A Basic Comparison of Different Memories for Object Oriented Computer Architecture

Ankit V. Patel and Paresh Mathur

U. V. Patel College of Engineering,
Ganpat University, Mehesana, Gujarat, India
{ankit.themark,paresh.2047}@gmail.com

Abstract. This position paper is focused on the abstract model of Object Oriented Computer Architecture and specifically the memory unit of such a computing system which would be able to handle the type of object oriented computing proposed in the paper. The memory unit plays a very crucial part because it will have to be radically different from what we call a memory unit in the present contexts. This parallel nature of computing in an OOCA allows it to use different memory architectures both as shared memory and distributed memory. Here we compare the pros and cons of using these types of memories in an OOCA.

Keywords: Computer Architecture, Object oriented models, parallel systems, Memory Organization.

1 Motivation

Even though there have been many advances in the computer hardware technology (better memory units, better control units, faster and more reliable processors, etc basically putting more transistors on one chip) there is a large divide between the way we think about hardware and software. We need to think outside of this system if we want to exploit the technological progress in semiconductor technology or VLSI design. Object oriented programming has revolutionized the world of programming from its inception in 1960s in the Simula language. The concepts like encapsulation, extensibility, modularity and abstraction are very powerful tools which may be used in the context of hardware to remove some of the bottle necks. But this would require a revolutionary change in the way we think about computation and processors.

2 Related Research

A remarkable amount of research has been done in the field of alternative computer architectures. By 1980's Myers [1] had correctly identified the need for resolving the semantic gap between the computer architectures and the programming languages. The Object Oriented Languages like Java and Ruby offer platform independent software development and also facilities like readability of code & higher

N. Meghanathan et al. (Eds.): CCSIT 2012, Part II, LNICST 85, pp. 435–443, 2012.

abstractions. The Exokernel research at MIT [2] focuses on increasing the performance of applications without using any commercially available Operating System. They emphasizes on treating the hardware as the operating system, thus eliminating the need of at least one level of the hierarchy. Also the bare PC research at Utah [3] will help developers write applications for application specific PC's again eliminating the need of an operating system.

The Bare Machine Computing research at Towson University, lead by Dr. Ramesh K. Karne [4] focuses on making application specific computing machines which run without any software installed on them. He has implemented applications like webmail, Secure VOIP, TLS protocol on a bare PC [5, 6, and 7].

Our previous position paper [8] tried to describe, in a very abstract way, how the memory model of an OOCA would work.

3 Small Introduction to OOCA Memory Model

The OOCA has to use a special kind of memory which is radically different from our current concepts. The memory unit in OOCA does not just store data but stores whole "objects". An object has two parts, one is the data member and the other is the function member. The data members are the processed results and parameters of the function members. Thus the memory also has some processing power in it to regularly keep updating the data members in accordance with the function members.

An OOCA memory is not just a *data* storage unit, but an *object* storage unit (the difference being the processing power). An abstract model of Object memory is given in the figure 1.

As seen in the figure, the Object Memory consists of three main parts.

Storage Unit: To store data members as well as function members.
Processing Unit: To handle memory management and function processing.
Interfaces: To interact with object processing unit.

3.1 Memory Unit

This is used to store the data as well as the function members. The data memory and function memory need not be separate, and they can be part of a large RAM memory. But it would be more efficient to have the data memory in a rewritable memory like a RAM or hard disk as it is dynamic. While on the other hand as the function memory is static for a given object it could be stored in a ROM. It is also not necessary to have a large data memory separate for each Object.

A much more efficient way would be to have a conventional memory unit like RAM or Hard Disk and a pointer storage unit as the data memory. The data memory can have pointers to the memory location where the data is stored in the hard disk. This will use the resources much more efficiently. Something similar can be done with the function memory. It can hold pointers to the location where the function program is stored in the memory.

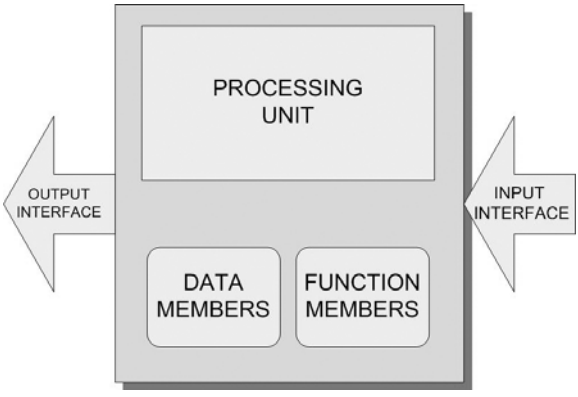

Fig. 1. An Abstract Model of Object Memory

3.2 Processing Unit

The processing unit is essentially an ALU combined with a control unit. The processing unit updates the data members by executing the functions defined inside the object. Apart from processing the function members, it also has to handle the interfaces. The object is accessed through a set of input and output interfaces. The commands passed to the input interface has to be processed to first determine the type of command (is it to pass an argument or to fetch some data or execute a function), then accordingly the control unit of the processing unit will execute the relevant function.

3.3 Interface

The object stored in the object storage unit interacts with the object processing unit or other objects by the means of interfaces. These interfaces are predefined and implemented in hardware. There are two types of interfaces

1) Input: Used to pass arguments to the object function members.
2) Output: Used to fetch data from the object.

The input interface is used to give arguments to the function members. Again the arguments are passed as pointers. But unlike the normal computers, these pointers will have two parts.

1) The object address, to tell the object from which we want to fetch data.
2) The data member address, to know which data member to fetch from a given object.

The input interface will also handle the queries that are done on the object. To fetch a data member or the executed result of the function member, the object processing unit will have to call the input interface with a predefined command (and pass necessary arguments) and then accept the result from the output interface.

In an object oriented language, there is no evident difference between the syntax of fetching a data member or a function member. In the same way in object oriented

computer architecture, the output interface will have no distinction between fetching a data member or a function member. This will encapsulate all the details of the object execution behind that interface. In an object oriented programming language there is a function defined which fetches the data members through an interface, like wise we can have the function members store their results inside the data memory and then a similar interface can be there to fetch a data or the result of the function member.

4 Options for Memories That Can Be Used in OOCA

As stated previously OOCA has a highly parallel architecture. Any real world implementation of OOCA would require more than one object with each object executing its functions in parallel. Thus we have to choose between the different topographies of parallel computing systems. The two most broad and basic categories for parallel memories are

1) **Shared Memory:** In this type of memory all the processors can access the memory as a global address space. In other words, there is one large storage location and all the processors can access any part of that memory at any given instant of time.
2) **Distributed Memory:** In this type of memory all the processors have their local address space and require a communication protocol to transfer data from one processor memory to another.

These two are the broad options that we have for our OOCA memory topology. As is evident there are advantages and disadvantages of each type of memory. Apart from common cons like fetch time or maintaining the state concurrency, there are some other pros and cons which are specific to OOCA memories.

An OOCA memory is supposed to facilitate the implementation of the concepts of an Object at hardware level. Each of the above said topology help in implementation of one or more concepts of Object orientation by their inherent nature. Other concepts can be added with a little modification to the topology.

5 Using Shared Memory in OOCA

In a shared memory, there is a huge storage space which is available as a global address space to all the processors. Thus all the objects will share the same data memory (and also memory to store functions). This type of memory allows all the objects to access the data or function members of every other object in the system. This ease in access and efficient use of memory is a trade off with the protection of data from accidental or intentional misuse by the user. As all the objects can access all the parts of global memory all the time, there may be buffer overflows or other ways of corruption of memory.

5.1 Using Uniform Memory Access Shared Memory

The block diagram of a uniform Memory Access Shared Memory in a 4 object OOCA is given in figure 2 below.

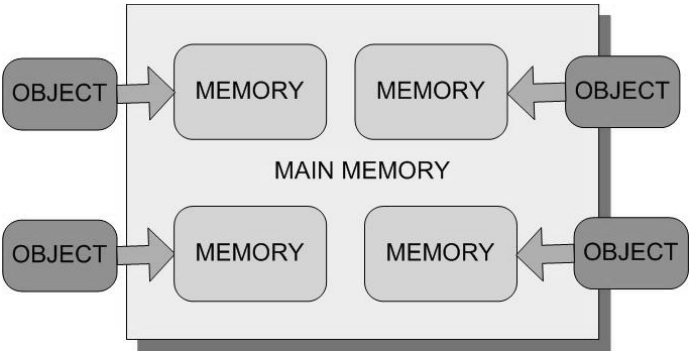

Fig. 2. Four object OOCA system with UMA shared memory

As can be seen in the figure, the main memory is divided into chunks of memories which serve as the data and function storage units for the four objects OOCA system. In this type of systems, each object will not have an actual dedicated memory as there storage unit. But the storage unit will comprise of a memory unit which will store pointers to the location of actual data in the big main memory. This approach allows dynamic memory and facilitates the implementation of a garbage collection unit.

Some characteristics of this type of topology are:

Efficient Use of Memory: As can be seen, with the UMA approach the main memory can be used very efficiently. While adding new objects, the new object can be assigned memory much more efficiently. With a memory control and garbage collection unit, one may as well give only that much memory to a given object that it is actually using.

Standard Interfaces with Memory: As the interface with the memory can be of only one type, all the objects will have to have an implementation of that standard interface in them. As is inherent in the OOCA model, each object will have a very different processing unit. This is because the processing unit has to process different kind of data for different types of objects. Thus the standard interfaces have to be implemented in every object.

Standard Fetch Time for data from other objects: As all the data is stored in a global address space, the fetch time for getting data of one object is exactly same as getting data from other object.

Load sharing of processing power may be done between objects: As all the memory is in the global address space, the value of data members and even the code of function members can be accessed by any processor in the system. Also fetch time of accessing memory of any object is exactly the same. Thus, if urgently required, we may bend the rules and the processing power of one object may be made accessible to another object. This type of arrangement would make judicious use of all the processing power available in the system.

5.2 Using Non Uniform Memory Access Shared Memory

The block diagram of a Non Uniform Memory Access Shared Memory in a 8 object OOCA is given in figure 3 below.

In a NUMA topology, there are two or more UMA shared memories connected with a bus. The given object may access its own memory, or fetch data from any other memory via this standard bus. In this topology the bus protocol has to be standard and implemented on top of all the objects.

It has the following characteristics:

Different types of processors in the same system: In a UMA topology, the most preferable and ideal condition is when all the processing units of all the objects are either exactly the same or very similar to each other. This is required because we have to access the same memory for all the objects. But when you use a NUMA structure, you may have lots of different types of processors which talk with their custom memory. This helps in implementing any optimization techniques one might want for faster memory access. But the only constrain here is that the processors must also be able to communicate over the bus protocol so it can transfer data to and from its memory.

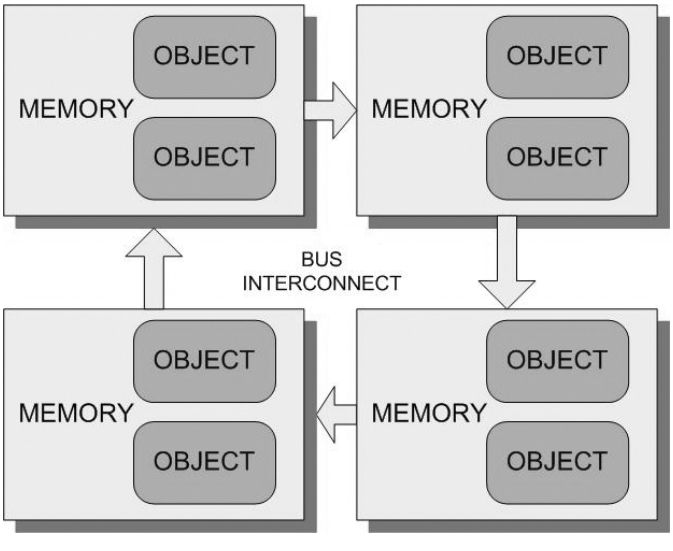

Fig. 3. Eight Object OOCA system with NUMA shared Memory

Higher Fetch Times for Fetching Memories of other Objects: For a UMA all the memories are defined in a globally mapped memory unit. Thus the fetch time to access the data of other objects is exactly the same as that of access data of the same object. But in NUMA the data of other object has to be fetched over the bus protocol which would make it slightly slower than the fetch time of accessing data from the same object.

It facilitates the implementation of Inheritance Principle of OOP: When one object inherits function and data members of another object it has said to have "inherited" from that object. From the block diagram of NUMA we can see how two objects can be grouped to use a single memory. This type of architecture helps in implementing an inheritance mechanism inside the architecture itself. Similar objects can be grouped together and inherit functions from other objects so as to implement the inheritance concept.

New objects can be easily added: One thing that is most exciting about a NUMA is the ease with which you can add new objects. If you want to add a new object to a UMA system, we have to first analyze the memory which is left in the system which poses a certain constrain over expansion of any given system. But with a NUMA, you just have to add it to the bus interconnection (which usually has a ring topology). But care has to be taken to reroute the fetch requests in such a way so as not to disturb the previous system.

6 Using Distributed Memory in OOCA

The block diagram of OOCA with distributed memory is given in figure 4 below.

In a distributed memory, a complete object (processing unit and storage unit) is one entity. These are connected via a bidirectional bus over which the objects can communicate.

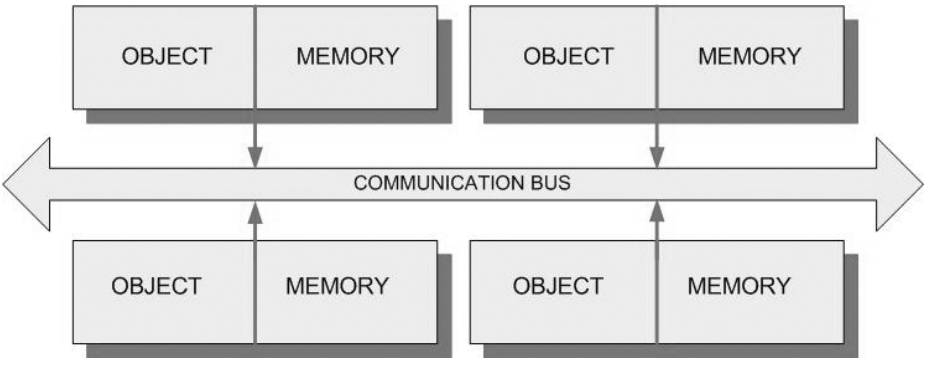

Fig. 4. Four object OOCA system with Distributed memory

Some characteristics of this type of topology are:

Scalability: This type of topology can be scaled infinitely in theory. We can add arbitrarily many objects to the bus. But of course the bus protocol and bus control unit that is used in the system will limit the amount of objects that may be added to the given bus.

High Level Of Abstraction Can Be Achieved: As all the processing and function members of the object are not visible to other objects, it would help in achieving a

high level of abstraction. If some object wants to access the data member or function member of any other object, it just has to give out a call on the bus and wait for the reply.

7 Conclusion

As in any other engineering problem, deciding the optimal type of parallel memory for OOCA is a question of tradeoffs. Using any particular type of memory for all the architectures would not be the right way to go about designing an OOCA computer. For deciding which type of memory topology to use, we must first have a clear idea of what the system would comprise of and what would be its future expansions.

Having said that, some rules of thumb may be established for deciding the type of memory that should be used. For example, if the system is small (say less than 10 object of low complexity) one may use UMA shared memory architecture. It is most suitable as it is easy to implement and also easy to manage.

But for a larger system, one has to use a NUMA or go for a distributed memory architecture altogether. These architectures allow for high complexity and high number of objects in a system. Also expanding the system by adding few more objects is a lot easier in these topologies as we have to just connect them to a bus and then the bus communication protocol will facilitate the rest.

References

[1] Myers, G.J.: Advanced Computer Architecture, p. 17. John Wiley & Sons (1982)
[2] Ford, B., Back, G., Benson, G., Lepreau, J., Lin, A., Shivers, O.: The Flux OSKit: A Substrate for OS and Language Research. In: Proc. of the 16nth ACM Symp. on Operating Systems Principles, St. Malo, France, pp. 38–41 (October 1997)
[3] Engler, D.R.: The Exokernel Operating System Architecture, Ph.D. Thesis, MIT (October 1998)
[4] Karne, R.K.: Application-oriented Object Architecture: A Revolutionary Approach. In: 6th International Conference, HPC Asia 2002 (December 2002)
[5] Khaksari, G.H., Wijesinha, A.L., Karne, R.: Secure VoIP using a Bare PC. In: 3rd International Conference on New Technologies, Mobility and Security, NTMS (2009)
[6] Emdadi, A., Karne, R.K., Wijesinha, A.L.: Implementing the TLS Protocol on a Bare PC. In: The 2nd International Conference on Computer Research and Development, ICCRD 2010, Kaula Lumpur, Malaysia (May 2010)
[7] Patrick, A., Wijesinha, A.L., Karne, R.: The Design and Performance of a Bare PC Webmail Server. In: The 12th IEEE International Conference on High Performance Computing and Communications, AHPCC 2010, Melbourne, Australia, September 1-3, pp. 521–526 (2010)
[8] Mathur, P., Patel, A.V.: A memory Model of Object Oriented Computer Architecture. In: The International Conference on Intelligent Systems and Data Processing
[9] Mancl, D., Havanas, W.: A study of the impact of C++ on software maintenance. In: Proceedings Conference on Software Maintenance (1990)

[10] Cockshott, P.: Performance evaluation of the Rekursiv object oriented computer. In: Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences, vol. i (1992)
[11] Harrison, A., Moulding, M.R.: Data fusion on the Rekursiv object-oriented architecture. In: IEE Colloquium on Principles and Applications of Data Fusion, pp. 3/1–3/3 (1991)
[12] Duncan, R.: A survey of parallel computer architectures. Computer 23(2), 5–16 (1990)