Universiteit van Amsterdam
BSc Informatica
Dr Clemens Grelck

Concurrency en
Parallel Programmeren
2014/2015

Assignment 2

Multithreaded Programming with OpenMP Directives

Assignment 2.1: Wave equation simulation

Reconsider the 1-dimensional wave equation studied in assignment 1.1. Parallelize your sequential simulation code using OpenMP and repeat the experiments of assignment 1.1. Report your results and compare them with those obtained by your pthread-based code developed for assignment 1.1.

Experiment with different scheduling techniques and block sizes using the number of threads that achieved the highest performance in your previous experiments. Report your results and explain why some schedulings perform better than others.

Assignment 2.2: Reduction operations

As explained during the lecture a *reduction operation* uses an associative (and potentially commutative) binary function to fold all values of an aggregate data structure (e.g. vector, matrix, tree) into a single value.

Parallelize the following C functions using OpenMP directives:

1. Reduction with fixed reduction operator

```
double sum (double* vec, int len)
{
  int i;
  double accu = 0.0;

  for (i=0; i<len; i++) {
    accu = accu + vec[i];
  }

  return accu;
}
```

2. Reduction with unknown reduction operator

```
double reduce ( double (fun)(double, double),
                double* vec, int len, double neutral)
{
  int i;
  double accu = neutral;

  for (i=0; i<len; i++) {
    accu = fun( accu, vec[i]);
  }

  return accu;
}
```

Run experiments with both functions using 1, 2, 4, 6, 8 threads assuming an 8-core experimental system and report speedup graphs for vector sizes 10^3, 10^4, 10^5, 10^6, 10^7. In order to obtain reproduce-able measurements use a high precision timer (see framework code provided for assignment 1.1) and repeatedly compute the same reduction operation on a precomputed vector.

Can you think of an efficient and scalable implementation of the `reduce` function using OpenMP? Be creative, but ensure deterministic results (modulo non-associativity of floating point computer arithmetic). Implement your solution and describe how you developed the conceptual idea.

Assignment 2.3: Loop scheduling

Write an OpenMP-parallelized C function that takes a matrix, represented as a vector of row vectors of possibly different length and yields the vector of row-wise sums.

As an argument to the above function create an upper triangular square matrix. The matrix shall be stored as a vector of N pointers to vectors of integer numbers of decreasing length, i.e., the first row of the matrix has N elements, the second row has $N - 1$ elements and the last row consists of a single element only. The elements of the matrix shall be initialized to the sum of their respective row and column index.

Use upper triangular matrices of sizes 100×100, 1000×1000 and 5000×5000; repeat the computation of row sums sufficiently often to obtain useful execution times (more than 10 seconds, but not to wait too long for results).

Experiment with the various scheduling techniques (and their parameters where appropriate) provided by OpenMP.

Instructions for submission:

For each assignment we expect two kinds of deliverables:

a) source code in the form of a tar-archive of all relevant files that make up the solution of a programming exercise with an adequate amount of comments ready to be compiled; and

b) a report in the form of a pdf file that explains the developed solution at a higher level of abstraction, illustrates and discusses the outcomes of experiments and draws conclusions from the observations made.

Please, submit one archive with all code files and one report per assignment series.

Assignment due date: November 14, 2014, midnight