Lecture 18

1 Overview

In the last lecture, we saw the rather crushing costs of communication. Today we will discuss "Universal networks". That is, networks that work as well as possible given certain constraints in the sense that they can simulate *any* other network. Moreover, we discuss some tools for programming to retain locality.

2 A universal network.

We say a network A is universal with slowdown t for a class of networks, if A can simulate a step of any network B in the class in t steps. (This is basic to computer science and should be familiar since a turing machine is suspected to be universal with (usually) polynomial slowdown. This is Church's thesis.)

We gave our first example of a universal network in $O(\log N)$ time at the beginning of the course. That is, the butterfly network is universal with slowdown $O(\log N)$ for the class of bounded degree graphs. Since, we can route any permutation in $O(\log N)$ time, we can route any d matchings in $O(\log N)$ time, and implement any degree d graph.

In terms of resources it uses N bounded degree chips. That is, the resources it uses is essentially optimal. The slowdown times the resources is $O(\log n)$ times, which we define to be acceptable.

On the other hand, what if our notion of resource included the area required to layout a butterfly. Here the layout area of a butterfly is $\Theta(N^2/\log^2 N)$. Many graphs, for example, binary trees or grids, require significantly less area.

2.1 A diversion

Before discussing area universal networks, we illustrate an alternate notion of simulation. That is, we could perhaps only wish to simulate T steps of "guest" network on a "host" network in sT time, where s is the slowdown. For example, the simulation implies a simulation of this form as well. The simulation above essentially requires an embedding of the graph. This is not always possible. For example, any embedding of a grid into a butterfly requires a wire to have length $\Omega(\log N)$, and thus a 1-step simulation must have this slowdown.

On the other hand, a $T = O(\log n)$ step simulation may be able to do better. Here, we will use the power of duplicating computation.

Here, we divide the grid into subgrids of size around $\sqrt{N}$. Then, we grow each subgrid to include all nodes within $\log^2 N$ hops of it. Then we recursively simulate the subgrids in subbutterflies. Then at the end, we refresh the state of the $N^{3/4}\log^2 N$ overlapping regions. The time recurrence, turns out to be

$T(N,t) = t/\log^2 N T(\sqrt{N} + N^{1/4}\log^2 N, \log^2 n) + O(\log N)$.

Here, the work grows a bit in each recursive level, but not by too much. But, the latency of the long edges is hidden. This sketch can be made formal and results in a simulation with constant slowdown.

3 Area Universal networks.

The problem perhaps for the butterfly is that per unit area there are very few actual processors per unit area. The bulk of the chip is composed of wire. An alternate network which is "dense" in processors is the grid. Here, there is a processor essentially everywhere.

This network is not very good for simulating small diameter networks. For example, to simulate even a binary tree, at least one of the edges must be $\Omega(\sqrt{n}/\log n)$ (this is the same argument that a wire must have length $\Omega(\sqrt{n}/\log n)$ in a binary tree.

Ok, how about a binary tree. Is it area universal? Well, the problem here is that the bandwidth across the root is meager to say the least. To simulate, for example, the grid of area A. The middle wire will have to support communication of $\Omega(\sqrt{A})$. Thus, the slowdown is $\Omega(\sqrt{A})$. Even worse than for the grid simulating the binary tree.

What to do? Well, one wishes to have a low diameter network, with reasonable bisection bandwidth. Let's examine the problem more closely.

We are given a area A layout of some graph (or circuit.) To simulate this circuit with small overhead (say polylogarithmic) we better have a bisection bandwidth of at around $\sqrt{A}$ and diameter $O(\log A)$ and around A processing units.

We can construct such a network based on a 4-ary tree. The root node is replaced by $\sqrt{A}$ nodes which we call its width the group is called the node group of the root node. The children of a node of width W node consists of $W/\sqrt{2}$ nodes. Connections are made by "butterfly"-like switches as follows. The ith node in some binary tree node group of width W is connected to the ith, $i+W$, $i+2W$, and the $i+3W$th node in its parent group.

Note, that this is a recursively defined network, i.e., removing the root group leaves for sub networks. We call it a fat tree.

We can use this network to simulate any area A circuit as follows. We divide the chip into four subpieces. Recursively simulate each of these subpieces. Then, we wish to route the wires between the four subproblems on our "fat" tree.

This can be done using benes like routing at each level in the tree.

Question 1: Describe a randomized scheme for routing a permutation on a fat tree where the maximum congestion is at most $O(\log n)$ assuming that no more than W messages are required to cross any width W node in the graph. Justify your answer but you don't need to give a full proof.

Now, what is the total area of this network. Well, the sidelength is defined by the following recurrence,

$$S(W) \leq 2S(W/2) + W,$$

where W is the width of the root node. This works out to $W \log W$. Thus, the area is $W^2 \log W$. If we take W to be $\sqrt{A}$, we see that we can simulate any area A circuit on a fixed circuit with area $O(A \log^2 A)$ in $O(\log A)$ time.

4 DRAM/ Conservative algorithm.

So far, we have seen the PRAM model of parallel computation that ignores bandwidth constraints. And the BSP model of parallel computation that explicitly models some aspect of communication.

Today, we discuss briefly, the DRAM model which models communication as follows. It defines the complexity of a set of messages M as the following load factor on a network. The load factor of a set of messages on a cut is the number of messages that flow across the cut divided by the number of edges available in the network, denoted by $\lambda(M, C)$. The load factor on the network, $\lambda(M)$, is defined as the maximum load factor of any cut.

Ouch, how does one even reason about such a network. Once again one uses abstraction.

Consider a data structure, that is embedded well into your network. That is, has low load factor. Then, one should try to preserve the good properties of the data structure. This can be done using the following lemma.

Lemma 1
The load factor of a set of pointers does not increase if one replaces the the pointers $\{(a, b), (b, c)\}$ *with* $\{a, c\}$.

Question 2: Prove this lemma.

The lemma can be used to show that the efficient randomized versions of list ranking are conservative.

Question 3: Is pointer jumping conservative?