Exercise for Course

Parallel High-Performance Computing

Dr. S. Lang

Return: 26. November 2015 at the beginning of the exercise or earlier

---

Task 10 Bakery Algorithm (5 points)

In the lecture multiple possibilities to realize mutual exclusion have been discussed (spin locks, ticketing, ...). The following algorithm presented in our abstract notation realizes mutual exclusion without special machine instructions for P processes.

```
1  parallel bakery
2  {
3    const int P            = 8;
4    int       number[P]    = {0[P]};
5    int       choosing[P]  = {0[P]};
6
7    process Proc [int i in {0,...,P-1}]
8    {
9      int mine;
10     while (true)
11     {
12       // entry protocol
13       choosing[i] = 1;
14       mine        = 0;
15
16       for (int k=0; k<P; k++) mine = max(mine,number[k]);
17       number[i]   = mine+1;
18       choosing[i] = 0;
19
20       for (int k=0; k<P; k++)
21       {
22         while (choosing[k]);
23         while (number[k]!=0 &&
24             (number[k]<number[i] || (number[k]==number[i] && k
     <i)));
25       }
26
27       // critical section follows here
28       // exit protocol
29       number[i] = 0;
30       // uncritical section follows here
31     }
32   }
33 }
```

Explain how the algorithm works to achieve mutual exclusion (Hint: Remember the *ticket algorithm*). Investigate why two processes cannot enter the critical section at a time, if one assumes sequential consistency of memory. Discuss, whether the algorithm guarantees deadlock-freeness and final entry (no formal proof necessary!).

Task 11 Goldbach theorem with OpenMP **(10 points)**

After the non-proofen Satz („Goldbach theorem“) each even number $p \geq 4$ can be represented as sum of two prime numbers (where 1 is not regarded as prime). A simple realisation of the test would be by example:

- An array $1 \ldots N$ stores a `bool` value, whether the array index $i \in 1 \ldots N$ is a prime number.
- By iteration with i over the array test for simultaneous primeness for i and $N - i$, whether the number pair $(N-i, i)$ fulfills the condition. Since $i+(N-i) = N$ has eventually a representation by a prime sum been found.

Therefore one necessitates a function, that determines (and stores in an array), whether i is a prime number:

```
1 inline bool is_prime(long i)
2 {
3   const long j = (long) std::sqrt(i);
4   for (long k=2; k<j+1; ++k)
5     if (i % k == 0)
6       return false;
7
8   return true;
9 }
```

Here there are of course still better alternatives, phrase *sieve of Eratosthenes*.

Write a program, that for all numbers i up to an upper limit N counts the number of possible representation sums. The program shall be parallelized with OpenMP this means each thread shall check a fraction of the numbers and count the number of representations. There are of course severeal possible realizations. There might be a global field read and written by all threads, that stores primeness of the numbers from 1 to N; or each thread stores its own array. Alternatively one can calculate primeness of a number also *On the fly* for each i with the above function. Take the approach that seems to be most efficient in your opinion and state shortly why.

Measure for your sequential (one thread) and parallel (with 2, 4 and 8 threads) program the runtime in s. Herefore you can use the already known OpenMP time measurement. What is the speedup you achieve in parallel execution? Measure for $N = 100$ up to at least $N = 10^5$ with sufficient many measurement points. Average over several measurements, avoid outlyers, and also provide information which optimization level you have used. Since the computing effort increases almost exponentially, the load distribution is inhomogenous. Test therefore also the scheduling parameters `static`, `guided` and `dynamic`. Discuss your results using your self-generated figures of computing time, speedup and problem size.