COMP 633: Parallel Computing

Midterm Practice Questions

1. (PRAM algorithms and OpenMP parallelization) In the string occurrence problem we are given a string \( P \) of length \( m \) and a string \( S \) of length \( n \), with \( m \leq n \), and we want to determine whether \( P \) occurs in \( S \). The KMP (Knuth-Morris-Pratt) algorithm provides an efficient sequential solution for this problem.

All you need to know about the KMP algorithm is that it first performs a preprocessing step that requires \( O(|P|) \) sequential time to construct a deterministic finite automaton (dfa). This dfa can read through a given string \( T \) to determine whether \( P \) occurs in \( T \) in \( O(|T|) \) sequential time. The function \( \text{occursP}(T, \text{dfa}) \) returns true if \( P \) occurs in \( T \) and false otherwise. A sequential solution for the string occurrence problem using KMP follows:

\[
dfa = \text{construct_dfa}(P); \quad // O(|P|) = O(m) \text{ time} \\
\text{result} = \text{occursP}(S, \text{dfa}); \quad // O(|S|) = O(n) \text{ time}
\]

for a total time of \( O(|P| + |S|) = O(m + n) = O(n) \). Make use of the \text{construct_dfa} and \text{occursP} procedures of KMP in the following.

(a) Construct a CRCW parallel algorithm in the Work-Time framework for the string occurrence problem with \( W(n, m) = O(n) \) and \( S(n, m) = O(m) \).

(b) Sketch an OpenMP implementation of this algorithm that maximizes parallelism and minimizes total work by using explicit knowledge of the number of processors \( p \). Explicitly mention the scheduling strategy for parallel loops.

(c) Point out where concurrent reads and writes occur in your Work-Time algorithm and describe how to minimize their performance impact in OpenMP.

2. (OpenMP work sharing) We are given the following loop with independent iterations to be executed in parallel using OpenMP work sharing.

```c
#pragma omp parallel for
for (i = 0; i < 8; i++) {
    perform W[i] units of work
}
```

Suppose \( W[0..7] \) specifies the following amounts of work

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
</table>

For each of the following permute the values in \( W \) so that the time to execute the loop is minimized using 2 processors under the indicated work-sharing schedule. In each case \( \text{chunksize} \) is the default for the schedule. You can assume that a unit of work is large compared to the overhead to acquire a new chunk.
3. **(Cache behavior and and consistency)** Consider a CC_NUMA shared memory machine using LL/SC to implement atomic operations. We are interested in the performance of the (single use) barrier synchronization for $p$ processors using the `fetch-and-add(m,r1)` operation, which atomically adds $r1$ to the variable $m$ in memory and returns the final value of $m$ in a register. The barrier implementation is shown below and uses two shared variables `count` and `release` each with initial value 0.

Barrier:

```c
if (fetch-and-add(count,1) == p) then release := 1 endif
while (release != 1) do skip enddo
```

(a) Using the implementation of `fetch-and-add` based on LL/SC, as described in Lecture CC-NUMA(3) slide 4, describe the worst case number of cache line misses between the time the first processor enters the barrier and the time the last processor leaves the barrier $p$.

(b) Describe a way to decrease this cost.