U.C. Berkeley — CS273: Parallel and Distributed Theory Lecture 4
Professor Satish Rao January 29, 2008
Lecturer: Satish Rao Last revised Scribe so far: Satish Rao

Lecture 4

1 Overview.

Today (and possibly next lecture), we give a simple model for parallel computers, design a simple algorithm to run on the computer, and describe an implementation of a restricted version of the model.

2 A simple model for parallel computers.

We will use Valiant's Bulk Synchronous Processing model of parallel computers. In this model, the computer proceeds in steps where one can compute for some time and have a round of communication. The computer is parameterized by p, the number of processors, L the "latency" parameter, and g the bandwidth parameter.

In each step takes time,

$$C_{\max} + (L + h_{\max} g),$$

where $C_{\max}$ is the maximum computation time over all processors, and $h_{\max}$ is the maximum number of messages sent or recieved by any one processor.

Intuitively, g is giving you cost per unit message, while L is giving you fixed cost per step (perhaps measuring communication.)

3 A simple application.

Physicial simulation problems often consists of a graph of some form and a computation that proceeds by exchanging information along the edges of this graph and updating values at the nodes. For example, when one simulates the weather generally activity at one "grid point" depends on the activity at neighboring grid points.

A canonical view of this is a repeated matrix-vector multiplication, where when one views the nonzeros in the matrix as the adjacency matrix of the graph. (The computation of eignvectors, or the Gauss-Seidel iteration, and even conjugate-gradient methods for solving linear systems follow this structure.)

A bit more precisely, we are given a sparse matrix A of dimension n, a vector x and wish to compute $x^t = A^t x$. (By the way, repeated squaring is a bad idea since the matrix becomes dense.)

We consider the graph $G = (V, E)$ be an n-node an edge (u, v) when $A[u, v]$ is nonzero, we define $w(u, v)$ to be $A[u, v]$. Furthermore, x_v^t denotes the value of the vth element of the vectore x^t. Clearly,

$$x_v^{t+1} = \sum_{e=(u,v)} w(u,v) x_u^t.$$

Thus, we define the neighbors of v, $N(v)$, to be the set of nodes u other than v with an edge to v, and the neighbors of a set of nodes S, $N(S)$, to be the neighbors of any node in S but not in S.

Fortunately, this computation is parallel in the sense that each value at step $t+1$ depends only on values at step i. Moreover, for any set of nodes S, the values only depend on the values of S an the values of the neighbors of S, $N(S)$. This leads to a natural parallel implementation.

- The nodes V are partitioned into $S_0, \ldots, S_p$, where S_i is defined as the home nodes of processor i.
- At each timestep t, each processor has all the values of its homenodes, and all the values for $N(S_i)$.

With this invariant, the computation can proceed by computing its new values locally, and then sending each value to whomever needs a copy. See previous lecture notes for psuedo-code.

The total number of messages sent by a processor i is $|N(V - S_i)|$ and the total number of messages received by a processor i is at most $|N(S_i)|$. Thus, the total communication time is

$$T(\max_{0 \le i \le p} |E_i| + L + g \max_{0 \le i \le p}(\max(N(S_i), N(V - S_i)))),$$

where E_i is the number of edges on processor i.

What is the goal here? We wish this time to be as small as possible. Thus, each $|E_i|$ should be around $|E|/p$, and the number of neighbors should be kept as small as possible. Clearly these may be competing concerns.

For some types of matrices that arise in practice, this can be done in a reasonable manner. Typically, in physical simulations, the volume (or the amount of computation) grows faster than the surface area. For example, let's consider a 2-dimensional grid. Here, an $N = k \times k$-node grid can be divided into p pieces (assume that p is a perfect square), such that each piece has size N/p and for each piece $N(S_i)$ (we assume the matrix is symmetric, i.e., $N(V - S_i) = N(S_i)$) has size $4k/\sqrt{p} = 4\sqrt{N/p}$. Thus, the running time becomes

$$T(4N/p + g4\sqrt{N}/p + L).$$

When is this efficient? Well, one notion is that the total speedup over a "good" sequential algorithm is about equal to the number of processors, let's see within a factor of 2. Notice, in this case, that the extra in the term above is simply the factors that we pay for communication. Notice, further that as N gets larger this term gets relatively smaller compared to the N/p computation that one preforms in each step. So, by choosing a large enough N, we will obtain an "efficient" parallel algorithm.

In particular, our "efficiency" is aboe 50% when

$$4N/p > L + g4\sqrt{N/p}.$$

Being a bit less precise, we can simplify to having $N >> Lp$ and $N >> g^2p$.

Question 1: If one had a 3 dimensional grid with $N = k \times k \times k$, how large would N have to be for one to get an efficient parallel algorithm. (Here, the latter form would be fine, i.e. $N >> \ldots$ and $N >> \ldots$.)

3.1 An aside.

Let me remark about this style of analysis. Does this make sense? Well, if a problem is very small why run it on a parallel computer? If on the other hand it is very very large, one might presume that it is easy to parallelize (this is not always the case, but bear with me). So, one notion of quality of an algorithm is how large does your problem need to be, in order to produce a good speedup?

On the other hand, this model ignored things like overlapping communication and computation, and also bulked up communications into large chunks. We suggest that this only changes the interesting problem sizes by a bit. That is, hopefully the problem is large enough to get good speedup. If not, with all this other tricks one could possibly speedup on somewhat smaller problems. But, the complexity is likely not worth getting this perhaps small set of problems that could be sped up.

3.2 ..returning....

Later in this course we will study other notions of quality.

We remark that there are large classes of graphs, beyond just grids, that have good partitions. For example, any graph that can be drawn in the plane essentially looks like a grid in terms of how easy it is to partition as specified by the following theorem.

THEOREM 1
(Fredrickson, Liption/Tarjan) There is a partition of any N-node planar graph into $\{S_0, \ldots, S_p\}$ for any ϵ, there is a c such that

1. $|E_i| \leq (1+\epsilon)|E_i|/p$ *and*
2. $|N(S_i)| \leq c\sqrt{N/p}$.

Here, inefficiency comes both from the imbalance in computation as defined by the ϵ as well as the cost of communicating the values of the neighboring nodes.

Using this theorem, one can again see that one can get an "efficient" algorithm when $N >> Lp$ and when $N = \Omega(g^2)$.

4 Finding good partitions.

We first see how to find a balanced separator. That is, given a graph $G = (V, E)$, find a subset of nodes C whose removal leaves A and B, such that $|A| \leq 2n/3$ and $|B| \leq 2n/3$, and there is no edge in E between a node in A and a node in B.

We start with an easy one.

Theorem 2
A tree has a separator of size 1.

Proof:
Direct each edge toward the side with more nodes. Remove the node with no outgoing arcs. Each component has size at most $n/2$. Prove the following claim.

Question 2. Given a set of components $A_1, \ldots, A_k$, all of which have size less than $n/2$, one can group them into sets A and B each of which has size at most $2n/3$.
□

Now, we will try to prove the following interesting theorem.

Theorem 3
Any n-node node planar graph has a separator of size $4\sqrt{n}+1$. *(The*

Proof:
We will first prove the following lemma.

Lemma 4
Any planar graph with a spanning tree of depth r has a set of nodes of size $2r+1$ *whose removal leaves no connected component that is larger than* $2n/3$.

We use the following fact.

Fact 1 Any closed curve in the plane divides the plane into two pieces, the inside and outside.

For us, we use this theorem by noting that the a planar graph can be drawn in the plane and a cycle in the graph corresponds to a closed curve in the plane. Thus, the nodes on the inside of the closed curve that corresponds to this cycle's embedding are separated from the nodes on its outside.

The first step is to triangulate each face in the planar embedding.

Now, recall that if we have a spanning tree, that any other edge in the graph induces a cycle in the graph. That is, any of the edges in the triangulated graph that is not in the spanning tree creates a cycle in the spanning tree and separates the graph.

We consider a "dual" graph that we obtain by placing nodes in each face of the original planar embedding and drawing edges between adjacent faces across the non-spanning tree edges. This forms a tree, of degree at most 3. Each edge in the tree corresponds to an non-spanning tree edge in the original graph, which induces a cycle. Furthermore, each edge corresponds to a cut in the graph. We consider an edge pointing in one direction to contain all the nodes pinned to the tree in a canonical way. (The other direction contain the nodes on the other side of this cycle.)

Let be the nodes on the other side of the cycle from the exterior face. A parent edge's inside contains the inside of both its children. That is, the cut induced by the parent edge contains all the nodes that the children edges contains do, when the are both pointing up.

Now, for each edge, we direct the edge toward the bigger side of the corresponding cut. Here, as with the tree, we will find a node with only incoming edges. At least one of the incoming edges corresponds to a cycle that contains at least 1/3 of the nodes on the other side. Thus, removing this cycle leaves only pieces of size $\leq 2/3$ of the whole graph. This proves the lemma.

Now, we first use a very basic fact about all graphs.

FACT 2 Any level in a breadth first search tree of a graph separates all the nodes above the level from those below the level.

Thus, we first find a breadth first tree, and consider the levels $V_0, V_1, \ldots, V_d$. If $d \leq \sqrt{n}$, we can use the lemma to get the theorem we are seeking.

Otherwise, we can remove a set of levels $V_o, V_{o+\sqrt{n}}, \ldots, V_{o+i\sqrt{n}}$ where $o \leq \sqrt{n}$ such that the total number of nodes in these levels is at most $\sqrt{n}$. This seperates all the nodes into pieces, where each piece can be augmented by adding one node to have a spanning tree of depth $\sqrt{n}$. (Try to sketch a figure here.)

Only one piece can have size larger than $2n/3$. By using the lemma above, we can separate this piece into pieces of size smaller than $2n/3$ using $2\sqrt{n}+1$ nodes.

$\square$

Question 3: Show how to get a 50-50 cut, where the separator size is $O(\sqrt{n})$ in any planar graph. A 50-50 cut, is a partition A, B, C where $|A|, |B| \leq n/2$ and C is the separator. (Hint: Use the planar separator theorem above inductively.)

Question 4: Show that for any n, there is a graph with average degree at most 10 and maximum degre $O(\log n)$, where every balanced cut has size $\Omega(n)$? (Hint: let each node choose 2 random nodes to connect to.)