Coursework for Threaded Programming

Mark Bull

The object of this assessment is to experiment with the loop scheduling options in OpenMP, and to implement an alternative scheduling algorithm. You will be required to conduct some experiments, and submit a report detailing the results of these experiments, as well as the source code you have written.

You are provided with a piece of code which contains two loops. The code measures the execution time for 100 repetitions of each loop, and includes a verification test for each loop.

You may choose to work with *either* the C (`loops.c`) *or* Fortran 90 (`loops.f90`)version.

You should always compile the code with the `-O3` option to ensure a high level of sequential optimisation, but you *may not* alter the code inside the body of the parallel loops.

We will use the term *chunk* in the same sense as in the OpenMP standard, i.e. a contiguous, non-empty subset of the iterations of a loop.

Parallelisation

Add OpenMP directives to parallelise the loops in the routines `loop1` and `loop2`. You should parallelise only the outermost loop in each case.

SCHEDULE clause options

Once you have parallelised the loops, run the code on 6 threads on ARCHER, using the following SCHEDULE clause options:

- STATIC
- AUTO
- STATIC,n
- DYNAMIC,n
- GUIDED,n

where for the latter three cases, n (the chunksize) takes the values $1, 2, 4, 8, 16, 32, 64$.

From these experiments, determine for each loop the best scheduling option on 6 threads. Using this option (which may be different for the two loops), run the code on $1, 2, 3, 6, 12$ and 24 threads.

Alternative schedule

An alternative loop schedule can be implemented by using a parallel region and assigning chunks of loop iterations to threads explicitly. You will find it useful to place each parallel loop in a routine which takes the lower and upper bounds of a chunk as dummy arguments.

Implement the following loop scheduling algorithm, and compare its performance to the SCHEDULE clause options:

Affinity scheduling

Affinity scheduling can be described as follows:

- Each thread is initially assigned a (contiguous) *local set* of iterations.
- For a loop with n iterations, and p threads, each thread's local set is initialised with n/p iterations. (If n does not divide p, choose a suitable distribution of the extra iterations.)
- Every thread executes chunks of iterations whose size is a fraction $1/p$ of the remaining iterations its local set, until there are no more iterations left in its local set.
- When a thread has finished the iterations in its local set, it determines the thread which has the most remaining iterations in chunks it has not yet started executing (the "most loaded" thread) and executes a chunk of iterations whose size is a fraction $1/p$ of the remaining iterations in the "most loaded" thread's local set.
- Threads which have finished the iterations in their own local set repeat the previous step, until there are no more iterations remaining in any thread's local set.

You should take great care with the implementation of this algorithm to synchronise threads correctly and avoid race conditions.Rrun the code on $1, 2, 3, 6, 12$ and 24 threads.

Submission

You are required to submit the following:

1. A written report.
   (Guideline length: 15-20 pages including figures and tables.)
2. Source code.

Your report should contain:

- a *short* introduction (there is no need to include background material on OpenMP or shared memory architectures, for example);
- graphs of the execution time of each loop versus the chunksize for the STATIC, n, DYNAMIC, n and GUIDED, n schedules.
- graphs of the speedup (T_1/T_p) for each loop using the best schedule versus number of threads.
- a discussion of your implementation of the affinity scheduling algorithm, with particular attention to the data structures used and how the threads are synchronised. (Note that this section must be comprehensible without reference to the source code);
- a discussion of the results of running the affinity scheduling algorithm, including appropriate graphs and/or figures;
- some *brief* conclusions.

Your source code submission should contain the parallel version of the code using OpenMP loop directives, with the best scheduling option for each loop on 6 threads, *and* the parallel version using affinity scheduling. The code will be marked on design and readability as well as correctness and performance.

Marks will be allocated as follows: Report 50%, Source code 50%