CIS 631
Parallel Processing

Lecture 12: Parallel Performance Analysis

Allen D. Malony
malony@cs.uoregon.edu

Department of Computer and Information Science
University of Oregon
Acknowledgements

- Portions of the lectures slides were adopted from:
Outline

- Review
- Parallel performance analysis problem
- Parallel performance analysis methodology
- Measurement and analysis techniques
Parallel Programming

- To use a scalable parallel computer, you must be able to write parallel programs.
- You must understand the programming model and the programming languages, libraries, and systems software used to implement it.
- Unfortunately, parallel programming is not easy.
Parallel Programming: Are we having fun yet?

Source: Bernd Mohr
Parallel Programming Models

- Two general models of parallel program
  - Task parallel
    - Problem is broken down into tasks to be performed
    - Individual tasks are created and communicate to coordinate operations
  - Data parallel
    - Problem is viewed as operations of parallel data
    - Data distributed across processes and computed locally

- Characteristics of scalable parallel programs
  - Data domain decomposition to improve data locality
  - Communication and latency do not grow significantly
Shared Memory Parallel Programming

- Shared memory address space
- (Typically) easier to program
  - Implicit communication via (shared) data
  - Explicit synchronization to access data
- Programming methodology
  - Manual
    - Multi-threading using standard thread libraries
  - Automatic
    - Parallelizing compilers
    - OpenMP parallelism directives
  - Explicit threading (e.g. POSIX threads)
Parallel Programming Model: Threads

- Global style
- Shared and private data
- Work distribution onto threads for global operations
- Domain decomposition determines work distribution
Distributed Memory Parallel Programming

- Distributed memory address space
- (Relatively) harder to program
  - Explicit data distribution
  - Explicit communication via messages
  - Explicit synchronization via messages
- Programming methodology
  - Message passing
    - Plenty of libraries to chose from (MPI dominates)
    - Send-receive, one-sided, active messages
  - Data parallelism
  - Shared virtual memory
Paralle Programming Model: Message Passing

- Local style
- Domain decomposition leads to data distribution
- Explicit communication and synchronization
- Higher programming overhead
- Message passing libraries
Basic Parallel Programming Paradigm: SPMD

- SPMD: Single Program Multiple Data
- One program executes on all processors
- Basic paradigm for implementing parallel programs
- Process-dependent cases are handled inside the program

```c
if (processor == 42) then
    call do_something()
else
    call do_something_else()
endif
```

- Parallelism is “programmed in”
- Easier to manage program for scalability
Parallel Programming: Still a Problem?
Parallel Computing and Scalability

- Scalability in parallel architecture
  - Processor numbers
  - Memory architecture
  - Interconnection network
  - Avoid critical architecture bottlenecks
- Scalability in computational problem
  - Problem size
  - Computational algorithms
    - Computation to memory access ratio
    - Computation to communication ratio
- Parallel programming models and tools
- Performance scalability
Amdahl’s Law

- $T_{seq}$: sequential execution time that cannot be parallelized
- $T_{par}$: sequential execution time that can be parallelized
- $T_1 = T_{seq} + T_{par} \implies T_{par} = T_1 - T_{seq}$
- $T_p = T_{seq} + T_{par}/p$ (assume fully parallelized)
- As $p \to \infty$, $T_p \to T_{seq}$
- Let $f_{seq}$ be the fraction $T_{seq}/T_1$ and $S_p = T_1/T_p$
- Speedup $= S_p = T_1/T_p = T_1/(T_{seq} + T_{par}/p)$
  $= 1/(f_{seq} + T_{par}/pT_1) = 1/(f_{seq} + (1-f_{seq})/p)$
  - As $p \to \infty$, $S_p = S_\infty \to 1/f_{seq}$
- Speedup bound is determined by the degree of sequential execution time in the computation, not # processors!!!
Amdahl’s Law and Scaled Speedup

- Amdahl’s Law makes it hard to obtain good speedup

<table>
<thead>
<tr>
<th>$f_{seq}$ * 100%</th>
<th>10%</th>
<th>5%</th>
<th>2%</th>
<th>1%</th>
<th>.1%</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_\infty$</td>
<td>10</td>
<td>20</td>
<td>50</td>
<td>100</td>
<td>1000</td>
</tr>
</tbody>
</table>

- Change perspective on the problem
- Consider scaling of problem size as # processors scale
- $T_{seq}$: sequential execution time (1 and $p$ processors)
- $T_{par}$: execution time in parallel mode on $p$ processors
- $T_p = T_{seq} + T_{par}$, $T_1 = T_{seq} + pT_{par}$
- Let $f_{par}$ be the fraction $T_{par} / T_p$
- Scaled speedup $= S_p = 1 + (p-1)T_{par} / T_p = 1 + (p-1)f_{par}$
Parallel Performance

- To use a scalable parallel computer well, you must be able to write high-performance parallel programs.
- To get high-performance parallel programs, you must understand and optimize performance for the combination of programming model, algorithm, language, platform, ...
- Unfortunately, parallel performance analysis and optimization is not an easy process.
Parallel Performance Evaluation

- Study of performance in parallel systems
  - Models and behaviors
  - Evaluative techniques
- Evaluation methodologies
  - Analytical modeling and statistical modeling
  - Simulation-based modeling
  - Empirical measurement, analysis, and modeling
- Purposes
  - Planning
  - Diagnosis
  - Tuning
Performance Observability (My Guiding Thesis)

- Performance evaluation problems define the requirements for performance analysis methods.
- *Performance observability* is the ability to “accurately” capture, analyze, and present (collectively observe) information about computer system/software performance.
- Tools for performance observability must balance the need for performance data against the cost of obtaining it (environment complexity, performance intrusion).
  - Too little performance data makes analysis difficult.
  - Too much data perturbs the measured system.
- Important to understand performance observability complexity and develop technology to address it.
(Parallel) Performance Analysis Process

Implementation

Preparation

Performance Analysis

Program Tuning

Production

Refinement

Measurement

Analysis

Ranking
Parallel Performance Analysis Environment

- **Parallel System**
- **Parallel Program**

- System/program characteristics plus performance knowledge used for initial hypothesis
- Constraints on observational capabilities, invocation of measurement tools

- **Performance Hypothesis**
- **Performance Observation**

- Hypothesis refinement from empirical results
- Analysis, modeling, and presentation of empirical data

- **Stored Performance Knowledge**
- General performance results add to performance knowledge base

- **Experimental Performance Data**
Performance Analysis and Tuning

- Successful parallel performance tuning process
  - Characterization: finding critical performance problems
  - Diagnosis: determining performance problem causes
  - Hypothesis testing: selection of performance optimization
  - Hypothesis validation: analyzing tuning results

- Reasoning and intuition only take you so far

- Need to make empirical observations
  - Performance instrumentation tools
  - Performance measurement tools
  - Performance analysis tools
Factors which determine a program's performance are complex, interrelated, and sometimes hidden

Application related factors
- Algorithms dataset sizes
- Memory usage patterns
- I/O communication patterns

Hardware related factors
- Processor architecture
- Memory hierarchy

Software related factors
- Operating system
- Compiler preprocessor

Task Granularity
Load Balancing
Amdahl's Law
I/O network
Communication protocols
Libraries
Utilization of Computational Resources

- Often resources are under-utilized or used inefficiently
- Identifying these circumstances can give clues to where performance problems exist
- Resources may be “virtual” (i.e., not a physical resource)
  - Thread or process
- Performance analysis tools are essential to optimizing an application's performance
  - Can assist you in understanding what your program is "really doing"
  - May provide suggestions how program performance should be improved
Performance Analysis and Tuning: The Basics

- Most important goal of performance tuning is to reduce a program's wall clock execution time
  - Iterative process to optimize efficiency
  - Efficiency is a relationship of execution time
- So, where does the time go?
- Find your program's hot spots and eliminate the bottlenecks in them
  - *Hot spot*: an area of code within the program that uses a disproportionately high amount of processor time
  - *Bottleneck*: an area of code within the program that uses processor resources inefficiently and therefore causes unnecessary delays
- Understand what, where, and how time is being spent
Sequential versus Parallel Performance

- Sequential performance is all about how time is distributed and what resources are used where and when.
- Parallel performance is about sequential performance AND parallel interactions.
  - Sequential performance is the performance within each thread of execution (i.e., its sequential performance).
  - Parallel interactions lead to overheads:
    - synchronization
    - communication
  - Parallel interactions also lead to parallelism inefficiency:
    - load imbalances
Sequential Performance Tuning

- Sequential performance tuning is a *time-driven* process.
- Find the thing that takes the most time and make it take less time (i.e., make it more efficient).
- May lead to program restructuring:
  - Changes in data storage and structure
  - Rearrangement of tasks and operations
- May look for opportunities for better resource utilization:
  - Cache management is a big one
  - Locality, locality, locality!
  - Virtual memory management may also pay off
- May look for opportunities for better processor usage
Parallel Performance Tuning versus Sequential

- In contrast to sequential performance tuning, parallel performance tuning might be described as conflict-driven or interaction-driven.
- Find the points of parallel interactions and determine the overheads associated with them.
- Overheads can be the cost of performing the interactions:
  - Transfer of data
  - Extra operations to implement coordination
- Overheads also include time spent waiting:
  - Lack of work
  - Waiting for dependency to be satisfied
Interesting Performance Phenomena

- Superlinear speedup
  - Speedup in parallel execution is greater than linear
  - $S_p > p$
  - How can this happen?

- Need to keep in mind the relationship of performance and resource usage

- Computation time (i.e., real work) is not simply a linear distribution to parallel threads of execution

- Resource utilization thresholds can lead to performance inflections
How Is Time Measured?

☐ How do we determine where the time goes?
☐ “A person with one clock knows what time it is, a person with two clocks is never sure.” — Confucious
☐ “Define time.” — Bill Clinton (attributed)
☐ Time is only as good (accurate) as the clock we use
☐ Clocks are not the same and, thus, time is not the same
  ☐ Wallclock time — measured against “real” time
  ☐ CPU (virtual) time — time accumulates (i.e., “ticks”) only when process is executing
  ☐ Clocks have different resolutions and overheads for access
    ➢ affects accuracy
Next Class

- Parallel performance tools