Design of Parallel Algorithms

Bulk Synchronous Parallel
A Bridging Model of Parallel Computation

Need for a Bridging Model

- The RAM model has been reasonable successful for serial programming
  - The model provides a framework for describing the implementation of serial algorithms
  - The model provides reasonably accurate predictions for algorithm running times
- A bridging model is a model that can be used to design algorithms and also make reliable performance predictions
- Historically, there has not been a satisfactory bridging model for parallel computations. Either the model is good at describing algorithms (PRAM) or is good at describing performance (network model) but not both.
- Leslie Valiant proposed the BSP model as a potential bridging model
  - Basically an improvement on the PRAM model to incorporate more practical aspects of parallel hardware costs

+

What is the Bulk Synchronous Parallel (BSP) model?

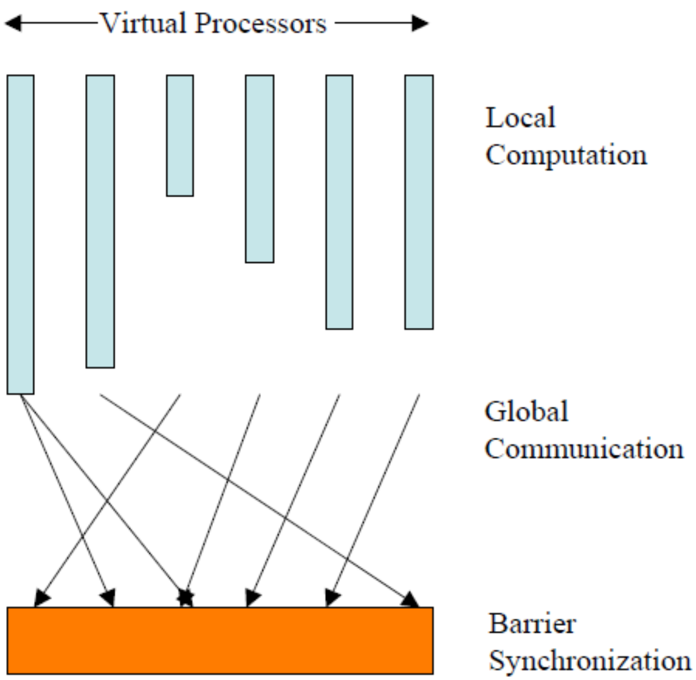
- Processors are coupled to local memories
- Communications happen in synchronized bulk operations
  - Data updates for the communications are inconsistent until the completion of a synchronization step
  - All of the communications that occur at the synchronization step are modeled in aggregate rather than tracking individual message transit times
- For data exchange, a one-sided communication model is advocated
  - E.g. data transfer through **put** or **get** operations that are executed by only one side of the exchange (as opposed to 2 sided where send-receive pairs must be matched up.)
- Similar to a coarse grained PRAM model, but exposes more realistic communication costs
- BSP provides realistic performance predictions

Bulk Synchronous Parallel Programming

- Parallel Programs are developed through a series of super-steps
- Each super-step contains:
  - Computations that utilize local processor memory only
  - A communication pattern between processors called an h-relation
  - A barrier step whereby all (or subsets) of processors are synchronized
    - The communication pattern is not fully realized until the barrier step is complete
- The h-relation:
  - This describes communication pattern according to a single characteristic of the communication identified by the parameter called h
  - h is defined as the larger of the number of incoming our outgoing interactions that occur during the communication step
  - Time for communication is assumed to be $mgh+l$ where m is the message size, g is an empirically determined bulk bandwidth factor, and l is an empirically determined time for barrier synchronization

Architecture of a BSP Super-Step

- The super-step begins with local computations
- In some models, virtual processors are used to give the run-time system flexibility to balance load and communication
- Local computations are followed by a global communication step
- The global communications are completed with a barrier synchronization
- Since every super-step starts after the barrier, computations are time synchronized at the beginning of each super-step



+

Cost Model for BSP

- The network is defined by two bulk parameters
  - The parameter g represents the average per-processor rate of word transmission through the network. It is an analog to t_w in network models.
  - The parameter l is the time required to complete the barrier synchronization and represents the bulk latency of the network. It is an analog to t_s in network models.

- The cost of a super-step can be computed using the following formula
  - $t_{step}=max(w_i) + mg\ max(h_i) + l$
  - w_i is the time for local work on processor i
  - h_i is the number of incoming or outgoing messages for processor i
  - m is the message size
  - g is the machine specific BSP bandwidth parameter
  - l is the machine specific BSP latency parameter

Example of BSP implementations of broadcast (central scheme)

- Since there is no global shared memory in the BSP model, we need to broadcast a value before it can be used by all processors
- There are several ways to implement broadcast algorithms, a central scheme would perform the broadcast by using one super-step with one processor communicating with all other processors. This we call the central scheme.
- In this approach the h relation will be $p-1$ since one processor will need to send a message to all other processors.
- The cost for this scheme is $t_{central} = gh+l = g(p-1) + l$

Example: BSP broadcast using binary tree scheme

- Broadcast using a tree approach where the algorithm proceeds in $\boldsymbol{\log p}$ steps
- Each step, every processor that presently has broadcast data sends to a processor that has no data
  - Processors that have broadcast data doubles in each step
- Since each processor either sends or receives one or no data each step, the $\boldsymbol{h}$ relation is always $\boldsymbol{h=1}$
- The time for each step of this algorithm is $t_{step} = g+l$
- The time for the overall broadcast algorithm that includes all log p steps
  - $t_{tree} = (g+l) \log p$

Optimizing broadcasts under BSP

- The central algorithm time:
  - $t_{central} = g(p-1) + l$
- The tree algorithm time:
  - $t_{tree} = (g+l) \log p$
- If $l >> g$ then for sufficiently small p, then $t_{central} < t_{tree}$
- Can we optimize broadcast for specific system where we know g and l?
  - There is no reason that we are constrained only double in each step, We could triple, quadruple, or more each step.
  - Combining the central and tree algorithm can yield an algorithm that can be optimized for architecture parameters

+

Cost of the hybrid broadcast algorithm

- Each step of the algorithm, processors that have data will communicate with $k\text{-}1$ other processors, therefore $h=k\text{-}1$ in each step
- After $\log_k p$ steps, all processors will have shared the broadcast data
- Therefore the cost of each step of the hybrid algorithm is $(k\text{-}1)g$ and so the cost of the hybrid algorithm is $t_{hybrid} = ((k\text{-}1)g + l)\log_k p$
- To optimize set k such that $t_{hybrid}'(k)=0$, from this we find optimal k set by
  - $l/g = 1+k*(\ln(k)\text{-}1)$
- For a general message of m words, the broadcast algorithm can be shown to be $t_{hybrid} = (m(k\text{-}1)g + l)\log_k p$, and the optimal setting for k becomes
  - $l/(mg) = 1+k*(\ln(k)\text{-}1)$

Practical application of BSP

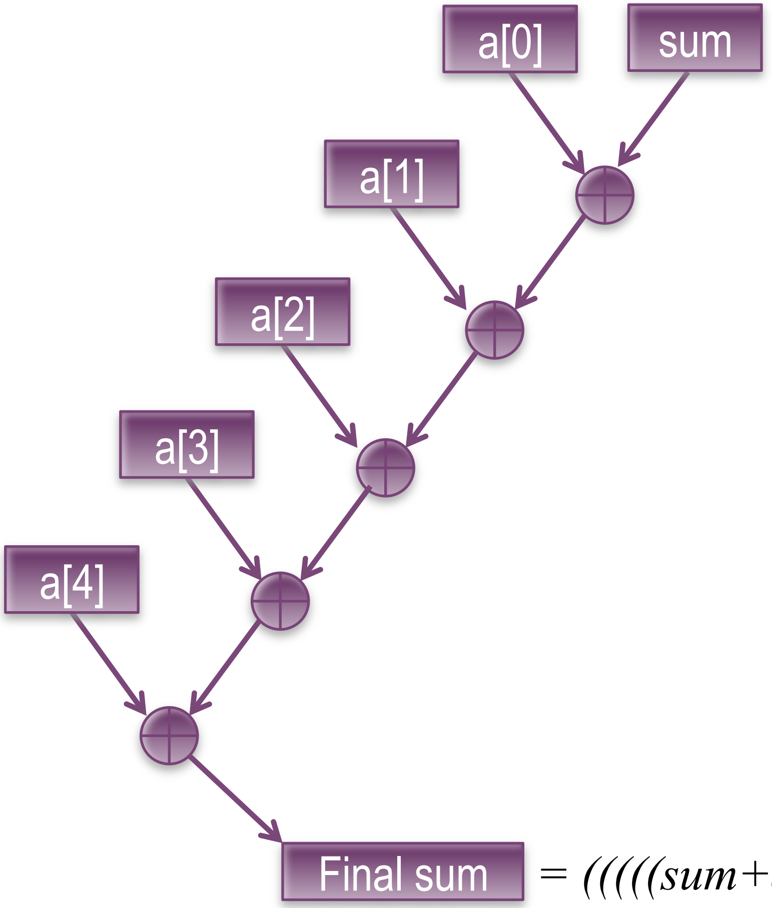
- Several parallel programming environments have been developed based on the BSP model
- The second generation of the MPI standard, MPI-2, has an extended its API to include a one-sided communication structure that can emulate the BSP model (e.g. it is one-sided + barrier synchronization)
- Even when using two sided communications, parallel programs are often developed as a sequence of super-steps. Using the BSP model, these can be analyzed using a bulk view of communications.
- The BSP model assumes that network is homogenous, but architectural changes, such as multi-core architectures, present challenges
  - Currently model is being extended to support hierarchical computing structures

Discussion Topic

- Implementation of summing n numbers using BSP model
- Serial Implementation:

```
int sum = 0 ;
for(int i=0;i<n;++i)
  sum = sum + a[i] ;
```

Dependency graph for serial summation



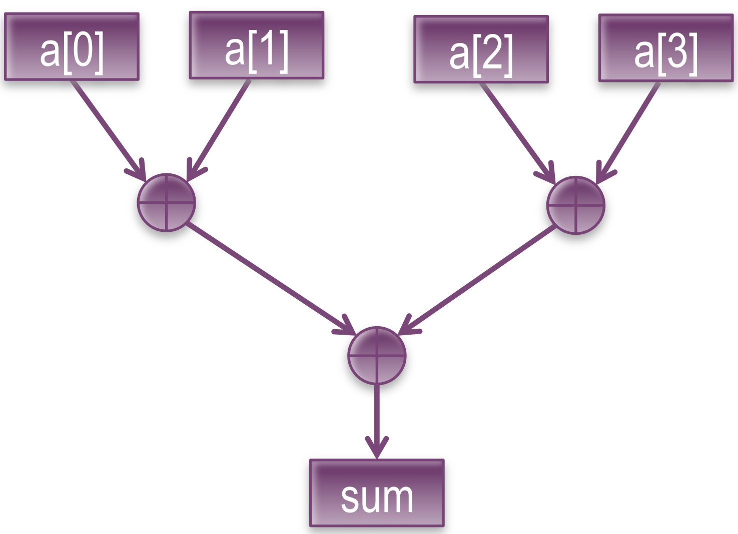
$= (((((sum+a[0])+a[1])+a[2])+a[3])+a[4])$

Problems with parallelizing the serial code

- The dependency graph does not allow one to perform subsequent operations.
  - It is not possible, as the algorithm is formulated, to execute additions in parallel
- We note that the addition operation is associative
  - NOTE! This is not true for floating point addition!
  - Although floating point addition is not associative, it is approximately associative
    - Accurately summing large numbers of floating point values, particularly in parallel, is a deep problem
    - For the moment we will assume floating point is associative as well, but note that in general an optimizing compiler cannot assume associativity of floating point operations!
- We can exploit associativity to increase parallelism

How does associativity help with parallelization?

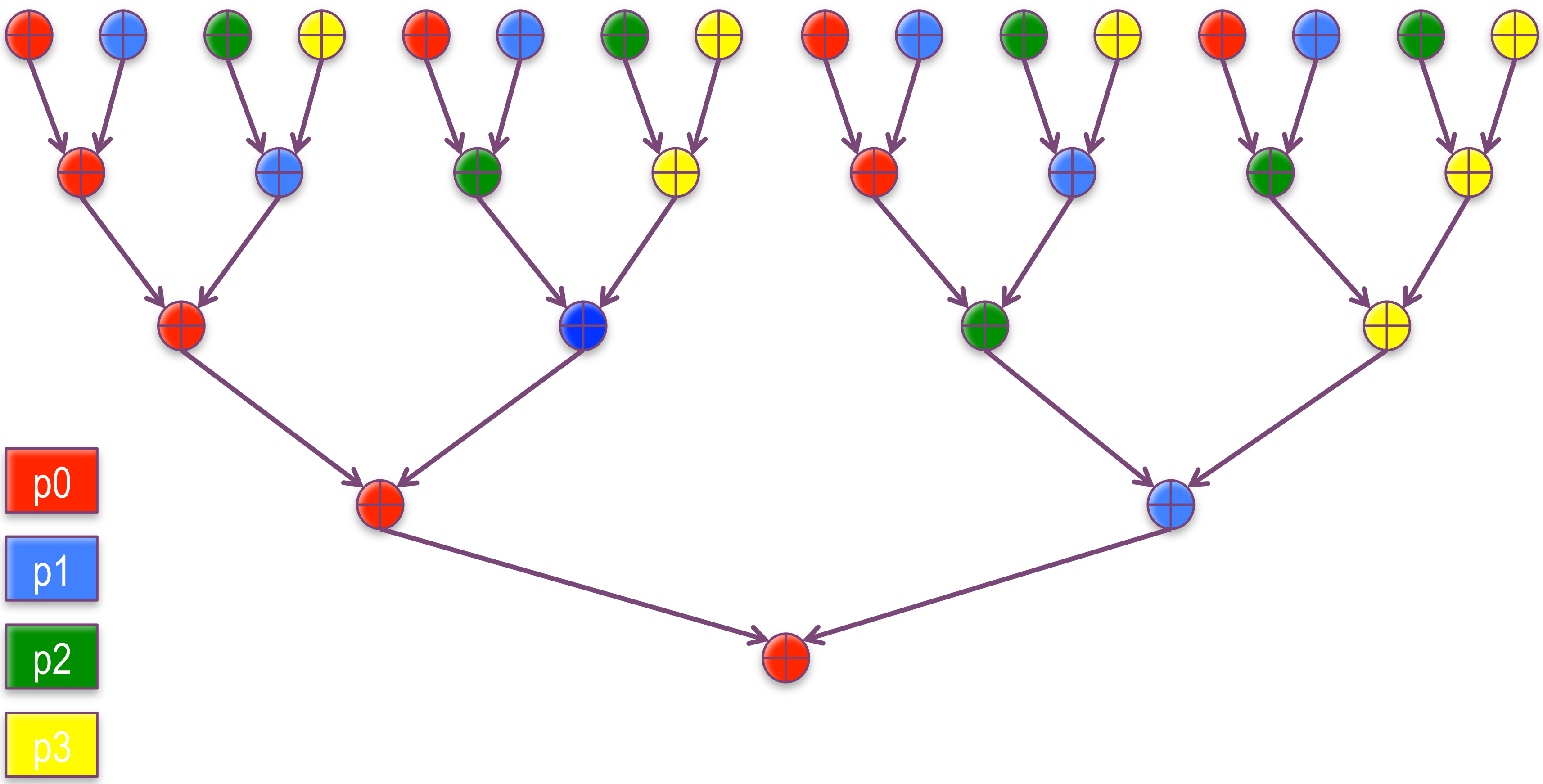
- We can recast the problem from a linear structure to a tree:
  - $((((a0+a1)+a2)+a3) = ((a0+a1)+(a2+a3))$
  - Now $a0+a1$ and $a2+a3$ can be performed concurrently!



What are the costs of this transformation

- Using operator associativity we are able to reveal additional parallelism, however there are costs
  - For the serial summing algorithm only one register is needed to store intermediate results (we used the sum variable)
  - For the tree based summing algorithm we will need to store $n/2$ intermediate results for the first concurrent step
- For summing where $2n >> p$, maximizing concurrency may introduce new problems:
  - Storing extra intermediate results increase memory requirements of algorithm and may overwhelm available registers
  - Assigning operations to processors (graph partitioning) is needed to parallelize the summation. Some mappings will introduce significantly more inter-processor communication than others

Mapping Operators to Processors
Round Robin Allocation

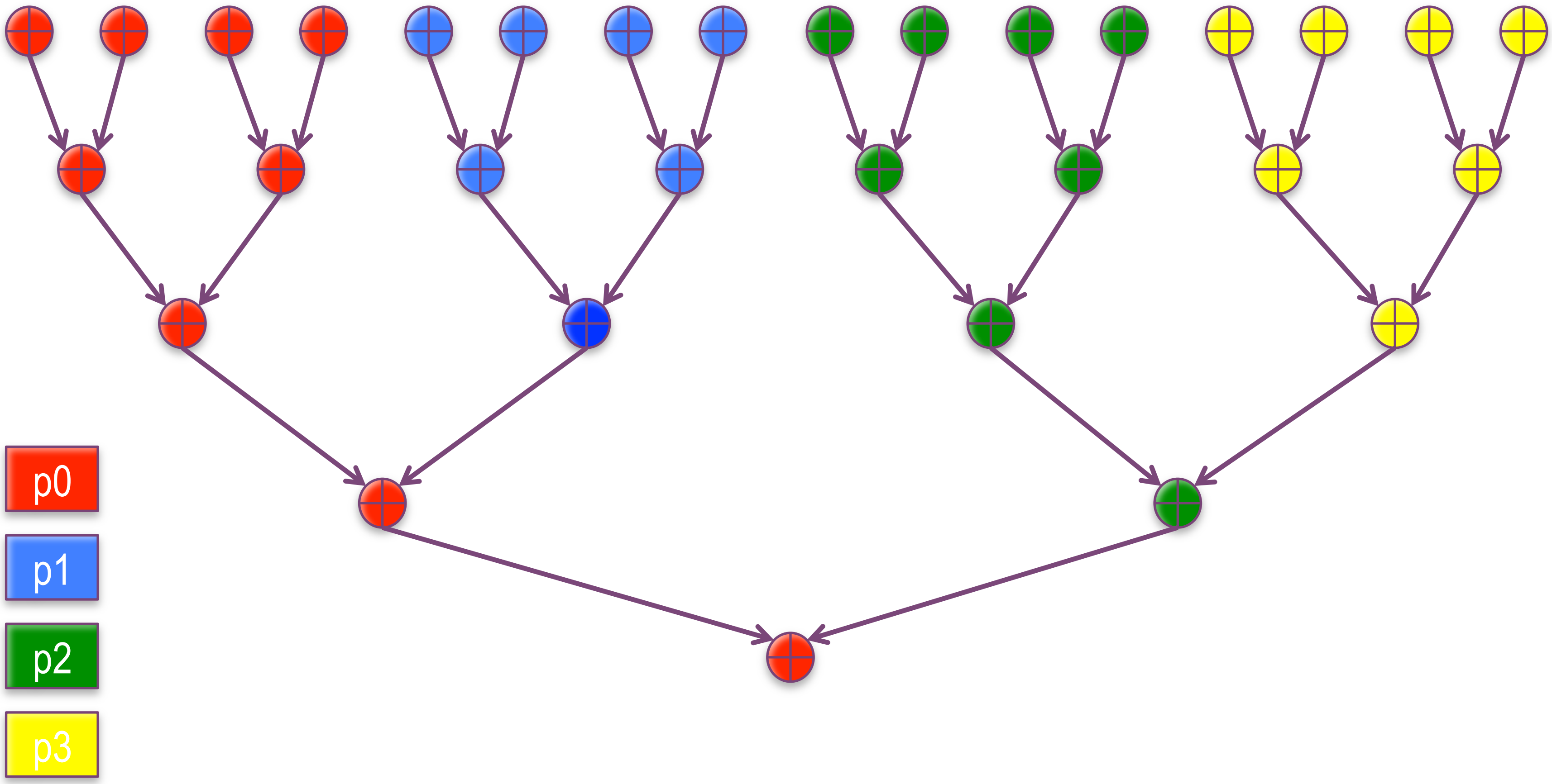


BSP model for round robin allocation of the tree

- Since there is communication for each level of the tree, there will be ***log n*** super-steps in the algorithm
- For level ***i*** in the tree, the algorithm will perform ***max(n/(2ip),1)*** operations on at least one processor.
- For level ***i*** in the tree, the algorithm will utilize an ***h*** relation where ***h*** = ***max(n/(2ip),2)***
- Therefore the running time to sum n numbers on p processors using the BSP model is

$$t_{sum} = \sum_{i=1}^{\log n} \left\{ \left\lceil \frac{n}{2ip} \right\rceil t_c + \left\lceil \frac{n}{4ip} \right\rceil 2g + l \right\} \cong \frac{n}{p}(t_c + g) + l \log n$$

Mapping Operators to Processors Communication Minimizing Allocation



+

BSP model for optimized allocation sum

- Notice that only the last ***log p*** levels of the tree will require communication between processors, therefore there will be only ***log p*** super-steps
- The first step will require ***n/p-1*** operations per processor, and the remaining steps will only require ***1*** operation
- During these final ***log p*** steps, at most a processor either receives or send one piece of information, and so ***h = 1*** for the h-relation
- From this the BSP model running time can be derived:

$$t_{sum} = \left(\frac{n}{p} - 1\right) t_c + \sum_{i=1}^{\log p} \left\{ t_c + g + l \right\} = \left(\frac{n}{p} - 1\right) t_c + \left(t_c + g + l\right) \log p$$

Comments on BSP analysis

- Obviously, in the BSP model, different allocations of work to processors can have radically different running times even though the work is equally balanced.
- For a PRAM model, both allocations would have had the same cost which is unrealistic.
- The cost structure of the BSP algorithms favors algorithms that have greater locality
- Even if we do not explicitly use a BSP model, we typically think of our algorithm going through a sequence of steps even if the implementation never explicitly enforces a barrier to get all processors to a unified state. Therefore the BSP model closely matches how we typically think about practical parallel programs.

Q&A