27.3-5 ★
Give a multithreaded version of \textsc{Randomized-Select} on page 216. Make your implementation as parallel as possible. Analyze your algorithm. \textit{(Hint: Use the partitioning algorithm from Exercise 27.3-3.)}

27.3-6 ★
Show how to multithread \textsc{Select} from Section 9.3. Make your implementation as parallel as possible. Analyze your algorithm.

Problems

27-1 Implementing parallel loops using nested parallelism
Consider the following multithreaded algorithm for performing pairwise addition on $n$-element arrays $A[1..n]$ and $B[1..n]$, storing the sums in $C[1..n]$:

\begin{verbatim}
SUM-ARRAYS\(A, B, C\)
1 parallel for \(i = 1\) to \(A.length\)
2 \(C[i] = A[i] + B[i]\)
\end{verbatim}

\(a.\) Rewrite the parallel loop in \textsc{Sum-Arrays} using nested parallelism (\texttt{spawn} and \texttt{sync}) in the manner of \textsc{Mat-Vec-Main-Loop}. Analyze the parallelism of your implementation.

Consider the following alternative implementation of the parallel loop, which contains a value \textit{grain-size} to be specified:

\begin{verbatim}
SUM-ARRAYS'\(A, B, C\)
1 \(n = A.length\)
2 \(\text{grain-size} = ?\) \(\text{\hspace{1cm}}\) // to be determined
3 \(r = \lceil n/\text{grain-size} \rceil\)
4 for \(k = 0\) to \(r - 1\)
5 \hspace{1cm} \text{spawn} ADD-SUBARRAY\(A, B, C, k \cdot \text{grain-size} + 1, \quad \min((k + 1) \cdot \text{grain-size}, n)\))
6 \hspace{1cm} \text{sync}

ADD-SUBARRAY\(A, B, C, i, j\)
1 for \(k = i\) to \(j\)
2 \(C[k] = A[k] + B[k]\)
\end{verbatim}
**27-2 Saving temporary space in matrix multiplication**

The P-MATRIX-MULTIPLY-RECURSIVE procedure has the disadvantage that it must allocate a temporary matrix $T$ of size $n \times n$, which can adversely affect the constants hidden by the $\Theta$-notation. The P-MATRIX-MULTIPLY-RECURSIVE procedure does have high parallelism, however. For example, ignoring the constants in the $\Theta$-notation, the parallelism for multiplying $1000 \times 1000$ matrices comes to approximately $1000^3/10^2 = 10^7$, since $\lg 1000 \approx 10$. Most parallel computers have far fewer than 10 million processors.

**a.** Describe a recursive multithreaded algorithm that eliminates the need for the temporary matrix $T$ at the cost of increasing the span to $\Theta(n)$. (Hint: Compute $C = C + AB$ following the general strategy of P-MATRIX-MULTIPLY-RECURSIVE, but initialize $C$ in parallel and insert a `sync` in a judiciously chosen location.)

**b.** Give and solve recurrences for the work and span of your implementation.

**c.** Analyze the parallelism of your implementation. Ignoring the constants in the $\Theta$-notation, estimate the parallelism on $1000 \times 1000$ matrices. Compare with the parallelism of P-MATRIX-MULTIPLY-RECURSIVE.

**27-3 Multithreaded matrix algorithms**

**a.** Parallelize the LU-DECOMPOSITION procedure on page 821 by giving pseudocode for a multithreaded version of this algorithm. Make your implementation as parallel as possible, and analyze its work, span, and parallelism.

**b.** Do the same for LUP-DECOMPOSITION on page 824.

**c.** Do the same for LUP-SOLVE on page 817.

**d.** Do the same for a multithreaded algorithm based on equation (28.13) for inverting a symmetric positive-definite matrix.