Impact of Locality and Dimensionality Limits on Architectural Trends

by
Douglas J. Matzke

Texas Instruments
P.O. Box 655474, MS 446
Dallas, Texas 75265
matzke@hc.ti.com

Abstract

Since computing is a physical activity, all forms of computing must obey locality constraints imposed by physics. Unknowingly, many software abstractions violate locality constraints because they represent high dimensional topologies that have higher degrees of freedom than is uniformly implementable by the underlying physical architecture. This semantic gap between abstractions implemented in the virtual architecture and the physical machine resources results in poor performance for certain classes of computing problems. This paper will discuss and analyze the impact of locality constraints and dimensionality limits upon software and architecture trends with the specific goals of improved performance, lower cost, and the longevity of architectural investments.

1.0 Importance of Locality to Architecture

Much work in the physics and computing community focuses on physical limits to computing, but very little effort is spent on the applying the consequences of those limits to architecture and software design. In the not too distant future, the computing industry will reach the point were lithographic scaling is either not cost effective or not possible any more. Many semiconductor experts state that this "post shrink era" is at least 15 years away.

Therefore, the next 15 years represents a "semiconductor time of plenty" compared to the low transistor density our industry was able to achieve during the last 25 years. Will this bounty of computing resources result in more of the same kinds of architectures that have been historically developed? Can we use the insight from physics and computing research to develop a truly modular, scalable, parallel architecture that can survive this entire time period?

Traditional computer system design has evolved in an environment where computing resources were scarce and expensive. Semiconductor densities have reached the point of no longer being limited by the number of transistors but by the ability to cool them. Heat density will be a major concern for all future semiconductor generations.

Each successive semiconductor generation must add transistors without increasing the total power usage, in order to continue using packages with similar power ratings. For the next several generations, an easy solution exists of lowering the power supply voltage. After that grace period, the only long term approach to maintaining a constant power per die area is to limit the drive power (and thus the drive distance) of most gates in the design.

Architectures that depend on increasing amounts of circuit locality are therefore a direct consequence of power limitations. Since computation is a physical process and physics mandates locality, this guideline should not be surprising. In his 1981 paper, Daniel Hillis stated[1] that "computer science is missing many of the qualities that make the laws of physics so powerful: locality, symmetry, and invariance of scale." Of these, locality is the most critical and this perspective is used for the remainder of this paper to judge computer architectures trends and features.

Locality arguments are the motivation behind the cellular automata (CA) efforts[2] and coincides with the quantum dot requirements[3]. Despite their impressive results, neither of these research areas has had a major impact on the mainstay computing industry. The MIT CAM8 machine has demonstrated supercomputer performance for certain kinds of simulations, yet other proposed simulations can execute faster on conventional sequential computers.

Spatial locality goals alone appear not to be sufficient to make machines that are both universal and high per-

formance over many classes of problems. Simple thought experiments demonstrate that locality is important to high performance architectures. This paper is the result of an effort to understand the importance of locality to conventional architectures. Hopefully this understanding will become a guidepost for spotting which architectural trends will remain critical over time.

The approach of this document is to investigate the impact of locality on memory systems, grain size, scalability, the dimensionality of an algorithm, and the impact on the virtual architecture. Out of this analysis, some important software and architectural trends will be isolated and discussed.

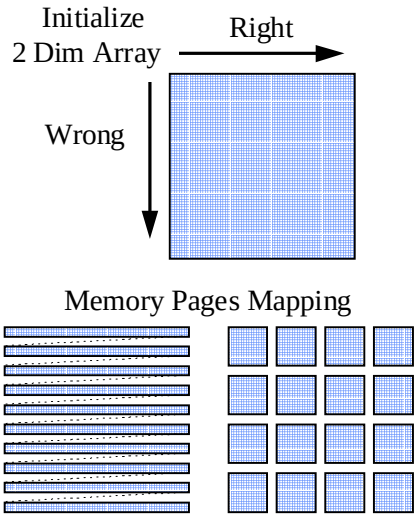
2.0 Spatial Locality within Virtual Memory

Virtual memory is now taken for granted on all modern computers large and small. Before virtual memory, programmers where much more aware of the physical nature of memory resources, especially when there were insufficient amounts of it. With the introduction of virtual memory and multiple processes (virtual CPU's), programming became more abstract and less tied to physical resources, which is both good and bad. The good part is programming became generally easier. The bad part is virtual resources are an abstraction based on physical resources and plus some important underlying assumptions. It is easy to write code that violates those assumptions and executes poorly.

The physical basis of the virtual memory system is a set of physical memory pages. When a page is referenced, it is retrieved from disk and placed in some physical memory resources. Parts of this page are then loaded into the fast caches of the processor which enable it to be accessed very fast. These pages are arranged in a large one dimensional address space that is accessible via a memory pointer. In order to achieve good performance from this virtual resource, the program must be well behaved by achieving good locality of references on a limited number of pages per unit time.

The spatial and temporal locality is a prerequisite for good program performance within a virtual memory system. Any pages that are active "appear close" to the processor and thus are fast to access. As demonstrated in the next paragraph, very simple programs can defeat the apparent locality of the pages to the processor. Some compilers detect and optimize the following kind of array reference code to enable good performance.

As shown in the following figure, mapping a two dimensional array (1000 x 1000) into virtual memory results in a preferred orientation. If the memory is laid out in memory by rows, then initializing the array row by row results in good performance with many accesses per page. Alternatively, accessing down the columns causes many page misses with 10x worse performance. This example assumes the entire data set does not fit into the physical memory of the machine and relies on the virtual memory mechanisms. Similar performance degradation can also occur as a result of the CPU data caches. Notice that any mapping will cause this non-symmetrical time performance, which is a direct result of mapping 2D semantics into a 1D implementation.



This example illustrates that virtual resources can disrupt the apparent locality (and execution speed) required by the application semantics. This non-symmetrical (or anisotropic) time performance is a direct result of trying to mimic a two or more dimensional space by embedding it in a one dimensional virtual memory. Applications that sweep across an entire large multidimensional data set will produce this effect because the memory pages create a preferred orientation or direction due to violating the locality of references. This becomes even more of a problem as the number of dimensions increases because the effective locality of each array element depends on the number of independent degrees of freedom required of the algorithm.

Densely packed data structures are represented using multidimensional arrays. Alternatively, sparsely packed data structures, such as object oriented graphs, are best represented using virtual memory pointers. Since a pointer represents a direct relationship between two unrelated pieces of physical memory, it semantically represents an

independent degree of freedom. This desired locality can be effected by the performance of the virtual memory system.

With modern object oriented programming tools it is easy to build objects with 8 or more outbound pointers. Each pointer represents a unique local direction that must be embedded into a one dimensional virtual memory system. Dynamic reclustering of active objects onto memory pages can improve performance [4] by reducing the total size of the working set, but requires readjusting of all the pointers in the memory space.

Pointers also point to program functions stored in virtual memory. Object oriented programming environments make it easier to build complex data structures and also makes it easy to build large, complex program structures. Functions are accessed by using pointers, and these degrees of freedom are above and beyond the explicit data dimensions. The following paragraph illustrates this point.

Imagine a distributed, 3D, object oriented, particle simulation, where each processor manages a volume of the simulation space. As each particle moves across a processor boundary, the current conditions it encounters would determine what functions are paged in from the common program store. Therefore, each object must have 3 regular data dimensions, plus a hidden dimension within each object to point to its private object functions. Even straight forward 3D data applications require 4D semantics when the program space is considered. Program size considerations have become a serious performance issue, because many applications automatically generate source code for compilation and execution. This practice results in very poor cache performance since the code size is much large than handwritten applications.

The MIT CAM8 group understands the relationship between locality and the number of explicit dimensions of the data set. The CAM8 machine is configurable to have 1D linear pages, 2D area pages, or a 3D volumetric memory configuration and the corresponding locality assumptions. The CAM8 machine has virtualized the memory dimensions and virtualized the number of cells per processor. The CAM8 does not have a true virtual memory facility thus limiting the maximum size problem for each piece of hardware[5]. The CAM8 does not directly address the virtual program size requirements because each processor can only contain two sets of global rules at a time.

Clearly virtual resources are valuable, but ignoring higher dimensional semantics (both data and program) can lead to performance problems in conventional virtual memory systems. Allowing layers of abstraction, without maintaining control over physical facilities impacting locality, can degrade performance. Pointers are an example of a primitive implementation feature with insufficient abstraction facilities and program controls.

3.0 Locality Impact on Grain Size

This section will discuss locality concerns regarding architecture grain size. Choosing the correct virtual grain size for an architecture can dramatically impact the cost, performance and flexibility. For example, many supercomputers manufacturers build high speed 32 bit wide microprocessors boards as their basic grain size. Field Programmable Gate Arrays (FPGAs) have a 1 bit wide grain size block that runs from 5 to 25 logic gates per block.

Conventional computers are made of transistors, logic gates, and wires that creates a discrete locality grid for both space and time. The light cone defines locality in physics by linking unit space and time together. Likewise, the distance (through wires and gates) that a signal must propagate determines the maximum frequency it can operate. Therefore, smaller grain computational elements can run faster than larger grain ones. Since each computational element has an area (or volume), all of these metrics are related for a specific lithography size.

Besides basic grain size impacting the maximum operating frequency, it also directly impacts the maximum performance due to parallelism. William Dally of MIT[6] has shown that cost balanced machines can have 50 times the overall memory bandwidth of conventional processors by maintaining the memory to logic die area ratio to be 10 (or about 1 Mbyte per 32 bit processor). His argument is that conventional processors are performance biased resulting in very large caches and very poor MIPS per unit area of total silicon. At fault is the traditional trend of CPU architects being performance oriented and memory systems being cost oriented.

Dally's approach is to look at cost and performance in relationship to the total die area of both the processor and memory. This idea of a basic building block containing both memory and logic at a 10 to 1 ratio is important because it suggests that the basic building block should be an "active memory". This approach is consistent with a modern relativity notion of a unified spacetime where space (or memory) and time (or CPU) are inseparable. Both cellular automata and object oriented paradigms have adopted an "active data" orientation to data structures.

Dally's parallel architecture approach concludes that smaller grain size "active memory" building blocks can result in higher performance for lower cost. The monolithic Von Neumann architecture results in the "Von-Neumann bottleneck" between the CPU and memory. This bottleneck should be relabeled the "Newtonian bottleneck" because computer architects are using last century's Euclidean notions of segregated space and time. Considering locality and scalability arguments, modular active memory building blocks align with modern notions of a unified spacetime.

Addressability of elements within a specific grain size is another important architectural decision. As the number of transistors per die has increased, the conventional processors have moved from 32 to 64 bit wide addresses and datapaths. Any increase in the number of computational element requires a wider address datapath in order to allow more address bits. The pointer size of a memory address is important because many object oriented applications use pointers to express relationships. In some applications, pointers occupy more bits expressing the relationships between data than is occupied by the data itself. For applications with lots of pointers, it might be more efficient to place two 32 bit processors per die than one 64 bit processor.

Applications with many pointers existing inside large monolithic address bases have the problem that they are not relocatable without translating all the pointers. Physics always acts in a local manner and has nothing that matches the nonlocal behavior of a pointer. Cellular automata machines are modeled after physics and therefore only represent directions without any pointers. Unfortunately, it is difficult and cumbersome to program without pointers, especially when pointers act as object ids for determining when two instances are the exactly the same object.

Once the grain size sets a limit on the maximum speed and number addressable elements (per unit volume), than additional parallelism and additional elements must come from multiple instances of that basic building block. Inside and outside addressability via a network are both important issues resulting from grain size issues. It is clear from this discussion that grain size, pointers, and networks are intimately related. Somehow future architectures and compilers must deal with the grain size and pointer issues to allow a family of grain size options giving high performance and low cost for a variety of applications.

4.0 Locality Impact on Scalability

Despite of the size of the physical universe, locality dominates its organization. Therefore it would seem that locality should be a prerequisite for scalability. In reality, two kinds of scalability exist and their locality arguments must be addressed separately. Upward scalability deals with maximum system size concerns and downward scalability deals with grain size and fabrication size issues. As discussed earlier, power restrictions and circuit locality dominates downward scalability concerns.

4.1 Locality and Degrees of Freedom

As architects of hypercube machines have discovered the hard way, not all computer network configurations can scale in size in 3 dimensions. Franco Preparata [7] has formally shown that only the mesh architecture is truly upward scalable in 3D. In addition, Preparata showed that downward scalability results in many unforeseen architectural restrictions (e.g. ineffectiveness of latency hiding) can surface when fundamental physical limits are reached.

Due to scalability reasons, Preparata concluded that parallel computer architectures will become more uniform over time. It is obvious that no company would want to invest in a suboptimal or short lived architecture. If performance and cost are also considered, the individual active memory nodes will also become more uniform over time.

Potentially a uniform, spatially oriented, computational work metric based on locality measures could define the degree of boundedness for both algorithms and architectures. For example, the number of implicit dimensions d, represents the measure of the amount of locality seen per node, but also represents a discrete choice of 2d directions. This choice requires log 2d addressing bits. Locality and scalability metrics could ultimately determine most architectures decisions.

Architects build networks of high degree because the problems they are attacking can benefit from more bandwidth and higher degrees of freedom. Unfortunately these systems are not scalable. This simply is the ultimate semantic gap between application semantics and physically realizable architectures. An architecture must either limit its degrees of freedom to 3 dimensions or limit its scalability. This restriction to computer architecture is intriguing since some theories underlying physics suggest that the universe has a 10 dimensional symmetry [8], which represents a better locality metric.

Locality issues completely control algorithm scalability (both performance and cost) for high dimensional algorithms. Languages that are designed to specifically to deal with high dimensional semantics may emerge [9,10]. This geometrical approach could remove raw pointers from being directly programmed to manage physical topology issues. Using this geometrical information, real time compiler techniques may allow execution costs to be predicted and managed.

Pointers are useful because they implement degrees of freedom for software and architecture, but they are semantically very weak. Pointers are the spatial equivalent of microcode and applications programmers should not need to use them. The development of stronger locality based geometrical abstractions and languages would allow virtual memory pointers to be relegated to the architectural dungeons where they belong.

4.2 Locality Impact on Virtual Architecture

Building architectures that deal more effectively with larger amounts of locality will present a smaller semantic gap for high dimensional languages and applications. Similar in nature to the CAM8 and FPGAs, the trend towards locality constrained volumetric computers will continue at the system architecture level and also at the device fabrication level. Mapping high dimensional algorithms into 3D mesh architectures will move from supercomputers down into workstations and personal computers as application specific compilers effectively deal with scalable, smaller grain size, active memory chip organization. Cost and performance issues will dictate the remaining architecture choices regarding grain size and total amounts of parallelism.

This product concept is already emerging in custom computing machines by using VHDL to program accelerator boards full of FPGA chips[11]. Currently programming these devices involves writing logic schematics and using automatic place and route tools. More general purpose medium grain, parallel systems, programmed by geometrically oriented object languages may be the future of custom computing accelerators.

Unless some breakthrough occurs in compiler technology, programming will become more physical and less abstract, in order to gain performance in parallel systems. Being able to predict performance, partition, and make space/time tradeoffs would require real time compiler techniques based on intimate knowledge of the architecture performance. Some new algebraic logic systems [12,13] uniformly represent both space and time resources and may be the key to better compiler technology. Most complex architectures are worthless without good languages and compilers.

5.0 Conclusion

The computer industry and its customers will continue to expect computer performance to double every year and a half. In the face of physical limits, this will only occur if the semantic gap between applications and architectures decreases. For the most demanding applications with higher dimensional semantics, this will only occur if the memory architectures become more physical, especially to support the 2D and 3D data volumes seen by physical simulations using finite element analysis.

Software languages and programming also must become more physical by adding geometrical based abstractions and reducing the dependency on semantically weak implementation mechanisms, such at virtual memory pointers and globally shared memories. Medium grain parallel, mesh connected, active memory architectures will be the targeted output from the real time, geometrically oriented object compilers.

These results were obtained by analyzing upward and downward scalability concerns to many key architectural ingredients of modern computers. Virtual memories will grow from one dimensional to three dimensional (or more). Languages will evolve from relational to geometrical and from sequential to distributed. Compilers and software tools will bridge the semantic gap, not the architectures. The architectures will be trying to support computational spaces that have more performance, locality, symmetry, and scalability requirements then current architectures.

Low cost and low power silicon solutions with these architectural characteristics will emerge, that others must either follow or lose market share. Locality concerns will ultimately impact all performance and cost decisions driving future systems to be architecturally more uniform, resulting in the longevity of architecture investments.

References

[1] Daniel Hillis, 1981, “New Computer Architectures and their Relationship to Physics or Why Computer Science is No Good”, *International Journal of Theoretical Physics*, Vol. 21, Nos 3/4, Pages 255-262.

[2] Norman Margolus and Tom Toffoli, December 1993: "CAM-8: A Computer Architecture based on Cellular Automata", Technical Report MIT CAM8 Group.

[3] Robert Bate, Gary Frazier, William Frensley, and Mark Reed, July/Aug. 1989: “An Overview of Nanoelectronics” , *TI Technical Journal*.

[4] Doug Johnson, April 1991: “The Case for a Read Barrier”, *Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS IV), Pages 96-107.

[5] N. Margolus, 1994: "Virtual Processor Emulation of Large Cellular Logic Systems", Technical Report MIT CAM8 group.

[6] William Dally, 1993: "A Universal Parallel Computer Architecture", *New Generation Computing*, vol. 11, pages 227-249.

[7] Franco Preparata, May 1993: "Horizons of Parallel Computation", Brown University, Technical Report No. CS-93-20.

[8] Kaku, Michio. 1994. *Hyperspace, A Scientific Odyssey Through Parallel Universes, Time Warps, and the Tenth Dimension*. Oxford University Press

[9] Theodore Omtzigt, 1991: “SagaDB: The Domain C Graphical Debugger”, Dept. of Electrical Engineering Report, Yale University.

[10] Theodore Omtzigt, Nov. 1994: “Computational Spacetimes”, *Proceedings of Workshop on Physics and Computation*, IEEE Computer Society Press.

[11] Jonathan Babb, et al, April 1994: “Emulation of the Sparcle microprocessor with the MIT Virtual Wires Emulation System”, *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, IEEE Computer Society Press.

[12] G. Spencer Brown, 1994: *Laws of Form*, Bookmasters, Ashland Ohio.

[13] Dick Shoup, Oct. 1992: “A Complex Logic for Computation with Simple Interpretations for Physics”, *Proceedings of the Workshop on Physics and Computation*, IEEE Computer Society Press.