aware" of its neighbor cells and together accomplishes certain basic functions like data migration to make room for new data and to manage locality for improved performance through reduced latency. A major goal of this project is to determine and specify the precise set of capabilities and their logic level realization within the boundaries of each cell.

Since the physical resources encompassed by a domain for which the τ factor does not exceed 1 will likely be sufficient to build a single cell, all intra-cell operations can be executed synchronously. This has an added benefit of reusing the layout and logical design tools available today and, due to removal, or at least significant reduction of the number of pipelined blocks, cuts down on resources not directly related to the functionality of the cell. By contrast, the inter-cell communications are expected to cross the $\tau = 1$ boundary and thus require an explicitly asynchronous design.

5. Summary and Conclusions

Over the next decade, semiconductor device technology will continue to reflect and track Moore's Law until the effects of nanoscale feature size, near the atomic limit force a cessation of continued reduction in device size. Even though this eventual "flat-lining" of Moore's law will block continued expansion of device density beyond a certain limit, the extremes in technology scale and clock rate will demand innovations in architecture as dramatically different structures and operational modes will be required to effectively utilize achievable nanoscale technology.

Of paramount importance is that in the limit, locality of action will dominate operational behavior of nanoscale computing systems and that bandwidth of storage access will determine the ultimate sustained performance of systems of fixed size. While locality even today is of importance and drives much of the attention on cache based hierarchies, these effects are trivial

when compared to the degree and tightness of locality that will be imposed on nanoscale computing structures and methods. The metric τ introduced in this paper suggests at least a three order of magnitude difference between current technologies at or just below 0.1 micron and the asymptotic technologies at the end of the next decade that will exhibit basic feature sizes almost a hundred times smaller. New architectures will be required that enable the exploitation of unprecedented amounts of fine grain parallelism and that are latency tolerant for remote interactions. In addition, the overhead mechanisms for managing the parallel physical structures and concurrent logical activities must be very efficient to achieve real scalability through the massively parallel execution of fine grain tasks.

Among the possible alternatives for nanoscale architectures of the next decade, fine grain cellular structures provide the highest storage access bandwidths while delivering very high peak performance per unit die area. They also provide very high bi-section bandwidth for nearest neighbor communication assuming 3-D toroidal mesh topologies. In addition, cellular structures can operate asynchronously and under distributed rather than centralized control both of which is necessary under projected extremes in latency. With their highly replicated instantiation of identical elements, they provide the opportunity for fault tolerance through graceful degradation, if the necessary fault detection and fault isolation mechanisms are incorporated.

This paper has described the Continuum Computer Architecture for near nanoscale technologies that exploits fine grain cellular structures for general purpose parallel processing. CCA is also appropriate for other technologies that exhibit high τ values due to very high speed clock rates, such as superconductor RSFQ technologies. Unlike conventional cellular automata, CCA incorporates mechanisms in support of the semantics of a general and global model of parallel computing. The operation of the CCA highly parallel system is governed by the ParalleX computing model developed under the leadership of Argonne National Laboratory in collaboration with the University of Delaware and managed by an ultra lightweight kernel runtime software system being developed in collaboration with Sandia National Laboratory and the University of New Mexico. The computing model reflects a message-driven split-transaction processing discipline for efficient latency tolerant fine-grain parallel processing and relies on a combination of dataflow and futures based synchronization.

CCA may provide the ultimate convergent architecture for nanoscale and ultra high clock rate technologies at the end of Moore's Law. In the limit, it constrains the local computing element to the minimum size capable of performing a single operation through the merging of memory, logic, and communication hardware in a single component. Therefore, it will support the smallest possible feature size and highest clock rate possible. For even higher τ, the CCA cell can be pipelined and handle overlapping incident request packets.

References

1. Baker, H. G. and C. Hewitt. *The Incremental Garbage Collection of Processes*. in *Proceedings of Symposium on AI and Programming Languages*. 1977, pp. 55-59.
2. Lent, C. S., et al., *Quantum Cellular Automata.* Nanotechnology, 1993. **4**: pp. 49-57.
3. Likharev, K. K., *Rapid Single-Flux-Quantum Logic*, in *The New Superconducting Electronics*. 1993, Kluver. pp. 423-452.

4. Kang, J.-H., A. H. Worxham, and J. X. Przybysz, *4.6 GHz SFQ Shift Register and SFQ Pseudorandom Bit Sequence Generator*. IEEE Transactions on Applied Superconductivity, 1995. **5**(2).
5. Gao, G., et al. *Hybrid Technology Multithreaded Architecture*. in *Proceedings of the 6th Symposium on the Frontiers of Massively Parallel Computation (Frontiers '96)*. 1996, pp. 98-105.
6. Sterling, T. and L. Bergman. *A Design Analysis of a Hybrid Technology Multithreaded Architecture for Petaflops Scale Computation*. in *Conference Proceedings of the 1999 International Conference on Supercomputing*. 1999: ACM SIGARCH, pp. 286-293.
7. Chen, W., et al., *Rapid Single-Flux-Quantum T Flip-Flop Operating up to 770 GHz.* IEEE Transactions on Applied Superconductivity, 1999. **9**(2): pp. 3212-3215.
8. Hewitt, C. and H. G. Baker. *Actors and Continuous Functionals*. in *Proceedings of the IFIP Working Conference on Formal Description of Programming Concepts*. 1977, pp. 367-390.
9. Noakes, M. D., D. A. Wallach, and W. J. Dally. *The J-Machine Multicomputer: An Architectural Evaluation*. in *Proceedings of the 20th Annual International Symposium on Computer Architecture*. 1993: IEEE Computer Society Press, pp. 224-236.
10. Eicken, T. v., et al. *Active Messages: A Mechanism for Integrated Communication and Computation*. in *Proceedings the 19th Annual International Symposium on Computer Architecture*. 1992, pp. 256-266.
11. Culler, D. E., et al. *Parallel Programming in Split-C*. in *Proceedings of the Supercomputing '93*. 1993: IEEE Computer Society Press, pp. 262-273.
12. Draper, J., et al. *The Architecture of the DIVA Processing-In-Memory Chip*. in *Proceedings of the ACM International Conference on Supercomputing (ICS'02)*. 2002.
13. Sterling, T. and H. Zima. *Gilgamesh: A Multithreaded Processor-In-Memory Architecture for Petaflops Computing*. in *Proceedings of Supercomputing '02*. 2002.
14. Halstead, R. H., *MultiLisp - A language for concurrent symbolic computation*, in *ACM Transactions on Programming Languages and Systems*. 1985. pp. 501-538.
15. Nikhil, R. S., S. Pingali, and Arvind, *Id Nouveau.* Technical Report Memo 265. Computational Structures Group, Laboratory for Computer Science, MIT, 1986.
16. Alverson, R., et al. *The Tera computer system*. in *Proceedings of the 1990 International Conference on Supercomputing*. 1990, pp. 1-6.
17. Smith, B. J. *Architecture and Applications of the HEP Multiprocessor Computer System*. in *Proceedings of the SPIE - Real Time Signal Processing IV*. 1981, pp. 241-248.
18. Hum, H. H. J., et al., *A Study of the EARTH-MANNA Multithreaded System.* International Journal of Parallel Programming, 1996. **24**: pp. 319-347.
19. Culler, D. E., et al., *TAM - A Compiler Controlled Threaded Abstract Machine.* Journal of Parallel and Distributed Computing, 1993. **18**(3): pp. 347-370.
20. Toffoli, T. and N. Margolus, *Cellular Automata Machines: A New Environment for Modeling*. 1987: The MIT Press.
21. Neumann, J. v., *Theory of Self-Reproducing Automata*, ed. A. Burks. 1966: University of Illinois Press.
22. Kung, H. T. and C. E. Leiserson. *Systolic Arrays (for VLSI)*. in *Sparse Matrix Proceedings 1978*. 1979: Society for Industrial and Applied Mathematics, pp. 256-282.