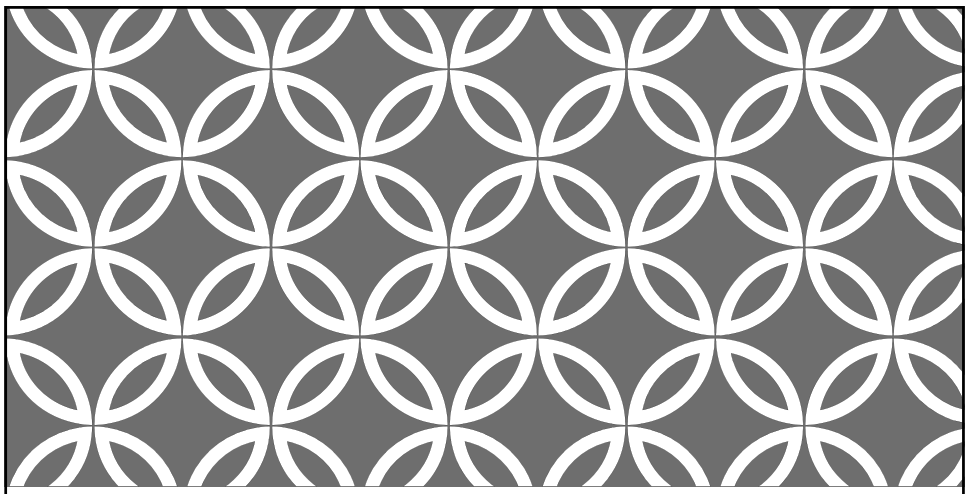
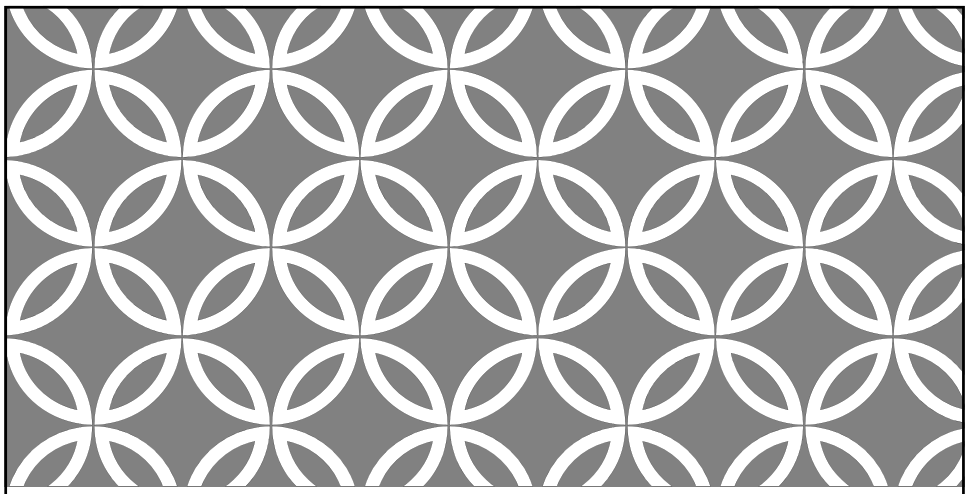


PARALLEL AND DISTRIBUTED ALGORITHMS

BY
DEBDEEP MUKHOPADHYAY
AND
ABHISHEK SOMANI

http://cse.iitkgp.ac.in/~debdeep/courses_iitkgp/PAlgo/index.htm



COMMUNICATION IN HYPERCUBES

2

OVERVIEW

Parallel Sum (Reduction) on Hypercubes

Broadcast

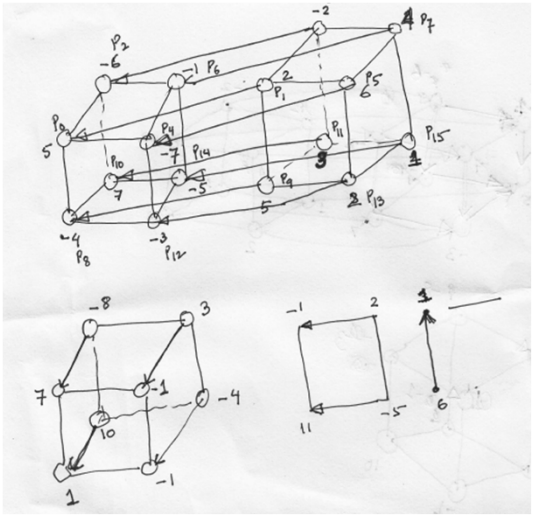
Gather and Scatter Functions

Analysis

Parallel Prefix Sum on Hypercubes

3

PARALLEL SUM (REDUCTION)



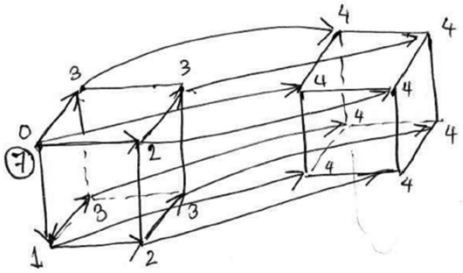
Final result is with processor 0

How can we perform the computation so that every processing element has a copy of the global sum?

By adding a broadcast stage at the end?

4

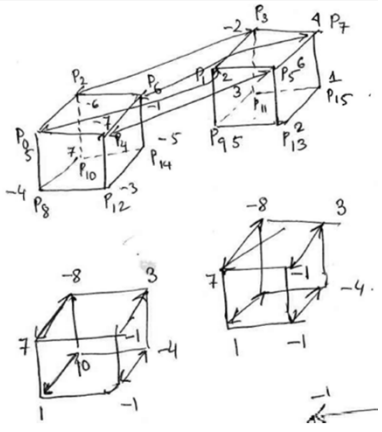
BROADCAST OF SUM FROM NODE 0

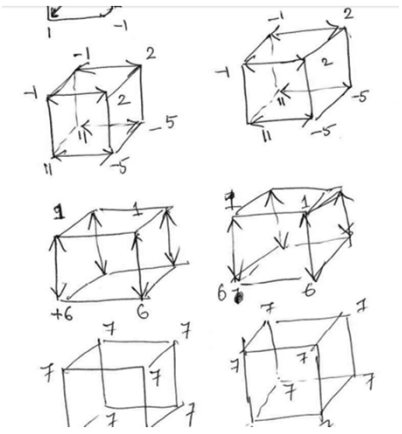


The numbers 1, 2, 3, 4 indicate the iteration number after which all nodes have the sum 7.

5

ALTERNATIVE STRATEGY





6

THE GATHERING OPERATION

There are several problems in which a set of computations must be performed on all pairs of objects in a set of n objects.

A straightforward sequential algorithm would require time $\theta(n^2)$

Gather operation is a parallel approach used in multiprocessors based on message passing.

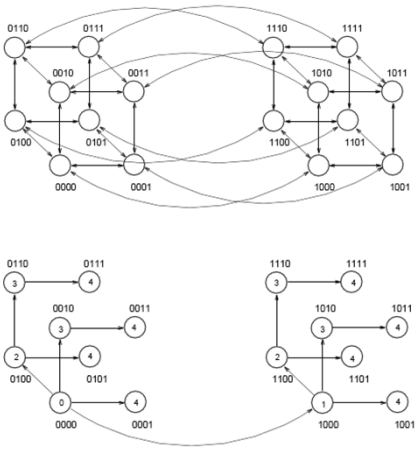
A Gather operation is a global communication that takes a data set distributed among a collection of tasks and gathers it into a single task.

- This is different from reduction, in the sense that reduction performs the **composition** of a binary reduction operation on all of the data. On the contrary, gather **copies** the data from each task into an array of these items in a single task.

All gather operation: collects the data from all tasks and makes a copy of the entire dataset in each task.

7

A HYPERCUBE BASED GATHER



There will be 4 iterations, one for each bit position.

In the first iteration, all nodes whose labels are identical except for the most significant bit position will exchange data.

There are $2^{4-1}=8$ pairs of such nodes in every iteration. Eg in 1st iteration:

0000 ⇔ 1000

0001 ⇔ 1001

0010 ⇔ 1010

0011 ⇔ 1011

0100 ⇔ 1100

0101 ⇔ 1101

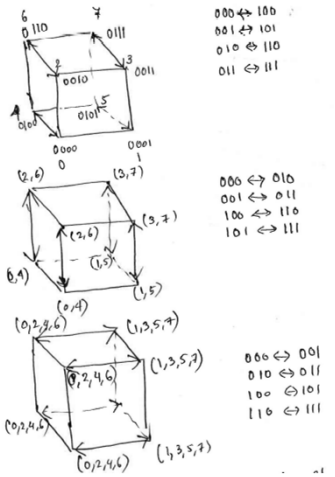
0110 ⇔ 1110

0111 ⇔ 1111

In the second iteration, all pairs of nodes whose labels are the same except for the second MSB will exchange values. And so on…

8

AN EXAMPLE



Observe that the message length in the channel doubles!

9

ANALYSIS: LATENCY AND BANDWIDTH

The all-gather communication takes logp steps, but the size of the messages double in every step.

In the first step, the messages have size n/p, in the second size 2n/p, and so on.

- In the k^{th} step, the messages have size $2^{k-1}n/p$.

In the previous analysis of reduction, we did not consider the message size in the delay because all the messages were the same size (why?).

10

MODELING COMMUNICATION DELAY

Amount of time required by a task to send a message has two components:

Latency: time to initiate the transmission

Transfer Time: Time spent sending the message through the channel. The longer the message, longer the transfer time.

We represent the latency by λ.

The channel bandwidth is represented by β (data items per unit time).

To send a message with d-data items, time required is $\lambda + d/\beta$.

In the kth step, the communication time is $\lambda + (2^{k-1}n)\ /(\beta p)$.

11

COMMUNICATION DELAY

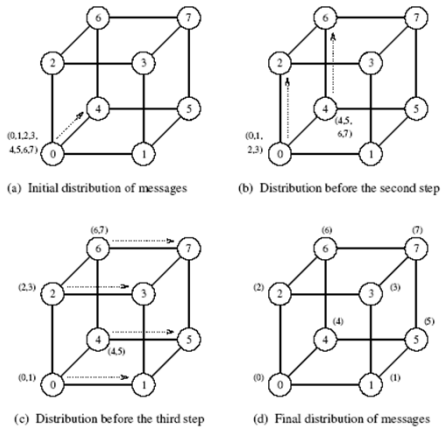
$$T_d = \sum_{k=1}^{logp} \left(\lambda + \frac{2^{k-1}n}{\beta p}\right) = \lambda logp + \frac{n}{\beta p}\sum_{k=0}^{logp-1} 2^k$$

$$= \lambda logp + \frac{n}{\beta p}\left(2^{logp} - 1\right)$$

$$= \lambda logp + \frac{n(p-1)}{\beta p}$$

12

EXAMPLE OF THE SCATTER OPERATION



The scatter operation on an eight-node hypercube.

COST OF SCATTER

There are *log p* steps, in each step, the machine size halves and the data size halves.

We have the time for this operation to be (where m is the message sent by each node), $t_s = \lambda, t_w = \frac{1}{\beta}, \frac{n}{p} = m$:

$$T = t_s \log p + t_w m(p-1).$$

This time holds for a linear array as well as a 2-D mesh (Read Introduction to Parallel Computing, Grama et. Al.)

ALL-REDUCE AND PREFIX-SUM OPERATIONS

In all-reduce, each node starts with a buffer of size m and the final results of the operation are identical buffers of size m on each node that are formed by combining the original p buffers using an associative operator.

Identical to all-to-one reduction followed by a one-to-all broadcast. This formulation is not the most efficient. Uses the pattern of all-to-all broadcast, instead. The only difference is that message size does not increase here. Time for this operation is $(t_s + t_w m) \log p$.

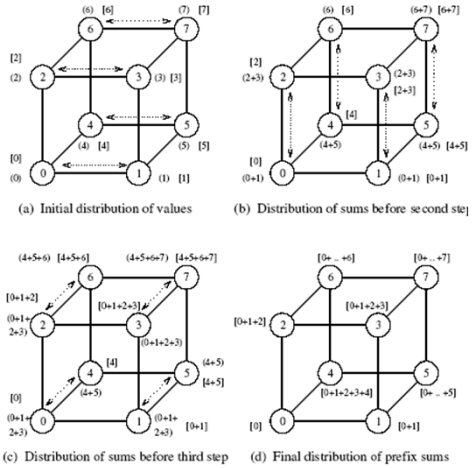
Different from all-to-all reduction, in which p simultaneous all-to-one reductions take place, each with a different destination for the result.

THE PREFIX-SUM OPERATION

Given p numbers $n_0, n_1, \ldots, n_{p-1}$ (one on each node), the problem is to compute the sums $s_k = \sum_{i=0}^{k} n_i$ for all k between 0 and $p-1$.

Initially, n_k resides on the node labeled k, and at the end of the procedure, the same node holds S_k.

THE PREFIX-SUM OPERATION



Computing prefix sums on an eight-node hypercube. At each node, square brackets show the local prefix sum accumulated in the result buffer and parentheses enclose the contents of the outgoing message buffer for the next step.

THE PREFIX-SUM OPERATION

The operation can be implemented using the all-to-all broadcast kernel.

We must account for the fact that in prefix sums the node with label *k* uses information from only the *k*-node subset whose labels are less than or equal to *k*.

This is implemented using an additional result buffer. The content of an incoming message is added to the result buffer only if the message comes from a node with a smaller label than the recipient node.

The contents of the outgoing message (denoted by parentheses in the figure) are updated with every incoming message.

THE PREFIX-SUM OPERATION

```
1.    procedure PREFIX_SUMS_HCUBE(my_id, my_number, d, result)
2.    begin
3.        result := my_number;
4.        msg := result;
5.        for i := 0 to d − 1 do
6.            partner := my_id XOR 2^i;
7.            send msg to partner;
8.            receive number from partner;
9.            msg := msg + number;
10.           if (partner < my_id) then result := result + number;
11.       endfor;
12.   end PREFIX_SUMS_HCUBE
```

Prefix sums on a d-dimensional hypercube.