

Parallel Programming: Performance

Todd C. Mowry
CS 418
January 22, 27 & 28, 2004

Introduction

Rich space of techniques and issues

- Trade off and interact with one another

Issues can be addressed/helped by software or hardware

- Algorithmic or programming techniques
- Architectural techniques

Focus here on performance issues and software techniques

- Point out some architectural implications
- Architectural techniques covered in rest of class

- 2 -

CS 418 S'04

Programming as Successive Refinement

Not all issues dealt with up front

Partitioning often independent of architecture, and done first

- View machine as a **collection of communicating processors**
 - balancing the workload
 - reducing the amount of inherent communication
 - reducing extra work

- **Tug-o-war even among these three issues**

Then interactions with architecture

- View machine as **extended memory hierarchy**
 - extra communication due to architectural interactions
 - cost of communication depends on how it is structured

- **May inspire changes in partitioning**

Discussion of issues is one at a time, but identifies tradeoffs

- Use examples, and measurements on SGI Origin2000

- 3 -

CS 418 S'04

Outline

Partitioning for performance

Relationship of communication, data locality and architecture

Programming for performance

For each issue:

- Techniques to address it, and tradeoffs with previous issues
- Illustration using case studies
- Application to grid solver
- Some architectural implications

Components of execution time as seen by processor

- What workload looks like to architecture, and relate to software issues

- 4 -

CS 418 S'04

Partitioning for Performance

Balancing the workload and reducing wait time at synch points

Reducing inherent communication

Reducing extra work

Even these algorithmic issues trade off:

- Minimize comm. => run on 1 processor => extreme load imbalance
- Maximize load balance => random assignment of tiny tasks => no control over communication
- Good partition may imply extra work to compute or manage it

Goal is to compromise

- Fortunately, often not difficult in practice

- 5 -

CS 418 S'04

Load Balance and Synch Wait Time

Limit on speedup: $Speedup_{problem}(p) \leq \frac{Sequential\ Work}{Max\ Work\ on\ any\ Processor}$

- Work includes data access and other costs
- Not just equal work, but must be busy at same time

Four parts to load balance and reducing synch wait time:

1. Identify enough concurrency
2. Decide how to manage it
3. Determine the granularity at which to exploit it
4. Reduce serialization and cost of synchronization

- 6 -

CS 418 S'04

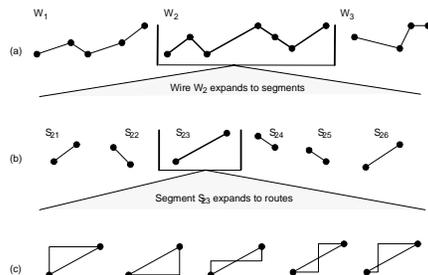
Identifying Concurrency

Techniques seen for equation solver:

- Loop structure, fundamental dependences, new algorithms

Data Parallelism versus Function Parallelism

Often see orthogonal levels of parallelism; e.g. VLSI routing



- 7 -

CS 418 S'04

Identifying Concurrency (contd.)

Function parallelism:

- entire large tasks (procedures) that can be done in parallel
- on same or different data
- e.g. different independent grid computations in Ocean
- pipelining, as in video encoding/decoding, or polygon rendering
- degree usually modest and does not grow with input size
- difficult to load balance
- often used to reduce synch between data parallel phases

Most scalable programs data parallel (per this loose definition)

- function parallelism reduces synch between data parallel phases

- 8 -

CS 418 S'04

Deciding How to Manage Concurrency

Static versus Dynamic techniques

Static:

- Algorithmic assignment based on input; won't change
- Low runtime overhead
- Computation must be predictable
- Preferable when applicable (except in multiprogrammed or heterogeneous environment)

Dynamic:

- Adapt at runtime to balance load
- Can increase communication and reduce locality
- Can increase task management overheads

- 9 -

CS 418 S'04

Dynamic Assignment

Profile-based (semi-static):

- Profile work distribution at runtime, and repartition dynamically
- Applicable in many computations, e.g. Barnes-Hut, some graphics

Dynamic Tasking:

- Deal with unpredictability in program or environment (e.g. Raytrace)
 - computation, communication, and memory system interactions
 - multiprogramming and heterogeneity
 - used by runtime systems and OS too
- Pool of tasks; take and add tasks until done
- E.g. "self-scheduling" of loop iterations (shared loop counter)

- 10 -

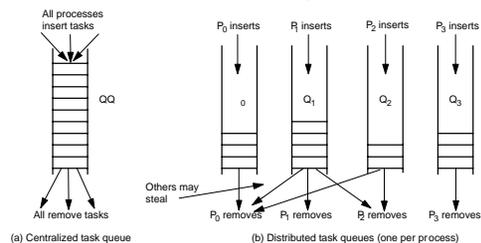
CS 418 S'04

Dynamic Tasking with Task Queues

Centralized versus distributed queues

Task stealing with distributed queues

- Can compromise comm and locality, and increase synchronization
- Whom to steal from, how many tasks to steal, ...
- Termination detection
- Maximum imbalance related to size of task

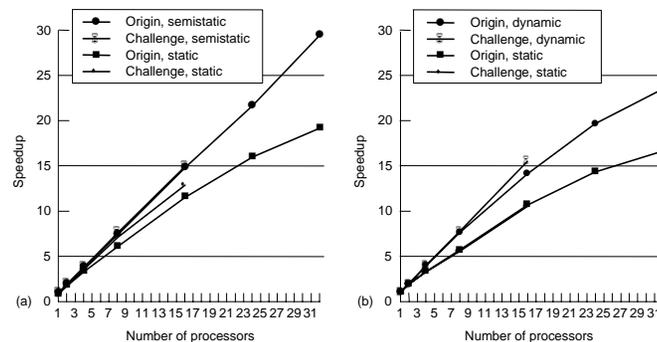


- 11 -

CS 418 S'04

Impact of Dynamic Assignment

On SGI Origin 2000 (cache-coherent shared memory):



- 12 -

CS 418 S'04

Determining Task Granularity

Task granularity: amount of work associated with a task

General rule:

- Coarse-grained => often less load balance
- Fine-grained => more overhead; often more communication & contention

Communication & contention actually affected by assignment, not size

- Overhead by size itself too, particularly with task queues

- 13 -

CS 418 S'04

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
 - e.g. searching for task to dequeue in task queue, building tree
- Stagger critical sections in time

- 14 -

CS 418 S'04

Reducing Inherent Communication

Communication is expensive!

Measure: *communication to computation ratio*

Focus here on inherent communication

- Determined by assignment of tasks to processes
- Later see that actual communication can be greater

Assign tasks that access same data to same process

Solving communication and load balance NP-hard in general case

But simple heuristic solutions work well in practice

- Applications have structure!

- 15 -

CS 418 S'04

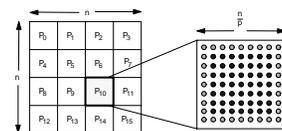
Domain Decomposition

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 3D)

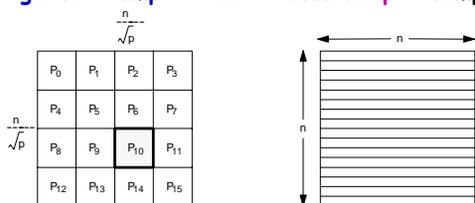
- Depends on n, p : decreases with n , increases with p

- 16 -

CS 418 S'04

Domain Decomposition (Continued)

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



Comm to comp: $\frac{4\sqrt{p}}{n}$ for **block**, $\frac{2\sqrt{p}}{n}$ for **strip**

- Retain block from here on

Application dependent: strip may be better in other cases

- E.g. particle flow in tunnel

- 17 -

CS 418 S'04

Finding a Domain Decomposition

Static, by inspection

- Must be **predictable**: grid example above, and Ocean

Static, but not by inspection

- **Input-dependent**, require analyzing input structure
- E.g. sparse matrix computations, data mining

Semi-static (periodic repartitioning)

- Characteristics **change but slowly**: e.g. Barnes-Hut

Static or semi-static, with dynamic task stealing

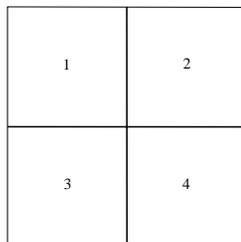
- Initial decomposition, but **highly unpredictable**: e.g. ray tracing

- 18 -

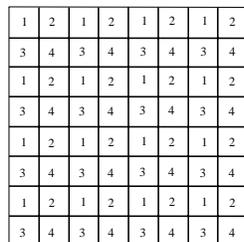
CS 418 S'04

Other Techniques

Scatter Decomposition, e.g. initial partition in Raytrace



Domain decomposition



Scatter decomposition

Preserve locality in task stealing

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

- 19 -

CS 418 S'04

Implications of Comm-to-Comp Ratio

If denominator is **execution time**, ratio gives **average BW needs**

If **operation count**, gives **extremes** in impact of latency and bandwidth

- **Latency**: assume no latency hiding
- **Bandwidth**: assume all latency hidden
- Reality is somewhere in between

Actual impact of comm. depends on structure & cost as well

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

- Need to keep communication balanced across processors as well

- 20 -

CS 418 S'04

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 communication and orchestration efficient

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

- 21 -

CS 418 S'04

Summary: Analyzing Parallel Algorithms

Requires characterization of multiprocessor and algorithm
Historical focus on algorithmic aspects: partitioning, mapping

PRAM model: data access and communication are free

- Only load balance (including serialization) and extra work matter

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

- Useful for early development, but **unrealistic for real performance**
- **Ignores communication and also the imbalances it causes**
- Can lead to poor choice of partitions as well as orchestration
- More recent models incorporate comm. costs; BSP, LogP, ...

- 22 -

CS 418 S'04

Limitations of Algorithm Analysis

Inherent communication in parallel algorithm is not all

- **artificial communication** caused by program implementation and architectural interactions can even dominate
- thus, amount of communication not dealt with adequately

Cost of communication determined not only by amount

- also how communication is structured
- and cost of communication in system

Both architecture-dependent, and addressed in orchestration step

To understand techniques, first look at system interactions

- 23 -

CS 418 S'04

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
- Programming model and comm. abstraction affect specific performance tradeoffs

Most of remaining performance issues focus on second aspect

- 24 -

CS 418 S'04

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

- 25 -

CS 418 S'04

Uniprocessor

Performance depends heavily on memory hierarchy

Time spent by a program

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

- Divide by instructions to get CPI equation

Data access time can be reduced by:

- **Optimizing machine:** bigger caches, lower latency...
- **Optimizing program:** temporal and spatial locality

- 26 -

CS 418 S'04

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**

- 27 -

CS 418 S'04

Artifactual Comm. in Extended Hierarchy

Accesses not satisfied in local portion 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.**
- **More on artifactual later; first consider replication-induced further**

- 28 -

CS 418 S'04

Communication and Replication

Comm. due to **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

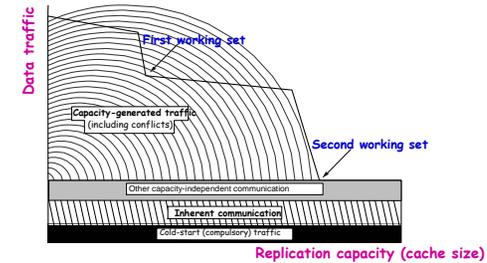
- **compulsory** or **cold** misses (no size effect)
- **capacity** misses (yes)
- **conflict** or **collision** misses (yes)
- **communication** or **coherence** misses (no)
- Each may be helped/hurt by large transfer granularity (spatial locality)

- 29 -

CS 418 S'04

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 **non-local** (communication)

- 30 -

CS 418 S'04

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

Let's examine techniques for both...

- 31 -

CS 418 S'04

Reducing Artifactual Communication

Message passing model

- **Communication and replication are both explicit**
- Even artifactual communication is in explicit messages

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

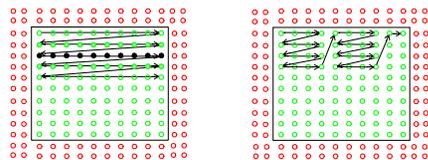
Use shared address space to illustrate issues

- 32 -

CS 418 S'04

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)

- 33 -

CS 418 S'04

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**

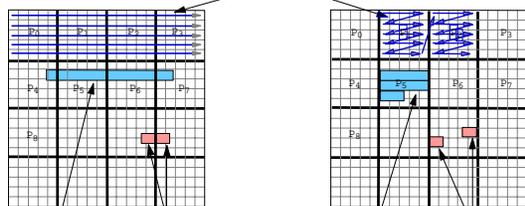
- 34 -

CS 418 S'04

Spatial Locality Example

- Repeated sweeps over 2-d grid, each time adding 1 to elements
- Natural 2-d versus higher-dimensional array representation

Contiguity in memory layout



Page straddles partition boundaries: difficult to distribute memory well

(a) Two-dimensional array

Cache block straddles partition boundary

(b) Four-dimensional array

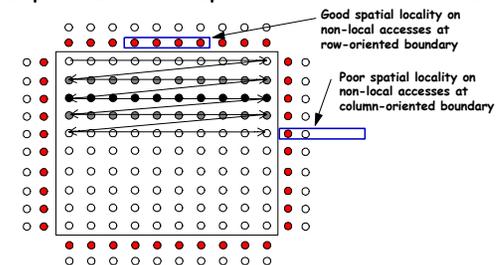
- 35 -

CS 418 S'04

Tradeoffs with Inherent Communication

Partitioning grid solver: **blocks** versus **rows**

- **Blocks** still have a **spatial locality problem** on remote data
- **Rows** can perform better despite worse inherent c-to-c ratio



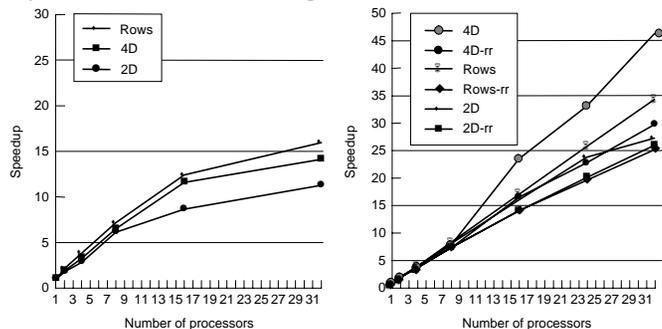
• Result depends on n and p

- 36 -

CS 418 S'04

Example Performance Impact

Equation solver on SGI Origin2000



- 37 -

CS 418 S'04

Structuring Communication

Given amount of communication, goal is to reduce cost

Cost of communication as seen by process:

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

- 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)
- That portion, ignoring overlap, is *latency* of a message

• **Goal: reduce terms in latency and increase overlap**

- 38 -

CS 418 S'04

Reducing Overhead

Can reduce # 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

- 39 -

CS 418 S'04

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

- 40 -

CS 418 S'04

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!*

- 41 -

CS 418 S'04

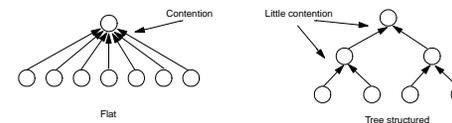
Types of Contention

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



• In general, reduce burstiness; may conflict with making messages larger

Module: all-to-all personalized comm. in matrix transpose

• **solution:** stagger access by different processors to same node temporally

- 42 -

CS 418 S'04

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

- 43 -

CS 418 S'04

Summary of Tradeoffs

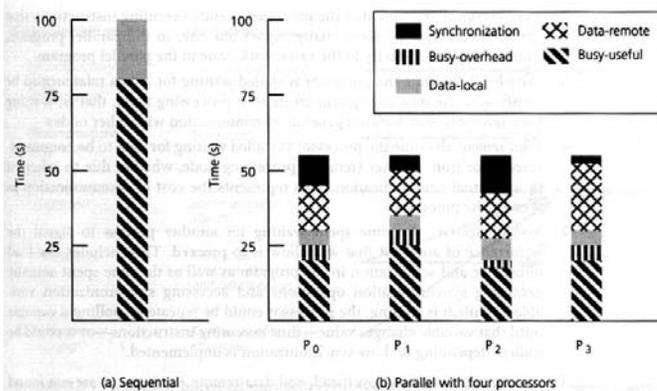
Different goals often have conflicting demands

- **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

- 44 -

CS 418 S'04

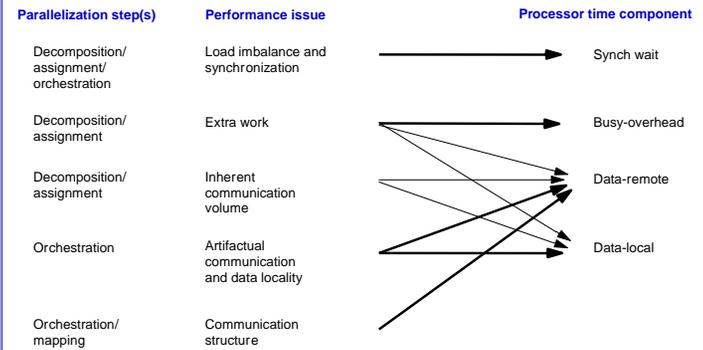
Processor-Centric Perspective



- 45 -

CS 418 S'04

Relationship between Perspectives



- 46 -

CS 418 S'04

Summary

$$Speedup_{prob}(p) = \frac{Busy(1) + Data(1)}{Busy_{useful}(p) + Data_{local}(p) + Synch(p) + Data_{remote}(p) + Busy_{overhead}(p)}$$

- Goal is to reduce denominator components
- Both programmer and system have role to play
- Architecture cannot do much about load imbalance or too much communication
- But it can:
 - reduce incentive for creating ill-behaved programs (efficient naming, communication and synchronization)
 - reduce artificial communication
 - provide efficient naming for flexible assignment
 - allow effective overlapping of communication

- 47 -

CS 418 S'04