CpSc 418

Final Exam

December 12, 2012

Graded out of **100 points** (even though 100 to 105 are possible depending on which problems you choose)

Time for the exam: 150 minutes.

Open book: anything printed on paper may be brought to the exam and used during the exam. This includes the textbook, other books, printed copies of the lecture slides, lecture notes, homework and solutions, and any other material that a student chooses to bring.

Calculators are allowed: no restriction on programmability or graphing. There are a few simple calculations needed in the exam, a calculator will be handy, but the fancy features will not make a difference.

No communication devices: That's right. You may not use your cell phone for voice, text, web-surfing, or any other purpose. Likewise, the use of computers, iPods, etc. is not permitted during the exam.

Do problems 1 and 2 and any three of 3-6.

Good luck!

CpSc 418 **Final** December 12, 2012

100 points

1. **DLS talk (5 points)**
Judea Pearl's talk: "The Mathematics of Cause and Effect" made frequent use of the term "counterfactual." What word or phrase below best describes "counterfactual" as Pearl used it in his talk:
   (a) a false statement;
   (b) an ancient story or account such as the Garden of Eden or the destruction of Sodom;
   (c) a paradox;
   (d) a "what if?" scenario;
   (e) a rebuttal to an argument.

2. **Short definitions (20 points).** Give a **short** definition for each term below. Give a brief (one or two sentences) definition **and** a simple example (one or two sentences).
   (a) deadlock
   (b) livelock
   (c) the zero-one principle
   (d) termination detection

3. **Reduce (25 points)** Given a list of numbers, report the length of the longest non-descending sequence, in other words the length of the longest sequence of the form $x_i, x_{i+1}, \ldots x_{i+k}$ such that $x_i \leq x_{i+1} \leq \cdots \leq x_{i+k}$. For example,

```
longest_ascending([1, 2, 3, 2, 4, 6, 7,-2, -3, -1, 0, 8, 13, 21, 16, 23, 21])
```

   is 6.

   (a) (**5 points**) Draw a figure showing how this computation can be performed with four processes using a reduce. Assume that the list is distributed over the processes as shown below:

   Process 0: [1, 2, 3, 2]
   Process 1: [4, 6, 7, -2, -3]
   Process 2: [-1, 0, 8, 13]
   Process 3: [21, 16, 23, 21]

   Your diagram should show what information is passed up the tree. You don't need to formalize the details of the data structure until part b below.

   (b) (**5 points**) Describe the type for the value that you will pass up the tree to do the reduce. For example, "a tuple of three elements where the first element is…", or "an integer", or "a list of …", etc.
   (c) (**5 points**) Sketch a function to use at the leaves of the tree to perform this reduce. You can write it in erlang, C, Java, or any reasonable resemblance of any of those languages.
   (d) (**5 points**) Sketch a function to use to combine values from subtrees. Use the same notation as you chose for part c.
   (e) (**5 points**) Sketch a function to generate the final answer given the result of the combine at the root.

   Each of your functions descriptions should be short.

4. **Peril-L and Pthreads (25 points)** Peril-L has full/empty variables. There isn't a direct equivalent in POSIX threads. Figure 2 shows a one way that full/empty variables could be implemented using POSIX threads.

   (a) (**10 points**) Write the body for function `FE_int_get`.

   Likewise, Peril-L doesn't provide mutexes or condition variables. Show that you can implement a mutex using Peril-L full/empty variables. Figure 1 shows an outline of the code that you should write. Note that I'm not asking you to write the functions to allocate, initialize, or free your mutex structs.

   (b) (**5 points**) Define a Peril-L `struct FE_mutex` for a mutex type that is implemented using one or more full/empty variables. In Peril-L, a variable whose name ends with an apostrophe is a full/empty variable, for example, `int v';`. Your struct can also use regular variables that are, presumably, global because this struct is used for synchronization. Note that Peril-L uses C-syntax plus the Peril-L extensions. Any reasonable approximation to Peril-L's syntax will be accepted.

   (c) (**5 points**) Write the function body for `FE_mutex_lock`. The textbook points out that it is a usage error if a thread attempts to acquire a lock that it already has. POSIX-threads doesn't check for this; so, you don't have to either.

   (d) (**5 points**) Write the function body for `FE_mutex_unlock`. It is an error to attempt to unlock a mutex that is not presently locked. Your code should call `error("bad unlock");` in this situation.

5. (**27 points**) Consider the parallel implementation of dynamic programming for computing editing distance from the November 1 lecture. The input consists of two strings, s_1 and s_2, each of length N. To compute the editing distance between the two strings, the algorithm computes the entries of a $(N+1) \times (N+1)$ tableau, T, where the row and column indices are in $\{0, \ldots, N\}$. Entry $T(i,j)$ is the minimum editing cost to convert the prefix or s_1 of length i into the prefix of s_2 of length j. For example, if $s_1 =$ "`helloworld`", and $s_2 =$ "`hewgold`", then $T(2,6)$ is the minimum editing cost to change `"he"` to `"hew go"`.

   In lecture, we showed that for all $0 \leq i,j \leq N$ the tableau entries $T(0,j)$ and $T(i,0)$ can be computed independently. For $i,j > 0$, the value for $T(i,j)$ can be computed from the values of $T(i-1,j)$, $T(i,j-1)$ and $T(i-1,j-1)$. This observation can be extended to blocks of tableau elements, and led to the parallel algorithm presented in class:

   - Let P be the number of processes. To avoid special cases, assume that N is a multiple of P.
   - The tableau is divided into P^2 blocks of size $(N/P) \times (N/P)$. Let $I_r = \{(r-1)*(N/P)+1, \ldots, r*(N/P)\}$; $J_c = \{(c-1)*(N/P)+1, \ldots, c*(N/P)\}$; and Let $\overline{I}_r = \{(r-1)*(N/P), \ldots, r*(N/P)\}$; Let $B(r,c)$ denote the $(r,c)^{th}$ block of the tableau. In other words, $B(r,c)$ has elements $T(i,j)$ for $i \in I_r$ and $j \in J_c$.
   - For $1 \leq k \leq P$, process k works on the k^{th} column of blocks. The main loop for process k is

```
top_row[$J_k$] = initial values;
for(int r = 0; r < P; r++) {
    if(k > 1)
        receive values for $T(\overline{I}_r, (k-1)*(N/P))$ from process k-1;
    update $B(r,k)$;
    if(k < N)
        send values for $T(\overline{I}_r, k*(N/P))$ to process k+1;
}
```

   The computation completes when process P computes $T(N,N)$.

   In lecture, we modeled the time for this computation assuming that a processor takes time t_{up} to update one cell of the tableau, and that the total for the sender and receiver to send and receive a message of m tableau elements

```
struct FE_mutex {
   // You declare the fields.
};

// FE_mutex_lock(m): acquire mutex m.
void FE_mutex_lock(struct FE_mutex *m) {
   // You write the function body.
}

// FE_mutex_unlock(m): release mutex m.
void FE_mutex_unlock(struct FE_mutex *m) {
   // You write the function body.
}
```

Figure 1: Implementing a mutex using full/empty variables.

```
struct FE_int {
   int value;
   boolean full;
   pthread_mutex_t lock;
   pthread_cond_t cond;
};

// FE_int_set: Set the value of *ef to val.
// If *ef is full, block until it is empty.
void FE_int_set(struct FE_int *ef; int val) {
   pthread_mutex_lock(&(ef->lock));
   while(ef->full)
      pthread_cond_wait(&(ef->cond), &(ef->lock));
   ef->value = val;
   ef->full = TRUE;
   pthread_cond_signal(&(ef->cond));
   pthread_mutex_unlock(&(ef->lock));
}

// FE_int_get: Get the value of *ef to val.
// If *ef is empty, block until it is full.
int FE_int_get(struct FE_int *ef) {
   // You get to write get!
}
```

Figure 2: Implementing full/empty variables using POSIX threads

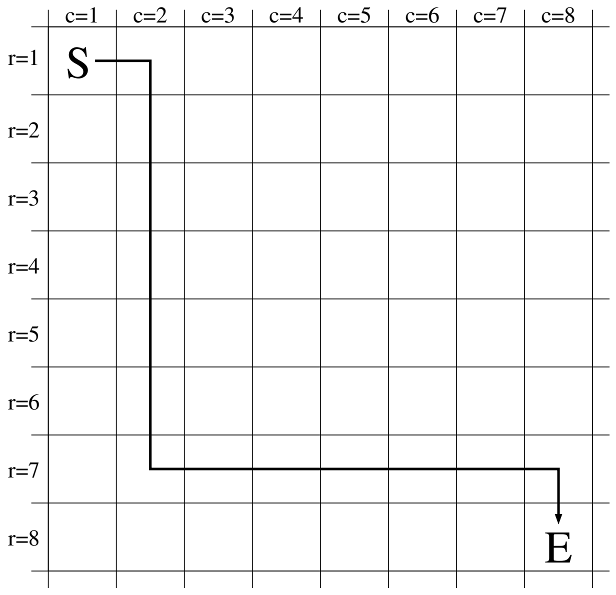


Figure 3: Critical path for parallel editing distance computation

is $t_0 + t_1 m$. With this model, the time for the sequential computation is $t_{seq} = t_{up}N^2$ and the time for the parallel computation is

$$t_{par} \quad = \quad (2P-1)\left(\frac{N}{P}\right)^2 t_{up} + 2(P-1)\left(t_0 + \frac{N}{P}t_1\right)$$

We did this by identifying a *critical path* for the computation, and determining the time for the critical path. Figure 3 shows such an example of a critical path that leads to this number. The components of the critical path are:

$(2P-1)\left(\frac{N}{P}\right)^2 t_{up}$: There are a total of $2P-1$ block updates on this path, each requiring time $(N/P)^2 t_{up}$.

$2(P-1)(t_0 + \frac{N}{P})t_1$: Updating $B(1,1)$ involves a message send, but no message receive. Likewise, updating $B(P,P)$ involves a receive but a send. Every message has size N/P (or $(N/P)+1$, but I'm assuming N/P is large enough that I can ignore the $+1$). So, this send and receive takes a total of $t_0 + (N/P)t_1$ time. The remaining $2P-3$ block updates each involve send and a receive. The total communication time is $2(P-1)(t_0 + \frac{N}{P}t_1)$.

(a) (**5** points) Assume that $t_{up} = 20$ns, $t_0 = 200$ns and $t_1 = 10ns$ (reasonable values on a shared-memory implementation using POSIX threads or Java threads). Note: 1ns = 10^-9 seconds). What is the speed-up if $N = 1000$ and $P = 16$?

(b) (**5** points) Show that in the limit that $N \to \infty$, the speed-up goes to $P/2$.

(**c**ontinued on next page)

We can divide the tableau into smaller blocks. For any integer $M > 1$, let each block have size $(N/(MP)) \times (N/(MP))$. Now, the entire tableau consists of MP rows of MP columns of these blocks. Assume that N is divisible by MP. The revised algorithm (for processor k) is:

```
for(int m = 0; m < M; m++) {
    c = m*(N/P) + k;
    top_row[$J_c$] = initial values;
    for(int r = 0; r < P; r++) {
        if(c > 1)
            **receive** values for $T(\overline{I}_r, (c-1)*(N/P))$ from process c-1;
        update $B(r,c)$;
        if(c < N)
            **send** values for $T(\overline{I}_r, c*(N/P))$ to process c+1;
    }
}
```

(c) (**4** points) Draw the critical path for this computation when $P = 8$ and $M = 2$. You may use the template from Figure 4 when drawing your answer.

(d) (**5** points) Derive a formula for the parallel execution time as a function of N, P, M, t_0, t_1, and t_{up}.

(e) (**4** points) What is the speed-up with $M = 8$ and the same parameters as for part a?

(f) (**4** points) Show that in the limit as $N \to \infty$, M can be chosen so that the speed-up will converge to P.

6. **Sorting on a Mesh (28 points).**
Let a be an array with of N values, $[a_0, a_2, \ldots, a_N - 1]$. Every operation that I describe in this problem can be implemented using sorting networks. Thus, we can assume that every element of a has a value of 0 or 1.

If $N = N_1 N_2$, then we can arrange the elements of a in a two dimensional array, A, where

$$\begin{array}{rcll} A_{i,j} & = & a_{N_2 * i + j}, & \text{if } i \text{ is even} \\ A_{i,j} & = & a_{N_2 * (i+1) - (j+1)}, & \text{if } i \text{ is odd} \end{array}$$

This scheme arranges a into a "snaking" in A with even indexed rows going left-to-right, and odd-indexed rows going right-to-left.

For simplicity, assume that N_1 is a power of 2 greater than 1. We will say that a row of A is "clean" if all of its elements have the same value, and "dirty" if the row contains at least one 0 and at least one 1.

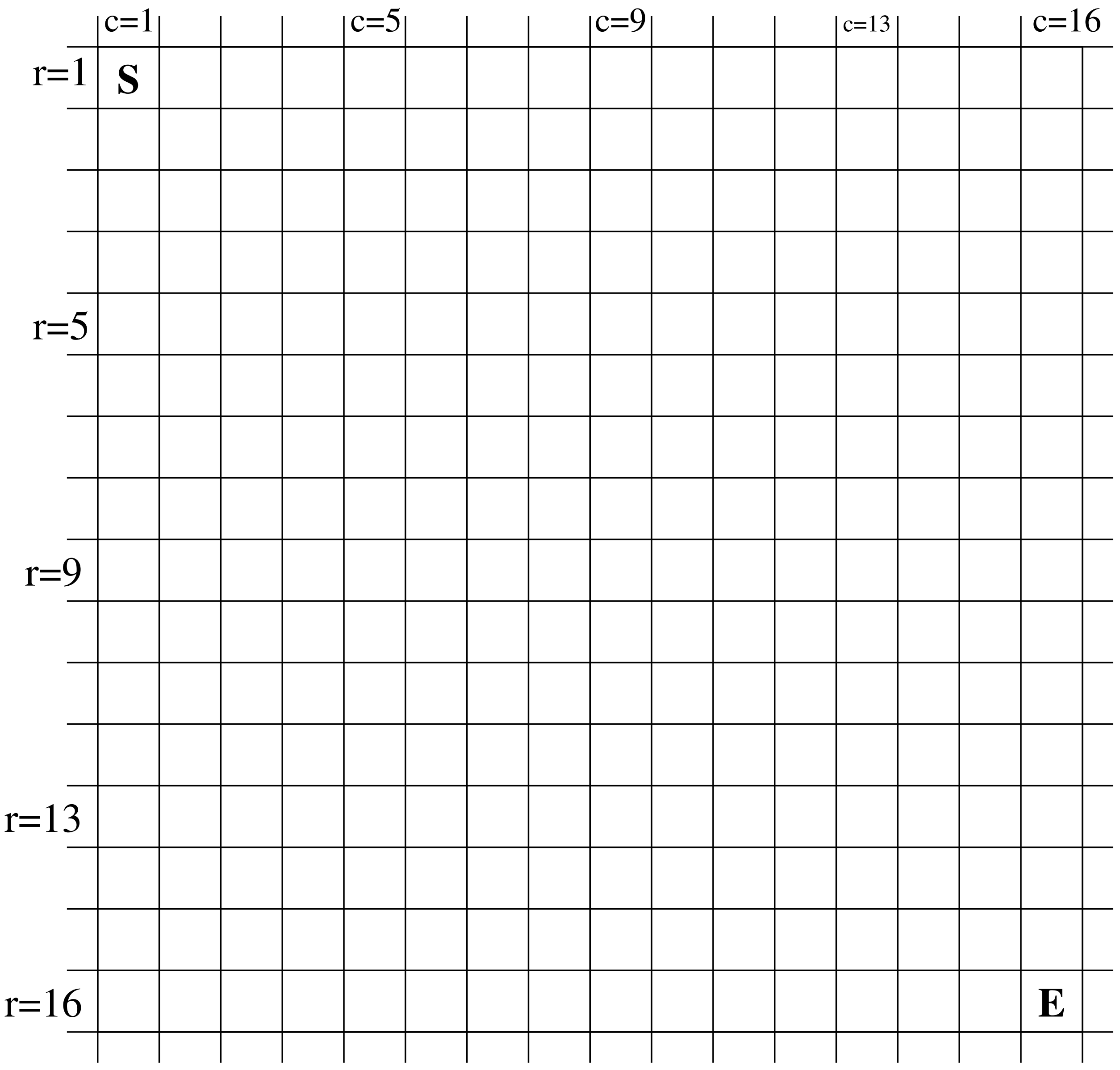
In the first step of our sorting algorithm, we will sort all even-indexed rows to be ascending to the right, and all odd-indexed rows to be ascending to the left, i.e.:

$$\begin{array}{ll} \forall 0 < j < N_2.\ A(i, j-1) \leq A(i,j), & \text{if } i \text{ is even} \\ \forall 0 < j < N_2.\ A(i, j-1) \geq A(i,j), & \text{if } i \text{ is odd} \end{array}$$

(a) (**5 points**) Show that initially, the number of clean rows of A can be any value from 0 to N_1 (inclusive).

(b) (**5 points**) For every even numbered i with $0 \leq i < N_1$, do a compare-and-swap of $A(i,j)$ and $A(i+1,j)$ for all $0 \leq j < N_2$. Show that after the compare-and-swap, at least one of row i or row $i+1$ must be clean.

Hint: consider what happens if the total number of zeros in rows i and $i+1$ is greater than the total number of ones. What if the number of ones is greater than the number of zeros?



Draw a critical path on the figure above and include it with your solutions.

Figure 4: Critical path template for revised parallel editing distance computation

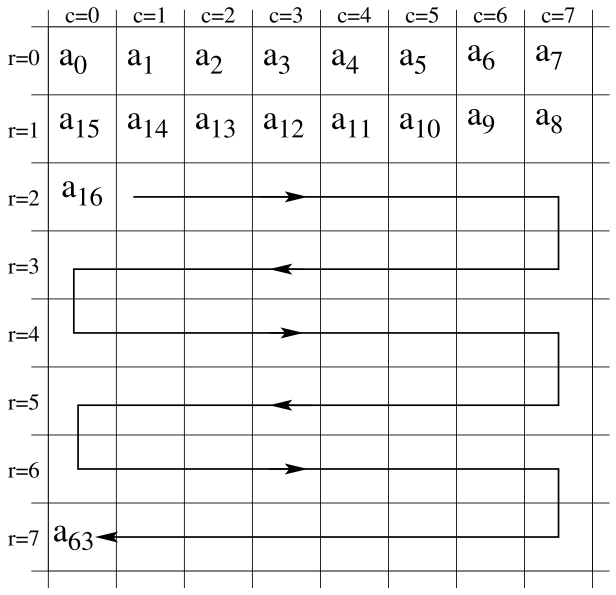

c=0
c=1
c=2
c=3
c=4
c=5
c=6
c=7
r=0
r=1
r=2
r=3
r=4
r=5
r=6
r=7
a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7
a_{15} a_{14} a_{13} a_{12} a_{11} a_{10} a_9 a_8
a_{16}
a_{63}

Figure 5: Snake-ordering of the elements of an array

(c) (**5 points**) Instead of doing the compare-and-swap from part b, sort each column into ascending order, i.e.:

$$\forall 0 < i < N_1.\ A(i-1,j) \leq A(i,j)$$

Show that after all of the columns are sorted in this way, A has at least $N_1/2$ clean rows.

(d) (**8 points**) Show that if the following algorithm is executed:

```
for k = 1 to $\log_2 N_1$ do
    forall i in (0..($N_1$-1)) do
        if(i is even) sort row $i$ of $A$ ascending to the right;
        else sort row $i$ of $A$ descending to the right;
    end
    forall j in (0..($N_2$-1)) do
        sort column $j$ of $A$ into ascending order;
end
```

array A has at most one dirty row.

(e) (**5 points**) Show that at the end of step d, A is sorted into ascending “snake” order (i.e. a is sorted into ascending order when mapped to A as described above).

☺

☺