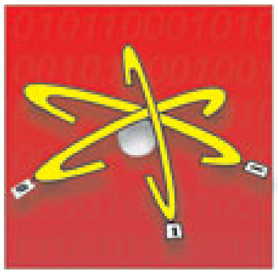
Crossroads
The ACM Student Magazine



Simulation of a Computer Architecture for Quantum Chromodynamics Calculations

by ***Sadaf Alam***, ***Roland Ibbett***, and ***Frederic Mallet***

Introduction

The study of Quantum Chromodynamics (QCD), a branch of Particle Physics, is considered a Grand Challenge application [2]; a **Grand Challenge** is a fundamental problem in science and engineering, with broad applications, whose solution would be enabled by the application of high performance computing resources [1]. Commercial and custom-built high-performance parallel computers have been used around the world for QCD research for the past 20-30 years. Custom-built massively-parallel QCD machines have a better cost/performance ratio compared with commercial high performance computers and are therefore more attractive to academic researchers . The success of special-purpose parallel computers has paved the way for research and development in many computer science disciplines including parallel architectures, processor design and high-performance compiler design.

This article outlines design steps of a recent state-of-the-art QCD computer called QCDOC (QCD On-a-Chip) [3] and presents an overview of its simulation model in HASE [9]. **HASE** is a Hierarchical computer Architecture design and Simulation Environment that allows rapid prototyping of both uniprocessor and multiprocessor architectures. Furthermore, the HASE visualization mechanism permits the graphical display of an architectural model to be animated, thus enabling the model designer to validate the model operations. The primary aim of this research is to explore the hardware and software factors that can influence the performance of the QCD computer.

Background

Quantum Chromodynamics

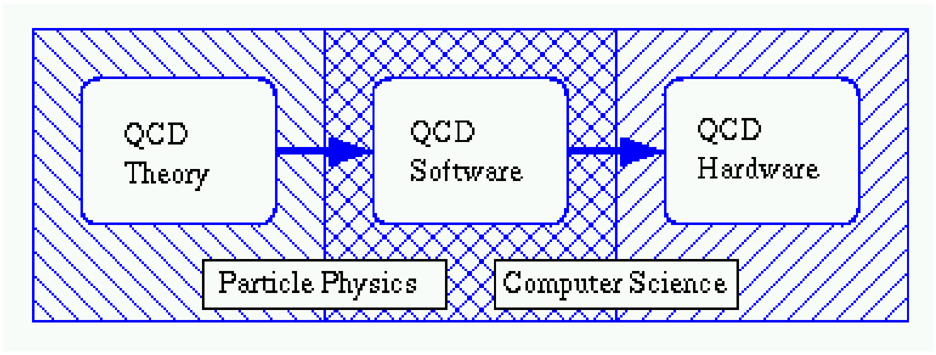


Figure 1: QCD Computation Process.

Particle physicists believe that when the universe began it was composed of particles, and that it is still a large particle system. Consequently, particle physics researchers attempt to discover what the tiniest particles are and how they interact. QCD (Quantum means discrete amount, Chromo for color) is the theory of quarks and gluons and their strong interactions. Quarks and gluons are the widely believed to be the fundamental particles of an atom. A proton, found in the nucleus of an atom, is composed of three quarks and is a hadron. Altogether there are six different types of quarks. Gluons are the carrier particles of strong interactions. There are many QCD theories that describe these quark masses and their interactions[10]. These theories are normally presented in a series of equations that cannot be solved analytically. Consequently, to validate these theories and compare their predictions with experimental results, computers are employed. **Figure 1** shows the QCD computation process on a parallel machine where a QCD theory, formulated as a series of equations, first gets transformed into parallel software and is then run on parallel hardware.

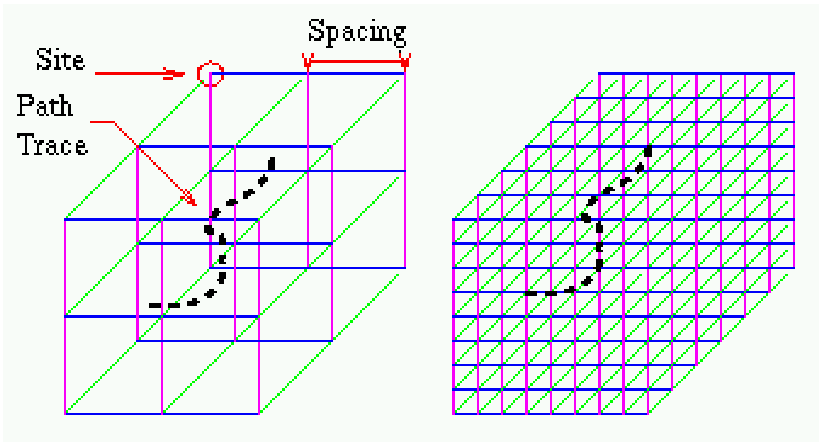


Figure 2: Example of a Coarse Lattice (left) and a Fine Lattice (right).

QCD calculations are commonly performed as lattice QCD simulations [6]; the space-time calculations are discretized and mapped on to a four dimensional space-time grid or lattice. Ideally, discretization error is minimized. For example, discretization errors can be minimized by reducing the distance between lattice points (called lattice spacing); a lattice spacing close to the continuum limit is ideal. **Figure 2** shows a path integral in two three-dimensional lattices; one with a large spacing containing a small number of lattice points (called lattice sites) per volume, and the second, a *fine* lattice with small spacing and a considerably large number of lattice points. Finer lattices tend to approximate the path integral with minimum discretization errors. The size of a QCD calculation depends on the number of lattice points. Typically, a lattice size of N^4 may involve solving a multiple of N^4-dimensional integral where N is, for most calculations, greater than 16. In practice, these integrals are solved using Monte Carlo methods on a parallel computer.

On a parallel computer, a lattice is evenly divided among the number of processing nodes available in each dimension. For instance, if lattice dimensions are N_x, N_y, N_z and N_t and the dimensions of the machine are n_x, n_y, n_z and n_t, then the number of lattice points mapped on to a node will be $N_x/n_x * N_y/n_y * N_z/n_z * N_t/n_t$.

Multiprocessor Simulation Models

A variety of techniques can be used to explore design tradeoffs in a computer architecture and to evaluate its performance. Evaluating the performance of an existing system is, in principle, straightforward as benchmarks can be run to measure the execution time. However, without extensive instrumentation (possibly involving both hardware and software), this provides little insight into the causes of performance limitations. Moreover, it offers very little opportunity to measure the effects of varying architectural design parameters. One alternative is to use analytical modeling. This has been done effectively for a variety of multiprocessor system components but the models are usually driven by workload models rather than benchmarks or real applications and the results can be unreliable, particularly if attempts are made to model complex systems containing a variety of components. Another approach is to build a prototype of the machine with relatively few processors. In practice, a 2-4 node prototype is constructed for a target machine of well over a 1000 processing nodes and hence is not an effective way to study the overall behavior of large parallel systems. Furthermore, this stage cannot be started until most of the design considerations are made final and therefore the designers are unlikely to explore a wider design space. Simulation has therefore become a popular approach to analyze dynamic behavior and to predict performance of complex systems.

Correctness and level of detail are the key considerations for a simulation model designer. In addition, the model should be flexible enough to explore and to investigate the design space of the target system. Multiprocessor simulation models are challenging to model because one must model not only their memory requirements, dimensions, and interconnection network topology, but also the interactions between the parallel software and hardware. Scalability of a system is one of the most sought-after characteristics of a parallel system. Experiments to measure the scalability of a parallel system can only be performed by providing a flexible way to increase the size of machines. Altering dimensions and size of a parallel machine is not straightforward, as an increase in the size of the simulation system not only multiplies its hardware requirements but also affects the interaction between parallel software and hardware. The representation of the parallel workload can be used to categorize computer system simulations [5,7]. Four categories of computer system simulations are briefly described as follows:

- **Distribution-driven simulation** - The workload is modeled stochastically by the distribution of memory references. These models are extremely efficient but limited to overall performance measures and unable to provide insight into the system. Also, it is difficult to identify stochastic models representative of the dynamic behavior of a parallel machine.
- **Trace-driven simulation** - Central to this mode is a trace file. A trace of memory references is generated once by executing the workload on a machine similar to the target system or by using a functional simulator. Results from simulations can be very effective and accurate as long as the properties of modeled machine and the machine from which the traces are obtained remain similar. In other words, these types of simulation are not particularly suitable to explore wide design spaces.
- **Program-driven simulation** - This method is generally considered as a complete simulation scheme because both the processors and the memory system of the target system are simulated. The workload is executed on simulated processors where each machine instruction of the target processor can take many machine cycles of the host computer. This is an extremely accurate technique but is not practical for large multiprocessor simulators due to speed and memory constraints.
- **Execution-driven simulation** - This scheme was proposed to address the inefficiencies of the program-driven simulations. The workload in this mode is executed on a host computer rather than on the simulated processor. At events of interest, for example, shared memory references, control is transferred to the simulation software, which simulates the memory system. Although this scheme is fast, it provides little insight into microprocessor-level parameters like Instruction Level Parallelism (ILP).

Which simulation technique to use depends on the hardware features of interest. For instance, for shared-memory multiprocessor coherency protocols, memory references are the critical values and what happens in the execution pipes is not as significant. On the other hand, when studying execution pipeline utilization, each pipeline stage has to be modeled accurately. For the former example, an execution-driven simulation is a suitable candidate, for the latter, a program-driven simulation model is necessary.

HASE

The Hierarchical computer Architecture design and Simulation Environment (HASE) developed at the University of Edinburgh allows rapid development and exploration of computer architectures at multiple levels of abstraction, encompassing both hardware and software. The main features of the HASE environment include an Entity Description Language (EDL), an Entity Layout (EL) file, a discrete-event simulation engine HASE++, a visualization mechanism, and results gathering tools [**4**].

The EDL definitions of the architectural components provide information about architecture parameter definitions, component parameters and ports, hierarchical on-screen structure, global system parameters, and component interconnect. Predefined multiprocessor topology templates are also included in EDL to aid in the rapid development and exploration of multiprocessor networks. The EDL description when combined with the Entity Layout file containing display information completely describes the architecture to be simulated.

Once the architecture is loaded into HASE from an EDL file, the simulation executable can be generated by combining the architecture information and user defined parameters with the individual component behavioral descriptions. The behavior of entities is described in HASE++, a C++-based discrete event simulation language. This simulation executable can then be executed with various input parameters specified in the architecture descriptions.

The visualization mechanism allows the graphical display of an architectural model to be animated by reading in trace files generated by the simulation execution, thus enabling the designer to inspect the model for correct operation. The hierarchical nature of HASE controls the displayed complexity of the simulation, according to the areas of the model being concentrated on.

This paper highlights only those features of HASE that are being included and extended for the simulation of the QCDOC model. But first, we give a brief description of a parallel QCD software and the key design features of the QCDOC architecture.

QCD Software and Hardware

QCD Programming Model

QCD application programs follow a Single Program Multiple Data (**SPMD**) style programming paradigm and are therefore suitable for SIMD (Single Instruction Multiple Data) and MIMD (Multiple Instruction Multiple Data) parallel machines. In an SPMD-style programming model each processing node effectively performs the same set of calculations on different sets of data elements. According to the Caltech Concurrent Computation Group classification of parallel applications [**8**], QCD simulations are considered as **Synchronous** applications. Synchronous applications have a regular structure, and in general, are the simplest to code and parallelize. These applications are characterized by a basic algorithm that consists of a set of operations, which are applied identically at every point in a data set. The structure of the problem is typically very clear in such applications, and they can be considered as parallel in nature.

Like any other parallel application, the total execution time of a QCD application depends on the amount of time spent in computation communication. Communication in QCD calculations can be further divided into nearest-neighbor communications and global sum routines. This is because nearly all communications in lattice QCD calculations involve access to nearest-neighbor points apart from a few floating-point global sum operations. These global sums have to be performed in the same order on each and every node so that all nodes have the same floating-point value (no rounding errors). Due to the extremely deterministic nature of the calculation, nearest-neighbor communications can be overlapped with computation. It is possible to send requests for neighboring data in advance before that data is needed in a calculation. On the other hand, the infrequent global sums operations cannot be overlapped.

The Conjugate Gradient algorithm is one of the most well known and the most frequently used iterative algorithms for solving the large sparse linear system of Lattice QCD [12]. The execution times for the algorithm can be expressed as:

$$T_{exe} = scale\_factor * (T_{comp} + T_{nncomm}) + T_{glbsum}$$

Where, T_{exe} is the total execution time, T_{comp} is the computation time, T_{nncomm} is time taken for the nearest-neighbor communications, and T_{glbsum} is the time required for the floating point global sums. `scale_factor` is used to represent the scaling of the application execution time with respect to the number of sites mapped on a node. As long as the dimensions of a parallel machine remain the same, the time required for the global sum remains the same, as the scaling factor does not apply on T_{glbsum}.

Ideally, hardware for a QCD software must have fast processing nodes with high-performance floating-point execution units and support for low latency high-bandwidth four-dimensional nearest-neighbor communications and global sums. Most mainstream commercial high-performance computers are not optimized for the QCD software requirements and their costs are out of reach for many research groups. Therefore, Particle Physics researchers have decided to build cost-effective custom-designed computers from commodity components. This phenomenon has been very successful. The QCDOC architecture is a recent example which is currently under construction in a collaboration between a number of universities and IBM. Its design is highly optimized for QCD calculations.

QCDOC Architecture

QCDOC is a 10 Teraflop-scale massively-parallel distributed-memory MIMD architecture with over 8000 processing nodes. The main components of the architecture are the processing nodes and the communication network[14].

The Processing Node

A QCDOC node consists of a custom-built Application Specific Integrated Circuit (ASIC) and an externally connected memory called Dual-In Module Memory (DIMM) Synchronous Dynamic Random Access Memory (SDRAM). It is called QCDOC (QCD On-a-Chip) because a node is effectively an ASIC plus an external memory module.

The ASIC is a complex piece of hardware built with mainly IBM library components. It contains a high-performance embedded PowerPC 440 core with co-processor Floating Point Unit (FPU), on-chip memory called Embedded DRAM (EDRAM), a high-performance bus, a Direct Memory Access (DMA) controller for on-chip and external memory transfers, a Serial Communication Unit (SCU) to support QCD specific communication routines, and additional IBM components to support the ASIC design. The custom-designed blocks are Prefetch EDRAM Controller (PEC) to support high-bandwidth and low-latency core and on-chip memory transfers and the SCU for four-dimensional nearest-neighbor and global communications.

The IBM PowerPC 440 [15] core contains a dual-issue seven-stage out-of order issue, execution, and completion pipeline. It has separate highly-associative (a 32K cache is 64-way set associative) instruction and data caches. A Translation Lookaside Buffer (TLB) supports multiple-sized pages and is responsible for other access and user-defined features. The cache mechanism has a powerful feature of being further sub-divided into locked, transient, and normal regions for optimization. In the PEC block, there is a Level 2 (L2) prefetch cache. The L2 cache maintains a set of prefetched data and buffer writes from the processor, and is also capable of delivering read data at the processor clock frequency.

Communication Network

There are two networks in the system. One is dedicated to the QCD calculations while the other is for boot, general input/output, and other system support features. The QCD network is a six-dimensional torus for inter-node communications, while a tree of standard Ethernet connections is used for system support. Out of the six dimensions, two are for software partitioning of the system and four are dedicated for the four-dimensional space-time QCD calculations.

The architecture of the QCDOC machine is indeed highly optimized for the QCD calculations. Now the question is, "What can be learned by modeling the QCDOC machine, which promises to be able to solve a range of previously unsolved or impossible-to-solve QCD calculations?" The answer follows in the next section.

HASE QCDOC Model

As the performance of a massively-parallel machines is rocketing to Teraflop and Petaflop ranges, the gap between the theoretical peak performance and application or user code performance is getting wider. The reasons frequently cited [13] are memory bandwidth, processor-to-memory speed ratio, and network latencies. The main motivation behind modeling the QCDOC system is to explore the influence of these factors on the ratio of the application code performance to the theoretical peak performance. Typically, without optimized kernels, the performance of application code ranges between 15% to 30% of the theoretical peak performance [16]. Because QCDOC is a custom-built parallel machine, it is believed that a higher than normal fraction of peak performance is possible for the application code. The custom components that can play decisive roles are the levels and bandwidths available for cache and memories, the bidirectional four-dimensional dedicated network, and the PowerPC core configuration. The HASE QCDOC model is designed to explore the design space of this machine.

Model Design

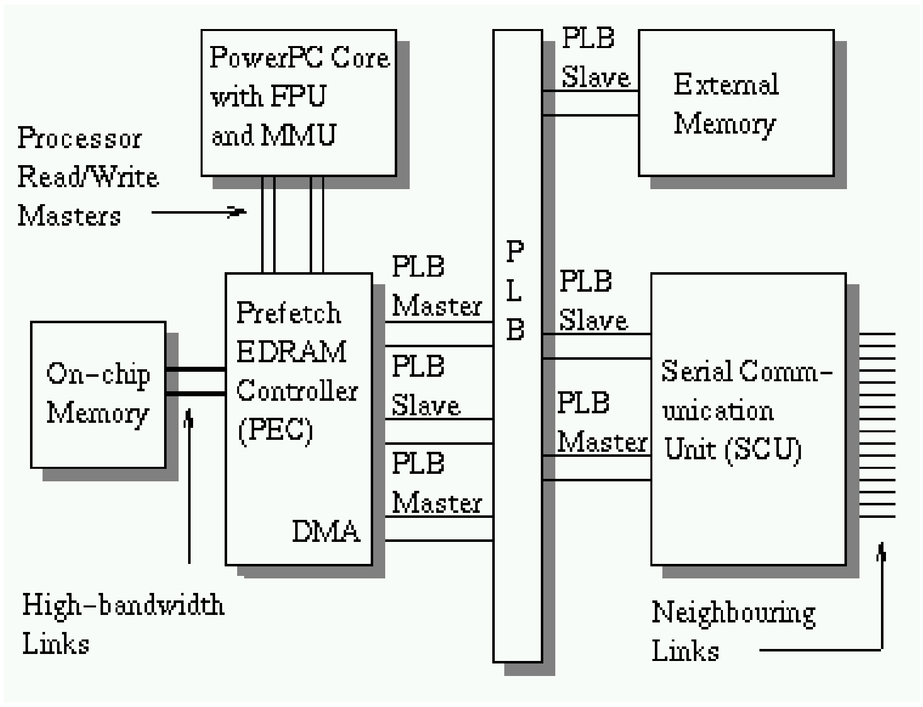


Figure 3: A HASE QCDOC Node.

In designing the simulation model, the key design issue was to decide what level of detail was necessary to model. Since this research aims to investigate the factors limiting the performance of the QCD application code on the QCDOC computer, it was decided to model only those components of the processing node that take part in the execution of the application code. **Figure 3** shows the design blocks of a HASE QCDOC node. Attention has been focused on detailed modeling of the custom components because the QCD code is written in higher level languages like C/C++, where compiler-generated output is unlikely to utilize system resources effectively. A snapshot of a 2x2x2x2 QCDOC computer model is shown in a HASE design window in **Figure 4**. The figure shows design entities, their parameter control block and multi-frequency clocks. The figure also illustrates the hierarchical levels of a QCDOC node. The contents of a memory module can also be viewed in the design mode and during the animation of a simulation.

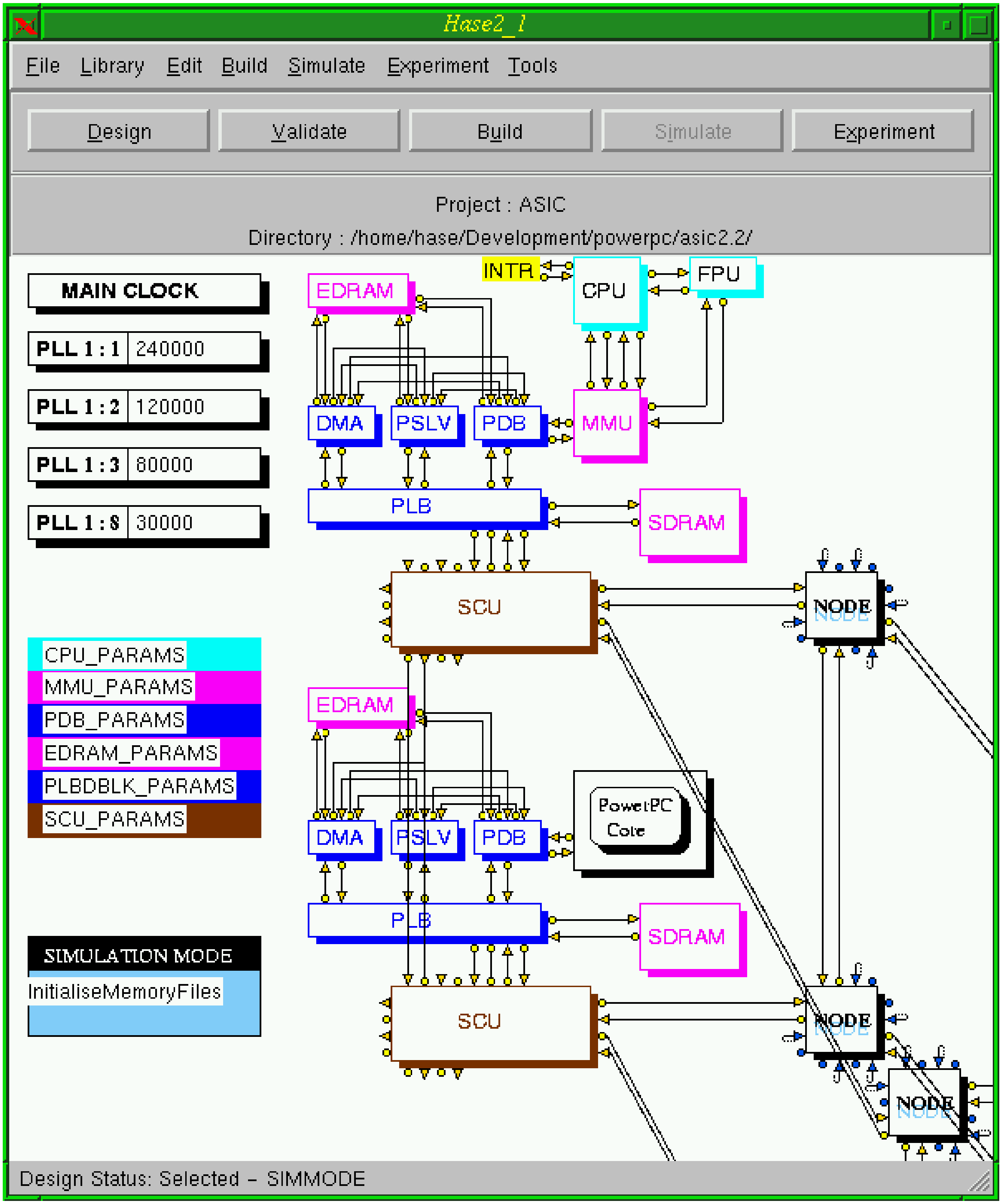
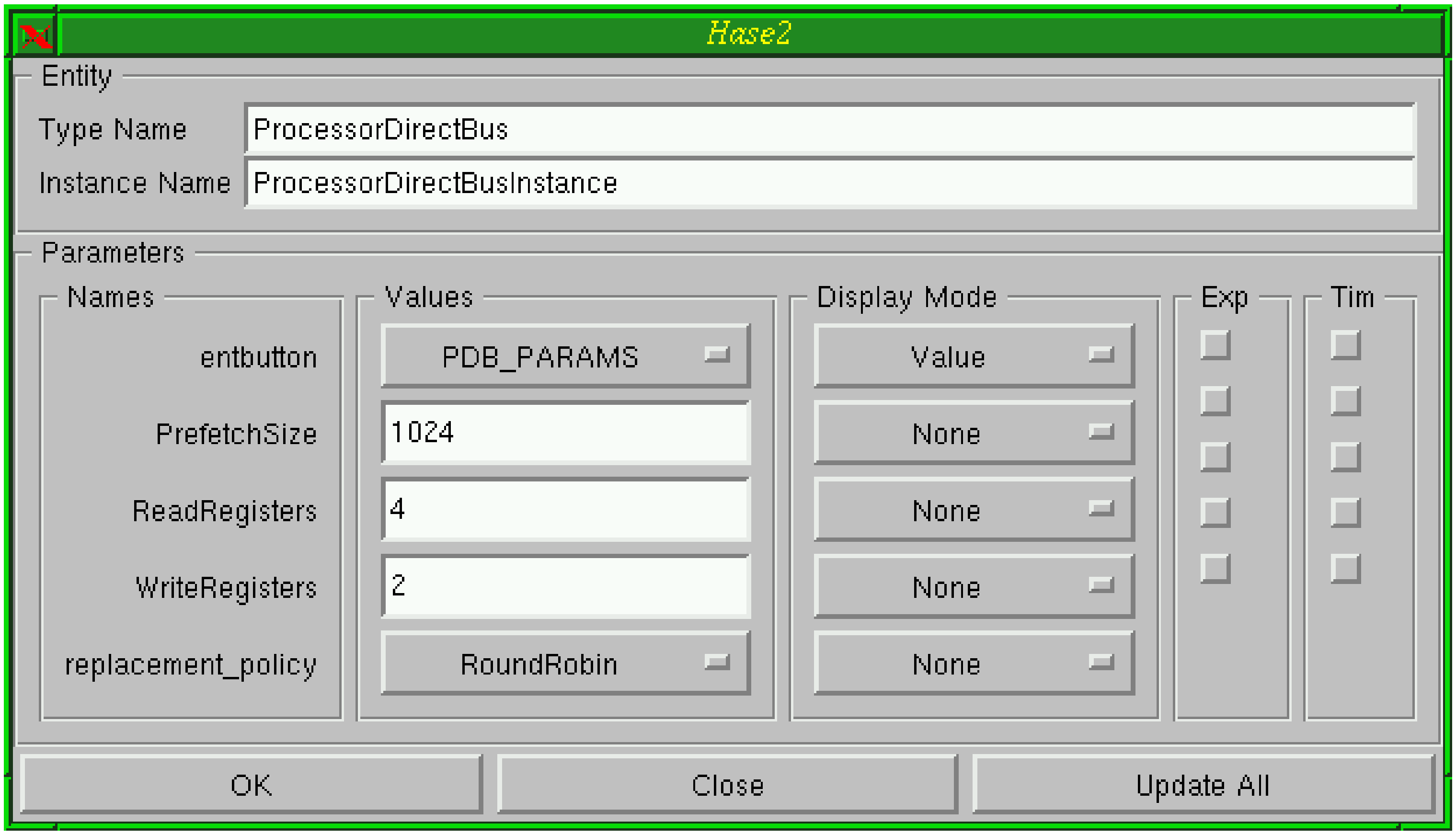


Figure 4: HASE QCDOC Model.

The second key issue was figuring out how to model many large configurations. The aggregate memory requirement of the four-dimensional QCDOC torus restricts the modeling of this machine to few tens of nodes. In order to be able to perform experiments with a considerably large number of nodes, it was decided to construct more than one model for the QCDOC computer. The first model (**Figure 4**) is program-driven and is constructed to identify and explore the design space of a node. This model can be composed of no more than 32 processing nodes, mainly because each node simulates all the memory systems of a QCDOC node. The second model will be an abstract version of the first model with no memory contents, and it will have a relatively large number of processing elements. This model will be trace-driven. The traces for the abstract model will be obtained from the first model. The abstract model will be capable of simulating all the communication events including the frequent nearest-neighbor communications. All other events will have a time stamp associated with them, which will help to estimate the overall execution time of a QCD test routine. The number and timings of the nearest-neighbor interactions will be the same as long as the number of sites mapped per node remain the same. However, the global sum timings, which depend on the size of the machine and its interconnection network topology, have to be adjusted. These adjustments are part of the future research plans.



Type Name	ProcessorDirectBus
Instance Name	ProcessorDirectBusInstance

Names	Values	Display Mode	Exp	Tim
entbutton	PDB_PARAMS	Value	☐	☐
PrefetchSize	1024	None	☐	☐
ReadRegisters	4	None	☐	☐
WriteRegisters	2	None	☐	☐
replacement_policy	RoundRobin	None		

Figure 5: A Parameter Window.

Finally, experiments are performed for performance evaluation of the model. The test routines for the model will be the code segments (in C++ and PowerPC assembly), which are optimized for the QCDOC machine. Experiments will be performed by altering parameters of HASE QCDOC model's entities to observe the dynamic behavior of computer architectures with a number of hardware and software combinations. **Figure 5** shows a parameter window for an entity

Processor Direct Bus (PDB) that contains the Level 2 cache. When a parameter is altered in the parameter window, the next simulation runs with new parameter values. For example, PDB allows changing the size of prefetch register (in figure 1024 bits), number of prefetch read registers, number of write buffer registers, and the replacement policy adopted for the selection of a victim read register. Using this facility, experiments will be performed; for example, with varying cache configurations, bandwidth and latency of network and application test routines.

Simulation Results

The result of a simulation is a trace file that can be animated to observe the dynamic behavior of the simulation entities. In addition to the animation facilities, HASE provides a Timing Diagram generation method that can be used to study the send and receive events among entities over time. As the QCDOC machine simulations will be quite large, the overall effect of various parameters cannot possibly be studied via animation of the simulation or from timing diagrams. As part of this research, a number of graph and chart generation facilities will be included to study, investigate, and represent the simulation results.

The performance of the model is evaluated by measuring the number of processor clock cycles spent. The number of clock cycles are well-known as logical time or simulation time. The time taken by a scalar host processor to simulate the model depends on the size and dimensions of the machine. The bigger the dimensions of a machine are, the more processing nodes and design entities it has, and therefore, the longer a host processor will take to generate and animate its simulation trace file. However, increasing dimensions of a machine will not effect the simulation time of individual processing nodes. For example, on a workstation, a 2x1x1x1 model or 2 processing nodes take 3.869 seconds to run 50 processor clock cycles, while a 2x2x2x2 system or 16 nodes take 13.855 seconds to run the same number of clock cycles.

HASE Extensions

HASE has evolved significantly as a result of the requirements of the QCDOC simulation model. Extensions were necessary to model the multiple dimensional meshes with a large number of nodes. The key extensions are the library clock mechanism and memory model options. The HASE mesh mechanism has also been extended from three-dimensional meshes to n-dimensional meshes with an option of with or without wrap-around links.

The Library Clock Mechanism

Like any complex ASIC, the QCDOC ASIC operates on a range of clock frequencies. The fastest block is the PowerPC core and all other components run at a speed relative to the processor's clock frequency. For example, the Processor Local Bus (PLB) operates at one-third of the processor clock and its frequency is represented as (1:3) of main clock. In total there are four different frequencies, in which the slowest is the speed of the communication link, which is 1:8 (one-eighth of the processor clock).

In order to model the cycle-accurate behavior of the QCDOC node, a flexible and extensible clock mechanism has been introduced in HASE [11]. It is implemented using an object-oriented inheritance mechanism, such that each entity that needs to be clocked should register with an ABSTRACT Clock entity. The registering mechanism is made transparent to the user by inheriting the synchronization behavior from a predefined ABSTRACT clocked entity. Inheritance from a clocked entity simplifies the behavior specification. Such an entity is only required to override the $pre, $phase0, and $phase1 sections of the behavior code to specify what should happen in the initialization phase ($pre) and during each synchronization phase ($phase0 and $phase1). For the model designer, the Clock entity is a library component, hence there is no need to rewrite its code. This Clock entity provides only one clock frequency, the range of frequencies are provided by another component called the PLL (phase-locked loop) entity. A PLL operates on a ratio relative to the main clock. For the QCDOC model, four PLLs are required for the four different clock speeds (1:1, 1:2, 1:3, and 1:8).

Memory Model Options

QCDOC is a distributed memory machine, each node has its own sets of memories. Conventionally in HASE, each memory module requires a text file with .mem extension that contains an array of an appropriate data structure that represents memory contents. For the QCDOC model, this scheme was not practical because of a large number of memory modules required per node. A QCDOC node's memory modules include an instruction cache, a data cache, a set of prefetch read registers, write buffer registers, on-chip memory, external memory, and send and receive registers. Because the QCDOC model is a four dimensional mesh, a slight increase in size of one dimension can increase the total number of nodes significantly, which will result in a huge number of memory text files being required.

In order to address this problem, two options are introduced for the HASE memory ARRAYs. Each of these options is transparent to the simulation user. The first option is well-suited to the SPMD programming model where all nodes share a single instruction file. The model designer can use a single physical text file to represent the contents of the instruction cache that will be shared among all nodes. However, in the model design window and during the animation, the nodes will appear to have their own copy of instruction cache. The second mechanism was introduced for the data memories. This option is quite similar to the first method in that there will be a single physical file. Unlike the instruction caches however, the contents of data memories need to be different. Therefore, the single .mem file in this case contains sections, where each section belongs to a separate memory instance. These sections are separated by a tag to identify the beginning and end of each section. Section mappings are restricted to follow the order in which memory instances are generated by HASE.

The introduction of the library clock mechanism and memory array options have made the project management of this and future massively-parallel system possible without compromising any simulation, animation, and performance analysis features of HASE.

Future Directions

This research is still in progress. The goals include providing a robust mechanism to visualize the dynamic behavior of the QCDOC computer design blocks and evaluating performance, which will be conducted by running the QCD code segments on the parameterized HASE QCDOC model. It is hoped that by varying architectural parameters, an optimal machine configuration can be explored. Furthermore, the HASE platform has become more sophisticated and suitable for rapid prototyping of complex multiprocessor systems. It is anticipated that HASE can provide an ideal environment for research and exploration of future computer architectures, which are expected to be proposed for recent advancements in theoretical Particle Physics.

Acknowledgements

The QCDOC computer simulation project is supported by the UK EPSRC under grant GR/R27129. I wish to thank the following individuals from Physics and Astronomy department, University of Edinburgh, for their help and support with this project. Professor Anthony Kennedy explained, for hours, the background of QCD calculations and the QCD programming model. Dr Peter Boyle and Dr Balint Joo, who are in QCDOC design team, provided the technical information about the QCDOC machine and shared their invaluable programming experience on massively-parallel QCD computers.

Bibliography

1

Bailey, F. R. and Simon, H. D. *Future Directions in Computing and CFD.* Proceedings, AIAA 10th Applied Aerodynamics Conference, 1992.

2

Bowler, K. C. and Hey, A. J. G. *Parallel Computing and Quantum Chromodynamics.* Parallel Computing 25 (1999) 2111-2134.

3

Chen, D. *et. al, QCDOC: A 10-teraflops scale computer for lattice QCD.* Nuclear Physics Proceedings Supplement, 94 (2001) 825-832.

4

Coe, P. S. *et. al, HASE: An Environment for Hardware/Software Codesign.* Technical Report CSG-41-98, University of Edinburgh, 1998.

5

Covington, R. G. *et.al. The Efficient Simulation of Parallel Computer Systems.* International Journal in Computer Simulation, Vol. 1, 1991.

6

Creutz, M. *Quarks, Lattices and Gluons.* Cambridge University Press, 1983.

7

Ferrari, D. *Computer Systems Performance Evaluation.* Prentice Hall, New Jersey, 1978.

8

Fox, G. C., Williams, R. D. and Messina, P. C. *Parallel Computing Works .* Morgan Kaufmann, 1994.

9

http://www.dcs.ed.ac.uk/home/hase

10

Gupta, R. *General Physics Motivations for Numerical Simulations of Quantum Field Theory.* Parallel Computing 25 (1999) 1199-1215.

11

Ibbett, R. N., Mallet, F. and Alam, S. R. *An Extensible Clock Mechanism for Computer Architecture Simulations.* 13th IASTED International Conference Modeling and Simulation, 2002.

12

Isgur N. and Negele, J. W. *Nuclear Theory with Lattice QCD.* A proposal submitted to the U.S. Department of Energy, unpublished(2000).

13

Lubeck, O. M., Simmons. M. L. and Wasserman, H. J. *The Performance Realities of massively-parallel Processors: A Case Study.* Proceedings of the 1992 conference on Supercomputing, 1992.

14

http://www.lqcd.org/hardware_qcdoc.htm

15

The PowerPC 440 Core, IBM Microelectronic Division, NC, 1999.

16

Sroczynski, Z. *Improved Performance of QCD code on ALiCE.* Contribution to Lattice2002 (machines).

Biography

Sadaf Alam (**SR.Alam-2@sms.ed.ac.uk**) is a PhD student at the University of Edinburgh, UK. She is a member of Institute for Computing Systems Architecture (ICSA), School of Informatics.

Professor Roland N. Ibbett (**rni@inf.ed.ac.uk**) is the principal investigator of the HASE QCDOC simulation model project.

Dr. Frederic Mallet (**fmallet@inf.ed.ac.uk**) is a post-doc at ICSA and a key contributor to the HASE developments.