# CMSC 714
# Lecture 6
# MPI vs. OpenMP and OpenACC

Alan Sussman

# Notes

- MPI project due Friday, 6PM
  - Questions on project?
- OpenMP project posted after MPI project due date
- More readings posted
  - Don’t forget to send questions if you are assigned

# OpenMP + MPI

- Some applications can take advantage of both message passing and threads
  - Questions is what to do to obtain best overall performance, without too much programming difficulty
  - Choices are all MPI, all OpenMP, or *both*
    - For *both*, common option is outer loop parallelized with message passing, inner loop with directives to generate threads
- Applications studied:
  - Hydrology – CGWAVE
  - Computational chemistry – GAMESS
  - Linear algebra – matrix multiplication and QR factorization
  - Seismic processing – SPECseis95
  - Computational fluid dynamics – TLNS3D
  - Computational physics - CRETIN

# Types of parallelism in the codes

- For message passing parallelism (MPI)
  - Parametric – coarse-grained outer loop, essentially task parallel
  - Structured domains – domain decomposition with local operations – structured and unstructured grids
  - Direct solvers – linear algebra, lots of communication and load balancing required – message passing works well for large systems of equations
- Shared memory parallelism (OpenMP)
  - Statically scheduled parallel loops – one large, or several smaller loops, non-nested parallel
  - Parallel regions – merge loops into one parallel region to reduce overhead of directives
  - Dynamic load balanced – when static scheduling leads to load imbalance from irregular task sizes

# CGWAVE

- Finite elements - MPI parameter space evaluation at outer loop, OpenMP sparse linear equation solver in inner loops
- Speedup using 2 levels of parallelism allows modeling larger bodies of water in a reasonable amount of time
- Master-worker strategy for dynamic load balancing in MPI part/component
- Solver for each component solves large sparse linear system with OpenMP to parallelize
- On SGI Origin 2000 (distributed shared memory machine), use first touch rule to migrate data for each component to the processor that uses it
- Performance results show that best performance obtained using both MPI and OpenMP, with a combination of MPI workers and OpenMP threads that depends on the problem/grid size
  - And for load balancing, a lot fewer MPI workers than components

# GAMESS

- Computational chemistry – molecular dynamics – MPI across cluster, OpenMP within each node
- Built on top of Global Arrays package – for distributed array operations
  - Which in turn uses MPI (paper says PVM) and OpenMP
- Linear algebra solvers mainly use OpenMP for dynamic scheduling and load balancing
- MPI versions of parts of code are complex, but can provide higher performance for large problems
- Performance results on “medium” sized problem from SPEC (Standard Performance Evaluation Corp.) are for a small system (4 8-processor Alpha machines) connected by Memory Channel

# Linear algebra

- Hybrid parallelism with MPI for scalability and OpenMP for load balancing, for MM and QR factorization
- On IBM SP system with multiple 4-processor nodes
- Studies tradeoffs of hybrid approach for linear algebra algorithms vs. only using MPI (running 4 MPI processes per node)
- Use OpenMP for load balancing and decreasing communication costs within a node
- Also helps to hide communication latency behind other operations – important for overall performance
- QR factorization results on “medium” sized matrices show that adaptive load balancing is better than dynamic loop scheduling within a node

# SPECseis95

- For gas and oil exploration
  - Uses FFTs and finite-difference solvers
- Original message passing version (in PVM) is SPMD, OpenMP starts serial then starts an SPMD parallel section
  - In OpenMP version, shared data is only boundaries, everything else local (like PVM version)
  - OpenMP calls all in Fortran – no C OpenMP compiler – caused difficulties for privatizing C global data, and thread issues (binding to processors, OS calls)
- Code scales equally well for PVM and OpenMP, on SGI Power Challenge (a DSM machine)
  - This is a weak argument, because of likely poor PVM message passing performance (in general, and especially on DSM systems)

# TLNS3D

- CFD in Fortran77, uses MPI across grids and OpenMP to parallelize each grid
- Multiple, non-overlapping grids/blocks that exchange data at boundaries periodically
- Static block assignment to processors – divide blocks into groups of about equal number of grid points for each processor
- Master-worker execution model for MPI level, then parallelize 3D loops for each block with OpenMP
  - Many loops, so need to be careful about affinity of data objects to processors across loops
- Hard to balance MPI workers vs. OpenMP threads per block – tradeoff minimizing load imbalance vs. communication and synchronization cost
- Seems to work best on DSMs, but can be done well on distributed memory systems
- No performance results!

# CRETIN

- Physics application with multiple levels of message passing and thread parallelism
- Ported onto both distributed memory system (1464 4-processor nodes) and DSM (large SGI Origin 2000)
- Complex structure, with 2 parts discussed
  - Atomic kinetics – multiple zones with lots of computation per zone – maps to either MPI or OpenMP
    - Load balancing across zones is the problem – requires complex dynamic algorithm that benefits both versions
  - Radiation transport – mesh/grid sweep across multiple zones, suitable for both MPI and OpenMP
    - Two MPI options to parallelize, which one works best depends on problem size – one needs a transpose operation for the MPI version
- No performance results

# 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 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 2.6, from November 2017
- Compilers available from PGI (now part of NVIDIA), Cray, CAPS (Exxact Corp.)