Homework 1

Philip Nelson

20th September 2021

Question 1

In following program example we were assuming that each call to Compute_next_value requires roughly the same amount of work in each loop iteration. For your reference:

```
my_sum = 0;
my_first_i = ... ;
my_last_i = ... ;

for (my_i = my_first_i; my_i < my_last_i; my_i++) {
   my_x = Compute_next_value(...);
   my_sum += my_x;
}
```

Let's now consider instead that a call when $i = k$ requires $k + 1$ units of time as the call when $i = 0$. Example: if the first call ($i = 0$) requires 1 millisecond, the second call ($i = 1$) requires 2, the third ($i = 2$) requires 3, and so on.

How much time each core will spend in the loop calling the function Compute_next_value if $n = 15$ and $p = 3$? Where n is the number of elements in the loop and p in the number of available cores.

p0 = 15 ms
p1 = 40 ms
p2 = 65 ms

Question 2

In the previous question, the load among the cores is not balanced anymore. How would you balance it?

Answer devising an algorithm (written in pseudo-code) which produces the following output:

1. an array containing the total amount of time (i.e., ms) spent on each core
2. a bi-dimensional array containing the indices of the iterations (i) you assign to each core

Finally, report a numerical example (i.e., content of the arrays).

The load could be balanced by assigning processes tasks in a "striped" manner such that each process had some of the shorter and longer tasks.

```
def striped_partition(n, p):
  indices = [[] for _ in range(p)]
  total_work = [0 for _ in range(p)]
  proc = 0
  for i in range(n):
    indices[proc].append(i)
    total_work[proc] += i + 1
    proc = (proc + 1) % p
  return total_work, indices


total_work, indices = striped_partition(n=15, p=3)
print('1.', total_work)
print('2.', indices)


1. [35, 40, 45]
2. [
     [0, 3, 6, 9,  12],
     [1, 4, 7, 10, 13],
     [2, 5, 8, 11, 14]
   ]
```

This does not always provide a perfectly balanced load, however; in general I believe this is the bin packing problem which is NP-hard so a decent solution is all we want.

Question 3

Derive formulas for the number of receives and additions that core 0 and core 4 carry out using

1. the original pseudo-code for a global sum, and
2. the tree-structured global sum.

Assume n (number of cores) is always a power of two.

Finally, report the numbers of receives and additions carried out by core 0 and core 4 (in two different tables) when the two methods are used with 2, 4, 16, 512 and 1024 cores.

1

Original global sum: each process sends their value to rank 0 and rank 0 adds all the values
$N_0 = n - 1$
$N_4 = 0$

2

Tree structured global sum
$N_0 = log_2(n)$

$$N_0 = \begin{cases} log_2(n) - 1 & \text{if } x < 8 \\ 4 & \text{if } x >= 8 \end{cases}$$

	Original - 0	Original - 4
2	1	0
4	3	0
16	8	0
512	511	0
1024	1023	0

	Tree - 0	Tree - 4
2	1	0
4	2	1
16	4	4
512	9	4
1024	10	4

Question 4

Suppose that your professor is organizing a pizza party for all the students and need your help to prepare things faster and clean up afterwards.

1. Identify tasks for students that will allow them to use "task-parallelism"
2. How can we use "data-parallelism" to partition the work of cleaning
3. How could you use a combination of the previous approaches?

Note: the use of any type of fruit on the pizza is strongly discouraged!

1

Tasks for the pizza party

- prepare for party
  - order pizza (no Hawaiian)
  - buy drinks, plates, cups
  - organize room
  - prepare music
  - prepare games
- clean up after party
  - wipe down tables
  - take out trash

2

Each table is assigned a student so the tables are wiped down in parallel. Then each trashcan is assigned a student so the trash is taken out in parallel.

3

Since wiping down tables and taking out trash are orthogonal operations, they can be performed simultaneously. Tables, and trash cans are assigned students and they are wiped down and taken out in parallel. There are likely more tables than trashcans, but wiping down a table takes less time than taking out a trashcan, so multiple tables are assigned to the same student and the load remains balanced.

Question 5

1. Explain how a write-through cache works.
2. If we implement in hardware a queue in the CPU, how could this be used to improve the performance of a write-through cache?

1

With wright-through cache, when a value is written to the cache, it is also written to the main memory.

2

Writing to main memory is slow compared to writing to cache, so a queue could be used to store waiting write-through operations and free the CPU to continue operating instead of waiting on the write-through operation to finish.

Question 6

Recall the example involving cache reads of a two-dimensional array. For your reference:

```
/* First pair of loops */

for (i = 0; i < MAX; i++)
  for (j = 0; j < MAX; j++)
    y[i] += A[i][j]*x[j];

/* Assign y = 0 */

/* Second pair of loops */
```

```
for (j = 0; j < MAX; j++)
  for (i = 0; i < MAX; i++)
    y[i] += A[i][j]*x[j];
```

How does a larger matrix and a larger cache affect the performance of the two pairs of nested loops?

What happens if `MAX = 8` and the cache can store four lines?

Hint: draw a table representing the cache lines, assume each cache line contains 4 elements of the array.

1

On a per iteration basis, a larger matrix will not impact the performance of the first loops. It might have a cache miss every time the outer loop goes around but this will be insignificant compared to the number of cache hits it has every time it reads the next value on the same row. For the second loops, a larger matrix will only accentuate the issue. Every time the inner statement is executed, there will be a cache miss. A larger matrix will only have more cache misses.
A larger cache could have a small impact on the second loops if it were large enough to hold multiple rows of the matrix. This would reduce the number of cache misses and increase the performance. For the first loops, a larger cache wouldn't do as much. It would require less cache reads which would improve performance but I don't think it would reduce cache misses unless it could hold multiple rows of the matrix.

2

If the following is stored in cache for the `ij`'th iteration of the loop then the second loops do not have a cache miss when going from the `[i][j]`'th iteration to the `[i+1][j]`'th iteration because both `A[i][j]` and `A[i+1][j]` are in the cache. After this, however, I think there would be a cache miss on the `A[i+2][j]`'th iteration but that would likely load the row with `A[i+3][j]` effectively reducing the cache misses by a factor of 2

A[i][j+0]	A[i][j+1]	A[i][j+2]	A[i][j+3]
A[i][j+4]	A[i][j+5]	A[i][j+6]	A[i][j+7]
A[i+1][j+0]	A[i+1][j+1]	A[i+1][j+2]	A[i+1][j+3]
A[i+1][j+4]	A[i+1][j+5]	A[i+1][j+6]	A[i+1][j+7]

Question 7

Discuss the differences in how a GPU and a vector processor might execute the following code:

```
sum = 0.0;
for (i = 0; i < n; i++) {
  y[i] += a*x[i];
  sum += z[i]*z[i];
}
```

A GPU might assign each iteration of the loop to a different core and execute them in parallel.
A vector processor might load a, x[i], and z[i] into vector registers and perform the multiplications in parallel.

Question 8

A parallel program that obtains a speedup greater than p —the number of processes or threads—is sometimes said to have superlinear speedup . However, many authors don't count programs that overcome "resource limitations" as having superlinear speedup. For example, a program that must use secondary storage for its data when it's run on a single processor system might be able to fit all its data into main memory when run on a large distributed-memory

system. Give another example of how a program might overcome a resource limitation and obtain speedups greater than p .

A program that is distributed across multiple machines may benefit from concurrent disk reads/writes which makes it less likely to become I/O bound.

Question 9

In a Partitioned Global Address Space (PGAS) language, the collective memory of all running processes is treated as a single global address space. It can be accessed by any process, however; locations that are not local to the process can take much longer to access. For this reason, PGAS languages provide mechanisms for explicitly declaring variables that will exist locally so the programmer is aware of the variable locality.

In a Message Passing (MP) language, each process run's a copy of the program in its own isolated memory space. Variables declared in the program are always in local memory and data is passed between processes using the explicit message passing API provided by the language or library.

	Pros	Cons
PGAS	higher level abstraction visible to the compiler (with varying degrees, depends on if the PGAS features are part of the language (fortran co-arrays) or a library (UPC++)	programming at a higher level can mean less control over how things work "under the hood", such as message passing, which can lead to inefficiencies
MP	widely used provide api for additional operations: broadcast, reduce, etc	cumbersome invisible to compiler

Question 10

Suppose that a vector processor has a memory system in which it takes 10 cycles to load a single 64-bit word from memory. How many memory banks are needed so that a stream of loads can, on average, require only one cycle per load?

10