Hybrid Model Based Testing Tool Architecture for Exascale Computing System

Muhammad Usman Ashraf *m.usmanashraf@yahoo.com*
Faculty of Information and Computer Technology
Department of Computer Science
King Abdulaziz University
Jeddah, 21577, Saudi Arabia

Fathy Elbouraey Eassa *fathy55@yahoo.com*
Faculty of Information and Computer Technology
Department of Computer Science
King Abdulaziz University
Jeddah, 21577, Saudi Arabia

Abstract

Exascale computing refers to a computing system which is capable to at least one exaflop in next couple of years. Many new programming models, architectures and algorithms have been introduced to attain the objective for exascale computing system. The primary objective is to enhance the system performance. In modern/super computers, GPU is being used to attain the high computing performance. However, it’s the objective of proposed technologies and programming models is almost same to make the GPU more powerful. But these technologies are still facing the number of challenges including parallelism, scale and complexity and also many more that must be fixed to achieve make computing system more powerful and efficient. In this paper, we have present a testing tool architecture for a parallel programming approach using two programming models as CUDA and OpenMP. Both CUDA and OpenMP could be used to program shared memory and GPU cores. The object of this architecture is to identify the static errors in the program that occurred during writing the code and cause absence of parallelism. Our architecture enforces the developers to write the feasible code through we can avoid from the essential errors in the program and run successfully.

Keywords: Exascale Computing, GPU, CUDA, OpenMP, Parallelism, High Performance Computing (HPC).

1. INTRODUCTION

High Performance Computing (HPC) technology architectures and algorithms are anticipated to transfer dramatically in the future. Accordingly, increasing on-chip parallelism is becoming the objective of computer companies to achieve high performance [1]. In modern computers, Multi-core technology is proving very good offer in order to get high performance and power efficiency. With perspective of programming model, in order to take advantage of multi core technologies architecture, OpenMP was introduced [5]. Another major challenge for exascale computing is to parallelism in computing.

1.1. CUDA Programming

CUDA (Compute Unified Device Architecture) is a parallel computing architecture developed by NVIDIA [2]. For this purpose, usage of GPU is introduced that consist of multi core resided in it. GPU is similar to CPU in a computer system but is very powerful. Many programming models are available to write program for GPU but CUDA by NVIDIA is the best option in order to achieve parallelism through GPU processing [8]. CUDA provides variety of key abstractions shared memory, parallelism in computing for multi cores and barrier synchronization as well. Moreover, CUDA architecture provides strong computational interfaces including OpenGL and Direct Compute [9]. CUDA programming model overcome the challenges that are face in parallelism.

1.2. OpenMP Programming

Open multi-processing OpenMP is a programming model that have capability to handle multithreading by computing in parallel module. The basic idea behind this programming model is data processing parallelly. OpenMP consists of number of directives and libraries that are called runtime [2]. It also processes the looping region as parallelized by inserting compiler directives in starting region of OpenMP module that makes the program more efficient and improves overall application performance [3]. An example of parallelism in loop region using OpenMP is show as follows:

```
# pragma omp parallel shared(a1,a2,a3,chunk) private(i)
{
#pragma omp for schedule (dynamic, chunk) nowait
for (i=0; i < N; i++) {
a3[i] = a1[i] + a2[i];
}
}
```

1.3. Hybrid Programming

In this paper, we have used a hybrid programming model by combining CUDA and OpenMP as well. The purpose to use both models at a time is to write program for of GPU and shared memory [8]. However we can write program for GPU in order to make parallel processing in both GPU and CPU as well. As CUDA will handle GPU parallelism and OpenMP to run the CPU process in parallel way [6].

1.4. GPU Architecture

In modern computers, GPU acts as a second computer. Similar to CPU It also has its own memory and processors. The CPU get input from user and classified either it is related to GPU or CPU itself for processing. In case of GPU processing, the information is forwarded to GPU for processing. GPU process the task and send the processed information back to CPU for further utilizing [12]. GPU consist of two main components.

- Global memory
- Streaming Multiprocessors (SMs)

Global memory is accessible by both GPU and CPU inside the system. Second component streaming multiprocessors is the core part of GPU that performs the actual computation for GPU [12]. SMs consist of multiple cores and shared memory where each core has its own control unit (CU), registers, execution pipeline and cashes. The basic architecture of a GPU is as follows:

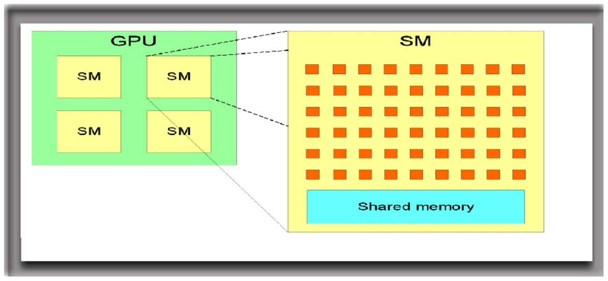

FIGURE 1: GPU Basic Architecture.

Rest of the paper is organized in such a way that, Section II describes the type of errors that are the hurdle to accomplish the GPU goal using CUDA and OpenMP. Section III deals with the errors discussed in section II and also solution to avoid such sort of hurdles in program. Section III presents the testing tool architecture that prevents from the type of errors that are discussed in section II. Further, a conclusion is presented in section IV.

2. TYPE OF ERRORS IN HYBRID MODEL

In this section, we have presented different type of errors that occurred during writing a program for GPU cores and shared memory using hybrid programming model. Let's discuss these errors one by one explaining how to produce in program.

2.1. Data Race

It is basically a computational hazard that comes up when the results of the program depend on the execution for another program. Such kind of error arises normally when two or more threads are running in parallel [6][7]. A simple scenario how data race error occurs in program is described in below example.

Let's x that points to a shared memory in a program. The requirement is to get the increment in x value. In sequential processing, this normally happen in three steps as:

1) Read the value of x into a register
2) Add 1 to the value read in step 1.
3) Write the results back to x and final results.

But what about parallel processing of same scenario, race condition produced or not? Let's discuss the same problem in parallel processing where multiple threads are present and process parallelly.

Assume that initial value of x is 3

1) Thread A reads the value 3 from x.
2) Thread B reads the value 3 from x.
3) Thread A adds 1 to its value 3, to make 4.
4) Thread A writes its value 4 back to x.
5) Thread B adds 1 to its value 3, to make 4.
6) Thread B writes its value 4 back to x.
7) Final value in x is 4 which is incorrect.

Why this happen even this was not to complex statement? Does it due to data race? Answer is yes but how we can avoid occurring such error in hybrid programming model.

Solution:
In CUDA, we can avoid from occurrence data race problem by adding some additional statements [11]. Normally Locking / Unlocking and atomic operation are used to handle data race error in CUDA. These operations process the threads in traditional sequential way.

Lock / Unlock operation
```
global void  test1( Lock lock , int  *nblocks)
{
        i f ( threadIdx.x == 0 )
        {
                lock.lock( ) ;
                * nblocks = *nblocks + 1 ;
                lock.unlock ( ) ;
        }
}
```
In above example code, the statement written inside lock and unlock block will be executed sequentially. In this way, there will be no data race but obviously the performance is affected.

Atomic Operation in CUDA

```
#include <stdio.h>
#include<cuda.h >
global   void   colonel ( int *b_f ) {
        atomicAdd ( b_f , 1 ) ;
}

int main ( )
{
int a = 0 , *a _d ;
colonel <<<1000,1000>>>(b_f ) ;
}
```

In above example code, atomic statement has been used to avoid from data race problem writing code in CUDA. Atomic is built in method used in CUDA for also other operation like: multiplication, subtraction, division etc.

Atomic Operation in OpenMP
Atomic operation is also used in OpenMP for same purpose but here atomic keyword is used as directive in the code. Adding this directive the code written inside the OpenMP block will be processed sequentially.

```
#include <omp.h>
int count;
void Tick()
{
  #pragma omp atomic
     count = count+1;
 }
```

In above code example atomic is the directive that specifically used for running the code sequentially inside it.

2.2. Use of lock without Unlocking
This is error related to using locking/unlocking statement in the program. As we have discussed the usage of lock and unlock statement to avoid data race condition in program, however it is also necessary to make sure that unlock statement is also present there once lock statement is used in the program [4].

2.3. Use of ordered clause without ordered construct
Another type of error occurred normally in OpenMP part from hybrid programming model which is the usage of ordered clause with ordered construct. Once an ordered clause is placed within for work-sharing construct and developer forgot to place a separate ordered clause inside for loop, an error will be occurred in the program and application will be crashed.

2.4. Use of critical when atomic operation is sufficient
Basically, this is performance related issue that normally considered when we use critical clause in the program even the problem could be solve by using simply atomic clause rather than critical [4]. By usage of critical clause in this scenario, the overall system performance will be affected.

2.5. Placing much code inside Critical region
Normally it has been seen that the novice programmer places a lot of code in the critical section and cause the number of errors. These errors could be cause of further two sub errors as follows:

- Blocking the other threads more than required.

- Paying the maintenance cost more than expected associated with that specific region [4].

Example code:

```
#pragma omp parallel for
for ( i = 0 ; i < N; ++i )
{
#pragma omp critical
{
        if ( arr [ i ] > val)
  {
        if ( arr [ i ] > max)
  max = arr [ i ]
  .
  .
  .
  .
  and a lot of more statements…
  }
}
```

So, it is suggested to resort reduce the code written inside the critical clause region.

2.6. Missing for in "#pragma omp parallel for"

It is very common type of error in OpenMP, mostly people use 'for' clause to include a loop statement inside the '#pragma' region but forgot to add 'for'. This error leads every thread the whole loop, but not only parts of it [4].

Example code:

```
        #pragma omp parallel for
        {
                if ( arr [ 0 ] > val_1)
                {
                        if ( arr [ 1 ] > val_2)
                        max = arr [ 2 ] ;
                }
        }
```

In above code, as 'for' clause is being used but no for is used inside the region. It will cause of error occurrence in the program.

3. PROPOSED ARCHITECTURES AGAINST DETECTED ERRORS

In order to avoid the errors that we have discussed in the last section, we have proposed the testing architectures for particular errors that must followed by the developer before/during writing code.

3.1 Data Race

As data race is the common error that occurs normally when we write program for GPU cores and shared memory as well. Basically, data race is type of error that occurs in a program when the same statement is required for two or more different process. Each process in a program want to compute its set of statements as soon as possible. But this could only be possible if all required statements are in hand and could be executed in parallel. In case of availability of all the parameters and statements that specific process could be executed in parallel otherwise a race condition will be occurred. It means that the same data is being used in any other statement or waiting for some result from other process. So, to avoid this issue in parallel processing, firstly we analyze the code and the resolve the issue on behalf of condition. One of the solution is to compute that specific part of code as sequentially using locking/unlocking statements.

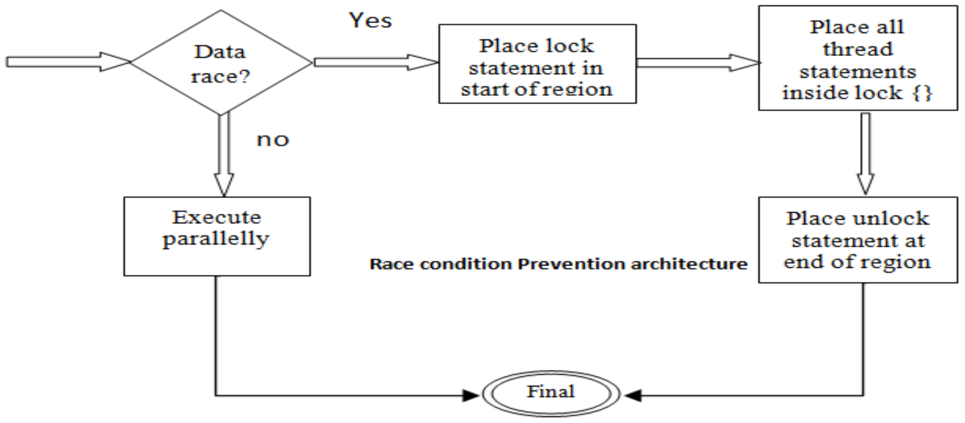
Data race?
Yes
Place lock statement in start of region
Place all thread statements inside lock {}
no
Execute parallelly
Race condition Prevention architecture
Place unlock statement at end of region
Final

FIGURE 2: Data Race prevention architecture for CUDA.

The above architecture is related to error for race condition (A) in CUDA and also the prevention of errors that occurs by adding locking (B) statement in the program.

Similarly, for data race problem in OpenMP, once you have analyzed the code to find out that either there is data race exist or not. After that you should follow the below proposed architecture that will ensure you to write error free code.

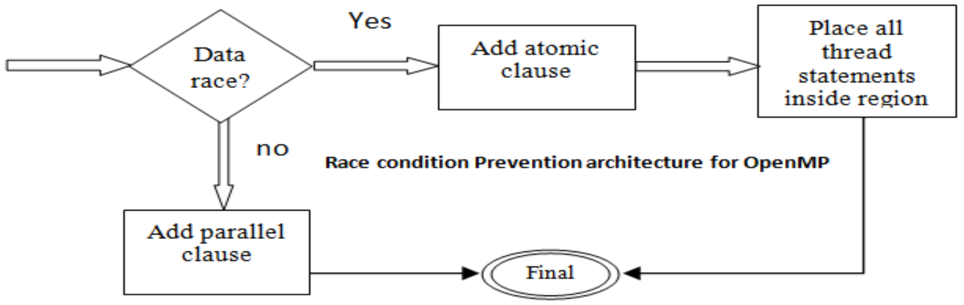
Data race?
Yes
Add atomic clause
Place all thread statements inside region
no
Race condition Prevention architecture for OpenMP
Add parallel clause
Final

FIGURE 3: Data Race prevention architecture for OpenMP.

3.2 Missing “For” keyword when using “For” Clause

Another architecture is presented to handle the error that occurs due to missing ‘for’ when ‘for’ clause is used in the program. This error is related to OpenMP from hybrid programming model. Below is the diagram representing that how we can insist a developer to must add ‘for’ when for clause has been used with ‘#pragma’ statement.

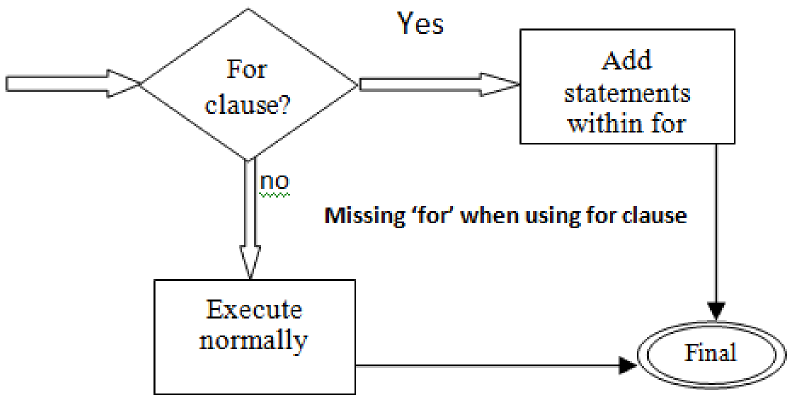
For clause?
Yes
Add statements within for
no
Missing 'for' when using for clause
Execute normally
Final

FIGURE 4: missing 'for' when using 'for' clause .

In previous section, rests of the errors are performance related, that must also be avoided to occurrence in the program. To handle these errors, the prevention statements should be followed before writing the program.

4. TESTING TOOL ARCHITECTURE EVALUATION

This testing tool architecture for hybrid model is proposed basically to improve exascale computing system. GPU is the basic unit that is being used to enhance the power of a system to achieve exascale computing. However, In order to achieve this certain level performance, CUDA, OpenMP do play a major role to program graphical processing unit. This architecture helps us to detect the possible number of errors from the code written in CUDA and OpenMP and improve the processing power of code as well. Keeping in view the programming layout in both the languages, we have proposed the testing architecture to evaluation the number errors in our code that could be cause to decrease the performance of a system. Using this proposed testing tool architecture, we can evaluate our code written in hybrid model languages such as CUDA and OpenMP and make error free by following it.

5. CONCLUSION

This paper is presented to make enhancement in exascale computing system. In order to obtain this objective, GPU which is the core unit for exascale system should be reviewed deeply to make more powerful. In this paper, we emphasized on parallelism and shared memory utilization in parallel and presented hybrid model based testing tool architecture for exascale computing system. In hybrid model, we add two CUDA and OpenMP programming models to enhance the system performance. We presented some major type of errors when a developer writes the program for GPU using this hybrid programming model. Further we discussed each error in detail by specifying that how these errors occurs in the program what is the cause of occurrence and how we can avoid from these errors during writing the program. We presented different architectures which specify that how to avoid these errors. Our architectures ensure to a developer to write an error free code if this is followed properly.

For future perspective, there is still need to emphasize some specific cases of coding like in OpenMP, how we can handle the 'nested' clause and nested loops inside this class to accomplish high performance [10]. There is also need to make a strong research on hybrid programming model when we use CUDA, OpenMP, MPI and other models to achieve the high performance computing leading to exascale system.

6. REFERENCES

[1] J. J. Shalf , S. Dosanjh, and J. Morrison, “Exascale Computing Technology Challenges”. Springer-Verlag Berlin Heidelberg. Pp. 1-25. 2011.

[2] J. M. Yusof et al, “Exploring weak scalability for FEM calculations on a GPU-Enhanced cluster”, 33.685–699. Nov, 2007.

[3] C.T. Yang, C.L. Huang and C.F. Lin, “Hybrid CUDA, OpenMP, and MPI parallel programming on multicore GPU”. Computer Physics Communications. Pp. 266-269. 2011.

[4] M. Suß and C. Leopold, “Common Mistakes in OpenMP and How To Avoid Them”. 2007.

[5] J.P. Hoeflinger and B.R. Supinski, “ The OpenMP memory model”. In: Proceedings of the First International Workshop on OpenMP - IWOMP .2005.

[6] D. A. Mey and T. Reichstein, “Parallelization with OpenMP and MPI A Simple Example (Fortran)”. Oct, 2007

[7] M. Zheng, V.T. Ravi, F. Qin, and G. Agrawal , “GRace: A Low-Overhead Mechanism for Detecting Data Races in GPU Programs”. ACM. Dec, 2011.

[8] G. Hager, G. Jost and R. Rabenseifner “Communication Characteristics and Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-core SMP Nodes”. Cray User Group Proceedings. 2009.

[9] D. Shreiner, M. Woo, J. Neider and T. Davis, “OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL(R)”, Version 2.1, 6th edition, Addison–Wesley Professional, Reading, MA, ISBN 0321481003, 2007.

[10] J. Gustedt, “Parallelizing nested loop in OpenMP,”,http://stackoverflow.com/questions/19193725/ parallelizing-nested-loop-in-openmp-using-pragma-parallel-for-shared, Oct. 5, 2013 [May 10, 2015]

[11] Alrikai, “CUDA racecheck,”http://stackoverflow.com/questions/13861017/cuda-racecheck-shared-memory-array-and-cudadevicesynchronize, Jan. 11,2013 [April 22, 2015]

[12] Daedalus, “How do CUDA blocks/threads map onto CUDA cores,”http://stackoverflow.com/questions/ 10460742/how-do-cuda-blocks-warps-threads-map-onto-cuda-cores, May 5, 2012 [May 14, 2015]