Lecture Parallel Computing, WS 2019

Assignment 2: OpenMP – Stencil Code

The assignment consists of 3 parts: development of a sequential scientific code, parallelization with OpenMP, and performance evaluation.

Part 1: Stencil Code and Sequential Program Version

Consider a square matrix $A \in \mathbb{N}^{N \times N}$ with $N = 46080$ and matrix elements represented as type **`unsigned int`**. The first column is initialized to **`150`**, the first row is initialized to **`250`**, and the first element of the matrix to **`0`**. For each element of the matrix except the left and upper border the following C stencil operation is performed:

```
A[i][j]= ((f1(A[i-1][j-1])+
          f1(A[i][j-1])+
          f1(A[i-1][j]))*
          f1(A[i-1][j-1]+A[i][j-1]+A[i-1][j]))
          % 1000;
if (A[i][j]==0) A[i][j]=999;
```

with function f1 defined as follows:

```
unsigned int f1(unsigned int x)
    {
      unsigned int n=0;
      unsigned int k=0;
      unsigned int m=1;

     for (k+=m; k<=x; k+=m)
        {
          n=n+1;
          m=m+2;
        }

     return n;
    }
```

Write a sequential C or C++ program **`stencil`** which iterates over the matrix and calculates **ALL** elements. The sequential version shall be used to check the correctness of the parallel version. Program **`stencil`** is called with 6 integer values denoting three elements which shall be written to the standard output (used for testing), e.g.

```
the following call outputs the values of the 3 elements:

$ ./stencil 25 48 3000 2000 25673 41983

  value of A[25][48], A[3000][2000], A[25673][41983]
```

Part 2: OpenMP Parallelization

The goal is to develop a highly efficient parallel code version. The execution time of the program shall be improved by optimizing the OpenMP code as well as by experimenting with compiler flags. The experiments shall be done with the Intel icc compiler (available compilers: gcc, g++, icc, icpc with flags –fopenmp or –openmp). During the experiments OS commands shall be used to monitor the system.

Part 3: Performance Analysis and Report

Tasks to be done:

1. Develop an OpenMP version and compile it with icc. Use a makefile for compiling and running your code (so we can see your options).
2. Determine the optimal number of threads and calculate the speed-up compared with the sequential version.
3. Examine the system during execution:
   How many processors are visible? How many threads are created? What about work balance and task scheduling?
4. Try to get better results by modifying the OpenMP source.
5. Try to get better results by experimenting with compiler flags.
6. Also run some experiments with gcc and compare it with icc.

Write a report document covering the following topics (not more than 3 pages):

1. Explain your parallelization strategy.
2. Report your main OpenMP pragmas. Is your solution based on the C or C++ ?
3. Report the best execution times (show the number of threads, options, etc. mainly responsible for obtaining good results).
4. Show the speed-ups compared with the serial version.
5. Report about the OS commands which you have been using for monitoring the system and their usefulness and the behavior of the system you have noticed.
6. Report on your gcc experiences.

Execution Environment

CORA Cluster – Hardware

- Heterogeneous cluster with CPUs and GPUs
- 1 front-end node and 8 computational node
- Per computational node
    - CPUs:
      2x Intel Xeon quad-core X5550 2.66Ghz,
      24GB DDR3-1333 system memory
    - GPUs: GPUs not used in this lecture

CORA Cluster – Software

- Red Hat Enterprise Linux Server release 6.6 (Santiago)
- Command „cat /etc/issue“
- main compiler Intel icc,
  list of all compilers: gcc, g++, icc, icpc with flags -fopenmp or -openmp

CORA Cluster – Access

- SSH to cora.par.univie.ac.at
- Username: aMatrikelnummer
- Password: supplied in lecture
- After first login change the password!
    - $ passwd

Lab and Moodle

Arrangements

1. Put your files in ~aMatrNr/pc/ex2 (note that the data files can be huge – don't waste storage!).
2. Run the OpenMP code on the nodes cora01–cora07 and not on the frontend node; i.e. use e.g. ssh cora02. For time measurements make sure that you are the only user on the node (command users).
3. Build a zip-file with
    a. sequential source file
    b. OpenMP source file **`stencil.c`** (best program version; the uploaded version **shall not** write matrix *A* to the output, but only the values of the 3 specified elements)
    c. makefile
    d. document assignment2-report.pdf

   and upload it in Moodle before the deadline.
4. Use forum for questions.

Requirements for positive grading

- Source files and report must have been submitted.
- Source files must compile/link correctly.
- Program **`stencil`**:
  - values must be computed correctly
  - performance measurements must have been done and documented
  - a substantial speedup must be achieved

Beware of the size

Dimension of 46.080x46.080 results in 2.123.366.400 elements and a matrix size of about 8.5 GB. A four-byte integer runs from -2.147.483.648 to 2.147.483.647.

Deadline

Deadline: 10 December 2019