CSCI-UA.0480-003

Parallel Computing

Homework Assignment 1

Name: Meisi LI

NetID: ml6095

Question 1:

a. n is divisible by p and n>p:

It is considered that the range is closed in my_first_i and opened in my_last_i.

For my_first_i: the index of the my_rank multiplied by the multiplied by the division between n and p. This means the expression is the *index of nucleus * n / p*.

For my_last_i: equals to my_first_i plus the n / p. In this condition, the *my_last_i = my_first_i + n / p*.

For example: n = 24, p = 3,

the first core: my_first_i = 0*24/3 = 0, my_last_i = 0+24/3 = 8;

the second core: my_first_i = 1*24/3 = 8, my_last_i = 1+24/3 = 16;

the third core: my_first_i = 2*24/3 = 16, my_last_i = 2+24/3 = 24;

b. n is not divisible by p:

It is more complicated for this situation. Assume k = n % p(0 < k < p-1), index of nucleus = i.

From 0 to k-1 in my_rank: my_first_i = i * ((n-k) / p + 1);

my_last_i = (i+1) * ((n-k) / p + 1);

From k to p-1 in my_rank: my_first_i = i * (n-k) / p + k;

my_last_i = (i+1) * (n-k) / p + k;

Question 2:

a. Derive a formula for the number of receives and additions that core 0 does in the first (non-tree) method

The first method is a fairly obvious generalization of the serial global sum: divide the work of adding among the cores and computed the self-sum. core 0 will receive the result of each of every core and will add each one. Therefore, the amount of data received will be equal to the amount of sums made. Thus, the number = (p – 1); where p is the number of cores.

b. The tree method

The master core will receive the value of the partial sums and sum, obtaining the total value. Therefore the amount of numbers received and added will be smaller than the cores. For 2 cores, the result will be 1, for 4 will be 2, for 8 will be 3. So the formula = log2(p); where p is the number of cores.

c.

Cores	Non-tree method	Tree method
2	1	1
4	3	2
8	7	3
16	15	4
32	31	5
64	63	6
128	127	7
256	255	8
512	511	9
1024	1023	10

d. Which operation do you think is more expensive: receive or addition? and why?

I think addition is more expensive than receive. The receive operation is a kind of communication between cores. For example, each core adds its assigned computed values in the first method and then the cores send their partial sums to the master core which adds them together. Each core works with its thread and

in the ring as a shared memory, all cores could write and read in each part of the ring. For addition operation, it is the cheap operation on the CPU but it has to get 2 numbers and change to the new result. Therefore, addition operation could ask more CPUs to finish. Thus, addition is more expensive than receive.

Question 3: If you are given a sequential program that you are required to parallelize, first you need to find the parts that are parallelizable. However, in some cases, it is not worth it to parallelize those parts, why?

In some cases, there is no simple way to parallelize the algorithm. The parallel algorithm can be essentially different from the sequential one. For example, sorting algorithms have a complexity O(n log n), but the parallel sort has complexity O(n log^2 n).

And in some case, the sequential code may look not parallel but the algorithm actually has parallelism. It is not necessary to parallelize the this code.

Question 4: There are several types of parallelism. What are they? For each one, given a definition, and specify whether exploiting that type needs programmer involvement or the hardware/compiler are enough to exploit it.

Parallelism in hardware: (No programmer involvement)

- Parallelism in a Uniprocessor: a uniprocessor system can perform 2 or more tasks simultaneously. A system that processes 2 different instructions simultaneously could be considered to perform parallel processing.
  - Pipelining

  - Superscalar
  - Out-of-order execution
  - Speculative execution
  - Simultaneous Multithreading, etc
- SIMD: parallelism achieved by dividing data among the processors(called data parallelism)
  - GPUs
  - Vector processors
- MIMD: Typically consist of a collection of fully independent processing units or cores, each of which has its own control unit and its own ALU
  - Multicore processors
  - Multiprocessor systems

Parallelism in software: (With programmer involved)

- Task-level (e.g. Embarrassingly parallel): Break application into tasks, decided offline
- Divide and conquer: is an algorithm design paradigm based on multi-branched recursion. A divide and conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly.
- Pipeline: A series of ordered but independent computation stages need to be applied on data
- Iterations(loops), Client-server, Hybrid, etc.

Question 5: What is the reason we have distributed memory architectures?

Processors have their own local memory. Memory addresses in one processor do not map to another processor and each processor operates independently.

Here are some advantages:

- Memory is scalable with number of processors. Increase the number of processors and the size of memory increases proportionately.
- Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain cache coherency.
- Cost effectiveness. They can use commodity, off-the-shelf processors and networking.

Question 6:

a. Yes. This loops is parallel and can be parallelized.

This loops satisfies simple rule of thumb:

- All assignments (to shared data) are to arrays
- Each element is assigned by at most one iteration
- No iteration reads elements assigned by any other iteration

b. It depends.