Exercise for Course

Parallel High-Performance Computing

Dr. S. Lang

Return: 20. November 2014 at the beginning of the exercise or earlier

---

Task 8 Fat-tree Network (5 points)

We consider a static *Fat-tree* network. The network topology is a tree, where the connection count between two nodes increases with decreasing distance to the root node: Leafs of the tree are connected with a single (here for simplicity duplex-) wire with their parent node, the next higher level has two of such connections, then four and so on. The topology is shown schematically in Figure 0.3. Determine for this topology the node degree, the total count of connections, the network diameter and the bisection bandwidth.

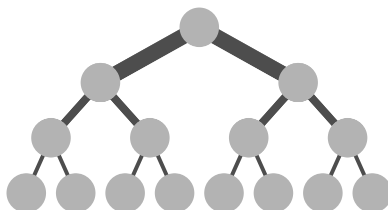

Abbildung 0.3: Static Fat-tree network, schematically. Source: Wikipedia

Task 9 OpenMP: Matrix Multiplication (10 points)

In Exercise 4 we have multiplied two (quadratic) matrices. We now parallelize this program using OpenMP and investigate the scalability of the problem concerning the number of threads. In general we measure in this task computing times and FLOPs related to problem size or count of the involved threads. Hints on time measurement are provided at the end of the task description.

(a) Write a program that realizes the matrx multiplication algorithm with OpenMP based parallelism. The outer `for` loop of the multiplication algorithm shall be processed each by an individual thread, such that each row traversal is started in a single thread that is forked. Measure the computing times and FLOPs rate for $m = 2^n,\ n \in 0, 1, \ldots, 11$ involved threads and for matrix sizes (side length of the matrix not total entries!) $N = 16, 32, 64, 128, 256, 512, 1024$ using `static` scheduling. Plot a time-m diagram for each problem size. Discuss when the overhead introduced by parallelisation pays off, and when not.

(b) Investigate and discuss how your program behaves with `dynamic` scheduling.

(c) Modify the program, that through nested parallel blocks also the instructions of the inner loop can be executed in parallel. Thus also for each scalar product within the row traversal a new threasd is started. For the nested parallelisation there is the OpenMP function `omp_set_nested (int k)`, that allows depending on the value of k several nested levels. The maximal possible nesting depth is controlled with the function `omp_set_max_active_levels(int max)`. For each level an individual `#pragma omp parallel` environment has to be used.
Perform a compute-time vs. m measurement as above with $m \in 1, 2, 4, \ldots, 128$ for e. g. $N = 256, 512$ and 1024 with static scheduling. Does the additional overhead of this variant pay off?

Remarks:

- On *nix systems you can compile OpenMP programs with the gcc-compiler beginning at version 4.1.2 using the option `-fopenmp`. If this option is suppressed the program is compiled as a serial one. To use OpenMP you have to include the file `omp.h`:

```
1 #ifdef _OPENMP
2 #include <omp.h>
3 #endif
```

  The thread count can be set by the environment variable `OMP_NUM_THREADS` in the shell or within the program by calling the function `omp_set_num_threads(int m)`. On the course homepage you find a small program, that illustrates the fundamental ingredients of an OpenMP program.

- For time measurement we cannot use the timing functions from `timer.h`, since these measure the CPU time and therefore the summed-up user time of *all* threads. Thus we measure the elapsed wallclock time of the main thread with the OpenMP method, e.g.

```
1 double tstart = omp_get_wtime();
2   // do something with multiple threads
3 double tend   = omp_get_wtime();
```

  with the unit of s (seconds). Hereby you should ensure, that the time measurement is not influenced too much by other running processe. To execute sufficiently many floating point operations you should execute serveral iterations if necessary.