Issues in Workload-Driven Evaluation

- Choosing workloads
- Scaling workload and architecture
- Picking parameters and analyzing sensitivity
- Metrics, and presenting results

Some Important Observations

Many important properties of a parallel program depend on:

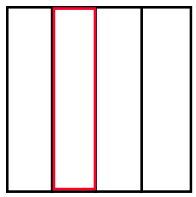
1. How the appl is partitioned/scheduled/orchestrated
2. Application parameters and number of processors
3. Working sets and cache/replication size

We should understand 1 and ensure that the programs we use are appropriate.

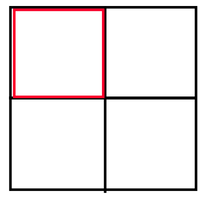
For 2 and 3 we should make sure we cover realistic regimes of operation.

Impact of Partitioning and Orchestration

- Communication to computation ratio
  - **nearest-neighbor on n-by-n grid with p (=4 in figure) processors**



$\frac{2n}{p}$



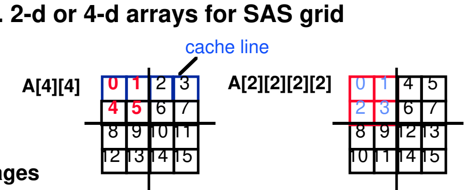
$\frac{4n}{\sqrt{p}}$

- *Communication cost*: long v. short messages
- *Contention*
- *Synchronization frequency and cost*
- *Temporal and spatial locality*

Often different levels of optimization are interesting

Levels of Optimization

- Many ways in which an application can be suboptimal
  - **algorithmic, e.g. blocking, load balancing**
  - **major data structures, e.g. 2-d or 4-d arrays for SAS grid problem**



  - **orchestration**
    - » **contention**
    - » **long versus short messages**
  - **random problems with “unimportant” data structures**
- Optimizing applications takes work
  - **many applications may not be very well optimized**
- May examine selected different levels to test robustness

Appl. Parameters and No. of Processors

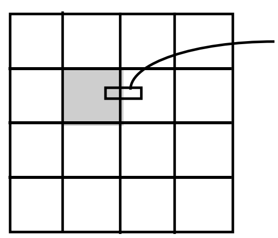
- *Comm. to comp. ratio***:** grid example: $\frac{4n}{\sqrt{p}}$
- Load balance and synch. frequency/cost
- *Working sets,* which affect
  - **local capacity misses**
  - **non-inherent communication**
- Spatial locality and false sharing

e.g. shouldn't use small data set with large number of processors if it doesn't correspond to realistic executions

Need to recognize realistic regimes, and cover those

Example I: Iterative Near-neighbor

n-by-n grid with p processors



n/p is large ⇒
- low communicatio to computation ratio
- good spatial locality with large cache lines
- false-sharing and data distribution not problems even with 2-d array
- working set doesn't fit in cache; high local capacity miss rate.

n/p is small ⇒
- high communication to computation ratio
- spatial locality may be poor; false-sharing may be a problem
- working set fits in cache; low capacity miss rate.

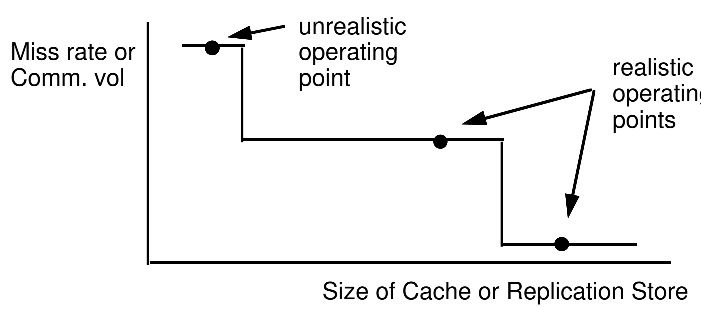
Shouldn't make conclusions about spatial locality based only on small problems, particularly if these are not very representative.

Example 2: Barnes-Hut

- Large problem size (large n or small θ) relative to p $\Rightarrow$ most of the time is in force-computation phase
  - **good load balance**
  - **low communication to computation ratio**
- Small problem size relative to p $\Rightarrow$ much of the time is in tree building phase
  - **poor load balance, large synchronization wait times**
  - **high communication to computation ratio**
- **In general, different problem size relative to p can change the nature of the dominant phase and bottleneck**
  - **e.g. whether it is load imbalance, communication, or local access**

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 communication
- **Some working sets scale with appl. parameters and/or p, some don't**
- **Some operating points are realistic, some aren't**
  - *operating point = f (cache/replication size, appl. parameters, p)*

Working Sets (contd.)

- Important to examine realistic operating points
  - **otherwise can miss important situations, and lead to poor decisions**
- Knee effect can dramatically affect:
  - **miss rate, local misses v. comm, what dominant bottleneck is, ...**
- e.g. in Grid computation, not fitting second WS makes local misses more important relative to communication, and reduces rate of comm.
  - **but can worsen comm. perf due to contention with local accesses**
- in Barnes-Hut, not fitting important WS increases rate of comm.
- **Need to understand growth rates of imp. working sets**
  - **sometimes indept. of data set size, sometimes of p.**
  - **this and knee effect can help prune the experimental space**

Evaluating a Real Machine

- *Machine parameters are fixed*
  - **p, cache size, line size, associativity, msg. overhead, latency, etc.**
- **Choose programs with some diff. levels of optimization**
- **Choose appl parameters: cover realisitic operating points**
  - **relationships among parameters**
  - ***comm. to comp. ratio***
  - ***working sets fitting in cache or replication store***
- **Look for knees in working set curve to prune the space**
- **Two useful ways to evaluate or compare machines:**
  - **How do they perform on a range of realistic operating points?**
  - ***What are the smallest and largest problems that they can run efficiently (by some measure)?***
    - » **but beware of artifacts at small end that are unlikely in practice**

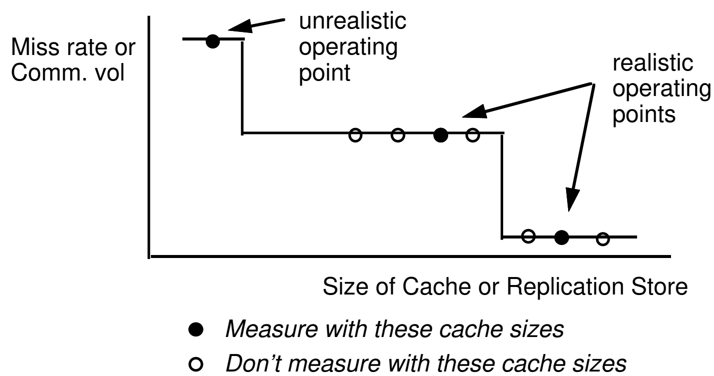
Evaluating an Idea Before Building It

- Two major problems:
  - **Huge design space**
    - » **application parameters (as before)**
    - » **number of processors**
    - » **cache/replication size**
    - » **associativity**
    - » **granularities of allocation, transfer, coherence**
    - » **comm. parameters (latency, bandwidth, occupancies)**
  - **Simulation is expensive**
    - » **cannot simulate the problem/machine sizes we care about**
    - » **often have to use scaled down problem sizes**

Pruning the Design Space

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

Example of Pruning using Knees



- **But be careful: applicability depends on what is being evaluated** (see update/invalidate protocols later)
- If growth rate of important WS can be predicted, can prune for other configurations too
- Often knees are not sharp

Another Example: Spatial Locality

- Some features are very sensitive to transfer granularity
  – **e.g. software-controlled prefetching**
- If we know program has perfect spatial locality, can predict impact of changing granularity
- Otherwise, have to perform sensitivity analysis
  – **unless we only care about particular granularities**

Which Parameters Can We Prune?

- **No. of Processors** *p*
- **Problem parameters:** data set size *n* etc.
- Cache/Replication size
- Transfer granularity
- Coherence granularity
- Allocation granularity
- Cache/Replication associativity
- Latency
- Bandwidth (node-to-network and bisection)
- Occupancies

Pruning Parameters (contd.)

- *Processors p*
  - **try to use the numbers of interest. If cannot, use growth rate analysis if possible**
- *Appl. parameters:* **pick some realistic sets**
  - **1. toward smallest end of what is reasonable for that p**
  - **2. toward max of what is reasonable and what can be simulated**
  - **3. in the middle, if 2. is very large**
- *Cache/Replication Size*
  - **use working sets approach to prune space**
- Granuarlities of Allocation, Transfer, Coherence
  - **either have exact knowledge or spatial locality, or**
  - **pick interesting points and do sensitivity analysis**

Pruning Parameters (contd.)

- *Associativity*
  - effect usually flattens quickly, so can prune
- Transfer/Communication Latencies
  - pick interesting points and do sensitivity analysis
- *Bandwidth*
  - doesn't matter until it matters (often knee-like saturation effect), so can prune
- *Occupancy / overhead*
  - pick interesting points and do sensitivity analysis. Sometimes knee-like saturation effect, so can prune.

Scaling Down Parameters for Simulation

- *Problem*: **Simulation is expensive in time and memory, so can't run very large problems.**
-
- *Questions*:
  - 1. How to scale down problem parameters to make simulation faster and consume less memory
  -
  - 2. How to scale machine parameters to match the scaling down of problem parameters

Scaling Down Problem Parameters

- Some parameters don't affect parallel performance much, but do affect runtime, and can be scaled down
  - **common example is number of time-steps in regular scientific appls**
  - **may need to keep a few to allow settling down, but don't need more**
  -
  -
- Other parameters can be scaled down according to earlier scaling arguments

Scaling Down Machine Parameters

- *Processors p*
  - **May have to scale down: behavior for small problems may be unrepresentative**
  - **e.g. may try to preserve comm-to-comp ratio of full problem**
  - **but subject to doubt: relative patterns, contention may change**
- *Cache/replication size*
  - **naive solution: decrease cache proportional to problem or data set**
  - **but cache need is function of working set, not data set or runtime**
  - **should be guided by growth rates of working sets**

Scaling Machine Parameters (contd.)

• *Granularity of allocation, transfer and coherence*

– correct solution different for different types of applications

– **difficult to find solution that works** for both comm and local access

• *Associativity*

– should stay unchanged, but again difficult

– like line size, assoc can have greater impact with scaled down caches

Conclusions for Scaled Data Sets

- A hard problem; difficult to have confidence
- Cache/replication sizes should be scaled according to working sets, not data set or problem size
- Other parameters more difficult
- Solutions and confidence levels application-specific