CMSC 714 Lecture 6 MPI vs. OpenMP and OpenACC

Guest Lecturer: Sukhyun Song (original slides by Alan Sussman)

### Parallel Programming with Message Passing and Directives

# MPI + OpenMP

- Some applications can take advantage of both message passing (DMP) and threads (SMP)
  - Question is what to do to obtain best overall performance, without too much programming difficulty
  - Choices are all MPI, all OpenMP, or both
  - For both, the common option is two loop levels.
    - outer loop parallelized with message passing
    - inner loop parallelized 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

### • Message passing parallelism (MPI)

- Parametric coarse-grained outer loop for task parallelism (assign different parameters to different tasks)
- Structured domains domain decomposition into structured and unstructured grids, communication among parallel tasks
- Direct solvers linear algebra (large systems of equations), lots of communication and load balancing required
- Shared memory parallelism (OpenMP)
  - Statically scheduled parallel loops one large loop w/ many subroutines, or several smaller loops
  - Parallel regions coordinates data structure access among a series of parallel loops (merge multiple loops into one parallel region to reduce overhead of thread scheduling)
  - Dynamic load balanced when static scheduling leads to load imbalance from irregular task sizes

## CGWAVE

- Hydrology problem
  - models wave motions of the sea
- Two levels of parallelism for speedup
  - MPI parameter space evaluation at outer loop
  - OpenMP sparse linear equation solver in inner loops
- Boss-worker strategy for dynamic load balancing
  - The boss process communicates with worker processes.
  - The strategy breaks down as # worker processes approaches total # parameter configurations. (Q)
- Performance results (Figure 1)
  - The best performance obtained when both MPI and OpenMP are used. (16 MPI workers and 4 OpenMP threads)

# GAMESS

- Computational chemistry
  - MPI across compute nodes, OpenMP within each node
- Run on top of Global Arrays library
  - for distributed array operations
  - The library uses MPI (paper says PVM) and OpenMP.
- Linear algebra solvers mainly use OpenMP
  - simpler than MPI code
- MPI provides high performance for large problems
  - can use a lot of processors in a distributed memory system
  - complicated code vs. high performance
- Performance results (Table 2)
  - "medium" sized SPEC benchmark
  - 32 CPUs speedup 5.11x over 4 CPUs. (Q: ideal speedup?)

## Linear algebra study

- MM (Matrix-Matrix multiplication), QR factorization
  - MPI (across compute nodes) for scalability
  - OpenMP (within each node) for load balancing
- Parallelize MM computation
  - Divide matrices by columns
  - Broadcast and compute sub-matrix
- Communication hiding
  - place the MPI broadcast outside OpenMP parallel region
  - overlap communication (broadcast) with computation
- Adaptive load-balancing
  - A communication thread takes a smaller matrix block.
- Performance results (Table 3)
  - "Hide" shows higher performance (MFLOPS) than "No Hide".
  - adaptive load-balancing increases performance

## SPECseis95

### • Seismic processing benchmark

- For gas and oil exploration
- FFTs (Fast Fourier Transforms) and finite-difference solvers

### • Two parallel versions

- Original message-passing variant (PVM or MPI)
- Conversion to OpenMP variant
  - Some issues about mixing C and Fortran codes

#### • Performance results (Figure 4)

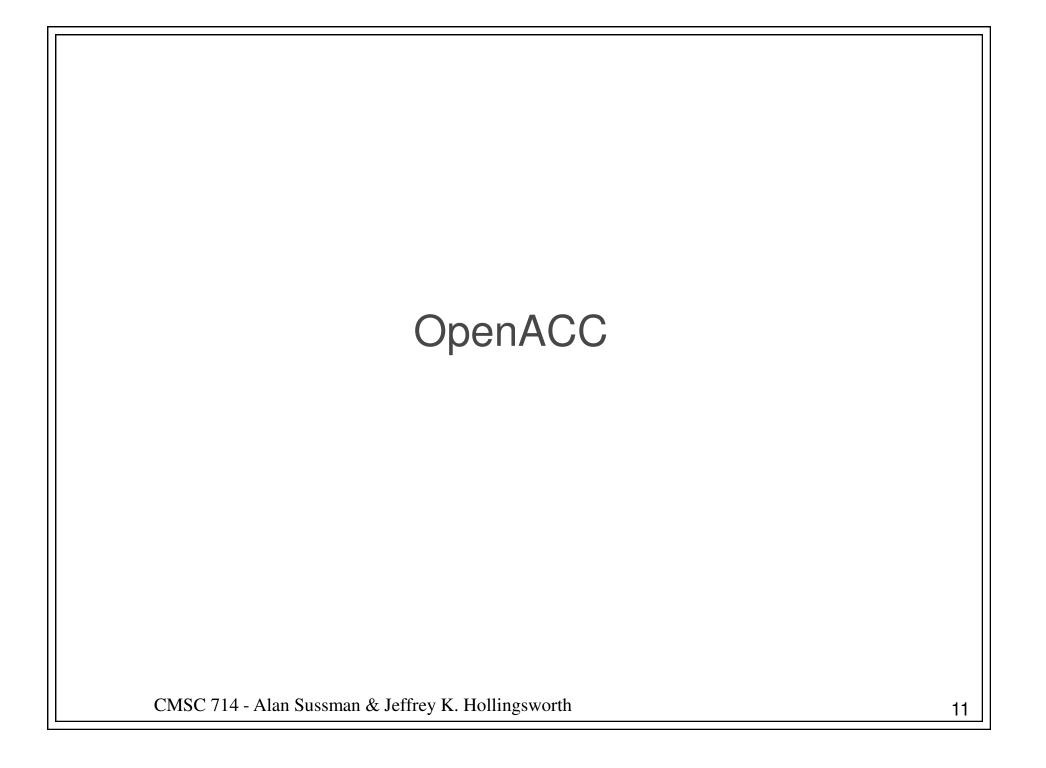
 Code scales equally well for PVM and OpenMP, on SGI Power Challenge

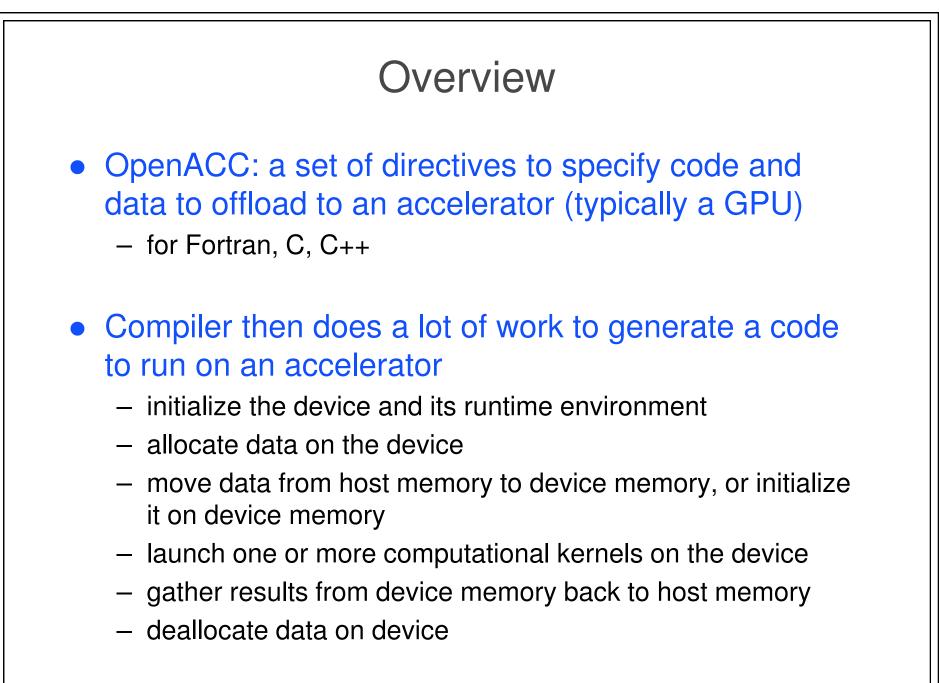
# TLNS3D

- CFD (computational fluid dynamics)
  - MPI across grids and OpenMP to parallelize each grid
- Input data sets contain multiple data blocks
- Static block assignment to MPI processes
  - divide blocks into groups, assign a group to an MPI process
  - MPI processes exchange data at boundaries periodically.
- Boss-worker execution model for MPI level
  - Boss performs I/O, workers do numerical computations.
- Add OpenMP directives
  - Exploit parallelism within each block
- Minimizing load imbalance vs. synchronization cost
  - Need to adjust # MPI processes and # OpenMP threads
- No performance results in the paper!

# CRETIN

- Physics application
  - multiple levels of message passing and thread parallelism
- Systems
  - IBM SP2 with 1464 four-processor nodes
  - SGI Origin 2000 with 48 128-processor nodes
- Atomic Kinetics
  - multiple zones with lots of computation per zone
  - map the loop over zones to either MPI or OpenMP
  - load balancing across zones (10<sup>5</sup>x)
- Line Radiation Transport
  - mesh sweep across multiple zones
  - use both MPI and OpenMP
  - boss performs memory allocation, passes zones to workers
- No performance results





## Programming model

### • Two loop levels

- an outer (fully parallel) loop level, called gang in OpenACC
  - no synchronization between threads in different gangs
- an inner synchronous (SIMD/vector) loop level
  - synchronization required

### • On an NVIDIA GPU

- each gang maps to one stream multiprocessor (a CUDA thread block)
- the inner loops map to threads within a gang executed as a group on the cores in one stream multiprocessor

## **OpenACC** Directives

### • Data construct

- !\$acc data ... (Fortran)
- 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

### • Kernels construct

- !\$acc kernels ...
- specifies a code region to be compiled into accelerator kernels (computation code)
- Using a loop construct inside a kernels construct, we can specify what type of parallelism to use to execute a loop (i.e. gangs/vectors)

## OpenACC Directives (cont.)

### • Parallel construct

- !\$acc parallel ...
- similar to OpenMP directives
- for more explicit user-specified parallelism
- immediately starts the requested number of gangs, where each gang contains a specified number of worker threads
  - Workers in each gang execute code redundantly
  - When they reach (\$!acc loop worker), workers parallelize loop iterations
- Kernels construct vs Parallel construct
  - kernels construct gives compiler more flexibility in scheduling loops and decomposing iterations across gangs/workers
  - But in kernels construct, loops need to be tightly nested for the compiler to be able to generate good code