CMSC 714
Lecture 5
UPC and OpenACC

Alan Sussman

Notes

- MPI project due 2 weeks from yesterday, Sept. 26
  - any questions about project spec, or running on zaratan cluster?
- Readings posted through next week
- Don't forget to send questions for readings
  - additional readings posted for next week, with who should send questions

UPC

- Extension to C for parallel computing
  - a **P**artitioned **G**lobal **A**ddress **S**pace (**PGAS**) language
  - others include Titanium (Java), Co-Array Fortran (part of the current Fortran standard), and Chapel (from Cray)
- Target Environment
  - Distributed memory machines
  - Cache Coherent multi-processors (so multi-core processors)
- Features
  - Explicit control of data distribution
  - Includes parallel for statement

UPC

- Characteristics
  - Local memory, shared arrays accessed by global pointers
  - Parallelism : single program on multiple nodes (SPMD)
  - Provides illusion of shared one-dimensional arrays
  - Features
    - Data distribution declarations for arrays
    - Cast global pointers to local pointers for efficiency
    - One-sided communication routines (memput / memget)
  - Compilers translate global pointers, generate communication
- Example

```
shared int *x, *y, z[100];
upc_forall (i = 0; i < 100; i++) {   z[i] = *x++ * *y++;  }
```

UPC Execution Model

- SPMD-based
  - One thread per process
  - Each thread starts with same entry to main
- Different consistency models possible
  - “strict” model is based on sequential consistency
  - “relaxed” based on release consistency

Forall Loop

- Forms basis of parallelism
- Add fourth parameter to for loop, “affinity”
  - Where code is executed is based on “affinity”
- Lacks explicit barrier before/after execution
  - Differs from OpenMP
- Supports nested forall loops

Split-phase Barriers

- Traditional Barriers
  - Once enter barrier, busy-wait until all threads arrive
- Split-phase
  - Announce intention to enter barrier (upc_notify)
  - Perform some **local** operations
  - Wait for other threads (upc_wait)

- Advantage
  - Allows work while waiting for processes to arrive
- Disadvantage
  - Must find work to do
  - Takes time to communicate both notify and wait

Additional info

- Implementations available at https://upc.lbl.gov/
  - And lots of other documentation and research
- Another active PGAS language is Chapel, from Cray/HPE
  - C-style too, but a new language with some new ideas and constructs, and an ongoing community project led by Cray
  - More info at https://chapel-lang.org/

OpenACC

Overview

- Like OpenMP, a set of *directives* to specify code and data to offload to an accelerator (typically a GPU)
  - for Fortran, C, C++
- Compiler then does a lot of the grunt work to run code on the accelerator with help from the host
  - initialize the device and its runtime environment
  - allocate data on the device
  - move data from host memory to device memory, or initialize it on device memory
  - launch one or more computational *kernels* on the device
  - gather results from device memory back to host memory
  - deallocate data on device

Programming model

- What to parallelize
  - an outer fully parallel loop (or loop nest, over a multi-dimensional domain), called *gangs* in OpenACC
    - no synchronization between threads in different gangs
  - and an inner synchronous (SIMD/vector) loop level (also can be multi-dimensional, so a loop nest)
    - explicit synchronization supported at this level
- On an NVIDIA GPU, each gang maps to one streaming multiprocessor (as for a CUDA thread block)
  - and the inner loops map to threads within a gang executed as a group on the cores in one streaming multiprocessor

OpenACC Constructs/Directives

- Data construct
  - defines a code region where data (arrays, subarrays, scalars) should be allocated on the device
  - with clauses to decide whether data is copied to/from host memory or just allocated on device
  - similar directives to have such info scoped across function calls, and to synchronize with the host while executing on the device
- Kernels construct
  - specifies a code region to be compiled into one or more accelerator kernels, executed in sequence
  - can take data clauses to also specify the data to allocate on the device for the kernels
  - loop construct inside a kernels construct says what type of parallelism to use to execute a loop (i.e. gangs/vectors)

OpenACC Constructs (cont.)

- Parallel construct
  - For more explicit user-specified parallelism
  - immediately starts the requested number of gangs, with the specified number of worker threads
    - then, like OpenMP parallel construct, all workers (as a set of threads) in a gang execute the code in the parallel construct, until they reach a loop construct, where each worker then executes a subset of the loop iterations
  - kernels construct gives compiler (or programmer) more flexibility in scheduling loops and decomposing iterations across gangs/workers

Summary

- For more info on OpenACC, see www.openacc.org
- Current version is 3.2, from November 2021
- Compilers available from PGI (now part of NVIDIA), Cray/HPE, AMD, several open source from universities/DOE labs/etc.
  - And the philosophy lives on in recent versions of OpenMP with accelerator directives