Workload-Driven Architecture Evaluation

Todd C. Mowry
CS 418
February 1 & 2, 2011

Evaluation for Uniprocessors

Decisions made only after quantitative evaluation

For existing systems: comparison and procurement evaluation

For future systems: careful extrapolation from known quantities

Wide base of programs leads to standard benchmarks

- Measured on wide range of machines and successive generations

Measurements and technology assessment lead to proposed features

Then simulation

- Simulator developed that can run with and without a feature
- Benchmarks run through the simulator to obtain results
- Together with cost and complexity, decisions made

Difficult Enough for Uniprocessors

Workloads need to be renewed and reconsidered

Input data sets affect key interactions

- Changes across each release of SPEC benchmarks

Accurate simulators costly to develop and verify

Simulation is time-consuming

But the effort pays off: Good evaluation leads to good design

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 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

A Lot Depends on Sizes

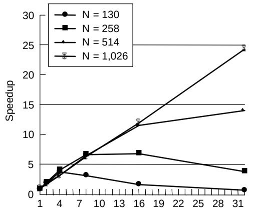
Application parameters and # of procs affect inherent properties

- Load balance, communication, extra work, temporal and spatial locality

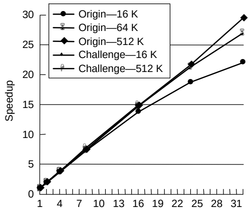
Interactions with organization parameters of extended memory hierarchy affect artifactual communication and performance

Effects often dramatic, sometimes small: application-dependent

Ocean



Barnes-Hut



Understanding size interactions and scaling relationships is key

Outline

Performance and scaling (of workload and architecture)

- Techniques
- Implications for behavioral characteristics and performance metrics

Evaluating an architectural idea/tradeoff through simulation

Scaling: Why Worry?

Fixed problem size is limited

Too small of 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 of a problem

- Difficult to measure improvement (next)

Too Large of a Problem

Suppose problem realistically large for big machine

May not "fit" in small machine

- Can't run
- Thrashing to disk
- Working set doesn't fit in cache

Fits at some *p*, leading to *superlinear speedup*

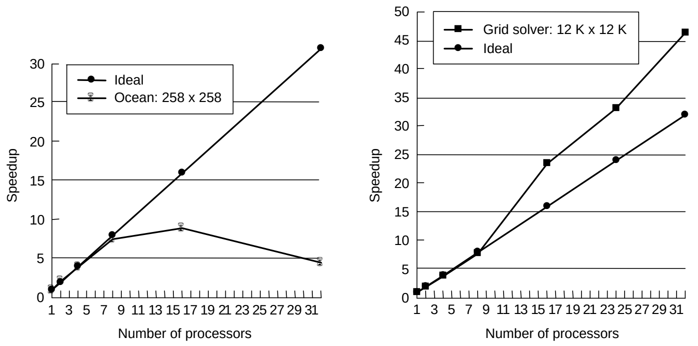
- Real effect, but doesn't help evaluate effectiveness

Finally, users want to scale problems as machines grow

- Can help avoid these problems

Demonstrating Scaling Problems

Small Ocean and big equation solver problems on SGI Origin2000



Questions in Scaling

Under what constraints to scale the application?

- What are the appropriate metrics for performance improvement?
  - work is not fixed any more, so time not enough

How should the application be scaled?

Definitions:

***Scaling a machine*: Can scale power in many ways**

- Assume adding identical nodes, each bringing memory

***Problem size*: Vector of input parameters, e.g. $N = (n, q, \Delta t)$**

- Determines work done
- Distinct from *data set size* and *memory usage*
- Start by assuming it's only one parameter *n*, for simplicity

---

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)

(TPC: transactions, users, terminals scale with "computing power")

Growth under MC and TC may be hard to predict

Problem Constrained Scaling

User wants to solve same problem, only faster

- Video compression
- Computer graphics
- VLSI routing

But limited when evaluating larger machines

$$\mathit{Speedup}_{PC}(p) = \frac{\mathit{Time}(1)}{\mathit{Time}(p)}$$

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* (e.g. pixie)

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

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

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 $\sqrt{p}$
- 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

Scaling Workload Parameters: Barnes-Hut

Different parameters govern different sources of error:

- Number of bodies (*n*)
- Time-step resolution (Δt)
- Force-calculation accuracy (θ)

Scaling rule:

- All components of simulation error should scale at same rate

Result: If *n* scales by a factor of *s*

- Δt and θ must both scale by a factor of $\frac{1}{\sqrt[4]{s}}$

Effects of Scaling Rule

If number of processors (*p*) is scaled by a factor of *k*

- **Under Time Constrained scaling:**
  - increase in number of particles is less than $\sqrt{k}$
- **Under Memory Constrained scaling:**
  - elapsed time becomes unacceptably large

Time Constrained is most realistic for this application

Effect on execution characteristics

- Each parameter has its own effect on c-to-c ratio, working sets etc.
- Scaling them inappropriately can lead to incorrect results
- e.g. working set in Barnes-Hut is $\frac{1}{\theta^2} \log n$
  - With proper scaling, grows much more quickly with θ than with *n*, so scaling only *n* would give misleading results
  - Unlike solver, working set independent. of *p*; increases under TC scaling

Performance and Scaling Summary

Performance improvement due to parallelism measured by speedup

Scaling models are fundamental to proper evaluation

Speedup metrics take different forms for different scaling models

Scaling constraints affect growth rates of key execution properties

- Time constrained scaling is a realistic method for many applications

Should scale workload parameters appropriately with one another too

- Scaling only data set size can yield misleading results

Proper scaling requires understanding the workload

Some Important Observations

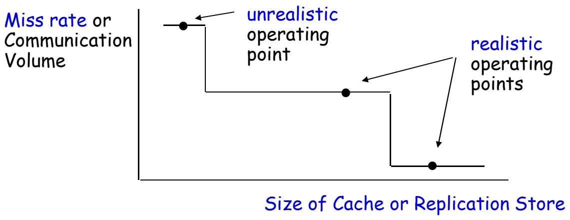
In addition to assignment/orchestration, many important properties of a parallel program depend on:

- Application parameters and number of processors
- Working sets and cache/replication size

Should cover realistic regimes of operation

Operating Points Based on Working Sets

Many applications have a hierarchy of working sets:



- A working set may consist of local and/or nonlocal data
  - not fitting it may dramatically increase local miss rate or even communication
- Some working sets scale with application parameters and p, some don't
- Some operating points are realistic, some aren't
  - *operating point = f (cache/replication size, application parameters, p)*

Evaluating an Idea or Tradeoff

Typically many things change from one generation to next

Building prototypes for evaluation is too expensive

Build a simulator

Case I: Want to examine in a specific context

- **Can assume technological and architectural parameters**
  - Simulate with feature turned off and turned on to examine impact
  - Perhaps examine sensitivity to some parameters that were fixed
- **Building accurate simulators is complex**
  - Contention difficult to model correctly
  - Processors becoming increasingly complex themselves

Case II: Want to examine benefit of idea in a more general context

- **Now machine parameters also variable**
  - Various sizes,granularities and organizations, performance characteristics

---

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

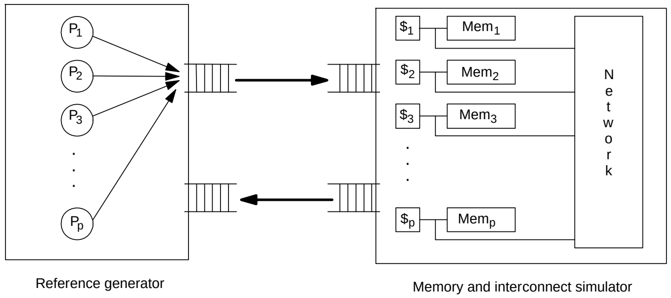
- **Trace-driven simulation: from generator to simulator**
- **Execution-driven simulation: in both directions (more accurate)**

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

Scaling Down Parameters for Simulation

Want scaled-down machine running scaled-down problem to be representative of full-sized scenario

- **No good formulas exist**
- **But very important since reality of most evaluation**
- **Should understand limitations and guidelines to avoid pitfalls**

First examine scaling down problem size and # of processors

Then lower-level machine parameters

Focus on cache-coherent SAS for concreteness

Scaling Down Problem Parameters

Some parameters don't affect parallel performance much, but do affect runtime, and can be scaled down

- **Common example is # of time-steps in many scientific applications**
  - need a few to allow settling down, but don't need more
  - may need to omit cold-start when recording time and statistics
- **First look for such parameters**
- **Others can be scaled according to earlier scaling arguments**

But many application parameters affect key characteristics

Scaling them down requires scaling down # of processors too

- **Otherwise can obtain highly unrepresentative behavior**

Difficulties in Scaling *N, p* Representatively

Many goals, difficult individually and often impossible to reconcile

Want to preserve many aspects of full-scale scenario

- **Distribution of time in different phases**
- **Key behavioral characteristics**
- **Scaling relationships among application parameters**
- **Contention and communication parameters**

Can't really hope for full representativeness, but can

- **Cover range of realistic operating points**
- **Avoid unrealistic scenarios**
- **Gain insights and estimates of performance**

Scaling Down Other Machine Parameters

Often necessary when scaling down problem size
- e.g. may not represent working set not fitting if cache not scaled

More difficult to do with confidence
- *Cache/replication size*: guide by scaling of working sets, not data set
- *Associativity and Granularities*: more difficult
  - should try to keep unchanged since hard to predict effects, but ...
  - greater impact with scaled-down application and system parameters
  - difficult to find good solutions for both communication and local access

Solutions and confidence levels are application-specific
- Require detailed understanding of application-system interactions

Should try to use as realistic sizes as possible
- Use guidelines to cover key operating points, and extrapolate with caution

---

Dealing with the Parameter Space

Steps in an evaluation study

- Determine which parameters are relevant to evaluation
- Identify values of interest for them
  - context of evaluation may be restricted
- Analyze effects where possible
- Look for knees and flat regions to prune where possible
- Understand growth rate of characteristic with parameter
- Perform sensitivity analysis where necessary

An Example Evaluation

Goal of study: To determine the value of adding a block transfer facility to a cache-coherent SAS machine with distributed memory

Workloads: Choose at least some that have communication that is amenable to block transfer (e.g. grid solver)

Choosing parameters is more difficult. We have 3 goals:

1. Avoid unrealistic execution characteristics
2. Obtain good coverage of realistic characteristics
3. Prune the parameter space based on
   - goals of study
   - restrictions imposed by technology or assumptions
   - understanding of parameter interactions

Let's use equation solver as example

Choosing Parameters

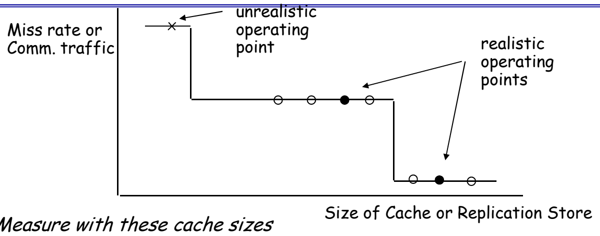
Problem size and number of processors

- Use inherent characteristics considerations as discussed earlier
- For example, low c-to-c ratio will not allow block transfer to help much
- Suppose one size chosen is 514-by-514 grid with 16 processors

Cache/Replication Size

- Choose based on knowledge of working set curve
- Choosing cache sizes for given problem and machine size analogous to choosing problem sizes for given cache and machine size, as discussed
- Whether or not working set fits affects block transfer benefits greatly
  - if local data, not fitting makes communication relatively less important
  - if nonlocal, can increase artifactual comm. So BT has more opportunity
- Sharp knees in working set curve can help prune space **(next slide)**
  - knees can be determined by analysis or by very simple simulation

Example of Pruning using Knees



● *Measure with these cache sizes*
○ *Don't measure with these cache sizes*

But be careful: applicability depends on what is being evaluated

- **what if miss rate isn't all that matters from cache (see update/invalidate protocols later)**

If growth rate can be predicted, can prune for other *n,p, ...* too

Often knees are not sharp, in which case use sensitivity analysis

---

Choosing Parameters (contd.)

Cache block size: issues more detailed

- **Long cache blocks behave like small block transfers already**
- **When spatial locality is good, explicit block transfer less important**
- **When spatial locality is bad**
  - waste bandwidth in read-write communication
  - but also in block transfer IF implemented on top of cache line transfers
  - block transfer itself increases bandwidth needs (same comm. in less time)
  - so it may hurt rather than help if spatial locality bad and implemented on top of cache line transfers, if bandwidth is limited
- **Fortunately, range of interesting line sizes is limited**
  - if thresholds occur, as in Radix sorting, must cover both sides

Associativity

- **Effects difficult to predict, but range of associativity usually small**
- **Be careful about using direct-mapped lowest-level caches**

Choosing Parameters (contd.)

Overhead, network delay, assist occupancy, network bandwidth

- **Higher overhead for cache miss: greater amortization with BT**
  - unless BT overhead offsets it
- **Higher network delay, greater benefit of BT amortization**
  - no knees in effects of delay, so choose a few in the range of interest
- **Network bandwidth is a saturation effect:**
  - once amply adequate, more doesn't help; if low, then can be very bad
  - so pick one that is less than the knee, one near it, and one much greater
  - take burstiness into account when choosing (average needs may mislead)

Revisiting choices

- **Values of earlier parameters may have be revised based on interactions with those chosen later**
  - e.g. choosing direct-mapped cache may require choosing larger caches

---

Summary of Evaluating a Tradeoff

Results of a study can be misleading if space not covered well

- **Sound methodology and understanding interactions is critical**

While complex, many parameters can be reasoned about at high level

- **Independent of lower-level machine details**
- **Especially: problem parameters, # of processors, relationship between working sets and cache/replication size**
- **Benchmark suites can provide and characterize these so users needn't**

Important to look for knees and flat regions in interactions

- **Both for coverage and for pruning the design space**

High-level goals and constraints of a study can also help a lot