CS2101 Due: Wednesday 23-rd of October 2013

Problem Set 1

CS2101 *Submission instructions on last page*

PROBBLEM 1. [40 points] The goal of the exercise is to experiment with the performance degradation caused by high rate of cache misses. We will also experiment with several implementations of the same operation and compare their running times within `Julia`. To this end, we will consider a fundamental and simple operation: matrix transposition. Reviewing this operation can be done at the wikipedia page,

`http://en.wikipedia.org/wiki/Transpose`

Question 1. [10 points] We saw in class, as part of the section dedicated to `Cilk`, a simple way to perform matrix transposition. A pseudo-code for this transposition principle is stated below. Please note that we state for an input rectangular matrix A, with `m` rows and `n` columns, and an output matrix ${}^tA$ (thus rectangular with `n` rows and `m` columns):

```
for i=1:m
    for j=1:n
        B[j,i] = A[i,j]
```

Hence this is necessarily an out-of-place procedure while the `Cilk` code seen in class was in place since it was stated for a square matrix.

You are required to write a `Julia` function that takes an `m` $\times$ `n` matrix A and returns its transpose ${}^tA$ following the transpose principle stated above. Please note that this `Julia` function is not required to use any of the parallelism constructs of the `Julia` language. However, please feel free to experiment with those parallelism constructs, if you like.

Question 2. [10 points] We investigate another approach for computing the transpose ${}^tA$. For simplicity, we focus on the case where A is a square matrix. The proposed approach is based on a divide and conquer scheme. In the formula below, we assume that n is a power of 2 and that $A_{1,1}, A_{1,2}, A_{2,1}, A_{2,2}$ denote square blocks of order $n/2$.

$$ {}^tA = \begin{cases} \begin{pmatrix} {}^tA_{1,1} & {}^tA_{2,1} \\ {}^tA_{1,2} & {}^tA_{2,2} \end{pmatrix} & \text{if} \quad A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \\ A & \text{if} \quad n = 1 \end{cases} \tag{1} $$

You are required to write a `Julia` function that

- takes as input a positive integer value n, which is assumes to be a power of 2,
- generate an $n \times n$ matrix `a` with random entries of type integer with values in the range $0 \cdots n-1$.

- transpose the matrix in place using this divide-and-conquer approach.

Once again, using parallelism constructs is not required, but you are welcome to experiment with it.

Question 3. [10 points] We consider a third approach for computing the transpose ${}^tA$. This is again a divide and conquer, called REC-TRANSPOSE. Now, no hypothesis is made on the shape or size of A. Thus A is an input rectangular matrix with `m` rows and `n` columns, as in Question 1. We present the principle below. Please note that the algorithm takes as parameters the input matrix A and the output transpose $B = {}^t A$. This implies that the algorithm modifies one its parameters, namely B, and works out-of-place as the algorithm in Question 1.

- If $n \geq m$, the REC-TRANSPOSE algorithm partitions

$$A = (A_1 \ A_2) \ , \quad B = \begin{pmatrix} B_1 \\ B_2 \end{pmatrix}$$

  and recursively executes REC-TRANSPOSE(A_1, B_1) and REC-TRANSPOSE(A_2, B_2).

- If $m > n$, the REC-TRANSPOSE algorithm partitions

$$A = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \ , \quad B = (B_1 \ B_2)$$

  and recursively executes REC-TRANSPOSE(A_1, B_1) and REC-TRANSPOSE(A_2, B_2).

You are required to write a `Julia` function that

- takes as input two positive integer value m and n,
- generates an $m \times n$ matrix `a` with random entries of type `int` with values in the range $0 \cdots n-1$.
- computes the transposed matrix ${}^tA$ using the REC-TRANSPOSE divide and conquer scheme.

An example of a similar procedure is the divide-and-conquer matrix multiplication in Exercise 3 of Lab 4, for which the solution is posted on the course web site.

Question 4. [10 points] You are required to compare experimentally the running times of the above three implementations of matrix transposition. This implies to choose a series of matrix sizes and apply these three implementations to each selected size.

PROBBLEM 2. [40 points] Here again, the goal of the exercise is to experiment with the performance degradation caused by high rate of cache misses. To this end, you are required to implement a well-known algorithm for sorting integer numbers and try it on larger and larger input data sets.

This algorithm is the counting sort algorithm. At its wikipedia page,

http://en.wikipedia.org/wiki/Counting_sort

you will find a detailed description of this algorithm, which is a very natural procedure. This algorithm was also presented during the labs.

We summarize below the principle and show pseudo-code for the algorithm:

- *Counting sort* takes as input a collection of n items, each of which known by a key in the range $0 \cdots k$.
- The algorithm computes a *histogram* of the number of times each key occurs.
- Then performs a *prefix sum* to compute positions in the output.

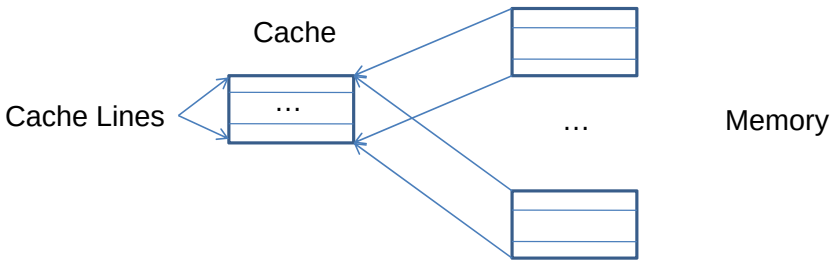
```
allocate an array Count[0..k]; initialize each array cell to zero
for each input item x:
    Count[key(x)] = Count[key(x)] + 1
total = 0
for i = 0, 1, ... k:
    c = Count[i]
    Count[i] = total
    total = total + c
allocate an output array Output[0..n-1]
for each input item x:
    store x in Output[Count[key(x)]]
    Count[key(x)] = Count[key(x)] + 1
return Output
```

Question 1. [30 points] Write a `Julia` program implementing the counting sort algorithm. The input of this program will be simply two positive integers `n` and `k`. The integer `n` is the size of the array to be sorted. The entries of this array will be random non-negative integers less or equal than `k`. Hence, one can use the following `Julia` command to create it: `rand(1:k,1,n)`. There will be no output for this program, since we are only interested in its running time. However, you should test that your program and make sure that it sorts properly. Note that to enforce the use of 32-bit integers the array `Count` can be initialized as follows: `Count = zeros(Int32,1,k+1)`. The `k+1` is because entry range is `0:k`.

Question 2. [5 points] Let `k` be the largest value of a `Julia` 32-bit positive integer. Measure the running time of your program for `n` equal to `i * 10000000` where `i` is successively 1,2,3,4,5,6.

Question 3. [5 points] Theoretically, the number of operations performed by the counting sort algorithm is (asymptotically) proportional to `n + k`. Do the experimental results of the previous question reflect that fact? If not, explain why. Hint: Try to estimate the number of caches incurred by the algorithm when both `n` and `k` are larger than the size of the L1 cache.

PROBBLEM 3. [20 points] The following four questions are using this simple cache memory; the same as in class.



We recall its key features.

- Byte addressable memory.
- The Cache has size 32Kbyte with direct mapping and 64 byte lines (512 lines); so the cache can fit $2^9 \times 2^4 = 2^{13}$ `int`.
- **Therefore**, successive 32Kbyte memory blocks can line up in cache.
- A cache access costs **1 cycle while.** a memory access costs **100 cycles.**
- How addresses map into cache
  - Bottom 6 bits are used as offset in a cache line,
  - Next 9 bits determine the cache line

For each of the following four questions, answers must be justified. Total access times should be expressed in terms of S. That is, do **not** replace S by its numerical value.

Question 1. [5 points]

```
// sizeof(int) = 4 and Array laid out sequentially in memory
#define S ((1<<20)*sizeof(int))
int A[S];
// Thus size of A is 2^(20) x 16 bytes
for (i = 0; i < S; i++) {
     read A[2];
}
```

What is the total access time of this program? What kind of locality does it have, if any? What kind of cache misses?

Question 2. [5 points]

```
// sizeof(int) = 4 and Array laid out sequentially in memory
#define S ((1<<20)*sizeof(int))
int A[S];
```

```
// Thus size of A is 2^(20) x 16 bytes
for (i = 0; i < S; i++) {
     read A[i];
}
```

What is the total access time of this program? What kind of locality does it have, if any? What kind of cache misses?

Question 3. [5 points]

```
// sizeof(int) = 4 and Array laid out sequentially in memory
#define S ((1<<20)*sizeof(int))
int A[S];
// Thus size of A is 2^(20) x 16 bytes
for (i = 0; i < S; i++) {
     read A[(32 * i) % S];
}
```

What is the total access time of this program? What kind of locality does it have, if any? What kind of cache misses?

Question 4. [5 points]

```
#define S ((1<<19)*sizeof(int))
int A[S];
int B[S];
// Thus, in the main memory, the cache lines of
// B are just after all the cache lines of A
for (i = 0; i < S; i++) {
     read B[i], A[i];
}
```

What is the total access time of this program? What kind of locality does it have, if any? What kind of cache misses?

Submission instructions.

Format: The answers to the questions of Problem 3 should be typed and submitted as a PDF file called `Pb3.pdf`. No format other than PDF will be accepted. Problems 1 and 2 involve programming with Julia: they must be submitted as two input files to be called `Pb1.jl` and `Pb2.jl`, respectively. Each of these two files must be a valid input file for Julia. In addition, each user defined function must be documented. To summarize, each assignment submission consists of three files: `Pb1.jl`, `Pb2.jl` and `Pb3.pdf`.

Submission: The assignment should be returned to the instructor **and** the TA by email.

Collaboration. You are expected to do this assignment *on your own* without assistance from anyone else in the class. However, you can use literature and if you do so, briefly list your references in the assignment. Be careful! You might find on the web solutions to our problems that are not appropriate. For instance, because the cache memory model is different. So please, avoid those traps and work out the solutions by yourself. You should not hesitate to contact the instructor or the TA if you have any question regarding this assignment. We will be more than happy to help.

Marking. This assignment will be marked out of 100. A 10 % bonus will be given if your answers are clearly organized, precise and concise. Messy assignments (unclear statements, lack of correctness in the reasoning, many typographical or language mistakes) may give rise to a 10 % malus.