Theory Exercises

Exercise 1

Consider a simple uniprocessor system with no caches. How does register allocation (applied by the compiler) affect memory consistency? Which language feature of C allows you to enforce sequential consistency for a variable?

Exercise 2

Assume that shared variables x and y happen to be placed in the same memory block (cache line) of a cache-based, bus-based shared memory system. Consider a program executed by 2 processors P_1 and P_2, each executing a loop with n iterations where processor P_1 reads variable x in each iteration of its loop and processor P_2 concurrently writes y in each iteration. There is no synchronization between loop iterations or between reads and writes, i.e., the read and write accesses will be somehow interleaved over time.

(a) Using the M(E)SI write-invalidate coherence protocol, how many invalidation requests are to be sent if sequential consistency is to be enforced?

(b) Show how thrashing can be avoided by using a relaxed memory consistency model.

Exercise 3

Consider a superscalar RISC processor running at 2 GHz. Assume that the average CPI (clock cycles per instruction) is 1. Assume that 15% of all instructions are stores, and that each store writes 8 bytes of data. How many processors will a 4-GB/s bus be able to support without becoming saturated?

Exercise 4

Give high-level CREW and EREW PRAM algorithms for copying the value of memory location $M[1]$ to memory locations $M[2],...,M[n+1]$. Analyze their parallel time, work and cost with $p \leq n$ processors. What is the asymptotic speedup over a straightforward sequential implementation?

Exercise 5

Write a simple program for a CRCW PRAM with n processors that calculates the logical OR of all elements in a shared array of n booleans in time $O(1)$.

Hint: The program has only 2 statements...

Exercise 6

On a RAM the maximum element in an array of n real numbers can be found in $O(n)$ time. We assume for simplicity that all n elements are pairwise different.

(a) Give an EREW PRAM algorithm that finds the maximum element in time $\Theta(\log n)$. How many processors do you need at least to achieve this time bound?

What is the work and the cost of this algorithm? Is this algorithm cost-effective with n processors? With $n/\log n$ processors?

(b) Give an algorithm for a Common CRCW PRAM with n^2 processors that computes the maximum element in constant time.

(*Hint:* In a Common CRCW PRAM concurrent write to the same memory location in the same clock cycle is permitted but only if all writing processors write the *same* value. — Arrange the processors conceptually as a $n \times n$ grid to compute all n^2 comparisons of pairs of elements simultaneously. An element that is smaller in such a comparison cannot be the maximum. Use the concurrent write feature to update the entries in an auxiliary boolean array m of size n appropriately, such that finally holds $m[i] = 1$ iff array element i is the maximum element. Given m, the maximum location i can be determined in parallel using n processors.)

What is the work and the cost of this algorithm? Is this algorithm work-optimal? Is it cost-optimal?

Further reading on the maximum problem:

Any CREW PRAM algorithm for the maximum of n elements takes $\Omega(\log n)$ time.
See [Cook/Dwork/Reischuk SIAM J. Comput. 1986]

There exist CRCW PRAM algorithms for n processors that take $O(\log\log n)$ time.
See [Valiant SIAM J. Comput. 1975, Shiloach/Vishkin J. Algorithms 1981]

Exercise 7

Show that the cost of a cost-optimal parallel algorithm A asymptotically grows equally fast as the work of the optimal sequential algorithm S, i.e., $c_A(n) = \Theta(t_S(n))$.

Exercise 8

Give a $O(\log n)$ time algorithm for computing parallel prefix sums on a parallel list.

(Hint: Use the pointer doubling technique.)

```
Algorithm FFT ( array x[0..n−1] )
        returns array y[0..n−1]
{
    if n = 2 then
        y[0] ← x[0] + x[1];  y[1] ← x[0] − x[1];
    else
        allocate temporary arrays u, v, r, s
            of n/2 elements each;
        for l in { 0.. n/2-1} do
            u[l] ← x[l] + x[l + n/2];
            v[l] ← w^l * (x[l] − x[l + n/2]);
        od
        r ← FFT ( u[0..n/2 − 1] );
        s ← FFT ( v[0..n/2 − 1] );
        for i in { 0.. n-1} do
            if i is even then y[i] ← r[i/2] fi
            if i is odd then y[i] ← s[(i − 1)/2] fi
        od
    fi
    return y[0..n−1]
}
```

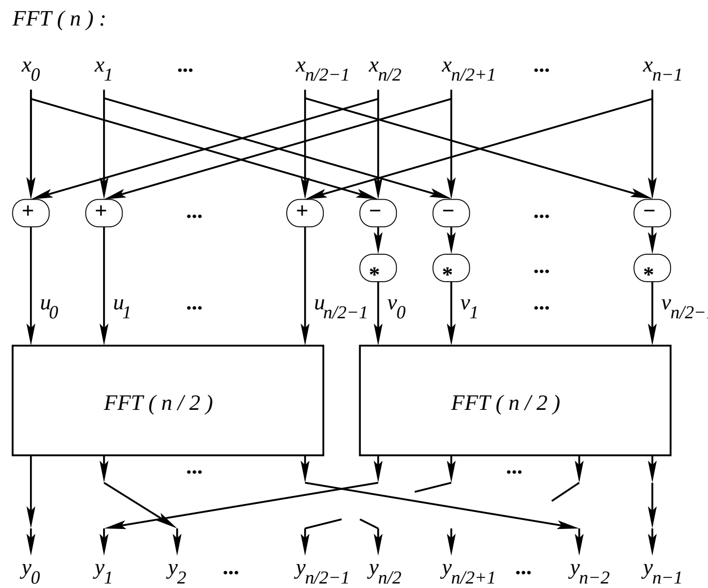

Figure 1: The sequential FFT algorithm.

Exercise 9 (from the main exam 2011)

The *Fast-Fourier-Transform* (FFT) is a (sequential) algorithm for computing the Discrete Fourier Transform of an array x of n elements (usually, complex numbers) that might represent sampled input signal values, using a special complex number w that is a nth root of unit, i.e., $w^n = 1$. The result y is again an array of n elements, now representing amplitude coefficients in the frequency domain for the input signal x. Assume for simplicity that n is a power of 2. A single complex addition, subtraction, multiplication and copy operation each take constant time.

Figure 1 shows the pseudocode of a recursive formulation of the FFT algorithm and gives a graphical illustration of the data flow in the algorithm.

1. Which fundamental algorithmic design pattern is used in the FFT algorithm?
2. Identify which calculations could be executed in parallel, and sketch a parallel FFT algorithm for n processors in pseudocode (shared memory).
3. Analyze your parallel FFT algorithm for its *parallel execution time*, *parallel work* and *parallel cost* (each as a function in n, using big-O notation) for a problem size n using n processors. *(A solid derivation of the formulas is expected.)*
4. Is your parallel FFT algorithm *work-optimal*? Justify your answer (formal argument).
5. How would you adapt the algorithm for $p < n$ processors cost-effectively?