Parallel Computing: Exercise session 1

Emil Løvbak & Sahar Chehrazad

19 October 2020

This exercise session consists of pen-and-paper exercises and covers the following topics:

- Ahmdal's law and limitations to parallizing software
- Communication overhead
- Common pitfalls in parallel software
- Data partitioning

For questions you can contact Emil Løvbak (`emil.loevbak@kuleuven.be`) or Sahar Chehrazad (`sahar.chehrazad@kuleuven.be`).

Question 1: Ahmdal's law

Consider the following segment of code:

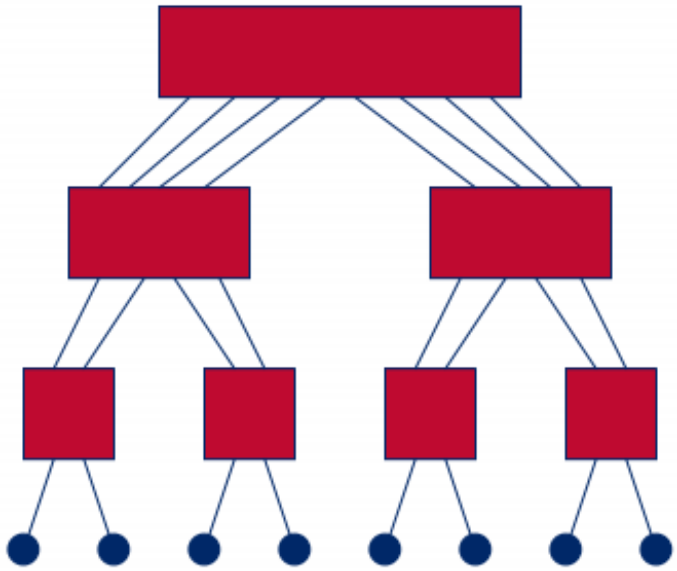
```
1  int n = 10000;
2  int A[n];
3  A[0] = 1;                    // 1e-5 ms
4  A[1] = 1;                    // 1e-5 ms
5  for (int i=2; i<n; i++)
6    A[i] = A[i-1] + A[i-2];    // 1e-5 ms
7  for (int i=0; i<n; i++) {
8    A[i] += 4;                 // 1e-5 ms
9    printf("%d ", A[i]);       // 0.01 ms
10 }
11 sumA = 0;                    // 1e-6 ms
12 for (int i=0; i<n; i++)
13   sumA += A[i];              // 1e-5 ms
14 printf("%d", sumA);          // 0.01 ms
```

1. Which operations can easily be parallelized? Which cannot?
2. What is the serial runtime of this code?
3. What would the runtime be on 20 cores? What speedup is achieved? (Ignore parallel overhead and reduction operations)
4. What would the runtime be on an infinite number of cores? What speedup is achieved?
5. Can we modify the code to make this better?

What can we conclude?

Question 2: The cost of parallelization

Consider that we are running some parallel program on the following architecture, where each link has a communication latency of 1 ms and a bandwidth $B = 10\ 000$ MB/s:



The program consists of two stages of computation, separated by an all-to-all communication step, where each core communicates directly with each other core. The first computation takes each core $T_1 = 10$ ms and the second computation takes each core $T_2 = 2$ ms. The total amount of data in the computation N is 100 MB. The data is distributed over the $p = 8$ cores. Assume that a switch can send data while receiving data. Also assume that a link is unidirectional, i.e., data cannot pass from A to B while it is also passing from B to A, but once a link is open A and B can alternately send each other data without incurring extra latencies.

1. How long does the program take to run?
2. How long would a serial program take to run on the same hardware? (Ignore caching effects.)
3. What percentage of the parallel runtime is due to communication?
4. What happens if the total amount of data in the computation is 1000 MB, assuming both computations are $\mathcal{O}(N)$?
5. What percentage of the parallel runtime is then due to communication?
6. What changes in the 100 MB case if there is only one link between each pair of switches?
7. **What can we conclude?**

Question 3: Problems with loops

Consider the following code segments. What issues can you see in each case?

```
1 for (int i=0; i<n; i++)
2   for (int j=0; j<n; j++)
3     C[i][j] = 0.0;
4     #pragma omp parallel for
5     for (int k=0; k<n; k++)
6       C[i][j] += A[i][k]*B[k][j];
```

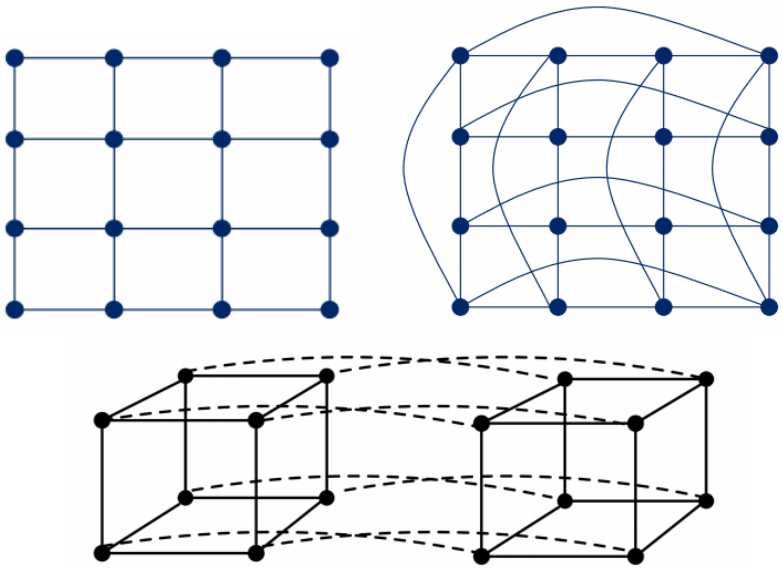
```
1 for (int i=0; i<n; i++)
2   #pragma omp parallel for
3   for (int j=0; j<n; j++)
4     B[i][j] = A[i][j];
```

```
1 #pragma omp parallel for
2 for (int i=0; i<n; i++)
3   for (int j=0; j<i; j++)
4     B[i][j] = [i][j];
```

```
1 #pragma omp parallel for
2 for (int i=0; i<n; i++)
3   A[i] += 2.3;
```

Question 4: Bandwidth

Consider the following graphs where each link has a communication latency ℓ and a bandwidth B:

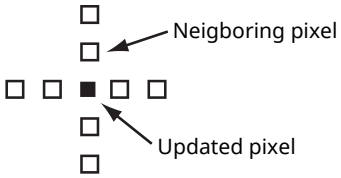


Answer the following questions for both a general number of cores and the cases given. (You can assume that the number of cores p is a "nice" number.)

1. What are the maximal and minimal communication latencies?
2. What are the bisection bandwidths?
3. Which graph in general has the highest connectivity?
4. Which graph in general is the least prone to failure?

Question 5: Image processing

A digital image is given as a matrix of $n \times n$ pixels with 3 numbers per pixel (R-G-B values). An iterative algorithm performs in each iteration step per pixel 50 operations ($+$, $\times$, ...) which involves the pixel values of 8 neighbours (see figure). The algorithm runs on a parallel distributed memory computer with 16 processors.



$Mflop$ rate	$166 Mflops$
startup time (t_s)	$50.00 \mu s$
time per word (t_w)	$0.10 \mu s$

The image size is 624×624 (i.e. n=624). How will you partition the data? Compute the speed-up S and the parallel efficiency E) if communication can be neglected, and ii) if communication cannot be neglected, taking into account the given $Mflop$ rate, startup time and communication time per word. How would you partition the data if the startup time is very large? **What can you conclude?**