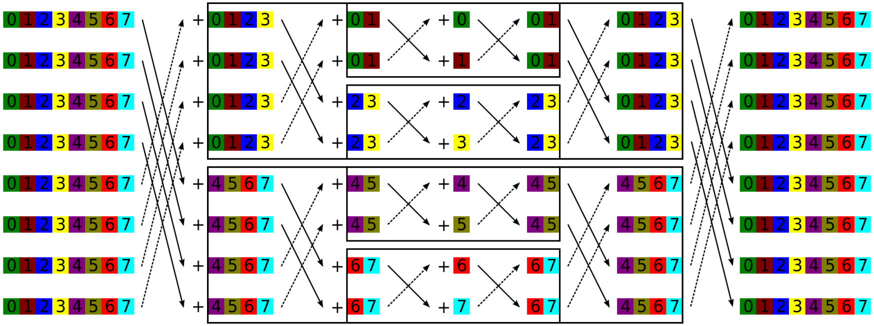
CS 598: Communication Cost Analysis of Algorithms
Lecture 2: optimal packet-based broadcast; communication cost models

Edgar Solomonik

University of Illinois at Urbana-Champaign

August 24, 2016

Recursive allreduce



Recursive allreduce

```
Algorithm 1 $[v] \leftarrow \text{Allreduce}(A, \Pi)$
Require: $\Pi$ has $P$ processors, $A$ is a $s \times P$ matrix, $\Pi(j)$ owns $A(1:s, j)$
  Define ranges $L = 1 : \frac{s}{2}$ and $R = \frac{s}{2} + 1 : s$
  Let $x = j + \frac{P}{2}$ and $y = j - \frac{P}{2}$
  if $j \leq P/2$ then
    $\Pi(j)$ sends $A(R, j)$ to $\Pi(x)$ and receives $A(L, x)$ from $\Pi(x)$
    $[v(L)] \leftarrow \text{Allreduce}\Big(A(L, j) + A(L, x), \Pi(1 : \frac{P}{2})\Big)$
    $\Pi(j)$ sends $v(L)$ to $\Pi(x)$ and receives $v(R)$ from $\Pi(x)$
  else
    $\Pi(j)$ sends $A(L, j)$ to $\Pi(y)$ and receives $A(R, y)$ from $\Pi(y)$
    $[v(R)] \leftarrow \text{Allreduce}\Big(A(R, j) + A(R, y), \Pi(\frac{P}{2} + 1 : P)\Big)$
    $\Pi(j)$ sends $v(R)$ to $\Pi(y)$ and receives $v(L)$ from $\Pi(y)$
  end if
Ensure: Every processor owns $v = [v(L); v(R)]$, where $v(i) = \sum_{j=1}^{P} A(i, j)$
```

Allreduce cost derivation

In the recursive/butterfly allreduce, every processor

1. sends and receives one message of size $s/2$
2. recurses with half the processors, with A of size $s/2 \times P/2$
3. sends and receives one message of size $s/2$

$$\begin{aligned} T^{\alpha-\beta}_{\text{allred}}(s,P) &= T^{\alpha-\beta}_{\text{allred}}(s/2,P/2) + 2\cdot(\alpha + s/2\cdot\beta) \\ &= 2\sum_{i=1}^{h} \alpha + \frac{s}{2^i}\cdot\beta \\ &\leq 2(h\cdot\alpha + s\cdot\beta) \end{aligned}$$

where $h = \log_2(P)$

Optimal broadcast

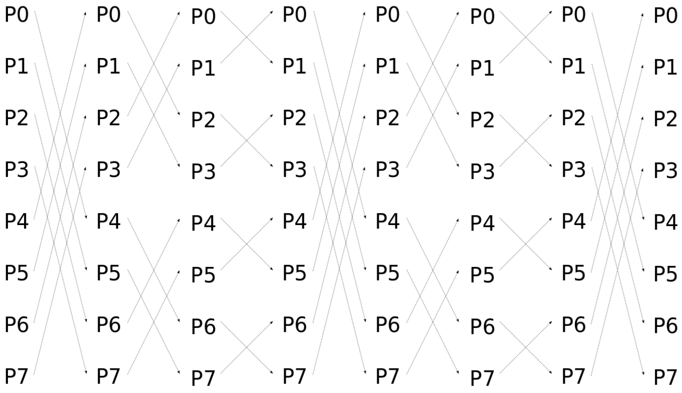
We will now cover a protocol for broadcast that achieves the cost,

$$T^{\alpha-\beta}_{\mathrm{bcast-TR}} = (\sqrt{h \cdot \alpha} + \sqrt{s \cdot \beta})^2 \leq 2(h \cdot \alpha + s \cdot \beta).$$

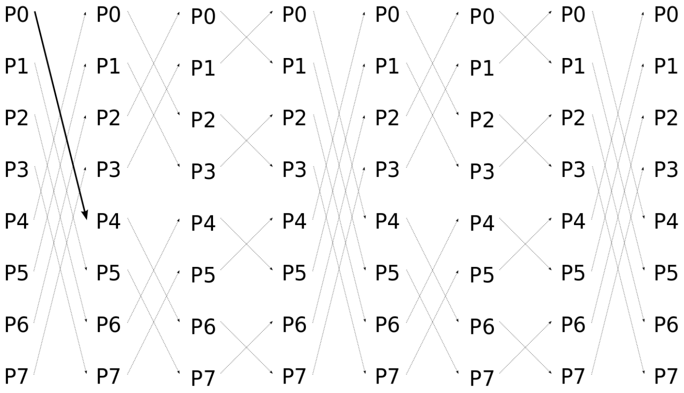
The protocol presented is based on that of Träff and Ripke (2008), but is restricted to power of two processor counts and presented differently.

The above cost is optimal, under the assumption that all messages sent are of the same size. Note that the butterfly collectives we covered do not adhere to this rule, but nevertheless do not have a lower cost.

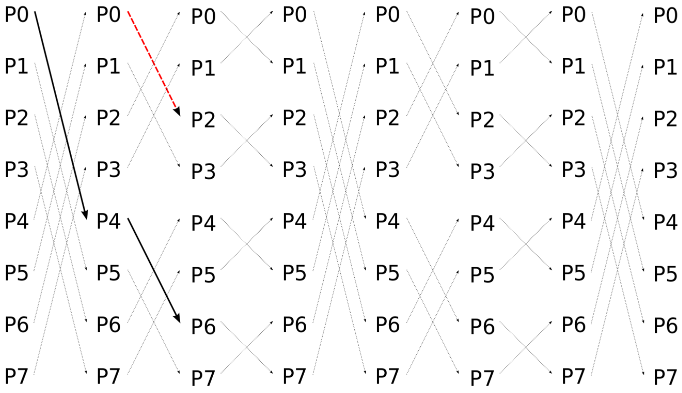
Binomial broadcast trees in a butterfly



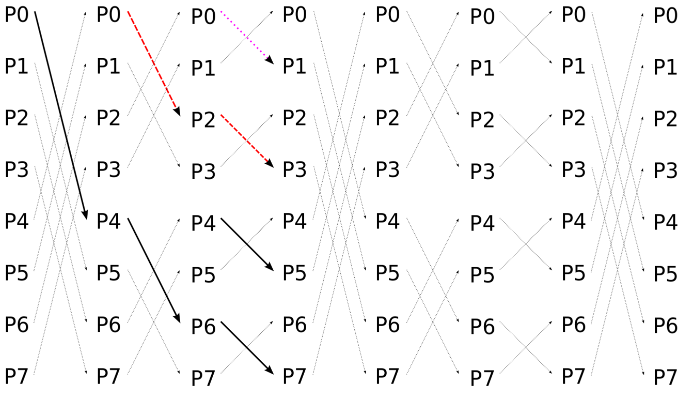
Binomial broadcast trees in a butterfly



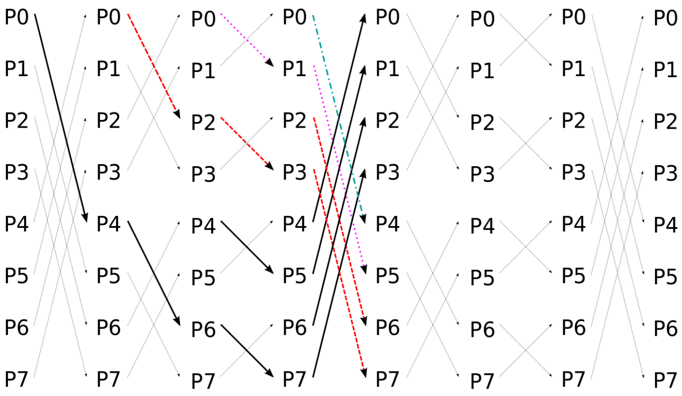
Binomial broadcast trees in a butterfly



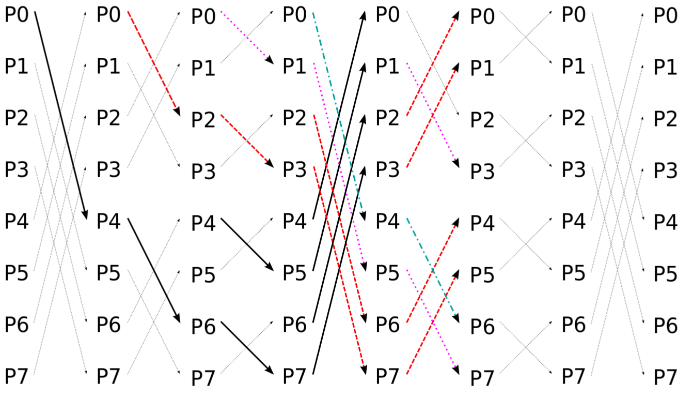
Binomial broadcast trees in a butterfly



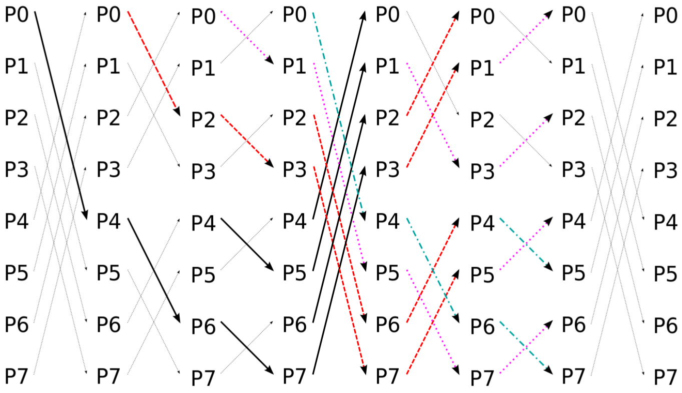
Binomial broadcast trees in a butterfly



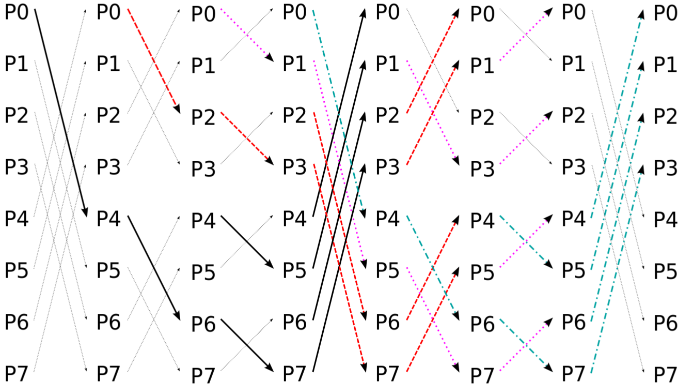
Binomial broadcast trees in a butterfly



Binomial broadcast trees in a butterfly



Binomial broadcast trees in a butterfly



Optimal broadcast construction

We would like to construct a set of binomial tree broadcasts that can execute simultaneously

- let the set of processors exclding broadcast root be $\Pi = \{1, \dots, p-1\}$, $|\Pi| = 2^h - 1$
- h binomial trees of height h with all nodes except root
- $S(i,j) \subset \Pi$ is the set of processors that send a message in the ith level of the jth tree, $i,j \in [0, h-1]$
- $|S(i,j)| = 2^i$
- $S(i,j) = \Pi \setminus \bigcup_{k=1}^{h-1} S(i-k \bmod h,\ j+k \bmod h)$
- for any k, the sets $S(i,j)$ and $S(i-k \bmod h,\ j+k \bmod h)$ are disjoint
- so the messages in these tree levels can be sent simultaneously
- given this construction, we have $s/k + h$ messages of size k to broadcast message of size s, with total cost

$$T(k) = (s/k + h) \cdot (\alpha + k \cdot \beta), \quad \min_k(T(k)) = (\sqrt{h \cdot \alpha} + \sqrt{s \cdot \beta})^2$$

Optimal broadcast

A wrapped butterfly network yields the desired construction

- intuition: the butterfly network has no cycles of length less than h
- each level of the butterfly connects nodes one bit flip away
- the broadcast root sends to node $2^j \bmod p$ at step j
- the root of the jth binomial tree is node 2^j
- the jth bit is not flipped again until that tree completes
- at the ith level the senders of the jth binomial tree are

$$S(i,j) = \left\{ 2^j + \sum_{r \in R} 2^{j+r \bmod h} \;:\; R \subseteq \{1, \ldots, i\} \right\}$$

- $S(i,j)$ and $S(i{-}k \bmod h,\; j{+}k \bmod h)$ are disjoint for any k, so long as $S(i,j)$ and $S(i{-}k,\; j{+}k \bmod h)$ are disjoint for any $k \le i$
- $S(i,j)$ and $S(i{-}k, j{+}k \bmod h)$ for $k \in [1, h-1]$ are disjoint as

$$S(i{-}k, j{+}k \bmod h) = \left\{ 2^{j+k \bmod h} + \sum_{r \in R} 2^{j+k+r \bmod h} : R \subseteq \{1, \ldots, i-k\} \right\}$$

won't include elements with summand 2^j since $k < h$ & $k + r \le i < h$

Not a binomial tree or butterfly

A short break!

Homeworks

First homework assignment:

- posted on Piazza (join "CS 598 ES")
- please send in pdf form to solomon2@illinois.edu, with email title including "CS 598"
- should be completed using latex
- if you get stuck on a problem for more than 2-3 hours, post your thoughts on Piazza or email me
- due Wednesday, Aug 31 by 9:30 am, late policy posted on website

The LogP model

Limitations of the α–β messaging model:

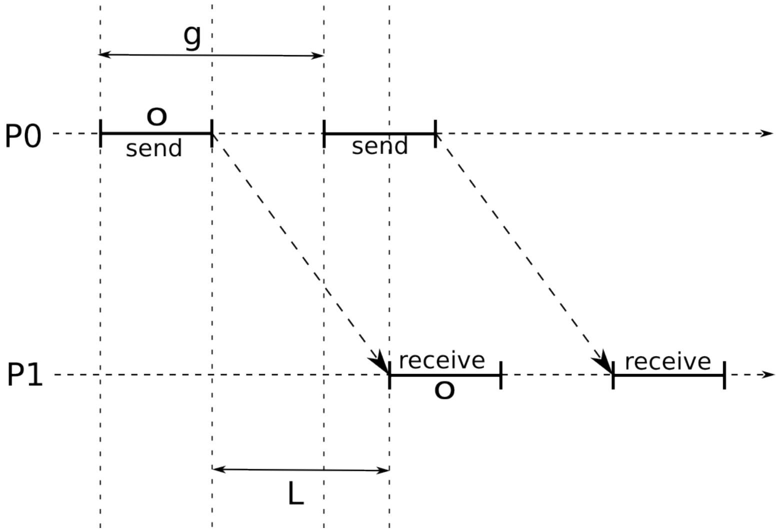
- both sender and receiver block until completion
- a processor cannot send multiple messages simultaneously
- no overlap between communication and computation

The **LogP model** (Culler et al. 1996) takes into account overlap by representing the cost of sending one message (packet) in terms of

- L – network **latency** cost (processor free)
- o – sender/receiver sequential **overhead** (processor occupied)
- $g \geq o$ – **gap** between two sends or two receives (processor free)
- P – number of **processors**
- the LogP communication cost for sending a message of s packets is

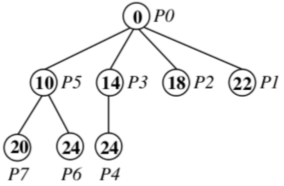
$$T_{\mathrm{sr}}^{\mathrm{LogP}}(s) = 2o + L + (s-1) \cdot g$$

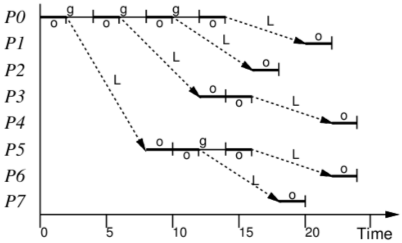
Messaging in the LogP model



Broadcasts in the LogP model

Same idea as binomial tree, forward message as soon as it is received, keep forwarding until all nodes obtain it (Karp et al. 1993)





difficult to define this tree explicitly from model parameters

Limitations of hte LogP model

The LogP model parameter g is associated with an implicit packet size k_{LogP}

- sometimes g is disregarded and o controls bandwidth
- this injection rate implies a fixed-sized packet can be sent anywhere after a time interval of g
- the implicit choice of packet size makes the model inflexible for expressing the cost of messages of a range of sizes

Pipelined binary tree broadcast in LogP

Send a fixed-size packet to left child then to right child

- as before, total message size s, tree height $h \approx \log_2(P)$
- if the LogP model datum size is k_{LogP} bytes, the LogP cost is

$$T_{\text{PBT}}^{\text{LogP}}(s, P) \approx h \cdot (L + 2g + o) + 2(s/k_{\text{LogP}}) \cdot g$$

- we get this cost irrespective of the logical packet size in the protocol k, so long as $k \geq k_{\text{LogP}}$
- we can observe that there is no latency term like $(s/k) \cdot \alpha$, so the protocol achieves noticeable overlap
- LogP may be a good fit for design of hardware-specific collectives

The LogGP model

The **LogGP model** (Alexandrov et al. 1997) introduces another bandwidth parameter G, which dictates the large-message bandwidth

- G – **Gap per byte**; time per byte (processor free)
- the L, o, and g parameters are now incurred for each variable size message, rather than packet
- LogGP time for sending a message of s bytes is

$$T_{\mathrm{sr}}^{\mathrm{LogGP}}(s) = 2o + L + (s-1) \cdot G$$

The LogGP model

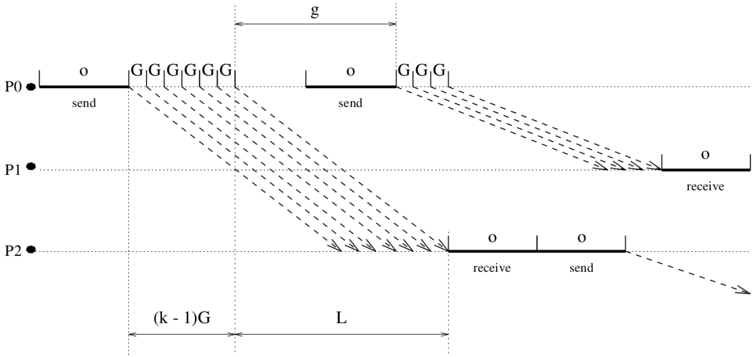


Diagram taken from: Alexandrov, A., Ionescu, M. F., Schauser, K. E., and Scheiman, C. LogGP: incorporating long messages into the LogP model–one step closer towards a realistic model for parallel computation. ACM SPAA, July 1995.

Pipelined binary tree broadcast in LogGP

Send a fixed-size packet to left child then to right child

- as before, total message size s, tree height $h \approx \log_2(P)$
- if the LogP model datum size is k_{LogP} bytes, the LogP cost is

$$T_{\mathrm{PBT}}^{\mathrm{LogP}}(s, P) \approx h \cdot (L + 2g + o) + 2(s/k_{\mathrm{LogP}}) \cdot g$$

- in the LogGP model, we can select a packet size k and obtain the cost

$$T_{\mathrm{PBT}}^{\mathrm{LogGP}}(s, P, k) \approx h \cdot (L + 2g + o + 2k \cdot G) + 2(s/k) \cdot (g + k \cdot G)$$

- minimizing the packet size k

$$k_{\mathrm{opt}}^{\mathrm{LogGP}}(s, P) = \underset{k}{\mathrm{argmin}}(T_{\mathrm{PBT}}^{\mathrm{LogGP}}(s, P, k))$$

  (via e.g. differentiation by k) we obtain the optimal packet size

$$k_{\mathrm{opt}}^{\mathrm{LogGP}}(s, P) = \sqrt{s/h} \cdot \sqrt{\frac{g}{G}}$$

  so the best packet size, depends not only on architectural parameters, but also on dynamic parameters: the number of processors and message size

Pipelined binary tree broadcast conclusions

The LogGP and the α–β models both reflect an input and architectural scaling dependence of the packet size

$$k_{\text{opt}}^{\text{LogGP}}(s, P) = \sqrt{\frac{s}{h}} \cdot \sqrt{\frac{g}{G}}$$

$$k_{\text{opt}}^{\alpha,\beta}(s, P) = \sqrt{\frac{s}{h}} \cdot \sqrt{\frac{\alpha}{\beta}}$$

The LogGP expression is perhaps more insightful, as g appears and not L, so the network latency overhead in the algorithm is partially overlapped.

For the majority of the course we will not analyze overlap, so we will primarily stick to the simpler $\alpha - \beta$ model.

BSP model definition

The **Bulk Synchronous Parallel (BSP) model** (Valiant 1990) is a theoretical execution/cost model for parallel algorithms

- we consider a ‘modern’ interpretation of the model
- execution is subdivided into **supersteps**, each associated with a global synchronization
- within each superstep each processor can send and receive up to h messages (called an h-**relation**)
- the cost of sending or receiving h messages of size m is $h \cdot m \cdot \hat{g}$
- the total cost of a superstep is the max over all processors at that superstep
- when $h = 1$ the BSP model is closely related to the α–β model with $\beta = \hat{g}$ and LogGP mode with $G = \hat{g}$
- we will focus on a variant of BSP with $h = P$ and for consistency refer to $\hat{g}$ as β and the cost of a synchronization as α

Synchronization vs latency

By picking $h = P$, we allow a global barrier to execute in the same time as the point-to-point latency

- this abstraction is good if the algorithm's performance is not expected to be latency-sensitive
- messages become non-blocking, but progress requires a barrier
- collectives can be done in linear bandwidth cost with $O(1)$ supersteps
- enables high-level algorithm development: how many collective protocols does the algorithm need to execute?
- global barrier may be a barrier of a subset of processors, if BSP is used recursively
- BSP can partition processors unevenly to design efficient schedules for irregular applications

(Reduce-)Scatter and (All)Gather in BSP

When $h = P$ all discussed collectives that require a single butterfly can be done in time $T_{\mathrm{butterfly}} = \alpha + s \cdot \beta$ i.e. they can all be done in one superstep

- Scatter: root sends each message to its target (root incurs $s \cdot \beta$ send bandwidth)
- Reduce-Scatter: each processor sends summand to every other processor (every processor incurs $s \cdot \beta$ send and receive bandwidth)
- Gather: send each message to root (root incurs $s \cdot \beta$ receive bandwidth)
- Allgather: each processor sends its portion to every other processor (every processor incurs $s \cdot \beta$ send and receive bandwidth)

when $h < P$, we could perform the above algorithms using a butterfly with 'radix'$=h$ (number of neighbors at each butterfly level) in time $T_{\mathrm{butterfly}} = \log_{h+1}(P) \cdot \alpha + s \cdot \beta$

Other collectives in BSP

The Broadcast, Reduce, and Allreduce collectives may be done as combinations of collectives in the same way as with Butterfly algorithms, using two supersteps

- Broadcast done by Scatter then Allgather
- Reduce done by Reduce-Scatter then Gather
- Allreduce done by Reduce-Scatter then Allgather

BSP preserves this hierarchical algorithmic structure and costs.

However, BSP with $h = P$ can do all-to-all in $O(s)$ bandwidth and $O(1)$ supersteps (as cheap as other collectives).

When $h < P$, the logarithmic factor on the bandwidth is recovered.

Nonblocking communication

Non-blocking messaging with synchronization barriers are used in practice:

- MPI provides non-blocking ‘I(send/recv)’ primitives that may be ‘Wait’ed on in bulk (these are slightly slower than blocking primitives, due to buffering)
- MPI and other communication frameworks also provide **one-sided** messaging which are *non-blocking and zero-copy* (no buffering)
- one-sided communication progress must be guaranteed by a barrier on all or a subset of processors (or MPI Win Flush between a pair)

Systems for one-sided communication

BSP employs the concept of non-blocking communication, which presents practical challenges

- to avoid buffering or additional latency overhead, the communicating processor must know be aware of the desired buffer location of the remote processor
- if the location of the remote buffer is known, the communication is called ‘one-sided’
- with network hardware known as Remote Direct Memory Access (RDMA) one-sided communication can be accomplished without disturbing the work of the remote processor

One-sided communication transfers are commonly be formulated as

- **Put** – send a message to a remote buffer
- **Get** – receive a message from a remote buffer

Partitioned Global Address Space (PGAS)

PGAS programming models facilitate non-blocking remote memory access

- they allow declaration of buffers in a globally-addressable space, which other processors can access remotely
- **Unified Parallel C (UPC)** is a compiler-based PGAS language that allows indexing into globally-distributed arrays (Carlson et al. 1999)
- **Global Arrays** (Nieplocha et al. 1994) is a library that supports a global address space via a one-sided communication layer (e.g. ARMCI, Nieplocha et al. 1999)
- MPI supports one-sided communication via declaration of **windows** that declare remotely-accessible buffers

Matrix multiplication

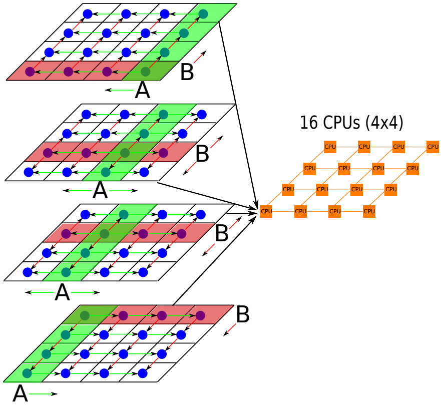
Matrix multiplication of n-by-n matrices A and B into C, $C = A \cdot B$ is defined as, for all i, j,

$$C(i,j) = \sum_k A(i,k) \cdot B(k,j)$$

A standard approach to parallelization of matrix multiplication is commonly referred to as **SUMMA** (Agarwal et al. 1995, Van De Geijn et al. 1997), which uses a 2D processor grid, so blocks A_{lm}, B_{lm}, and C_{lm} are owned by processor $\Pi(l,m)$

- SUMMA variant 1: iterate for $k = 1$ to $\sqrt{P}$ and for all $i, j \in [1, \sqrt{P}]$
    - broadcast A_{ik} to $\Pi(i,:)$
    - broadcast B_{kj} to $\Pi(:,j)$
    - compute $C_{ij} = C_{ij} + A_{ik} \cdot B_{kj}$ with processor $\Pi(i,j)$

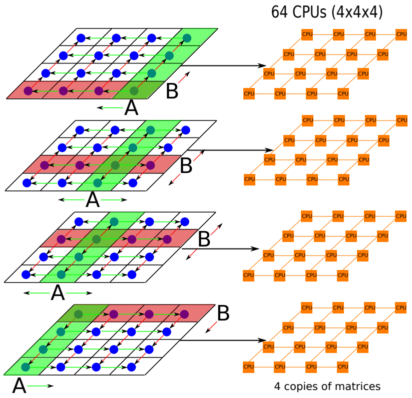
SUMMA algorithm



$$T^{\alpha,\beta}_{\mathrm{SUMMA}} = 2\sqrt{P} \cdot T^{\alpha,\beta}_{\mathrm{broadcast}}(n^2/p, \sqrt{P}) \leq 2\sqrt{P} \cdot \log(P) \cdot \alpha + \frac{4n^2}{\sqrt{P}} \cdot \beta$$

3D Matrix multiplication algorithm

Reference: Agarwal et al. 1995 and others



$$T^{\alpha,\beta}_{\mathrm{3D-MM}} = 2\,T^{\alpha,\beta}_{\mathrm{broadcast}}(n^2/P^{2/3}, P^{1/3}) + T^{\alpha,\beta}_{\mathrm{reduce}}(n^2/P^{2/3}, P^{1/3})$$
$$\leq 2\log(P)\cdot\alpha + \frac{6n^2}{P^{2/3}}\cdot\beta$$

Conclusion and summary

Summary:

- important parallel communication models: α–β, LogP, LogGP, BSP
- collective communication: binomial trees are good for small-messages, pipelining or butterfly good for large-messages
- collective protocols provide good building blocks for parallel algorithms
- recursion is a thematic approach in collectives and communication-efficient algorithms

Backup slides