CSC 400 Assignment: ITRG miniproposal

Performance portable fine-grained parallel programming model for heterogeneous computing system

Project Summary

For scientific computing, applications have to be run on hundreds or thousands of computing devices parallel because of the huge amount of data and calculations. Due to the various kinds of computing devices, programmers have to write and optimize their code according to the architectures of the computing devices and the way they connected. This process is tedious and not portable when the devices or the connection changed. So this project is to propose a performance portable parallel programming model for heterogeneous computing system. An abstract execution platform is to be designed, so the programmers do not need to care about the real-world execution machine. A runtime is to be designed to automatically distribute the tasks which the programmers assigned to the abstract execution platform to real-world machine. And a source-to-source code generator is to be designed to adapt the code from the architecture of the abstract execution platform to the real-world computing devices.

Project Description:

Introduction

Modern computing devices are made to support parallel computing, such as multi-core CPU and many-core GPU in every personal computer. On these devices, the optimized parallel program can run tens or even hundreds times faster than the sequential one\(^1\). Various parallel programming models are designed and used to implement real-world applications on parallel computing devices. They can be divided into two categories based on the view of programmers\(^2\):

• Explicit parallel programming model

  Explicit parallel programming model is to give programmers full control of each thread in the parallel program, such as task partition, synchronization, communication and so on. This provides the possibility for programmers to write and tune a program to its peak performance. But explicit parallelism makes this approach time-consuming and difficult.

• Implicit parallel programming model

  Implicit parallel programming model is to add compiler directives to sequential programs. It seems to be an easy and natural way for parallel programming, because the
sequential codes are still reserved, such as OpenMP, OpenACC and etc. But if optimizations like loop tiling, loop interchange, vectorization and so on are introduced, the sequential programs are as hard to understand as explicit programming languages.

OpenCL[^4] is one parallel programming model designed for heterogeneous computing. Its code can run on different computing devices without modification. But it has limitations: (1) OpenCL does not provide a method to adjust its programs’ parallel granularity according to different architectures. So its performance portability is poor[^3]. (2) OpenCL does not provide support for network communication. So if programmers want to use clusters, they have to use MPI interface. Kim designed an OpenCL based framework which can support network communications.[^5] But the first limitation is still an open issue.

Our programming model chooses to be explicit one with two-level parallelism (OpenCL style). The explicit parallel programming model can expose more parallelism to the runtime and compiler, this will reduce the burden of our runtime and compiler and higher performance is more likely to achieve. Threads and thread blocks are provided, so the parallel granularity can be flexibly controlled according to the architecture of the computing device.

The framework of the performance portable fine-grained parallel programming model is shown in Fig. 1.

---

**Research Plan**

There are three phases of this project: design an abstract execution platform, design a source-to-source code generator and design a runtime for workload distribution.

- **Abstract execution platform design**

  The abstract execution platform should contain all the features of the modern computing devices. So base on the analysis of the architecture of CPU, GPU, MIC and etc, the organization of the computing unit and the memory hierarchy will be carefully designed.

- **Source-to-source code generator design**

  In order to make the performance portable, source-to-source code generator should be design to transfer the code written for abstract execution platform to real-world computing devices. Thread mapping and memory mapping should be done by this code transformation. Thread mapping should carefully choose parallel granularity according to the execution devices. Thread coalescing and thread division method should be designed. Also different architectures have different levels of memory from the abstract execution platform, so...
memory mapping should be designed to reduce the memory access.

- Runtime design

After the mapping, the runtime should do a profiling to determine the workload of each computing devices in the real-world platform.

**Time table**

<table>
<thead>
<tr>
<th>No.</th>
<th>Schedule</th>
<th>Year 1</th>
<th>Year 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Abstract execution platform design</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Source-to-source code generator design</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Runtime design</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Test</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**References Cited:**


