CSE341T/CSE549T **09/22/2014**

More Examples for Using Scan (including Quicksort)

Lecture 8

1 Fibonacci Numbers

With a carefully chosen matrix, we can use `scan` to compute the Fibonnaci numbers. In the extremely unlikely event that you've forgotten, the Fibbonacci numbers are defined as follows:
Definition: The Fibbonacci numbers are an integer sequence given by the following recurrence[1]

- $F_{-1} = 1$
- $F_0 = 0$
- $F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}$

We make the following claim about this definition, which we will prove by induction:
Claim:

For all natural numbers n,

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$$

Proof: We'll prove this by induction on n.
Base Case: $n = 0$

Any $n \times n$ matrix to the zero power is the $n \times n$ identity matrix, so

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} F_1 & F_0 \\ F_0 & F_{-1} \end{pmatrix}$$

which is exactly as desired.

Inductive Case:

[1] It is slightly contrived, but harmless, to define the -1^{st} element of the Fibbonacci sequence. The other base cases are such that the recursive case will never use it, so this could be any constant and produce the same sequence of integers. This one happens to make the proof work, though.

Assume that

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$$

We want to show that

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n+1} = \begin{pmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{pmatrix}$$

It suffices to show that

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{pmatrix}$$

Recall matrix multiplication, specifically in the case of taking the product of two 2×2 matrices:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{pmatrix}$$

Therefore,

$$\begin{aligned} \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} &= \begin{pmatrix} F_{n+1} + F_n & F_{n+1} \\ F_n + F_{n-1} & F_n \end{pmatrix} \\ &= \begin{pmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{pmatrix} \\ &= \begin{pmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{pmatrix} \end{aligned}$$

This is exactly as desired and concludes the proof.

The above proof means that we can compute the first Fibbonacci numbers by applying scan to a matrix multiplication function — note that it is an associative operator, so we can apply scan. The work is $O(n)$ and span is $O(\lg n)$.

However, if we only care about the nth fibonacci number, and not all of the first n fibonacci numbers, then we can simply use repeated squaring to compute it in work and span $O(\lg n)$.

2 Stock Market Problem

We can also use `scan` to solve the Stock Market problem from homework. Here's the idea:

- We do a min-`scan`. That is given a sequence S, say $M[0..n] \leftarrow$ min-scan(S). Now $M[i]$ is the minimum of the prices until day i.

- We then calculate $S[i] - M[i-1]$ for all i to get the best day to buy if we wanted to sell on day i.
- We then can just do a max-reduce on this new sequence to get the final answer.

3 Quicksort

Quicksort is another recursive sorting algorithm. It is a randomized algorithm, however. The basic algorithm picks a pivot, partitions the array around the pivot and then sorts the two parts recursively.

```
QuickSort(A, p, q)
1  if p = q
2     then return
3  r ← PARTITION(A, p, q)
4  spawn QUICKSORT(A, p, r − 1)
5  QUICKSORT(A, r + 1, q)
6  sync
7  return
```

The sequential partitioning algorithm simply picks a random element in the sequence and places all the elements smaller than the pivot to the left of the pivot and all the elements larger than the pivot to the right of the pivot.

If we use sequential partition algorithm, then the work recurrence is $W(n) = W(k) + W(n - k - 1) + \Theta(n)$ where k is the rank of the partition element. The worst case work is when we always pick $k = 0$ or $k = n - 1$. In this case, the work is $\Theta(n^2)$. The span recurrence is $S(n) = \max\{S(k), S(n-k-1)\} + \Theta(n)$, which is also solves to $\Theta(n^2)$ in the worst case.

Since it is a randomized algorithm, we analyze the expected work and expected span. We analyzed the expected work in CSE241, and it was $\Theta(n \lg n)$; we won't repeat the analysis here. The span recurrence is $S(n) = \max\{S(k), S(n-k-1)\} + \Theta(n)$. Without even solving it, we can see that $E[S(n)] = \Omega(n)$. Actually, this is the same recurrence time recurrence for the randomized select algorithm that you saw in CSE241 also. Therefore, without solving it, let me just assert that it the $E[S(n)] = \Theta(n)$. Therefore, the parallelism is $\Theta(\lg n)$ which is not very high. Again, as in merge sort, we must parallelize the partition algorithm.

So what is the problem with parallelizing partition? We can easily compare all elements with the pivot in parallel. However, we don't know where to put them in the partitioned array.

1. Pick a random pivot at index p.

2. Create an array B of booleans, s.t., $B[i] = 1$ iff $A[i] \leq A[p]$.

3. Create array C, which is invertion of B. That is $C[i] = 1$ if $B[i] = 0$ and vice versa.

4. Do a parallel prefix-sums of B and C. Add $B[n]$ to all elements of C.

5. Now note that $B[i+1] \neq B[i]$, then $A[i] \leq p$, and in this case, $B[i+1]$ represents the location $A[i]$ should go in the partitioned array. Similarly, for C and elements greater than p.

6. We can use this fact to properly partition the array. $P[B[i+1]] = A[i]$ if $B[i+1] > B[i]$ and $P[C[i+1]] = A[i]$ if $C[i+1] > C[i]$.

The total work of this partition operation is $O(n)$ and the span is $O(\lg n)$ (assuming that we use the corresponding Scan algorithm). Therefore, using this method, the work of quicksort is still $O(n \lg n)$. Lets try to analyze the span of quicksort. The recurrence is $S(n) = \max\{S(k), S(n-k-1)\} + \Theta(\lg n)$.

As in 241, define an indicator random variable X_k as 1 if kth rank element is the pivot when partitioning the array. Therefore, the recurrence is

$$S(n) = \Theta(\lg n) + \sum_{k=0}^{n-1} X_k \max\{S(k) + S(n-k-1)\}$$

assuming that scan has a span of $O(\lg n)$.

We can now take expectation

$$\begin{aligned}
E[S(n) &= \Theta(\lg n) + E[\sum_{k=0}^{n-1} X_k \max\{S(k) + S(n-k-1)\}] \\
&= \Theta(\lg n) + \sum_{k=0}^{n-1} E[X_k \max\{S(k) + S(n-k-1)\}] \\
&= \Theta(\lg n) + \sum_{k=0}^{n-1} E[X_k] E[\max\{S(k) + S(n-k-1)\}] \\
&\leq \Theta(\lg n) + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[S(k)]
\end{aligned}$$

Now you can guess the solution of this recurrence is $O(\lg^2 n)$ and try to prove it using induction. Say $E[S(n)] = c_1 \lg^2 n + c_0$.

Base case: When $n = 1$, it trivially works as long as c_0 is large enough.

Inductive step: Say that it is true for all $k \leq n$. We can say

$$\begin{aligned}
E[S(n)] &\leq k \lg n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[S(k)] \\
&\leq k \lg n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} (c_1 \lg^2 k + c_0) \\
&\leq k \lg n + \frac{2c_0}{n}(n - 1 - (n/2 - 1) + 1) + c_1 \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} \lg^2 k \\
&\leq k \lg n + c_0 + 2c_0/n + c_1 \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} \lg^2 k
\end{aligned}$$

Now we must compute $\sum_{k=\lfloor n/2 \rfloor}^{n-1} \lg^2 k$. There are various ways of doing this, but we can approximate using an integral and get something like $n/2 \lg^2 n - n \lg n + n$. Substituting it back,

$$\begin{aligned}
E[S(n)] &\leq k \lg n + c_0 + 2c_0/n + c_1 \frac{2}{n}(\frac{n}{2} \lg^2 n - n \lg n + n) \\
&\leq k \lg n + c_0 + 2c_0/n + c_1 \lg^2 n - 2c_1 \lg n + 2c_1 \\
&\leq c_1 \lg^2 n + c_0 - (2c_1 \lg n - k \lg n - 2c_1 - 2c_0/n)
\end{aligned}$$

The residual is larger than 0 as long as we make c_1 large enough.