A Scaleable Multiprocessor Architecture with Multiple Read-Write Memory Model

Cemal Köse,[†] *and Özcan Özyurt*[††]*,*

Karadeniz Technical University, Department of Computer Engineering, 61080 Trabzon TURKEY

Summary
This paper presents a scalable multiprocessor architecture with multiple access memories and multi-way busses. This parallel architecture with more intelligent memory model and efficient multi-way interconnection network organization is called as CRrCW (Concurrent Read and restricted Concurrent Write) scaleable multiprocessor system. The memory and network model provides concurrent memory accesses and more memory bandwidth for a CRrCW scaleable multiprocessor system. Thus, the proposed multiprocessors system with the memory model and network take advantages of both multiprocessor systems with shared memory and distributed memory. In this paper, conventional shared, distributed memory multiprocessor and the CRrCW multiprocessor are examined under various conditions and also compared with each other. A sliding caches memory model is used to reduce the cost of the CRrCW multiprocessor system. A cluster controller is also used to enhance the broadcast and multicast ability of the interconnection network. Simulation results of the sliding caches memory model show that memory requirements of a CRrCW multiprocessor system is almost independent of scale of the multiprocessor. In addition this, simulation results of the interconnection network also show that even small number of the connection per processors greatly enhances the scalability of the multiprocessor system. Two scientific applications, volume visualization and N-body problem, is chosen to examine performance of the system. These applications case studies represent a range of important behaviors found in parallel computing. The CRrCW multiprocessor system with special interconnection network and intelligent memory organization provides high speed-up and efficiency over conventional shared and the distributed memory multiprocessor. Therefore, the CRrCW multiprocessor system is a potential candidate for large-scale parallel computing applications.

Key words:
Multiprocessor systems, multiprocessor architectures, memory architectures, scaleable multiprocessor, simulation of multiprocessor systems.

Introduction

Conventional multiprocessor systems can be classified as shared (centralized) and distributed memory multiprocessor. A shared memory multiprocessor is known as tightly coupled systems that whole processors on the system share a common memory space. All data exchanges and synchronization are done on the shared memory space. The shared memory multiprocessor allows the processors to communicate by reading from and writing to the common address space, so a shared memory multiprocessor based on buses or cross-bar switches are convenient for programming but are not scalable due to memory contention when processors accesses memory. A distributed memory multiprocessor is also known as loosely coupled systems ([1], [22], [32], [38], [40] [53]). Each processor on this system has its own private memory. Inter-processor communications are done by sending messages to appropriate processor along an interconnection network ([4], [8], [23], [25], [45]). This increases delay and cost of the communication. Therefore, message-passing or distributed memory multiprocessor is scalable but more difficult to program than the shared memory multiprocessor.

Scalable shared memory multiprocessors aim to combine the benefits of both shared memory and distributed memory multiprocessors, by supporting a shared address space on top of physically distributed main memory (virtually shared memory multiprocessors) ([1], [2], [12], [14], [28], [36], [42]). There are several shared memory multiprocessors but they are not scaleable and efficient enough for large-scale applications ([4], [15], [19], [27], [45], [56]). In this paper, proposed model of parallel processors with a special network and memory organization offers a solution for scalability and programmability. Here, multiple access memory organization is proposed that it allows concurrent reading and restricted concurrent writing to a remote memory module. A clustered broadcast and multicast interconnection network is also proposed to enhance communication across the multiprocessor system. Then, a memory access unit employed to handle communication, memory accesses, and maintain consistency across the system. This reduces complexity and overhead of processors because of the communication.

This proposed multiprocessor architecture with an intelligent memory and a special network organization is similar to the distributed memory multiprocessor because each processor on the system has its own private memory. In addition to this each local memory is also represented in

Manuscript received September 5, 2006.
Manuscript revised September 25, 2006.

a common shared memory space. In other words, all local memories are presented in a shared memory space that can be directly accessed by any processor without interrupting the local processor. Thus system with the intelligent memory network organization allows concurrent reading and restricted concurrent writing. Consequently, this model takes advantages of both shared and distributed memory multiprocessors such therefore it is a scalable shared memory multiprocessor ([4], [9], [12], [13], [27], [38]). On the other hand, this proposed distributed multiprocessor model with low memory contention and delay is more programmable than the conventional distributed memory multiprocessors. Simulation results also show that the CRrCW multiprocessor system is a suitable candidate for general-purpose parallel computing ([1], [7], [11], [16], [30], [35], [39], [54]).

2. Scalable Multiprocessor Architecture

The main difference between shared and distributed memory multiprocessor is in their organization of interconnection methods. A connection method on a shared memory multiprocessor system allows all processors to access shared memory. If more than one processor wants to access the shared memory at the same time, an arbitration mechanism ensures one processors access that memory portion at a time. Thus, increasing the number of processors on a shared memory multiprocessor creates a bottleneck of memory access. On the other hand, each processor on a distributed memory multiprocessor has a private memory instead of a shared memory for all processors. If one processor wishes to access another processor's private memory, it can only do by sending a message to appropriate processor through the interconnection network. Therefore, distributed memory multiprocessor has no memory contention problem because each processor on the system has its own private memory that can be accessed by only the local processor. Here, the network and hardware organization of the distributed memory multiprocessors have to be known by the programmer due to the fact that these details may make difficult programming on a distributed memory multiprocessors. Also efficiency of a distributed memory multiprocessors may not be very high because of limited number of communication channels, high density of the messages across the system and high memory latency at remote memory accesses. However, distributed memory multiprocessor systems are more scalable than a shared memory multiprocessor systems but not easily programmable.

All processors on a CRrCW multiprocessor system as illustrated in Figure 1 have a private memory and cache for scalability. All of these private memories and caches are also represented on a common shared memory space so that all processors on the system can access them. In this case, the CRrCW multiprocessor behaves as a shared memory multiprocessor and represents characteristics of a shared memory multiprocessor system. So other remote processors can access any memory module on the shared memory space without interrupting access of the other memory modules on a local memory module. Then, for a processor reading from and writing to its private memory is called as local memory access, and whereas reading from and writing to a remote memory module is called as remote memory access. Here, P_x, M_x, C_x, SC_x, and MAC represent Processors, Memory, Cache and Memory Access Controller respectively. This multiple access memory architecture enhances the bandwidth between processors and memory modules.

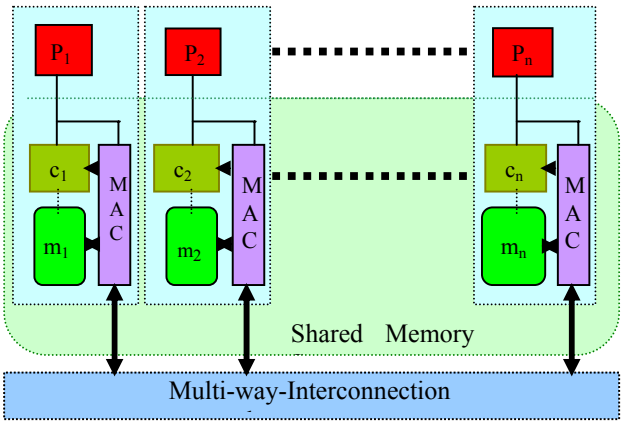


Fig. 1 A simple CRrCW multiprocessor model.

3. Concurrent Read and restricted Concurrent Write Multiprocessor Architecture

In literature, several scalable-shared memory models were presented so that these models differ in how to handle read or write conflict from the main memory ([1], [12], [13], [14], [38]). Several multiprocessor systems with different global memory access policies were implemented such as DDM and KSR1 ([26], [51], [56]). Recently, several research companies have announced a few high performance parallel computers. One example of these high performance computers is introduced by Silicon Graphics so that it has a more loosely coupled globally shared memory with larger average remote memory delays [52]. Another latest system, relies on multidimensional networks, is introduced by Pittsburgh Supercomputing Center ([21], [46]). In general, these systems have very complex memory and network structure ([2], [6], [20], [29], [37], [41], [48], [49]) and data consistency protocols of these systems are very complex and expensive. For example, exclusive writing is very complex and expensive

operation because it requires cancellations of other copies of a data across multiprocessors. Scalability of these systems may be questioned. Proposed CRrCW multiprocessor model supports Concurrent Read restricted Concurrent Write (CRrCW) ([13], [14]). In this model, concurrent reading is allowed but restricted concurrent writing is permitted. In other words, this model allows restricted concurrent writing for remote processors and the number of concurrent writes to a remote memory location is limited by connection network and current status of the sliding caches on the memory unit.

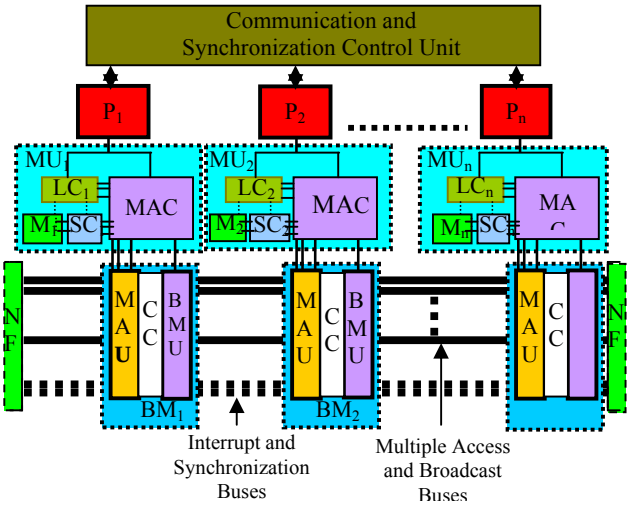


Fig. 2 A CRrCW multiprocessor architecture.

Each processor on CRrCW multiprocessor system as illustrated in Figure 2 has a private cache and memory for scalability. All these private memories are also represented on a common shared memory space that can be accessed by all other processors. In this case, the CRrCW multiprocessor behaves as a shared memory multiprocessors and represents characteristics of a shared memory multiprocessors. A processor can access any memory module on the shared memory space without interrupting access of the other memory modules so each processor can access its own private memory without any interrupt by any other processors. Thus, any processor through the special network can access any memory module on the shared memory space. Here, the number of the processors on the system does not limit this access but the number of the buses. For a processor reading from and writing to its private memory is called as local memory access, and whereas reading from and writing to a remote memory module is called as remote memory access. The amount of remote memory accesses may affect the efficiency of the system but the number of the local memory accesses. Therefore, one thread may be suspended awaiting a remote data item, the other threads may still be able to continue. To maximize the performance of the system multi-threading overlaps the communication and computation. If there are sufficient threads then any processing element should always be performing useful computation. Hence, a large-scale this multiprocessor system with enough number of buses and the proposed memory organization provides more potential for parallel processing. Here, it is suggested that memory accesses, broadcast and multicast on the shared memory space are also performed by hardware, so reducing the communication overhead increases scalability and efficiency of the CRrCW multiprocessor system multiprocessors. Then, a CRrCW multiprocessor system with the shared memory space and private memories of each processor is a scalable-shared memory multiprocessor architecture.

Architecture of the CRrCW multiprocessor system illustrated in the Figure 2 consists of processors, memory units, bus managers, and busses. A memory unit consists of a multiple access Local Cache (LC_x) and a private Memory (M_x), Sliding Caches (SC_x), and Memory Access Controller (MAC). Each bus manager consists of a Memory Access Unit (MAU), a Bus Management Unit (BMU), and Cluster Controller (CC). Each processor on a CRrCW multiprocessor has a private memory so each processor can exclusively read and writes its own local memory. Each these local memories are also represented on the shared memory space by memory modules. These multiple access memory modules are concurrently read by all processors and written by the local processor and restricted concurrently written by the remote processors. The scaleable (CSCU) uses a released consistency model so that date items across the system are updated by using the consistency repair mechanism but paging tables is updated constantly. Then, no false sharing false sharing is assumed and implemented because the communication and synchronisation control unit maintains the data consistency across the parallel system. The communication and synchronization busses are used to synchronize the processors and help the clustering. Network Folder folds and connects the busses with each other, and then clusteral usage of the buses is allowed. Of course without folding a bus multiple clusters can be created on a bus, but folding a bus increases the potential number of cluster that could be created on a bus. Then each bus can be broken into small clusters and each cluster on a bus is used simultaneously. This improves the performance of the applications with local communication tendency. Communication and synchronisation control unit (CSCU) co-ordinates the communication, synchronisation and consistency across the system.

3.1 Multiple Access Memory Model for the CRrCW Multiprocessor System

As illustrated in Figure 2 a memory model for the CRrCW multiprocessor is not organized as a conventional memory. The system allows two different types of memory access operation. The first type is a locally read/write operation as local memory access. The second type is a remote read and restricted write operation as remote memory access. So this memory model allows one or more processors to access to a memory modules under some restrictions. While a local processor accesses a local memory on its cache and local memory without any restriction, a limited number of remote processors also access to the memory module on the memory unit simultaneously. Here, the limitations depend on the number of buses and multiple access memory modules on a memory unit. This memory model with the network organization allows exploiting full bandwidth of the communication network dynamically so that this isn't related with the communication demand or pattern of a problem.

All processors may write its own private memory and all memory modules on its memory unit at the same time but it can only read from its private memory. A processor may read from any remote memory module at a remote location and a limited number of remote write to a remote location is allowed. Here, two of the ways of sharing and updating the global variables are possible. When a processor needs a remote data, a bus between processor and remote location is allocated for the processor. A remote memory module at the remote location is connected to the processor. The processors execute limited number of read operations at the remote location. The number of read operations is limited because of fair usage of the buses. Second is the way of the sharing of global values is that the producer of the global values updates all the copies of the global variables at once via the clusteral multicast busses. Here, multiple copies of the global shared variables may be created on each processor. Each writing processor broadcasts to update these global variables. On the other hand, a process produces a result and updates its copy of the shared data and the remote copies of the data via the clusteral busses. Here, system software may manage shared variables on each sharing processor. Then, when it is needed, a processor updates a data at a remote location while it is updating its local copy.

The number of buses and scale of the system determines the memory requirement of the multiprocessor system. Then, the required memory for a system with *n* processors and *m* buses is *n times m times* of the local memory size if non-restricted concurrent memory access is allowed. Here, each local memory unit consists of *m* memory module so that each memory module on the same memory unit holds the same data. This increases the cost and complexity of the system dramatically. In this model of parallel computers, the memory size of the system increases with the number of buses. For a large-scale system, a memory unit cannot be designed as single memory module because of the wiring complexity and memory contention. Then, two methods may be applied to reduce the memory size of the system. In the first method, memory modules are divided into sub-modules and a processor accesses each sub-module simultaneously. Each unit may be divided into separate memory modules so this reduces memory contention. Thus, the number of parallel accessible memory modules may be chosen equal to number of busses, and then each bus will have corresponding memory module on each memory unit. This makes almost impossible implementation of the memory models because of wiring complexity and cost of the memory. In the second method, a module may satisfy more than one memory request at the same time by using sliding caches memory model. Sliding caches memory provides a caching technique that any cache module may correspond to any memory module and overlap with another cache modules according to the memory access pattern. When a sub-memory module is contented with memory request, more copies of the sub-memory modules may be created to satisfy memory request simultaneously, and these copies satisfy some of the later memory accesses. A simple profiling technique is used to organize the sliding caches. Then, the number of memory modules is not equal to the number of the buses so memory requirements reduced substantially by using the multiple access memory modules and sliding caches.

3.1.1 A Typical Multiple Access Memory Module for the Multiprocessor System

One or more processors may access same memory unit simultaneously. The proposed memory module is illustrated in Figure 3, and L_x represents multi-way latches and decoders decode addressees on the busses. If two processors need to access the same memory location at the same time, one of these could be blocked. This restriction is the most important problem of the memory organization. Thus, there are four possibilities. Firstly, both processors read the same memory location simultaneously. Here there is no problem so both processors may continue reading the memory location simultaneously. Secondly, while a local processor is writing a memory location, another processor is reading the same memory location. In this case, data consistency may be disrupted because of the simultaneous access. Thirdly, while a remote processor is writing to a memory location, the local processor reads from the memory location. Lastly, both remote and local processors

are writing the same memory location. When the four cases are concerned, any remote processors should not block the local processors while they are operating on a remote memory. If a processor should be blocked because of operating on the same memory location, it must be the remote processor. Here, it is aimed that any processor fully able to control its local memory without considering data consistency. Data consistency is maintained by CSCU and higher-level protocols.

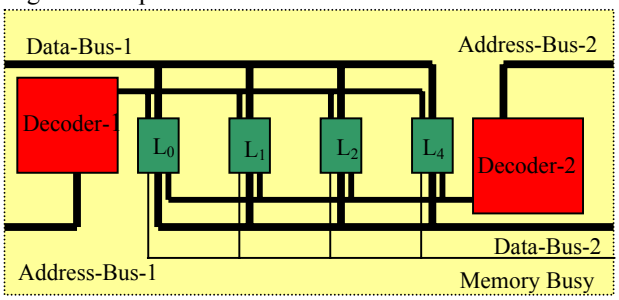


Fig. 3 A typical multiple access memory module.

Here, all cases should be examined carefully because the efficiency of a system widely depends on organization of the memory unit. When the details of the model are carefully examined, three critical timing problems are observed. The first problem occurs while a local processors writing a local memory location, another remote processor wants to read/write from/to the same location. In this case remote processor is blocked by the memory busy signal. The local processor finishes the operation and then the remote processor is released. The second problem is that a processor is reading/writing from/to a remote memory location then the local processor started to write at the same memory location. The remote processor may continue immediately after the local processor finishes the reading/writing. The third problem arises when a local processors starts to write to a memory location and a remote processor starts to read/write from/to the same memory location at the same time. The local processor finishes its operation and the remote processor reads/writes. Here, data dependencies are not considered and data consistency is maintained by a higher-level protocol ([2], [3], [5], [22], [32], [44], [51], [56]). Here, dividing the memory access operation into time slices and one micro level memory access operation is executed in a memory access cycle such as address decoding, reading from, writing to, and etc. A memory access is divided into number of stages of memory operations as equal time slices. If more than one processor operates on the same memory location at the same time, remote processor may be delayed at the most two micro cycles. Then, memory delays may be reduced to a small fraction of the time that is determined by period of the clock.

3.1.2 Multiple Access Sliding Caches Memory Model

Increasing the number of processor on the multiprocessor system increases the memory requirements of the system. Dividing memories into small memory modules is still expensive because large-scale multiprocessor requires large memory sizes for scalability. Using several single blocks overlapping caches are also not flexible and feasible to support all memory access patterns. On the other hand, implementation cost of the memory model is at acceptable level when sliding caches memory model is used and independent of scale of parallel system. As illustrated in Figure 4.a and b this memory model may answer to all sequential, local and random memory accesses. Simulation results shows that sliding caches memory model reduces the memory requirements of the parallel system so that doubling local memory size makes the parallel system almost independent of scale of the system.

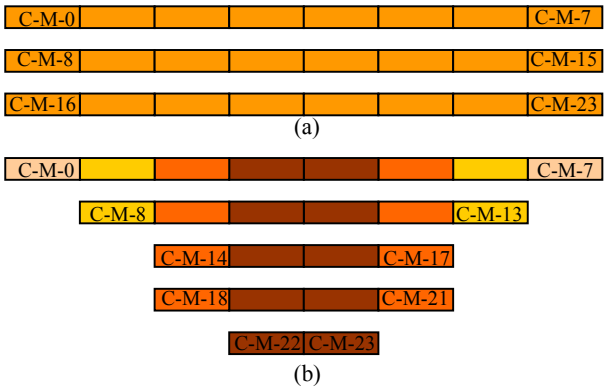


Fig. 4 Sliding caches for (a) a random (b) a sequential access pattern.

A sliding caches memory unit is consists of many multiple access sliding caches overlap with each other according to the memory access demand. This memory model with small cache blocks provide a caching technique that any cache module may correspond to any memory module and overlap with another cache modules according to the memory access pattern. This memory model can serve to all local and global memory accesses without increasing the memory delay. Then, memory requirement or memory size of a CRrCW multiprocessor system is almost independent of the scale of system. This is one of the important factors while designing a large-scale CRrCW multiprocessor system.

3.2 Bus Manager for the Multiprocessors System

Both memory and bus organization are very important for an efficient multiprocessor system. As illustrated in Figure 5 a bus manager consists of three units. The first unit on the bus manager is the Memory Access Unit (MAU) that connects buses to target multiple access memory modules. This memory access unit decodes addresses on the bus (DB: Decoder and Bus Directory) and connects the bus and memory module for a remote access. The second memory unit on the bus manager is a Bus Supplier Unit (BSU), which supplies buses for each processor pending a bus. The third unit is Cluster Controller (CC) for creating clusters for multicast according to the demand of the problem. Here, Bus Switch (BS) is used to allow multiple segments (cluster) on a bus simultaneously. Then, processors uses this multicast clusters to broadcast messages to its neighboring processors.

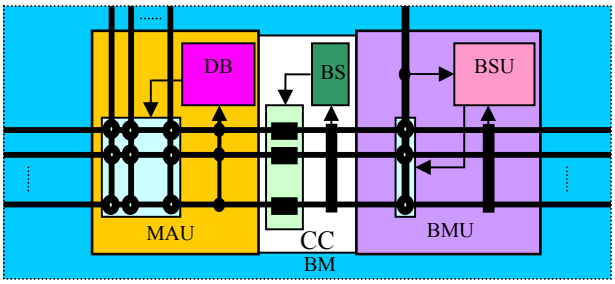


Fig. 5 Typical bus manager architecture.

The bus manager unit supplies buses for each bus requesting processor. When a bus supplier receives a bus request for a processor, the bus supplier allocates a non-occupied bus for that processor immediately. If more than one processor are racing for a single non-busy bus, a fair bus allocation schema is need for the best performance. The bus management unit provides a fair allocation of busses for simultaneous accesses. Then bus is allocated for one of the racing processors and other processor waits for a bus to be released. Two bus management policies may be applied. These policies are centralized and distributed bus management. For each requesting processor the bus manager allocates a bus that is not in use. Then bus director directs the bus to a memory module on the memory unit by checking the address available on the bus. Broadcast and multicast writes is performed on each processors along the spanning cluster. Finally processor bus memory connection is established. For limited period of time, the processor or a direct memory access unit transfers the data. Then, the bus is released for another usage. Bus contention may occur on large-scale system because of excessive remote memory accesses at a short time period. When the system supplied with satisfactory number of buses for a problem, it is expect that the bus contention would not occur. Experiments show that the number of buses needed is about one fifth of the number of processors on the system for an efficient multiprocessor system. So, the number of busses needed for a large-scale CRrCW multiprocessor is a small percentage of the number of processors on the system. Of course, this also depends on the application running on the system.

Although complexity of the system mostly depends on the number of the buses on a system, a large-scale CRrCW multiprocessor should be supplied with satisfactory number of buses. Here, three implementations of buses are possible. The first model includes complete control, address and data buses that are necessary for fast memory access. In this case, wiring complexity may make impossible design of the system. In the second model, using special technologies such as fiber optic technology and multiplexing techniques reduce numbers of bits on the buses. Memory accesses may not be as fast as those of first model because of the technology but wiring complexity is reduced dramatically. The last model may use technologies such as Local Area Networks that may be recommended for medium and coarse grain parallelism ([7], [23], [24], [43], [45], [55], [57]). In this study, the second type of bus organization with clustering ability is suggested for the multiprocessor system. Each cluster on a bus may span several nodes related to the communication-demand of the application. The Interrupt and synchronization busses are used to coordinate communication and synchronization across the system.

4. Parallel Application Case Studies

Multiprocessor systems are used for a wide range of applications from scientific to commercial computing. In this study, two scientific case studies are chosen from scientific computing; one is from computer graphics, and another is from astrophysics. These case studies are chosen to represent a range of important behaviors found in other parallel programs as in ([17], [22], [33], [43], [45], [47], [53]). The first case study is from computer graphics. Volume visualization technique traverses large three-dimensional objects with local memory access pattern and renders it into a two-dimensional image for display.

The second case study is chosen as another important form of scientific computing problem. The computational complexity is determined by large number of bodies that interact with each other and move around in three-dimensional space as a result of these interactions. This N-body problem is commonly used for simulating galaxies in astrophysics, proteins, and electromagnetic interaction. Hierarchical N-body algorithm is also been used to solve

important problems in computer graphics. The hierarchical N-body problem has more irregular and unpredictable memory access pattern. These case studies are part of benchmark suite that is widely used in architectural evaluations in literature and they are used to illustrate architectural performances in this study.

4.1 Parallel Volume Visualization

Volume visualization is the generation of a series of images from discrete samples by moving the viewpoint when examining three-dimensional volume data. Then, volume visualization provides an important tool for extracting meaningful information from three-dimensional objects in a non-intrusive manner. Volume visualization process requires a very large amount of computing and memory resources, and thus even the synthesis of a simple image may take many minutes, even hours on a sequential computer ([17], [18], [19], [32]). Therefore, this is a good candidate problem for benchmarking a multiprocessor architecture. Volume visualization has also inherent significant variations in computational complexity associated with the tasks required to solve the problem and very large memory requirements thus complicating its parallel implementation.

The computational models for parallel volume rendering may be classified as either volume partitioning or image partitioning depending on how these tasks are carried out in the parallel implementation. In volume partitioning, the volume data is divided in to a number distinct or overlapping regions and each region is assigned to a particular processor on the system. The volume partitioning approach performs the reconstruction and re-sampling tasks with the volume data held at each processor. As large volume data set is distributed amongst the processors, each processor may only compute partial results of the tasks using their allocated portions of the volume date. In order to render the final image, it is necessary to combine the partial results computed by several processors. In image partitioning the image plane is initially evenly partitioned amongst the processors. Each processor is responsible for computing the pixel values for its allocated image partition. The workload at each PE is proportional to the number of pixel of the image plane to be computed.

Image partitioning may require a processor to fetch data items from other processors in order to complete its tasks. The computational complexity of a task will vary from pixel to pixel and thus ensures an even load balance so that it must be possible to migrate some task from those processors allocated complex tasks to those whose initial allocation contained computationally easier ones. Use of early termination may significantly alter the computational effort required to complete task. Volume partitioning is less able to the use of early termination and the reduction in a solution would be less effected as the opacity of the volume increases. Here, increased variations in computational complexity will effect the computation to communication ratios. Consequently the load balancing within a multiprocessor system will become more pronounced as the number processor increases.

4.2 Simulating the N-body Problem

The second case study is also from scientific computing which analyzes that what happens when galaxies collide or how a random collection of stars folds into a defined galactic shape. This simulation would allow us to understand the evaluation of stars in a system of galaxies overtime. The N-body problem involves simulating the motion of a number of bodies moving under forces exerted on each by the others. Computing the forces among bodies is the most expensive part of a time-step ([11], [32]). The forces on each body are computed, and then the position, velocity, and other attributes of each body are updated in each time-step.

A simple method to compute forces is to calculate pair-wise interactions amongst all bodies. This has $O(n^2)$ computational complexity. The distribution of bodies in three-dimensional space is highly irregular so that bodies are denser in some regions and sparser in others. This hierarchical approach implies that bodies in denser regions interact more with other bodies because these bodies may be taken as a single body at the center of denser body areas. Thus, bodies in denser regions have more work associated with them than bodies in sparser regions. These hierarchical algorithms are able to reduce the complexity to O(n log n). This makes it feasible to simulate problems with millions of stars in a reasonable time but only using powerful multiprocessors. A simple partitioning technique can be used for parallelization of the N-body problem so that each processor responsible for computing of each portion of the problem, so each data exchanges between processors is carried in distinct message. Thus, a large number of messages could result. A clever divide-and-conquers approach to the problem uses this clustering bodies idea with the whole space in which one cube contains bodies. Then, the recursive divide-and-conquer algorithm is applied to each cube and sub-cubes until every sub-cube contains one body or less.

This N-body simulation problem has far more irregular and dynamically changing behavior than the volume visualization problem. Unlike volume visualization, which has more regular and predictable structure of computation

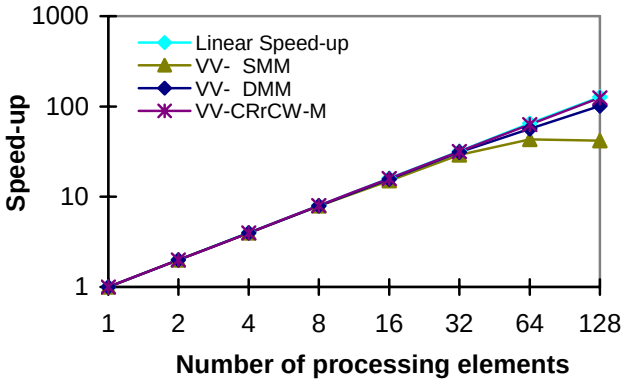
and communication, the hierarchical N-body application presents many challenges for effective parallel computing. The hierarchical algorithm for computing forces on each body is an efficient method for solving the N-body problem in O(n log n) time complexity. The problem is executed in parallel in each time step and a global barrier mechanism synchronizes processors at each time steps.

5. Results

An experimental cycle-by-cycle simulator is implemented on a personal computer by using an object oriented programming language. Experimental parameters (values) are chosen from the references ([3], [8], [10], [11], [18], [34], [44], [50], [57]). Two parallel application case studies with the experimental parameters (values) run on the simulator. First one is the volume visualization and second is N-body problem. Programs for the case studied are developed and applied to the simulator. The CRrCW multiprocessor system and conventional multiprocessor are compared with each other related to the applications. Then, the requirements of a CRrCW multiprocessor system are explored for various situations. Memory multiple access model with sliding caches is tested for changing parameter and examined for best performance. The cluster controlling technique is also employed to enhance the performance of the multiprocessor system. Simple multiple access bus structure and allocation technique is applied and tested for the scientific applications. The simulations results show that a CRrCW multiprocessor system with memory and bus organization, and the right number of busses and memory modules is more scalable than a distributed memory multiprocessor. A CRrCW multiprocessor system with general-purpose powerful processors exploits the local and global parallelism across the system at medium and coarse grain levels.

Several metrics such as speed-up, efficiency and average memory access time are used to measure and compare performance of the systems. Following abbreviations are used on the figures *SMM, DMM, and CRrCW-M stands for Shared Memory Multiprocessor, and Distributed Memory Multiprocessor, and* CRrCW Multiprocessor *respectively. CB, NC, VV and NB stand for Threads, Clusteral Bus, No Cluster, Volume Visualization and N-Body respectively.*

Simulation results show that a CRrCW multiprocessor system provides the best scalability for medium and coarse grain tasks. A CRrCW multiprocessor system with a large number of processors is more efficient than a shared and distributed memory multiprocessor. As it can be seen from the Figure 6, the CRrCW multiprocessor with two threads provides the best speed-up for larger number of processors. Here, the dominant communication in volume visualization is amongst the neighboring processors. The distributed memory multiprocessor system yields speedup, which is quite better than the shared memory multiprocessors system. Even the shared memory multiprocessors with private caches produce satisfactory speedup for volume visualization.



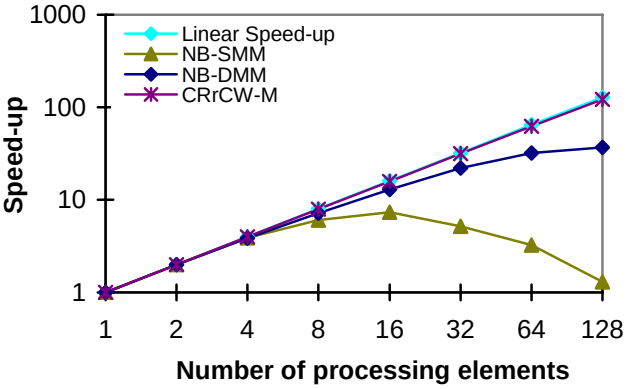


Fig. 6 A comparisons of the multiprocessors models for volume visualization.

Fig. 7 A comparisons of the multiprocessors models for N-body problem.

The speedup results for the multiprocessor systems are illustrated in the Figure 7. Here, CRrCW multiprocessor system with two threads produces the best performance for the N-body application. The shared and distributed memory multiprocessor systems are not scaleable because the communication in N-body is global and many non-regular memory accesses are needed. Thus, the CRrCW multiprocessor system uses the broadcast ability of the architecture to provide the best performance.

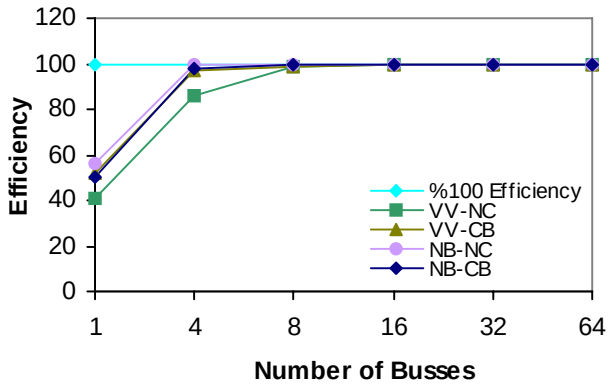


Fig. 8 The multiprocessors system with and without clustering for varying number of busses.

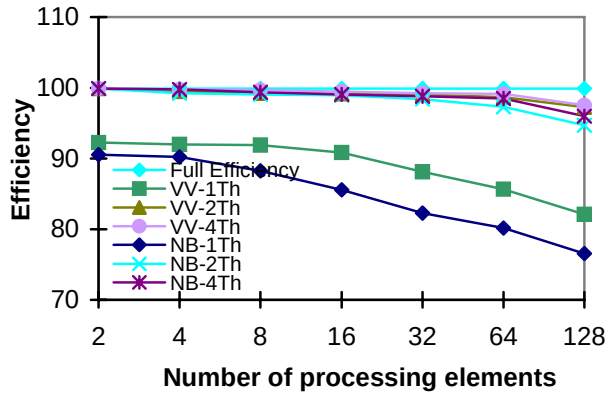


Fig. 9 A comparisons applications on the system with varying number of threads.

Figure 8 shows that how the number of buses on a CRrCW multiprocessor system affects the efficiency of the system when chosen applications run on the simulator. The relation between number of buses and number of processors on the system are shown in the figure for 1, 2, 4, 8, 16, 32, and 64 buses. The numbers of buses needed for an efficient parallel system largely depends on the scale of the system and applications. If an application runs on the parallel system that requires less remote memory accesses, the system with a small number of buses provides a linear speed-up for the application. If the relationship between the number of processors and the number of buses on the system are considered, a large-scale parallel system with enough number of buses provides high scalability. The CRrCW multiprocessor system with 16 processors, two threads and 3 buses provide over 90% efficiency.

Here, the Network Folder is used to create a ring of busses in order to enhance the clustering and multicast ability of the network. The use of the interrupt and synchronization buses coordinates synchronization and communication between processors. Here, the volume visualization exploits the clusteral usage of the busses because communication in parallel volume visualization tends neighboring processors ([17], [18], [31], [42], [54]). Thus, more local communication implies that more clusters can be formed on a bus and used simultaneously. On the other hand, the clustering technique does not help to improve the performance of the N-body application. Even, small performance degradation can be observed. Because, the clusteral busses technique uses extra bits to determine length of the cluster originating from the central processor at the cluster center. Therefore, this extra information affects the performance of the network when network suffering from high level of the communication demand**.**

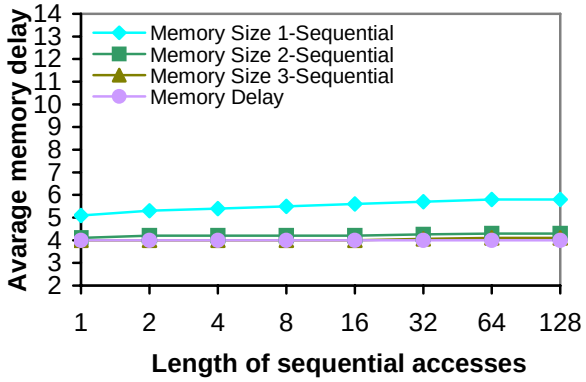


Fig. 10 Changing number of memory modules.

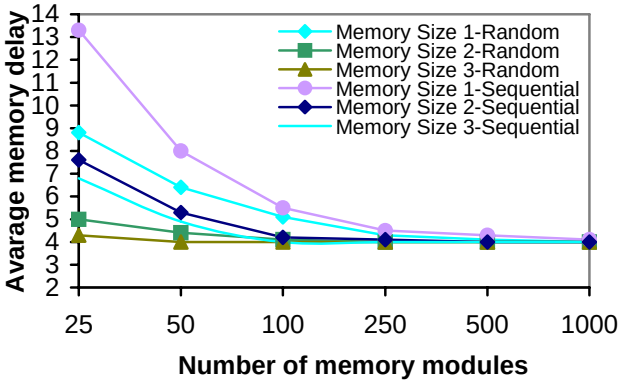


Fig. 11 Changing size of each sequential access.

Results shown on the Figure 9 are obtained for the multiprocessor system with one to four threads. Even two threads may improve the performance of the system dramatically. More than four threads do not provide further improvement over the performance. This is depended on the scale and communications activities of the system and application run on the system. The memory model with sliding caches is also simulated on the personal

computer. The benefits of the sliding caches memory model can be seen in Figure 10 and 11. The average memory access times obtained using various number of memory modules is shown in Figure 10. For lower number of memory modules more cache modules are needed to reduce average memory delay. As can be seen in the figure, when a large number of memory modules are used, the need of the sliding caches is decreased dramatically. Figure 11 illustrates the effect of increasing length of sequential memory accesses. The length of sequential fetches has a little effect on average memory delay so that doubling number of cache modules is enough for the best performance. These last two graphs show that sliding cache memory provides the best performance for the CRrCW multiprocessors. Finally it can be said that memory size and cost of the CRrCW multiprocessor system are independent of scale of the parallel computers.

5. Conclusion and future work

In this paper a CRrCW multiprocessor system have been proposed, examined and compared with other multiprocessor models. This multiprocessors system with the intelligent and multiple access memory organization, and network organization provides better performance than all other multiprocessor systems. This memory organization for the CRrCW multiprocessor system adapts itself according to the demand of the application and uses concurrent read and restricted write for the best performance. The dynamic network organization also allows exploiting full bandwidth of the network without regarding the communication demand pattern of the application. The clusteral busses enhance the performance of the system. On the other hand, all processors on the system share a common memory space so programming on the system is easier than those of on a distributed memory multiprocessor systems. An efficient CRrCW multiprocessor system can be designed by using general-purpose powerful processors. This makes this system scalable parallel computers architecture. Hence, large-scale parallel computers with general-purpose processors may provide wide range of multi-user and multi-processing ability for many parallel-computing applications. As a final word, simulation results show that CRrCW multiprocessor architecture is potential candidate for general-purpose parallel processing.

In this study, a CRrCW multiprocessor system have been proposed, examined and compared with other multiprocessor models. There are several future works that has to be done. One of these works is that the performance of the CRrCW multiprocessor system may be examined on other scientific applications such as ray tracing. A real implementation of the multiple access memory models with sliding caches would be considered as an important future work. For a typical parallel computing applications bus access algorithms should be tested for the best performance. New bus topologies and technologies may also be examined for the best performance. The performance of the clusteral bus controlling technique may be explored for large range of scientific applications.

Acknowledgments

I would like to thank the Karadeniz Technical University, in Trabzon, Turkey for supporting this research.

References

[1] A. Basu, A classification of parallel processing systems, International Conference on Computer Design, 21-32, 1984.
[2] Adve S. V., designing memory consistency models for shared memory multiprocessors, Ph.D. diss., Department of Computer Science, University of Wisconsin-Madison, Technical Report, 1993.
[3] Adve S. V. and Gharachorloo K. Shared Memory Consistency Model. IEEE Computer 29(12):66-76.
[4] A. G. Chalmers and J. Tidmus, Practical parallel processing: an introduction to problem solving in Parallel, (International Thomson Computer Press, 1996).
[5] A. G. Chalmers and D. J. Paddon, Communication efficient MIMD configurations, In 4th SIAM Conference on Parallel Processing For Scientific Computing, Chicago, 1989.
[6] A. L. Decegam. The Technology of parallel processing: Parallel Processing Architecture and VLSI Design. Prentice Hall International Inc, 1989.
[7] Alexey Lastovetsky, Adaptive parallel computing on heterogeneous networks with mpC, Parallel Computing, Volume 28, Issue 10, Pages 1369-1407, 2002.
[8] Andrea Clematis and Angelo Corana, Modeling performance of heterogeneous parallel computing systems, Parallel Computing, Volume 25, Issue 9, Pages 1131-1145, 1999.
[9] Anton H. J. Koning, Karel J. Zuiderveld and Max A. Viergever, Volume visualization on shared memory architectures, Parallel Computing, Volume 23, Issue 7, Pages 915-925, 1997.
[10] Archibald J. and. Barer J. L. Cache coherence protocols: Evaluations Using a Multiprocessor simulation model. ACM Transaction on Computer System 4(+):273-298, 1986.
[11] Barry Wilkinson and Michael Allen, Parallel Programming: Techniques and Applications using networked workstations and parallel computers, Prentice Hall, 1999.
[12] Bisiani R. and Ravishankar M. PLUS: A distributed shared-memory system. Proceeding 17th International Symposium on Computer Architecture pp.115-124. 1990.
[13] Cemal Köse, A scaleable semi-shared memory multiprocessors architecture with sliding caches, Proceeding of the IASTED International Conference; Parallel and Distributed Computing and Networks, Insbruck-Austria. Pp: 415-420, 2004.

[14] Cemal Köse, A semi-shared Memory architecture for scaleable multiprocessor, International Journal on Signal Processing, ICSP 2003.
[15] Cemal Köse, Multi-threaded parallel ray tracing for fast and realistic rendering, High performance computing and data processing symposium 2002, pp:1-8, Gebze, 2002.
[16] Cemal Köse, Multi-threading for efficient parallel volume visualization, The Chambers of Electrical-Electronic and Computer Engineers. 8. National Congress. 2001.
[17] Cemal Köse and Alan Chalmers, Clusteral models for efficient parallel volume visualization, Parallel and Distributed Computing Practices, Vol 3, Number 3, NOVA Publisher, 2000.
[18] Cemal Köse, Parallel volume visualization, (Ph.D. thesis, University of Bristol, Department of Computer Science, 1997.
[19] Cemal Köse and Alan Chalmers, Profiling for efficient parallel volume visualization, Parallel Computing special issues on applications parallel graphics and visualization, ELSEVIER, 1997.
[20] Cemal Köse and A. G. Chalmers, Memory Management for parallel volume rendering, 19th World Occam and Transputer User Group Meting, P. 113-126, March 1996.
[21] Cray XT3 System Overview, CRAYDOC : S-2423-13, Pittsburgh Supercomputing Center, 2005.
[22] David E. Culler, Jaswinder Pal Singh and Anoop Gupta, Parallel Computer Architecture; A hardware/software approach, Morgan Kaufmann Publisher, Inc., 1999.
[23] E. Barton, J Cownie, and et al., Message passing on Meiko CS-2, Parallel computing, I20, 497-507, 1994.
[24] E. Gallizzi, A deadlock-free communication system for a Transputer network, 4th North American Transputer Users Group, p. 11-18, 1990.
[25] E. Barton, J Cownie, and et al., Message passing on Meiko CS-2. Parallel computing, 20(4):497-507. 1994.
[26] Hagersten E., Landin A., and Haridi S, DDM Cache Only memory architecture, IEEE Computer 25(9):44-55, 1992.
[27] Hey A. J. G. The genesis distributed memory benchmarks. Parallel Computing 17:1111-1130., 1991.
[28] Jorge Buenabad-Chavez, Virtual memory on data diffusion architectures, Ph.D. thesis, University of Bristol, Department of Computer Science, 1998.
[29] Jorge Buenabad-Chávez, Henk L. Muller, Paul W. A. Stallard and David H. D. Warren, Virtual memory on data diffusion architectures, Parallel Computing, Volume 29, Issue 8, Pages 1021-1052, 2003.
[30] J. Challinger, Parallel volume rendering on a shared-memory multiprocessors, Computer and Information Science, 1991.
[31] John D. Watts, Parallel algorithms for coupled-cluster methods, Parallel Computing, Volume 26, Issues 7-8, Pages 857-867, 2000.
[32] John Hennessy and David Patterson, Computer Architecture: a quantitative approach, Morgan Kaufman Publishers Inc., 1990.
[33] Jörg Wensch and Ben Sommeijer, Parallel simulation of axon growth in the nervous system, Parallel Computing, Volume 30, Issue 2, Pages 163-186, 2004.
[34] Karl Solchenbach, Grid applications on distributed memory architectures: Implementation and evaluation, Parallel Computing, Volume 7, Issue 3, Pages 341-356, 1988.
[35] K. Batcher, Design of a massively parallel processors, IEEE Transactions on Computers, I29: 836-840, 1980.
[36] K. Donovan, Performance of shared memory in a parallel computer, IEEE Transactions on Parallel and Distributed Systems, I2(2), 253-256, 1991.
[37] Leonadis I Kontothanassis and Michael L Scott, Memory Model, Department of Computer Science, University of Rochester, 1994.
[38] Lipton R. and Sandberg J. PRAM: Scaleable shared memory. Technical Report. Computer Science Department, Princeton University. 1988.
[39] Michael Florian and Michel Gendreau, Applications of parallel computing in transportation, Parallel Computing, Volume 27, Issue 12, Pages 1521-1522, 2001.
[40] Michael J. Quinn, Parallel Computing; theory and practice, McGrawHill, Inc., 1994.
[41] M. J. Flynn. Some Computer Organization and their Effectiveness. IEEE Transactions on Computer Graphics, 21(9):948-960, 1972.
[42] Mohammed Atiquzzaman, Pradip K. Srimani, Parallel computing on clusters of workstations, Parallel Computing, Volume 26, Issues 2-3, Pages 175-177, 2000.
[43] O. A. McBrayn, An overview of message passing environments, Parallel Computing, 20:418-444. 1994.
[44] Paul W. A. Stallard, Henk L. Muller, and David H. D. Warren. Performance Evaluation of Parallel Programs on the Data Diffusion Machines. In Performance Evaluation of Parallel Systems, PEPS '93, pages 94-101, UK, 1993.
[45] P. W. Liu, and et al., Distributed computing: new power for scientific visualizations, IEEE Computer Graphics and Application, 16(3), 42-51, 1996.
[46] Pittsburgh Supercomputing Center, http://www.psc.edu/general/hardware.html, 2005.
[47] R. B. Mueller-Thuns, D.G. Saab, R.F Damiano and J. A. Abraham, Benchmarking parallel processing platforms: an applications perspective, IEEE Transactions on Parallel and Distributed Systems, I4(8), 42-51, 1993.
[48] Rhys Francis and Ian Mathieson, Synchronized execution on shared memory multiprocessors, Parallel Computing, Volume 8, Issues 1-3, Pages 165-175, 1988.
[49] S. A. Green and D. Paddon, A non-shared Memory multiprocessor architecture for large database problems, Proceedings of the IFIPS WG 10.3 working Conference on Parallel processing, Pissa. 1988.
[50] S. Bandini, G. Mauri and R. Serra, Cellular automata: From a theoretical parallel computational model to its application to complex systems, Parallel Computing, Volume 27, Issue 5, Pages 539-553, 2001.
[51] Technical Summary (KSR, KSR1), Kendall Square Research Corporation, 1992.
[52] SGI (Silicon Graphics, Inc), http://www.sgi.com/products/servers/altix/memory_performance.html, 2005.
[53] Thomas Branul. Parallel programming an introduction, Prentice Hall International, 1993.

[54] Thomas W. Crockett, An introduction to parallel rendering, Parallel Computing, Volume 23, Issue 7, Pages 819-844, 1997.
[55] V. Chaudhary and J. K. Aggarwal, A generalized scheme for mapping parallel algorithms, IEEE Transactions on Parallel and Distributed Systems, I4(3), 328-346, 1993.
[56] Umakishore Ramachandran, Gautam Shah, S. Ravikumar and Jeyakumar Muthukumarasamy, Scalability study of the KSR-1, Parallel Computing, Volume 22, Issue 5, Pages 739-759, 1996.
[57] U. Neumann, Communication cost for parallel volume rendering algorithms, IEEE Computer Graphics and Applications, I2, 49-58, 1994.

Cemal Köse was born in Trabzon, Turkey. He received the B.Sc. and M.Sc. degrees in Electrical and Electronics Engineering from Karadeniz Technical University (KTU) in 1986 and 1990, respectively. He received the Ph.D. degree from The University of Bristol in 1997 in computer science. Currently, he is an Assistant Professor in the Department of Computer Engineering at KTU. His major research interests are in parallel computers, pattern recognition and natural language processing.

Özcan Özyurt was born in Trabzon, Turkey. He received the B.Sc. and M.Sc. degrees in Department of Computer Engineering from Karadeniz Technical University (KTU) in 2001 and 2006, respectively. He is currently Ph.D. student at the same department. He is also a lecturer in Beşikdüzü Vocational High School at KTU. His major research interests are in parallel computers and natural language processing.