A^3: A SIMPLE AND AYSMPTOTICALLY ACCURATE MODEL FOR COARSE-GRAINED MACHINES

Sanjay Ranka

University of Florida
http://www.cis.ufl.edu/ ranka

March 1996

Outline of the talk

1. Important Features of coarse grained machines.
2. Classification of known models based on these features.
3. A^3 (Approximate Model for Analysis of Aggregate Communication Operations) and Computation Model).
4. Performance of the model on key applications.
5. Limitations of the Model.

Framework for an architecture-independent computation model

figure=model1.eps,height=5in

Figure 1:

1. Evolutionary trends of parallel computers have converged to a general architecture consisting of a set of processing elements connected through an interconnection network.
2. Memory is normally organized into blocks. Access by a processor to a memory block precludes concurrent access to the same block by other processors (*seclusive memory* model)
3. Memory is physically distributed across the processors. Interaction between processors is either through message passing or through a shared address space.
4. Processing nodes consist of a processor with on chip cache, a second level cache and a large memory (typically ranging from 16 - 128 megabytes). The clock speeds range up to a several hundred MHz.
5. Memory hierarchy plays a critical role at such clock rates. Locality of data reference even within a processor is necessary to achieve good per-processor performance.

6. The interconnection network of conventional parallel computers are k-d meshes (a hypercube is is $2{:}\log p$ mesh), multistage networks,crossbars etc.

The Communication Model

The cost of communicating a message arise from various sources. These delays can be categorized into three classes:

- Message-size dependent delay: This delay is a consequence of copying the message into buffers and due to the finite capacity of the interconnection network. We capture these delays in a constant t_w, henceforth referred to as the *per-word transfer time*.
- Path-length dependent delay: This delay is a consequence of the overhead at intermediate network interfaces and the signal propagation time on each link. This delay is represented by constant t_h, also referred to as the *per-hop time*.
- Start-up latency: This delay is independent of message length and number of hops. It is incurred due to handshaking between processors, buffer management and routing. This is represented by t_s, also referred to as the message startup time.

A message of size m traversing d hops of a cut-through (CT) routed network incurs a communication delay given by

$$T_{comm}^{CT} = t_s + t_h d + t_w m$$

The corresponding cost for a store-and-forward (SF) network is

$$T_{comm}^{SF} = t_s + (t_h + t_w m)d$$

The communication model (contd.)

The startup time t_s is often large, and can be several hundred machine cycles or more. The per-word transfer time t_w is determined by the link bandwidth. Lower degree networks typically have higher link bandwidth (and consequently lower t_w). This is because the individual channels can be made fatter for the same overall cost.

The per-word transfer time t_w is often higher than t_c, the time to do a unit computation on data available in the cache.

The per-hop component $t_h d$ can often be subsumed into the startup time t_s without significant loss of accuracy. This is because the diameter of the network, which is the upper bound on distance d is relatively small for most practical sized machines, and t_h also tends to be small.

For large traffic network gets songested and a a good measure of the capacity of the network to carry multiple messages is its cross-section bandwidth (also referred to as the bisection width).

This is defined as the minimum number of independent paths available between two equal partitions of the machine. For a hypercube, the bisection width is $p/2$, for a wraparound mesh $2\sqrt{p}$, and for a shared bus 1.

Shared memory Systems

On shared address space systems, reading m consecutive words from a remote processor's memory also takes time $t_s + t_h d + t_w m$. Often, $t_h d$ is constant for all memory modules, and can be subsumed into t_s. The parameter m is usually the cache line size. The value of t_s tends to be much smaller than its counterpart in message passing systems. As before, the communication time corresponds to lightly loaded networks. If there are more memory transfers in progress than the cross-section bandwidth of the network, the time for these operations will be higher.
The communication overhead in both shared address space and message passing computers can often be reduced. In many parallel computers, it is possible for the communication to proceed after a certain startup cost has been paid at the source and destination processors. In this case, the transit time for messages, given by $t_h d + t_w m$, can be effectively masked by overlapping it with computation at source and destination processors.

Bulk Access Locality

In conventional parallel computers, message startup latency t_s tends to be much larger than the per-word time t_w. This makes it more attractive to access data in bulk. This property of the parallel algorithm is called *bulk access locality*.
Parallel computation models such as PRAMs, LPRAM, HPRAM , BSP, do not reward bulk access, and thus encourage design of parallel algorithms that do not incorporate granularity.
Models such as BPRAM, YPRAM, Postal and LogP model reward bulk access locality.
Bulk access locality plays an important role even for serial computers with memory hierarchy. Most processors use caches with line size greater

than one word.

Data Volume Locality

Access to words in non-local memory in a parallel computer typically costs more than local memory because data needs to travel over a communication network.

Data volume locality is rewarded by the per-word transfer time (t_w) and by the overall bandwidth of the network. Data volume locality is important for machines in which effective per-word transfer time is higher than local memory access time.

Note that the effective per-word transfer time can be higher than t_w due to congestion in the network (which can happen due to limited cross-section bandwidth of the network and/or due to the nature of the underlying communication pattern).

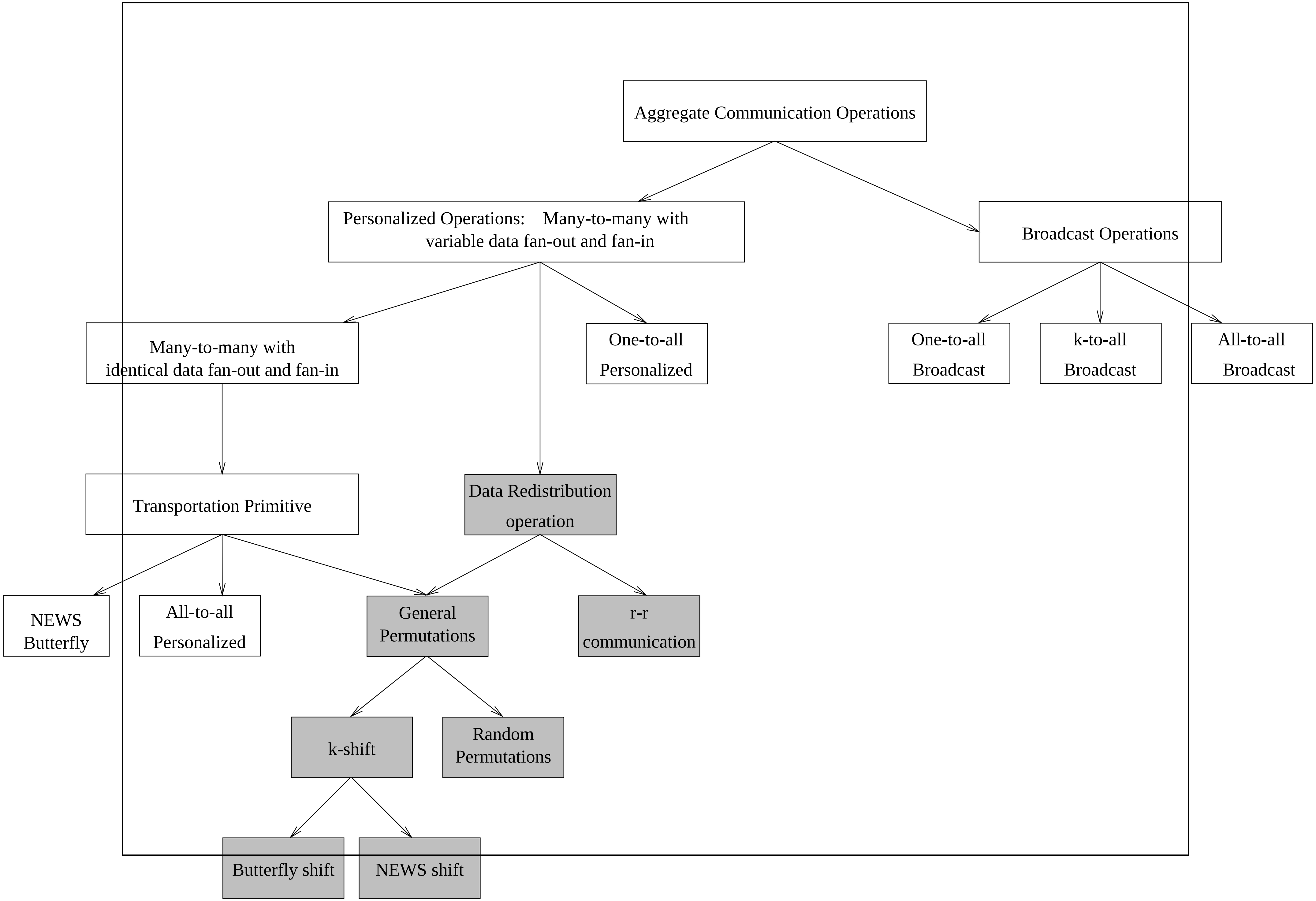
Parallel computation models can be classified into those that reward data volume locality (LPRAM, YPRAM, HPRAM, LogP model), and those that do not (PRAMs, BPRAM, BSP).

Models in the latter class tolerate high volumes of communication thus resulting in poor performance on real machines.

Structured Communication Operations

Different communication patterns congest the network to varying degrees. Data communication patterns in most applications have a definite structure. It is possible to exploit these structures by imposing these intelligently on to the underlying machine topology. It is therefore desirable that models reward parallel algorithms that induce structured communication patterns that minimize congestion.

Most synchronous algorithms interact after phases of computation. All processors (or subsets thereof) participate in the communication step. Since the communication involves more than a single pair of processors (aggregates or groups), such communication operations are also referred to as aggregate or collective communication operations. Nearly all parallel programs written in data-parallel languages such as HPF fall in this category. Programs based on MPI that use only collective communication operations also fall in this category.


Aggregate Communication Operations
Personalized Operations: Many-to-many with variable data fan-out and fan-in
Broadcast Operations
Many-to-many with identical data fan-out and fan-in
One-to-all Personalized
One-to-all Broadcast
k-to-all Broadcast
All-to-all Broadcast
Transportation Primitive
Data Redistribution operation
NEWS Butterfly
All-to-all Personalized
General Permutations
r-r communication
k-shift
Random Permutations
Butterfly shift
NEWS shift

Existing Models for Parallel Computation

LogP Model

The LogP model characterizes a parallel computer using four parameters: the communication delay (L), the communication overhead (o), the communication bandwidth (g), and the number of processors (p). The communication delay L is the delay (latency) experienced by a fixed-size message consisting of a small number of words from the source to the destination. Sending a message from one processor to another involves overheads at the source and destination processors. These overheads include copying buffers, and preparing headers. When the processor is performing these functions, it cannot do anything else. This overhead is modeled by the communication overhead o. The rate at which individual packets can be injected into the network is limited by the bandwidth available to each processor. If a processor has to wait a minimum of g units of time between two consecutive message sends, then g is referred to as the gap. Clearly, the gap is determined by the per-processor communication bandwidth of the network. Finally, the machine is assumed to have p processors. A single unit of time is defined as the time is takes to perform a single local computation.

Messages can be injected into the network at intervals of g. Of this time, o is spent by the processor preparing the next message. The remaining time $g - o$ (assuming $g > o$) can potentially be used to perform useful computation although it is more complex to utilize computation time with such small granularity.

The major attractiveness of LogP compared to other previous models such as BSP is that it rewards both bulk access locality (using o) and data volume locality using (L and g). In fact, the LogP model is quite similar to the general communication model discussed in Section 2.1 with the exception that effective t_w in the LogP model is scaled to compensate for the cross-section bandwidth c by

$$t_w^{effective} = t_w \times \frac{p}{c}.$$

The major drawback of LogP is that it is still unable to exploit structure in underlying communication patterns. The model assumes a random mapping of computation onto processors. It is therefore incapable of distinguishing benign data redistributions from those that congest the network. It assumes that all redistributions can be accomplished in time proportional to message size per processor. Let us study the implications of this assumption closely. Assume that each processor has a data item of size m that it needs to communicate to a unique, randomly chosen destination (random permutation). LogP assumes that this permutation can be performed in $\Theta(m)$ time.

A^3: A Parallel Computation Model for Architecture Independent Programming and Analysis

Assumptions:
We assume in our model that the programming paradigm is synchronous. Groups of processors go through phases of local computation followed by communication for non-local data. At any given snapshot, all processors within a group are in the same phase (*i.e.*, computing or communicating). We have identified the following set of aggregate communication operations that appear most frequently: many-to-many broadcast (and their dual - reduce, and variations - scan), many-to-many personalized, k-shifts, 2-D grid based communication - NEWS (and its generalization to $\log p$ dimensions - butterfly).

Accurate Costs

Table **??** tabulates the various aggregate communication operations along with their cost on k:d cubes. Here, t_s is the startup latency, t_h is the per-hop time and t_w is the per-word transfer time.

Operation	Linear Array	2-D Mesh (wraparound, square)	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 with $\Theta(p)$ slack	$2(t_s p + t_w m)$	$2(2t_s\sqrt{p} + t_w m)$	$2(t_s \log p + t_w m)$
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 personalized	$(t_s + t_w m)(p-1)$	$2t_s(\sqrt{p}-1) + t_w m(p-1)$	$t_s \log p + t_w m(p-1)$
Bandwidth Sensitive Operations			
All-to-all personalized	$(t_s + t_w mp/2)(p-1)$	$(2t_s + t_w mp)(\sqrt{p}-1)$	$(t_s + t_w m)(p-1)$ $+(t_h/2)p\log p$
Circular q-shift	$(t_s + t_w m)\lfloor p/2 \rfloor$	$(t_s + t_w m)(2\lfloor \sqrt{p}/2 \rfloor + 1)$	$t_s + t_w m$ $+t_h \log p$

Table 1: Various communication operations and their cost on a k:d cube.

A^3: A Simpler Model

The space of data-redistribution operations can be partitioned into two parts: operations sensitive to the cross-section bandwidth (in bold at the bottom of the table), and those insensitive to the cross-section bandwidth. Sensitivity to cross-section bandwidth is determined by the volume of data and the redistribution pattern.

For example, generalized data-redistribution operations such as random permutations and some structured redistributions such as k-shifts are cross-section bandwidth sensitive.

Let the maximum amount of data entering or leaving any processor during an aggregate communication operation be m. If m grows at a particular rate with respect to p, it is possible to derive concise estimates of the asymptotic time requirements of various communication operations based solely on m and the cross-section bandwidth.

This rate of growth of m is referred to as the *slack* in communication.

The amount of slack required is different for different algorithms and architectures. A slack of $\Theta(p)$ is adequate for all cases illustrated in Table **??**. In many of these cases, a lower slack is adequate. But in some other cases, a higher slack may be required.

In the presence of adequate slack, the communication times can be approximated as follows:

Aggregate Communication Operations Sensitive to Cross-Section Bandwidth: Aggregate communication operations such as many-to-many personalized and k-shifts are cross-section bandwidth sensitive operations. Let m be the maximum amount of data leaving or entering any processor and c be the cross-section bandwidth of the network. For such operations, each link in the network has an effective bandwidth of $c/(t_w p)$. The time taken by any aggregate communication operation in this class is given by

$$T_{comm} = t_w m(p/c) \tag{1}$$

Specifically, for a k:d cube, the cross-section bandwidth is given by $c = p^{\frac{d-1}{d}}$. Substituting the value of c in Equation **??**, we get

$$T_{comm} = t_w m p^{\frac{1}{d}} \tag{2}$$

Equation **??** gives the time for a aggregate communication operation in this class for a k:d cube.

Aggregate Communication Operations Insensitive to Cross-Section Bandwidth: Aggregate communication operations such as many-to-many broadcast and NEWS belong to this class. NEWS is defined as 2-D grid structured communication. The communication operation is bandwidth insensitive on the family of k:d meshes for $d \geq 2$. The cost of these operations is largely independent of the number of processors. If m is the maximum amount of data entering or leaving any processor, the time for these operations is given by:

$$T_{comm} = t_w m \tag{3}$$

A parallel algorithm can be analyzed for a k:d cube by using these two communication costs applied to appropriate aggregate communication operations.

In both these cases, the A^3 model ignores the impact of the startup latency (t_s) and the per-hop time (t_h). This restricts the applicability of the model to cases where there is adequate slack. In other cases, it yields lower bounds on the communication time for the operation.

Analysis of Algorithms

In this section, we present results of analysis of a variety of parallel algorithms. We compare the A^3 and LogP models to the accurate model. We show that in each of the cases the A^3 model yields accurate asymptotic estimates of communication overhead. The LogP model fails to do so for the mesh for all these algorithms. Furthermore, although it may seem that the LogP model gives accurate runtimes for the hypercube, it gives no hints about placement of tasks on processors. Randomly placing subtasks on processors may cause contention over the network, degradation of performance, and worse runtimes. In our model it is assumed that the library call to aggregate communication operations ensures proper placement of subtasks on processors. Note that this library needs to be constructed just once for each architecture. From the analysis, it also becomes clear that our model makes it simpler to analyze algorithms. Table **??** summarizes the communication overheads predicted

Algorithm	Hypercube			Mesh		
	A^3	Accurate	LogP	A^3	Accurate	LogP
Matrix Product	$2t_w \frac{n^2}{\sqrt{p}}$	$2(t_s \log p + t_w \frac{n^2}{\sqrt{p}})$	$2(t_w \frac{n^2}{\sqrt{p}})$	$2t_w \frac{n^2}{\sqrt{p}}$	$2t_s\sqrt{p} + 2t_w \frac{n^2}{\sqrt{p}}$	$2(t_w n^2)$
Sample sort	$t_w p \log^2 p$ $+2t_w n/p$	$\log^2 p(t_s + t_w p)$ $+2t_s p + 4t_w n/p$	$\log^2 p(t_w p)$ $+2t_w n/p$	$2t_w p^{1.5}$ $+t_w 2n/\sqrt{p}$	$2(t_s\sqrt{p} + t_w p^{1.5})$ $+4t_s\sqrt{p} + 4t_w n/\sqrt{p}$	$\log^2 p(t_w p^{1.5})$ $+2t_w n/\sqrt{p}$
Gauss. Elim.	$t_w n^2/2$	$2t_s n \log p + t_w n^2$	$t_w n^2$	$t_w n^2/2$	$4t_s n\sqrt{p} + t_w n^2$	$t_w \sqrt{p} n^2$

Table 2: A comparison of communication overheads predicted by the A^3, accurate and LogP models.

by each model for matrix-matrix product, sample sort and Gaussian elimination.

For the LogP model, we assume that the overhead o is small and that the gap $g > o$. This allows us overlap of computation and communication where feasible. Furthermore for asymptotic analysis, as message sizes increase, it is possible to drop the term L of the LogP model.

An example

The value of t_s is 77 microseconds and t_w is 0.54 microseconds/word.

Conclusions

Concluding Remarks

The A^3 model is valid for synchronous algorithms based on aggregate communication operations. This methodology of parallel program design offers several advantages:

- Algorithms are portable across a variety of platforms with a minimal loss of efficiency.
- The parallel model hides the complexity of writing efficient communication routines for different machines from the user.
- Programming parallel machines using this paradigm is much simpler.
- The model brings the algorithm design process closer to the functionality of MPI and HPF.

- The model provides a means of determining good data partitioning and placement strategies. Data placement and the communication operations they induce together determine the figure of merit for an algorithm. The key to designing efficient algorithms is to identify favorable combinations of the two.