CpSc 418

Final Exam Review

The questions in this review use convolution as an example. I chose convolution because convolution was covered in two lectures; convolution has an big role in signal processing (e.g. multimedia), machine learning, communication systems (e.g. mobile computing), and many other areas; and we haven't had a HW question about it yet. I'm using convolution to cover a wide range of topics from the term such as: speed-up, work-span, Amdahl's Law, Brent's Law, communication costs (e.g. λ), network bandwidth constraints, memory bandwidth constraints (e.g. CGMA), parallel architectures, SIMD and message-passing paradigms, etc.

The actual final will cover the same range of topics – not necessarily the exact same choices as this assignment, but the sample here is representative. The final exam will not base all of the questions on a single algorithm. We've covered quite a few algorithms: reduce, scan, map (i.e. "embarrassingly parallel" such as the Monte Carlo simulation problem in HW2, the recurrences in HW5), sorting, matrix multiplication, and convolution.

1. **Convolution and multiplication.**
   Let $x_0, x_1, \ldots, x_n$ be a sequence of numbers. Likewise, let $y_0, y_1, \ldots, y_m$ be another sequence of numbers. We write $\mathsf{value}(x, b)$ to denote the "value" of the sequence x as a base-b number:

   $$\mathsf{value}(x, b) \quad = \quad \sum_{i=0}^{n} x_i b^i$$

   Let $z = x\#y$ denote the convolution of x and y:

   $$z_i \quad = \quad \sum_{j=0}^{i} x_j y_{i-j}$$

   where we treat x_j as 0 if $j > n$ and $y_{i-j} = 0$ if $i - j > m$. Show that

   $$\mathsf{value}(x\#y, b) \quad = \quad \mathsf{value}(x, b) \cdot \mathsf{value}(y, b)$$

   where $\cdot$ denotes ordinary, scalar multiplication.

   Example: let $b = 10$ so we get the familiar, decimal representation of numbers. Let `X = [1,2,3]`. `value(X, 10) = 321` – the list `X` is least-significant digit first. Let `Y = [4,7,9,2]`. `value(Y, 10) = 2974`. From the definition of convolution, `conv(X, Y) -> [4,15,35,41,31,6]`, and
   `value(conv(X,Y),10) = value([4,15,35,41,31,6],10) = 954654 = 321*2974`.
   Code for `value` and `conv` is provided in `hw6.erl`.

   Note: this connection between convolution and multiplication is a key part of many "fast" algorithms for multiplying large numbers, i.e. numbers with thousands of digits. A fast algorithm for convolution can be used to obtain a fast algorithm for multiplication.

2. **Systolic Convolution**
   Consider the the systolic algorithm for convolution shown in class. In this algorithm we are given two vectors $A = (a_1, ..., a_n)$ and $B = (b_1, ..., b_n)$ of size n and an array of cells from C_{2n} down to C_1, going from left to right.

   The algorithm proceeded as follows. 1. First we send vector A into the cells from the left where a delay, denoted by $\bullet$, was added between consecutive elements of A (i.e., a_1, $\bullet$, a_2, $\bullet$, ...) whereby on time step 1, a_1 enters cell C_{2n} and proceeds to the right on each time step. Due to the delay a_2 enters cell C_{2n} at time step 3, etc..

   2. Similarly, we reverse the B vector and add a delay at the start of the B vector as well as between every consecutive element (i.e., $\bullet$, b_n, $\bullet$, b_{n-1}, ...) whereby on time step 2, b_n enters cell C_1 and proceeds to the left on each time step. Due to the delay b_{n-1} enters on cell C_1 at time step 3, etc..

3. Finally, each cell C_k is initialized to 0 and whenever a_i and b_j meet in cell C_k we perform the appropriate convolution operation and accumulate it at C_k (e.g. $c_k = c_k + a_i * b_j$).

Prove that this algorithm correctly computes the convolution where cell C_k computes the sum of a_i and b_j such that $i + j = k - 1$. What is the total number of steps to perform the computation? (Hint: derive formulas giving the location of a_i and b_j at step t.)

3. **GPU Convolution**
Often, we want to compute the convolution of X and Y where X is large (e.g. thousands or millions of elements) and Y is of moderate size (e.g. 10 to 100 elements). To compute such a convolution on a GPU, we can divide X across blocks. Let m be the number of elements in Y If we write $Z = X\#Y$, we note that Z_i depends on X_{i+1-m} through X_i and all of Y. Thus, a block that holds n values of X can compute $n + 1 - m$ values of Z. This also means that blocks will segments of X that overlap slightly on their ends. We can keep this overlap relatively small if the number of elements of X that a block can store in shared memory is much larger than the number of elements of Y.

Let's use all 48Kbytes of the shared memory for a block to hold X. If each element of X is a four-byte float, then a block can hold 48K/4 = 12K = 12*1024 = 12288 elements of X. If Y has length m, then the block can compute the convolution for $12289 - m$ elements of x.

(a) How many floating point operations are required to compute *one* element of `Z`?
(b) Can the computation of `Z` take advantage of fused multiply-add instructions?
(c) How many global memory reads does a block perform to copy the 12288 elements of X from global memory into shared memory?
(d) Can these global memory accesses be coalesced?
(e) How many global memory writes are required to store the values of Z computed by this block?
(f) Can these global memory accesses be coalesced?
(g) What is the CGMA for this computation? Hint: it depends on m.
(h) How large must m be to achieve a CGMA of 80?

4. **Message Passing Convolution**
Consider computing $Z = X\#Y$ on a message passing computer with P processors where X has N elements, Y has M elements. We will assume that $N >> M$, each processor holds P/N elements of X, and all M elements of Y. To compute the convolution, processor p_i (for $1 \leq i \leq P$):

- sends the last $m - 1$ of its segment of X to processor p_{i+1} (if $i < P$),
- receives $m - 1$ elements from processor p_{i-1} (if $i > 1$)
- computes the the convolution of its N/P elements of Z. I.e. p_i has elements $\frac{N}{P}(i-1)\cdots\frac{N}{P}i - 1$ of X and computes elements $\frac{N}{P}(i-1)\cdots\frac{N}{P}i - 1$ of Z.

Assume that each processor can perform a multipy-and-add in one unit of time. Assume that sending and receiving a message of $M - 1$ elements takes time $\lambda + M - 1$ (total for the send and the receive). You can assume that λ is big enough that we don't care about the difference between $\lambda + M - 1$ and $\lambda + M$.

(a) How much time does it take each processor p_i to send a message to p_{i+1} and receive a message from $pi - 1$? Don't worry about the end cases with $i = 1$ or $i = P$.
(b) How much time does it take processor p_i to compute its values for Z after receiving the message from p_{i-1}?
(c) What is the speed-up? Write your answer as a function of N, M, P, and λ. Give values for the speed-up when

- $N = 1000000$, $M = 50$, $P = 100$, and $\lambda = 10000$.
- $N = 1000000$, $M = 50$, $P = 1000$, and $\lambda = 10000$.
- $N = 1000000$, $M = 10$, $P = 1000$, and $\lambda = 10000$.

(d) What is the parallel-efficiency? Write your answer as a function of N, M, P, and λ. Give values for the speed-up for the same values listed above.

5. **A few more questions**

(a) Consider implementing the convolution described in the previous problem on a mesh of 64×64 processors. Is the cross-section bandwidth a bottleneck for this computation or can we just consider the bandwidth for links between neighbouring processors? Justify your answer.

(b) What is the work for a convolving a vector of length N with a vector of length M? Justify your answer.

(c) What is the span for a convolving a vector of length N with a vector of length M? Justify your answer.

(d) Use Brent's Lemma to bound the speed-up for convolution with

- $N = 1000000$, $M = 50$, $P = 100$, and $\lambda = 1000$.
- $N = 1000000$, $M = 50$, $P = 1000$, and $\lambda = 1000$.
- $N = 1000000$, $M = 10$, $P = 500$, and $\lambda = 1000$.