High performance reconfigurable computing with cellular automata

Murtaza, S.

Publication date
2010

Chapter 9

Manycore Paradigm – What, Why and How?

The previous chapters demonstrated the performance modeling and evaluation of CAs with hundreds of processing elements computing millions of cells. This chapter focuses on running massively parallel systems capable of crunching numbers at a rate of 10^{18} operations per second. The chapter starts with a literature review of the manycore paradigm-a requirement to reach the exaflops scale [2, 67, 88]. Some of the important publications reviewed in this chapter include - a report on DARPA IPTO sponsored ExaScale study [67], a report on parallel computing research compiled in December 2006 by a multi-disciplinary group of researchers from Berkeley over a span of two years [5], and work presented by 50 leading experts from academia and industry at EDF organised workshop on using million cores systems [2].

Within the framework of the literature review, we analyse our FPGA based CA implementations and draw parallels between the two. Most importantly we are keen to find out how our work contributes to the existing research. Following are some of the main highlights of the literature review as presented in the following sections:

- The traditional model of exploring hardware and software as a single-domain research initiative in isolation can no longer address the current challenges [67].
- Much can be learned by examining the success of parallelism at the extremes of computing spectrum like high performance computing [5].
- Our ability to scale up applications to millions of processors, or even port conventional codes to a few dozen cores is almost non-existent [67].
- Widening, mitigating, or eliminating the von Neumann bottleneck must be the thrust of research [61, 67].

9.1 A Marriage of Convenience – von Neumann and Moore

Computer simulation is the key high performance computing technique to find solutions to global challenges [49] ranging from weather forecast to terrorist threats [15]. Running HPC simulations demand improvements in computational accuracy and speed, and it is no surprise that the major milestones in the high performance computing are pushed by the emergence of faster computer systems. In general, when the aggregate performance of a computer system first crosses a threshold of 10^{3k} operations per second, for some k, it is defined as a major milestone [67]. Billion floating point operations per second or Gigascale (10^9) was first achieved by Cray-2 in 1985 [108], the next milestone Terascale (10^{12}) was reached almost a decade later in 1997 by the Intel ASCI Red system at Sandia National Laboratory and the most recent milestone of Petascale (10^{15}) was achieved by Roadrunner at Los Alamos National Laboratory in 2008 [6]. Considering the time line with previous milestones and assuming the continued acceleration of progress, Exascale (10^{18}) computing is expected to be around in 2015 [67].

Consistent advances in the CMOS technology have been one of the main building blocks behind these milestones. Technological advancement has resulted in smaller logic and shorter signaling distances within the processor cores, and has been primarily used to enable higher processor clock frequencies, resulting in faster processor computations. Also known as the famous Moore's law, this has been interpreted as the doubling of the transistor density per unit area on a chip every 18-24 months. Moore's law has also been misinterpreted as doubling of performance of the general purpose computer (von Neumann architecture) every 24 months.

The von Neumann architecture, a term coined in 1945 [114], refers to a sequential computer architecture with a processing and a memory unit where memory stores both instructions and the data. Predominantly, it is a sequential thread execution model, where a program is a sequence of instructions that are executed logically one at a time, until completion. This architecture has been the dominant computing paradigm ever since and what we also call as general purpose computing [114].

9.2 The Three Walls

Till 2000, with each turn of the Moore's Law, processors kept getting faster. Simultaneously, in terms of access time and data bandwidth, memory has not been able to keep up with the improvements in processor clock rates. As a consequence, modern computers end up spending most of its time moving data rather than processing the same. This limited data transfer rate between the processing unit and the memory unit, also known as the von Neumann bottleneck [114], has been one of the main performance limiters of the von Neumann architecture ever since its inception.

To get around this memory wall, a great deal of complexity was pushed within the processor architecture. And in an attempt to keep most of the relevant data for a program close to the processing logic and not violate sequential execution model [67], architecture include deep memory hierarchies also known as multi-level caches. For example, one of the existing fastest super computers, the IBM BG/L [107] can accomplish four floating-point mathematical operations in one clock cycle but requires around 100 such cycles to fetch each operand from the memory [67]. Even worse, since 2004, advances in the uniprocessor performance also dropped down to about 20% per year due to: power dissipation also called power wall and cooling restrictions; limited instruction-level parallelism (ILP) left to exploit; and almost unchanged memory latency [68]. As a consequence the three walls (power, memory and ILP) forced the computer industry to change its course from a uniprocessor to a multiple cores per processor systems; with Intel cancelling its high performance uniprocessor project, and joining IBM and Sun Microsystems in envisioning high performance microprocessors based on multicore processors [5, 54, 68].

9.3 The von Neumann Architecture and Multicore Bond

Moore's Law is still going strong; and in an attempt to exploit improved feature-size and logic density, modern processors also include multiple cores per die called multicore. Other than increasing the number of functional units per chip or the spatial efficiency, multicore architecture also limits the energy consumption per operation and constrains the uniprocessor complexity growth [98].

Most of the current popular multicore processors are based on symmetric multiprocessor (SMP) architecture [113] where the few available powerful cores share the same memory in a symmetric manner, and each core is a conventional processor, executing as per the von Neumann model. This trend of simply replicating conventional processor cores on the same die, by the main stream processor industry, faces some serious limitations. These limitations include situations wherein, a) workloads with multiple sequential threads would benefit but how will individual tasks become faster [5], b) the complexity within the processor architecture is further increased, with multiple cores attempting to share the same memory, and collaborate by explicitly executing separate parts of a program in parallel [67], and c) a single shared memory will end up as a bottleneck as the number of cores increases and the paradigm is expected to reach a dead end with the number of cores reaching sixteen or so [68].

Moreover, [5] states, "Switching from sequential to modestly parallel computing will make programming much more difficult without rewarding this greater effort with a dramatic improvement in power-performance. Hence, multicore is unlikely to be the ideal answer." And the more challenging tasks with multicore that remain at large include: effectively exploiting multiple-thread parallelism; parallel computing and programming models; aggravated memory wall; slow and flatten rate of pin growth or the interface between the chip to the outside world; and the mechanisms required for efficient inter-process coordination like synchronisation, mutual exclusion and context switching.

Although software and hardware architects are still in the infancy of multicore era, multicore have consolidated their place as being the building blocks of the fastest current supercomputer called Roadrunner [6, 112]. Roadrunner is a hybrid system built out of two different multicore processor architectures, dual-core Opteron processor from AMD and PowerXCell 8i- a cell processor from IBM, reaching the Petascale milestone at Los Alamos National Laboratory in May 2008. One of the proposed machine and programming model for the era of multicore chips are described in [62].

9.4 Exascale Computing

Chips with hundreds of cores are already in the market, for example, 130nm process based single chip with 188 RISC cores from Cisco [5]. More recently, Nvidia announced its latest Fermi GPU [80] with 512 cores assumed to reach a peak performance of 0.5 to 1 teraflops [44]. Evidently Moore's Law is pushing the idea of having manycores per chip and bringing the supercomputing capabilities to everyday devices [53]. This growth of cores per chip is in line with what is required to reach Exascale [88]. [67] describes Exascale systems as, "A 2015 data center sized capacity system is one whose goal would be to allow roughly 1,000X the production of results from the same applications that run in 2010 on the Petascale systems of the time." And to achieve this 1000X increase in the aggregate computational rates, [67] explains that either by improving the computational speed of a single program by three orders of magnitude or have the capability to concurrently run 1000 such jobs at the same time. It is not only about improving the computational capabilities but also the need to reduce the system sizes that perform current level of computing into far smaller packages [67].

9.4.1 Why?

With reference to Exascale systems and related applications, [67] writes, “there is a continuing need to run critical applications at much higher rates of performance than using Petascale machines and such applications are evolving in more complex entities when compared to those of the Terascale era.” It further points out that it is not only the classical scientific and engineering applications but also the applications, requirements related to the internet boom in general that demand such systems.

The enormous amount of data moved around the internet via applications like internet commerce, sharing multimedia content, social networking, blogging, search, news, 24/7 connectivity to internet over mobile devices, and many more have pushed the limits of required computing infrastructure both in terms of real-time performance and efficiency. Supporting internet in general requires computing resources and infrastructure at the speed rivaling those of any supercomputer. Situations demanding immediate attention include: power requirements, storage space, physical space, speed etc. Considering such computing requirements in general, [67] mentions, “thus in a real sense the need for advanced computing has grown significantly beyond the need for just flops”, and to recognise this fact [67] introduces the use of terms Gigascale, Terascale, Petascale, etc to reflect such systems.

Several studies and workshops, like [2, 67], discuss questions related to manycore systems and the Exascale computing in details. Such studies clearly indicate that the HPC community aims to reach Exascale milestone by 2015 [67].

9.4.2 How?

A serious discussion concerning the Petascale computing was published around 1994 [99] and only after a span of 16 years, in 2008 the first Petascale milestone was actually reached [112]. However, how the system would evolve in term of the technologies, architectures and programming models was known and this was similar to the Terascale machines which were remarkably foreseen early [67]. On the contrary, another 1000X performance would require altogether a different set of tools both in hardware and software as is evident from some of the following comments from industry experts, “To achieve Exascale performance, single, dual or even quad core systems frequency and ops/cycle might improve by 2-4X in next the 8-10 years, only opportunity for dramatic performance is in number of compute engines” [103] and b) “all that is needed to reach Exascale is a 5-10 TF/chip and 100-200K of such chips” [88].

The message is clear that the manycore systems are required to reach the Exascale [103], and we have to look for methods on how to map and glue hardware and software together guaranteeing overall system efficiency both in terms of solving and specifying the problem. Before we hit the road towards Exascale computing, let us consider some of the key challenges and how these are different compared to the Terascale era.

9.4.3 Challenges are Different

Previously Moore's Law played a significant role in terms of doubling the processor performance every two years, but this is no longer valid as it has hit the power wall; and has essentially resulted in flattening of the clock rates ever since. This also marks the end of dependence on the performance improvements with a single threaded model and pushing the needs to have processor architecture exploit concurrency and locality [67, 88]. If parallelism is the only mechanism in silicon to increase overall system performance then how to exploit parallelism remains a challenge [67]. Towards the software side, the challenge is to find enough concurrency in the applications [88].

Another challenge is to design a fault tolerant system. Smaller feature sizes, design, manufacturing, and low-energy operation requirements, are likely to suffer from higher failure rates due to higher thermal and quantum error rates [120]. Each new process generation, as a result, rapidly increases permanent and transient faults within the chip and therefore, designing a chip with irregular structures becomes increasingly difficult [120]. Hence it is no surprise that fault tolerant design has been cited as one of the major challenges of the future technologies that are expected to mitigate a rapid reduction of yield and reliability [5]. Consequently, [49] calls for a paradigm shift where focus should to be more on fault tolerant software rather than on eliminating failures. Related challenges include: scalability issues with monitoring and notification [49]; development of scalable and naturally fault tolerant parallel algorithms [49]; identifying the class of problems that can be made scalable and self-healing [49]; the type of cooperation between the cores both within and outside a chip [70]; and the overall hardware and software resiliency [88].

Thirdly, right from the start, memory technologies were unable to keep pace with increase in processing speed. With manycore systems the situation gets worse, limiting performance of future designs [23, 88]. How to limit the aggravated memory wall, remains a challenge.

Designing a modern processor itself has become a mammoth challenge. Processor design using latest technologies is quite an expensive and sophisticated job that only few companies can afford. Therefore, the traditional model of exploring hardware and software as a single-domain research initiative in isolation can no longer address the current challenges [67].

Scalability is yet another challenge. There are no clues on how to scale up applications to the millions of processors and even porting conventional programs to a few dozen cores is almost non-existent [67].

With respect to the unique challenges we are facing with the manycore paradigm, [67] concludes that the current technology trends are simply insufficient and to bring alternatives in-line requires explicit direction and support in new research, otherwise a wait of another 16 years will not guarantee the emergence of Exascale systems.

9.4.4 Dos and Don'ts

In the face of the manycore challenges, some of the recommendations by the industry and academia experts are as follows:

Think beyond von Neumann architecture: Academia and industry based manycore focus groups like [5, 61, 67] expect the solutions to be beyond the nature of von Neumann architecture. [61] suggest to look for new computation models that either widen or mitigate the von Neumann bottlenecks and system that provide self-healing and trustworthy hardware and software. [5] call for models that are completely not von Neumann in nature and that exploit Moore's Law in a way different from what has been done historically.

Multicore is unlikely to be the ideal answer: Focus on the manycore instead of multicore systems [5]. UCB review [5] announce a desperate need for new solutions for parallel software and hardware, instead of relying on evolutionary methods to address problems of parallelism via multicore solutions. [5] goes further to write, "It is unwise to presume that multicore architecture and programming models suitable for 2 to 32 processors can incrementally evolve to serve manycore systems of 1000s of processors", and concludes: evolutionary approaches to parallel hardware and software are not going to scale beyond 16 and 32 processors; and switching from sequential to modestly parallel computing techniques, used with multicore, make programming difficult.

Learn from the extremes of computing: The years of experience gained from embedded and high performance computing are highlighted as the launching pads. [5] points out that embedded and high performance computing, the two extremes of the computing spectrum, are both concerned with challenges like power, hardware utilisation, feature size, unauthorised access and viruses. Therefore much can be learned by examining the success of parallelism there. Plus much can be learned from MPI based implementation [5] as MPI will remain where HPC applications may become a set of VMs [13].

Use Dwarfs as stand-ins: To understand and learn how to build manycore systems, [5] recommend using 13 benchmark dwarfs (categories of applications that would target

Dwarf	Description
Dense Linear Algebra	Dense datasets
Sparse Linear Algebra	Data with many zeros
Spectral Methods	FFT
N-Body Methods	Interactions between many discrete points
Structured Grids	Regular grid
Unstructured Grids	Irregular grid
Monte Carlo	Embarrassingly parallel computation on subsets of data
Combinational Logic	Simple operations on massive amounts of data, e.g., encryption
Graph Traversal	Indirect lookups in search
Dynamic Programming	Problem solving via solving simpler sub-problems
Backtrack and Branch-and-Bound	Global optimisation for huge search spaces
Graphical Models	Bayesian Networks and Hidden Markov Models
Finite State Machines	Interconnected States as used in parsing; embarrassingly sequential

Table 9.1: *13 Dwarfs. A benchmark suite from [5], that captures a pattern of computation and communication common to a class of important applications that can help draw broader conclusion about the parallel computing requirements of the future. See [5] for full details.*

manycore systems as shown in Table 9.1, see [5] for details) as a promising approach and using FPGA based system emulators for rapid design space explorations.

Some of the notable initiatives towards Exascale computing include; Chapel- A new parallel language developed by Cray for HPCS [88] and Corona- Optically enabled 256 core processor (10 TFLOP 3D chip) [13]

9.5 Limits of CMOS and Beyond

The existing CMOS technologies feature size of 45nm with an integration capacity of 8 billion transistors is expected to reach 8nm with 256 billion transistors by 2018 [16, 17, 18]. Pushing the limits of CMOS, these numbers translate to having a chip with thousands of simple processors in the near future [53]. With billions of nanoscale electronic devices per square centimeter, the challenge is to design a system that addresses issues like fault tolerance, the three walls, and the power performance ratio as presented in the previous sections. Such requirements have dictated a shift towards scalable and highly parallel microprocessor architectures. Architectures like [12] are actively being studied and some seek to explore different information processing and computing approaches like the nanoscale dynamics systems [2, 120].

[5] states that the trade off between power and performance is going to be the most important factors across the entire spectrum of current and future system applications. Therefore, the current trend is to go manycore with a hope that it will lead us to the Promised Land. Benefits of having a massively parallel architecture with smaller processing units include: ability to perform dynamic voltage scaling and power control at a fine-grain level (and therefore easier to shut down in the face of defects); and easy to

predict their performance and power characteristics [5]. Another advantage of having a massively parallel architecture is its inherent regularity, regularity within the hardware simplifies the diagnoses and isolation of faulty nodes and therefore simplifies fault tolerant implementations. Other advantages include ease of design and functional verification, and an energy efficient way to achieve performance [5].

Assuming availability of manycore systems as inevitable, one would like to know how its processing units, memory and the interconnect network are going to evolve. Before summarising what current technology literature recommends, let us consider some scenarios based on our own experience of massively parallel application implementations as presented in previous chapters in an attempt to get some feel about the deployment of such systems in the future.

9.5.1 Manycore Systems and CA Simulations

Few things are clear about manycore chips of the future: hundreds of simple cores per chip; massively parallel hardware and software setup; and scalable architecture for optimal power-performance ratio. Plus features like, the regular structure for feasible fabrication of chip, configurable processing units tailored to suit application requirements, and using 13 benchmark dwarfs as stand-ins for parallel applications of the future, make evaluation of such systems more realistic and easy. As expected, lot of research in programming models needs to be carried out in terms of how to parallelise conventional sequential codes to massively parallel paradigm. For now we have one less thing to think about as CAs are inherently parallel with structured grids (also among the list of dwarfs, see Table 9.1 for details), and at the same time are a benchmark application that would target manycore systems of the future. Our Maxwell based LBM implementation with tens of processing units computing millions of cells in parallel also has much in common with manycore architectures of the future [5, 10]. Therefore, first let us consider the Maxwell system and find out how much performance we can squeeze out of it, based on our performance model implementation.

9.5.1.1 Squeezing Maxwell

First some assumptions, all of the Maxwell's 64-FPGAs are of the same type (see Appendix 10 for exact details), that is, each FPGA runs 61 LBM PEs (existing multi-FPGA based implementation could only accommodate 8 out of the algorithm's theoretical maximum of 61 PEs), and in aggregate all FPGAs on-board memory banks store around 214 million LBM cells (each FPGA comes attached with a source memory bank of size 256 Mbytes and each cell size is 80 bytes, see Chapter 5 for details). With this available

k (cells read in parallel)	0.1	0.2	0.5
p (PEs per FPGA)	61	123	307
GFLOPS (150 MHz clock freq)	1.43	2.86	7.16
GFLOPS (250 MHz clock freq)	2.39	4.77	10.9
GFLOPS (500 MHz clock freq)	4.77	9.55	23.9
GFLOPS (1.0 GHz clock freq)	9.55	19.1	47.7
GFLOPS (1.5 GHz clock freq)	14.3	28.6	71.6

Table 9.2: *A 64-FPGA parallel system based D2Q9 LBM performance prediction as shown in Equation (7.4). Based on a combination of different values for the given system parameters like number of PEs per FPGA, FPGA and on-board memory interface, and clock frequency. The performance for a particular clock frequency and specified in GFLOPS represents a theoretical maximum a single FPGA chip can achieve. Note, single FPGA performance is a part of 64-FPGA D2Q9 LBM simulations size of 214 million cells.*

hardware running at 150 MHz, a 1-D domain decomposition of 1000x214400 D2Q9 LBM lattice, spreads over 64 FPGA, takeing 0.25 seconds to compute a single iteration based on the performance model in Equation (7.4). Each LBM cell update includes 38 sums, 27 multiplications and 10 divisions, and applying Dongarra's metrics (where sum or a multiplication is a single FLOP and division counts as four FLOPS) as explained in [69], takes 105 FLOP. Therefore, a single iteration computation spread over 64 FPGAs in parallel, performs 22.5 billion FLOP in 0.25 seconds, in other words a peak performance of 90 GFLOPS. To simplify things, lets say a peak performance of 1.4 GFLOPS per FPGA. Up to this point the only unreal thing about the given numbers is 61 PEs per FPGA. Theoretically, this is within the limits of the model, but it is difficult to achieve considering logic capacity of the current FPGAs.

Moving ahead, lets start pushing the given set of parameters of the model. Clock frequency can be increased say up to 1.5 GHz as this is a standard driving frequency for conventional laptop cores. Memory bandwidth can also be increased say from 64-pin to 320-pin interface (each for read and write operation). Increase in data-pin interface also results in increase of theoretical maximum for PEs per FPGA.

As shown in Table 9.2, the core clock frequency when improved by one order combined with five times improved memory bandwidth, results in the improvement of the overall performance per FPGA chip from 1.4 to 71.6 GFLOPS. However, a FPGA with 307 PEs and running at 1.5 GHz is quite distant from the current generation of FPGAs. Fitting 16 PEs per FPGA, as presented in the previous chapters, has been quite a challenge especially for the design to meet signal propagation timing requirements. One of the main hurdles faced during the FPGA based LBM implementation has been the long wire lengths running between a PE and the memory bank interfaces. With the increase in PEs per FPGA the wire lengths increase proportionally, and the design required more registering around signals both to boost signal strength and meet design timing constraints. This also reduced the driving clock frequency of the circuit. Other than that, the main limitations of this layout is the maximum possible number of PE for a given set of parameters, especially the cell update time. With improved cell update time (that is, consuming less than 675 core clock cycles to update a cell), the theoretical maximum of PE goes down, making this a major limitation. What is desired, is a layout with as many computationally efficient PEs as possible; and not to forget we are looking at thousand cores chips in future and application needs to be scaled accordingly in order to achieve the desired power-performance efficiency.

9.5.1.2 One-to-one Mapping

Let us consider a pure CA hardware implementation to get around the performance model limitation with the number of PEs per chip. Since we assume availability of thousand core chips and our application of interest being inherently parallel algorithm with local connectivity requirements, why not consider a one-to-one mapping?

To make calculations simple, let us focus on a single chip implementation and find out the possibilities of how to improve overall performance. Consider, a 32x32 D2Q9 lattice size with solid or bounce-back boundary conditions, 32-bit single precision floating points for state representations and a 1024 core processor platform. With these assumptions, a one-to-one mapping requires: each cell directly connects to eight of its neighbours; each core has its own floating point unit; and an IO interface for global communication requirements. One of the main strength of this mapping is the uniform hardware structure with even distribution of power-performance ratio. Cell connectivity confined to local neighbours means shorter wire lengths other than the global signals for synchronisation and control. With direct cell-to-cell connectivity resulting in short signal lengths, cores can be run at higher clock frequency. One of the main disadvantages of having one-to-one mapping is the local cell-to-cell data width requirements. Here each cell is to send and receive a 32-bit number from eight of its neighbours and that sums to numerous wires per cell. The situation becomes worse when a one-to-one mapping setup spreads over multiple manycore chips. One solution is to have a bit serial data transfer among cells.

Let us consider some numbers, 105 FLOP for the collision, 675 core clocks ticks to compute a collision, another 100 core clock ticks to exchange data across neighbouring cells (propagation part) since this is a one-to-one mapping, and a clock frequency of 150 MHz. 675 cycles to compute 105 FLOP per cell running at 150 MHz is based on a) our

Core clock ticks to compute a cell	675	338	168
Core clock ticks to exchange boundary	100	100	100
GFLOPS (150 MHz clock freq)	20.8	36.8	60.2
GFLOPS (250 MHz clock freq)	34.7	61.4	100.0
GFLOPS (1.0 GHz clock freq)	146.6	261.8	431.7
GFLOPS (2.5 GHz clock freq)	378.8	677.9	1117.6

Table 9.3: *A 1024 cores processor. D2Q9 LBM lattice size of 1024 cells mapped to 1024 core processor with required cell-to-cell local connectivity. Based on a combination of different values for the given system parameters like core clock ticks to update a cell and cell-to-cell data exchange. The performance for a particular clock frequency and specified in GFLOPS represents a theoretical maximum that a single FPGA chip can achieve.*

LBM implementations and b) considering a bit serial data transfer between cells also at 150 MHz consuming 100 clock cycles.

As shown in Table 9.3, to reach a teraFLOPS scale would require a 1024 core chip running at 2.5 GHz with each core (floating point units inclusive) computing collision (105 FLOP) and data transfer (propagation) in 268 clock ticks. And to reach a target of 5-10 teraFLOPS per chip, the only option is to increase the number of cores per chips by an equal portion. Another question would be, how to use 1024 core chip for lattice size larger than the available cores? Considering again the previous situation of direct mapping of 32^2 lattice to 1024 core processor, let us name 32^2 as the base grid. Now if we have a lattice that is l times base grid and imagine the lattice layout in a zigzag form as shown in Figure 9.1, a direct mapping is again possible using 1024 core processor. As long as the cores have sufficient memory, each lattice layer is mapped onto cores with each core storing and processing l cells each belonging to the corresponding layer. As expected, cores start by computing the first layer of the zigzag, followed by the second and so on.

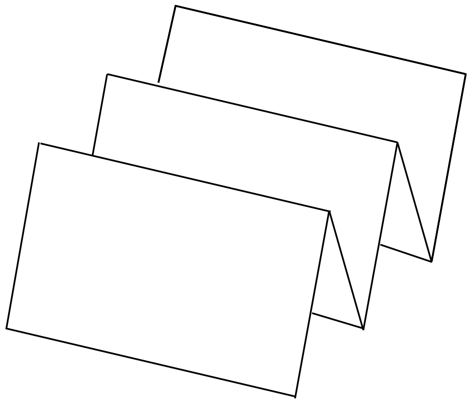

Figure 9.1: *Zigzag shaped: Two dimensional rectangular lattice.*

Another advantage with this layout is the ease of implementing wrap-around boundary conditions as well.

The same strategy can be applied to three-dimensional CAs as well, with lattice visualised as a stack of two-dimensional planes and each plane mapped to a manycore chip. Again the main requirement is the core capacity to hold the required number of cells and inter-cellular routing.

9.6 Manycores of Future

Parallelism, uniform structure, processor hungry supercomputing application, reconfigurable processing elements and interconnects, and scalability are some of the common elements that stand out when drawing parallels between the manycore review and our FPGA based CA implementations work. Based on these findings let us summarise the various types of manycore chip systems especially in terms of their processing units, memory and interconnects, that are to be expected in the future.

Starting with the processing units or cores, one would like to find out if these are going to be of same size or heterogeneous, what would be their optimal size and design, and so on. [5] believes that the building blocks of the future microprocessors are likely to be, “simple, modestly pipelined, floating point units, vectors, and SIMD processing elements.” And as discussed in previous sections, both size and design parameters of processing elements would be determined by the application under consideration. Identical or heterogeneous processing elements both come with their respective pros and cons. Simple identical elements offer ease of design and implementation in hardware while different sized processing units improve spatial chip efficiency and the parallel speedup by reducing the runtime for the less parallel codes [5]. Additionally, a thousand core chip both in terms of hardware and software in itself is a complex structure to deal with, and including heterogeneous cores would further make things difficult. One approach to include heterogeneity could be to have a chip composed of simple and uniform cores, and then have some of the cores, in aggregate, perform a specific task as and when required. Reconfigurability within the chip can be considered to address the on-demand aggregation of cores to perform specific functions. However, including reconfigurable units also elevate the possibilities of using the available logic space for a variety of different application domains, for example, our D2Q9 LBM PE, with little modifications, can be reconfigured to compute D3Q19 LBM cells.

The memory wall has been the major challenge ever since and [5] report this as the main obstacle for almost half of their identified dwarfs. Manycore designs mean much higher number of instruction executions (or MIPS) per chip and therefore, more memory bandwidth requirements. The current trends with memory capacity and speed suggest the memory wall getting worse [5]. And considering this MIPS explosion, situation demands innovations beyond traditional von Neumann architecture where the main memory is assumed to be a separate entity connected via standard interfaces. Keeping in view our experience with CA implementation, one of the main limitations of our D2Q9 LBM computation model, as presented in previous chapters, was also due to memory layout. Having a one giant memory block as an input source bank to multiple PEs processing cells limited the data transfer rates and also imposed the limit on the number of PEs within the design. For a thousand core chip this design is not scalable and therefore a model based on distributed memory within the chip is required. Hence, some important class of computations like stencil based operations has a very regular and entirely local memory access. Such applications can benefit from innovations in memory designs and lead the way towards novel designs.

Finally, the interconnects, [70] point out, a random communication pattern on a very large number of cores leads to enormous bisection bandwidth yielding complexity, latency and reliability issues, and simultaneously applications that scale to 10^6 cores with structured communications. Again, it is the application requirements that ask for innovations with how to interconnect hundreds of available cores. Currently, multicore systems employ buses or crossbar switches between the cores and the cache banks and such solutions are not scalable to systems with thousands of cores [5, 113]. Therefore, manycore systems require on-chip connectivity that scales linearly with the system size and at the same time prevents its complexity from dominating the overall costs. Ability to configure the wiring topology between the cores that meets the application communication requirements is an essential requirement in order to have a scalable architecture. Hence, one of the straight forward approaches is to use the flexibility of reconfigurable computing that allows the configuration of the on-chip network topology.

From the above discussion and the various listed desirable features of the machine in future, one thing that is common to most of the elements is the flexibility, not only in terms of software but hardware as well. Whether it's the processor internal structure, memory layout and the access pattern or the interconnect network, all demand to be quite flexible and customisable as per the application requirements. Without doubt, the ability to configure hardware is going to be an integral part of the manycore systems but in what proportion, that would only evolve with time. The current status with reconfigurable devices or FPGA technology in particular, is making rapid progress [104] and doubling in capacity every 18 months. FPGAs, have the capacity of millions of gates and memory bits, and also provide the ease and the speed of reconfiguration like modifying software. They are also narrowing down the concept of having a processor and the memory as separate entities. FPGAs have recently demonstrated their strength in HPC applications. FPGAs when compared to an equivalent ASIC processor in terms of flexibility, large scale production volumes, and low cost development both in terms of time and cost, suggests their use not only as prototyping or system emulation but also as flexible processors within the manycore system. No doubt that we are very soon to enter the manycore era, however, formulating a power and performance efficient computation is more or less an open question. [5] conclude their report with, “we do not claim to have resolved these questions. Rather our point is that resolution of these questions is certain to require significant research and experimentation”. Plus questions ranging right from the micro to the macro level of manycore systems are around, for example, comparing processor to a transistor like a basic building block or like a NAND gate in a standard-cell library [5]. Question about the evolution of manycore processing or the dominant computer architecture of the future in general [120].

As the hunt for answers continues, it would be interesting to investigate a chip with hundreds of simple and uniform cores, each core flexible enough to perform a function either on its own or as a part of a larger unit that is aggregation of cores. This would require cores to be configurable to function as a processing unit, or a memory unit, or as an interconnect conduit.