```
*****************************************************************************
```
Authors : Albert Owusu-Asare , Box 4497, <owusuasa@grinnell.edu>
: Ezra Edgerton , Box 3503, <edgerton@grinnell.edu>

This document contains answers to questions from CS213 Lab: Threads

Questions:
http://www.cs.grinnell.edu/~weinman/courses/CSC213/2014F/labs/threads.html

```
*****************************************************************************
```

Answers

Relevant information:

Page size: 4096

Number of processors available (nproc) = 4;
processor 0:
cpu MHz = 1600.000
cache size = 3072 KB
cpu cores = 4
address sizes : 36 bits physical, 48 bits virtual

processor 1:
cpu MHz = 1600.000
cache size = 3072 KB
cpu cores = 4
address sizes : 36 bits physical, 48 bits virtual

processor 2:
cpu MHz = 1600.000
cache size = 3072 KB
cpu cores = 4
address sizes : 36 bits physical, 48 bits virtual

processor 3:
cpu MHz = 1600.000
cache size = 3072 KB
cpu cores = 4
address sizes : 36 bits physical, 48 bits virtual

Part B
---

3)

a.

The average times taken for a set of threaded processes to run a matrix of
size N is the average time for 1 thread divided by the number of threads in the pro
cess.
As a result the plots of the different threaded processes showed less
growth in average time(seconds) vs matrix size as the number of threads rose.

As we increase the number of threads, we are able to obtain more
concurrency because as we saw from nproc, we have 4 available processors on
the computer. The more threads we use, the less number of rows the thread
is responsible for calculating in the matrix and the greater number of rows
can be computed at a time; thus we have less overall time calculating
the matrix. In the case of 4 threads being used, each thread uses a
processor to calculate the rows that it is responsible for so the average
run time is 1/4 of the one threaded average run time. This holds for the 2
and 3 threaded runs being 1/2 and 1/3 respectively of the run time of one thread as
well.

b. The trend in speedup as N increases is that the differently threaded
speedups hold more or less constant. Two Threads holds at just below 2,
Three threads holds at a little more below 3 than 2 threads held below 2,
and four threads holds at a little more below 4 than 3 threads held below
3.The difference from the numbers that the speedups were just below is
caused by the overhead from creating and running the threads, the overhead
will increase when we create more threads, bringing the speedup time below
the round number that just running the program with overheadless threads
would be. 4 threads is the farthest below 4 because the 4 threads would
have the most overhead.

The best answer we could come up with is that the deviations at the
beginning must have something to do with the page or cache size. That at an
N of between 256 and 384, each row would pass the limit of the
page size this would cause a larger number of tlb misses and slow down the
process significantly. Before this, the tlb misses of each thread would
decrease as the number of threads increased, so the speedup time for runs
with greater threads would increase more quickly.

c.
Efficiency seems to remove the advantage that the speedup metric measures
by dividing by the number of threads, it seems to measure the differences
in overhead, because with one thread, the efficiency is one. As N
increases, we see that 2 threads is the most efficient, holding at a little
below 1, while 3 and 4 threads are less than the smaller number of
threads. This is because, as we said in b, the overheads for more threads
increases as we build more threads.

At smaller values of N, the deviation from the trend is the same as our
answer for part b. However, we see that 4 threads goes above 1 in
efficiency at N = 256. This must be due to a reason similar to what was
mentioned in b.

d.
As P increases the speedup increases linearly at a rate of slightly less
than 1. As we said previously, the overhead causes the slight decrease in
growth below 1. Without the overhead of thread creation and joining, the growth
would be at a constant increase of one because the number of threads decreases
the run time of the matrix multiply at that rate, as we explained in 3a
above. More threads mean more concurrent processing of rows, and as a
result a faster overall calculation of the matrix.

e. As P increases the efficiency decreases linearly at a rate of roughly -.025. Thi
s measures the
amount of time that is spend in overhead for each increase in threads. The
more threads we have, the more time we spend creating and joining them, and
that cancels out a small amount of the speedup effect.