Using Symmetric Multiprocessor Architectures for High Performance Computing Environments

Mohsan Tanveer
Dept. of SE, Foundation University, Institute of Engineering and Management Sciences (FUIEMS), Rawalpindi, Pakistan

M. Aqeel Iqbal
Dept. of SE, Foundation University, Institute of Engineering and Management Sciences (FUIEMS), Rawalpindi, Pakistan

Farooque Azam
Dept. of CE, College of Electrical & Mechanical Engineering, National University of Sciences and Technology (NUST), Islamabad, Pakistan

ABSTRACT

Performance enhancement for high speed computing can be carried out by using many techniques and architectures at software and high hardware level. Performance enhancement using hardware techniques may include the use of multiple computing nodes or a single node consisting of multiple processors. Symmetric multiprocessor is one of the modern architectures used to perform extensive computations. Symmetric multiprocessors have many configuration modes to carry out these heavy computations. The performance of Symmetric multiprocessors is analyzed and compared with high-fidelity models. Processors models are used to design and construct the architectures of symmetric multiprocessors. In this research paper such kind of critical design aspects of symmetric multi processors have been analyzed for further enhancement of the existing technology.

1. SYMMETRIC MULTI-PROCESSORS

The demand for the processing power unit is growing day by day. The capability of execution according to speed and efficiency can be increased by different type of ways like enhancing the CPU programming like inserting new programming, arrange new registers to the model of microprocessors and grouping up the CPUs [9]. Chip improvements are required for the first two options but the third can boldly increase the processing power. However, the “CPU grouping” approach is affordable because:

– If we enhance the CPU programming, more efforts would be required to integrate the programs and registers.
– If one processor is faulty, the life of the computer would be increased by multi processors. Hence we have to design a new commercial scale super computer.

Hence, we have a choice, to rely on internal changes of the CPU or we combine multiple processors/CPUs. Symmetric multiprocessing is a case of parallel multiprocessing [7], [8]. In the symmetric multiprocessing system all processors behave identically and Kernel of operating system can assign any process to any processor. A Single instance of the operating system manages all processor s. Applications have uniform access to memory and I/O. These operating systems are more special and complex unlike typical operating systems.

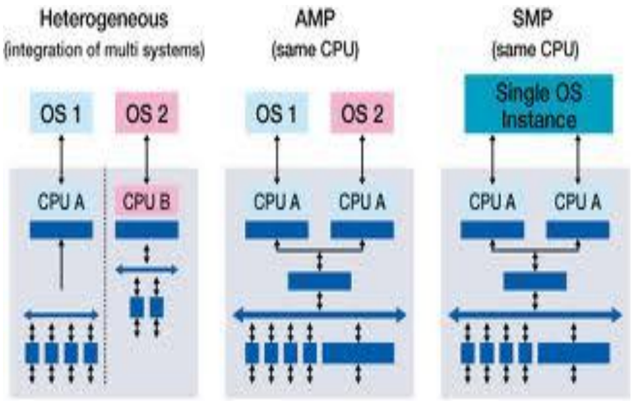


Figure-1: Heterogeneous, Asymmetric Multi-processing (AMP), Symmetric Multi-processing (SMP)

To gain the maximum advantages of symmetric multiprocessing, we required an additional synchronization code for data structures to maintain the consistency and balance the work load between multiple threads of multiple processors [8], [9]. On a multiprocessor, scheduling is multi dimensional. The scheduler allocates processes to the CPUs to execute it. This complicates the processing paths and signals of multiprocessors. Thus efficient multiprogramming is required to avail the full and maximum processing. Symmetric processors have their own front side bus that’s why they have the advantage over cores. The scalability of symmetric multiprocessors can be increased by using ***mesh architecture.*** SMP is one of the earliest types of computer architecture mostly used for up to 8 processors.

These multiprocessors share a common main memory and I/O. A microcontroller(s) controls data flow throughout the processors and main memory [6]. Each processor has a dedicated cache for better latency and data brought into each processor’s registers can be transferred through its cache rather than from main memory. The question arises here that may be a process on data can be cached by multiple processors. To avoid this incidence there is a task called cache coherence that ensures each processor is working on recent copy of data. The basic architecture we use for coherence is snoopy bus architecture (discussed later).

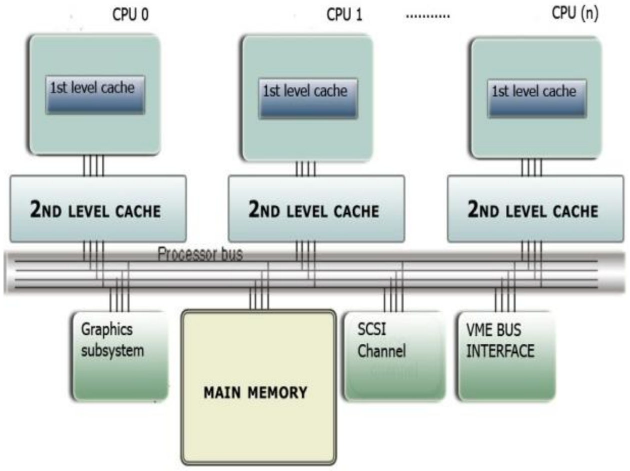


Figure-2: Basic Architecture of SMP

1.1 SMP Clusters

A computer cluster is a team of linked computers forming fast local networks. Clusters are usually used to increase the speed and performance over single computer. They are cost effective and available to high performance computing. They are operated on having redundant CPU nodes [5], [6]. The capability to control more clients by giving more jobs and data access is done by scaling server side processor. We can have cluster of shared memory computers like:

Each node has its own local memory, and Nodes share data by passing data over the network. Client computers bonded to clusters of SMP server has given the computing power of ***Divide and Conquer Algorithms***. Clusters provide the scaling of I/O, processors, and storage but not of client management methods, or security. Grid's computing domain is scaling. Grid computing provides services like client management or security. With some careful analysis on SMP nodes and cluster architecture, we can scale these systems precisely and with very limited waste of resources. Also the communication between cluster nodes is much grater than that of between multiprocessing in SMP.

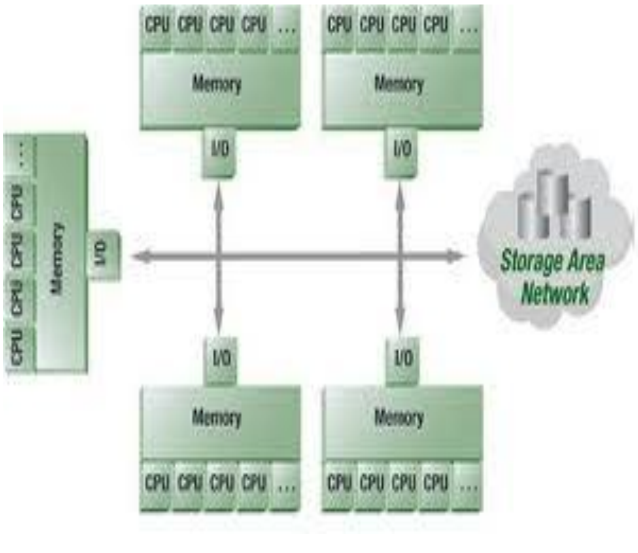


Figure-3: BSMP Cloud

2. SOFTWARES CHARACTERISTICS FOR SMP

As discussed earlier, in multiprocessors all processors use the same order of rules just like in a single processor system. The thread which comes out of this rule of processing, are globally analyzed and reordered with respect to each other in the shared area. The thread visits the processors as multi thread. So multi programming is required to produce multi threads. If an operating system is not partitioning the threads in multi-way multiprocessors then it is better to use a uni-processor instead. In fact it could be a worse situation, because it may suffer more locking overhead and process delays when dispatched to other processors, it may be slower.

There are different ways to achieve parallel threads execution of a single program:

1. Make parallel calls to library subroutines to create parallel multi threads that can run at a same instance.
2. Execute the program with a parallelizing compiler. It will help us to detect threads that are not dependent on other thread that is to be executed on second instance and generate a parallel multi threaded parsing code.
3. Use a multi threaded software.

The maximum improvement can be analyzed and achieved by a rule that is called ***Amdahl's Law:*** It says increase of speed can be achieved by a formula equals to uni-processor time divided with the sum of sequence time and time of multi-processor. For example:

Sequential time = 50% and Parallel time= 50%

Time improvement = less than a factor of 2 (= 1.6 in 4-way µp).

2.1 Operating System Scheduling

A SMP unit has a similar view of the memory; any task has the capability of running on any processor. But in fact, it is not a correct way to let a task wandering between different processors to be processed [3]. When a thread migrates to a processor, the data which is currently present in first cache of that processor also has to be move towards the other processor. So each processor can have its copy and can be replaced again on the memory. This is handled by the cache coherency. If we introduce the processor property to each task and bound that task to execute on the same processor, this method is known as processor affinity. But this method is not suitable for locking because for example a task is running on that processor and suddenly blocked.

The task which is already waiting in a queue and ready for execution would suffer extra time [3], [4]. Therefore modern SMPs can assign any task to any processor. In Windows NT there are no separate schedulers. A thread produces events. These events are handed over to the event handler modules of the windows kernel. Events Like creating a new task, task asleep, blocking of tasks on synchronization and task terminator. Windows scheduling is totally based on the time quantum mechanism. Each task has a time period. It is a time in which operating system checks the task priorities. To do task switching there is time tick period. The tick is custom set to 10 ms for uni-processors, and 14 ms for SMP. On single tick, time is decreased by 3. When time period value reaches zero, task will be put away and recalled again soon.

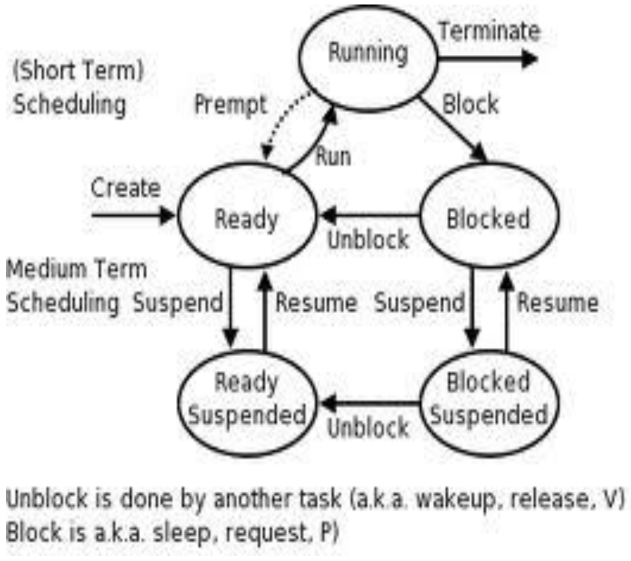


Figure-4: Seven States Model for OS Processes/Threads

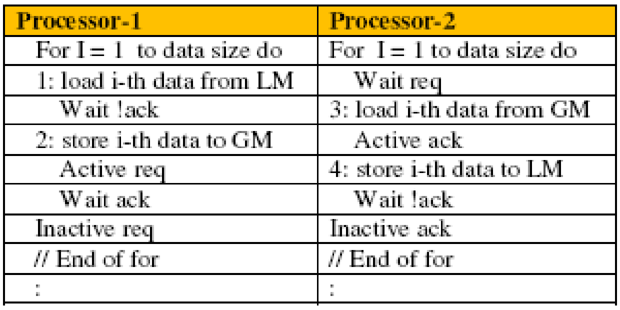
Normally, Windows NT has "32" priorities. "0" priority is for idle process. Windows may keep on changing priority to avoid starvation, indefinite postponement deadlock etc. The OS kernel maintains 1 queue ready for each thread priority. There is a bit mask of (32 bit) which tells which task is ready to be performed and if its idle it tells scheduler that processes are idle. If none of any processor is found idle, the scheduler will preempt the lower priority task on interrupt. Every processor has assigned each task and the last processor on which it was executed is saved.

One another way can be that we do process switching but the fact is process switching costs more then thread switching. So it's better to divide the threads and allocate to the multiple processor.

2.2 Locks:

A uni-processor blocks a task. While in any Operating system executing parallel codes, there is a need of locking technique. Lock is used to ensure that no other task is executing outside a limiting point. The purpose of lock is to grant that task which is waiting for the lock permission to carry on. Locks provide a way to for process communication and synchronization. Locking concept is used to prevent other processes access to incomplete data. Interrupt disabling is not the solution to prevent data modification by another processor in SMP case.

Therefore locking mechanism controls data between the multiple processors. There is a lock variable which has to be free to acquire processor by writing some value to it. After the first processor, another processor is able to read and write the lock variable. Thus lock is free for the both processors. This lock is applied to tasks and ISR and can be applied to the cache-line of the processors. If we apply lock to Cache, no unrelated bus traffic disturbance is expected. The locked cache will hit the bus until another processor will need this for operations like Exchange, Compare and Addition etc.



Processor-1	Processor-2
For I = 1 to data size do	For I = 1 to data size do
1: load i-th data from LM	Wait req
Wait !ack	3: load i-th data from GM
2: store i-th data to GM	Active ack
Active req	4: store i-th data to LM
Wait ack	Wait !ack
Inactive req	Inactive ack
// End of for	// End of for
:	:

Table-1: Efficient Polling Protocol

Lock Qualities

A programmer must know how many locks should be created. For example if spin lock is created:

- A spinlock must not be recursive, as the processor would be continuously spinning on the lock with no one to release the lock.
- Too much poll will affect bandwidth and too low will delay. So remain balanced.
- The existence of multiple locks makes a deadlock possible.
- More then needed locks effects throughput for example in case of mutual-exclusion locks, with 9 instances of a program running in parallel. 9 instances would not be synchronized effectively to avoid waiting for other process.

3. CACHE COHERENCY

In symmetric multiprocessors every processor has its own cache, so the obvious possibility is every cache has the same copy of data to be executed. If more then two threads modify the same data, it concludes with no data coherency [1]. The solution is to invalidate other copies of data except one, by broadcasting on the shared bus. Invalidation is performed by cache controller hardware [2]. Cache controller hardware watches flow of data over the bus. This method is known as snooping protocol. Directory based coherence protocol is another model [11]' [13].

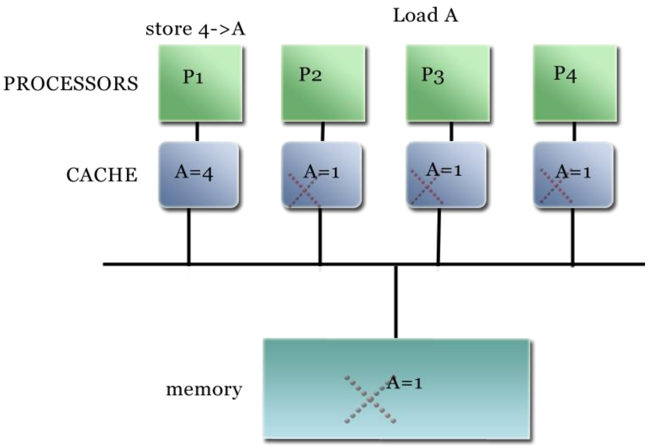


Figure-5: Cache Coherency in Multi-processor Architectures

After invalidation, there are two methods to update main memory [14]:

Write through: In this method the controller updates the memory as soon as possible for the other processors after writing the data.

Write back: In this method the controller doesn't updates the memory cell unless another thread comes and demand for that cell. If one processor has demanded the same data in the memory, it is better to retrieve it from the cache of other processor. Main memory will take more time to recognize the recall.

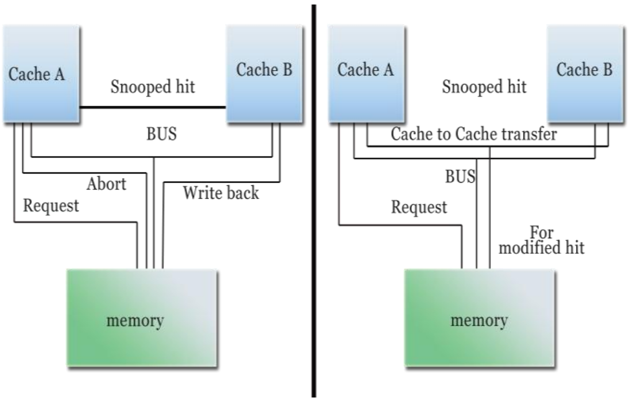


Figure-6: Cache Coherency Solutions

There are three states of cache blocks of Snoopy protocol; Shared (block is ready to fetch and read), Exclusive (block is ready to write and there are no other copies of it), Invalid (block has no data) [12].These states are implemented when CPU demands for any cache block.

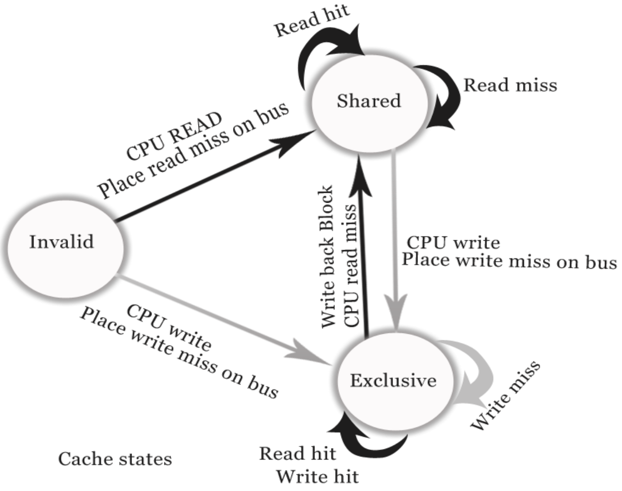


Figure-7: Snoopy Protocol [10]

3.1 MESI Protocol

The snoopy protocol model is ideal for on chip supported caches. But in most of the small scale SMP's MESI protocol model has been implemented. There are four states of MESI protocol (in fig).INVALID, SHARED, MODIFIED & EXCLUSIVE.

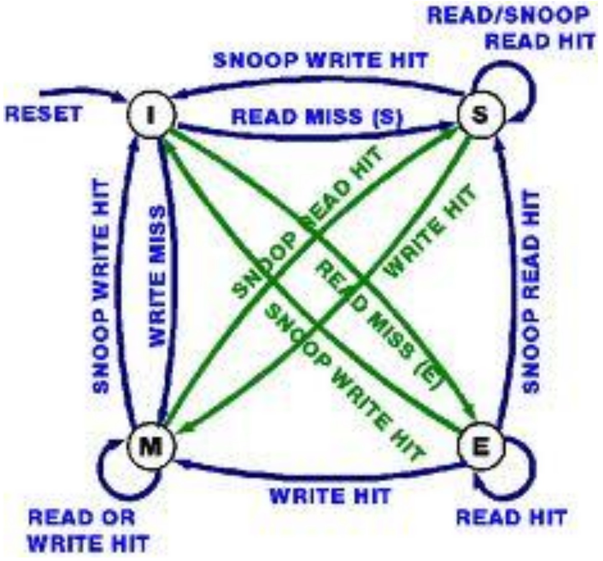


Figure-8: MESI Protocol

For example cache has been hit and sent from modified state to shared state. Now address has been shared in both the caches. The process is modified and arrived towards the cache which one was requesting for it. On the other side cache which has the modified data can refuse to share, writing it back to the main memory and then requester can get data from the main memory. Read and write are not enough, we have to add some more to increase the performance efficiency of coherency model. The processors address bus must be available to the controller so that tags of the addresses can be matched and state of invalidation can be performed.

3.2 Token Coherency Protocol:

The message passing technique was difficult in direct connections so a new protocol model was designed for direct interconnections CPUs and as well as for switched based interconnections in 2003. Token coherency technique uses counting and exchange of tokens simply. Each block is mapped with fixed number of tokens. Processor should have all the tokens in order to write a block but to read a block at least 1 token is required [15]. Encoding bits of tokens is done by the formula Log$_2$N (N = no. of tokens).

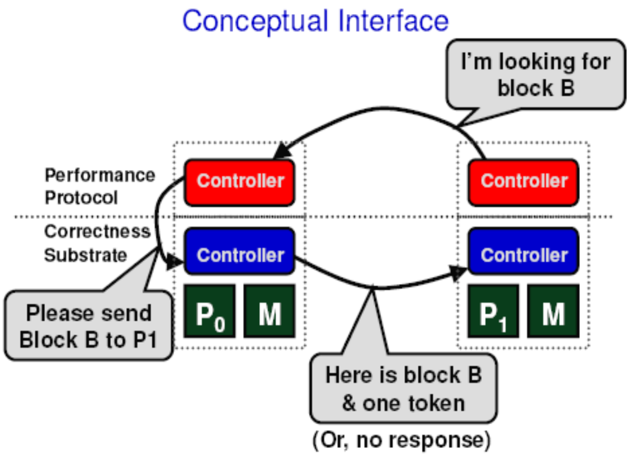


Figure-9: Basic Concept of Token Protocol:

In this protocol processors can predict and ask other processor for the required token if it has. Token coherence model can perform (18-25%) faster than the snoopy protocol. Counting of tokens gives safety to coherence invariant (single writer and multiple readers). If a processor failed to acquire data, a timeout message will be sent to requestor, and a persistent request will be sent. Request persists until it is satisfied and deactivated upon completion. It reduces starvation as well. Token snooping is more efficient in direct processors interconnection. The graph given below is between indirect interconnected SMP on Normalized runtimes.

3.3 Advanced Programmable Interrupt Controller (APIC)

Interrupts are controlled through APIC (Advanced Programmable Interrupt Controller) unit. ICC (interrupt controller communication) will be a pathway to control and communicate multiple I/O APIC units collectively.

3.4 Scalability

Scalability is the performance of Multiprocessors. As expected, adding more processors should increase the overall performance accordingly, for example two processors should increase the performance twice as compared to one processor. But in actual as adding processors increase the scalability so it reduces some as well. That is due to:

1. Cache coherency pipelining.
2. Time taken by number of cycles by spin lock.
3. Synchronization conflict.

The fact is, if one processor provides 1 speed. Two processors provides 1.75 with a increase of 0.85 and eight processors will provide (5.2 - 5.5) (see Amdahl's law) the speed and performance gradually decreases but not a big deal as compared to uni-processors.

Thus scalability and performance can be increased by increasing memory band-width, shortening the latency of memory access time (may be we should design a new memory scheduling techniques and algorithms), and by reducing the gap between memory so that starvation could be less possible.

The Threads received by SMP is distributed to all processors equally. If we run Windows NT on a single CPU, as we know that threads are distributed by multi threaded operating system. The result of parallel threads can be shown by this graph.

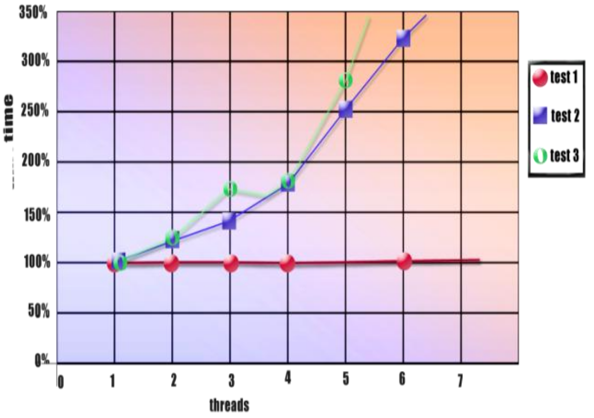


Figure-10: Threads Increment Behavior Running On Single CPU

4. SMP NETWORK ARCHITECTURE

4.1 NUMA Architecture:

SMP is a basic form of UMA (uniform memory access) architecture. The interconnection of SMP with another SMP through Interconnection network(s) and switches forming the clusters are known as NUMA (non uniform memory access). UMA is best for not more then 8 processors due to scalability issue. But NUMA or (cache coherence NUMA) makes it preferable due to its scalability for more than 8 processors, because every unit of processors have their own local physical memory which is easy to access but logically there is one address shared space.

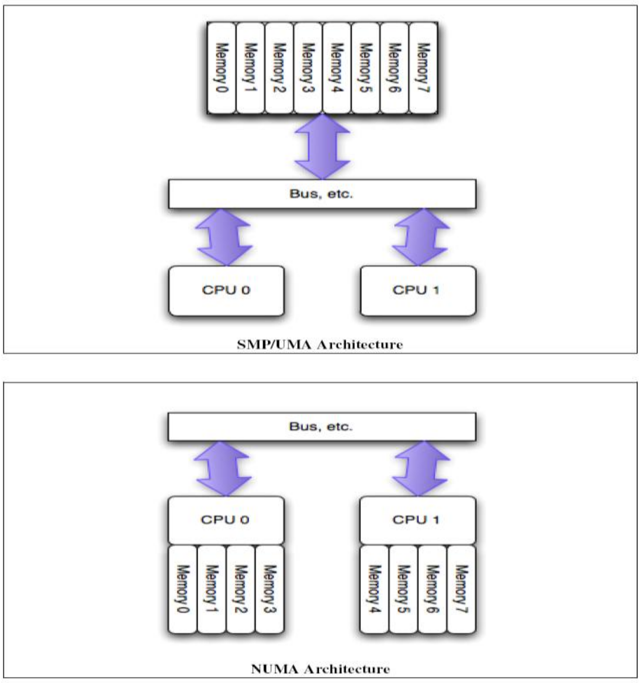


Figure-11: UMA vs. NUMA Architectures

Users prefer NUMA on multicomputer architecture because they believe that programming is easy and due to non required extra library of the compilers. The time required to access data depends upon its location, whether it is present in local memory or may be residing over remote memory. A single image of operating system runs all over the network. If one processor will modify any data the other processors will also update data into their cache (for cc-NUMA machines). Logical address space contains pages; these pages have some states which are passed to track the position.

STAGES:

1. "NO- PRESENCE: they are in remote memory
2. SHARED: copies are distributed to local memories
3. EXCLUSIVE: In local memory"

The latency for accessing data in comparison of both local memory as well as non local memory is calculated by NUMA Factor. For example data access from the neighbor node is faster then the node which is present on a distant level. For NUMA factor we have to architect different inter-connection design for the nodes. For example: Indirect fat tree, 2D torus, 4D hyper cube, Hierarchical Inter connection, Omega network, Ring, cross bar... etc.

4.2 COMA Architecture:

COMA (Cache Only Memory Architecture) is made for large SMP networks just like NUMA. In this structure memory are replaced by cache memory or we can say acting like cache (attraction memory). Their addresses are hashed to DRAM cache lines. Data is readable at any of modulo at any single instance and is moved by hardware. This ability of making copies, proved this structure extremely time effective. If the OS algorithms are poor, COMA compensates it, but it requires separate memory boards along with coherence interconnection memory board.

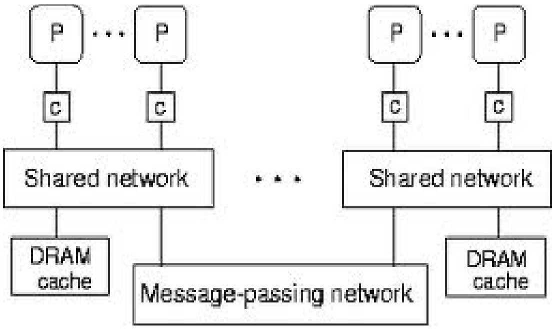


Figure-12: Cache Only Memory Architecture (COMA)

5. FUTURE WORK

In research paper, as mentioned, there are some facts due to which the scalability of SMP decreases. These facts include the issues like synchronization and the pipelining. So we have to re-design a protocol and algorithms which can be used to produce maximum performance for every processor used in SMP. Re-designing the interconnection scheme and by utilizing the most suitable material for interconnection can also resolve performance issue to some extent.

1. Multithreaded and hyper threaded programming might be a suitable benchmark to research for increasing the speed and performance of SMP.
2. We ought to design such network architectures and maps as to keep memories close to the processors and interconnection switches.
3. As size of microprocessor is decreasing, the ability of handling data and low latency in SMP can be the platform for producing SMP based cell phones in the commercial market. These cell phones will act dual.
4. Combining GPU processors and SMP architectures on experimental bases and performance can lead us to a real time high embedded super computing machine.
5. The clouds of SMP can lead to highly vast global scale network.

6. CONCLUSION

Since last few decades, the distributed computing has evolved dramatically to fulfill the emerging requirements of computational extensive applications. There are different ways to implement distributed computing environments in which most prominent solution is to use symmetric multiprocessors architecture for achieving high performance distributed platforms. Symmetric multiprocessor architectures have an extensive capability to manage multiple real time threads for active application. The current architectural scenarios being adopted for the design of symmetric multiprocessor architectures are not demonstrating the enough computational power and hence are good candidates for further research. Different issues including synchronization, pipelining, protocols, hyper threading, coupling of GPU and SMP and dealing with SMP clouds, have been pointed out for further consideration and research.

7. REFERENCES

[1] Hung. Cache Coherency for Symmetric Multiprocessor Systems on Programmable Chips. M.A.Sc. Thesis, University of Waterloo, Waterloo, August 2004.

[2] A. Hung, W. Bishop, and A. Kennings. Enabling Cache Coherency for N-Way SMP Systems on Programmable Chips. In Proceedings of the 2004 Intl. Conference on Engineering of Reconfigurable Systems and Algorithms,Las Vegas, Nevada, June 2004.

[3] W. Stallings. Operating Systems (6th ed.): Internals and Design Principles. Prentice-Hall, Inc. UpperSaddle River, NJ, USA, 2008.

[4] Simon Kågström: Performance and Implementation Complexity in Multiprocessor Operating System Kernels. Blekinge Institute of Technology, 2005.

[5] Serveurs Architectures: Multiprocessors, Cluster Parallel Systems, Web Servers, Storage Solution René J. Chevance,2004

[6] B. Senouci, A. M. Kouadri M, F. Rousseau, F. Petrot Multi-CPU/FPGA Platform Based Heterogeneous Multiprocessor Prototyping: New Challenges for Embedded Software Designers The 19th IEEE/IFIP International Symposium on Rapid System Prototyping, 2008. RSP '08

[7] John P. Shen & Mikko Lipasti. Modern Processor Design: Fundamentals of Superscalar Processors. McGraw-Hill 2002.

[8] The von Neumann Architecture (http://www.csupomona.edu/~hnriley/www/VonN.html).

[9] Sahoo, D., J. Jain, S. K. Iyer, D. L. Dill and E. A. Emerson, Multi-threaded reachability, 2005.

[10] Martin, M.M.K., Sorin, D.J., Hill, M.D., and Wood, D.A.: 'Bandwidth adaptive snooping'. Proc. 8th Int. Symp. on High-performance Computer Architecture, Anaheim, CA, February 2002.

[11] J. Tuck, L. Ceze, and J. Torrellas. Scalable Cache Miss Handling for High Memory-Level Parallelism. In Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, Dec. 2006.

[12] M. M. K. Martin. Formal Verification and its Impact on the Snooping versus Directory Protocol. In International Conference on Computer Design. IEEE, Oct. 2005.

[13] S. Kim, D. Chandra, and Y. Solihin. Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture. In Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, Sept. 2004.

[14] S. V. Adve and K. Gharachorloo. Shared Memory Consistency Models: A Tutorial. IEEE Computer, 29(12):66–76, Dec. 1996.

[15] R. Fernandez-Pascual, J. M. Garcia, M. E. Acacio, and J. Duato. A Low Overhead Fault Tolerant Coherence Protocol for CMP Architectures. In Proceedings of the Thirteenth IEEE Symposium on High-Performance Computer Architecture, Feb. 2007