USING NP-COMPLETE PROBLEMS TO COMPARE A CPU, GPU, AND THE INTEL® XEON PHI™ COPROCESSOR

David M. Toth, Zachary Goodwyn, Jerome Mueller
University of Mary Washington
1301 College Avenue
Fredericksburg, VA 22401, USA
dtoth@umw.edu, zgoodwyn@umw.edu, jmueller@umw.edu

ABSTRACT
As accelerators are integrated into many of the newest workstations and supercomputers to provide large amounts of additional processing power, selecting the appropriate one is critical for achieving the best performance. The new Intel® Xeon® Phi™ coprocessor provides more processing cores than a CPU but less than a graphics processing unit (GPU). However, the Phi's cores do not have the limitations of a GPU's cores and can also run code written for traditional CPUs instead of requiring code written specifically for GPUs. We used the traveling salesman problem, the knapsack problem, and the party problem, three NP-complete problems, as benchmark applications to compare the relative performance of the Phi with a contemporary GPU and CPU. Programs were written to solve the problems on the CPU and on the coprocessor that could be easily ported to run on the GPU with only minor modifications. The length of time the programs took to complete on the coprocessor, the GPU, and the CPU was measured. While the GPU attained speedups of almost 14 to 80 over a single CPU core for the problems, the coprocessor only attained speedups between 4 and 7 for the problems.

KEY WORDS

1. Introduction

Although originally intended to increase the speed of creating graphical output for computers, graphics processing units (GPUs) have been used for a number of years to increase the speed of non-graphical applications. Often these devices are now referred to as accelerators, since some of the most recent GPUs, such as NVIDIA's Tesla K20c, do not have video output ports. The accelerators are added to computers by connecting them to PCI Express slots in computers. Many computers have multiple PCI Express slots, allowing them to support multiple accelerators. To run programs on these accelerators, one needs to use a special libraries such as CUDA or OpenCL to develop code that offloads computations to the accelerators. Code written for a general purpose x86 or x64 CPU will not run on one of these accelerators without modification.

Intel® has developed an accelerator that differs significantly from the existing accelerators. While NVIDIA's accelerators have up to 3072 processing cores which are not fully independent x86 or x64 processors, the Intel® Phi™ coprocessor has a much smaller number of x64 processing cores (up to 61) which are fully independent [1, 2]. Like the other accelerators, the Phi connects to the host computer through the PCI Express slot. To use the Phi, one does not need to use additional libraries like CUDA or OpenCL, and code written for an x86 or x64 CPU will run on the Phi with no modifications. In addition to this, code written with OpenMP calls that can run on a multi-core computer and take advantage of the multiple cores can be run on the Phi with no modifications. The Phi became generally available on Jan 28, 2013, so there has been relatively little work done comparing its performance to CPUs and GPUs [3].

We compare the performance of the Phi to the performance of a contemporary CPU and GPU for solving three CPU-bound NP-complete problems: the traveling salesman problem, the party problem, and the knapsack problem. In the traveling salesman problem, a set of cities and the costs to get between each city is given. The solution is the least cost circuit through the cities that returns to the starting city [4]. The party problem, from a branch of mathematics called Ramsey theory, asks how many people must attend a party to guarantee that there are m people there that all know each other or n people there who are all total strangers [5]. The knapsack problem is an optimization problem with the goal of selecting items with weights and values in order to maximize the value of the items selected while keeping the total weight of the items below a set value [6]. We use NP-complete problems because of the computationally-intensive nature of the problems and because they are easily parallelized to take advantage of all of the multiple processing units on the hardware. Thus, we expect to be able to maximize the utilization of the processing units.
2. Related Work

GPUs have been used as accelerators for a wide variety of problems in science and math. In physics, they have been used to solve n-body problems and gas flow problems [7, 8]. In biology, they have been used to speed up sequence alignment and fix errors in DNA sequencing [9, 10]. GPUs have been used for medical research including cancer research and chemical informatics [11, 12]. They have also been used in weather prediction and to try to make progress on the party problem [13, 14].

The Phi has been used to achieve speedups in code for high energy physics and molecular dynamics [15, 16]. It has also been used to increase the speed of discovery of biological networks [17]. However, these projects have involved optimizing code for the applications.

Although there has been a significant amount of work done to try to optimize methods for solving the traveling salesman, party, and knapsack problems, we note that our work is not trying to solve those problems more efficiently, but instead use them to compare the performance of different types of hardware.

3. Algorithms

We used brute force solutions for the traveling salesman, party, and knapsack problems. While we acknowledge that there are faster algorithms for these problems, the purpose was to compare hardware with code that is as similar as possible rather than find optimal algorithms for each platform. This is because a significant portion of high performance computing users are not computer scientists, but rather natural and physical scientists or engineers who are looking for easy performance gains from new hardware, rather than spending large amounts of time optimizing their code. Therefore, we wanted to give a comparison that would be useful to that audience. However, we acknowledge that using optimal algorithms may result in different relative performance between the various types of hardware.

Each of the problems we tested can be broken up into blocks of tasks that can be solved separately. Once the blocks of tasks have been solved, the solution to the entire problem can be quickly determined based on the results of each block. For the traveling salesman, each of the \( n \) processing cores from a piece of hardware computes the value of \( 1/n^3 \) of the possible tours and saves the best value. Then the best value from each of the \( n \) blocks is determined by one core. A similar approach is used for both the party problem and the knapsack problem. For the knapsack problem, each core tests \( 1/n^3 \) of the possible permutations of items that can go in the knapsack, saving the permutation with the best value and the value. Once again, the best value from each of the \( n \) blocks is determined by one core. For the party problem, we refer the reader to [14] for the algorithm.

All three algorithms use very little memory because each thread uses only a tiny amount of memory and there are very few threads, except for the GPU. However, the threads use so little memory that even the GPU doesn't use much memory, even with millions of threads. The memory used was well below the GPU's 5 GB of onboard memory. The traveling salesman and knapsack algorithms perform mostly addition operations with some comparisons. The party problem algorithm performs mostly comparisons and very few arithmetic operations.

4. Methodology

To conduct our tests, we used a custom computer with two 4-core Intel® Xeon® E5-2609 2.4 GHz processors and 16 GB of RAM. The GPU we used was an NVIDIA® Tesla® K20c GPU which has 2496 CUDA cores running at 706 MHz and 5 GB of onboard memory. The Intel® Xeon® Phi™ 5110P has 60 cores running at 1.053 GHz and 8 GB of onboard memory. The display for the system was handled by a second graphics card so that the GPU didn’t need to spend cycles processing the video for the system. To get a baseline to which we could compare the GPU and Phi's performance, we ran the OpenMP code on the CPU using 1, 2, 4, and 8 cores using 1, 2, 4, and 8 threads respectively.

Our only attempt to tune the programs beyond the simplest port from the CPU and Phi to the GPU was to write our own version of the C++ Standard Template Library (STL) next_permutation function for the GPU version of the code since that function could not be used on the GPU [18].

To get the best performance from the GPU and Phi, it was recommended that the code be run with different numbers of blocks and threads per block for the GPU and different numbers of threads for the Phi and to use the values that perform best when running a program [19, 20, 21, 22]. Therefore, we also ran the programs with different combinations of blocks and threads per block for the GPU and different numbers of threads for the Phi, as that's a simple test that we expect all people will do since it can affect the runtimes of programs significantly. For the GPU, we ran the code using each combination of 8192, 16384, 24576, 32768, 40960, 49152, and 57344 blocks and 64, 128, 256, 512, and 1024 threads per block. For the Phi, we ran the tests with 120 and 240 threads. We found there was a significant impact on the runtime of the programs. For the GPU, the longest runs of the traveling salesman, knapsack, and party problems were 8%, 30%, and 148% longer than the shortest runs of those programs. For the Phi, the longest runs of the traveling salesman, knapsack, and party problems were 4%, 33%, and 10% longer than the shortest runs of those programs.
5. Results

For each program on each piece of hardware, we conducted 10 trials per parameter set (number of blocks and threads per block for the GPU and number of threads for the coprocessor) and took the average of the trials small enough for us to conclude that 10 trials were sufficient to get accurate results. As an example, the range between the times the trials took was on the order of 1 second for the GPU trials. The average time that each program took to complete on the CPUs, GPU, and coprocessor for each problem are shown in Figures 1, 2 and 3. In all of the problems, the GPU outperformed all 8 CPU cores significantly. For the party problem, the GPU took only one tenth of the time the 8 CPU cores did to complete the program. The coprocessor always outperformed 4 of the 8 CPU cores in the system, but never all 8 CPU cores. The average completion times and speedups are shown in Tables 1, 2, and 3. By increasing the number of CPU cores, we observed linear speedup for the traveling salesman and party problems and nearly linear speedup for the knapsack problem.

Figure 1 - Traveling Salesman Problem Results
Figure 2 - Party Problem Results

Figure 3 - Knapsack Problem Results
Table 1 - TSP Times and Speedups

<table>
<thead>
<tr>
<th>Device</th>
<th>Time (min)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU: 1 core</td>
<td>112.5</td>
<td>--------</td>
</tr>
<tr>
<td>CPU: 2 cores</td>
<td>56.2</td>
<td>2.0</td>
</tr>
<tr>
<td>CPU: 4 cores</td>
<td>28.1</td>
<td>4.0</td>
</tr>
<tr>
<td>CPU: 8 cores</td>
<td>14.1</td>
<td>8.0</td>
</tr>
<tr>
<td>GPU</td>
<td>5.7</td>
<td>19.8</td>
</tr>
<tr>
<td>Coprocessor</td>
<td>24.9</td>
<td>4.5</td>
</tr>
</tbody>
</table>

Table 2 - Party Problem Times and Speedups

<table>
<thead>
<tr>
<th>Device</th>
<th>Time (min)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU: 1 core</td>
<td>138.0</td>
<td>--------</td>
</tr>
<tr>
<td>CPU: 2 cores</td>
<td>69.1</td>
<td>2.0</td>
</tr>
<tr>
<td>CPU: 4 cores</td>
<td>34.6</td>
<td>4.0</td>
</tr>
<tr>
<td>CPU: 8 cores</td>
<td>17.3</td>
<td>8.0</td>
</tr>
<tr>
<td>GPU</td>
<td>1.7</td>
<td>79.4</td>
</tr>
<tr>
<td>Coprocessor</td>
<td>19.8</td>
<td>6.9</td>
</tr>
</tbody>
</table>

Table 3 - Knapsack Problem Times and Speedups

<table>
<thead>
<tr>
<th>Device</th>
<th>Time (min)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU: 1 core</td>
<td>419.5</td>
<td>--------</td>
</tr>
<tr>
<td>CPU: 2 cores</td>
<td>212.0</td>
<td>2.0</td>
</tr>
<tr>
<td>CPU: 4 cores</td>
<td>106.6</td>
<td>3.9</td>
</tr>
<tr>
<td>CPU: 8 cores</td>
<td>53.8</td>
<td>7.8</td>
</tr>
<tr>
<td>GPU</td>
<td>30.4</td>
<td>13.8</td>
</tr>
<tr>
<td>Coprocessor</td>
<td>68.0</td>
<td>6.2</td>
</tr>
</tbody>
</table>

6. Conclusion and Future Work

Our results have shown that for the programs we tested, a modern GPU significantly outperformed not only a contemporary CPU, but also the comparably priced Intel Phi coprocessor. For the programs tested, the coprocessor outperformed a single quad-core CPU, but not two quad-core CPUs.

Both GPUs and the coprocessor have advantages that should be considered carefully when choosing the appropriate accelerator for an application. A minor advantage of the coprocessor is that multiple traditional sequential programs can be run on the coprocessor at one time, allowing it to function as multiple slow CPUs for parameter sweep applications and making it more versatile than a GPU. Multiple multithreaded programs can also be run on it, as well as a single multithreaded program, adding to its versatility. A more significant advantage of the coprocessor over GPUs is development time for programs is significantly less for the coprocessor. To make use of the coprocessor, one does not need to learn any special libraries that they would not already know to make use of a multi-core CPU. In fact, with the coprocessor, code written for a CPU does not need to be altered to run on the coprocessor. In contrast, code must be modified to run on a GPU. The coprocessor allows the use of OpenMP and all the usual STL features while some of the STL features are not available for use on GPUs.

GPUs also have some significant advantages over the coprocessor. The performance gains from the GPU for our applications were far more significant than those from the coprocessor, with the GPU achieving a speedup of 79.4 for one problem when the coprocessor only managed a speedup of 6.9 for that problem. In addition to this, the coprocessor, while slightly cheaper than the top of the line GPU requires a special system to host it, while the GPU does not. The GPUs can be used in any system with a PCI express slot and adequate power supply. However, the coprocessor requires an Intel-based system and because our model is passively cooled, our system needed a special case with more fans that have to be run on full speed constantly to keep the coprocessor from overheating. The cost of the new system, case, and fans far outweigh the higher cost of the GPU. We expect that the case and fan issues might go away with the release of an actively cooled coprocessor. However, an Intel-based computer will still be required to host the coprocessor.

Future work includes testing the hardware with additional programs that utilize different instruction mixes, which may impact the relative performance of the devices. For example, we expect that algorithms that have a significant amount of branching may perform better on devices other than the GPU. We also intend to test optimized algorithms for the traveling salesman and knapsack problems for each platform to demonstrate the speedup that can be achieved when specifically writing optimized code for each device. It is possible that such tests may demonstrate significant differences in the relative performance of the hardware. Finally, we intend to evaluate the Phi’s suitability for virtual screening by comparing the performance of molecular docking software on the Phi to its performance on regular CPUs.

Acknowledgements

We wish to thank the University of Mary Washington and NVIDIA for their support of this research. The University of Mary Washington provided funding for our research experience through its Summer Science Institute program. This funding enabled to undergraduates to spend their summer break working with a faculty member on this project. We also want to thank NVIDIA for providing the Tesla K20c GPU we used in our research through their Academic Partnership program.

References


Using Intel® Xeon® Processors and Intel® Xeon Phi™ Coprocessors


