Heterogeneous Coherent virtual Machine (HCVM)

By
Tamer Elsharnouby
CMSC818k High Performance Computing

1 Introduction

CVM is originally implemented to support various systems. But they are applicable only to homogeneous systems. The main motivations to extend this system to support heterogenous system are:

- *Heterogeneous systems are common.* Taking the Junkfood lab as an example, there are available 2 different types of machines. So, making the application applicable only to certain type of systems results in loosing the power of the other workstations.
- *High performance computing*. Again, referring to the previous example, running the application on the whole available systems may result in much more processor power. This supports the idea of high performance computing.

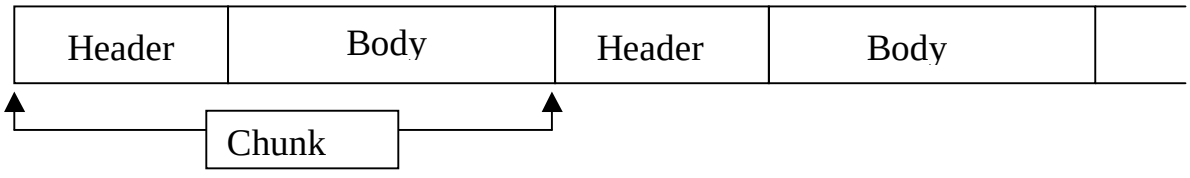
This introduces two different problems, which are:

- Communication.
  - Communication establishment between various machines.
  - Message Understanding. Communication bodies must be interpreted correctly on different machines.
- Memory Handling
  - Ensure that data can be interpreted correctly.
  - Data is dealt with correctly.

Actually, both communication part and data part share the way of dealing with data. The idea is to transform all data to unique format then interpreting this format back to each host.

2 Message Understanding

Messages are the means of communications between various instances of the same application. We must ensure that all types of data are understood using different hosts.



Each message consists of several chunks of data. Each chunk has:

- Header
  - Type of Message (what type of data is available)
  - Size of the following body (in bytes).

- Body
  - Data to be shipped.

2.1 Message Body

There are different types of messages in the CVM system. They are listed in the following table.

Communication Messages	Barrier Messages
Memory Protocol Messages	Lock Messages
Reduction Messages	Statistics Messages
Page Transfer Messages	User Messages

All the messages, mentioned in the previous table, are modified, in order to be interpreted on different machines.

2.1.1 Communication Messages

By this, we mean messages used to handle initial communication between processes (first thing was transferring port numbers). Next, the communication functions **cvm_accept**, and **cvm_connect** plus the message handler in file **comm.c**. Also, by communication messages, we mean the header of the message itself, which is used in sending, resending, forwarding, and replying messages. All these are mainly available in **msg.c**.

2.1.2 Memory Protocol Messages

We modified only "Lazy Multiple Writers" protocol. This is considered in prot_lmw.c and prot_lmw_page.c. The functions modified are: adding, and reading ***diffs***. A lot of other functions are modified in this stage.

2.1.3 Reduction Messages

Mainly, we changed the functions regarding reading and writing reductions in file reduction.c, e,g.; **add_reductions** and **read_reductions**. Nothing else could be changed.

2.1.4 Page Transfer Messages

This concerns with moving a whole page. It is available in Page::fault in page.c. (This was one of the main errors in my system. I discovered it very late.)

2.1.5 Barrier Messages

Barriers are used in synchronization. This part is one of the most important parts in the system. **barrier_arrive** in barrier.c is modified as it is one mainly used. Other functions places in the protocol files are modified.

2.1.6 Lock Messages

This handles mutual exclusion access of memory. It is available in lock.c plus scattered functions in protocol files.

2.1.7 Statistics Messages

The collected statistics have different representations in different system. I decided to change this one, just to collect statistics to present it to this report.

2.1.8 User Messages

I did not touch this one, as I do not know what the user intentions are. This is available in interface.c

3 Dealing with data

Here we will introduce how to deal with different types of data.

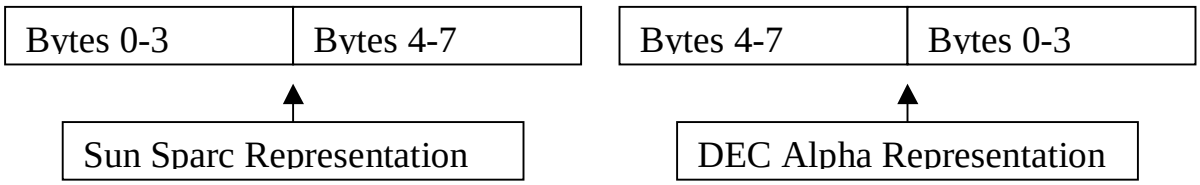
3.1 Integers

For short integers (**short** and **unsigned short**), we use **ntohs()** and **htons()** functions available in C compiler. This transforms network to host and vice versa. Normal integers (**int**, **unsigned int**) are transformed using **ntohl()** and **htonl()**. **Long int** are mapped to normal integers and then send to the other hosts. This is due to the different size of representations (some machines use 8 bytes long, and others use 4 bytes long).

3.2 Float

We have two types, which are **float** and **double**. In case of float, we just deal with it in the same way of integers. My previous error was in dealing with it by copying the data directly without any change.

For double, it is a little bit more complex.

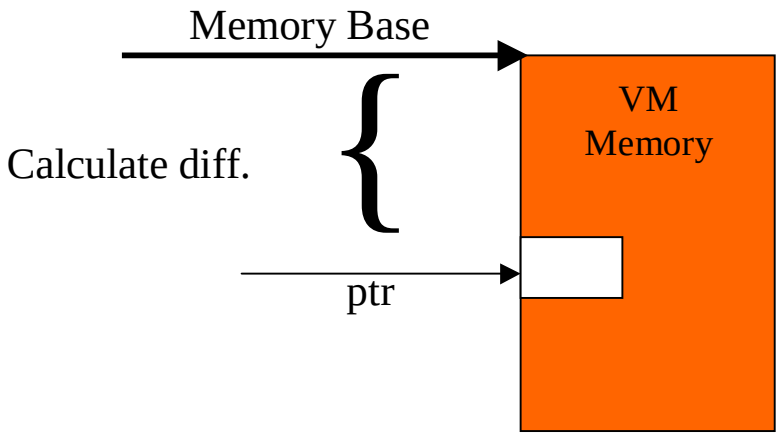


So, the idea is to swap the two words in case of DEC Alpha, while no change in case of Sun Sparc.

3.3 Pointers

All the pointers in the CVM shared memory must point to a place within this shared memory. In memory manager (**class MemoryManager**), it allocates number of consecutive pages determined by variable **npages** and rooted at the variable **base**.

The main idea is to ship the difference between the value of any pointer and the value of the memory base pointer. This value is aggregated on the memory base of the other host to get the correct value of the pointer. See the following figure.



Finally, nothing to be done with characters. They are shipped as they are.

4 Memory Handling

Local memory of each host has to be managed correctly to ensure that the whole application is running with the same version of data. This involves allocation of equal **size** of data on all hosts. All data are handled in the same way mentioned in previous section.

4.1 Where to handle data

Mainly there are two places:

- Page handler (Page fault functions in memory coherency protocols plus those available in page.c, applying of ***diffs***)
- Reductions

4.2 How to handle this data

You have to check the data types using the registered. When allocating data, we must know the structure of this data, so we can react according to these data types. Registering is done manually according to the application. I did not have time to make it automatic. But it is not a big deal, just add extra parameters to **cvm_alloc**(). Once structure is known, it can be handled.

5 Performance Testing

In these performance tests, I tried to demonstrate the effect of running applications on workstations of different types. I have chosen 4 Sun Sparc workstations and 4 Dec alpha workstations. Two types of applications were chosen fast fourir transform (FFT) and travel salesman problem (TSP). Both are available within CVM program. TSP was chosen as a representative of the integer application, and FFT was selected as a representative of floating point application.

All the applications ran on 4 different workstations. The whole permutations were tried (4 Sun machines and 0 Dec alpha, 3 Sun machines and 1 DEC alpha, ….., 0 Sun machines and 4 DEC alpha). I tried that to show the performance effect of running single application using all possible combinations.

My main **performance metrics** are:

- CVM time (time spent in CVM system)
- Elapsed time (time of running the whole application)

All other metrics is irrelevant to our work. Actually, they should remain constant.

Figure 1.a shows the CVM time results. The graph clarifies that the overhead of heterogeneity is really small. In figure 1.a, the difference between CVM 4*0 (4 SPARC and 0 DEC) and the HCVM 4*0 is really small. We can say the same thing about CVM 0*4 and HCVM 0*4. We included the MAX Overhead in Figure 1.b as it changes due to this curve and I do not know why. The Elapsed Time is almost the same as CVM Time with difference +0.01 sec. This is because of the application communication pattern, which consumes most of the time. We ran TSP on the 8 different workstations and we got CVM time of 35.9 seconds.

Figure 2 shows the same performance metrics of running FFT on 4 different workstations. The results were not reasonable enough as (it is impossible 2*2 is the sane 3*1 for example). Also, the CVM took longer time than HCVM, which is correct. The main excuse is that I do not have any control on the load on these workstations. I ran these applications in the morning where the load varies a lot.

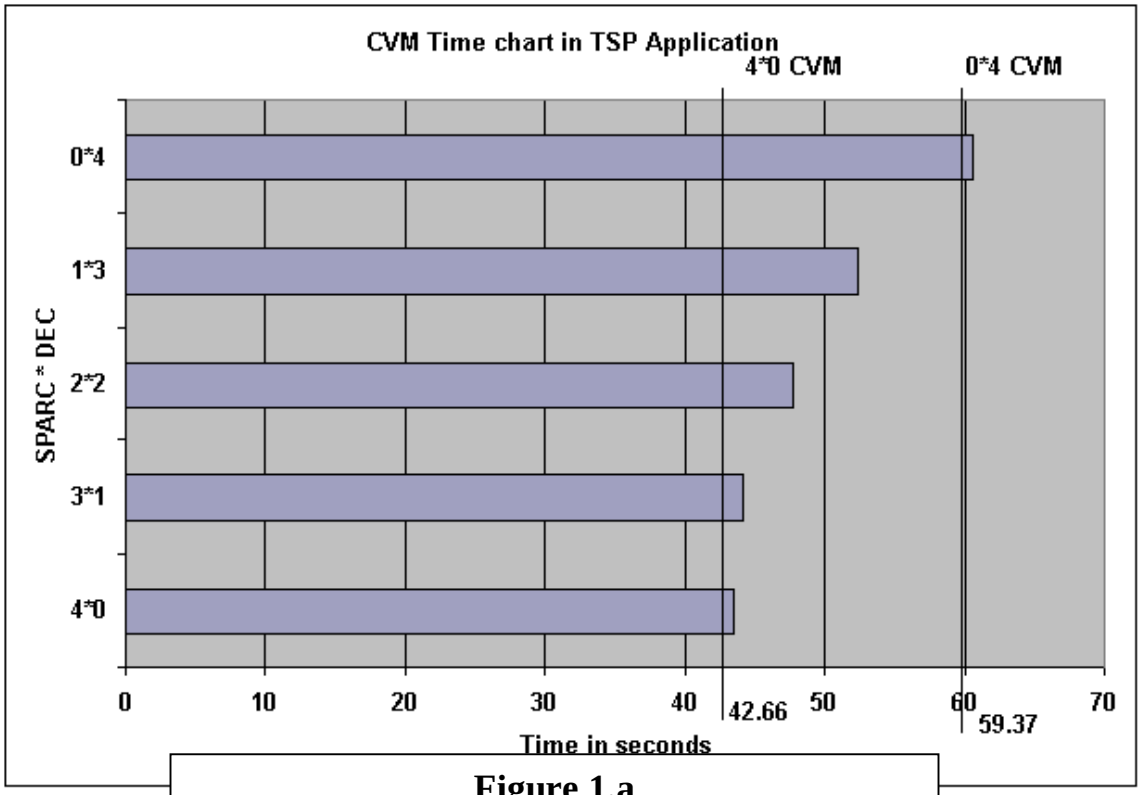


Figure 1.a

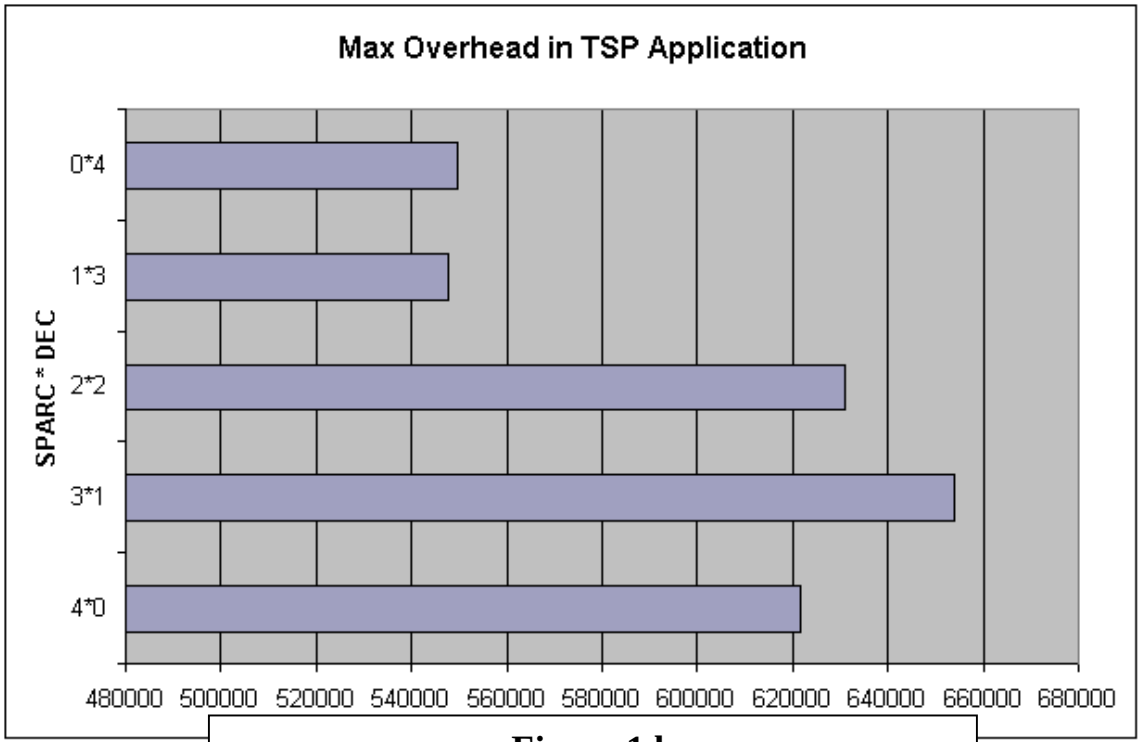


Figure 1.b

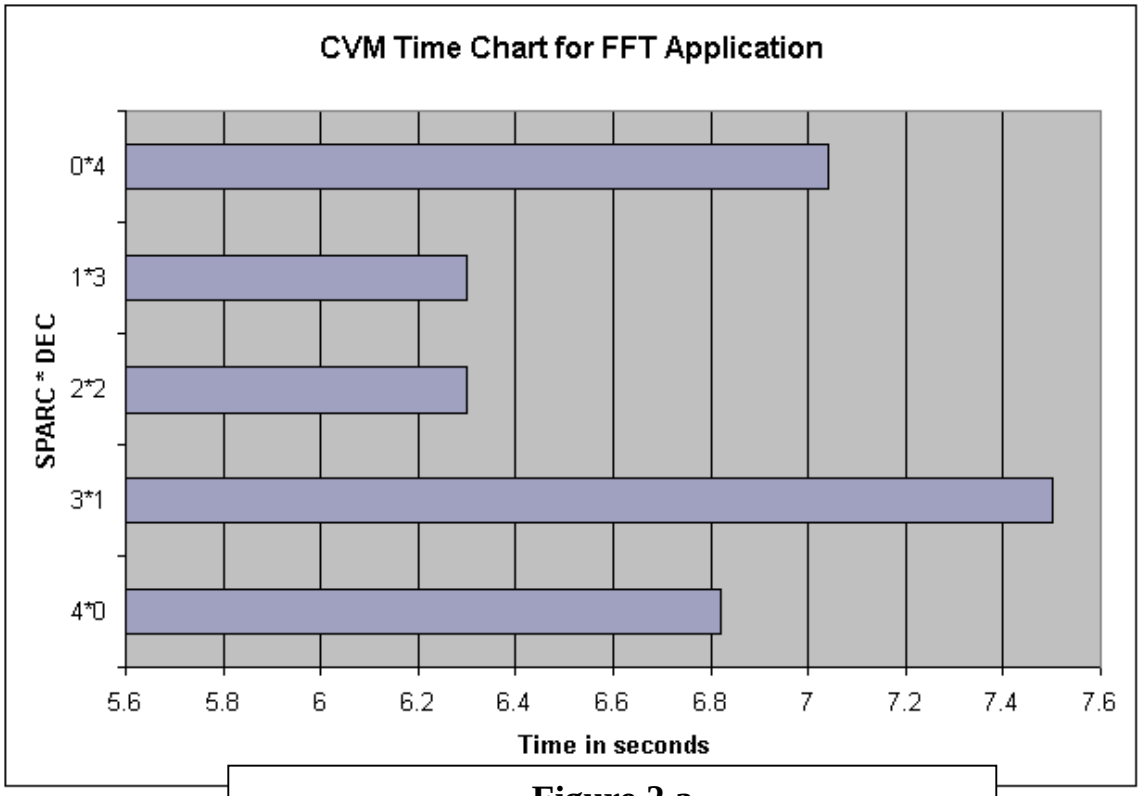


Figure 2.a

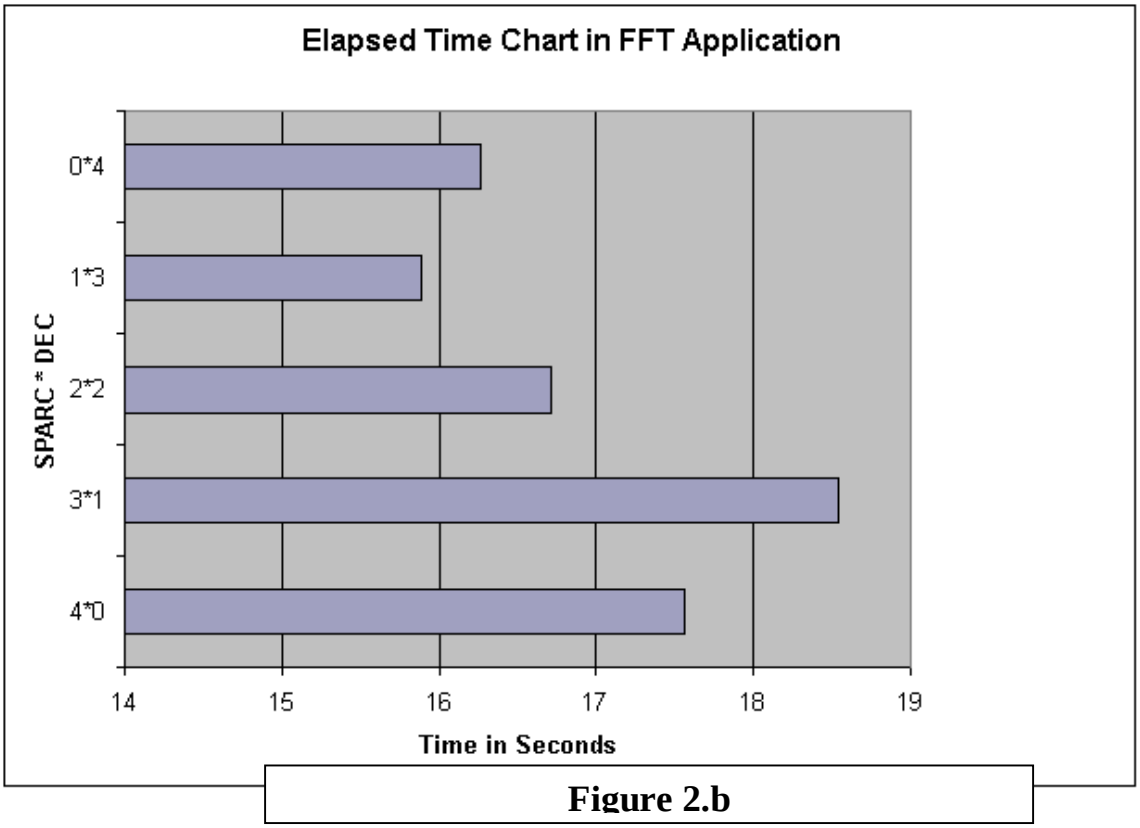


Figure 2.b

6 Limitations

6.1 Systems

The original homogeneous CVM supports different types of systems Sun Sparc running SunOS 5.5, DEC Alpha workstations running OSF4.0, IBM SP-2 running AIX4.1 using MPI, and Linux on PCs. I did not have the time to try on all these systems. My goal is to run on two systems at least. I tried the two available to me, which are:

1. Sun Sparc workstations running SunOS 5.0 and SunOS 5.5
2. Dec Alpha workstations running OSF 4.0

6.2 Consistency Protocols

The original systems supports multiple systems like lazy multipe writers update protocol, and tape protocol. Heterogeneous CVM supports lazy multiple writers update protocols. The idea of modifying the other protocols to be applicable to Heterogeneous CVM is straightforward.

6.3 Data Types

In homogenous system, it supports all types of data. HCVM supports all types of data, and all of them are working till now, except double. I discussed this before.

6.4 Operations

You can not increment pointer to data of different size. For example, you cannot say

```
{
        long * ptr = X;  /* X is an array of long integers allocated
                                using cvm_alloc() */
        …..
        ptr++;
}
```

The main idea is that ptr will be incremented in different ways on different machines.

7 Conclusions

- Run applications on heterogeneous workstations of two types. The idea may be extended to the rest of the systems supported by CVM. We may extend that in the future.
- Modifying of single memory consistency protocol, which is “Lazy Multiple Writers” protocol. Other protocols can be modified in a similar way.
- The performance gained by HCVM is good. The overhead of keeping track of data is not big. Using larger amount of systems my greatly increase the performance. For example, using all the 8 available machines is much better than using any 4 homogeneous machines.
- Better performance can be achieved by doing much more optimizations.
- Learning how a real system application runs. I worked on a different other application, but this is the first time to be exposed to a distributed system.
- Learning new techniques like copying pages, interrupt handling for messages.