Computer Science Division
UC Berkeley
CS273: Parallel Models, Spring 2001 Lecturer: Satish Rao
Lecture #2 and 3

Relating an Application to a Parallel Architecture

We give an example of a parallel application and a parallel architecture, and discuss how to use a model to argue about the execution of the application on the architecture.

1 A Simple Parallel Computer

A parallel computer consists of p processors and an interconnection network that communicates in rounds or *batches* and that is characterized by two parameters, L and g. A communication batch requires $L + gh$ time where h is the maximum number of messages that are sent or received by any processor. We call such a batch of messages an h-relation. (This is Valiant's BSP Model. We note that for the following discussion we could equally well use the LogP model.)

2 Iterative Relaxation

As an example of a parallel application, we consider the following problem, called *iterative relaxation.*

$$\text{Input : Sparse matrix } A\text{; vector } x_{input}\text{; integer } T$$
$$\text{Computation: } x^i = A^i \cdot x_{input}$$
$$\text{Output: } x^T.$$

A graph $G = (V, E)$ can be naturally defined from A, where each node in V corresponds to a row of A, and where the set of edges E consists of all directed arcs (u, v) such that $A(v, u)$ is non-zero. We further define $w(u, v)$ to be the value of the matrix element $A(v, u)$. Then x_v^{i+1}, the value at node v in timestep $i + 1$, can be computed as $\sum_{(u,v)\in E} x_u^i \cdot w(u, v)$. We say that a node u is in $N(v)$ if $(u, v) \in E$ and $u \neq v$. We define $N(S) = (\bigcup_{v\in S} N(v)) \setminus S$.

Note that the value at node v at timestep $i+1$ can be computed from the values of $N(v)$ at timestep i. This observation leads us to the following straightforward algorithm for iterative relaxation.

- The nodes V are partitioned into p sets $S_0, \ldots S_{p-1}$ among processors. We call the nodes in S_i the *home nodes* of processor i.

- Before execution of a time step, each processor has the current values of its home nodes and of all nodes in $N(S_i)$.

Relax (G, T)
    for($i = 1$ to T)
        for each home node v of p
            $x_v^i = 0$
            for each $e = (w, v) \in E$
                $x_v^i = x_v^i + x_w^{i-1} \cdot w(w, v)$
        RefreshBorderValues(x)

Notice that communication occurs only in RefreshBorderValues, and can be performed in a single batch. The total number of messages sent by processor i is $|N(V - S_i)|$, and the total number received by processor i is $|N(S_i)|$. Thus, the total time required by the procedure above is

$$T \cdot \left(L + g \cdot \max_{0 \leq i < p} \left(\max(N(S_i), N(V - S_i)) \right) + \max(E_i) \right),$$

assuming that each iteration of the inner loop takes one time unit. (E_i is defined as the set of edges directed to a node in D_i.) The time for a sequential machine is $T \cdot |E|$.

We assume in the following that the graph is planar and that it is laid out according to the following theorem.

Theorem 1 *(Fredrickson, Lipton/Tarjan) There is a partition $\{S_0, ..., S_{p-1}\}$ for any N-node planar graph such that*

(1) $\max(|N(S_i), |N(V - S_i)|) \leq c\sqrt{N/p}$, *and*

(2) $|E_i| \leq (1 + \epsilon)|E|/p$,

where $\epsilon > 0$ and c is a constant depending on ϵ.

Thus, the total time becomes

$$T \cdot (L + g \cdot c\sqrt{N/p} + (1 + \epsilon) \cdot |E|/p).$$

When is this a good result? We define success as .50-efficiency, i.e., a speedup of $p/2$ on p processors. Now we assume that $\epsilon = 0$ (helps us in terms of speedup) and $|E| = N$ (hurts us), for ease of calculation. Here the overhead of each iteration, $L + g \cdot c\sqrt{N/p}$, should be less than the amount of useful work done, N/p. This holds as long as

$$N \geq \max(2pL, 4c^2g^2p). \tag{1}$$

3 The Circuit Switched Benes Network

A *butterfly network* can be inductively defined as follows. A 1-input butterfly network is a single node. A butterfly network with $p = 2^n$ inputs consists of 2^n input nodes each labelled by an n-bit string $x_0x_1..x_{n-1}$, two sub-butterfly networks with 2^{n-1} inputs, and a switch for each pair of inputs $x_0x_1...x_{n-1}$ and $\overline{x_0}x_1...x_{n-1}$ that connects these nodes to the inputs of the two sub-butterfly networks that are labelled $x_1, ...x_{n-1}$. As an interconnection network, the butterfly has some nice properties.

- High bisection width: $\Theta(p)$.
- Low diameter: $\Theta(\log p)$.
- Network can easily establish a connection between any input and any output, by changing the bits in order while traversing the network.

Actually, the last property is a little bit problematic, as there may be more than one packet using the same path. In fact, for almost all possible communication patterns there will be conflicts that need to be resolved.

An extension of the butterfly is the *Benes network*, defined as two back-to-back butterfly networks. The Benes network has the same nice properties except that there are many paths between any pair of inputs and outputs. (As an exercise it might be interesting to see how many?) The Benes network also has the following nice property, often referred to as *rearrangeability.*

Theorem 2 *Any one-to-one mapping of the inputs to the outputs of a Benes network can be realized in a conflict-free manner.*

Proof sketch: Use induction. Assume that any permutation can be established in a Benes network with 2^{n-1} inputs. A 2^n-input Benes network consists of two sub-Benes networks along with extra columns of switches before and after the subnetworks. The first column of switches selects which subnetwork is used by a path, while the final column of switches brings the connections out of the subnetwork to the final endpoint. We need to set the connections from these two columns to the subnetworks in such a way that

1. at most one connection is made to each input of a sub-Benes network, and
2. at most one connection is made from each output of a sub-Benes network during the routing of the given permutation.

We form a bipartite graph B, called a *conflict graph*, consisting of two sets of nodes I and O of cardinality $p/2 = 2^{n-1}$ each. Each node is labelled with an $(n-1)$-bit string. For each connection that has to be established between input $i = i_0..i_{n-1}$ and output $o = o_0...o_{n-1}$, we add an edge from node $i_1...i_{n-1}$ to node $o_1...o_{n-1}$ in the graph.

Note that each node has two incident edges, which correspond to conflicting connection requests that may occur at the corresponding node in the network. We wish to assign each of the two connection requests to a different sub-Benes network. This corresponds to a 2-edge coloring of the bipartite graph, i.e., a coloring of the edges with colors 0 and 1 such that no two edges incident to the same node have the same color. Thus, to finish the proof it suffices to show that any bipartite graph of degree 2 can be 2-edge colored. We leave this as an exercise. □

Using the above result, we can build an interconnection network, called a *circuit-switched Benes network*, with the following properties. (We point out that this is essentially the network used in the IBM GF11 machine.)

(1) For any permutation π, we can compute an appropriate setting of the switches in time $p \log p$ (using a simple sequential algorithm). Given an h-relation, an appropriate collection of switch settings can be computed by first partitioning the h-relation into h permutations using, say, the matching algorithm of Hopcroft and Karp, and then computing the settings for each permutation. (Actually, there may be more efficient ways of computing the settings.)

(2) Once an appropriate switch setting has been computed, a permutation can be routed through the network in time s. (Thus, s is the time it takes to set the switches and have a data packet traverse the circuit-switched network.) Consequently, an h-relation can be routed in time hs.

4 Iterative Relaxation on the Benes Network

We will now discuss how our iterative relaxation algorithm can be implemented on the circuit-switched Benes network. Note that, since the structure of the graph G stays the same throughout the computation, the communication pattern in RefreshBorderValues will be the same in every iteration. By Theorem 1, this pattern is an h-relation with $h \leq c\sqrt{N/p}$. Once we have precomputed the appropriate set of switch settings for this pattern, we can assume a BSP cost model for this computation, with $L = 0$ and $g = s$. If T is large, we can ignore the time spent on the precomputation phase.

Thus, it follows from Equation (1) that we get efficient speedup when

$$N \geq 4c^2 s^2 p,$$

that is, with only $4c^2 s^2$ nodes per processor. We expect that s should be fairly small, in which case we obtain an efficient implementation even for small problem size.

5 Iterative Relaxation via BSP on the Benes Network

Instead of adding the additional precomputation phase, would it have been viable to use the unchanged BSP algorithm for iterative relaxation, and then implement the BSP model on top of the Benes network? To answer this question, we use the following observation about the Benes network.

Corollary 3 *Given an arbitrary packet-routing network Δ of degree k, any communication step of Δ can be simulated in time ks on the circuit-switched Benes network.*

We also use the following result, which will be discussed in more detail later in the course.

Theorem 4 *There exists a graph of degree k that can route any h-relation in $(2 + o(1)) \cdot \frac{h}{k} \log_k p + O(\log p)$ steps, with high probability.*

By choosing $k = p - 1$ (that is, choosing the complete graph) and using Corollary 3, we can implement a BSP computer with $L = p \log p$ and $g \approx 2s$. Thus, we could simply run the generic BSP algorithm for iterative relaxation and get efficient speedup when

$$N \geq \max(2p^2 \log p, 4c^2 s^2 p).$$

For small problem sizes, the efficiency of this approach will be limited by the latency parameter L. If we have more than $p \log p$ nodes per processor, then the efficiency of the algorithm will of course still be limited by the bandwidth parameter g. Note, however, that we only lose a factor of 2 in bandwidth over the direct implementation on the circuit-switched Benes network, which translates into a factor of 4 in the minimum problem size where efficiency is obtained.

Thus, given a sufficiently large problem size (a sufficient amount of parallelism), we can afford to implement our application in a simple and general model, and still achieve a decent level of performance (compared to what we could hope to achieve with a machine-specific implementation). On the other hand, in this particular case, it is maybe not that difficult to write the algorithm directly for the underlying hardware.

6 Under the Rug

For the sake of simplicity, we have swept many issues under the rug in this discussion. Some of these issues are:

(1) It turns out that the decompositions of the planar graphs incur more communication cost if we try to obtain a smaller load imbalance. (That is, there is a trade-off between the parameters ϵ and c in Theorem 1.

(2) We explicitly ignored the time required to compute the data layout and to precompute the switches. We argued that this is justified if we can assume that the time T is sufficiently large. This is an assumption that should always be viewed with suspicion.

(3) The Benes network is nice in bisection bandwidth and, thus, may be expensive to implement for large values of p.