ECE 259 / CPS 221
Advanced Computer Architecture II
(Parallel Computer Architecture)

Parallel Programming

Copyright 2010 Daniel J. Sorin
Duke University

Slides are derived from work by
Sarita Adve (Illinois), Babak Falsafi (CMU/EPFL),
Mark Hill (Wisconsin), Alvy Lebeck (Duke), Steve
Reinhardt (Michigan/AMD), and J. P. Singh (Princeton).
“To understand and evaluate design decisions in a parallel machine, we must get an idea of the software that runs on a parallel machine.”

--Introduction to Culler et al.’s Chapter 2, beginning 192 pages on software
Outline

• Applications

• Creating Parallel Programs

• Programming for Performance

• Scaling

• Synchronization Basics
Applications

- **Scientific**
  - Simulation of natural phenomena (protein folding, planetary motion, molecular interactions, etc.)
  - Large mathematical problems

- **Commercial**
  - Online transaction processing (OLTP)
  - Decision support systems (DSS)
  - Web serving (e.g., Apache)
  - Application serving (a.k.a. middleware)

- **Multimedia/home**
  - Audio/video, games, word processing, speech recognition, etc.
Scientific: The SPLASH2 Benchmarks

• Kernels
  – Complex 1D FFT
  – Blocked LU Factorization
  – Blocked Sparse Cholesky Factorization
  – Integer Radix Sort

• Applications
  – Barnes-Hut: interaction of N bodies
  – Adaptive Fast Multipole (FMM): interaction of bodies
  – Ocean Simulation
  – Hierarchical Radiosity
  – Ray Tracer (Raytrace)
  – Volume Renderer (Volrend)
  – Water Simulation with Spatial Data Structure (Water-Spatial)
  – Water Simulation without Spatial Data Structure (Water-Nsquared)
Scientific: The SpecOMP Benchmarks

- Parallel scientific benchmarks based on Spec CPU
- Written in OpenMP (shared memory “library”)
  - wupwise quantum chromodynamics
  - swim shallow water modeling
  - mgrid multi-grid solver in 3D potential field
  - applu parabolic/elliptic partial differential equations
  - galgel fluid dynamics analysis of oscillatory instability
  - art neural net simulation of adaptive resonance theory
  - equake finite element simulation of earthquake modeling
  - ammp computational chemistry
  - fma3d finite-element crash simulation
  - apsi solves problems regarding temperature, wind, etc.
  - gafort genetic algorithm code
Ocean Simulation

- Simulate ocean currents
- Discretize in space and time
• Computing the mutual interactions of N bodies
  – N-body problems
  – Stars, planets, molecules…
• Can approximate influence of distant bodies
Online Transaction Processing: TPC-C

• TPC-C is a standard OLTP benchmark

• Models database transactions for company
  – Customers make orders of company
  – Company orders from suppliers to stock warehouses

• Goal: high transaction throughput

• Specifics
  – Specifies database size, number of clients, etc.
  – Does not specify implementation!
Outline

• Applications

• **Creating Parallel Programs**
  – In general
  – Two examples

• Programming for Performance

• Scaling

• Synchronization Basics
Creating a Parallel Program

• In theory, can be done by programmer, compiler, runtime system, or OS

• In practice, parallel programs created with
  – Explicitly parallel language (e.g., High Performance Fortran, CUDA)
  – Library for implementing a programming model
    » Shared memory library (POSIX, PARMACS, OpenMP)
    » Message passing library (Message Passing Interface)

• What you should realize at end of this section?
  – Parallel programming is difficult!
A Little Terminology

• A **Task** is a piece of work  
  – Ocean simulation: grid point, row, plane  
  – Apache web server: single query

• **Task granularity**  
  – Small $\rightarrow$ fine-grain task  
  – Large $\rightarrow$ coarse-grain task

• A **process** (thread) performs tasks  
  – According to OS: process = thread(s) + address space

• A **process** is executed on a **processor (core)**
Steps for Creating a Parallel Program

- **Decomposition** into tasks
- **Assignment** of tasks to processes (threads)
- **Orchestration** of data access, communication, etc.
- **Mapping** processes to processors
Decomposition

- Decompose computation into set of tasks
- Could be dynamic
- Maximize concurrency
- Minimize overhead of managing tasks
- Remember Amdahl’s Law!

\[
\text{Speedup}_{\text{overall}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}
\]
Assignment

- Assign tasks to processes (static vs. dynamic)
- Balance workload (load balancing)
- Reduce communication
- Minimize overhead
- **Assignment + Decomposition = Partitioning**
Orchestration

- Choreograph data access, communication, and synchronization
- Reduce cost of communication and synchronization
- Preserve data locality \( (\text{data layout}) \)
- Schedule tasks \( (\text{order of execution}) \)
- Reduce overhead of managing parallelism
- Must have good primitives \( (\text{architecture and model}) \)

(C) 2010 Daniel J. Sorin from Adve, Falsafi, Hill, Lebeck, Reinhardt, Singh

ECE 259 / CPS 221
Mapping

- Map processes to physical processors
- Static
- Dynamic
  - Processes migrate
  - What about orchestration (data locality)
  - Task queues
OS Effects on Mapping

- Ability to bind process to processor

- Space Sharing
  - Physical partitioning of machine

- Gang Scheduling
  - All processes context switched simultaneously
Outline

• Applications

• **Creating Parallel Programs**
  – In general
  – *Two examples*

• Programming for Performance

• Scaling

• Synchronization Basics
Data Parallel Example: Ocean Simulation

- Contains equation solver “kernel”
  - Kernel = small piece of important code (not OS “kernel”)
- Update each point based on adjacent neighbors
- Compute average difference per element
- Convergence when diff small → exit
Equation Solver Decomposition

while (!converged) {
    for { // over all points in x-dimension
        for { // over all points in y-dimension

    • The loops are not independent! But …
    • Exploit properties of problem
        – Don’t really need up-to-date values (approximation)
        – May take more steps to converge, but exposes parallelism
    • Red-Black
        – Like checkerboard: update of red point depends only on black points
        – Alternate iterations over red, then black
    • Asynchronous
        – Each processor updates its region independent of other’s values
        – Global synch at end of iteration to keep things somewhat up-to-date
Decomposition: The FORALL Statement

while (!converged) {
    forall{ // execute all iterations in parallel
        forall{ // execute all iterations in parallel
            // execute all iterations in parallel
        }
    }
}

- Data parallel execution, like in HPF (High Perf Fortran)
- Decomposition: tasks = loop iterations
  - Can execute the iterations in parallel
- Each grid point computation (n² parallelism)

while (!converged)
    forall{
        for{
            // execute all iterations in parallel
        }
    }

- Computation for rows is independent (n parallelism)
  - Less overhead
**Equation Solver Assignment**

- Each process gets a contiguous block of rows

(C) 2010 Daniel J. Sorin from Adve, Falsafi, Hill, Lebeck, Reinhardt, Singh
Writing Shared Memory Code with Pthreads

• Library of shared memory routines
  – Portable across most platforms

• Some of the included routines:
  – pthread_create and pthread_exit
  – pthread_mutex_init – create a mutex (lock)
  – pthread_mutex_lock – lock a mutex
  – pthread_mutex_unlock – unlock a mutex

• Other shared memory libraries
  – “OpenMP is a specification for a set of compiler directives, library routines, and environment variables that can be used to specify shared memory parallelism in Fortran and C/C++ programs.”
  – Solaris System V Shared Memory, PARMACS, MM for Linux
  – Goals: simplify programming, abstract away hardware
Equation Solver: The Ugly Code

main()
    A = G_MALLOC(size of big array);
    CREATE(nprocs-1,Solve, A);
    Solve(A)
    WAIT_FOR_END;
end main

Solve(A)
    while (!done)
        for i = my_start to my_end
            for j = 1 to n
                compute new_A[i,j];
                mydiff += abs(new_A[i,j] - old_A[i,j]);
            LOCK(diff_lock);
                diff += mydiff;
            UNLOCK(diff_lock);
            if (convergence_test) then done = 1
        BARRIER
SM/MP Example: Standard Cell Router

- **LocusRoute (VLSI standard cell router)**

```c
while (route_density_improvement > threshold)
{
    for (i = 1 to num_wires) do
    {
        rip old wire out
        explore new route
        place wire using best new route
    }
}
```

(C) 2010 Daniel J. Sorin from Adve, Falsafi, Hill, Lebeck, Reinhardt, Singh
Shared Memory Implementation

• Shared memory algorithm
  – Divide cost array into regions
    » Logically assign regions to processors
  – Assign wires to procs based on the region in which center lies
  – Do load balancing using stealing when local queue empty

• Pros:
  – Good load balancing on average
  – Mostly local accesses
  – High cache hit ratio

• Cons:
  – Non-deterministic (why is this bad?)
  – Potential for hot spots
Message Passing Implementations

• **Method 1:**
  – Distribute wires and cost array regions as in SM implementation
  – When wire-path crosses to remote region
    » Send computation to remote PE, or
    » Send message to access remote data

• **Method 2:**
  – Distribute only wires as in SM implementation
  – Fully replicate cost array on each PE
    » One owned region, and potential stale copy of others
    » Send updates so copies are not too stale
  – Consequences:
    » Waste of memory in replication
    » Stale data → poorer quality results or more iterations

• **Both methods require lots of thought for programmer**
MPI: Message Passing Interface

• From the MPI website:

“MPI is a library specification for message-passing, proposed as a standard by a broadly based committee of vendors, implementors, and users.”

• Popular and portable message passing library for
  – Massively parallel machines
  – Clusters of PCs or workstations
Review: Creating a Parallel Program

- Can be done by programmer, compiler, run-time system or OS

- Steps for creating parallel program
  - Decomposition of work into tasks
  - Assignment of tasks to processes
  - Orchestration of processes
  - Mapping of processes to processors

- In practice, parallel programs created with
  - Explicitly parallel language (HPF, Split-C, CUDA)
  - Shared memory library (pthreads, PARMACS)
  - Message passing library (MPI)
Map-Reduce

- PRESENTATION