Gyarmathy Tímea
Group 30443

Granville numbers

Problem statement

Find the Granville numbers within a given interval [A, B]. Solve the problem both with sequential and parallel programming. Parallelize the problem and compare running times and calculate speedup on a number of threads ranging from 1 to 8, using various programming techniques:

- threads
  - C (pThreads)
  - Java/C++/C#
  - Prolog
- OpenMP [C/C++]

In this project, A is chosen to be 2 and B is chosen such that the sequential implementation (on 1 thread) should run for approximately 5 minutes.

Definition and properties of Granville numbers

Granville numbers are an extension of the perfect numbers. They are also called S-perfect numbers. S-perfect numbers constitute a subset of the Granville set S, which is defined by the following:

$$1 \in S \text{ and for all } n \in N, n>1 \text{ let } n \in S \text{ if } \sum_{d \vee n, d<n, d \epsilon S} d \leq n$$

S-perfect numbers are the natural numbers that are equal to the sum of their divisors in S. Every perfect number is also an S-perfect number, however there are numbers such as 24 which are S-perfect but not perfect.

The first few Granville numbers are:

6, 24, 28, 96, 126, 224, 384, 496, 1536, 1792, 6144, 8128, 14336, 15872, 24576, 98304, 114688, 393216, 507904, 917504, 1040384, 1572864, 5540590, 6291456, 7340032, 9078520, 16252928, 22528935, 25165824, 33550336, 56918394, 58720256, 100663296, 133169152.

Algorithm idea

In order to calculate the Granville numbers we start from a set containing only the number 1.

Then, starting from 2, we verify if all the numbers in the set which are divisors of the current number sum up to a result smaller or equal to the current number. In both cases, we add the current number to the set. In case it is equal, we found a Granville number and we print it.

When we parallelize the problem, we are dependent on the result of previous iterations, thus in each iteration we have to wait until all threads added their possible numbers to the set. We synchronize threads with a barrier at the end of the iteration.

Load balancing is done by starting each thread from an offset number then giving it numbers step by step, where the step size is equal to the total number of threads.

Since the synchronization overhead is pretty big, the intervals are not significantly big as to exceed the limit of primitive types.

C (pthreads)

B = 460 000.

Synchronization is done using *pthread_barrier_t* and the critical region is protected by a *pthread_rwlock_t.* The threads are started in a for loop and saved in a thread array. In another for loop the threads are joined.

Time is measured with the *timespec* structure and is printed in seconds.

Within the thread function, first we retrieve a local current set size in order to be able to iterate on the set without having the risk that its size changes in the meantime. At the end of each iteration, we update this number. Reading the array elements is thread safe, adding an element is protected by the above mentioned lock.

C#

B = 300 000.

The implementation has two classes, the main Program class which, similar to the C implementation, creates a list of Thread objects, each starting a method operation with an offset. In another loop, the threads are joined.

The class Granville is instantiated with the total number of threads, and it contains the calculation method. The synchronization is done with a *Barrier* object and with the use of the *lock(...){…}* construct we define an *object* to have the role of the lock over the set.

Prolog

B = 80 000.

Since it would be really complicated to realize synchronization in Prolog, that part is not added, thus for the beginning of computations we might observe inaccuracies. Result set is not retrieved, but printed at the moment when the numbers are found.

The numbers in the set are saved with asserts, then verified at addition.

In contrast to the other implementation, here the divisors are verified to be in the set and not the numbers from the set are verified to be divisors.

Gyarmathy Tímea
Group 30443

OpenMP

B = 550 000.

Thread number is set with the construct given by OpenMP. The calculation loop is preceeded by the construct to define a parrallel omp loop and to list the private and the shared variables.

Synchronization is done with the constructs `#pragma omp critical{…}` and `#pragma omp barrier`.

Charts

Time measurements and graphs can be found on the following sheet:

https://docs.google.com/spreadsheets/d/1pFrtfsGEZTthNYL0sg20CBeV6KvmLLVoa1Lgt_K-5A4/edit?usp=sharing