
Performance Issues*

Zeljko Zilic

McConnell Engineering Building

Room 546



McGill

* Ch. 3 in the textbook



McGill

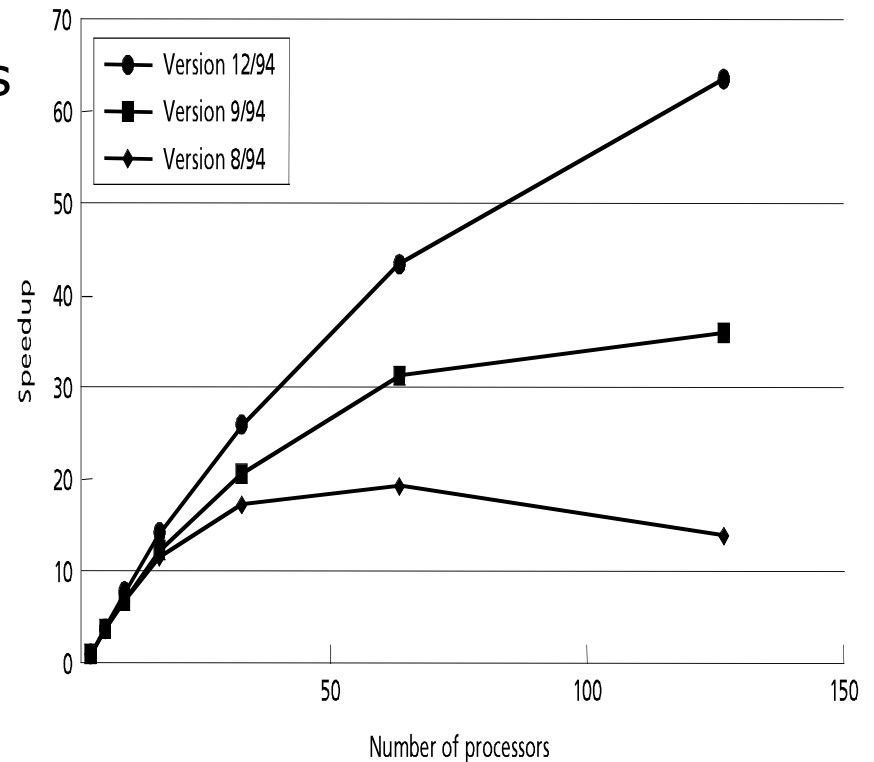
Overview

- Process of creating a parallel program
- Performance issues
- Architectural interactions
 - Three major programming models
 - What primitives must a system support?



Performance Goal: Speedup

- Architect Goal
 - Observe how program uses machine and improve the **design** to enhance performance
- Programmer Goal
 - observe how the program uses the machine and improve the **implementation** to enhance performance
- What do you observe?
- Who fixes what?



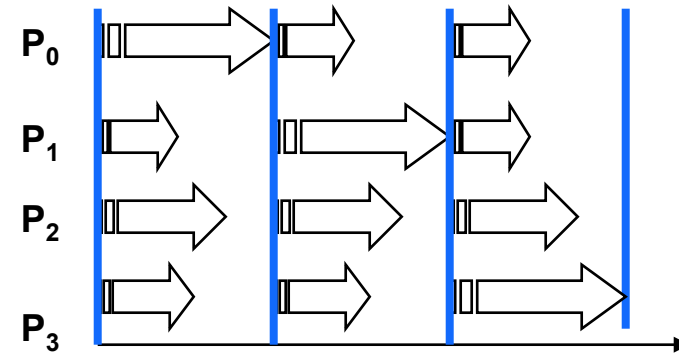
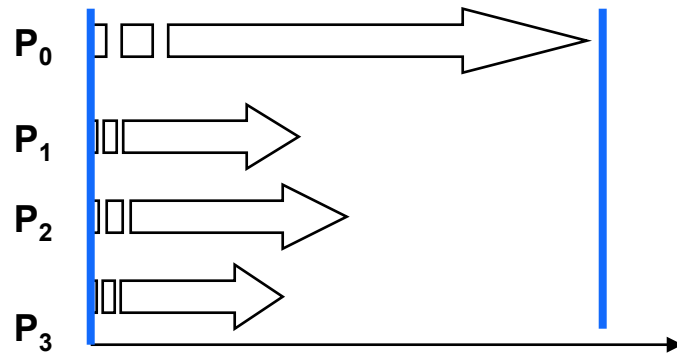
Analysis Framework

$$\text{Speedup}_p < \frac{\text{Sequential Work}}{\text{Max (Work + Synch Wait Time + Comm Cost + Extra Work)}}$$

- Communication and load balance NP-hard in general
 - Heuristic solutions work well in practice
- Fundamental Tension among:
 - Balanced load
 - Minimal synchronization
 - Minimal communication
 - Minimal extra work
- Good machine design mitigates the trade-offs

Load Balance and Synchronization

$$\text{Speedup}_{\text{problem}}(p) \leq \frac{\text{Sequential Work}}{\text{Max Work on any Processor}}$$



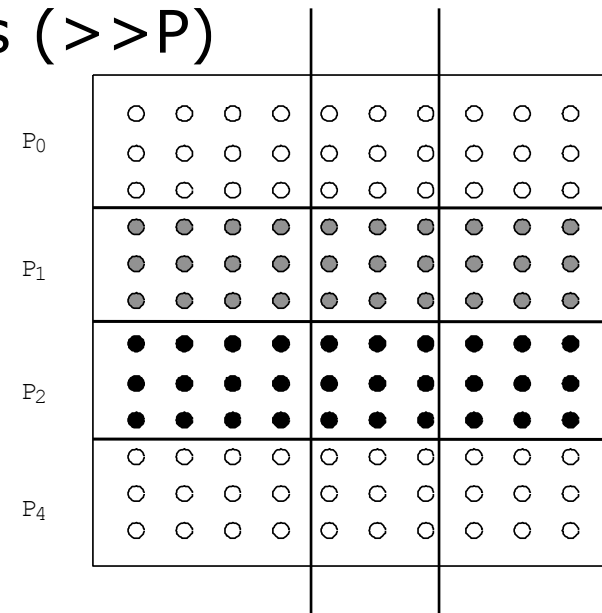
- Instantaneous load imbalance revealed as wait time

- at completion
- at barriers
- at receive
- at flags, even at mutex

$$\frac{\text{Sequential Work}}{\text{Max (Work + Synch Wait Time)}}$$

Improving Load Balance

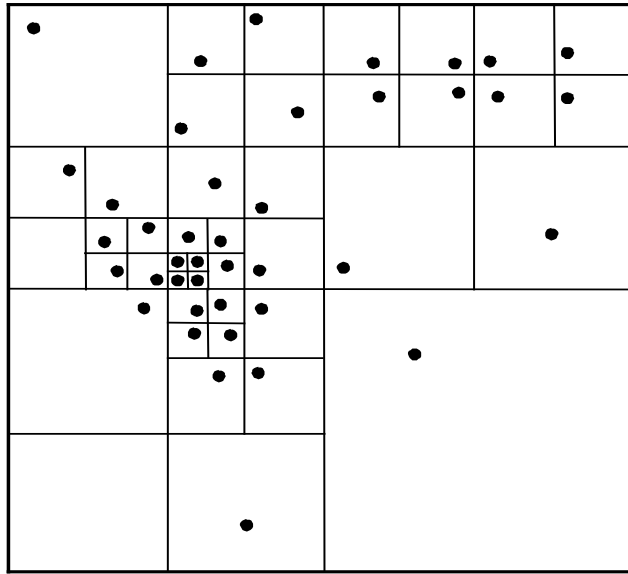
- Decompose into more smaller tasks ($\gg P$)
- Distribute uniformly
 - Variable-sized task
 - Randomize
 - Bin packing
 - Dynamic assignment
- Schedule more carefully
 - Avoid serialization
 - Estimate work
 - Use history info.



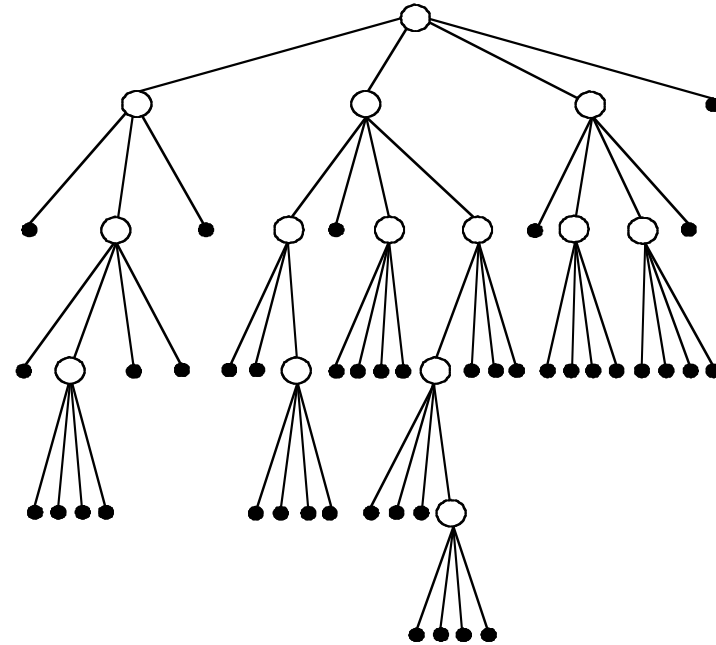
```

for_all i = 1 to n do
  for_all j = i to n do
    A[ i, j ] = A[i-1, j] + A[i, j-1] + ...
    
```

Example: Barnes-Hut



(a) The spatial domain

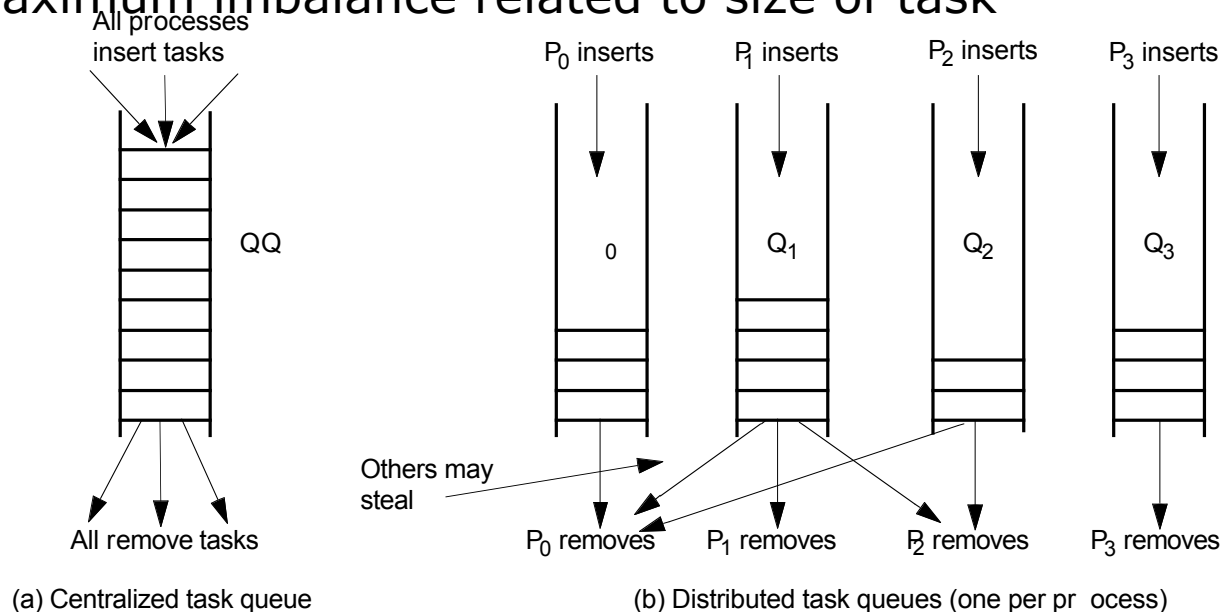


(b) Quadtree representation

- Divide space into roughly equal # particles
- Particles close together in space -> same processor
- Nonuniform, dynamically changing

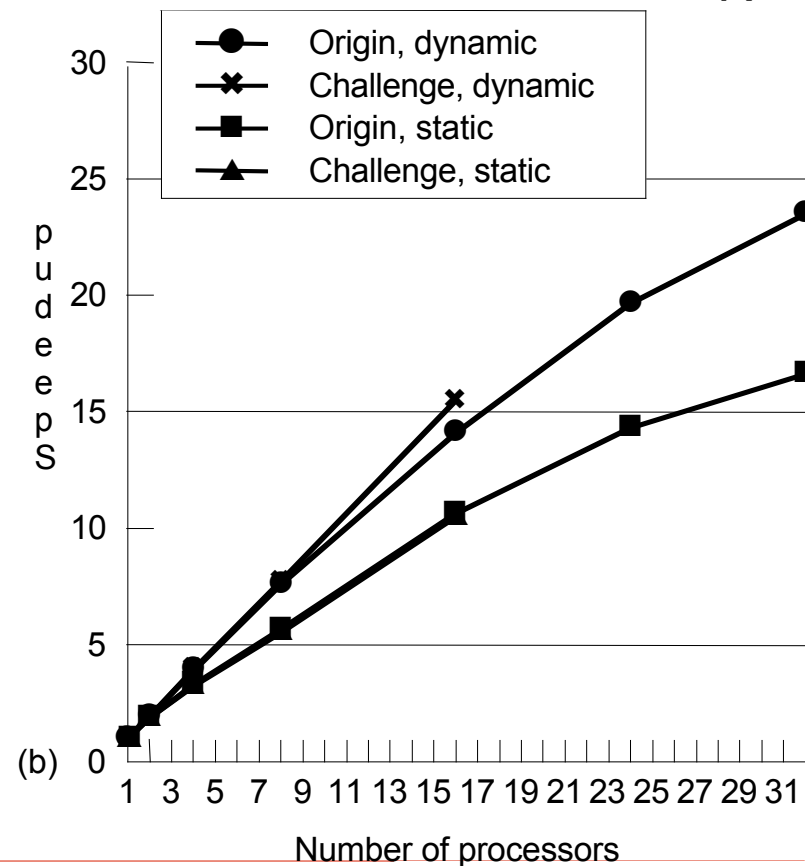
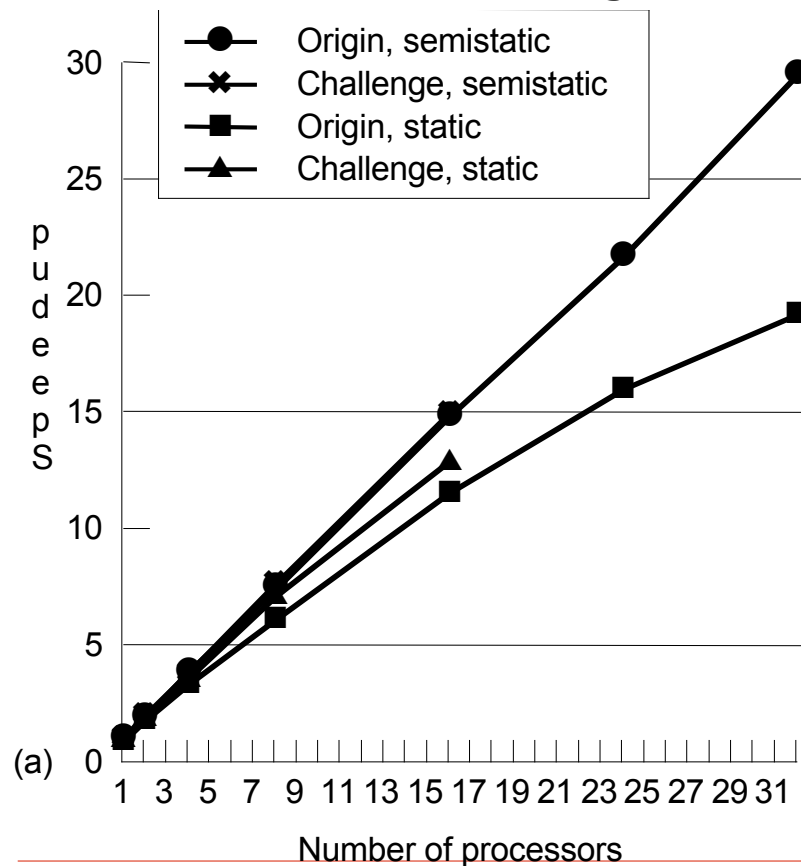
Dynamic Scheduling: Task Queues

- Centralized versus distributed queues
- Task stealing with distributed queues
 - Compromise comm & locality, increase synchronization
 - Whom to steal from, how many tasks to steal, when done...
 - Maximum imbalance related to size of task



Impact of Dynamic Assignment

- Barnes-Hut on SGI Origin 2000 (cache-coherent shared memory):



Self-Scheduling

```
volatile int row_index = 0;      /* shared index variable */

while (not done) {
    initialize row_index; barrier;
    while ((i = fetch_and_inc(&row_index) < n) {
        for (j = i; j < n; j++) {
            A[ i, j ] = A[i-1, j] + A[i, j-1] + ...
        }
    }
}
```



Reducing Serialization

- Careful about assignment and orchestration
 - Including scheduling
- Event synchronization
 - Reduce use of conservative synchronization
 - e.g. point-to-point instead of barriers, or granularity of pt-to-pt
 - But fine-grained synch more difficult to program, more synch ops.
- Mutual exclusion
 - Separate locks for separate data
 - e.g. locking records in a database: lock per process, record, or field
 - Lock per task in task queue, not per queue
 - Finer grain => less contention/serialization, more space, less reuse
 - Smaller, less frequent critical sections
 - Don't do reading/testing in critical section, only modification
 - Stagger critical sections in time

Impact of Load Balance Effort

- Parallelism Management overhead?
- Communication?
 - Amount, size, frequency?
- Synchronization?
 - Type? frequency?
- Opportunities for replication?
- What can architecture do?

Arch. Implications of Load Balance

- Naming
 - Global position-independent naming separates decomposition from layout
 - Allows diverse, even dynamic assignments
- Efficient Fine-grained communication & synch
 - More, smaller
 - Msgs
 - Locks
 - Point-to-point
- Automatic replication

Reducing Extra Work

- Common sources of extra work:
 - Computing a good partition
 - e.g. partitioning in Barnes-Hut or sparse matrix
 - Using redundant computation to avoid communication
 - Task, data and process management overhead
 - Applications, languages, runtime systems, OS
 - Imposing structure on communication
 - Coalescing messages, allowing effective naming
- Architectural Implications:
 - Reduce need by making comm. and orchestration efficient

$$\text{Speedup} \leq \frac{\text{Sequential Work}}{\text{Max (Work + Synch Wait Time + Comm Cost + Extra Work)}}$$

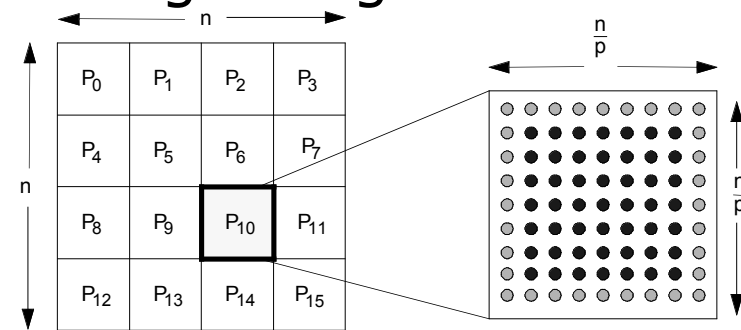
Reducing Inherent Communication

$$\text{Speedup} \leq \frac{\text{Sequential Work}}{\text{Max (Work + Synch Wait Time + Comm Cost)}}$$

- Communication is expensive!
 - Measure: *communication to computation ratio*
 - Inherent communication
 - Determined by assignment of tasks to processes
 - One produces data consumed by others
- => Use algorithms that communicate less
- => Assign tasks that access same data to same process
- same row or block to same process in each iteration

Domain Decomposition

- Works well for scientific, engineering, graphics, ... applications
- Exploits local-biased nature of physical problems
 - Information requirements often short-range
 - Or long-range but fall off with distance
- Simple example: nearest-neighbor grid computation

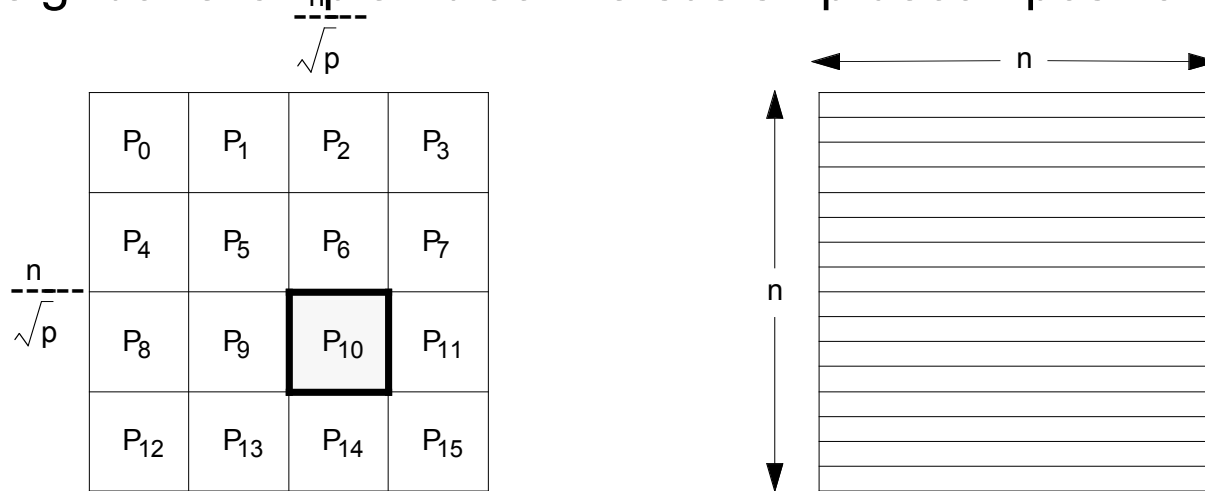


Perimeter to Area comm-to-comp ratio (area to volume in 3-d)

• Depends on n, p : decreases with n , increases with p

Domain Decomposition (contd)

Best domain decomposition depends on information requirements
Nearest neighbor example: block versus strip decomposition:



- Comm to comp: $\frac{4*\sqrt{p}}{n}$ for block, $\frac{2*p}{n}$ for strip
- Application dependent: strip may be better in other cases
 - E.g. particle flow in tunnel

Relation to load balance

- Scatter Decomposition, e.g. initial partition in Raytrace

12	
3	4

Domain decomposition

Preserve locality in task stealing

- Steal large tasks for locality, steal from same queues, ...

12		12		12		12	
3	4	3	4	3	4	3	4
12		12		12		12	
3	4	3	4	3	4	3	4
12		12		12		12	
3	4	3	4	3	4	3	4
12		12		12		12	
3	4	3	4	3	4	3	4

Scatter decomposition

Implications of Comm-to-Comp Ratio

- Architects examine application needs to see where to spend effort
 - bandwidth requirements (operations / sec)
 - latency requirements (sec/operation)
 - time spent waiting
- Actual impact of comm. depends on structure and cost as well
- Need to keep communication balanced across processors as well

$$\text{Speedup}_n < \frac{\text{Sequential Work}}{\text{Max (Work + Synch Wait Time + Comm Cost)}}$$

Structuring Communication

- Given amount of comm., goal is to reduce cost
- Cost of communication as seen by process:

$$C = f * (o + l + \frac{n_c/m}{B} + t_c - overlap)$$

- f = frequency of messages
 - o = overhead per message (at both ends)
 - l = network delay per message
 - n_c = total data sent
 - m = number of messages
 - B = bandwidth along path (determined by network, NI, assist)
 - t_c = cost induced by contention per message
 - *overlap* = amount of latency hidden by overlap with comp. or comm.
- Portion in parentheses is cost of a message (as seen by processor)
 - ignoring overlap, is *latency* of a message
 - Goal: reduce terms in latency and increase overlap

Reducing Overhead

- Can reduce no. of messages m or overhead per message o
- o is usually determined by hardware or system software
 - Program should try to reduce m by coalescing messages
 - More control when communication is explicit
- Coalescing data into larger messages:
 - Easy for regular, coarse-grained communication
 - Can be difficult for irregular, naturally fine-grained communication
 - may require changes to algorithm and extra work
 - coalescing data and determining what and to whom to send
 - will discuss more in implications for programming models later

Reducing Network Delay

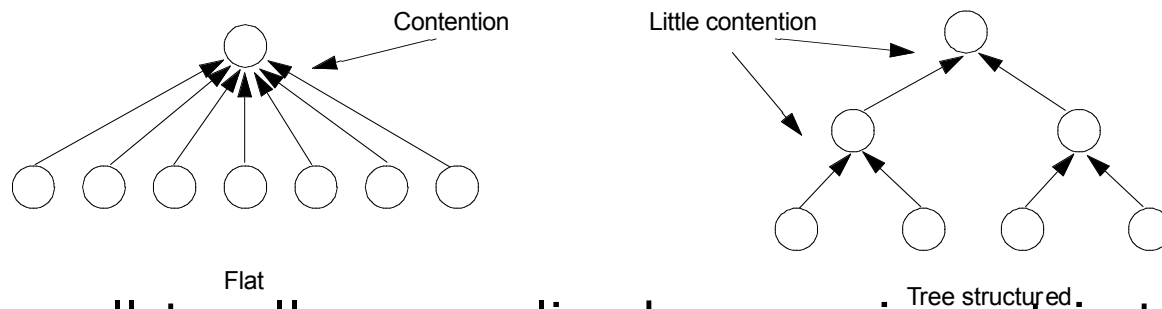
- Network delay component = $f * h * t_h$
 - h = number of hops traversed in network
 - t_h = link+switch latency per hop
- Reducing f : communicate less, or make messages larger
- Reducing h :
 - Map communication patterns to network topology
 - e.g. nearest-neighbor on mesh and ring; all-to-all
 - How important is this?
 - used to be major focus of parallel algorithms
 - depends on no. of processors, how t_h , compares with other components
 - less important on modern machines
 - overheads, processor count, multiprogramming

Reducing Contention

- All resources have nonzero occupancy
 - Memory, communication controller, network link, etc.
 - Can only handle so many transactions per unit time
- Effects of contention:
 - Increased end-to-end cost for messages
 - Reduced available bandwidth for individual messages
 - Causes imbalances across processors
- Particularly insidious performance problem
 - Easy to ignore when programming
 - Slow down messages that don't even need that resource
 - by causing other dependent resources to also congest
 - Effect can be devastating: *Don't flood a resource!*

Types of Contention

- Network contention and end-point contention (hot-spots)
- Location and Module Hot-spots
 - Location: e.g. accumulating into global variable, barrier
 - solution: tree-structured communication



- Module: all-to-all personalized comm. in matrix transpose
 - solution: stagger access by different processors to same node temporally
- In general, reduce burstiness; may conflict with making messages larger

Overlapping Communication

- Cannot afford to stall for high latencies
 - Even on uniprocessors!
- Overlap with computation or communication to hide latency
- Requires extra concurrency (*slackness*), higher bandwidth
- Techniques:
 - Prefetching
 - Block data transfer
 - Proceeding past communication
 - Multithreading

What is a Multiprocessor?

- A collection of communicating processors
 - View taken so far
 - Goals: balance load, reduce inherent communication and extra work
- A multi-cache, multi-memory system
 - Role of these components essential regardless of programming model
 - Prog. model and comm. abstr. affect specific performance tradeoffs

Memory-oriented View

- Multiprocessor as Extended Memory Hierarchy
 - as seen by a given processor
- Levels in extended hierarchy:
 - Registers, caches, local memory, remote memory (topology)
 - Glued together by communication architecture
 - Levels communicate at a certain granularity of data transfer
- Need to exploit spatial and temporal locality in hierarchy
 - Otherwise extra communication may also be caused
 - Especially important since communication is expensive

Uniprocessor

- Performance depends heavily on memory hierarchy
- Time spent by a program

$$Time_{prog}(1) = Busy(1) + Data\ Access(1)$$

- Divide by cycles to get CPI equation
- Data access time can be reduced by:
 - Optimizing machine: bigger caches, lower latency...
 - Optimizing program: temporal and spatial locality

Extended Hierarchy

- Idealized view: local cache hierarchy + single main memory
- But reality is more complex
 - Centralized Memory: caches of other processors
 - Distributed Memory: some local, some remote; + network topology
 - Management of levels
 - caches managed by hardware
 - main memory depends on programming model
 - SAS: data movement between local and remote transparent
 - message passing: explicit
 - Levels closer to processor are lower latency and higher bandwidth
 - Improve performance through architecture or program locality
 - Tradeoff with parallelism; need good node performance and parallelism

Artifactual Communication

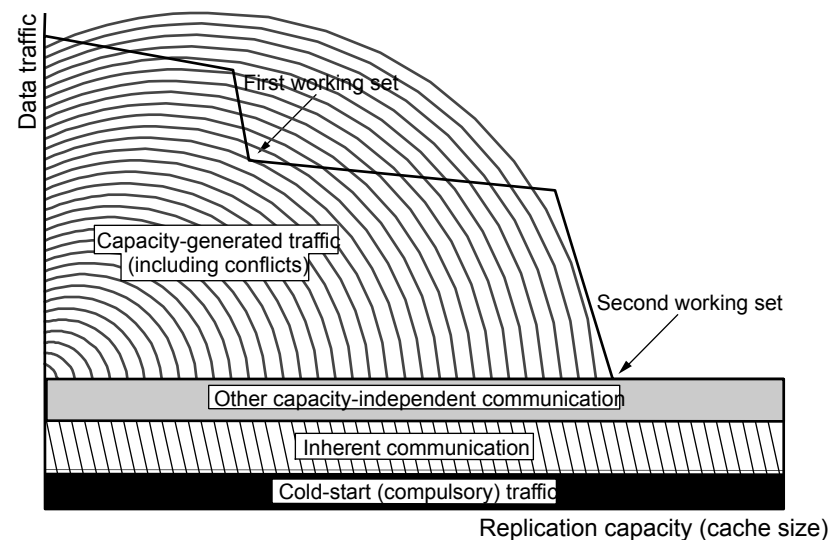
- Accesses not satisfied in local portion of memory hierarchy cause communication
 - Inherent communication, implicit or explicit, causes transfers
 - determined by program
 - Artifactual communication
 - determined by program implementation and arch. interactions
 - poor allocation of data across distributed memories
 - unnecessary data in a transfer
 - unnecessary transfers due to system granularities
 - redundant communication of data
 - finite replication capacity (in cache or main memory)
 - Inherent communication assumes unlimited capacity, small transfers, perfect knowledge of what is needed.

Communication and Replication

- Comm induced by finite capacity is most fundamental artifact
 - Like cache size and miss rate or memory traffic in uniprocessors
 - Extended memory hierarchy view useful for this relationship
- View as three level hierarchy for simplicity
 - Local cache, local memory, remote memory (ignore network topology)
- 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)
 - Each may be helped/hurt by large transfer granularity (spatial locality)

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)

Orchestration for Performance

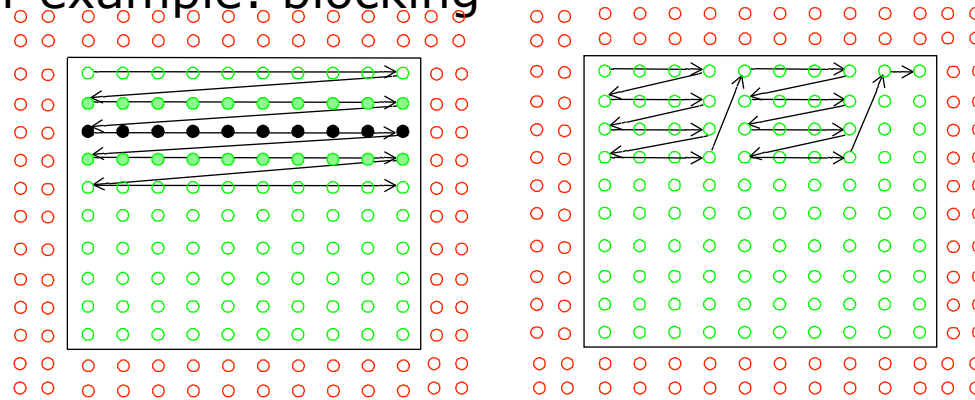
- Reducing amount of communication:
 - Inherent: change logical data sharing patterns in algorithm
 - Artifactual: exploit spatial, temporal locality in extended hierarchy
 - Techniques often similar to those on uniprocessors
- Structuring communication to reduce cost

Reducing Artifactual Communication

- Message passing model
 - Communication and replication are both explicit
 - Even artifactual communication is in explicit messages
 - send data that is not used
- Shared address space model
 - More interesting from an architectural perspective
 - Occurs transparently due to interactions of program and system
 - sizes and granularities in extended memory hierarchy
- Use shared address space to illustrate issues

Exploiting Temporal Locality

- Structure algorithm so working sets map well to hierarchy
 - often techniques to reduce inherent communication do well here
 - schedule tasks for data reuse once assigned
- Multiple data structures in same phase
 - e.g. database records: local versus remote
- Solver example: blocking



(a) Unblocked access pattern in a sweep

(b) Blocked access pattern with $B = 4$

- More useful when $O(n^{k+1})$ computation on $O(n^k)$ data
 - many linear algebra computations (factorization, matrix multiply)

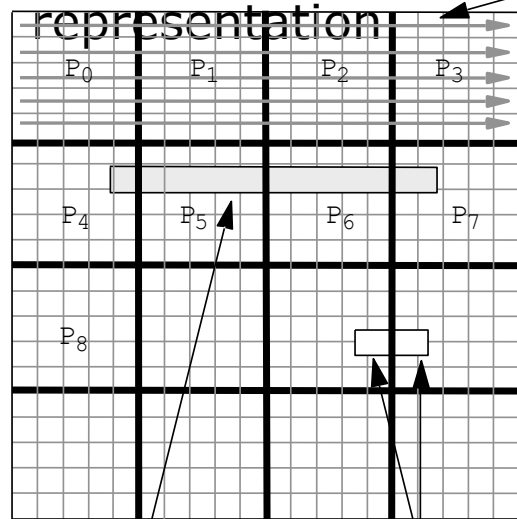
Exploiting Spatial Locality

- Besides capacity, granularities are important:
 - Granularity of allocation
 - Granularity of communication or data transfer
 - Granularity of coherence
- Major spatial-related causes of artifactual communication:
 - Conflict misses
 - Data distribution/layout (allocation granularity)
 - Fragmentation (communication granularity)
 - False sharing of data (coherence granularity)
- All depend on how spatial access patterns interact with data structures
 - Fix problems by modifying data structures, or layout/alignment
- Examine later in context of architectures
 - one simple example here: data distribution in SAS solver

Spatial Locality Example

- Repeated sweeps over 2-d grid, each time adding 1 to elements

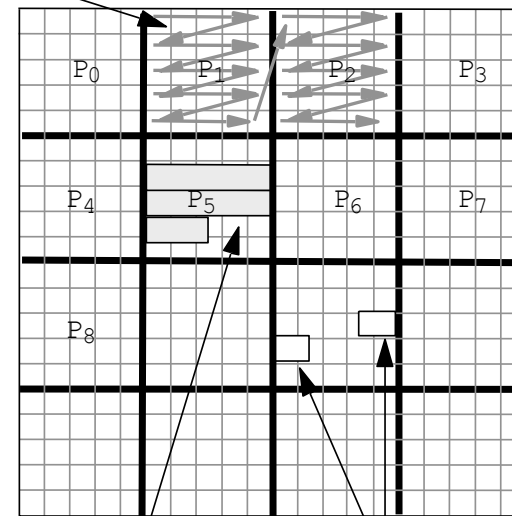
- Natural 2-d versus higher-dimensional array representation



Page straddles partition boundaries: difficult to distribute memory well

Cache block straddles partition boundary

(a) Two-dimensional array



Page does not straddle partition boundary

Cache block is within a partition

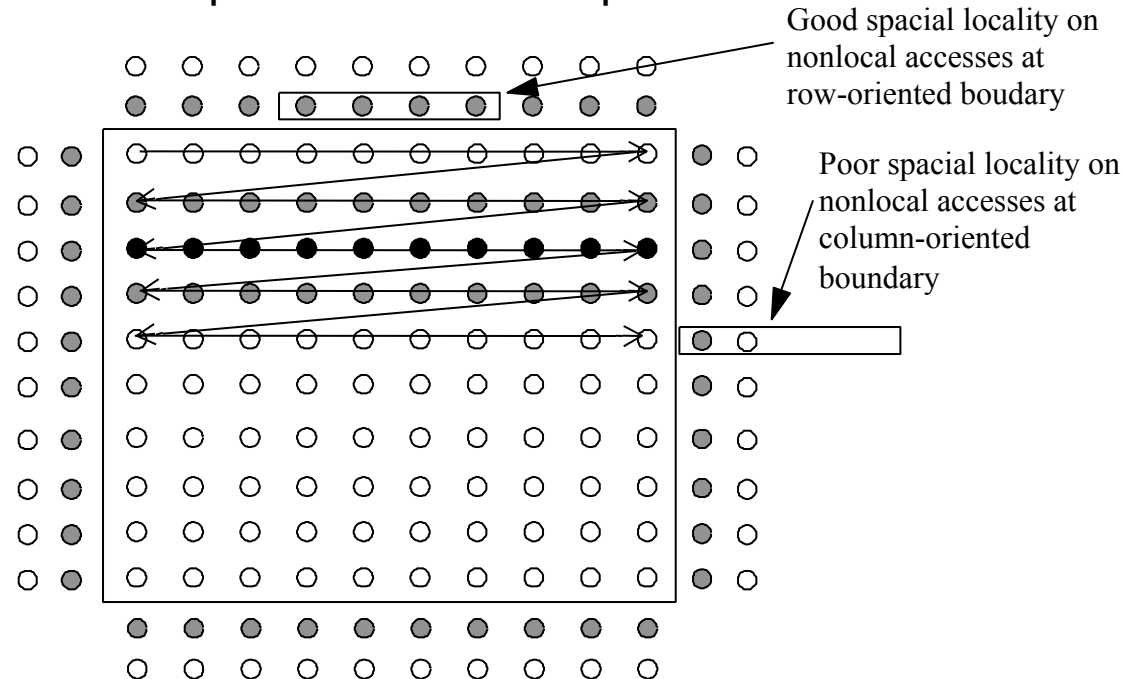
(b) Four-dimensional array

Architectural Implications of Locality

- Communication abstraction that makes exploiting it easy
- For cache-coherent SAS, e.g.:
 - Size and organization of levels of memory hierarchy
 - cost-effectiveness: caches are expensive
 - caveats: flexibility for different and time-shared workloads
 - Replication in main memory useful? If so, how to manage?
 - hardware, OS/runtime, program?
 - Granularities of allocation, communication, coherence (?)
 - small granularities => high overheads, but easier to program
- Machine granularity (resource division among processors, memory...)

Tradeoffs with Inherent Communication

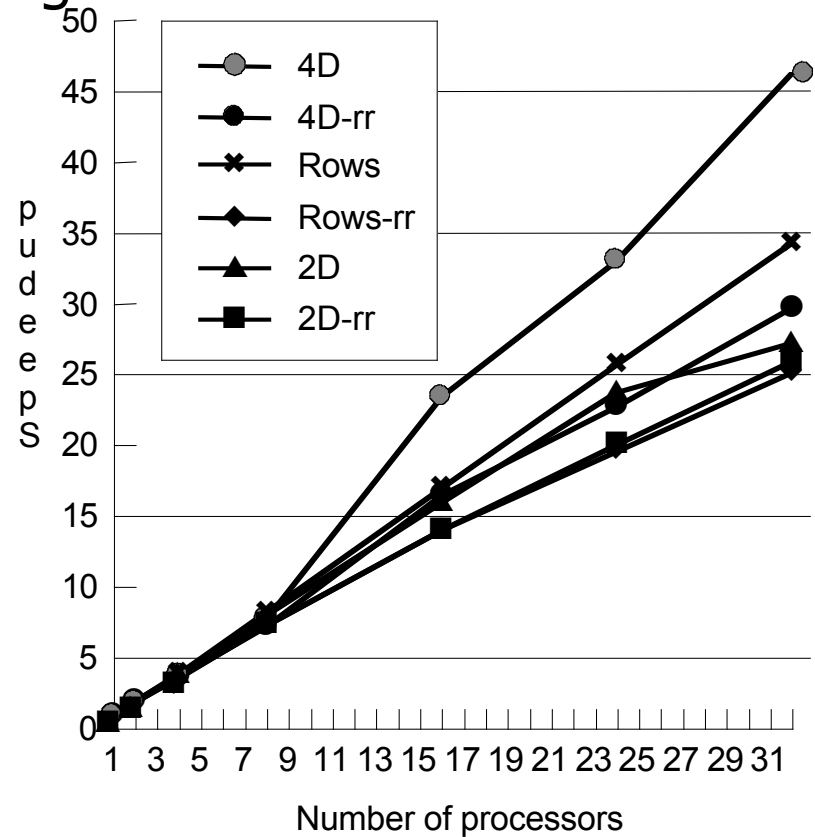
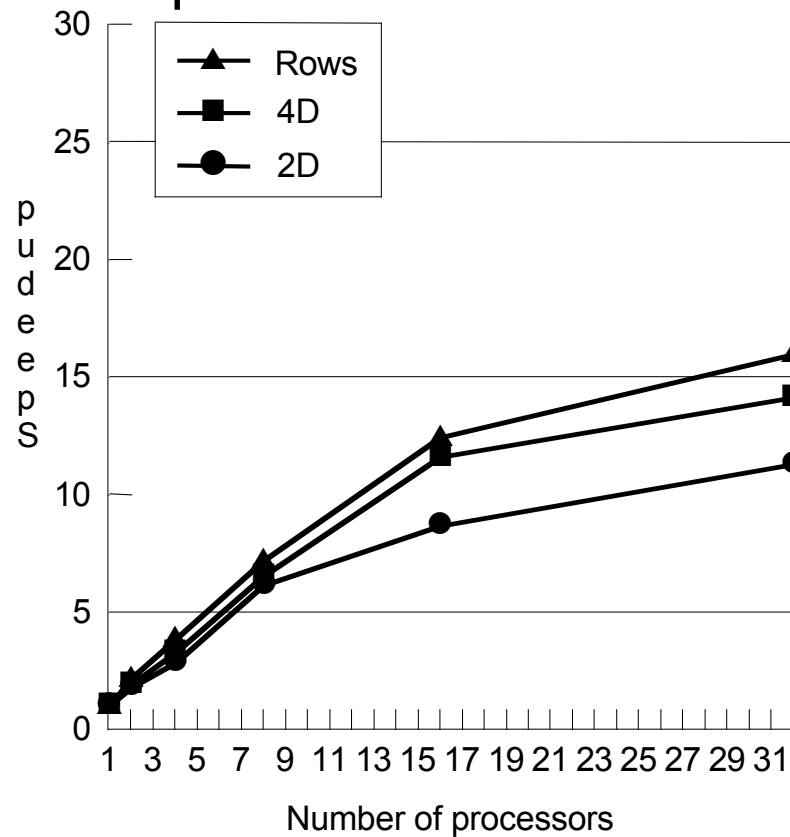
- Partitioning grid solver: blocks versus rows
 - Blocks still have a spatial locality problem on remote data
 - Rowwise can perform better despite worse inherent c-to-c ratio



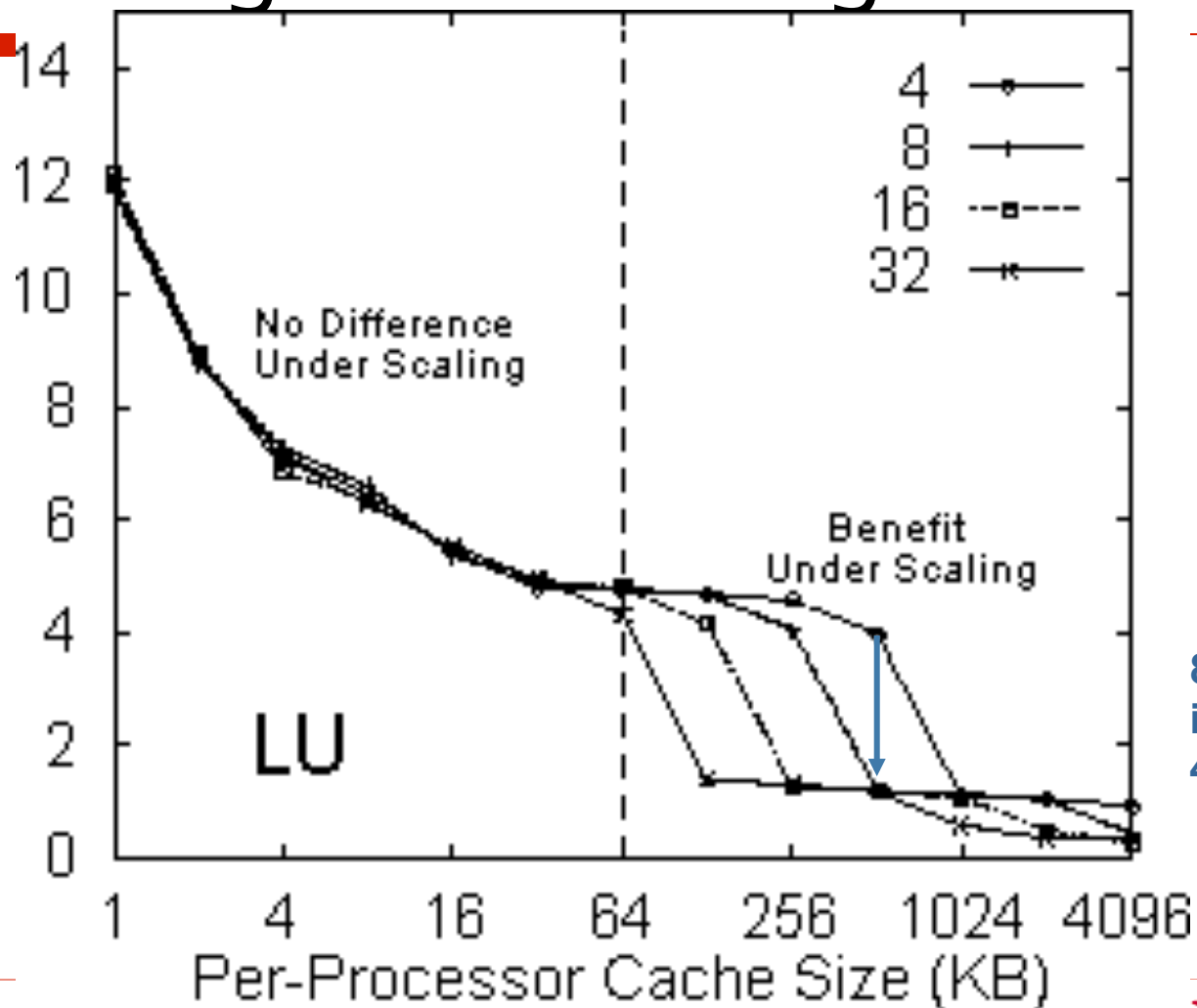
•Result depends on n and p

Example Performance Impact

○ Equation solver on SGI Origin2000

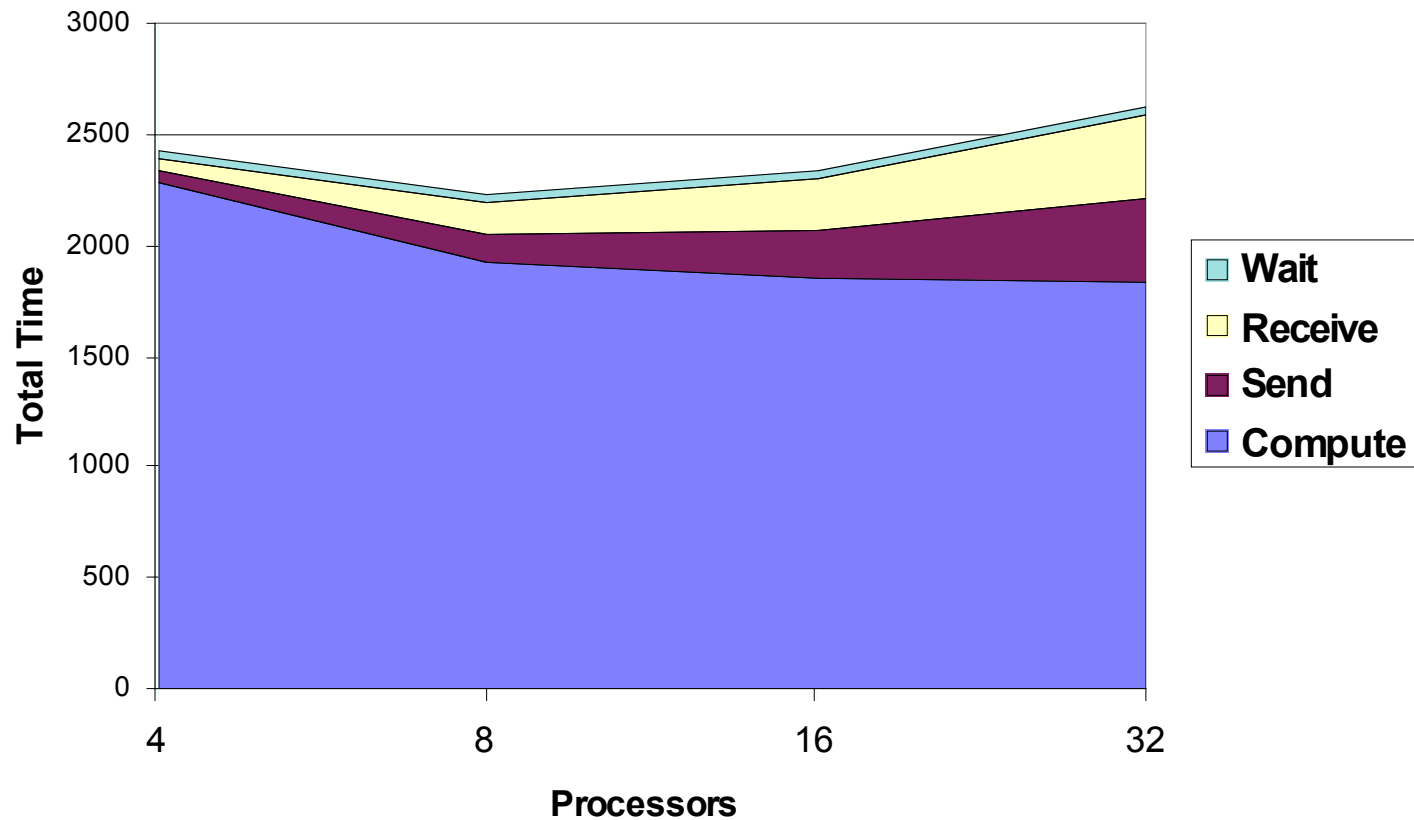


Working Sets Change with P



8-fold reduction
in miss rate from
4 to 8 proc

Where the Time Goes: LU-a



Summary of Tradeoffs

- Different goals often have conflicting demands
 - Load Balance
 - fine-grain tasks
 - random or dynamic assignment
 - Communication
 - usually coarse grain tasks
 - decompose to obtain locality: not random/dynamic
 - Extra Work
 - coarse grain tasks
 - simple assignment
 - Communication Cost:
 - big transfers: amortize overhead and latency
 - small transfers: reduce contention

