LabX-7, BY 01 December 2021

How to do the exercises in this set

We are now increasing the usage of the bash scripts and pick up additional utility commands. Please consult the material in the Files section of canvas for this course, it may help with the syntax in the examples of the exercises. Use the C book of Kernighan & Ritchie (see Pages/Textbooks on canvas[1]). In canvas Files you have a summary of C language in slides file 3.1_Linux-C[2] . Also use the summary slides in file 3.1_Linux_Bash[3]

In this assignment we consider the OpenMP parallelization. This can be done traditionally by distributing the workload and also by increasing the computational volume as new resources (threads) are added.

On the delivery of your results: When you are ready with your program assignment, please create a directory with your name and copy the files there:

```
mkdir /trinity/home/LINUX_SUPERCOMPUTERS/yourname/   #it should exist already from exe1

mkdir /trinity/home/LINUX_SUPERCOMPUTERS/yourname/labx7

cp yourprogram /trinity/home/LINUX_SUPERCOMPUTERS/yourname/labx7/yourprogram
```

After you have finished copying submit also to canvas, so we can grade the arrivals. You do not have to send us e-mails when you complete this exercise.

Of course, if there are any problems or questions, please email me and/or Andrey:

"Igor Zacharov" I.Zacharov@skoltech.ru

"Andrey Kardashin" Andrey.Kardashin@skoltech.ru

[1] http://cslabcms.nju.edu.cn/problem_solving/images/c/cc/The_C_Programming_Language_(2nd_Edition_Ritchie_Kernighan).pdf

[2] https://skoltech.instructure.com/files/257760/download?download_frd=1

[3] https://skoltech.instructure.com/files/262799/download?download_frd=1

Further commands and Bash structure

Login to your sandbox account: ssh –p 2200 yourname@sandbox.zhores.net

```
sed
diff
sdiff
bc
```

Some additional commands

a) The stream editor, is an automated way to make changes in text files. For example

```
cat blackhol.txt | sed ‘1,10d’ >new.txt
```

will remove the first 10 lines in the file. The argument to sed is a command text very similar to vi with the structure

```
[addr]X[options]
```

Where addr is the specification for the range, X is the (single letter) sed command. In the exercises later in this Lab we use sed to modify the comments in the bash script. For example, the command sed ‘/cpus/s/=.*$/=4/’

Will read the text, grab the line with the pattern cpus and execute the substitute (s) command using the wild card: = (matched exactly) . (any character) * (0 or more occurences) $ (until the end of line) by the exact expression =4

Do this: cat blackhol.txt | sed ‘/H O L E/s/I N/in/’ > blackhol2.txt
to replace on the lines containing ‘H O L E’ the word ‘I N’ with ‘in’.

b) utilities to compare files. From the previous exercise, do

```
diff  blackhol*.txt
```

```
sdiff blackhol.txt blackhol2.txt | more
```

The sdiff command shows the changes per line of text. This is useful in an exercise later in this Lab.

c) Observe a trick in the bc calculator for the truncate function. This is often useful when obtaining integers in bash scripts, since bash doesn’t work with the floating point. Here is the receipt:

```
echo “scale=0; 105.78/1”| bc
```

should display 105 as the result. Grab it into a variable and compare in a bash script, eg.:

```
x=105.78
myvar=$(echo “scale=0; $x/1” | bc)
echo $myvar
```

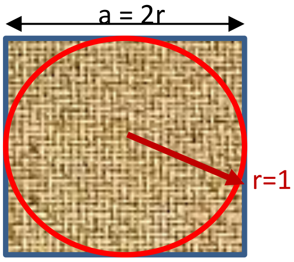
Writing C programs

1) Get the example time measurement program from ~/../LINUX_SUPERCOMPUTERS/programs directory: **timemetric.c**

The task is to add the macro for the dseconds comparison, in analogy with the other definitions and print it out where time difference is printed. The examples for the microsecond metric and the nanosecond metric are given in the program source:

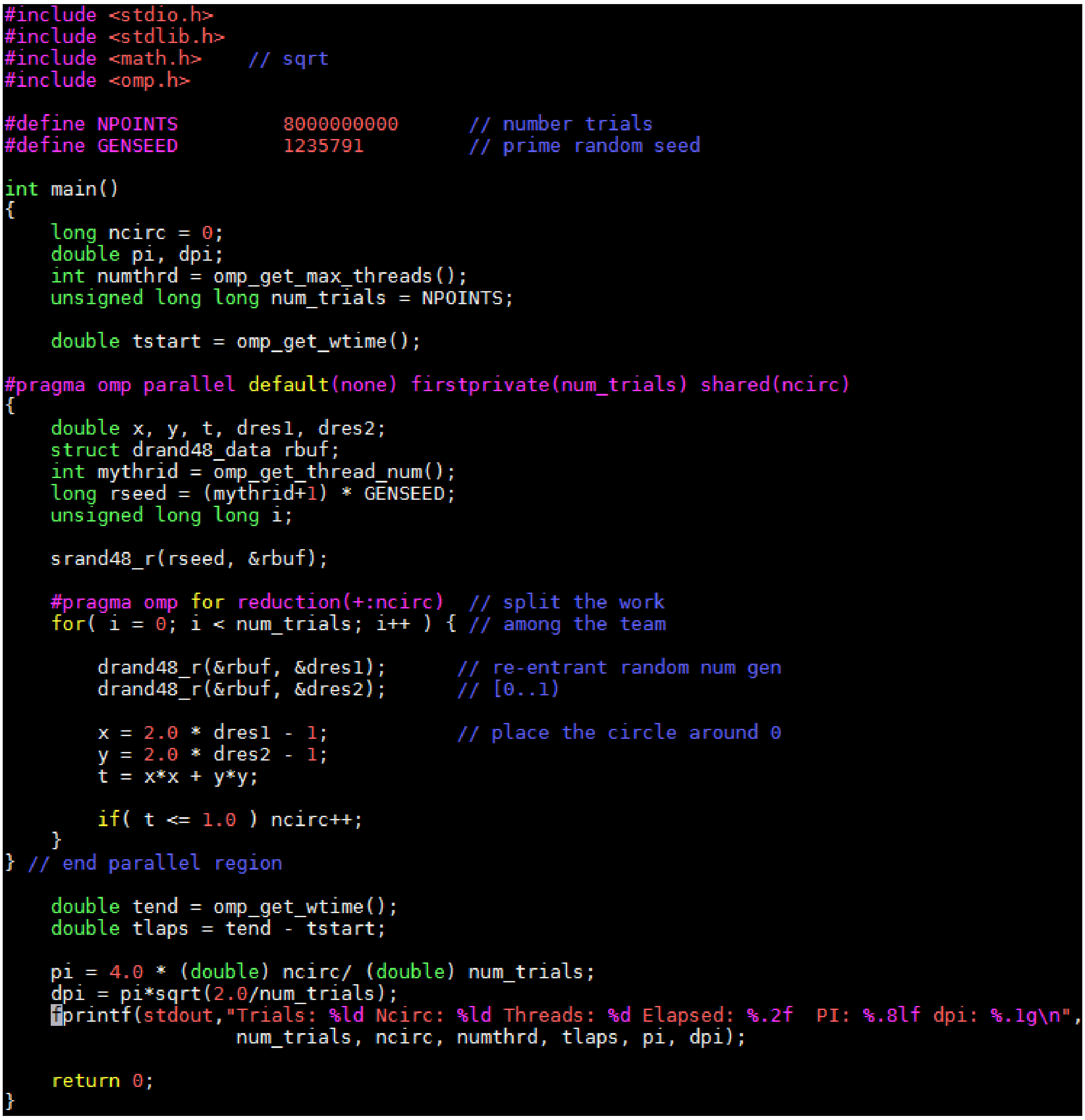
```
#define NANOSEC_in_SEC 1000000000LL

#define dmicrosec(tx,ty) ((double)((ty.tv_sec)-(tx.tv_sec))*1000000.0+(double)((ty.tv_nsec)-(tx.tv_nsec))/1000.0)

#define llnanosec(tx,ty) ((long long unsigned) (ty.tv_sec - tx.tv_sec) * NANOSEC_in_SEC + (long long unsigned)(ty.tv_nsec - tx.tv_nsec))
```

2) Computing pi (=3.1415926…) with random number generator. This methods is based on comparing the area of the square (side=2) with circle (r=1). Consider N uniformly distributed random points covering all of the square and the embedded circle. The area of the square is ~N, while the area on the circle is proportional only to number of points inside the circle N$_c$. Therefore:



$\frac{N_c}{N} = \frac{A_c}{A_s} = \frac{\pi r^2}{(2r)^2}$, therefore $\pi = 4\,N_c/N$, since $r = 1$. This is a very slow method, as the statistical uncertainty is given by: $\left(\frac{\Delta\pi}{\pi}\right)^2 = \left(\frac{\sqrt{N_c}}{N_c}\right)^2 + \left(\frac{\sqrt{N}}{N}\right)^2$ and therefore $\Delta\pi = \pi\sqrt{2/N}$ since $N_c \approx N$ for order of the numbers. The number trials N should grow by 100 to move one significant point in π calculation.

Here is the program called **piprog_A.c** . Compile with gcc –o piprog –fopenmp –O3 piprog_A.c



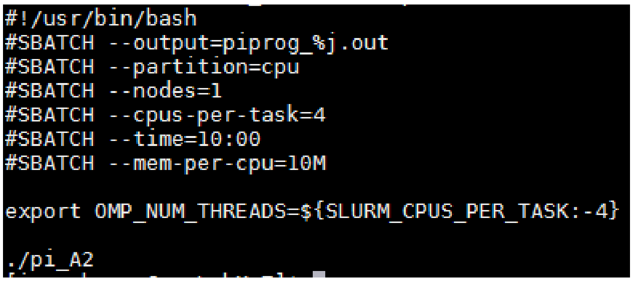
```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>    // sqrt
#include <omp.h>

#define NPOINTS         8000000000      // number trials
#define GENSEED         1235791         // prime random seed

int main()
{
    long ncirc = 0;
    double pi, dpi;
    int numthrd = omp_get_max_threads();
    unsigned long long num_trials = NPOINTS;

    double tstart = omp_get_wtime();

#pragma omp parallel default(none) firstprivate(num_trials) shared(ncirc)
{
    double x, y, t, dres1, dres2;
    struct drand48_data rbuf;
    int mythrid = omp_get_thread_num();
    long rseed = (mythrid+1) * GENSEED;
    unsigned long long i;

    srand48_r(rseed, &rbuf);

    #pragma omp for reduction(+:ncirc)  // split the work
    for( i = 0; i < num_trials; i++ ) { // among the team

        drand48_r(&rbuf, &dres1);      // re-entrant random num gen
        drand48_r(&rbuf, &dres2);      // [0..1)

        x = 2.0 * dres1 - 1;           // place the circle around 0
        y = 2.0 * dres2 - 1;
        t = x*x + y*y;

        if( t <= 1.0 ) ncirc++;
    }
} // end parallel region

    double tend = omp_get_wtime();
    double tlaps = tend - tstart;

    pi = 4.0 * (double) ncirc/ (double) num_trials;
    dpi = pi*sqrt(2.0/num_trials);
    fprintf(stdout,"Trials: %ld Ncirc: %ld Threads: %d Elapsed: %.2f  PI: %.8lf dpi: %.1g\n",
                    num_trials, ncirc, numthrd, tlaps, pi, dpi);

    return 0;
}
```

a) run the program with different settings of the number of threads: export OMP_NUM_THREADS=x, where x=1, 2, 4 and note the time for the parallel region printed as Elapsed in the program. For this, make a script that automates the process, for example:

```
for x in 1 2 4
do
  OMP_NUM_THREADS=$x
  ./piprog
done
```

b) same as (a), but now submit the program to a batch queue by preparing a script in the same directory with a name runpi.sh. Here:

```
sbatch runpi.sh
```



```
#!/usr/bin/bash
#SBATCH --output=piprog_%j.out
#SBATCH --partition=cpu
#SBATCH --nodes=1
#SBATCH --cpus-per-task=4
#SBATCH --time=10:00
#SBATCH --mem-per-cpu=10M

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK:-4}

./pi_A2
```

The run can be monitored with **squeue** command. The output will be in file piprog_JOBID.out, where JOBID is the number from **squeue** output.

The value of cpus-per-task is the number of threads used to run and should be changed for each submission of the script with 1, 2, 4 threads. Change them by hand to see that the output of the program really prints the #threads correctly.

c) automate the submission of the runpi.sh with a script, for example:

```
for x in 1 2 4
do
  cat runpi.sh | sed “/cpus/s/=.*$/=$x/” | sbatch
done
```

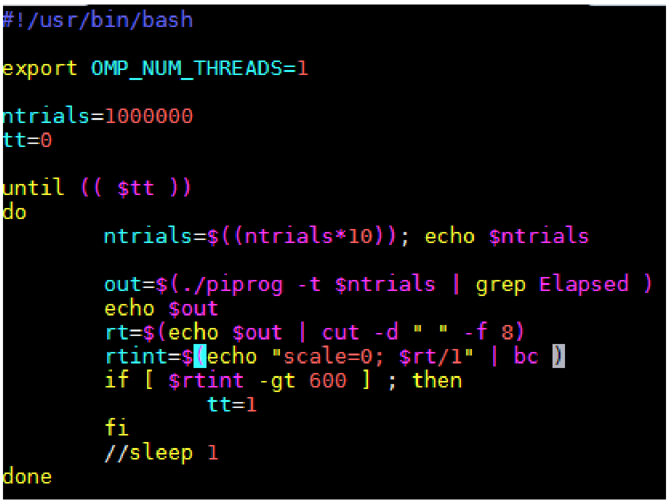
Immediately after the script finishes observe with **squeue** that several jobs have been started in the queue.

d) Compare the runtimes in exercise (a) when run interactively and (b) or (c) when submitted. Fill up the table:

times/#threads	1	2	4	4 (interactive)
Elapsed parallel region				
Real time from time cmd				
User time from time cmd				

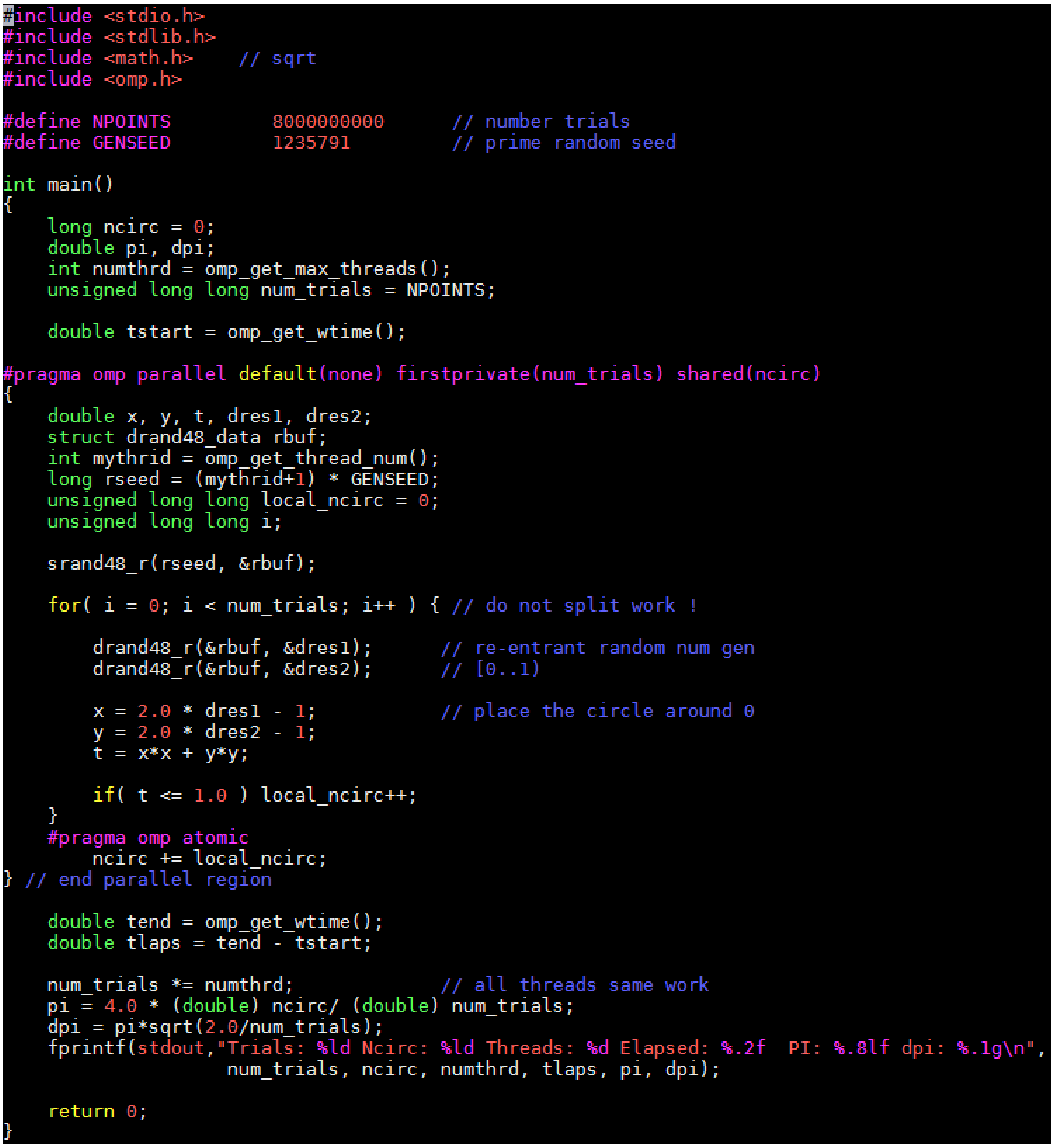
e) modify the program to accept the random generator seed and the number of trials from the command line, thus for the usage: ./prog –s 1235791 –t 8000000000 with appropriate defaults.

Write a script to successively increase the number of trials until about 5 minutes of runtime for 1 CPU. Here is the script (may need more testing, try it out):



```
#!/usr/bin/bash

export OMP_NUM_THREADS=1

ntrials=1000000
tt=0

until (( $tt ))
do
        ntrials=$((ntrials*10)); echo $ntrials

        out=$(./piprog -t $ntrials | grep Elapsed )
        echo $out
        rt=$(echo $out | cut -d " " -f 8)
        rtint=$(echo "scale=0; $rt/1" | bc )
        if [ $rtint -gt 600 ] ; then
                tt=1
        fi
        //sleep 1
done
```

3) An alternative way to parallelize a program is to increase the computational volume with each thread. Same program as before, but now not distributing the workload in the for-loop. All threads do same amount of work and cooperate only on building up the N_c . Here is the program **piprog_G.c**



```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>    // sqrt
#include <omp.h>

#define NPOINTS         8000000000      // number trials
#define GENSEED         1235791         // prime random seed

int main()
{
    long ncirc = 0;
    double pi, dpi;
    int numthrd = omp_get_max_threads();
    unsigned long long num_trials = NPOINTS;

    double tstart = omp_get_wtime();

#pragma omp parallel default(none) firstprivate(num_trials) shared(ncirc)
{
    double x, y, t, dres1, dres2;
    struct drand48_data rbuf;
    int mythrid = omp_get_thread_num();
    long rseed = (mythrid+1) * GENSEED;
    unsigned long long local_ncirc = 0;
    unsigned long long i;

    srand48_r(rseed, &rbuf);

    for( i = 0; i < num_trials; i++ ) { // do not split work !

        drand48_r(&rbuf, &dres1);      // re-entrant random num gen
        drand48_r(&rbuf, &dres2);      // [0..1)

        x = 2.0 * dres1 - 1;           // place the circle around 0
        y = 2.0 * dres2 - 1;
        t = x*x + y*y;

        if( t <= 1.0 ) local_ncirc++;
    }
    #pragma omp atomic
        ncirc += local_ncirc;
} // end parallel region

    double tend = omp_get_wtime();
    double tlaps = tend - tstart;

    num_trials *= numthrd;             // all threads same work
    pi = 4.0 * (double) ncirc/ (double) num_trials;
    dpi = pi*sqrt(2.0/num_trials);
    fprintf(stdout,"Trials: %ld Ncirc: %ld Threads: %d Elapsed: %.2f  PI: %.8lf dpi: %.1g\n",
                    num_trials, ncirc, numthrd, tlaps, pi, dpi);

    return 0;
}
```

a) observe the differences with the command: sdiff piprog_A.c piprog_G.c

b) make the Elapsed times table with different threads by submitting into the batch with 1, 2, 4 threads as follows (choose fixed number trials such that it runs for about 5 mins for 1 thread):

category/#threads	1	2	4
Elapsed time [s]			
value pi			
Error in pi			