CMSC 714
Lecture 7
MPI w/OpenMP and PETSc

Alan Sussman

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
- Boss-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
- Boss-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

PETSc

- Portable, Extensible Toolkit for Scientific Computation
- Library to encapsulate commonly used functions and data structures for numerically solving partial differential equations
- Targeted at message passing for scalability, but hides it (mostly) from application
- Uses object-oriented programming techniques
    - Data encapsulation
    - Polymorphism
    - Inheritance
    - but implemented in C, so no compiler support
- Essentially SPMD style programming, but w/o explicit message passing

6 guiding principles

- For performance
  - overlap communication and computation
  - determine details of repeated communication patterns, and optimize message passing across multiple calls (inspector/executor model)
  - allow user to decide when communication occurs (if needed)
  - allow user to aggregate data for later communication
- For ease of use
  - allow user to work on distributed objects (arrays) without knowing which processor owns which data elements
  - manage communication at higher levels, on objects, instead of directly using message passing

Distributed Objects

- Low level data structures
  - Vectors
  - Matrices
  - Index Sets
- Low level algorithms
  - Create and assemble a vector or matrix – vector scatter/gather, sparse matrix examples in paper
- Higher level algorithms
  - PDE solvers
  - Linear and non-linear equation solvers
  - Time steppers
  - Preconditioners
- All functions take an MPI_Comm as an argument

Six Guiding Principles (again)

- Managing communication within higher level data structures and algorithms
  - MPI calls generated to perform communication needed to perform higher level ops on distributed objects
  - Implication is no optimizations across calls
- Overlap communication and computation
  - Separate start and end of complex operations, so other computations can go on in between, like MPI non-blocking operations
- Precomputing communication patterns
  - Generate a pattern of sends/receives for an operation on a distributed object (which may need communication), then reuse the pattern for subsequent data movement operations
  - Often called inspector/executor model

Guiding Principles (cont.)

- Programmer management of communication
  - User can explicitly start and end communication via specific PETSc calls
  - Often to enable overlap of communication with computation
- Work on distributed objects, not on individual data elements
  - Avoids programmer having to move data between application data structures and library data structures
  - Can build PETSc data structures from any process, with data for any process (not just local to a process)
    - This is what is meant by "assembly"
- Aggregate data for communication
  - To minimize number of messages
  - Communication cost proportional to number of messages, plus per byte cost

PETSc status

- Current version is 3.22
  - See https://petsc.org
  - Integrated with TAO library for large-scale optimization problems
- GPU support is available
  - Through CUDA for NVIDIA GPUs
  - Through OpenCL,HIP, Kokkos for AMD and Intel GPUs
- Interfaces for C, C++, Fortran, Python