Lecture 6: Measurement Tools
Abhinav Bhavele, Department of Computer Science
Summary of last lecture

• Shared-memory programming and OpenMP
• Fork-join parallelism
• OpenMP vs MPI: ease of programming, performance
Performance analysis

- Parallel performance of a program might not be what the developer expects
- How do we find performance bottlenecks?
- Two parts to performance analysis: measurement and analysis/visualization
- Simplest tool: timers in the code and printf
Using timers

double start, end;
double phase1, phase2, phase3;

start = MPI_Wtime();
    ... phase1 code ...
end = MPI_Wtime();
phase1 = end - start;

start = MPI_Wtime();
    ... phase2 ...
end = MPI_Wtime();
phase2 = end - start;

start = MPI_Wtime();
    ... phase3 ...
end = MPI_Wtime();
phase3 = end - start;
Using timers

double start, end;
double phase1, phase2, phase3;

start = MPI_Wtime();
   ... phase1 code ...  
end = MPI_Wtime();
phase1 = end - start;

start = MPI_Wtime();
   ... phase2 ...  
end = MPI_Wtime();
phase2 = end - start;

start = MPI_Wtime();
   ... phase3 ...  
end = MPI_Wtime();
phase3 = end - start;

Phase 1 took 2.45 s
Phase 2 took 11.79 s
Phase 3 took 4.37 s
Performance Tools

• Tracing tools
  • Capture entire execution trace
  • Vampir, Score-P

• Profiling tools
  • Provide aggregated information
  • Typically use statistical sampling
  • Gprof, pyinstrument, cprofile

• Many tools can do both
  • TAU, HPCToolkit, Projections
Metrics recorded

- Counts of function invocations
- Time spent in code
- Number of bytes sent
- Hardware counters
- To fix performance problems — we need to connect metrics to source code
Tracing tools

- Record all the events in the program with timestamps
- Events: function calls, MPI events, etc.
Tracing tools

- Record all the events in the program with timestamps
- Events: function calls, MPI events, etc.

Tracing tools

- Record all the events in the program with timestamps
- Events: function calls, MPI events, etc.

Profiling tools

- Ignore the specific times at which events occurred
- Provide aggregate information about different parts of the code
- Examples:
  - Gprof, perf
  - mpiP
  - HPCToolkit, caliper
- Python tools: cprofile, pyinstrument, scalene

Gprof data in hpctView
Calling contexts, trees, and graphs

- Calling context or call path: Sequence of function invocations leading to the current sample
- Calling context tree (CCT): Dynamic prefix tree of all call paths in an execution
- Call graph: Merge nodes in a CCT with the same name into a single node but keep caller-callee relationships as arcs

![Diagram showing a call graph with nodes and edges labeled with function names such as 'main', 'physics', 'solvers', 'mpi', 'hypre', 'mpi', 'psm2', 'psm2'.]
Calling context trees, call graphs, …
Calling context trees, call graphs, ...

Calling context tree (CCT)
Calling context trees, call graphs, ...

Contextual information
- File
- Line number
- Function name
- Callpath
- Load module
- Process ID
- Thread ID

Calling context tree (CCT)
Calling context trees, call graphs, ...

Contextual information
- File
- Line number
- Function name
- Callpath
- Load module
- Process ID
- Thread ID

Performance Metrics
- Time
- Flops
- Cache misses

Calling context tree (CCT)
Calling context trees, call graphs, …

Contextual information
- File
- Line number
- Function name
- Callpath
- Load module
- Process ID
- Thread ID

Performance Metrics
- Time
- Flops
- Cache misses

Calling context tree (CCT)

Call graph
Questions?

Abhinav Bhatele
5218 Brendan Iribe Center (IRB) / College Park, MD 20742
phone: 301.405.4507 / e-mail: bhatele@cs.umd.edu