The Columbia .8Tflop Project

Igor Arsenin [a], Dong Chen [a], Ping Chen [a], Norman H. Christ [a], George Fleming [a], Chulwoo Jung [a], Adrian Kaehler [a], Steve Kasow [a], Roy Luo [a], Catalin Malureanu [a], Robert D Mawhinney [a], ChenZhong Sui [a], Pavlos Vranas [a], Alan Gara [b], John Parsons [b], Robert Edwards [c], Tony Kennedy [c], Jim Sexton [d], Sten Hanson [e], Greg Kilcup [f]

[a] *Columbia University*

[b] *Columbia University (Nevis Laboratories)*

[c] *SCRI at Florida State University*

[d] *Trinity College, Dublin*

[e] *Fermi National Accelerator Laboratory*

[f] *Ohio State University*

The Columbia .8 Tflop project has designed and is nearing completion of the assembly of a custom MIMD machine. This machine is optimized for lattice problems such as QCD (Quantum Chromodynamics) which require an enormous amount of floating point power. The computer is based on 16384 processing nodes connected in a 4-dimensional torus. Each node is comprised of a DSP (digital signal processor), a custom gate array, and conventional DRAM. The nodes are extremely cost effective allowing for a large, scalable machine at a cost far less than anything commercially available. The simplicity of the hardware architecture enables a reasonably simple software environment in which to develop massively parallel applications. A commercial assembler and a C compiler for the DSP (Texas Instruments) as well as a C++ compiler all enhance the software environment. A detailed description of the architecture as well as the current status and measured performance will be given.

Key words: Parallel Processing; Lattice Gauge Theory; QCDSP;

Preprint submitted to Elsevier Preprint *29 July 1997*

1 Introduction

The Columbia .8Tflop project (QCDSP) was originated to build a custom computer that could reach a new level of computing performance for Lattice Quantum Chromodynamics (LQCD). This paper will discuss the development of this computer.

The bulk of the computing time in many LQCD calculations is contained in a relatively small number of algorithms which is typically optimized to obtain maximum performance from the available computing resource. We have chosen to exploit this feature by choosing a relatively simple processor (TMS320C31 Digital Signal Processor) which is extremely cost effective for floating point ($40 for 50Mflops) but is still flexible enough to be able to handle arbitrarily complex C-code at a reduced level of efficiency. The hardware design was to follow the constraints of the problem as closely as possible. This has led to a MIMD architecture where the nodes are arranged in a 4-d torus.

To keep the hardware and software design as simple as possible, we have also chosen a programming model that is extremely simple. When writing code for the entire machine, one simply writes code for a single node. If simple rules are followed concerning data transfers to/from neighbors then the overall size of the lattice (or the number of nodes) doesn't effect the code. In practice, one physical node is further broken into logical sites, which results in some complication as to which sites are on the node boundaries but this is done entirely at the single node level. We have also built into the hardware the ability to handle global sums very efficiently. Internode synchronization is only required at a very weak level for LQCD, and this is nicely taken care of by the internode communication protocol. Each node has an independent DMA controller for each of the 4 off-node directions, which allows all the communications to be offloaded from the DSP. While there are no restrictions in the hardware from using parallelizing compilers, we have not pursued this route as this simple route allows us to more quickly and easily achieve our physics goals within LQCD.

2 Design Method

In designing the QCDSP machine we had a number of choices to make:

- What is the ‘best’ processor (cost, performance, code efficiency),
- What are the necessary functions of the custom ASIC (cache,DRAM controller, off node communications),
- What does the motherboard look like? (how many nodes/motherboard?)

– How large should a node be? (cost, complication)

These questions are all very coupled, and it was at this stage that the importance of having a SMALL group committed to the design allowed quick convergence. The processor type we chose was the DSP largely due to it cost effective floating point power and its simplicity. The choice of which DSP floating point manufacturer was narrowed down to Texas Instruments and Motorola because of their strong performance and simple interfaces. We chose the Texas Instruments TMS320C31 because of its low cost even though it was somewhat weaker than the Motorola competitor in terms of DMA capability because this was a feature which we could compensate for in our ASIC.

The processor cost sets the scale for the single node cost which then, given a finite budget, sets the DRAM cost. There is of course the additional constraint that the DRAM be large enough to handle the physics requirements. The minimum memory required for the physics was set at 2Mbytes. This was initially achieved using five 512k by 8bit DRAM chips on each node (1 was for EDC). As we progressed we found the 256k by 16bit DRAM chips were becoming the industry standard, we therefore made the ASIC capable of handling these as well. We also allowed for larger memory on the first node of each motherboard, which was equipped with 1Meg by 16bit memory chips.

We wanted to put as many nodes as possible on a motherboard to keep the total number of off-board connections as low as possible. The solution which we chose was to use the SIMM technology used in commercially and have all of our nodes removable much like the DRAM one has in personal computers. This allows us to have a high node density on a single motherboard (64) without relying on a complex motherboard.

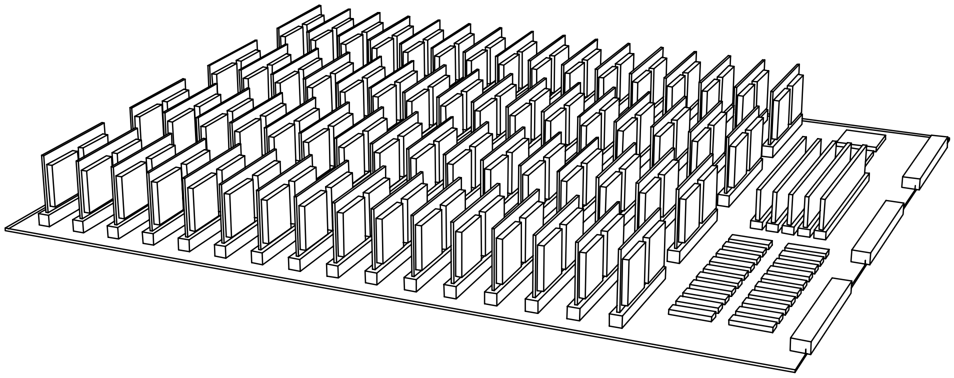

Fig. 1. Drawing of the 64 Node Motherboard for the QCDSP machine

We have adopted a motherboard with the nodes arranged as 4x4x2x2. All of the directions, with the exception of one of the x4 directions, are extended off board. These three off-board directions can be cabled to allow for arbitrary 3-dimensional node arrangements. The daughter boards are larger than SIMM modules due to the larger total chip area so we decided to put 8 motherboards in each crate. This wide slot spacing allows ample room for the 160 signal wires contained in 6 groups of connectors to be placed on the crate backplane for each slot. With this configuration, an 800Gflop machine is achieved with 8 racks of electronics (each 4 crates high).

The design of the custom ASIC was the most challenging part of this machine, and there were many choices which we had to make concerning its functionality. We decided to put all the necessary functions of the node were was not carried by the DSP and DRAM into this single ASIC. The necessary functions are:

- Interface to the DSP.
- Interface to the DRAM including refresh control.
- Provide EDC for all DRAM transfers.
- Provide a buffer that prefetches data to allow for SRAM like access times.
- Provide a serial unit for each of the directions which has it's own DMA controller(50Mbit/sec/link).
- Provide global interrupt ability that goes beyond the DSP's interrupt capability.

There were a few different buffering methods that were investigated. These were each investigated in terms of their physics performance. At this stage this was done by parameterizing the system efficiency in terms of DSP bandwidth, DRAM bandwidth and buffer efficiency. Below is the plot that describes the results.

3 Architecture Overview

The single node design was consistent with the goal of building something that was as simple as possible. This lead to the SIMM module idea previously mentioned, which has been extremely useful in both the initial debugging stage (only one node to debug) and in the full system (a "bad node" can be trivially swapped out).

The remaining system organization involves the interboard interconnects for the off-node serial lines and two SCSI interfaces/motherboard. The SCSI is used to allow access to a hard disk array and also to control the machine from an outside workstation. As each motherboard is equipped with two SCSI busses, the potential bandwidth to disk for the full system is extremely high. There is a single node on each motherboard, which resides directly on the motherboard and handles the system requirements of the board. This node has the following capabilities:

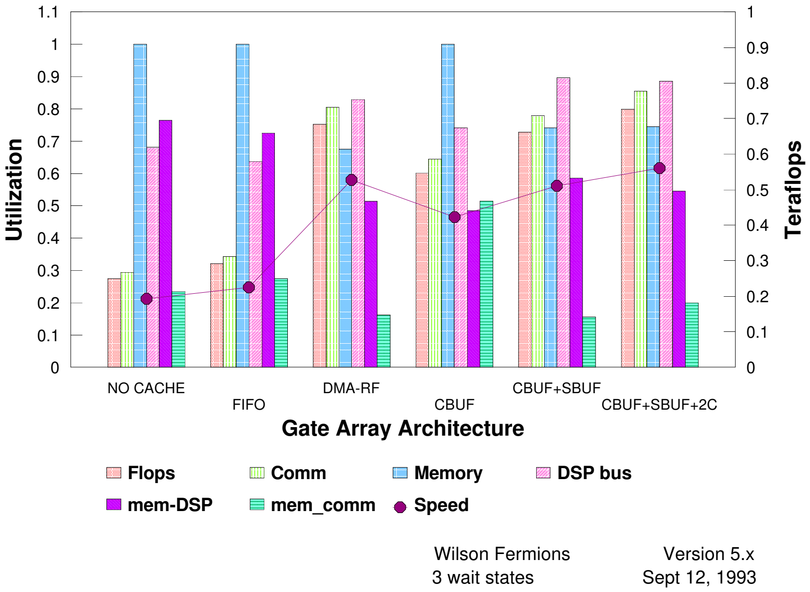

Fig. 2. Code efficiency for various buffer design configurations

- Communicate directly with the two SCSI bus interfaces.
- Configure a network of fan-outs/fan-ins that allow this DSP to talk directly to any other DSP on this board using the DSP's own serial ports. This is used for booting and controlling the other nodes.
- Read from EPROM configuration constants and also boot from this EPROM.
- Configure the physics serial off-board signal drivers to be clocked or unclocked. This is useful as the serial unit in the ASIC can be run at 25Mhz or 50Mhz.

The off-board serial links are required to transfer 160 bidirectional signals at 50Mhz. The internode serial links which are not off board are unidirectional. These unidirectional signals are converted into a bidirectional signal by using BTL transceivers on the interface. This allows us to reduce the total number of wires to 160 for off board and makes the connector arrangement feasible. We have chosen to use a high density cable to minimize the cable mass and the connector sizes. The SCSI interfaces on each motherboard are used to configure the motherboards as well as to access the disk. The SCSI busses on different motherboards are arranged in a tree manner which allows the single host

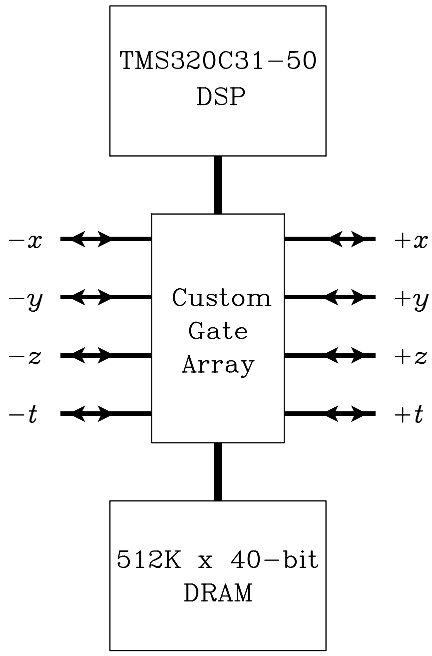

Fig. 3. Block Diagram of Single Node

workstation to access information or download control to any motherboard (and therefore any node) in the entire system.

4 Hardware Design

The specific hardware design requirements have been previously discussed. The design environment for the hardware design has also been crucial to the success of this project. We have chosen to design the QCDSP custom ASIC using software from Viewlogic. This software was sophisticated enough for us to allow thoughly testing of our design while giving us a simple interface in which to create the design.

The custom ASIC was designed with the unusual luxury of knowing exactly what code was to be run on it. We purchased from Logic Modeling (now purchased by Synopsis) a model for the DSP and the DRAM that allowed us to execute the real code (C or assembly) on our simulation model within the Viewlogic environment. This gave us a very powerful method to further check efficiencies of code and to find design bugs before fabricating the ASIC.

The design flow for the ASIC went smoothly. We provided the gate array manufacturer (Atmel) with the design as well as test vectors, which they then checked for consistency. They then laid out the design in silicon and gave us a back annotation file with all of the gate timings. Our suite of tests were then resimulated (about 1 week of cpu time on a Sun workstation) to verify that everything was OK. We then signed off and received prototypes shortly after. We have found no discrepancies between the simulation and the silicon at any level. This has meant that no iterations of the ASIC have been necessary.

The buffer design and the serial control module both contain elements that are novel and warrant further description. The buffer in the ASIC was to be small but very efficient for the code that we were to run. There are two block sizes that are commonly used in LQCD (3x3 complex and 3x1complex). The order in which we fetch these from DRAM is always sequential but they are used more than once in a nonsequential fashion. In addition to this, the TMS320C31 has a severe timing restriction on the data handshake that makes a zero-wait-state access impossible under normal conditions. We have been able to handle these restrictions by allowing different images of the DRAM to be operated in configurable modes. These modes can be set up to allow for zero-wait-state operation when desired as long as the fetches from that image of DRAM do not violate the rules set by the mode configuration. This restricts zero-wait-state operation to code that is tightly controlled, but this is certainly the case for the speed critical portions of the code. The multiple images of DRAM allow one to switch the fetching configuration easily and quickly depending on the code being executed. This operation of the buffer allows us to achieve nearly the same performance with DRAM as one would achieve with fast SRAM.

The serial unit in the ASIC also has some novel features that allow us to efficiently transfer blocks of data to the neighboring nodes as well as perform global system operations. LQCD requires that the data to be sent in one of the directions corresponding to those on one side of a 4-Dimensional cube. For this reason, it is not possible to arrange the memory such that all surface sites are contiguous memory locations. This is nicely handled by allowing the DMA controllers to have an additional increment, which allows one to effective cover the entire face without any DSP intervention. The other important requirement of the serial unit is that it is able to handle global sums. This requirement has been met by implementation of a 4-dimensional serial adder on each node. Each node sends out its result as soon as possible (a clock after the corresponding input bits are received). The important point here is that the latency for the result is minimal since the total sum spans many nodes due to the serial nature of the sum. This allows one to handle a global integer sum in a very small number of clocks.

In order to allow for a global floating point sum, we have also included a global maximum operation, which works much like the global addition. This results in the maximum of the inputs of all the nodes involved. This allows one to achieve a floating point global sum by comparing all the exponents and then normalizing all the floating point numbers to be added by this maximum exponent. A global broadcast operation is also implemented to allow for the very efficient broadcasting of information over many or all nodes.

A note here on the cost effectiveness of this machine is warranted. The current cost of a node with 2Mbytes of DRAM is about \$100 including parts and assembly labor. The full system cost is dominated by this node cost and is about \$3/MFLOP. This cost is completely scalable (in units of 3.2 GFLOPS) and therefore makes this machine very attractive economically.

5 Software Support

We have developed software in both C, C++ and assembly. We have a C compiler for the DSP, which was supplied by Texas Instruments. We also have a C++ compiler from Tartan with which we are now gaining experience. The assembler is also from Texas Instruments. We have developed a simple model for the kernal operating on each node. Basically, each node interprets the packets that arrive on the SCSI bus or over the DSP serial links (depending on whether the node is "node-0" or not on each motherboard) as follows,

- Load data/code into memory,
- Execute code at pointer (supplied),
- Upload data from memory .

The kernal running on node-0 of each mother board is somewhat more complicated since it has to be able to handle requests from nodes which may be further down the SCSI tree. The user interface to the QCDSP machine is a standard X-interface which is fully functional.

6 Status/Conclusion

The current status of our machine is as follows:

- We now have 8 crates available for motherboards. The motherboard count is steadily increasing.
- We have operated 1024 nodes (two crates, 50 Gflops).
- We expect to run 2048 nodes (1 rack, 100Gflops) by April '97
- We expect to be running on 8192 nodes (4 racks, 400Gflops) by fall of '97.

We have are also involved with systems at other institutions:

- RIKEN at BNL where we expect to collaborate using a 600Gflop machine we are helping to build.
- SCRI (At FSU) (collaborators).
- Ohio State University (collaborators).

– University of Wuppertal.

We have successfully built a massively parallel computer based on DSP that is very cost effective and is efficient for LQCD. We have followed a model of fitting the hardware architecture to the problem. This has resulted in a 4-Dimensional array of MIMD nodes that are each extremely simple. This simplicity has allowed us to build, debug, and maintain this machine with minimal effort. We have chosen to use a software model that is also very simple as it is all that we require to handle LQCD.