

Workload-Driven Evaluation

CS 258, Spring 99
David E. Culler
Computer Science Division
U.C. Berkeley

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

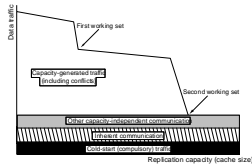
2/12/99

CS258 S99

2

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)

2/12/99

CS258 S99

3

Example Application Set

Table 4.1 General Statistics about Application Programs

Application	Input Data Set (M)	Total Instructions (M)	Total FLOPS (M)	Total References (M)	Total Reads (M)	Total Writes (M)	Shared Reads (M)	Shared Writes (M)	Barriers	Locks
LU	512, 512 matrix 16 x 16 blocks	489.52	92.20	151.07	103.09	47.99	92.79	44.74	66	0
Ocean	258 x 258 grids, iteration = 10 ⁷ 4 time-steps	376.51	101.54	99.70	81.16	18.54	76.95	16.97	364	1,296
Barnes-Hut	16-K particles d=10 3 time-steps	2,002.74	239.24	720.13	406.84	313.29	225.04	93.23	7	34,516
Radix	256-K points radius=1,024	84.62	—	14.19	7.91	6.38	3.61	2.18	11	16
Raytrace	Car scene	833.35	—	290.35	210.03	80.31	161.10	22.35	0	94,456
Radosity	Room scene	2,297.19	—	769.56	486.84	282.72	249.67	21.88	0	210,485
Multigr op	SGR PRX 6.2, User	1,296.43	—	500.22	350.42	149.80	—	—	—	—
Multigr op	Two compress Kernel	668.10	—	212.58	178.14	34.44	—	—	—	621,505

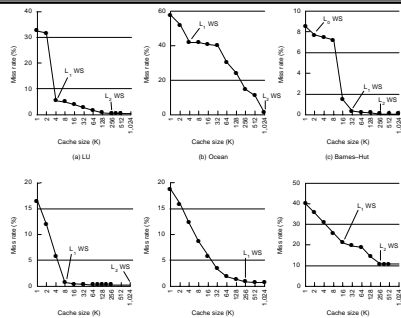
For the parallel programs, shared reads and writes simply refer to all nonstack references issued by the application program. Such references do not necessarily point to data that is truly shared by multiple processes. The Multigr op workload is not a real application, so it does not access shared data. A dash in a table entry means that this measurement is not applicable to one or more entries for that application (e.g., Radix has no calling-point operations). (M) denotes that measurement is in millions.

2/12/99

CS258 S99

4

Working Sets (P=16, assoc, 8 byte)

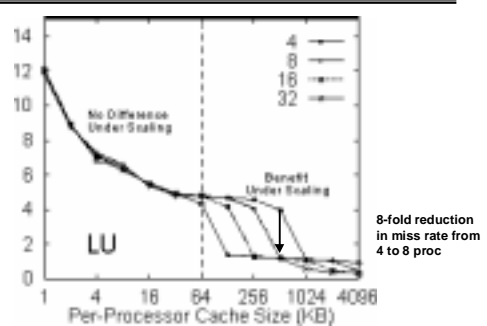


2/12/99

CS258 S99

5

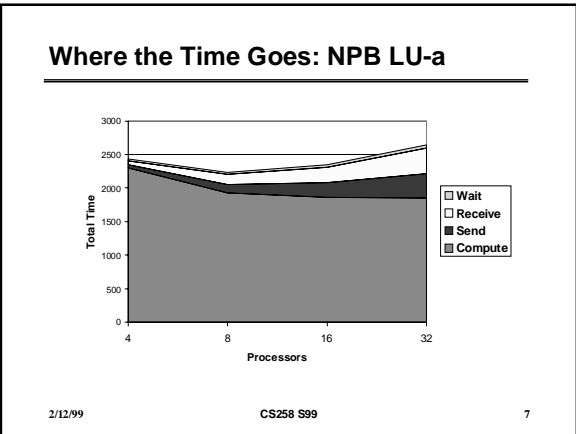
Working Sets Change with P (NPB)



2/12/99

CS258 S99

6



False Sharing Misses: Artfactual Comm.

- Different processors update different words in same block
- Hardware treats it as sharing
 - cache block is unit of coherence
- Ping-pongs between caches

2/12/99 CS258 S99 8

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?**

2/12/99 CS258 S99 9

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**

2/12/99 CS258 S99 10

Problem Constrained Scaling

- **User wants to solve same problem, only faster**
 - Video compression
 - Computer graphics
 - VLSI routing
- **But limited when evaluating larger machines**

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

2/12/99 CS258 S99 11

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**

$$Speedup_{TC}(p) = \frac{Work(p)}{Work(1)}$$

- **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)

2/12/99 CS258 S99 12

Memory Constrained Scaling

- Scale so memory usage per processor stays fixed
- Scaled Speedup: $\text{Time}(1) / \text{Time}(p)$ for scaled up problem

– Hard to measure $\text{Time}(1)$, and inappropriate

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

- 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 ...

2/12/99

CS258 S99

13

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

2/12/99

CS258 S99

14

Types of Workloads

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

• Multiprog. ↔ Appls ↔ Kernels ↔ Microbench.

Realistic
Complex
Higher level interactions
Are what really matters

Easier to understand
Controlled
Repeatable
Basic machine characteristics

Each has its place:

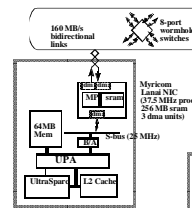
Use kernels and microbenchmarks to gain understanding, but applications to evaluate effectiveness and performance

2/12/99

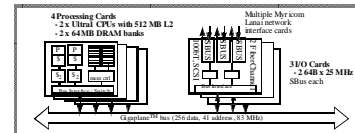
CS258 S99

15

NOW Ultra 170 vs Enterprise 5000



- **Workstation UPA**
 - cross bar
- **SMP**
 - switch between Ultrasparc coherence protocol (MOESI) and bus protocol (MSI)

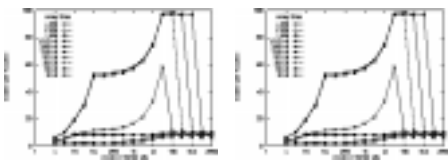


2/12/99

CS258 S99

16

Microbenchmarks



- **Memory access latency (512KB L2, 64B blocks)**
 - Enterprise 5000: 51 cycles
 - other L2: 84 cycles
 - Ultra 170: 44 cycles
- **Memory copy bandwidth**
 - Enterprise 5000: 184 MB/s
 - Ultra 170: 168 MB/s
- Arithmetic, floating point, graphics, ...

2/12/99

CS258 S99

17

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**

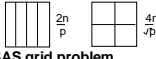
2/12/99

CS258 S99

18

Coverage: Levels of Optimization

- Many ways in which an application can be suboptimal

- *Algorithmic*, e.g. assignment, blocking 
- *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

- May examine selected different levels to test robustness of system

2/12/99

CS258 S99

19

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.

2/12/99

CS258 S99

20

Workload/Benchmark Suites

- Numerical Aerodynamic Simulation (NAS)
 - Originally pencil and paper benchmarks
- SPLASH/SPLASH-2
 - Shared address space parallel programs
- ParkBench
 - Message-passing parallel programs
- ScaLapack
 - Message-passing kernels
- TPC
 - Transaction processing
- SPEC-HPC
- . . .

2/12/99

CS258 S99

21

Evaluating a Fixed-size Machine

- Many critical characteristics depend on problem size

- Inherent application characteristics
 - » concurrency and load balance (generally improve with problem size)
 - » communication to computation ratio (generally improve)
 - » working sets and spatial locality (generally worsen and improve, resp.)
- Interactions with machine organizational parameters
- Nature of the major bottleneck: comm., imbalance, local access...

- Insufficient to use a single problem size

- Need to choose problem sizes appropriately
 - Understanding of workloads will help

2/12/99

CS258 S99

22

Our problem today

- Evaluate architectural alternatives
 - protocols, block size
- Fix machine size and characteristics
- Pick problems and problem sizes

2/12/99

CS258 S99

23

Steps in Choosing Problem Sizes

1. Appeal to higher powers

May know that users care only about a few problem sizes
But not generally applicable

2. Determine range of useful sizes

Below which bad perf. or unrealistic time distribution in phases
Above which execution time or memory usage too large

3. Use understanding of inherent characteristics

Communication-to-computation ratio, load balance...

For grid solver, perhaps at least 32-by-32 points per processor
40MB/s c-to-c ratio with 200MHz processor

No need to go below 5MB/s (larger than 256-by-256 subgrid per processor) from this perspective, or 2K-by-2K grid overall

2/12/99

CS258 S99

24

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

2/12/99

CS258 S99

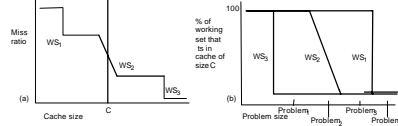
25

Choosing Problem Sizes (contd.)

4. Use temporal locality and working sets

Fitting or not dramatically changes local traffic and artifactual comm.

E.g. Raytrace working sets are nonlocal, Ocean are local



- Choose problem sizes on both sides of a knee if realistic
 - » Critical to understand growth rate of working sets
- Also try to pick one very large size (exercises TLB misses etc.)
- Solver: first (2 subrows) usually fits, second (full partition) may or not
 - » Doesn't for largest (2K) so add 4K-b-4K grid
 - » Add 16K as large size, so grid sizes now 256, 1K, 2K, 4K, 16K (in each dimension)

2/12/99

CS258 S99

26

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

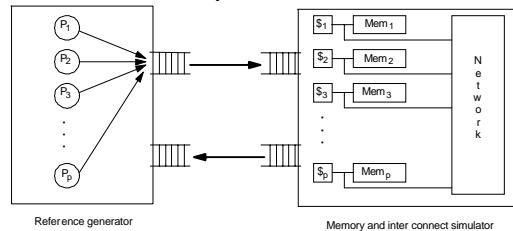
2/12/99

CS258 S99

27

Execution-driven Simulation

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



2/12/99

CS258 S99

28

Difficulties in Simulation-based Evaluation

- **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

2/12/99

CS258 S99

29

Choosing Parameters

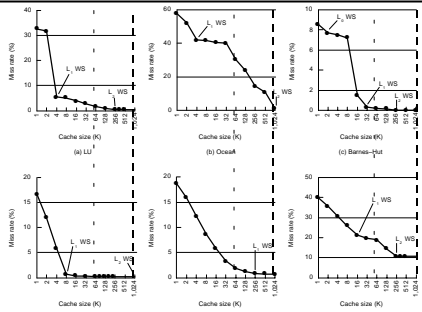
- **Problem size and number of processors**
 - Use inherent characteristics considerations as discussed earlier
 - For example, low c-to-c ratio will not allow block transfer to help much
- **Cache/Replication Size**
 - Choose based on knowledge of working set curve
 - Choosing cache sizes for given problem and machine size analogous to choosing problem sizes for given cache and machine size, 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
 - » Knees can be determined by analysis or by very simple simulation

2/12/99

CS258 S99

30

Our Cache Sizes (16x1MB, 16x64KB)



2/12/99

CS258 S99

31

Focus on protocol tradeoffs

Methodology:

- Use Splash II and Multiprogram workload (ala Ch 4)
- Choose S parameters per earlier methodology
 - default 1MB, 4-way cache, 64-byte block, 16 processors; 64K cache for some
- Focus on frequencies, not end performance for now
 - transcends architectural details, but not what we're really after
- Use idealized memory performance model to avoid changes of reference interleaving across processors with machine parameters
 - Cheap simulation: no need to model contention
- Run program on parallel machine simulator
 - collect trace of cache state transitions
 - analyze properties of the transitions

2/12/99

CS258 S99

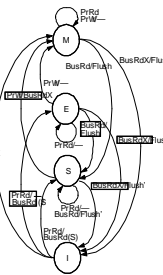
32

Bandwidth per transition

Bus Transaction	Address / Cmd	Data
BusRd	6	64
BusRdX	6	64
BusWB	6	64
BusUpdgd	6	--

Ocean Data Cache Frequency Matrix (per 1000)

	NP	I	E	S	M
NP	0	0	1.25	0.96	0.001
I	0.64	0	0	1.87	0.001
E	0.20	0	14.00	0.0	2.24
S	0.42	2.50	0	134.72	2.24
M	2.63	0.00	0	2.30	843.57



2/12/99

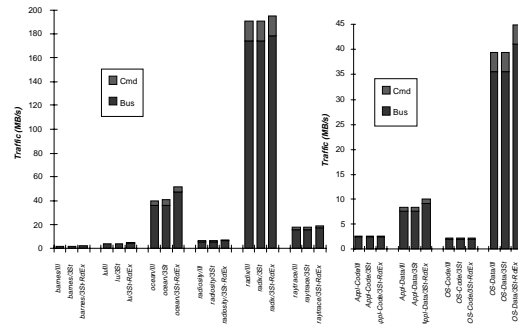
CS258 S99

33

Bandwidth Trade-off

1 MB Cache, 200 MIPS / 200 MFLOPS Processor

E → M are infrequent BusUpgrade is cheap

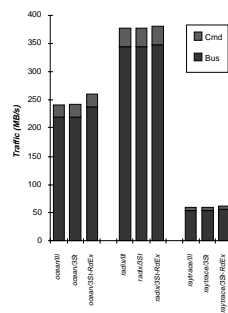


2/12/99

CS258 S99

34

Smaller (64KB) Caches



2/12/99

CS258 S99

35

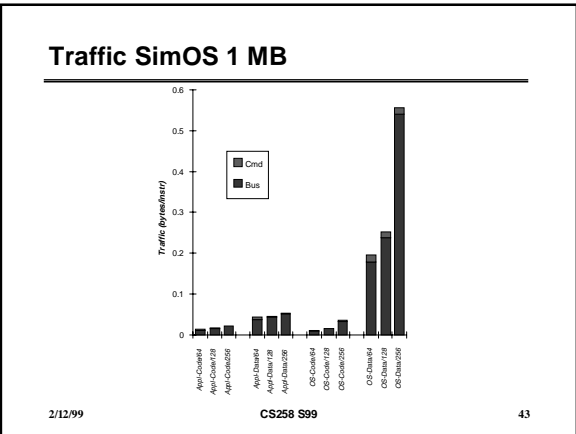
Cache Block Size

- Trade-offs in uniprocessors with increasing block size
 - reduced cold misses (due to spatial locality)
 - increased transfer time
 - increased conflict misses (fewer sets)
- Additional concerns in multiprocessors
 - parallel programs have less spatial locality
 - parallel programs have sharing
 - false sharing
 - bus contention
- Need to classify misses to understand impact
 - cold misses
 - capacity / conflict misses
 - true sharing misses
 - one proc writes words in a block, invalidating a block in another processor's cache, which is later read by that process
 - false sharing misses

2/12/99

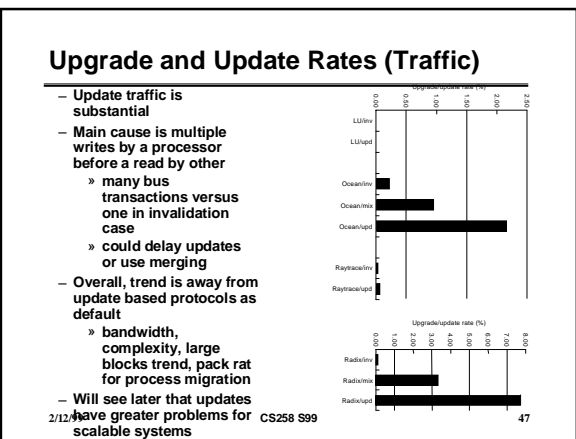
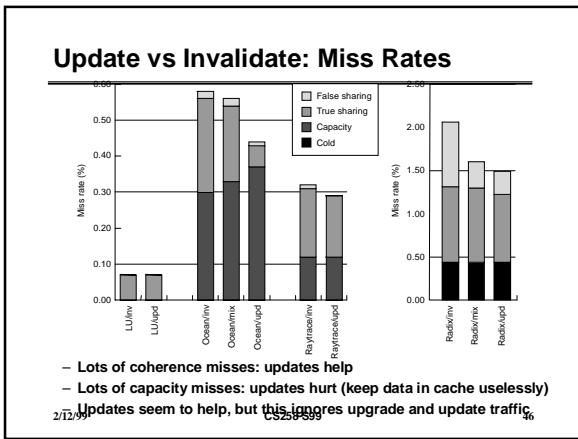
CS258 S99

36



- ### Making Large Blocks More Effective
- **Software**
 - Improve spatial locality by better data structuring (more later)
 - Compiler techniques
 - **Hardware**
 - Retain granularity of transfer but reduce granularity of coherence
 - » use subblocks: same tag but different state bits
 - » one subblock may be valid but another invalid or dirty
 - Reduce both granularities, but prefetch more blocks on a miss
 - Proposals for adjustable cache size
 - More subtle: delay propagation of invalidations and perform all at once
 - » But can change consistency model: discuss later in course
 - Use update instead of invalidate protocols to reduce false sharing effect
- 2/12/99 CS258 S99 44

- ### Update versus Invalidate
- Much debate over the years: tradeoff depends on sharing patterns
 - Intuition:
 - If those that used continue to use, and writes between use are few, update should do better
 - » e.g. producer-consumer pattern
 - If those that use unlikely to use again, or many writes between reads, updates not good
 - » "pack rat" phenomenon particularly bad under process migration
 - » useless updates where only last one will be used
 - Can construct scenarios where one or other is much better
 - Can combine them in hybrid schemes (see text)
 - E.g. competitive: observe patterns at runtime and change protocol
- 2/12/99 CS258 S99 45



- ### Summary
- FSM describes Cache Coherence Algorithm
 - 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
- 2/12/99 CS258 S99 48