Open Questions:

- What is a good uniprocessor model? What memory system performance can we expect from such a model?

- Can we develop a unified model for serial and parallel computers?

- What language constructs can we provide for optimizing compilers to be able to deliver good memory system performance?

- What is the relationship between memory system performance and processor performance (superscalar / VLIW etc.)?

- What are good models for algorithm design and performance evaluation for Distributed Shared Memory (DSM) computers?

Examples of Blocked Serial Algorithms

- ❏ Matrix Algorithms:

  Matrix Multiplication / Factorization (Blocked BLAS3 calls)

  Sparse Matrix Algorithms (Mat-vecs, factorization)

- ❏ Fast Fourier Transforms (Transform Based vs. Binary Exchange)
- ❏ Sorting (Sample Sort)

Implications for Serial Processors

- Conventional microprocessors have **deep cache hierarchies**.

- Issues of **spatial and temporal locality** are just as important if not more (e.g. a FFT routine can yield anywhere from 15 to 120 MFLOPS on a R6000/590 depending on how it is coded)

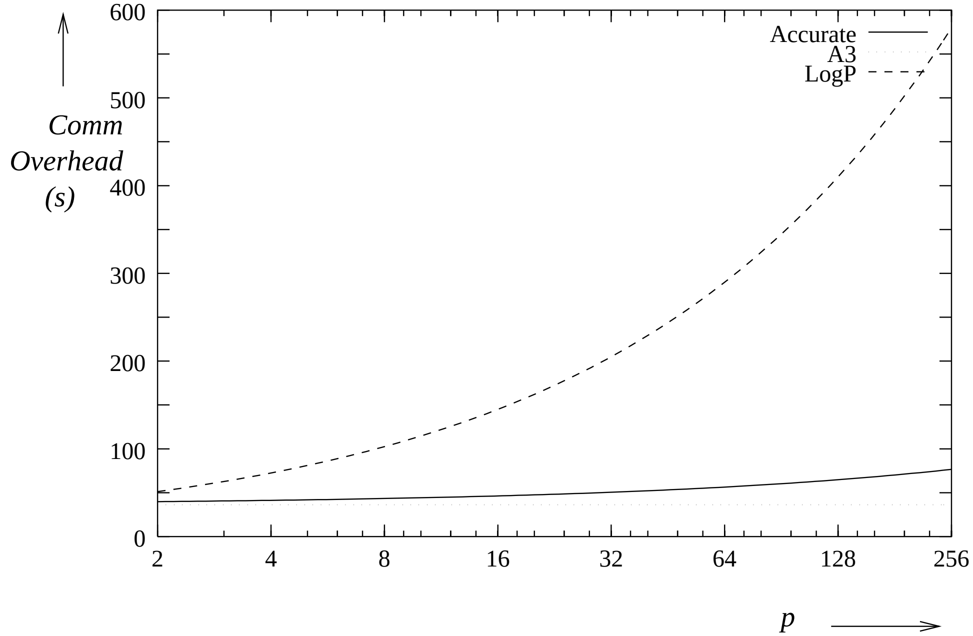
- A processor's view of the memory is "similar" to that in a multiprocessor with a fully connected network (modulo conflicts on data accesses).

- A good parallel algorithm must therefore also have good cache performance on uniprocessors.

Other Tested Algorithms:

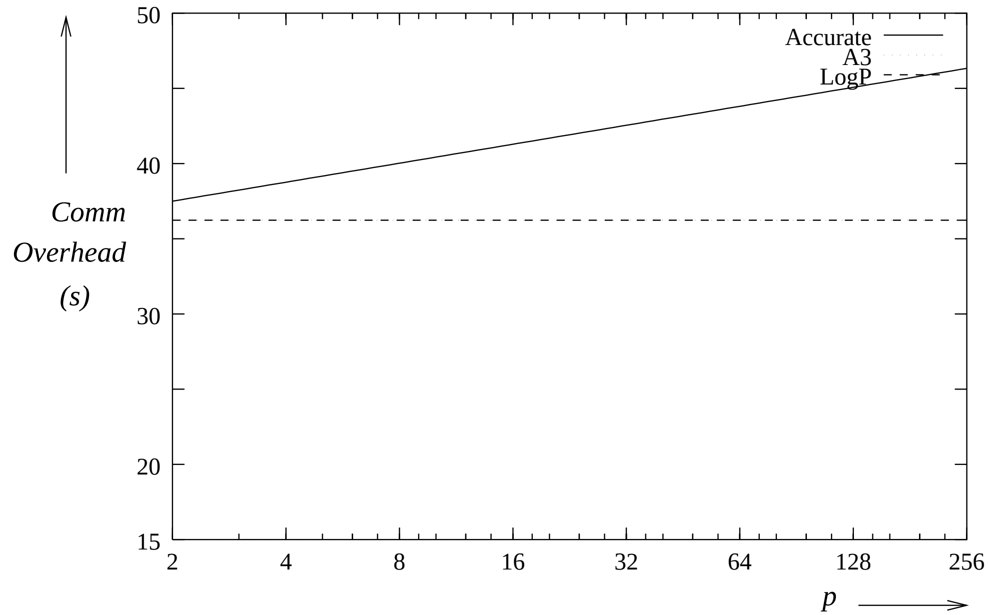
- ❏ Fast Fourier Transforms
- ❏ Sample Sort

Dense Matrix Factorization: Comparison of Models - Mesh.



Gaussian Elimination on a mesh (n = 8192)

Dense Matrix Factorization: Comparison of Models - Hypercube.



Gaussian Elimination on a Hypercube (n = 8192)

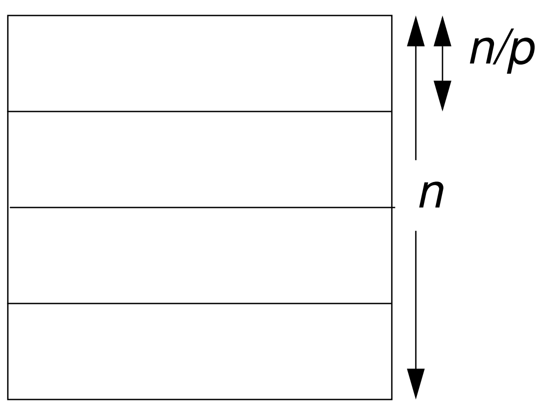
Gaussian Elimination (Dense Matrix Factorization): Performance

- Overhead results from one-to-all broadcasts.
- Algorithm works in n steps; in step *i*, we broadcast *n-i* data.
- Since $n > p$, we can use a **two step broadcast** (one-to-all personalized, followed by an all-to-all broadcast)
- Total communication cost = $2 \sum_{i=1}^{n} (n-i) t_w = t_w n^2$

	Hypercube	Mesh
LogP	$t_w n^2$	$t_w n^2 \sqrt{p}$
A3	$t_w n^2$	$t_w n^2$
Accurate	$2 t_s n \log P + t_w n^2$	$4 t_s n \sqrt{p} + t_w n^2$

Case Studies: Gaussian Elimination (Dense Matrix Factorization)

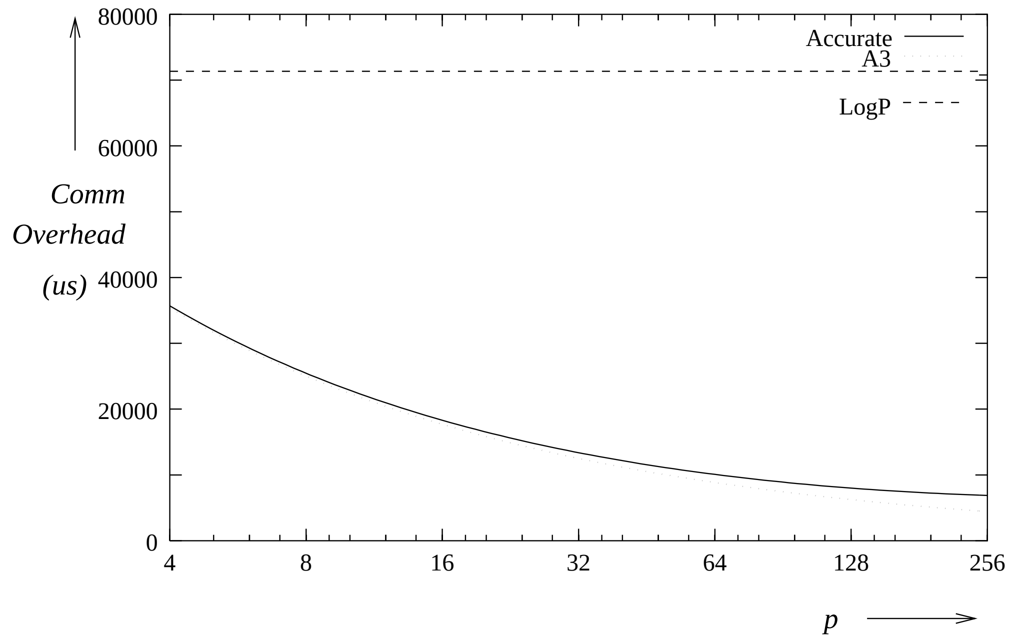
- ❏ Gaussian elimination is used for reducing a matrix into a triangular form.
- ❏ Pick a pivot; eliminate corresponding variable from all other equations.
- ❏ Parallel formulations are based on row-wise, blocked or cyclic partitioning.



Here we consider a row-wise partitioning:

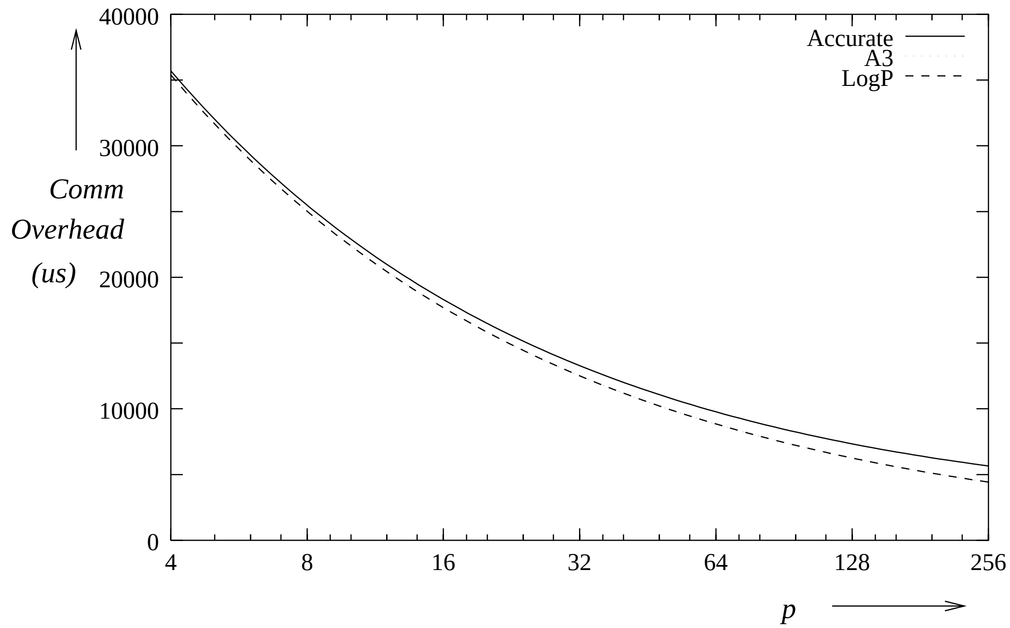
- ❏ Pick pivot.
- ❏ Broadcast pivot row ($n-i$ numbers in the i^{th} iteration) to all processors.
- ❏ Eliminate variable corresponding to pivot row.

Dense Matrix Multiplication: Comparison of Models - Mesh.



Matrix Multiplication on mesh (n = 256)

Dense Matrix Multiplication: Comparison of Models - Hypercube.



Matrix product on hypercube (n = 256)

Dense Matrix Multiplication - Analysis:

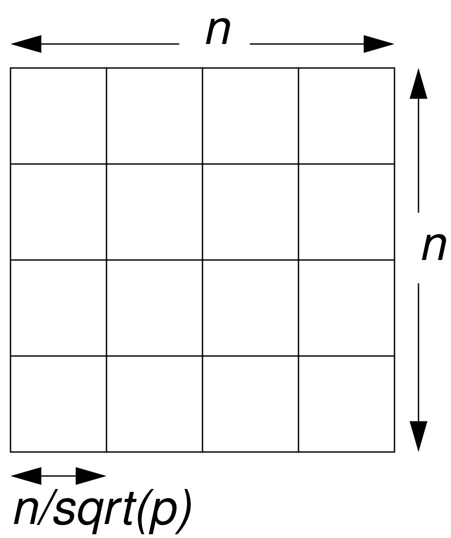
- Communication cost is due to two all-to-all broadcasts.
- The amount of data involved in each is $n^2/\sqrt{p}$.
- Both operations are bisection-insensitive.

The total communication cost is therefore given by $2*t_w*n^2/\sqrt{p}$ for the applicable class of architectures. The corresponding computation time is $t_c n^3/p$.

	Hypercube	Mesh
LogP	$2t_w n^2/\sqrt{p}$	$2t_w n^2$
A3	$2t_w n^2/\sqrt{p}$	$2t_w n^2/\sqrt{p}$
Accur.	$2(t_s \log p + 2t_w n^2/\sqrt{p})$	$2t_s\sqrt{p} + 2t_w n^2/\sqrt{p}$

Case Studies: Dense Matrix Multiplication

Matrix partitioning: 2-D checkerboard partitioning.



Each processor is assigned blocks of $n/\sqrt{p}$ x $n/\sqrt{p}$. The algorithm proceeds in following steps:

- All-to-all broadcast of rows of matrix A to all processors in the row.
- All-to-all broadcast of columns of matrix B to all processors in the column.
- Compute corresponding sub-blocks of matrix C.

A^3: Algorithm Design Methodology

- ❏ Determine candidate synchronous algorithm.
- ❏ Determine the aggregate communication operations required.
- ❏ Determine if they are bandwidth sensitive or insensitive.
- ❏ Evaluate communication cost of operation for given architecture.

Communication costs in A^3:

Given an architecture with bisection-width c, link bandwidth t_w, and p processors, the cost is as follows:

If the maximum amount of data entering or leaving any processor during a communication phase is m, then the cost of the operation is:

- ❑ For bisection-sensitive operations:

$$t = t_w m(p/c)$$

- ❑ For bisection-insensitive operations:

$$t = t_w m$$

A^3: A Two-Level Model for Estimating Communication Overheads

The space of aggregate communication operations can be partitioned into two: those that are **sensitive to bisection** width, and **those that are not**.

Bisection Insensitive	Bisection Sensitive
Broadcasts (one-to-all, all-to-all)	Random permutations
One-to-all Personalized	All-to-all Personalized
NEWS Shifts	Circular q-shift

A NEWS shift for a *k-d* mesh is defined as a communication with *d* nearest neighbors. In 2-D, it refers to the north, east, west, and south neighbors.

Timings for Various Aggregate Communication Operations

Operation	Linear Array	2-D Mesh	Hypercube
Bandwidth Insensitive Operations			
One-to-all Broadcast	$(t_s+t_w m) \log p + t_h(p-1)$	$(t_s+t_w m)\log p + 2t_h(\sqrt{p}-1)$	$(t_s+t_w m)\log p$
One-to-all Broadcast	$2(t_s p + t_w m)$	$2(2t_s\sqrt{p} + t_w m)$	$2(t_s \log p + t_w m)$
All-to-all Broadcast	$(t_s+t_w m)(p-1)$	$2t_s(\sqrt{p} -1)+t_w m(p-1)$	$t_s \log p + t_w m(p-1)$
One-to-all Person.	$(t_s+t_w m)(p-1)$	$2t_s(\sqrt{p} -1)+t_w m(p-1)$	$t_s \log p + t_w m(p-1)$
Bandwidth Sensitive Operations			
All-to-all Pers.	$(t_s+t_w mp/2)(p-1)$	$(2t_s+t_w mp)(\sqrt{p} -1)$	$(t_s+t_w m)(p-1) +(t_h/2)p \log p$
Circular q-shift	$(t_s+t_w m)\left\lfloor \frac{p}{2} \right\rfloor$	$(t_s+t_w m)(2\left\lfloor \frac{p}{2} \right\rfloor + 1)$	$t_s + t_w m + t_h \log p$

Role of Aggregate Communication Operations

Most synchronous algorithms are composed of a small set of aggregate communication operations:

- **Broadcast Operations**: data item needs to go to all processors.
  - **One-to-all**: one processor has data that must be communicated to all others.
  - **All-to-all**: all processors have a piece of data that must be made available to every other processor.
- **Personalized Operations**: data items have unique destinations.
  - **One-to-all**: One processor has p data items, each with a destination that must be communicated.
  - **All-to-all**: Each processor has p data items, each with a destination that must be communicated.
- **Circular *q*-shift**: Shifting each data item q hops defined by some linear ordering of the processors.

Characteristics of Algorithms

Parallel algorithms may be classified into one of two categories:

- **Asynchronous Algorithms**: Processors are allowed to communicate asynchronously; independent of each other.

- **Synchronous Algorithms**: In this class of algorithms, execution proceeds in interleaved steps of computation and communication. All processors (or groups thereof) participate in these steps.

The design philosophy of synchronous algorithms is consistent with the MPI philosophy of aggregate communication operations. (Synchronous algorithms should not be confused with SIMD algorithms. SIMD is a strict subset of synchronous algorithms).

(MPI: message passing interface; a library of message passing functions that is used for programming parallel computers)

Summary of Existing Models:

- Most models incorporate communication costs for bulk access and data reuse locality.

- None of the models capture data access patterns and therefore are limited in their coverage of architectures or algorithms.

- Most models are either too loose in their bounds; or yield lower / upper bounds based on algorithms and architectures.

Realistic models must therefore incorporate data access patterns in addition to abstract machine features.

Other Models:
Hierarchical PRAM (HPRAM), YPRAM.

- Models based on **recursive decomposition**.
- The computation is assigned to subgroups of processors.
- The cost of **communicating within a subgroup of *p* processors is given by some function *f(p)***. This cost function is more accurate than those of *LogP* and BSP since it induces some locality.
- The cost function **does not cover a large set of data access patterns** that are not based on recursive decomposition.
- The predicted communication cost can be **asymptotically incorrect by a factor equal to the diameter** of the interconnection network.

Drawbacks of the LogP Model

The logP model allows us to capture bulk and reuse localities. Its communication cost is based on permutations. A permutation of m words takes $O(m)$ time. This time is scaled by a bandwidth factor to account for other networks.

❑ The model is still based on ***O(p)*** **bisection width** networks.

❑ Provides a **convoluted cost model for capturing locality**.

❑ Provides **no hints with regards the assignment of tasks to processors**.

❑ Does **not capture structured communication patterns**.

❑ Yields **neither lower, nor upper bounds** on performance (depending on the data access pattern and architecture).

LogP Model of Computation

- *LogP* assumes an **ensemble of processors with local memory connected over a completely connected network** (whose aggregate b/w can be scaled).

- Communication in *LogP* occurs by **breaking messages into smaller packets**. The time for a packet to be communicated is modeled using

  - **Latency *L***: the time for a single small message to travel from source to destination.

  - **Overhead *o***: overhead paid for copying message into buffers, setting up network routers etc.

  - **Gap *g***: the time that must be allowed between two messages. This is dictated by the network bandwidth.

- LogP allows multiple packets to be pipelined.

Drawbacks of the BSP Model

❑ Requires ***O(p)*** **cross-section** bandwidth for the network to sustain the traffic (dense networks: crossbars, hypercubes).

❑ **Large constants** associated with data accesses.

❑ To mask startup cost t_s, atleast t_s requests must go to the same processor. Therefore, atleast $O(pt_s)$ threads must be assigned to each processor. This corresponds to a much **larger effective slack.**

❑ Due to randomization, access to multiple words within a processor is made to non-contiguous words. This results in **poor cache performance**. It is impossible to get good performance from such a system even for log p threads running on one processor.

Bulk Synchronous Parallel (BSP) Model

- Consider the following: p kids throwing stones randomly into p buckets. ***What is the expected value of the maximum number of stones in one bucket?*** Answer: *log p / (log log p)*.

- Now instead of one stone, each kid throws *log p* stones into p buckets. ***What is the expected value now?*** Answer: *3 x log p*.

- The kids can now be looked at as processors generating memory requests. The buckets can be viewed as the p processors' local memories.

- Assume that in one instruction, one processor generates one memory request. If the memory allocation is randomized, the request goes to a random processor. In such a case, *log p* requests can be asymptotically optimally serviced in *3 x log p* time if the network can sustain the traffic.

The BSP model is based on this premise.

Locality Based Models

- **LPRAM (Latency-PRAM)**: Incorporates data volume locality into PRAM cost model. Fetching m words from remote memory takes $t_w m$ cycles (local accesses are assumed to take one cycle).

- **BPRAM (Block-PRAM)**: Incorporates bulk access locality. Fetching m words from remote memory takes $l+m$ cycles (l is the latency).

(neither model addresses blocking of memory)

Completely Connected Network (CCN): Set of processors connected via a completely connected network. Remote access of m words costs time $t_s + t_w m$.

(the model does not account for structured data accesses).

Other models such as Hierarchical PRAM (HPRAM) and YPRAM account for a class of recursively expressible data accesses.

Drawbacks of PRAMS:

- ❏ The model is not realizable due to **complexity of the interconnect**.
- ❏ The model **does not account for communication costs**. Therefore, algorithms designed for PRAMs might have very different performance characteristics on real machines.
- ❏ PRAMs **do not have a concept of memory banking**.

Subsequent enhancements of PRAM models (**seclusive PRAM**) assume banked memories.

The **module parallel computer (MPC)** is based on the seclusive PRAM. It assumes that the memory is organized into modules, and access to the modules is exclusive.

Models for Parallel Computing: Parallel Random Access Machine (PRAM)

- Perhaps the first and best known abstract model for parallel computers.
- Synchronous shared-memory MIMD computer.
- The theoretical model assumes unlimited private and shared memory.
- The model assumes single cycle access to private and shared memory.
- Concurrent access to words in PRAMs are handled differently based on the model (EREW, CREW, CRCW).

Role of Data Locality

The cost of communicating a m words between processors l hops apart is $t_s + mt_w + lt_h$. For longer messages, this time can be adequately modeled as $t_s + mt_w$. This time implies two forms of data locality:

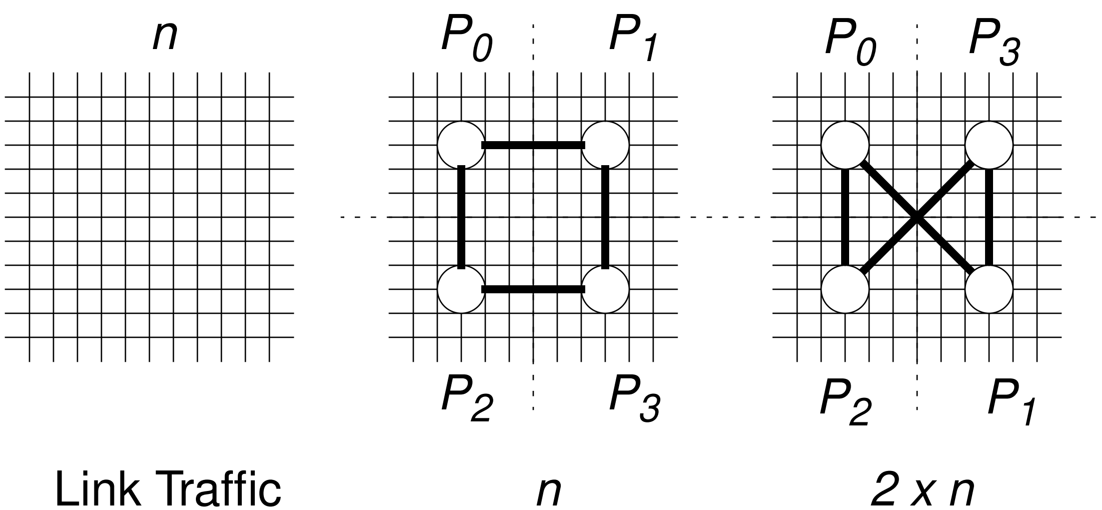
- **Bulk Access Locality** (spatial locality): While accessing remote data, it is desirable to access a sequence of words. In the context of uniprocessors, it is referred to as spatial locality.

- **Data Reuse Locality** (temporal locality): Minimize remote data accesses, reuse as much data as possible. In the context of uniprocessors, it is referred to as temporal locality.

- **Structured Data Access**: Certain data access patterns are more expensive than others. This is what results in architecture dependence of parallel algorithms.

Does cut-through routing not solve the problem?

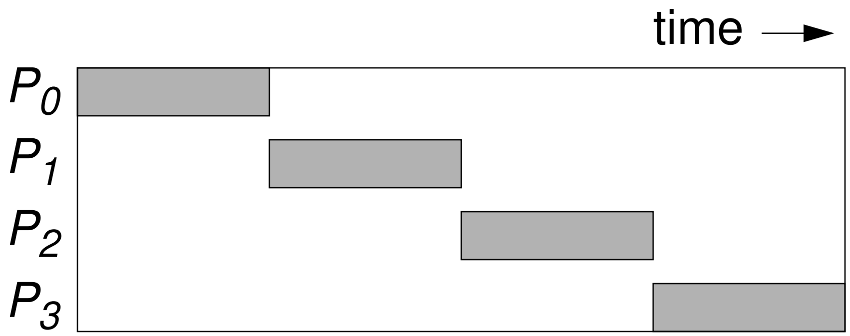
✐ By increasing the size of the message (m), the bandwidth term (mt_w) can be made to dominate the per-hop term (lt_h). So it does not matter how far a message must travel. Doesn't this solve the problem of data placement and communication schedules? What gives?



✐ By making n large, the per-hop term can be masked, but in the second case, the bandwidth term is twice; in general it is $p/2$ times optimal. ***It is all about bandwidth****!!!*

Routing Techniques

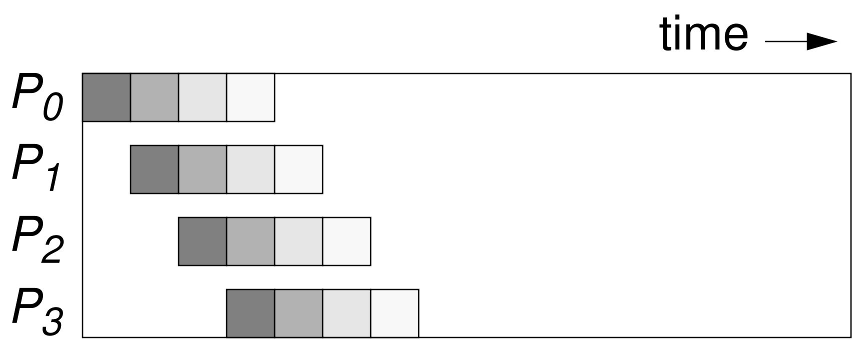
Store-and-Forward Routing



m: message length
l: number of hops

$t = t_s + lmt_w$

Cut-Through Routing



$t = t_s + lt_h + mt_w$

Connectivity Measures for Interconnection Networks

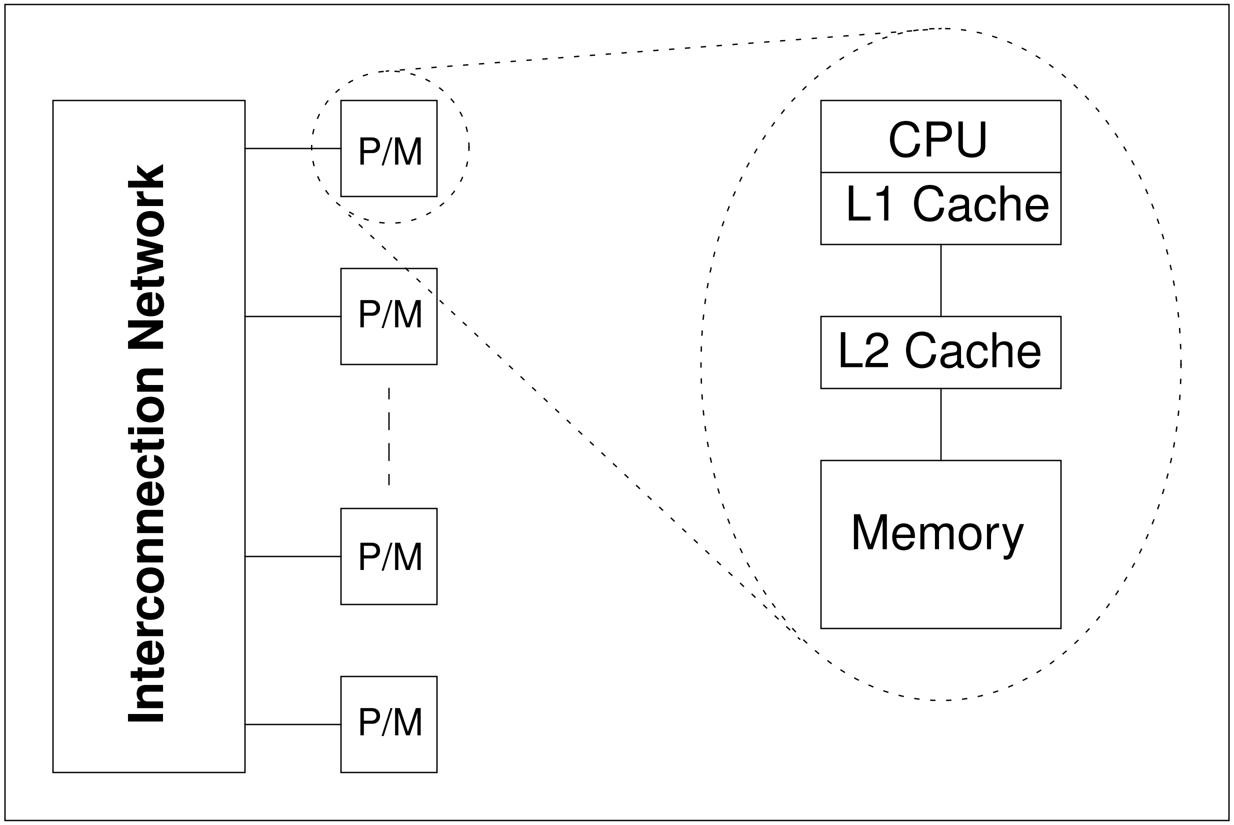
Diameter: Number of links between farthest pair of processors.

Arc Connectivity: Minimum number of links that must be removed to break the network into two parts.

Bisection width: Minimum number of links that must be removed to break the network into two **equal** parts.

Network	Bisection Width	Diameter
Hypercube	$p/2$	$\log p$
k-d cube	$2k^{d-1}$	$kd/2$
complete net.	$p^2/2$	1

Conventional Parallel Architectures



Popular Interconnects: *k-d* meshes (T3D, T3E, Paragon, Delta), Hypercubes (n-cube, cosmic-cube), fat trees (CM5).

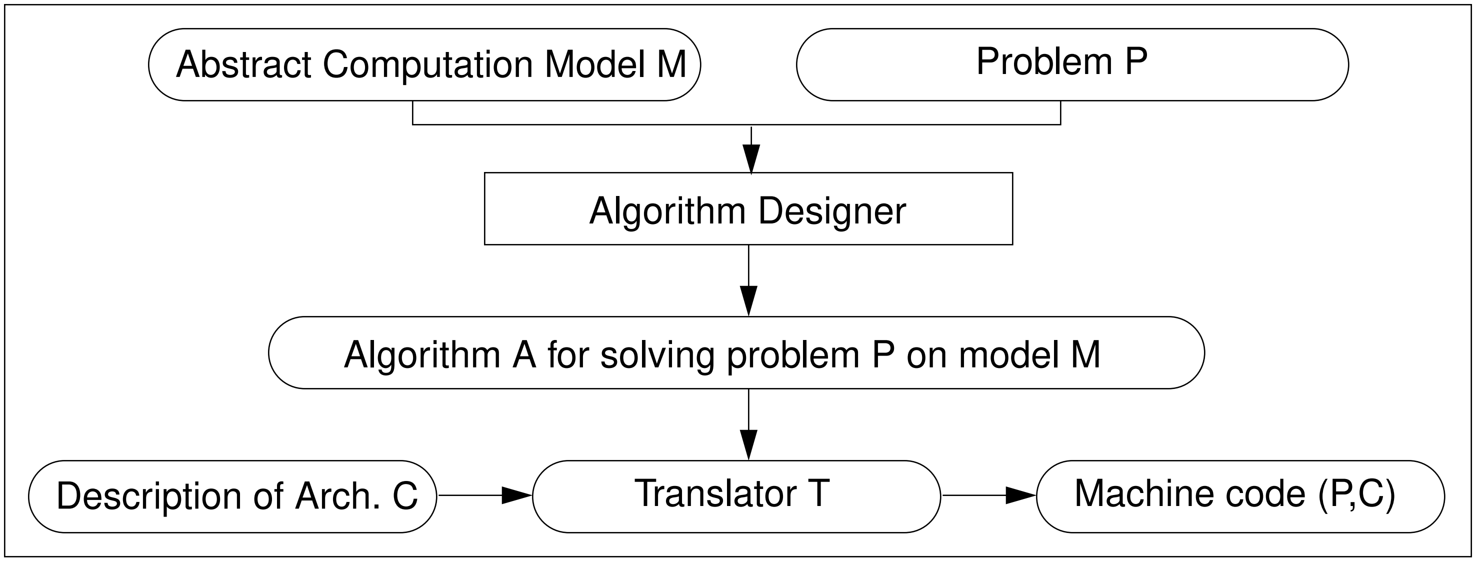
Why are abstract parallel computation models difficult to develop?

- Processors in a parallel computer must communicate to accomplish assigned task.
- The communication schedules are often sensitive functions of underlying connectivity between processors.
- An abstract model for a parallel computer must make some assumptions about the connectivity.
- A model assuming a low degree of connectivity will yield sub-optimal algorithms for higher degree networks such as hypercubes; and vice-versa.
- The connectivity (and therefore the communication overhead) is therefore very difficult to model in abstract models.

Need for Parallel Algorithm Design Models.

- ❑ **Portability:** A program designed for a computer must be portable across a “reasonably” wide range of computers and subsequent generations of the same computer.
- ❑ **Shorter design cycles** and **longer software life**.
- ❑ **Economics**.

What is an Algorithm Design Model?



- Abstract computation models should not be confused with programming models.

- von Neumann model provides the abstract computational model for serial computers.

A^3: A Simple and Asymptotically Accurate Model for Parallel Computation

Ananth Grama, Vipin Kumar, Sanjay Ranka, Vineet Singh

Department of Computer Science

Purdue University
W. Lafayette, IN 47906
ayg@cs.purdue.edu

University of Minnesota
Minneapolis, MN 55455
kumar@cs.umn.edu

University of Florida
Gainesville, FL 32611
ranka@cise.ufl.edu

IBM TJ Watson Research
Hawthorne, NY 10532
vsingh@watson.ibm.com

(papers relating to this talk are available at *http://www.cs.umn.edu/~kumar*)