1 Parallel Partition Using Scan

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 inversion 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 expected work of quicksort is still \( O(n \lg n) \). Lets try to analyze the span of quicksort.

2 MCSS Problem

We looked at the maximum contiguous subsequence sum problem last week; given a sequence \( S \), we wanted to find a contiguous subsequence that had the largest sum.

Lets say that we (for the heck of it) do a sum-scan of the sequence and store the result in array \( X \). What is the meaning of \( X[j] \)? It is the sum of the subsequence from \( S[1] \) to \( S[j] \). What if we wanted to calculate the sum of subsequence from \( i \) to \( j \) (both inclusive) — we can simply calculate \( X[j] - X[i-1] \). Can we use this to solve a simpler problem: What if you wanted to find out the MCSS that ended at a particular index \( j \)? Can I write down this solution in terms of just \( X \)?
Well this is

\[ R_j = \max_{i=1}^{j} \sum_{k=i}^{j} S_k = \max_{i=1}^{j}(X[j] - X[i-1]) = X_j - \min_{i=0}^{j-1} X[i] \]

What is the last term? It is just the minimum value of \( X \) up to \( j \) (exclusive). Now we want to calculate it for all \( j \), so we can use a scan (min) operation. Once we have \( R_j \), we can calculate the minimum \( R_j \) using reduce. Let’s write down the pseudocode:

1. \( X \leftarrow \text{sum-scan}(S) \)
2. \( Y \leftarrow \text{min-scan}(X) \)
3. \( \text{parallel_for all } i \)
4. \( \quad \text{do } R[i] \leftarrow X[i] - Y[i-1] \)
5. \( \text{return max-reduce}(R) \)

The work of each of the steps (two scans, a parallel-for loop and reduce) is \( O(n) \) and the span is \( O(\log n) \). We therefore have a routine that is even better than any of our divide and conquer routines.

### 3 Remove Duplicates

You are given a sorted array \( A \) which contains duplicate elements. How can we create an array \( B \) which contains all the elements of \( A \), but without the duplicates?

We first simply create a boolean array \( C \) such that \( C[i] = 1 \) if \( A[i] \neq A[i-1] \) and 0 otherwise. We can now do a sum-scan on \( C \) — if \( D[i] = k \), then the number of distinct elements in \( A \) which are smaller than or equal \( A[i] \) is \( k \). Therefore, \( A[i] \) should go in the \( k \)th position in \( B \).

1. \( \text{parallel_for } i \leftarrow 1 \) to \( n \)
2. \( \quad \text{do if } A[i] \neq A[i-1] \)
3. \( \quad \quad \text{then } C[i] \leftarrow 1 \)
4. \( \quad \quad \text{else } C[i] \leftarrow 0 \)
5. \( D \leftarrow \text{sum-scan}(C) \)
6. \( \text{parallel_for } i \leftarrow 1 \) to \( n \)
7. \( \quad \text{do if } A[i] \neq A[i-1] \)
8. \( \quad \quad \text{then } B[D[i]] \leftarrow A[i] \)

The work and the span of this algorithm are the same as that of the Scan algorithm.
4 Fibonacci Numbers

With a carefully chosen matrix, we can use scan to compute the Fibonacci numbers. In the extremely unlikely event that you’ve forgotten, the Fibonacci numbers are defined as follows:

**Definition:** The Fibonacci 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:**

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

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

\(^1\)It is slightly contrived, but harmless, to define the \(-1^{st}\) element of the Fibonacci 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.
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 $2 \times 2$ 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,
\[
\begin{pmatrix}
F_{n+1} & F_n \\
F_n & F_{n-1}
\end{pmatrix}
\cdot
\begin{pmatrix}
1 & 1 \\
1 & 0
\end{pmatrix}
= 
\begin{pmatrix}
F_{n+1} + F_n & F_{n+1} \\
F_n + F_{n-1} & F_n
\end{pmatrix}
\]
\[
= \begin{pmatrix}
F_{n+2} & F_{n+1} \\
F_{n+1} & F_n
\end{pmatrix}
\]
\[
= \begin{pmatrix}
F_{n+2} & F_{n+1} \\
F_{n+1} & F_n
\end{pmatrix}
\]

This is exactly as desired and concludes the proof.

The above proof means that we can compute the first Fibonacci 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 $n$th 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 $O(\lg n)$. 