COMPUTER CHESS

CHESS HIERARCHY MEMORY

MUHAMMAD AMIRUDDIN BIN ZULKIFLI
School of Computing
University of Technology Malaysia
amiruan12@gmail.com

***Abstract*—In this paper , I introduce how the CHESS computer memories work. CHESS is a distributed memory multiprocessor based on a grid topology. Processors and memories alternate on both directions of the grid to form a processing surface on which parallel programs are mapped. Memories are used for communication, to reduce the communication overhead, as well as for storing programs for execution This paper also will present a move-ordering algorithm that tend to mimic something of the human thought process when choosing chess moves.**

Keywords-component; Richard Greenblatt; processors; multiprocessors; CHESS hierarchy memory

I. INRODUCTION

Modern computers have the power to follow generalized sets of operations, called programs. These programs enable computers to perform an extremely wide range of tasks. Many existing computers today utilized the power of more than one CPU to provide better cost-performance than a single CPU or a supercomputer. These computers named "multis" by Gordon Bell [5] are mostly based on the shared bus - shared memory model, where several processors are connected to a shared memory module through a shared bus. Sequent, Firefly and the Encore are one of the examples of these machines.

These computers are easy to program and supply significant speed-ups thanks to its shared memory. .However, the cons that also intact, is that the limited bandwidth of the shared bus which restricts the amount of processors within the system which will be connected to an equivalent memory, and thus the performance of the system. To unravel this problem we will increase the speed of the bus [11], which isn't always easy due to technology limitations, or we will use more wires to attach to the memory. For a given technology, more wires provide more bandwidth, but it's not obvious which is that the best way to connect the wires due to complications like caches, code sharing and system complexity. Distributed memory architectures promise a far better capacity to be changed in size or scale, but it's harder to program efficiently. In [1] an architecture called CHESS was suggests which mixes the simplest of both worlds: that's , it's a distributed memory architecture, which is programmed consistent with the shared memory ideas.

Need to be mention that in this paper, I will simplified the Memory Hierarchy of the CHESS architecture which have been a successful research by D. Lioupis, N. Kanellopoulos and M. Stefanidakis in 1993 [9].

II. THE 'CHESS' HISTORY

From 1955 to 1958 the first actual computer chess programs appeared. In [1], there were three programs which corresponded exactly to the three categories that have been specified. Unfortunately, all three of these programs played very mediocre chess, for reasons that are now quite clear, but were hard to appreciate at that time. Because of this lack of success, computer chess was for a time thought to be too difficult to tackle effectively, although an isolated Batchelor's thesis was produced by Kotok 2 at MIT under the direction of John McCarthy.

In 1966, Richard Greenblatt [6] from MIT paved the way for his revolutionary efforts in computer chess. He wrote the first program that could command like an actual player. This program was good and far more better than any of its predecessors had. MacHack-6 (as Greenblatt's program was named) soon played in human tournaments, established a performance rating in the 1400-1500 USCF range (Class "C") and showed that this program capable of beating tournament caliber players. Its achievements spawned a broadside of efforts all over the world. Since 1970 there have been annual U.S. Computer Chess Championships (held concurrently with the ACM annual meetings), and a World Computer Championship series was begun at IFIPS in Stockholm 1974.

Unfortunately, Greenblatt, who started it all, could not be enter his program in any of these events. During this period, the Northwestern University chess program, programmed by David Slate and Larry Atkin, and variously named CHESS, that also evolve inti CHESS 3.0 to CHESS 4.2 dominated the chess scene. It won every U.S. Computer Championship except the 1974 event. Years after years improvising the program , the fall of 1975, CHESS 4.4, the latest incarnation of the powerful Northwestern University Chess program, again won the U.S. Computer Championship.

III. CHESS ARCHITECTURE

3.1 Rationale

In order to achieve high performance when executing parallel programs in a parallel architecture, a minimum communication time must be kept. Communication is used to transfer work to other processors and synchronize operations in different processors. In shared memory architectures, communication between processors is performed through the memory while, in distributed memory architectures, communication is based on message passing.

It is important to note that when a processor is assigned a work granule, the appropriate code must be tranferred to its memory if it is not already there. It is therefore advantageous to store the code directly in the memory rather than passing it through a communication network. CHESS achieves this by using the memory as a communication media..

3.2 Logical Model

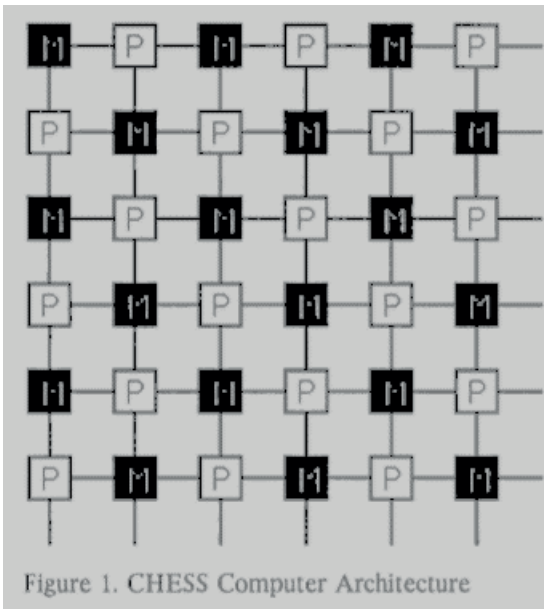

Figure 1. CHESS Computer Architecture

Figure 1, show the CHESS architecture which is a partially shared, distributed memory architecture. Each processor (P) is connected to four memories and each memory to four processors. The grid that is formed by such an interconnection is wrapped around in both directions which can be expanded in either direction by connecting more memories and processors. The architecture, therefore, can support large number of processors (>100) which can execute parallel programs.

3.3 Execution model

The execution model of CHESS is predicted on the idea of mapping the execution tree of a parallel program onto a grid of processors. One processor is assigned the root of the tree (which is the whole program) and spreads out the work to its neighbours until the program is executed. The rules that monetized this map is the diffusion algorithm and the programming model.

The diffusion algorithm determines where and when the work are going to be assigned. Its effort is to specialized balancing the load on the processor grid so as to extend the efficiency. Therefore, it plan to distribute the work as many as possible while putting communication cost at its lowest. The detailed description of this algorithm isn't adressed in this paper but we will assume that the OS running on CHESS includes such an algorithm. Algorithms similar to those utilized in [2], can be used

Spreading out work consists of copying a work granule into a processor's memory. A work granule is a self contained piece of work 50-500 lines of code. Each granule consists of :

(a) *a header* which indicates the needed resources for the execution of work and

(b) *a body* that contains the code to be executed.

A granule is assigned to a processor consistent with the data dependencies and the processor's load.

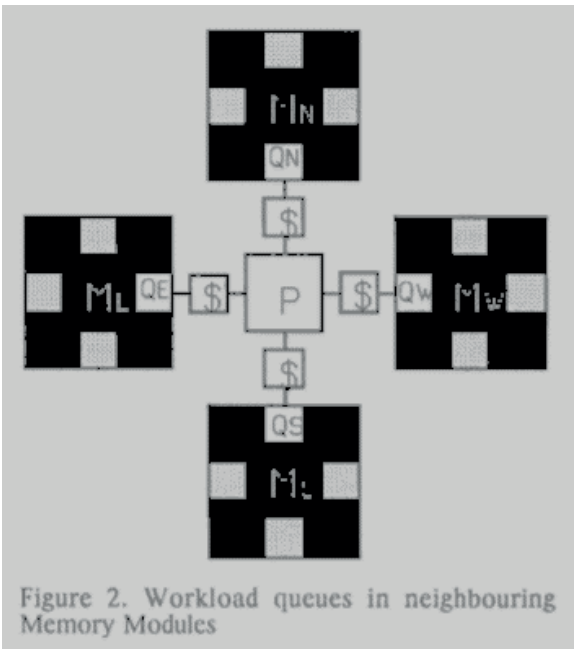

Figure 2. Workload queues in neighbouring Memory Modules

Figure 2 shows a processor's work queues. Due to the topology of CHESS there are four work queues for each processor, one for each neighbouring memory. QN is the north queue in the north memory module (MN). When a work granule

is assigned to this processor, a pointer is set in its work queue. The processor services each job in turn from its work queue.

Each memory module also contains another set of four queues which are used to buffer through traffic generated by processor communication. The OS assigns higher priority to these queues enabling other waiting processors to continue their work.

Let us assume that the processor north of P wants to assign some work to processor P. The code associated with this work is then copied by this processor in MN and a pointer to this code in QN. Processor P periodically scans the work queues and executes any remaining work. This operation is supervised by the OS which ensures correct operation. In order to simplify the architecture, the OS code is also executed by the processors.

3.4 Physical Implementation

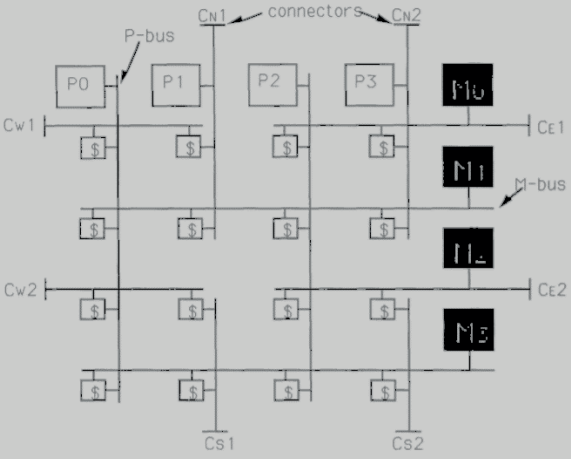

Figure 3 . CHESS Physical Bus Interconnection Tile

Since a memory connects to four processors, the four ports of this memory are implemented using four caches ($) interconnected to a memory bus (M-bus) as can be seen in figure 3 (M 1 memory bus). The four processors, four memories and bus network (P-buses and M-buses) define a CHESS physical tile. The first and third M-bus as well as the second and fourth P-bus are shifted by two places in order to form the required grid interconnection. The lines at the end of the split buses indicate connectors (CN1, CE1, etc). the smallest configuration possible which is 4 processors and 4 memories, is formed by connecting together the north connector with the south and therefore the east with the west.

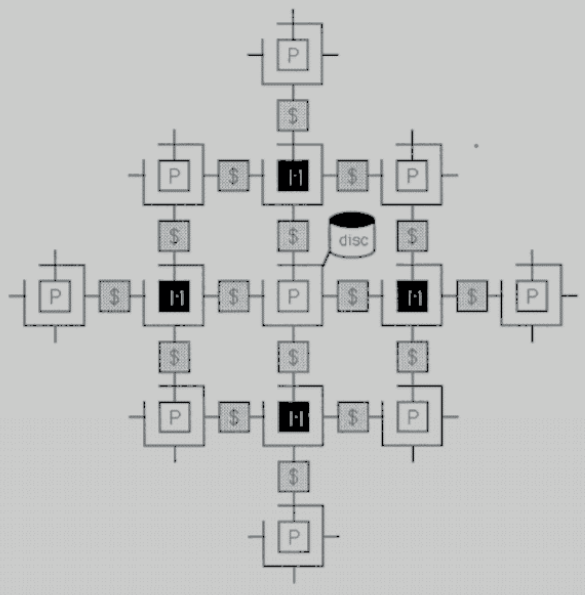

Figure 4 . A CHESS Memory Cluster

IV. MEMORY HIERARCHY

Basically, the memory hierarchy of CHESS computer are so near to resembles the memory hierarchy of shared bus multiprocessors. Figure 4 shows a memory cluster which is when a processor connected to its four neighbouring memories that in turn are connected to their neighbouring processors (eight processors all together). The boundaries of the memory cluster are the eight surrounding processors (boarder processors) which are shared by the neighbouring clusters. To increase the bandwidth to memory, we use a multiple bus architecture. The resulting architecture uses a grid of buses with a cache at each cross-point to attach to memory.

Each memory is split into pages. For each page, it is swapped to the disc when a memory module becomes full.The disc controller is connected to the central processor P-bus of a memory cluster, which controls the paging of its four neighbouring memory modules. This suggests that only one disc is required per four memory modules which is consistent with the physical implementation described. For an instant, in figure 3 shown the physical tile where the disc controller can be connected to the P2 processor, which connects to all four memories of this particular tile.

There are two typical operations within the CHESS architecture concerning to the memory. The information been safely transferred in this traffic because is it marked as non-cacheable.

a) Memory-to-memory transfer within cluster:

this operation can be performed directly either by the processor located in the center of the cluster, or by a processor common to both memories

b) Intercluster memory-to-memory transfers:

this operation can be performed directly if there is a processor common to both memories (i.e. a border processor), otherwise, the transfer will be routed through the shortest route.

Using memory as a buffer for communication simplifies the architecture design in the expense of increased traffic on the bus network. Since there are only four processors on the M-bus we expect that memory buses will not be heavily loaded. The cache write back replacement strategy employed will further reduce this load.

V. MEMORY CONSISTENCY

How great and impactful the programming model and memory hierarchy of CHESS shown when it can allow multiple copies of an equivalent variable to exist in the system so as to extend performance and reduce hot spots. Since any processor can update this variable at any time, the architecture must provide a mechanism to ensure global data consistency. A program being executed on CHESS contains private and shared data. The local variables and constants of a procedure are the private data which reside together with the code in the same memory module. There is no consistency problem with these variables. Shared data on the other hand are global variables accessed by more than one procedure. These procedures can be executed by different processors, all of which need access to the variables. According to how far away these processors are, consistency will be maintained by two protocols:

1. Neighbouring processors which share data through a common memory module are connected on a shared bus through a cache. These caches employ a simple bus snooping consistency protocol like those described in [3] in order to maintain consistency as described in section 5.1.

2. If a processor needs to access a shared variable which does not reside in its memory, it forwards a requesting message to a processor neighbouring the memory module which holds this shared variable. The reply message contains the data including all the required information to implement a weak order consistency protocol called "freeze" consistency protocol which is described in section 5.2.

5.1 Snooping Cache Consist

Due to the inconsistency occurs in the cacheswithin the first level of the implementation of CHESS, any cache attached to a memory can request a copy of a variable. Since these caches are connected to a common bus, standard bus snooping protocols can be used to maintain consistency. Shared data, however, generates some inconsistencies which are handled by the snooping protocol. With the introduction of multis a lot of research effort was directed in the area of cache consistency on a shared bus. In the CHESS architecture, due to the small number of processors connected on the same bus there is no great strain on the bus and consequently no strain on the snooping protocol. Since we plan to use off-the-self components we will adopt the snooping protocol implemented in the cache used. We consider that a protocol similar to the one implemented by the SPARC cache controller is adequate for CHESS.

4.2 Freeze Consistency Protocol

While memory modules are consistent with their caches, memories must remain consistent with each other since copies of shared data are allowed to exist in different memory modules. In maintaining memory module consistency, CHESS operates as a weak ordered system. As defined in [8] , weak ordered systems consistency is enforced only on synchronization barriers. A few consistency protocols have been defined for weak order systems. The release consistency described in [4] which was designed for shared memory systems. It allows computations to carry on after the release of the lock. This protocol cannot be used in a distributed memory system.

CHESS was defined as a weak order algorithm because it relies on the use of synchronization primitives to protect the shared data in order to maintain the consistency. As defined by the programming model of CHESS, data moves together with the code from memory to memory until it is assigned to a particular processor. When a processor writes a new value to this data for the first time, further movement is restricted. That is, it is "frozen", hence the name of the algorithm. At that time the data is marked as a master copy, indicating that the master copy of the data will reside in this home memory from now on. If another processor requires to use this data, it must obtain the lock that protects it and then fetch a copy of the data from the home memory.A request will be forwarded to the appropriate neighbouring processor which will attempt to obtain the lock. When this is achieved, the processor will reply with the required copy which is then stored in the requesting processor's neighbouring memory. The processor will exit the critical section and sends a message back to the home memory module with a copy of the updated data right after the processor finished using the data. The same message contains the lock release which ensures that data will be updated before the lock is released again.

VI. RESULTS AND FUTURE WORK

A simulator was developed in C using SIMON [7] to allow an experimentation with the diffusion algorithm. In [7] also demonstrate a performance evaluation and software development and most of the program performance measurements were obtained which indicate the time spent by CHESS accessing memory because of :

a) program execution,
b) diffusing work to other processors and
c) communication traffic.

Based on obtained results CHESS simulator in [9] , CHESS spends about 10%-35% of total executeion time for work which is not directly connected with program execution depending on the benchmark.

For message passing take only 10% except;

1. Matrix multiplication [10] that performs a classical multiplication of two matrices (NxN), one of which is fixed in a memory. This program stored as global arrays to produce the result matrix. Each processor calculates a part of the resultant matrix determined by its number, which is used to index the resultant array. This means that eventually each processor will fetch all elements of the first matrix and a column of the second matrix.
2. SOR(NxN) [10] which applies a Square Overelaxation transform to a matrix (NxN). This program repeatedly computes the value of grid locations by taking the average of the four surrounding points. Eventually, this method converges to a solution of Laplace's equation for the given boundary values. Several passes are performed before the final result.

This greatly depends on the algorithm that the application used. For example the matrix multiplication which was written on purpose to generate a worst case situation shows large message passing time. Similarly, the SOR operates on three consecutive rows of the array two of which are shared with other processors. A better coherency protocol which will allow shared variables to be distributed amongst the processors will improve this time.

Load calculation takes about 10 - 15% of the total execution time.. More experimentation with the simulator is required to evaluate the relative benefits of infrequent load calculation. The two large benchmarks which are Matrix multiplication and SOR show very little load calculation time because the mapping tree is not very deep and thus work is diffused immediately.

The obtained results have shown that programs can be executed on CHESS with small overhead (up to 35%) and have indicated areas where the CHESS architecture could be improved. One of the definite parts is the OS and the diffusion algorithm which needs to be developed further. As mentioned before, The OS need to construct a higher priority to these queues enabling other waiting processors to continue their work.

Also another parallel kernel called "Wanda" which was developed at Cambridge University, can be used which allows integration of custom code into the kernel. The OS code will be ported on the prototype and developed further to be customized for CHESS.

VII. CONCLUSIONS

I have summarized a little bit about the memory hierarchy of a distributed memory parallel computer, CHESS computer in [9]. CHESS computer program that started by David Slate and Larry Atkin which inspired by the their rivalry occurred with MacHack-6 was an impactful and revolutionary acts in human history. From what I even have presented, the CHESS memory hierarchy resembles that of a shared bus multi. simulated performance of a family of multiprocessor architectures based on a global shared memory. The processors are connected to the memory through caches that snoop one or more shared buses in a crossbar arrangement.We can solve the problem of keeping the memory consistent by a combination of two methods: a snooping protocol and a weak order protocol. The majority of memory accesses, as indicated by the obtained results, were performed to a neighbouring memory module. By reducing the long haul communications and the consistency problems that these remote accesses imply, we expect the CHESS computer to give a very high performance.

REFERENCES

[1] "A Chronology of Computer Chess and its Literature" by Hans J. Berliner Computer Science Department, Carneyie-Mellon University, Pittsburyh, PA, U.S.A.

[2] JOHN SUSTERSIC AND ALI HURSON, "Coherence Protocols for Bus-Based and Scalable Multiprocessors, Internet, and Wireless Distributed Computing Environments: A Survey," in The Department of Computer Science and Engineering Pennsylvania State University 202 Pond Laboratory University Park, PA 16802 USA.

[3] Shaily Mittal, Nitin, " A New Approach to Directory Based Solution for Cache Coherence Problem ", Department of computer science, Chitkara University,Baddi- 174103, Solan, Himachal Pradesh (India)

[4] Omri Bahat∗, Armand M. Makowski, "Measuring consistency in TTL-based caches Department of Electrical and Computer Engineering, The Institute for Systems Research, University of Maryland, College Park, MD 20742, United States

[5] C. Gordon Bell, "Multis: A New Class of Multiprocessor Computers," American Association for the Advancement of Science, 26 April 1985, Volume 228, pp. 462-467.

[6] Greenblatt, R. D. et al., The Greenblatt chess program, Proceedings of the 1967 Fall Joint Computer Conference (1967) 801-810.

[7] Richard M. Fujimoto," SIMON : a Simulator of Multieomputer Networks ", Report No.UCB/CSD 83/140, Computer Science Division, Univercity of California Berkeley, September 1983.

[8] ROBERT C. STEINKE , GARY J. NUTT, "A Unified Theory of Shared Memory Consistency", University of Colorado at Boulder

[9] D. Lioupis, N. Kanellopoulos and M. Stefanidakis, "The Memory Hierarchy of the CHESS Computer", University of Patras, School of Engineering, Dept. of Computer Engineering & Informatics, 26500 Rio-Patras-Greec

[10] Andy Hopper, Alan Jones, Dimitris Lioupis., Multiple vs Wide Shared Bus Multiprocessors, Olivetti Research Ltd, Keynes House 24A Trumpington Street, Cambridge CB2 1QA, England

[11] Digital Equipment Corporation, "CVAX-based Systems", Digital Technical Journal no. 7, August 1988