# Solution Set 2

**Due:** In class on Wednesday, February 18.
Starred problems are optional.

**Problem 2-1.** A *segmented prefix computation* (also called *segmented scan*) consists of a sequence of disjoint prefix computations. For example, given input $x_{11}, \cdots, x_{1N} | x_{21}, \cdots, x_{2N} | \cdots | x_{N1}, \cdots, x_{NN}$, we might want to compute $y_{ij} = x_{i1} \otimes \cdots \otimes x_{ij}$ for $1 \leq i, j \leq N$. Show how to compute an arbitrary segmented scan as a single prefix computation. Your solution should treat the location of segment boundaries as part of the input (e.g., $x_i ='' |''$ if and only if a new segment starts at location i).

**Solution:** In class, we saw that the prefix problem can be solved over any semigroup with linear hardware and logarithmic depth, if computing the composition law of the semigroup is taken as the unit measure for hardware and time. Let $(A, \otimes)$ be the semigroup of the $x_{ij}$'s. Now we define a new semigroup $(B, \odot)$, which also includes the separation characters. Define $B = A \cup \{|\} \cup \{|a : a \in A\}$. The $\odot$ operation is defined as follows:

$$
\begin{array}{rl}
| \odot | = | & \\
| \odot y = |y & | \odot |y = |y \\
x \odot | = | & |x \odot | = | \\
x \odot y = x \otimes y & x \odot |y = |y \\
|x \odot y = |(x \otimes y) & |x \odot |y = |y
\end{array}
$$

To prove that this is a semigroup, all we need is the associativity of $\odot$, namely that $a \odot (b \odot c) = (a \odot b) \odot c$. We can show this by case analysis:

- if $c = |$ or $c = |z$, then both expressions evaluate to $c$. Otherwise, let $c = z$.
- if $b = |$, both expressions evaluate to $|z$.
- if $b = |y$, then both expressions evaluate to $|(y \otimes z)$. Otherwise, let $b = y$.
- if $a = |$, both expressions are $|(y \otimes z)$.
- finally, if $a = x$ or $a = |x$, this boils down to the associativity of $\otimes$.

Assume for simplicity that the first character is a separator (we can insert it explicitly). It is easy to see that this semigroup implements the segmented prefix problem, and generates all correct results in the "barred version" (i.e. if a result should be $x$, this algorithm generates $|x$). This is because a separation character destroys all previous knowledge, and starts a regular computation in the new segment, which goes on until the next separator.

The last observation is that $\odot$ can be implemented using $O(1)$ overhead to the $\otimes$ operation. To represent an element of the new semigroup, we only need $O(1)$ bits in addition to the representation of the input values. The definition of $\odot$ can be handled with $O(1)$ logic for the special cases, and one realization of the $\otimes$ operation.

---

**Problem 2-2.** Show how to add 2 N-bit numbers on a $\sqrt{N} \times \sqrt{N}$ mesh in $O(\sqrt{N})$ steps. How long does it take on an $N^{1/3} \times N^{1/3}$ x $N^{1/3}$ mesh? Explain your reasoning. (*Hint:* This problem has a very short solution.)

**Solution:** We may assume that the bits of the two input numbers and of the output are distributed over the mesh in a regular way (e.g., row-major order). In other words, every processor will hold the corresponding bits of the two inputs, and the output. This effect is easy to achieve, even if the inputs arrive $\sqrt{n}$ bits at a time on one edge, and the result must be output in a similar fashion (simply propagate the data to the correct positions).

The computation is basically identical to the $O(\lg n)$-depth solution presented in class, except that we now use a tree with a branching factor of $\sqrt{n}$ for a 2D mesh, and $n^{1/3}$ for a 3D mesh. The depth of the tree is now constant, but the dominating term comes from propagating the neccessary data among $\sqrt{n}$ siblings. From a 2D-mesh perspective, the algorithm has the following phases:

1. each processor adds the two bits it has, and generates the corresponding $kpg$ value.
2. the first column of processors outputs its $kpg$ values to the second column. Processors of the second column compose this with their own value, and output the results to the third column. This continues until the composition of the $kpg$ values in each row is known by the processor at the end of the row.
3. the processors in the rightmost column implement a compose-and-propagate stage. Now the processor at the end of each row knows the composition of all rows before it.
4. these values are passed along from processors in the rightmost column to the entire row. Each processor finds its corresponding prefix sum by composing the value passed to it with the prefix sum of its row, known from phase 2.

For a 3D-mesh, the computation goes along the same lines (find prefix sums for lines, then planes, then the cube, and then propagate values in reverse). This achieves $O(n^{1/3})$ time.

---

**Problem 2-3.** * Prove that the bisection width of the $\sqrt{N} \times \sqrt{N}$ mesh is at least $\sqrt{N}$.

**Solution:** So, we color $n/2$ nodes in red, and $n/2$ nodes in blue, and we must prove that at least $\sqrt{n}$ edges have ends of different colors. Define a "cross" to mean the union of two paths of the same color, one from the top margin to the bottom margin, and another from left to right. It's easy to show that we can't simultaneously have red and blue crosses. Assume by symmetry that no red cross exists.

Now analyze each red connected component. Assume our component contains $a$ nodes. If we project this component of the horizontal and vertical axes, we get two sets of consecutive integers (because the component is connected). Call the sizes of these sets the width $w$ and, respectively, the height $h$. Clearly, $w \cdot h \geq a$. We now distinguish two cases:

- the component contains a path from the top margin to the bottom margin, or from the left to the right (both cannot happen, since that would be a cross). Assume, by symmetry, that the component connects the top to the bottom. Then, either the component contain no node on the left margin, or no node on the right margin. Assume, by symmetry, that the component does not touch the right margin. Then, consider the rightmost node on every row, from this component. There are $\sqrt{n}$ such nodes, and they all have blue neighbors to the right, so the number of edges that have ends of different colors is at least $\sqrt{n}$. So the problem is solved, even without analyzing other components.
- the component does not connect any two opposite sides. Assume, by symmetry, that the component does not contain any node that is on the right or bottom margins. Then, considering the rightmost nodes on every row, and the bottommost nodes on every column, we get that the number of edges with an end in this component and another end in a blue node is at least $h + w$. Since $w \geq \frac{a}{h} \Rightarrow h + w \geq h + \frac{a}{h} = \sqrt{a}\left(\frac{h}{\sqrt{a}} + \frac{\sqrt{a}}{h}\right) \geq 2\sqrt{a}$.

Now, assume that we are in the second case for all components (otherwise, the problem is solved). Then, we know that $\sum a_i = n/2$. On the other hand, the contribution of each component towards the number of bad edges is at least $2\sqrt{a_i}$. It is easily seen that $\sum \sqrt{a_i} \geq \sqrt{n/2}$ (just raise both sides to the square, and you get the previous relation plus some positive terms on the lhs), so the number of edges is at least $\sqrt{2n}$ (this is a stronger lower bound, but only holds if no component connects two opposite margins).