ECS223a Parallel Algorithms
Lecture Notes of BSP

The BSP (Bulk Synchronous Parallel) model is intended to get closer to real distributed systems, particularly clusters or networks of processors. It introduces an explicit cost of communication, and lets us state asymptotic results about how a parallel algorithm balances communication and computation.

1 The BSP model

A BSP computer has multiple processors, each with a substantial local memory, connected together by a router. All non-local memory access must take place over the router. An algorithm is organized into supersteps, which involve both local computation and non-local memory accesses. Local computation is measured in instructions, all of which take the same amount of time, exactly as in the usual sequential asymptotic analysis; so we use a sequential instruction as our unit of time. During a superstep, we assume that each processor can send and receive at most h messages of constant size. This assumption puts a limit both on total router traffic and traffic at each processor; we assume that when these limits are met the router can deliver each message reliably in a fixed time g, which is measured in instructions. The data received in one superstep is available only during the next superstep. We define the parameters an algorithm requires as:

$$L = \frac{\text{instructions}}{\text{superstep}}$$

$$h = \frac{\text{messages}}{\text{superstep}}$$

$$g = \frac{\text{instructions}}{\text{message}}$$

In general we describe the running time of a BSP algorithm as

$$L * k + g * h * k$$

where k is the number of supersteps required.

2 Matrix Multiply

Say we multiply two $n \times n$ matrices together to get another $n \times n$ matrix. This is an example of an embarrassingly parallel algorithm. We parallelize by breaking the output matrix up into p sub-matrices, also square, each of size $n/\sqrt{p} \times n/\sqrt{p}$ and containing n^2/p elements each, and letting one processor compute each sub-matrix. On a BSP, the number of supersteps is one.

To compute a submatrix, a processor has to read $n/\sqrt{p}$ rows and $n/\sqrt{p}$ columns, and multiply all row-column pairs, so the computation at each processor is $\Theta(n^3/p)$.

We can use the BSP model to get a sense of how slow the router can afford to be with respect to the processors. Straightforward matrix multiply is $O(n^3)$ sequentially, so if we can parallelize it perfectly on p processors, the running time, including both computation and communication would be $O(n^3/p)$. So we set

$$L * k + g * h * k = \Theta(n^3/p) + g\Theta(n/\sqrt{p}) = O(n^3/p)$$

Obviously, the smaller g the better. But solving for g, we find that even when $g = O(n/\sqrt{p})$, the algorithm remains optimal. Since this is quite a large value of g, we see that that the algorithm remains dominated by computation.

3 Broadcast

To broadcast a value from one processor, say p_0, to all of the others, we use a tree-shaped communication pattern. So p_0 sends the value to d other processors, where d is the degree of the tree, they each send it on to d others, and so on. This tree will have $\lg_d p$ levels. The number of messages each processor sends or receives in a superset, h, must be at least d, and $2 \leq d \leq p$. We can use the BSP model to consider how we could choose d given the other machine parameters.

The BSP running time is:

$$L * k + g * h * k = \Theta(1) \lg_d p + gd \lg_d p = \Theta(g \lg p(d/\lg d))$$

since $\lg_d p = \lg p / \lg d$. This tells us that we should reduce d to speed up the program, since the linear time required to do d-way communication swamps the fact that we reduce the number of levels in the tree; we should always choose $d = 2$.

But this does not coincide with our experience of real parallel systems Generally the network can handle some constant level of traffic per superstep, and only when the traffic is greater does communication become linear in the number of messages. Also, there is a cost for synchronization at every superstep. Another version of the model, the Coarse Grained Multiprocessor BSP (CGM-BSP), promoted by Dehne and Fabri, adds a factor s to represent this minimum amount of time per superstep. In CGM-BSP, we get time

$$L * k + g * h * k + s * k$$

Applying this to broadcast, we find that the time is

$$(\Theta(1) + gd + s)(\lg p / \lg d)$$

To optimize this asymptotically, we should choose d as small as possible so that $gd \geq s$, that is, $d = \Theta(s/g)$.

4 Parallel Prefix

Parallel prefix is also tree-shaped computation. On a BSP machine, the prefix computation on an input array of n elements begins by breaking the computation into subarrays of size n/p, and computing the prefix operation on those subarrays. These are then combined in groups of size d and forwarded to p/d processors on the second level. These do the prefix computation on subarrays of size d, and pass the results on in groups of d to processors on the third level, and so on.

The total time (again in the CGM-BSP model) is:

$$\begin{aligned}\Theta(n/p) + L * k + g * d * k + s * k &= \Theta(n/p) + \Theta(d) \lg_d p + gd \lg_d p + s \lg_d p \\ &= \Theta(n/p) + (\Theta(d) + gd + s)(\lg p / \lg d)\end{aligned}$$

Similar to the analysis for broadcast, we find that choosing d such that $d = \Theta(s/(c+g))$, where c is a lower bound on the $\Theta(d)$ computation per node, gives the best running time.

The paper asks a different question: for what range of d is the algorithm optimal, assuming that $h = d$, in the basic BSP model? Since parallel prefix is $O(n)$ time sequentially, an optimal time on p processors would be $O(n/p)$. So for the running time to be optimal on p processors, we would want:

$$\begin{aligned}\Theta(n/p) + (d + gd)(\lg p / \lg d) &= O(n/p) \\ d/\lg d &= O(n/(gp \lg p)) \\ d &= O([n/(gp \lg p))] \lg[n/(gp \lg p))])\end{aligned}$$

Since $2 \le d \le p$, this bound is interesting when $n = \Omega(gp \lg p)$ and $n = O(gp^2)$.