On Impact of Ordering Memory Accesses in Dynamically Scheduled Parallel Applications

Rakhi Hemani*, Abhishek Swaroop
Computer Science and Information Technology
Jaypee Institute of Information Technology
Noida, India
rakhi.hemani@jiit.ac.in, abhishek.swaroop@jiit.ac.in

Abstract— **Scheduling of parallel tasks is a complex problem as many computational resources are available. The scheduling decision is usually guided by cost models that typically take into account computation time and memory access times. These models do not try to schedule tasks on the basis of the actual memory accesses made by the tasks. It has been proved that sequential memory accesses are much more beneficial than random memory accesses. Thus it may be concluded that the scheduling decision should take into account the order of memory accesses made by different parallel tasks.**

In this work we demonstrate the impact of ordering blocked memory accesses in a dynamically scheduled parallel application (7 point Stencil). We observe a performance degradation of 30%-50% incases of random memory accesses as compared to sequential memory access.

Keywords- Memory Accesses, Multi-core, Task scheduling;

I. Introduction

Multi-core architectures have emerged as the technology for getting more performance [1] today. To benefit from this underlying architecture, user applications must execute in parallel and engage all available resources at all instants of time. Usually parallelizing of an application involves three distinct issues: - identification of parallelism, expression of parallelism and matching of parallelism to the available platform. In this paper we focus on the last aspect - "matching of parallelism to the available platform".

* Student Author

An application may be parallelized by splitting it into many smaller tasks and scheduling each task on a computational unit. This scheduling may be done statically or dynamically. Static scheduling of applications usually involves an in-depth analysis of application and can be tailor made to specific set of resources. Dynamic scheduling is more flexible and, in our opinion, it is easier to apply on any type of platform. Such type of scheduling usually involves predicting time for completion of various tasks. This is usually achieved by help of parameterized cost models for each task. These parameterized cost models take into consideration resource and task based parameters. It then predicts time taken for executing a task on each available resource. The appropriate resource may then be selected. Typical cost models take into consideration time for computations and memory accesses.

It may also be noted that with almost static clock frequencies, memory accesses are no longer neglected. Apart from the time taken to fetch memory, the order of memory accesses also impacts memory performance. In [2] authors have reported that due to pre-fetching memory accesses are faster if sequential memory is accessed.

In this work, we focus on the impact of ordering memory accesses in a dynamically scheduled parallel application, by comparing performance of random and sequential accesses to blocks of memory. Our experiments show that if all other factors are constant the time taken for random accesses is always more than the time taken for sequential accesses. The degradation observed was in the range of 30%-50% in our experiments.

The remainder of this paper is organized as follows: in Section II of this paper we describe the scheduling platforms available today, in particular the StarPU platform. In section III of the paper we describe the stencil application and its subsequent scheduling on StarPU platform. Our experimental platform, methodology and results are discussed next. Finally in the last section we conclude our work and discuss the future direction of our work.

II. Scheduling Parallel Applications

[3] and [4] have considered scheduling of parallel tasks on heterogeneous parallel platforms. The first paper presents a highly optimized static library for heterogeneous architectures, and the second focuses on dynamic scheduling of parallel tasks integrated with data management. The second approach is found to be competitive and yields performance similar to the first. It is obvious that static scheduling of all applications is difficult, as it involves in-depth analysis of application and can be tailor made to specific set of resources.

Run time systems like StarPU rely on cost models for scheduling parallel tasks. StarPU gives the facility of using many scheduling policies and cost models. The cost models take into consideration time taken for computation and memory accesses (dmda). Developing good cost models is a separate subject of research. For example [5] have recently developed cost models for multi-core machines. Their model includes impact of memory contention. By the proper use of such models it is possible to decide the computational resource on which each of the parallel tasks is scheduled. It is also possible to calculate the time taken to transfer memory when distributed memory resources are available.

However, none of the cost models discussed above, consider the actual memory accesses made each task. For example, consider the situation when multiple tasks having the same computation cost and accessing same amount of memory have to be scheduled on a shared memory, multiple core processors. In this scenario, the current cost models will not differentiate among the different tasks, as all will have the have the same cost.

In [2] it has been reported that when blocks of memory are accessed after some skip length, the memory performance reduces as the block length is decreased. This is because in sequential accesses pre-fetching plays a major role. This work is reported when all memory accesses were made by a single processor.

In this paper, we consider the scheduling of tasks that have same number of computations and access same amount of memory. However, each task accesses different memory. We want to find the impact of ordering such tasks on basis of memory accesses that they make. For example, all tasks accesses M amount of memory and comprise of C number of computations. Now if number of such tasks exceeds the number of computational resources present, then our objective is to find the impact of scheduling tasks which access sequential memory and tasks which access random memory.

In the next section, a basic 7 point stencil kernel and its adaptation to StarPU platform is described.

III. Application Design

Stencil kernel is used for solving partial differential equations. We consider a 7 point stencil application which operates on two distinct, three dimensional matrices *a* and *b* of size *NX* * *NY* * *NZ*. We consider updating the first matrix *a* based on the values of the second matrix *b*. The formulation is:

```
for ( i = 1; i < NX - 1 ; i++ )
    for (j = 1; j < NY - 1 ; j++ )
        for ( k = 1 ; k < NZ – 1 ; k++)
{
a( i , j , k ) =        b( i , j , k )
            + b( i-1 , j , k ) + b( i+1 , j , k )
            + b( i , j-1 , k ) + b( i , j+1 , k )
            + b( i , j , k-1 ) + b( i , j , k+1 )
}
```

Stencil updates are considered memory intensive operations and many previous researches for example [2] have optimized stencil operation.

For adapting this application to the StarPU platform tasks are defined by splitting the computations. The splitting is done by dividing the *a* matrix along the x axes (i.e. the iterations of the outermost for loop), into blocks of size *BLOCK_SIZE*. This divides the matrix into many slices each with dimension *BLOCK_SIZE * NY * NZ*. The computation of each such slice is a task and is submitted to StarPU platform. Note that this division requires appropriate division of B matrix also.

Insertion of tasks is done in sequential or random order on basis of a Slice_Order vector. For sequential accesses slices are inserted in their natural order, i.e. slices that access sequential memory are inserted sequentially. For random accesses, first Slice_Order vector is randomly assigned values in the range (*0 – ((NX / BLOCK_SIZE) – 1)*), then tasks are inserted in this random order.

IV. Experiments

A. Platform

All experiments were carried on i3-540 processor, with 3.06 GHz clock rate. This processor has dual core architecture with 2 hyper-threads, 32 KB L1 cache, 256 KB L2 cache and 4096 KB L3 cache. The system shares L3 cache among all cores, where as L1 and L2 caches are private. It has 1752 MB of RAM and 150 GB of hard disk and runs Fedora 13 Operating System.

StarPU 0.9 was installed with hwloc library support. This helps StarPU to schedule all tasks on the different cores. No task was scheduled on hyper-threads.

B. Methodology

For our experiments we considered a matrix of size 2050 * 514 * 514. BLOCK_SIZE was varied from 4 to 512. Tasks were submitted to the StarPU platform in random as well as sequential order. Each experiment was repeated 5 times. To ensure minimum interference with other processes the experiments were run using run-level 2 of the Operating System.

C. Results

The results obtained are shown in tabular form in Table 1 and 2. The same results are plotted in Figure 1. Both average and minimum time taken for random and sequential task insertions are described. It is observed that time taken by random strategy is always more than time taken for sequential strategy. From the experiments we notice that performance degradation is somewhat inversely proportional to block size. This is expected, because as block size decreases, the number of blocks increase, and more memory is fetched in random order.

Table 1 Average Performance Degradation

Block Size	Random Avg	Sequential Avg	% Degradation Avg (Random on Sequential)
4	137.42	95.02	30.85
8	125.90	74.67	40.69
16	117.90	60.65	48.56
32	125.72	60.43	51.94
64	108.39	53.42	50.72
128	87.72	56.84	35.21
256	86.85	51.90	40.24
512	83.32	49.95	40.05

Table 2 Minimum Performance Degradation

Block Size	Random Min	Sequential Min	% Degradation Min (Random on Sequential)
4	100.03	65.96	34.06
8	105.77	53.62	49.31
16	105.89	51.56	51.31
32	107.93	55.28	48.78
64	89.43	39.93	55.35
128	59.65	47.38	20.57
256	67.96	41.64	38.73
512	60.81	48.66	19.97

V. Conclusion and Future Work

The experiments show that ordering of memory accesses impacts performance of an application. We plan to extend our study to more applications for example graph applications as they have random memory accesses. We also plan to extend run-time frameworks like StarPU to include memory access information while scheduling tasks.

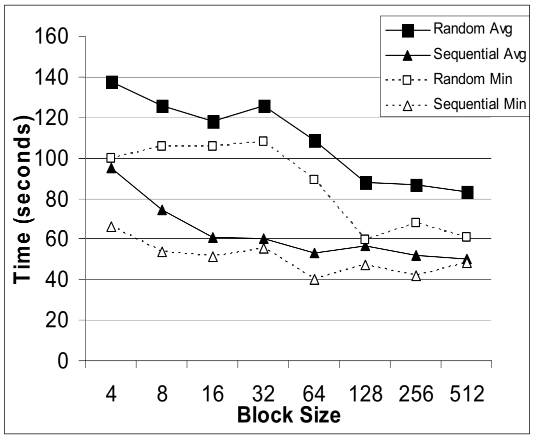

Figure 1 Impact of ordering memory accesses on Stencil Application

REFERENCES

[1] Asanovic, Krste, et al., "The Landscape of Parallel Computing Research: A View from Berkeley ", Technical Report No. UCB/EECS-2006-183, December 18, 2006.

[2] Datta K. etal, "Optimization and Performance Modelling of Stencil Computations on Modern Microprocessors", SIAM Review, Number 51, Issue 1, Pages 129-159, 2009.

[3] Tomov S., Dongarra J., and Baboulin M., "Towards dense linear algebra for hybrid GPU accelerated manycore systems", Parallel Computing, Volume 36, Issue 5-6, June 2010.

[4] Augonnet C. etal, "StarPU: a unified platform for task scheduling on heterogeneous multicore architectures", Concurrency and Computation: Practice and Experience, Special Issue: Euro-Par 2009, Volume 23, Issue 2, February 2011.

[5] Wu X. and Taylor V., "Performance Modeling of Hybrid MPI/OpenMP Scientific Applications on Large-scale Multicore Cluster Systems" in the 14th IEEE International Conference on Computational Science and Engineering (CSE-2011), Dalian, China, August 24-26, 2011