CS59200CGP: Compilers for GPUs

Fall 2024

Administrative Matters

Credit hours (3)

Prerequisites CS352 or by instructor consent

Instructor Professor Zhiyuan Li

Purdue Email Address ci@purdue.edu

Course Contents:

GPU architecture, multithreads hardware, memory hierarchy, applications targeted by GPUs, CPU-GPU heterogeneous execution model, CUDA language and the OpenCL programming model, OpenCL compiler tool kit, OpenMP and the OpenACC programming model, compilation challenged with OpenACC targeting GPUs, compilation challenges with automatic generation of CUDA code from traditional languages such as C.

Learning Outcomes and Course Organization

The course will begin with a set of lectures given by the instructor that introduce the fundamentals that prepare the students for the exploration of the key issues that impact GPU performance and understand the nature of programs for which GPU offers distinctive performance advantage over CPUs, with the following main concepts:

- GPU architecture
    - Multithreads hardware
    - memory hierarchy and latency

- Applications targeted by GPUs and why the performance advantage over CPUs
    - Program characteristics of such applications and implication to the compilers
- Heterogeneous execution with GPU as performance accelerator for the CPU
- A brief overview of the CUDA language and the OpenCL programming model

With this preparation, students will start the effort of assembling a set of sample application programs and kernels that can be used to analyze their suitability for GPU-based performance acceleration. Students will present their findings by **giving presentations in the class and submitting summaries based on the feedback on their presentations.**

During this time, additional lectures will be given by the instructor to further introduce the predominant programming languages for parallel execution in general, and for GPU execution in particular:

- Mapping the CUDA language components and the OpenCL model to GPU hardware
- An open-source OpenCL compiler tool kit
- A higher level and more general parallel programming abstraction based on OpenMP and the OpenACC programming model,

Students will start the effort of porting and rewriting a set of sample application programs and kernels to run on the NVIDIA GPU-based system (available on Purdue RCAC's Scholar). Students will present performance results and performance analysis (understanding the performance) by **giving presentations in the class, with the possibility to rerun the experiments based on the feedback on their presentations. Summaries are submitted at the conclusion of such experiments.**

Next, the course moves to compiler issues:

- Students will analyze the open-source OpenCL compiler tool kit (in the framework of LLVM) and understand
    - the techniques implemented in such a tool kit
    - how such techniques are designed to exploit GPU hardware
    - alternative ways to write CUDA programs and how they may impact on the performance.

Students will present their findings in class and submit summaries based on the feedback on their presentations.

- Lectures will be given by the instructor for an overview of the OpenMP and OpenACC programming model, with the focus of features suitable for GPUs.

- Students will compare such a more general model with OpenCL and discuss potential advantages in easing the programming effort for GPUs, using a set of sample applications and kernels, including those assembled previously.

- Students will explore and critique the state of the art of compilers that automatically convert OpenACC to OpenCL and identify the main program analyses required for successful conversion, again using a set of sample applications and kernels, including those assembled previously.

- Students will explore how to implement modules in the LLVM framework to transform OpenACC to OpenCL and experiment with their designs. Students will present their designs and experience with implementation of selected components

- Students will review existing literature of automatic parallelization techniques and identify techniques that can be effective when targeting a subset of the sample applications and kernels assembled previously.

Grading Criteria

- Presentations: 70% (quality of findings 50%, clarity and organization of presentations/summaries 20%)

- Compiler design and implementation experiments: 30% (quality of designs: details, feasibilities, initial implementation and experiments 20%, clarity and organization of presentations/summaries 10%)

Reference Materials

- Programming Massively Parallel Processors: A Hands-on Approach (4th Edition),

by Wen-mei W. Hwu, David B. Kirk, Izzat El Hajj. Publisher: Morgan Kaufmann

- Optimizing Compilers for Modern Architectures – A Dependence Based Approach, by Randy Allen and Ken Kennedy. Publisher: Morgan Kaufmann

- High Performance Compilers for Parallel Computing, by Michael Wolfe. Publisher: Addison-Wesley

- Selected papers