Outline

° Evaluation of applications is important
° Simulation of sample data sets provides important information
° Working sets indicate grain size
° Preliminary results offer opportunity for tuning
° Understanding communication costs
  • Remember: software and communication!
Workload-Driven Evaluation

- Evaluating real machines
- Evaluating an architectural idea or trade-offs
  => need good metrics of performance
  => need to pick good workloads
  => need to pay attention to scaling
    • many factors involved

- Today: narrow architectural comparison
- Set in wider context
Evaluation in 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
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 no. of procs affect inherent properties
  - Load balance, communication, extra work, temporal and spatial locality

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

- Effects often dramatic, sometimes small: application-dependent

Understanding size interactions and scaling relationships is key
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 (next)
Too Large 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

![Graph showing speedup vs. number of processors for different workloads.](image-url)
Communication and Replication

- View parallel machine as extended memory hierarchy
  - Local cache, local memory, remote memory
  - Classify “misses” in “cache” at any level as for uniprocessors
    - compulsory or cold misses (no size effect)
    - capacity misses (yes)
    - conflict or collision misses (yes)
    - communication or coherence misses (no)

- Communication induced by finite capacity is most fundamental artifact
  - Like cache size and miss rate or memory traffic in uniprocessors
Working Set Perspective

• At a given level of the hierarchy (to the next further one)

- Hierarchy of working sets
- At first level cache (fully assoc, one-word block), inherent to algorithm
  - *working set curve* for program
- Traffic from any type of miss can be local or nonlocal (communication)
Workload-Driven Evaluation

° Evaluating real machines
° Evaluating an architectural idea or trade-offs

=> need good metrics of performance
=> need to pick good workloads
=> need to pay attention to scaling
  • many factors involved
Questions in Scaling

- **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*

- **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?**
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
  - Video compression
  - Computer graphics
  - VLSI routing

○ But limited when evaluating larger machines

\[
\text{Speedup}_{PC}(p) = \frac{\text{Time}(1)}{\text{Time}(p)}
\]
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

\[
\text{Speedup}_{TC}(p) = \frac{\text{Work}(p)}{\text{Work}(1)}
\]

- 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: $\frac{\text{Time}(1)}{\text{Time}(p)}$ for scaled up problem
  - Hard to measure $\text{Time}(1)$, and inappropriate
  
  $$Speedup_{MC}(p) = \frac{\text{Work}(p)}{\text{Time}(p)} \times \frac{\text{Time}(1)}{\text{Work}(1)} = \frac{\text{Increase in Work}}{\text{Increase in Time}}$$

- 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...
Scaling Summary

° Under any scaling rule, relative structure of the problem changes with $P$
  - PC scaling: per-processor portion gets smaller
  - MC & TC scaling: total problem get larger

° Need to understand hardware/software interactions with scale

° For given problem, there is often a natural scaling rule
  - example: equal error scaling
Types of Workloads

• **Kernels**: matrix factorization, FFT, depth-first tree search
• **Complete Applications**: ocean simulation, crew scheduling, database
• **Multiprogrammed Workloads**

<table>
<thead>
<tr>
<th>Multiprog.</th>
<th>Apps</th>
<th>Kernels</th>
<th>Microbench.</th>
</tr>
</thead>
</table>

Realistic
Complex
Higher level interactions
Are what really matters

Easier to understand
Controlled
Repeatable
Basic machine characteristics

Each has its place:

*Use kernels and microbenchmarks to gain understanding, but applications to evaluate effectiveness and performance*
Coverage: Stressing Features

- Easy to mislead with workloads
  - Choose those with features for which machine is good, avoid others

- Some features of interest:
  - Compute v. memory v. communication v. I/O bound
  - Working set size and spatial locality
  - Local memory and communication bandwidth needs
  - Importance of communication latency
  - Fine-grained or coarse-grained
    - Data access, communication, task size
  - Synchronization patterns and granularity
  - Contention
  - Communication patterns

- Choose workloads that cover a range of properties
Coverage: Levels of Optimization

° Many ways in which an application can be suboptimal
  • Algorithmic, e.g. assignment, blocking
    \[
    \begin{array}{c|c}
    \hline
    & \frac{2n}{p} \\ \hline
    \end{array}
    \begin{array}{c|c}
    \hline
    & \frac{4n}{\sqrt{p}} \\ \hline
    \end{array}
    \]
  • Data structuring, e.g. 2-d or 4-d arrays for SAS grid problem
  • Data layout, distribution and alignment, even if properly structured
  • Orchestration
    - contention
    - long versus short messages
    - synchronization frequency and cost, ...
  • Also, random problems with “unimportant” data structures

° Optimizing applications takes work
  • Many practical applications may not be very well optimized
Concurrency

° Should have enough to utilize the processors
  • If load imbalance dominates, may not be much machine can do
  • (Still, useful to know what kinds of workloads/configurations don’t have enough concurrency)

° Algorithmic speedup: useful measure of concurrency/imbalance
  • Speedup (under scaling model) assuming all memory/communication operations take zero time
  • Ignores memory system, measures imbalance and extra work
  • Uses PRAM machine model (Parallel Random Access Machine)
    - Unrealistic, but widely used for theoretical algorithm development

° At least, should isolate performance limitations due to program characteristics that a machine cannot do much about (concurrency) from those that it can.
Steps in Choosing Problem Sizes

° Variation of characteristics with problem size usually smooth
  • So, for inherent comm. and load balance, pick some sizes along range

° Interactions of locality with architecture often have thresholds (knees)
  • Greatly affect characteristics like local traffic, artifactual comm.
  • May require problem sizes to be added
    - to ensure both sides of a knee are captured
  • But also help prune the design space
Our Cache Sizes (16x1MB, 16x64KB)

- (a) LU
- (b) Ocean
- (c) Barnes–Hut
- (d) Radiosity
- (e) Raytrace
- (f) Radix
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.
Summary

- Evaluate design tradeoffs
  - many underlying design choices
  - prove coherence, consistency

- Evaluation must be based on sound understanding of workloads
  - drive the factors you want to study
  - representative
  - scaling factors

- Use of workload driven evaluation to resolve architectural questions