Exercises for lab 5 of CS2101a

Instructor: Marc Moreno Maza, TA: Xiaohui chen

November 28, 2012

In this lab session, you will write a multithreaded program in `Cilk++` for performing matrix multiplication. First, you will parallelize a program that performs matrix multiplication using the naive iterative method based on three nested loops. Then, you will write a serial program to perform matrix multiplication by divide-and-conquer and parallelize it by inserting `Cilk++` keywords. For simplicity, all matrices are in row-major layout.

1 Exercise 0

The directory `qsort` of our `Cilk++` examples contains a program for the *quicksort* algorithm, together with a `Makefile` to compile it. You can learn about the quicksort algorithm from the page:

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

Now you know all popular sorting algorithms!

Open the file `Cilk-Programmers-Guide.pdf` which you can download from:

`http://www.clear.rice.edu/comp422/resources/Intel_Cilk++_Programmers_Guide.pdf`

1. Read pages 12 to 15. Repeat the command lines of the Section `BUILD, EXECUTE AND TEST`.
2. Make sure you understand how to measure the work and the span of your program using `cilkview`. You will find the necessary information in the user guide.
3. Analyze the work and the span of your program. To this end, you can conduct complexity analysis and/or use `cilview`.

2 Exercise 1

The following `C` program implements the naive iterative method for multiplying (square) matrices. The matrices are dense and random with `int` coefficients, for simplicity.

```
#include <stdio.h>
#include <stdlib.h>

/* mm_loop_serial is the naive iterative method */
void mm_loop_serial(int* C, int* A, int* B, int n)
{
    /* DO NOT MODIFY THIS CODE.  THIS IS USED TO VERIFY THAT YOUR
     * CODE IS PRODUCING THE RIGHT ANSWER. */
    int i,j,k;
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            for (k=0; k<n; k++)
                C[i*n+j] += A[i*n+k] * B[k*n+j];
}

/* mm_loop_parallel is the parallel implementation of mm_loop_serial */
void mm_loop_parallel(int* C, int* A, int* B, int n)
{
    /* MODIFY THIS CODE TO MAKE IT PARALLEL */
    int i,j,k;
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            for (k=0; k<n; k++)
                C[i*n+j] += A[i*n+k] * B[k*n+j];
}

/* random_matrix creates a random n-by-n matrix */
void random_matrix(int* A,  int n)
{
  int i,j;

  for(i=0;i<n;i++) {
    for(j=0;j<n;j++) {
      A[i*n + j] =  rand()%n;
    }
  }
}

/* Print an n-by-n matrix */
void print_matrix(int* a,  int n)
{
  int i,j;
  for(i=0;i<n;i++) {
    for( j=0; j<n; j++) {
      printf("%d ", a[n*i+j]);
      if (j == n-1) printf("\n");
```

```
    }
  }
  printf("\n");
}


int main() {
  int n, s;
  int* a;
  int* b;
  int* c;
  int* d;
  printf("n = ");
  scanf("%d", &n);
  printf("\n");
  s = n * n;
  if (s < 1000000000) {
    printf("s = %d\n", s);
    a = (int *) malloc(s * sizeof(int));
    random_matrix(a,n);
    b = (int *) malloc(s * sizeof(int));
    random_matrix(b,n);
    if (n < 10) print_matrix(a,n);
    if (n < 10) print_matrix(b,n);
    c = (int *) malloc(s * sizeof(int));
    mm_loop_serial(c,a,b,n);
    if (n < 10) print_matrix(c,n);
    d = (int *) malloc(s * sizeof(int));
    mm_loop_parallel(d,a,b,n);
    if (n < 10) print_matrix(d,n);
    free(a);
    free(b);
    free(c);
    free(d);
  }

  return 0;
}
```

1. Modify the above program such that you can compile with the `cilk++` compiler using a `Makefile`.
2. Parallelize the function `mm_loop_parallel` using the `cilkfor` construct. Compile and test your program.
3. Once your parallelized `mm_loop_parallel` function. works properly, com-

ment out the call to the `mm_loop_serial` function. Measure the running times for $n = 2^9$, $n = 2^{10}$ and $n = 2^{11}$. Compare with the running times obtained with the function `mm_loop_serial`.

3 Exercise 2

We propose to improve the performances of the previous matrix multiplication program by integrating a divide-and-conquer strategy.

Assume that the input matrices are square matrices A and B of order n where n is a multiple of 2. Then we decompose each of A, B and C into 4 blocks of equal format:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \; B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \; C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix},$$

where each of A_{ij}, B_{ij}, C_{ij} is a square matrix of order $n/2$. Then we have

$$\begin{array}{rcl} C_{11} & = & A_{11}B_{11} + A_{12}B_{21} \\ C_{12} & = & A_{11}B_{12} + A_{12}B_{22} \\ C_{21} & = & A_{21}B_{11} + A_{22}B_{21} \\ C_{22} & = & A_{21}B_{12} + A_{22}B_{22} \end{array}$$

Observe that

- one can first compute the four products $A_{11}B_{11}$, $A_{11}B_{12}$, $A_{21}B_{11}$, $A_{21}B_{12}$ *in parallel* and store them respectively in C_{11}, C_{12}, C_{21}, C_{22}.
- one can secondly compute the four products $A_{12}B_{21}$, $A_{12}B_{22}$, $A_{22}B_{21}$, $A_{22}B_{22}$ *in parallel* and add them respectively to C_{11}, C_{12}, C_{21}, C_{22}.

Modify the program of Exercise 1 so as to add a function implementation the above observation, that we will refer to *divide-and-conquer matrix multiplication.* For the computations of the products $A_{11}B_{11}$, $A_{11}B_{12}$, $A_{21}B_{11}$, $A_{21}B_{12}$, $A_{12}B_{21}$, $A_{12}B_{22}$, $A_{22}B_{21}$, $A_{22}B_{22}$ consider successively and compare experimentally the following approaches:

1. these 8 products are computed by `mm_loop_serial`
2. these 8 products are computed by `mm_loop_parallel`
3. these 8 products are computed by the divide-and-conquer matrix multiplication until `n` reaches a value which is small enough (say 64 or 16) and then by `mm_loop_serial`.