Project Assignment 3

—

OpenMP Programming

Multiprocessor Systems, DV2544

Håkan Grahn
Blekinge Institute of Technology

April 24, 2014

1 Introduction

The task in this project assignment is to implement two parallel algorithms on a shared-memory computer with 8 cpus using OpenMP.

The examination of this projekt is done by sending in the project, with complete source code and a short report in PDF format, before the examination deadline. The examination deadline is May 28, 2014 at 23:55. The reports and source code should be submitted on the course page at It's Learning.

2 Goals

This project assignment serves two purposes:

- Introduce you to basic OpenMP programming.
- Give some experiece of how work and data partitioning impact the performance of a parallel application on a shared address-space computer.

The rationale behind these goals are that OpenMP has emerged as a de-facto standard to implement high-performance parallel applications on shared address-space computers. On such machines, data communication is implicit but may still impact the performance. Further, the partitioning of work into parallel threads, the synchronization of parallel threads, and the protection of shared data may also have a significant impact on performance.

3 Preconditions

3.1 Prerequisites

- You are supposed to have good programming experience and not be new to C programming.
- You are supposed to have basic knowledge about working in a Unix/Linux environment.
- Operating systems issues and concepts should not be unfamiliar to you.

3.2 Laboratory groups

You are encouraged to work in groups of two. Groups larger than two is not accepted.

Discussion and help between laboratory groups are encouraged. It is normally not a problem, *but* watch out so that you do not cross the border to cheating, see section 3.5 below.

3.3 Lecture support

There is one lecture on programming with OpenMP in general, e.g., introducing the general concepts of shared-memory programming and an overview of the OpenMP directives and library functions. In addition to this, you are expected to search for additional information on your own.

3.4 Examination

See section 4.3.

3.5 Cheating

All work that is not your own should be properly referenced. If not, it will be considered as cheating and reported as such to the university disciplinary board.

4 Project Tasks to Complete

In this project assignment you are going to implement parallel versions of two different algorithms:

1. Implement a parallel version of Quicksort using OpenMP (described in Section 4.1).
2. Implement a parallel version of Gaussian elimination using OpenMP (described in Section 4.2).

The algorithms shall be implemented using OpenMP directives and library functions, and compile and execute correctly on a Linux-based shared-memory machine with 8 cpus. I will use `kraken` to test and verify that your applications are correct.

4.1 Parallel Quicksort implementation

4.1.1 Problem Description

The first application we shall parallelize is Quicksort. We saw in the laboratory exercise that Matrix multiplication is easy to parallelize using data decomposition and extracting loop parallelism. In contrast, Quicksort fits well for recursive decomposition. A sequential version of Quicksort is found in the file `qsort_seq.c`. The code for the sequential implementation of the Quicksort algorithm is listed in Appendix A.

4.1.2 Tasks to Complete

You are supposed to do the following:

- Write a parallel version of Quicksort using OpenMP directives and library functions.
- Compare the performance (i.e., execution time) of your solution with the performance the sequential version of Quicksort given to you. Your application shall have a speedup of at least 2 on 8 cpus.

The source code for the sequential implementation of Quicksort is found on the course home page at It's Learning.

4.2 Gaussian elimination

4.2.1 Problem Description

The problem to be solved in this task is to implement a parallel version of Gaussian elimination using OpenMP. The algorithm is described in Section 8.3 in [1] (presented as Algorithm 8.4). A sequential version of the algorithm is shown in Figure 1. You shall initialize the matrix `A`, and the vectors `y` and `b` with reasonable values.

The code for a sequential implementation of Gaussian elimination is found in Appendix A. The code is based on Algorithm 8.4 in Section 8.3 in [1].

```
1.  procedure GAUSSIAN_ELIMINATION (A, b, y)
2.  begin
3.    for k := 0 to n − 1 do                    /* Outer loop */
4.    begin
5.      for j := k + 1 to n − 1 do
6.        A[k, j] := A[k, j]/A[k, k];   /* Division step */
7.      y[k] := b[k]/A[k, k];
8.      A[k, k] := 1;
9.      for i := k + 1 to n − 1 do
10.     begin
11.       for j := k + 1 to n − 1 do
12.         A[i, j] := A[i, j] − A[i, k] × A[k, j]; /* Elimination step */
13.       b[i] := b[i] − A[i, k] × y[k];
14.       A[i, k] := 0;
15.     endfor;          /* Line 9 */
16.   endfor;            /* Line 3 */
17. end GAUSSIAN_ELIMINATION
```

Figure 1: Serial Gaussian elimination algorithm (Algorithm 8.4 in [1]).

4.2.2 Tasks to Complete

You are supposed to do the following:

- Write a parallel implementation of gaussian elimination using OpenMP.
- Measure the speedup of your parallel version on 8 cpus.
- The resulting OpenMP implementation shall have a speedup of at least 1.5 (on 8 cpus) over the sequential version.

4.3 Examination

Prepare and submit a `tar`-file (or `zip`-file) containing:

- **Source code:** The source-code for working solutions to the tasks in sections 4.1 and 4.2, i.e., a listing of your well-commented source code for your parallel version of the applications.
- Corresponding `Makefile`(s), or a text-file describing how to compile the projects.
- **Measurements:** You should provide execution times for three cases: the sequential version of the application, the parallel version of the application running on one cpu/thread, and the parallel version of the application running on eight cpus.
- **Implementation:** A short, general description (one-two pages) of your parallel implementation, i.e., you should describe how you have partitioned the work between several cpus, how the data structures are organized, etc. The format of the report must be pdf or plain text.

All material (except the code given to you in this assignment) must be produced by the laboratory group alone.

The examiner may contact you within a week, if he needs some oral clarifications on your code or report. In this case, all group members must be present at that oral occasion.

References

[1] A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to parallel computing, 2nd edition*, Addison-Wesley, 2003.

Appendix: Source code listings

```
/**********************************************************************
 *
 * Sequential version of Quick sort
 *
 **********************************************************************/

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

#define KILO (1024)
#define MEGA (1024*1024)
#define MAX_ITEMS (64*MEGA)

#define swap(v, a, b) {unsigned tmp; tmp=v[a]; v[a]=v[b]; v[b]=tmp;}

static int *v;

static void
print_array(void)
{
    int i;

    for (i = 0; i < MAX_ITEMS; i++)
        printf("%d ", v[i]);
    printf("\n");
}

static void
init_array(void)
{
    int i;

    v = (int *) malloc(MAX_ITEMS*sizeof(int));
    for (i = 0; i < MAX_ITEMS; i++)
        v[i] = rand();
}

static unsigned
partition(int *v, unsigned low, unsigned high, unsigned pivot_index)
{
    /* move pivot to the bottom of the vector */
    if (pivot_index != low)
        swap(v, low, pivot_index);

    pivot_index = low;
    low++;

    /* invariant:
     * v[i] for i less than low are less than or equal to pivot
     * v[i] for i greater than high are greater than pivot
     */

    /* move elements into place */
    while (low <= high) {
        if (v[low] <= v[pivot_index])
            low++;
        else if (v[high] > v[pivot_index])
            high--;
        else
            swap(v, low, high);
    }

    /* put pivot back between two groups */
    if (high != pivot_index)
        swap(v, pivot_index, high);
    return high;
}
```

```
static void
quick_sort(int *v, unsigned low, unsigned high)
{
    unsigned pivot_index;

    /* no need to sort a vector of zero or one element */
    if (low >= high)
        return;

    /* select the pivot value */
    pivot_index = (low+high)/2;

    /* partition the vector */
    pivot_index = partition(v, low, high, pivot_index);

    /* sort the two sub arrays */
    if (low < pivot_index)
        quick_sort(v, low, pivot_index-1);
    if (pivot_index < high)
        quick_sort(v, pivot_index+1, high);
}

int
main(int argc, char **argv)
{
    init_array();
    //print_array();
    quick_sort(v, 0, MAX_ITEMS-1);
    //print_array();
}
```

```
/****************************************************
 *
 * Gaussian elimination
 *
 * sequential version
 *
 ****************************************************/

#include <stdio.h>

#define MAX_SIZE 4096

typedef double matrix[MAX_SIZE][MAX_SIZE];

int N;  /* matrix size  */
int maxnum;  /* max number of element*/
char *Init;  /* matrix init type */
int PRINT;  /* print switch  */
matrix A;  /* matrix A  */
double b[MAX_SIZE]; /* vector b */
double y[MAX_SIZE]; /* vector y */

/* forward declarations */
void work(void);
void Init_Matrix(void);
void Print_Matrix(void);
void Init_Default(void);
int Read_Options(int, char **);

int
main(int argc, char **argv)
{
    int i, timestart, timeend, iter;

    Init_Default();  /* Init default values */
    Read_Options(argc,argv); /* Read arguments */
    Init_Matrix();  /* Init the matrix */
    work();
    if (PRINT == 1)
      Print_Matrix();
}

void
work(void)
{
    int i, j, k;

    /* Gaussian elimination algorithm, Algo 8.4 from Grama */
    for (k = 0; k < N; k++) { /* Outer loop */
      for (j = k+1; j < N; j++)
          A[k][j] = A[k][j] / A[k][k]; /* Division step */
      y[k] = b[k] / A[k][k];
      A[k][k] = 1.0;
      for (i = k+1; i < N; i++) {
          for (j = k+1; j < N; j++)
            A[i][j] = A[i][j] - A[i][k]*A[k][j]; /* Elimination step */
          b[i] = b[i] - A[i][k]*y[k];
          A[i][k] = 0.0;
      }
    }
}

void
Init_Matrix()
{
    int i, j;

    printf("\nsize = %dx%d ", N, N);
    printf("\nmaxnum = %d \n", maxnum);
    printf("Init = %s \n", Init);
```

```
    printf("Initializing matrix...");

    if (strcmp(Init,"rand") == 0) {
      for (i = 0; i < N; i++){
          for (j = 0; j < N; j++) {
            if (i == j) /* diagonal dominance */
                A[i][j] = (double)(rand() % maxnum) + 5.0;
            else
                A[i][j] = (double)(rand() % maxnum) + 1.0;
          }
      }
    }
    if (strcmp(Init,"fast") == 0) {
      for (i = 0; i < N; i++) {
          for (j = 0; j < N; j++) {
            if (i == j) /* diagonal dominance */
                A[i][j] = 5.0;
            else
                A[i][j] = 2.0;
          }
      }
    }

    /* Initialize vectors b and y */
    for (i = 0; i < N; i++) {
      b[i] = 2.0;
      y[i] = 1.0;
    }

    printf("done \n\n");
    if (PRINT == 1)
      Print_Matrix();
}

void
Print_Matrix()
{
    int i, j;

    printf("Matrix A:\n");
    for (i = 0; i < N; i++) {
      printf("[");
      for (j = 0; j < N; j++)
          printf(" %5.2f,", A[i][j]);
      printf("]\n");
    }
    printf("Vector b:\n[");
    for (j = 0; j < N; j++)
      printf(" %5.2f,", b[j]);
    printf("]\n");
    printf("Vector y:\n[");
    for (j = 0; j < N; j++)
      printf(" %5.2f,", y[j]);
    printf("]\n");
    printf("\n\n");
}

void
Init_Default()
{
    N = 2048;
    Init = "rand";
    maxnum = 15.0;
    PRINT = 0;
}

int
Read_Options(int argc, char **argv)
{
    char *prog;
```

```
    prog = *argv;
    while (++argv, --argc > 0)
      if (**argv == '-')
          switch ( *++*argv ) {
          case 'n':
            --argc;
            N = atoi(*++argv);
            break;
          case 'h':
            printf("\nHELP: try sor -u \n\n");
            exit(0);
            break;
          case 'u':
            printf("\nUsage: sor [-n problemsize]\n");
            printf(" [-D] show default values \n");
            printf(" [-h] help \n");
            printf(" [-I init_type] fast/rand \n");
            printf(" [-m maxnum] max random no \n");
            printf(" [-P print_switch] 0/1 \n");
            exit(0);
            break;
          case 'D':
            printf("\nDefault: n = %d ", N);
            printf("\n Init = rand" );
            printf("\n maxnum = 5 ");
            printf("\n P = 0 \n\n");
            exit(0);
            break;
          case 'I':
            --argc;
            Init = *++argv;
            break;
          case 'm':
            --argc;
            maxnum = atoi(*++argv);
            break;
          case 'P':
            --argc;
            PRINT = atoi(*++argv);
            break;
          default:
            printf("%s: ignored option: -%s\n", prog, *argv);
            printf("HELP: try %s -u \n\n", prog);
            break;
          }
}
```