UNIL | Université de Lausanne

Exercise sheet – OpenMP

Simon Scheidegger
simon.scheidegger@gmail.com
July 23th, 2019
Open Source Macroeconomics Laboratory – BFI/UChicago

Supplementary material for the exercises is provided in
OSE2019/day3/Exercise_day3/supplementary_material_omp

Including adapted teaching material from of G. Hager & G. Wellein,
and the Swiss Supercomputing Centre (CSCS)

1. Normalize a vector

normalize vec.f90/normalize_vec.cpp:

- Find code snippets here: day_3/Exercise_day3/supplementary_material_omp
- Write a makefile to compile the file
- Write a parallel version of the subroutine normalize vector(), i.e. normalize_vector_omp(v,n) (go to line 37 of the example).
- Scale the vector by the norm to give length=1.
- Experiment with different OMP_NUM_THREAD counts.
- Play with different sizes of the vector and observe running times.
- Run the code by using an adjusted job submission script (openmp.sh).

2. Dot product

dot prod.f90/dot_prod.cpp:

- This code finds the dot product of two vectors.
- Add OpenMP directives to parallelize the code.
- Write a makefile to compile the file.
- Experiment with different OMP_NUM_THREAD counts.
- Play with different sizes of the vector and observe running times.
- Run the code by using an adjusted job submission script (openmp2.sh).
- How does it scale from 1 to 8 threads? (submit 1 to 8 thread job).
- Try arrays of different length e.g. 10,000 up to 500,000,000 entries.
- How does it affect scaling? (remember Amdahl's Law?).

3. Estimating Pi with Monte Carlo and OpenMP

Let's consider the problem of estimating Pi by utilizing the Monte Carlo method. Suppose you have a circle inscribed in a square (as in the figure). The experiment simply consists of throwing darts on this figure completely at random (meaning that every point on the dartboard has an equal chance of being hit by the dart). How can we use this experiment to estimate Pi?

The answer lies discovering the relationship between the geometry of the figure and the statistical outcome of throwing the darts.

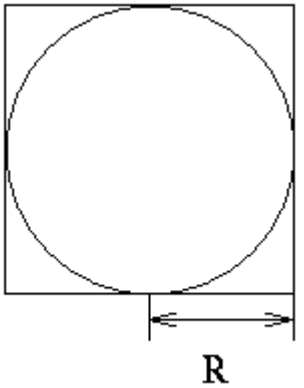
Let's first look at the geometry of the figure. Let's assume the radius of the circle is R, then the Area of the circle = Pi R^2 and the Area of the square = 4 R^2.

Now if we divide the area of the circle by the area of the square we get Pi / 4.

But, how do we estimate Pi by simulation? In the simulation, you keep throwing darts at random onto the dartboard. All of the darts fall within the square, but not all of them fall within the circle. Here's the key. If you throw darts completely at random, this experiment estimate the ratio of the area of the circle to the area of the square, by counting the number of darts in each.

Our study of the geometry tells us this ratio is Pi/4. So, now we can estimate Pi as

Pi = 4 x (Number of Darts in Circle) / (Number of Darts in Square).



→ Revisit this example by adding OpenMP clauses.
→ Experiment with the number of random number you create (N = 100, 1,000, 10,000).
→ Run the code both in batch mode with varying number of threads.

4. Project 1 – Overall tasks

- Write a makefile instead of compiling by hand (add the compile flag -fopenmp).
- Parallelize the problem with OpenMP (for both Europen as well as the Asian option) (look at BS.cpp – take care of the random seeds!).
- **Check the speed-up** for combinations of threads and a variety of **sample points** on one node of MIDWAY (use slurm and the omp_get_wtime()).
- Generate Speed-up graphs (normalize to the 1 CPU result).
- Ensure that the serial and parallel return the same results.

Recall – Pricing an Option

The following algorithm illustrates the steps in simulating n paths of m transitions each. To be explicit, we use Z_{ij} to denote the j-th draw from the normal distribution along the i-th path:

Add OpenMP clauses here. Take care about the random number sequence

for $i = 1, \ldots, n$
  for $j = 1, \ldots, m$
    generate Z_{ij}
    set $S_i(t_j) = S_i(t_{j-1}) \exp\left([r - \frac{1}{2}\sigma^2](t_j - t_{j-1}) + \sigma\sqrt{(t_j - t_{j-1})} Z_{ij} \right)$
  set $\bar{S} = (S_i(t_1) + \cdots + S_i(t_m))/m$
  set $C_i = e^{-rT}(\bar{S} - K)^+$
set $\hat{C}_n = (C_1 + \cdots + C_n)/n$

5. Project 3 – Tasks

- Parallelize the problem with OpenMP (look at the routine solver.cpp).
- **Check the speed-up** for combinations of threads and a variety of **the discretization level** on one node of **MIDWAY** (use slurm).
- Generate Speed-up graphs (normalize to the 1 CPU result).
- Ensure that the serial and parallel return the same results.

Recall DSDP

$$V_{new}(k,\Theta) = \max_{c} \left(u(c) + \beta \mathbb{E}\{V_{old}(k_{next},\Theta_{next})\}\right)$$

$$\text{s.t.} \quad k_{next} = f(k,\Theta_{next}) - c$$

$$\Theta_{next} = g(\Theta)$$

States of the model:

- k : today's capital stock → There are many independent k's
- Θ: today's productivity state → The Θ's are independent

Choices of the model:

- k_{next}

→ k, k_{next} , Θ and Θ_{next} are limited to a finite number of values

solver.cpp

```
for (int itheta=0; itheta<ntheta; itheta++) {

    /*
     Given the theta state, we now determine the new values and optimal policies corresponding to each
     capital state.
     */

    for (int ik=0; ik<nk; ik++) {

        // Compute the consumption quantities implied by each policy choice
        c=f(kgrid(ik), thetagrid(itheta))-kgrid;

        // Compute the list of values implied implied by each policy choice
        temp=util(c) + beta*ValOld*p(thetagrid(itheta));

        /* Take the max of temp and store its location.
         The max is the new value corresponding to (ik, itheta).
         The location corresponds to the index of the optimal policy choice in kgrid.
         */
        ValNew(ik, itheta)=temp.maxCoeff(&maxIndex);

        Policy(ik, itheta)=kgrid(maxIndex);

    }
}
```

loop to worry about with OpenMP