Pipeline Parallelism with OpenMP

Kathleen Chalas, Knox College

ADVISORS: Ralph Johnson, Nick Chen

PARSEC [par-seek] A unit of measure

OpenMP: A popular API that consists of compiler directives used to enable shared-memory parallel programming. It’s mostly used in high-performance computing and supercomputing. Its forte is loop parallelism, but it is not very good with irregular programs. It has no concurrent data structures. You have to build them up from STL another.

WHY PIPELINES?


WHY PIPELINES?

Pipeline: A pipeline is a design construct that, when implemented, allows multiple stages to run concurrently.

Goals

#1: Use only features currently in OpenMP to see if it can support pipelines.
#2: Do this in an object-oriented manner to allow more modular design.

A micro-benchmark was used first in order to implement a pipeline with OpenMP. This benchmark had three stages.

State Design Pattern

Each object inherits from State. New objects are generated and put in a queue.

Generate Class

• Generates a number

Transform Class

• Waits n seconds

• Squares a number

Print Class

• Prints the number

Threads take first object in queue and call next() on it, thus changing it into the following stage.

Fundamental Questions/Challenges

Limitations of OpenMP

Can’t cancel threads or put them to sleep; end up having to poll the queue for more work while spinning in a busy-waiting cycle.

Can’t put conditions on OpenMP #pragma lines. You need to know exactly where you’ll want to spawn threads before running the program.

A list of things are implementation-dependent. What works with one compiler may not work with another.

A lot of things are implementation-dependent. What works with one compiler may not work with another.

Because there are no concurrent data structures, the self-implemented ones can become bottlenecks because access to these structures has to be protected.

OpenMP is very basic. It has no concurrent data structures. You have to build them up from STL libraries.

Irregular algorithms or other forms of parallelization like the pipeline require a lot of additional code and transformation. They also require multiple critical sections because of data race possibilities.

The pipeline is easy to understand conceptually and used in many modern applications, such as Intel’s RMS, compression, image processing, and video processing.

Related Work/Interaction with Other Projects

1. S. MacDonald, D. Szafron, and J. Schaeffer, “Rethinking the Pipeline as Object-Oriented States with Transformations,” HP 2004.
