Workload-Driven Architectural Evaluation

Difficult Enough for Uniprocessors

Workloads need to be renewed and reconsidered

Input data sets affect key interactions

- Changes from SPEC92 to SPEC95 to SPEC2000 to …

Accurate simulators costly to develop and verify

Simulation is time-consuming

Quantitative evaluation increasingly important for multiprocessors

- Maturity of architecture, and greater continuity among generations
- It’s a grounded, engineering discipline now

Good evaluation is critical, and we must learn to do it right

More Difficult for Multiprocessors

What is a representative workload?

Software model has still not stabilized

Many architectural and application degrees of freedom

- Huge design space: no. of processors, other architectural, application
- Impact of these parameters and their interactions can be huge
- High cost of communication

What are the appropriate metrics?

Simulation is expensive

- Realistic configurations and sensitivity analysis difficult
- Larger design space, but more difficult to cover

Understanding of parallel programs as workloads is critical

- Particularly interaction of application and architectural parameters

Outline

What workload should we use?

What metrics should we use?

How do we evaluate?

Workloads

Purely Synthetic Workloads/Microbenchmarks

- Short programs exercising specific features (memory, I/O, FLOPs, etc.)

Kernels

- Well defined and time-consuming parts of real applications.
- e.g. Multigrid kernel in Ocean

Entire Applications

- Not necessarily realistic (multiprogramming, etc. may be ignored)

Metrics

Absolute performance

- Most important to end user

Performance improvement due to parallelism (Relative)

- *Speedup(p) = Performance(p) / Performance(1)*, always

Performance = Work / Time, always

Work is determined by input configuration of the problem

If work is fixed, can measure performance as 1/Time

- Or retain explicit work measure (e.g. transactions/sec, bonds/sec)
- $Speedup(p) = \dfrac{Time(1)}{Time(p)}$ or $\dfrac{Operations\ Per\ Second\ (p)}{Operations\ Per\ Second\ (1)}$

Scaling: Why Worry?

Fixed problem size is limited

Too small a problem:

- May be appropriate for small machine
- Parallelism overheads begin to dominate benefits for larger machines
  - Load imbalance
  - Communication to computation ratio
- May even achieve slowdowns
- Doesn’t reflect real usage, and inappropriate for large machines
  - Can exaggerate benefits of architectural improvements, especially when measured as percentage improvement in performance

Too large a problem

- Difficult to measure improvement (may not run on small machine)

Under What Constraints to Scale?

Two types of constraints:

- User-oriented, e.g. particles, rows, transactions, I/Os per processor
- Resource-oriented, e.g. memory, time

Which is more appropriate depends on application domain

- User-oriented easier for user to think about and change
- Resource-oriented more general, and often more real

Resource-oriented scaling models:

- *Problem constrained* (PC)
- *Memory constrained* (MC)
- *Time constrained* (TC)

Problem Constrained Scaling

User wants to solve same problem, only faster

But limited when evaluating larger machines

$$Speedup_{PC}(p) = \frac{Time(1)}{Time(p)}$$

Time Constrained Scaling

Execution time is kept fixed as system scales

- User has fixed time to use machine or wait for result

Performance = Work/Time as usual, and time is fixed, so

$$SpeedupTC(p) = \frac{Work(p)}{Work(1)}$$

How to measure work?

- Execution time on a single processor? (thrashing problems)
- Should be easy to measure, ideally analytical and intuitive
- Should scale linearly with sequential complexity
  - Or ideal speedup will not be linear in *p* (e.g. no. of rows in matrix program)
- If cannot find intuitive application measure, as often true, measure *execution time with ideal memory system on a uniprocessor*

Memory Constrained Scaling

Scale so memory usage per processor stays fixed

Scaled Speedup: Time(1) / Time(p) for scaled up problem

- Hard to measure Time(1), and inappropriate

$$Speedup_{MC}(p) = \frac{Work(p)}{Time(p)} \times \frac{Time(1)}{Work(1)} = \frac{\text{Increase in Work}}{\text{Increase in Time}}$$

Can lead to large increases in execution time

- If work grows faster than linearly in memory usage
- e.g. matrix factorization
  - 10,000-by 10,000 matrix takes 800MB and 1 hour on uniprocessor
  - With 1,000 processors, can run 320K-by-320K matrix, but ideal parallel time grows to 32 hours!
  - With 10,000 processors, 100 hours ...

Time constrained seems to be most generally viable model

Impact of Scaling Models: Grid Solver

MC Scaling:

- Grid size = $n\sqrt{p}$-by-$n\sqrt{p}$
- Iterations to converge = $n\sqrt{p}$
- Work = $O(n\sqrt{p})^3$
- Ideal parallel execution time = $O\left(\frac{(n\sqrt{p})^3}{p}\right) = n^3 \sqrt{p}$
- Grows by $n\sqrt{p}$
- 1 hr on uniprocessor means 32 hr on 1024 processors

TC scaling:

- If scaled grid size is k-by-k, then $k^3/p = n^3$, so $k = n \sqrt[3]{p}$.
- Memory needed per processor = $k^2/p = n^2 / \sqrt[3]{p}$
- Diminishes as cube root of number of processors

Impact on Solver Execution Characteristics

Concurrency: PC: fixed; MC: grows as p; TC: grows as $p^{0.67}$

Comm to comp: PC: grows as $\sqrt{p}$; MC: fixed; TC: grows as $\sqrt[6]{p}$

Working Set: PC: shrinks as p; MC: fixed; TC: shrinks as $\sqrt[3]{p}$

Spatial locality?

Message size in message passing?

- Expect speedups to be best under MC and worst under PC
- Should evaluate under all three models, unless some are unrealistic

How to Evaluate?

Run on actual machine.

Execution-driven simulation

- Specific artifacts
- Complete system

Analytical Models

Multiprocessor Simulation

Simulation runs on a uniprocessor (can be parallelized too)

- Simulated processes are interleaved on the processor

Two parts to a simulator:

- Reference generator: plays role of simulated processors
  - And schedules simulated processes based on *simulated time*
- Simulator of extended memory hierarchy
  - Simulates operations (references, commands) issued by reference generator

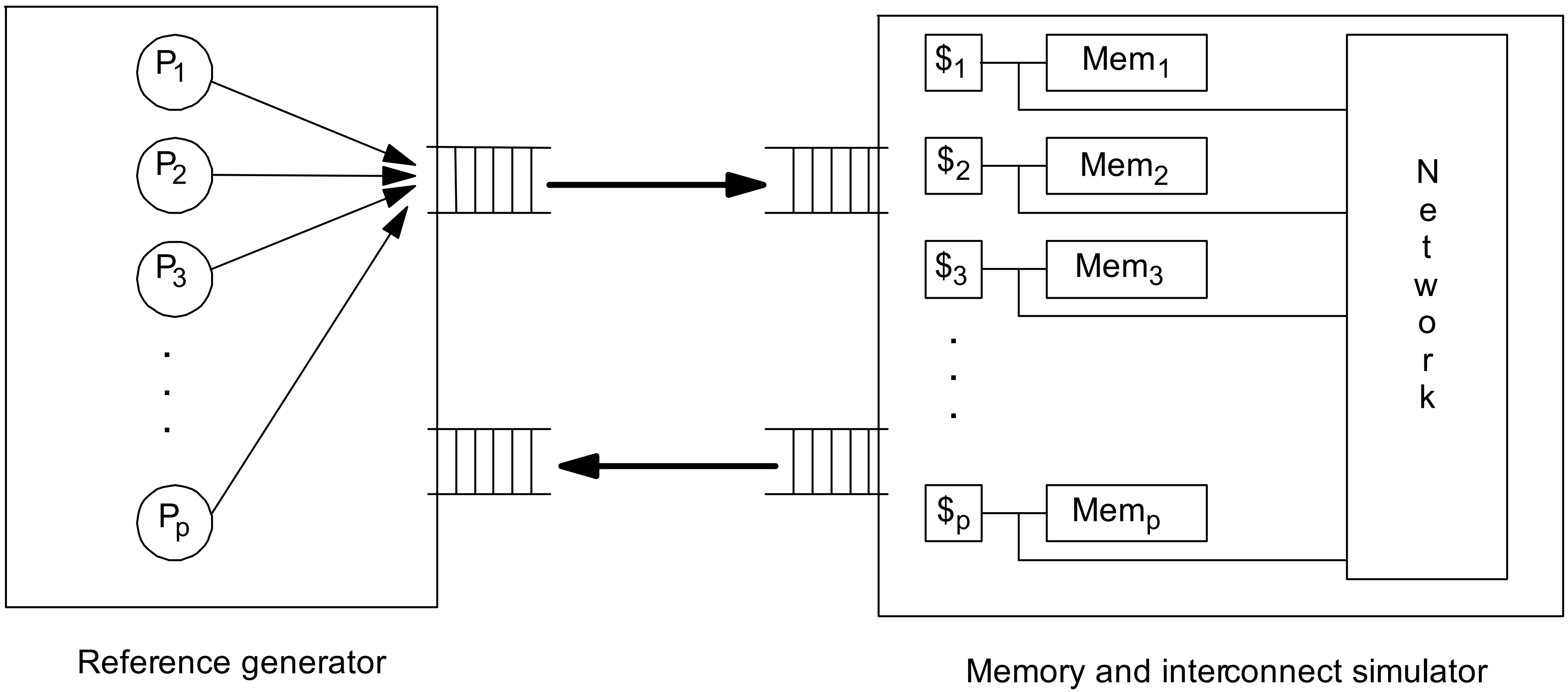
Coupling or information flow between the two parts varies

- Trace-driven simulation: from generator to simulator
- Execution-driven simulation: in both directions (more accurate)

Simulator keeps track of simulated time and detailed statistics

Execution-driven Simulation

Memory hierarchy simulator returns simulated time information to reference generator, which is used to schedule simulated processes



Difficulties in Simulation-based Evaluation

Two major problems, beyond accuracy and reliability:

- Cost of simulation (in time and memory)
  - cannot simulate the problem/machine sizes we care about
  - have to use scaled down problem and machine sizes
    - *how to scale down and stay representative?*
- Huge design space
  - application parameters (as before)
  - machine parameters (depending on generality of evaluation context)
    - number of processors
    - cache/replication size
    - associativity
    - granularities of allocation, transfer, coherence
    - communication parameters (latency, bandwidth, occupancies)
  - cost of simulation makes it all the more critical to prune the space