CIS 631
Parallel Processing

Lecture 16: Parallel Application Design, PETSc, and CCA

Allen D. Malony
malony@cs.uoregon.edu

Department of Computer and Information Science
University of Oregon
Acknowledgements

Outline

- General parallel application issues
- PETSc
- CCA
Questions for Applications and Parallelization

- Should it be parallelized?
  - What type of application is it? How is its structure?
- Appropriate target hardware architectures?
  - May determine the approach for parallelization
- What is known about the algorithms?
  - Are there alternatives more suitable to target platform?
- What software technologies are recommended?
  - Programming languages, libraries, tools, …
  - This is an important issue since it determines other things
- Identify general characteristics useful for classifying issues in parallelization
What are the Applications Goals?

- Portability
  - Across parallel systems, operating systems, software, …
- Reusability
  - Providing interfaces for others to incorporate
- Interoperability
  - Works with other software tools and applications
- Performance
  - Performance portability
- Scalability
- Accuracy and stability
Application Types

- Geometrical, numerical, search, …
- Discrete or continuous
- Dynamical systems (particle, continuous)
- Useful to think of application as a “complex system”
  - Linked set of entities
  - Parallel issues depend on application type, BUT it also depends on overall application characteristics
- Often find similar issues for different applications
- This way of thinking can relate the parallelization strategies of seemingly very different problems
Application Properties

☐ Problem representation
  ☑ Natural or geometric
  ☑ Spatial approximations
  ☑ Varying dimensionality: scale, resolution, adaptivity

☐ Decomposition
  ☑ Regular or irregular
  ☑ Flat or hierarchical
  ☑ Linked across dimensions

☐ Simulation
  ☑ Time-stepped versus event-driven
  ☑ Affects synchronization
Temporal Structure of Applications

- Numerical systems can be classified in four broad areas:
  - **Synchronous**
    - system advances in lock step (approximates SIMD)
    - synchronization is typically coordinated point-by-point
  - **Loosely synchronous**
    - synchronous within sub-domains
    - natural form of SPMD implementations (most like this)
    - appears with irregularities or hierarchy in decomposition
  - **Asynchronous**
    - event-driven system, not formulated as time-step or iteration
    - harder to parallelize
  - **Pleasingly parallel**
Parallelization of Basic Complex Systems

- Problems set up as computational or numerical systems
- Characterized by dynamic, spatial, and temporal structure

*Diagram*

- Mapping of original problem to structure of computational system
- Mapping back of computational results to problem domain
Past Experience in Parallelization Applications

- Synchronous or loosely synchronous problems perform well on parallel machines if the problem is large enough.
- For a given machine, there is a typical sub-domain size.
  - Part of the problem stored on each node
  - Grain size
- Above a particular sub-domain size, one can expect to achieve good performance.
  - Roughly constant ratio of parallel speedup to number of processors
  - Scale problem with respect to fixed sub-domain size
  - Scale total size proportional to number of processors
Other Things to Think About

- Input and output
  - How does this affect performance?
- Batch or interactive
  - Affects the execution structure
- Visualization
  - May require access to large application data online
- Requirements for time to solution
- Single execution case or replicated
  - Parametric analysis can lead to large number of executions
- Checkpoint and restart
- Resource requirements and access
Application Libraries

- Encapsulate some set of functionality in routine library
  - Common data structures, algorithms, I/O, graphics, communication, …
  - Numerical LA, graphs, meshes, PDEs, optimization, …
- Encapsulate algorithms and their parallelization
- Support portability
- Raise the level of parallel programming
  - High-level interface to specialized capabilities
  - Gain benefit of expertise, engineering, and optimization
- Hides implementation
  - May be at expense of reduced observability
Problem Solving Environments

- Application specific environments
- Capture whole process of problem solving
  - “Soup to nuts” support for process
    - data management
    - input data and preprocessing
    - parallel execution (single and replicated)
    - interaction and visualization
    - results output and management
    - feedback and steering
- “Domain specific” problem solving environments
  - More generally defined to allow extension and integration
  - Problem solving workbench support
TIERRA (Computational Science Institute, UO)

- Tomographic Imaging Environment for Ridge Research and Analysis (Cuny, Toomey, et al.)
- High-performance, domain-specific environment for seismic tomography
  - parallelized tomography code
  - runtime distributed array access
  - computational steering via MatLab frontend
  - full problem solving process for seismic tomography
- Led to new discoveries for three-dimensional melt migration beneath the East Pacific Rise
Scientific Problem Solving “Workbench”

- Integrated application development environment
  - “Component-based” application programming
  - High-level data, control, and interaction objects
- Raises level of application programming to problem-specific abstraction
  - Relieves developer from low-level parallel programming
  - Captures problem semantics in programming abstractions
  - Richer data object and operational components
  - Provides support for extension (e.g., Matlab toolboxes)
- Retargetable to related application problem domains
SCIRun (Johnson, University of Utah)

- Scientific programming environment
  - large-scale simulations
  - “computational workbench”
  - visual programming interface
  - dataflow model of computing
    - modules: operation or algorithm with I/O ports
    - network: set of modules and their interconnections
    - widgets: 3D user interaction
  - data types: Mesh, Surface, Matrix, Field, Geometry
  - extensible module library
  - computational steering
SCIRun User Interface

- Visual programming lets users select, arrange, and connect modules into a desired network.

- Interactive steering of design, computation, and visualization allows more rapid convergence.
Meta-Problems and Meta-Applications

- Problems may be more complex, requiring the coordinated solution of several problem components
  - Simulation of physical systems often involve multiple models to be solved simultaneously
- Meta-applications to solve meta-problems are built up from multiple application modules
  - Modules are implemented separately and interact through interfaces defined with respect to problem semantics
  - Linkage between modules is often timed asynchronously
- Meta-applications require richer software architectures and more robust technologies
  - Modules may be implemented very differently
  - Implementation on computational grids is possible
Scalable Solution
of PDE-based Applications Using PETSc

Lois Curfman McInnes
Mathematics and Computer Science Division
Argonne National Laboratory

Seminar
National Institute of Standards and Technology
November 5, 2002
Outline

- Motivation
  - Complex, multi-physics, multi-scale applications
  - Distributed, multi-level memory hierarchies

- Parallel Software for PDEs
  - Approach
  - Performance

- Next Generation Software
  - What are components?
    - Common Component Architecture (CCA) Forum

- Ongoing Challenges
Acknowledgements

- U.S. Department of Energy – Office of Science
  - Core funding in the MCS Division of Argonne through the Mathematical, Information, and Computational Science (MICS) program
  - Advanced Computational Testing and Simulation (ACTS) toolkit
  - Scientific Discovery through Advanced Computing (SciDAC) program

- National Science Foundation
  - Multi-Model Multi-Domain Computational Methods in Aerodynamics and Acoustics
Motivating Scientific Applications

- Optimization
- Derivative Computation
- Molecular structures

- Physics
- Meshes
- Discretization
- Algebraic Solvers
- Data Redistribution
- Parallel I/O

- Adaptive Solution
- Diagnostics
- Steering
- Visualization
- Astrophysics
- Aerodynamics
- Fusion

Lecture 16
Target Architectures

- Systems have an increasingly deep memory hierarchy
- Time to reference main memory 100’s of cycles

![Diagram showing the memory hierarchy with CPU, Cache, Main Memory, SMP, and Interconnect]
Computational Challenges

- Community Perspective
  - Life-cycle costs of applications are increasing
    - Require the combined use of software developed by different groups
    - Difficult to leverage expert knowledge and advances in subfields
    - Difficult to obtain portable performance

- Individual Scientist Perspective
  - Too much energy focused on too many details
    - Little time to think about modeling, physics, mathematics
    - Fear of bad performance without custom code
    - Even when code reuse is possible, it is far too difficult

- Our Perspective
  - How to manage complexity of numerical tools?
    - Numerical software tools that work together
    - New algorithms (interactive/dynamic, algorithm composition, …)
    - Multi-model, multi-physics simulations
Algorithmic needs of targeted applications?

- Large-scale, PDE-based applications
  - Multi-rate, multi-scale, multi-component

- Need
  - Fully or semi-implicit solvers
  - Multi-level algorithms
  - Support for adaptivity
  - Support for user-defined customizations (e.g., physics-informed preconditioners, transfer operators, and smoothers)

- Our focus
  - Algorithms
  - Analysis
  - Software – design, implementation, optimization
  - Applications
Interface Issues

- How to hide complexity, yet allow customization and access to a range of algorithmic options?
- How to achieve portable performance?
- How to interface among external tools?
  - including multiple libraries developed by different groups that provide similar functionality (e.g., linear algebra software)
- Criteria for evaluation of success
  - efficiency (both per node performance and scalability)
  - usability
  - extensibility
PETSc ([http://www.mcs.anl.gov/petsc](http://www.mcs.anl.gov/petsc))

- **Portable, Extensible Toolkit for Scientific Computation**
- Balay, Buschelman, Gropp, Kaushik, Knepley, McInnes, Smith, Zhang, Argonne National Laboratory
- Targets the parallel solution of large-scale PDE-based applications
- Over 50 publications about computations with PETSc
- Freely available and supported research toolkits
  - Hyperlinked documentation, many examples
  - Usable from Fortran 77/90, C, and C++
- Portable to any parallel system supporting MPI
  - Tightly coupled systems and loosely coupled systems
Programming Model

- Goals
  - Portable, runs everywhere
  - Performance
  - Scalable parallelism

- Approach
  - Distributed memory, “shared-nothing”
    - Requires only a compiler (single node or processor)
    - Access to data on remote machines through MPI
  - Can still exploit “compiler discovered” parallelism
  - Hide communication details within parallel objects
  - User orchestrates communication at a higher abstract level
PDE Application Codes

- Computation and Communication Kernels
  - MPI, MPI-IO, BLAS, LAPACK
- Profiling Interface
- Grid Management
- Object-Oriented Matrices, Vectors, Indices
- Linear Solvers Preconditioners + Krylov Methods
- Nonlinear Solvers, Unconstrained Minimization
- ODE Integrators
- Visualization
- Interface
- PETSc PDE Application Codes
### PETSc Numerical Libraries

#### Nonlinear Solvers
- Newton-based Methods
- Line Search
- Trust Region
- Others

#### Time Steppers
- Euler
- Backward Euler
- Pseudo Time Stepping
- Others

#### Krylov Subspace Methods
- GMRES
- CG
- CGS
- Bi-CG-STAB
- TFQMR
- Richardson
- Chebychev
- Others

#### Preconditioners
- Additive Schwartz
- Block Jacobi
- Jacobi
- ILU
- ICC
- LU (Sequential only)
- Others

#### Matrices
- Compressed Sparse Row (AIJ)
- Blocked Compressed Sparse Row (BAIJ)
- Block Diagonal (BDIAG)
- Dense
- Matrix-free
- Others

#### Distributed Arrays
- Vectors

#### Index Sets
- Indices
- Block Indices
- Stride
- Others
Sample: Nonlinear Solvers

- **Goal**
  - For problems arising from PDEs, support the general solution of \( f(u) = 0 \), where \( f: \mathbb{R}^n \rightarrow \mathbb{R}^n \)

- **User provides:**
  - Code to evaluate \( f(u) \)
  - Code to evaluate Jacobian of \( f(u) \) (optional)
    - or use sparse finite difference approximation
    - or use automatic differentiation (AD)
      - see [http://www.mcs.anl.gov/autodiff](http://www.mcs.anl.gov/autodiff)
Nonlinear Solvers (SNES)

- Newton-based methods, including
  - Line search strategies
  - Trust region approaches
  - Pseudo-transient continuation
  - Matrix-free variants
- Single solver interface, supporting
  - Uniprocessor and parallel versions
  - All algorithms for a particular problem class
  - Real and complex arithmetic
- User can customize all phases of the solution process
Nonlinear PDE Solution

Application Driver

Nonlinear Solvers (SNES)

Linear Solvers (SLES)

PC
KSP

Solve $F(u) = 0$

PETSc

Function Evaluation
Jacobian Evaluation
Post-Processing

Application Initialization

User code  PETSc code
Basic Nonlinear Solver Code (C/C++)

```c
SNES  snes;   /* nonlinear solver context */
Mat    A;      /* Jacobian matrix */
Vec    x, F;   /* solution, residual vectors */
int    n, its; /* problem dimension, number of iterations */
ApplicationCtx usercontext; /* user-defined application context */

... MatCreate(MPI_COMM_WORLD,n,n,&J);
VecCreate(MPI_COMM_WORLD,n,&x);
VecDuplicate(x,&F);

SNESCreate(MPI_COMM_WORLD,SNES_NONLINEAR_EQUATIONS,&snes);
SNESSetFunction(snes,F,EvaluateFunction,usercontext);
SNESSetJacobian(snes,J,EvaluateJacobian,usercontext);
SNESSetFromOptions(snes);
SNESolve(snes,x,&its);

MatDestroy(J); VecDestroy(x); VecDestroy(F); SNESDestroy(snes);
```
Uniform Access to Solvers

- Procedural interface
  - SNESSetType(snes,”ls”);
  - SNESSetLineSearch(snes,SNESQuadraticLineSearch);
  - ...

- Run-time interface
  - -ksp_type [cg,gmres,bcgs,tfqmr,…]
  - -pc_type [lu,ilu,jacobi,sor,asm,…]
  - -snes_type [ls,tr,…]
  - -snes_line_search <line search method>
  - -sles_ls <parameters>
  - -snes_convergence <tolerance>
  - ...

CFD on an Unstructured Mesh

- 3D incompressible Euler
- Tetrahedral grid
- Up to 11 million unknowns
- Based on a legacy NASA code, FUN3d, developed by W. K. Anderson
- Fully implicit steady-state
- Primary PETSc tools: nonlinear solvers (SNES) and vector scatters (VecScatter)

Results courtesy of D. Kaushik and D. Keyes
Fixed-size Parallel Scaling Results

Dimension = 11,047,096

Aggregate Gflop/s
vs. # nodes
Fixed-size Parallel Scaling Results

Execution Time (s) vs. # nodes

- Asci Blue
- T3E
- Asci Red
ONERA M6 wing test case, tetrahedral grid of 2.8 million vertices (about 11 million unknowns) on up to 3072 ASCI Red nodes (each with dual Pentium Pro 333 MHz processors)
One-to-One Interfacing with PETSc

- Linear solvers
  - BlockSolve95  [http://www.mcs.anl.gov/BlockSolve95](http://www.mcs.anl.gov/BlockSolve95)
  - DSCPACK  [http://www.cse.psu.edu/~raghavan/dscpack](http://www.cse.psu.edu/~raghavan/dscpack)
  - Hypre  [www.llnl.gov/casc/hypre](http://www.llnl.gov/casc/hypre)
  - Spooles  [http://www.netlib.org/linalg/spooles](http://www.netlib.org/linalg/spooles)
  - SuperLU  [http://www.nersc.gov/~xiaoye/SuperLU](http://www.nersc.gov/~xiaoye/SuperLU)
One-to-One Interfacing with PETSc

- Optimization software
  - TAO  http://www.mcs.anl.gov/tao
  - Veltisto  http://www.cs.nyu.edu/~biros/veltisto

- Mesh and discretization tools
  - Overture  http://www.llnl.gov/CASC/Overture
  - SAMRAI  http://www.llnl.gov/CASC/SAMRAI
  - SUMAA3d  http://www.mcs.anl.gov/sumaa3d

- ODE solvers
  - PVODE  http://www.llnl.gov/CASC/PVODE

- Others
  - Matlab  http://www.mathworks.com
  - ParMETIS  http://www.cs.umn.edu/~karypis/metis/parmetis
What are components?

- OO techniques are useful for building individual components by relatively small teams; component technologies facilitate sharing of code developed by different groups by addressing issues in
  - Language interoperability
    - via interface definition language (IDL)
  - Well-defined abstract interfaces to enable “plug-and-play”
  - Dynamic composability
    - components can discover information about their environment (e.g., interface discovery) from framework and connected components

Desire to build scientific applications by hooking together components

DOE Common Component Architecture (CCA) provides a mechanism for interoperability of high-performance components developed by many different groups in different languages or frameworks.

Existing component architecture standards such as CORBA, Java Beans, and COM do not provide support for parallel components.

Latency between components:
- MPI
- CCA
- CORBA/Java

Latency:
- $10^{-6}$ sec
- $10^{-4}$ sec
- $10^{-1}$ sec
- 1 sec
CCA History and Participants

- 1998: CCA Forum originated
  - Participation from researchers who were exploring one-to-one software interfacing in the DOE ACTS Toolkit program
  - Open to everyone interested in HPC components
  - See http://www.cca-forum.org

- 2001: Center for Component Technology for Terascale Simulation Software (CCTTSS) founded
  - Support from the DOE SciDAC Initiative
  - CCTTSS team is a subset of the CCA Forum
  - Leader: Rob Armstrong (SNL)
  - See http://www.cca-forum.org/ccttss
High-Performance Component Architecture

- Simple and flexible
  - To adopt
  - To understand
  - To use
- Support a composition mechanism that does not impede high-performance component interactions
- Permit the SPMD paradigm in component form
- Meant to live with and rely on other commodity component frameworks to provide services, etc.
  - JavaBeans, CORBA, …
CCA Approach

- CCA specification dictates a basic set of interfaces (and corresponding behaviors)
  - Ports define the connection model for component interactions
  - Provides/Uses design pattern
- Components can be manipulated in framework
- CCA specification does not dictate frameworks or runtime environment.
  - Create components usable in all frameworks
  - Provide a means for discovering interfaces
  - Exclude how the components are linked
  - Provide language-independent means for creating components (via SIDL)
CCA Framework Prototypes

- CCAFFEINE
  - SPMD/SCMD parallel, direct connect
  - Direct connection

- CCAT / XCAT
  - Distributed network
  - Grid Web services

- SCIRun
  - Parallel, multithreaded, direct connect

- Decaf
  - Language interoperability via Babel

- Legion (under development)
Component Wiring Diagram

- Using GUI tool within CCAFFEINE framework

- Black boxes: components
- Blue boxes: provides ports
- Gold boxes: uses ports
CCA Concept of SPMD Components

MPI application using CCA for interaction between components A and B within the same address space

Direct Connection supplied by framework at compile/runtime

Adaptive mesh component written by user 1

Solver component written by user 2

Process

Proc1

Proc2

Proc3

etc...
Collective Port Modularizes Decomposition

Combining previous parallel component with another parallel component in a different framework

container composed of mesh and solver components

parallel visualization component

collective port connecting M procs with N procs
Intra- and Inter-component performance engineering

Four general parts:

- Performance observation
  - integrated measurement and analysis
- Performance query and monitoring
  - runtime access to performance information
- Performance control
  - mechanisms to alter performance observation
- Performance knowledge
  - characterization and modeling

Consistent with component architecture / implementation
**TAU-based CCA Performance Component**

- Two instrumentation paths using TAU API

- Two query and control paths using TAU API

- Alternative implementations of performance component

---

Lecture 16  53  CIS 631 - Parallel Processing
Flame Reaction-Diffusion Demonstration

CCAFFEINE
Meeting CCA Performance Engineering Goals?

- Language interoperability?
  - SIDL and Babel give access to all supported languages
  - TAU supports multi-language instrumentation
  - Component interface instrumentation automated with PDT

- Platform interoperability?
  - Implement observability component across platforms
  - TAU runs wherever CCA runs

- Execution model transparent?
  - TAU measurement support for multiple execution models

- Reuse with any CCA-compliant framework?
  - Demonstrated with SIDL/Babel, CCAFEINE, SCIRun
Meeting CCA Performance Engineering Goals?

- Component performance knowledge?
  - Representation and performance repository work to do
  - Utilize effectively for deployment and steering
  - Build repository with TAU performance database

- Performance of component compositions?
  - Component-to-component performance
    - Per connection instrumentation and measurement
    - Utilize performance mapping support
  - Ensemble-wide performance monitoring
    - connect performance “producers” to “consumers”
    - component-style implementation
CCA References

☐ Web sites

☐ CCA Forum
   ➢ http://www.cca-forum.org

☐ Center for Component Technology for Terascale Simulation Software (CCA SciDAC Center)
   ➢ http://www.cca-forum.org/ccttss

☐ Introductory paper

Next Class

- Parallel applications
- C-SAFE and Uintah