Problem set 2, Part 1

Problem 1

Consider a parallel program working on a n × n 2D array of data. The program is run on a cluster with p = qr nodes, where q and r are even integers. When the array is subdivided, each node thus gets a rectangular subdomain of size n/q × n/r

---

a) Find an expression for the total number of grid elements which must be sent when the grid is distributed and gathered.

n rows and columns are distributed to p nodes which can perform q rows and r columns.

Considering one cluster to be local (process 0), we need to send:

$$grid_{sent} = \frac{n^2}{(q * r)} - 1$$

After distribution and calculation on all available nodes, we need to gather all data giving a total number of grid sendings:

$$total\,grid_{sent} = 2(\frac{n^2}{(q * r)} - 1)$$

b) Find an expression for the total number of grid elements which must be sent (by all the processors combined) when the processors perform a periodic border exchange. (That is, when each processors sends the border of its subdomain to each of its four neighbours, (wrapping around at the edges)).

n_i : Number of sending for node with i neighbouring nodes

Number of sent grids:

$$grid_{sent} = 2 * n_2 + 3 * n_3 + 4 * n_4 \tag{1}$$

Two neighbours yield for corner nodes (number of corners are constant):

$$n_2 = 4 \tag{2}$$

Three neighbours:

$$n_3 = 2(q-2) + 2(r-2) \tag{3}$$

Four neighbours:

$$n_4 = (r-2) + (q-2) \tag{4}$$

Combining the calculation of neighbours in equation (1) gives:

$$grid_{sent} = 2 * 4 + 3 * (2((q-2) + (r-2))) + 4 * ((r-2)(q-2))$$

$$\begin{aligned} &= 4qr - 2q - 2r \\ &= 2(2qr - q - r) \end{aligned} \tag{5}$$

c) Assume that each processor can perform full-duplex communication with at most one other processor at a time, that multiple pairs of processors can communicate at the same time, and that the processors can communicate at b bytes/second with a startup time of s. Find an expression for the time required to perform a periodic border exchange, assuming each grid element is d bytes.

b : bytes/second

s : startup (second)

d : grid elements bytes

$$t_{periodicEx} = grid_{sent} * t_{gridSend} \tag{6}$$

$$t_{gridSend} = s + \frac{d}{b} \tag{7}$$

$$t_{periodicEx} = 2 * (2qr - q - r)(s + \frac{d}{b}) \tag{8}$$

d) Consider the case where p = 64. Which values of q and r minimize the cost of computation?

$$q = \frac{p}{r} \tag{9}$$

Set in (9) in (8):

$$t_{periodicEx} = (4p - 2pr^{-1} - r)(s + \frac{d}{b}) \tag{10}$$

Derivate with respect to r and set equal to zero

$$\frac{dt}{dr} = (\frac{2p}{r^2} - 2) = 0 \tag{11}$$

$$r = \sqrt{p} \tag{12}$$

For $p = 64$ minimum cost gives

$$q = r = 8 \tag{13}$$

Problem 2

Consider the parallel odd-even sort described in Section 3.7.2 of Pacheco.

---

a) Find the speedups and efficiencies of the program. Select some appropriate problem sizes and number of processes.

Time serial sorting numbers (fastest sorting):

$$T_s = O(n * log(n)) \tag{14}$$

To sort in parallel it reqires $O(\frac{n}{p}log(\frac{n}{p}))$ for sorting and $O(n)$ for merging and processor communication.

$$T_p = O(\frac{n}{p}log(\frac{n}{p})) + O(n) \tag{15}$$

Speedup:

$$S = \frac{T_s}{T_p}$$

$$= \frac{O(nlog(n))}{O(\frac{n}{p}log(\frac{n}{p}) + O(n))} * \frac{\frac{1}{O(nlog(n))}}{\frac{1}{O(nlog(n))}}$$

$$= \frac{1}{O(\frac{\frac{n}{p}log(\frac{n}{p})}{nlog(n)}) + O(\frac{n}{nlog(n)})} = \frac{1}{O(\frac{1}{p}\frac{log(\frac{n}{p})}{log(n)}) + O(\frac{1}{log(n)})} \tag{16}$$

Efficiency:

$$E = \frac{p * T_s}{T_p}$$

$$= \frac{1}{O(\frac{log(\frac{n}{p})}{log(n)}) + O(\frac{p}{log(n)})}$$

$$= \frac{1}{1 - O(\frac{log(p)}{log(n)}) + O(\frac{p}{log(n)})} \tag{17}$$

b) Does the program obtain linear speedups? Explain why/why not.

Equation (16) shows that both number of processors p and problem size p gives both linear and logarithmic growth which suggest that speedups will not be linear.

c) Is the program weakly scalable? Explain why/why not.

Equation (17) shows that the sorting function is most efficient for p = log(n) suggesting that the program is most suitable for low number of processors. This limits the possibility of scaling up the program with more processors, leaving it weakly scalable.

d) Is the program strongly scalable? Explain why/why not.

As the optimal cost is given for a low number of processors (p = log(n)) suggesting that the program will be weakly scalable.

Problem 3

What are two of the main differences between threads and processes, in general?

- A process is an executing instance of a program which can contain multiple threads. Threads are generally smaller processes in complexity than processes.
- A thread has shared memory while a process has private memory blocks

Problem 4

Consider the thread based global sum example described in Section 4.5 of Pacheco. Explain why using private variables, and moving the busy-wait flag lock outside the loop improves the performance.

Using private variables and moving the busy-wait flag lock outside loop will make sure that the threads less often try to access the flag and block other threads. This will improve the performance drastically as the local threads will run more seamlessly without having to wait for other threads.

Problem 5

a) What is a race condition?

When threads or processors attempt to simultaneously access a resource, and the accesses can result in an error, the threads have a race condition. A race condition will leave the computation output depend on which thread wins the race.

b) What is a critical section?

A block of code that can only be executed by one thread at a time is called a critical section.

c) Name three schemes for protecting access to critical sections, and list two pros and cons of each.

- Mutex
  - Pros
    - Fully protects the access to the critical section
    - Has support in the underlying hardware
  - Cons
    - Enforces serialisation to the critical section
    - No way of controlling accessing order for different threads
- Busy-waiting
  - Pros
    - Simple to understand and implement
    - Known order of thread access
  - Cons
    - Can be very wasteful of system resources
    - Repeated checks of flag
- Semaphores
  - Pros
    - Exists some easier thread synchronisation than in mutex
    - No ownership associated with semaphores (compared to mutex)
  - Cons
    - Enforces serialisation to the critical section
    - No way of controlling accessing order for different threads