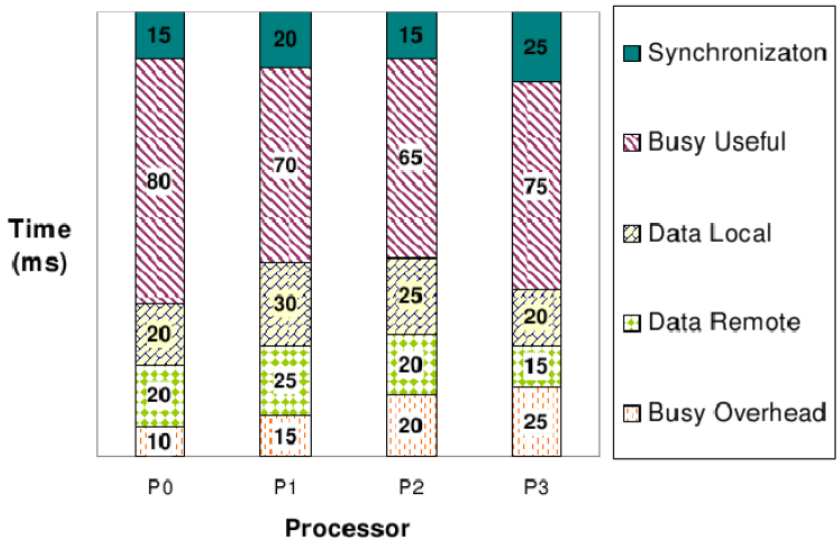
1- A uniprocessor application is parallelized for 4 processors, yielding a 3.8x speedup. Given the time breakdown of the various function seen in the graph, what is the minimum total time that the uniprocessor application spent while Busy and in performing Data Access?



From the notes:

$$\text{Speedup} < \frac{\text{Sequential Work}}{\textit{Max}\,(\text{Work} + \text{Synch Wait Time} + \text{Comm Cost} + \text{Extra Work})}$$

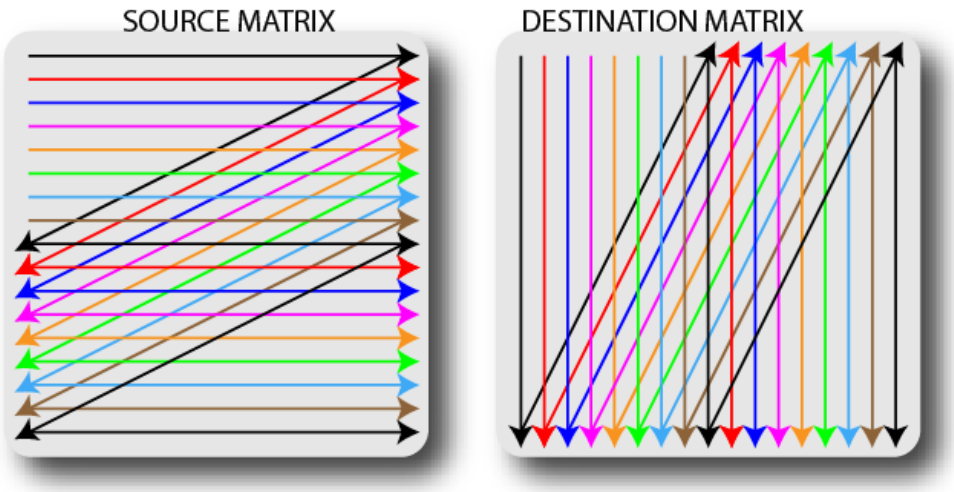
Work for various processors:
P1: 15+80+20+20+10 = 145ms
P2: 20+70+30+25+15 = 160ms
P3: 15+65+25+20+20 = 145ms
P4: 25+75+20+15+25 = 160ms

So
Speedup ≤ Sequential Work / Max(145, 160, 145, 160)ms
Speedup ≤ Sequential Work / 160 ms
3.8 ≤ Sequential Work / 160 ms
Sequential Work ≥ 3.8 * 160ms
Sequential Work ≥ 608ms

So the minimum time a uniprocessor would have spent on that task would have been 608ms

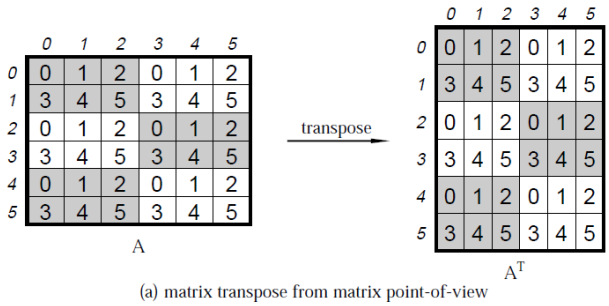
2-Consider transposing a matrix in parallel from a source matrix to a destination matrix.
a- How might you partition the two matrices among processes? Discuss some possibilities and the trade-off.

There would be many ways of splitting the work amongst processors. The first method would be to split the work in manner of row/columns.

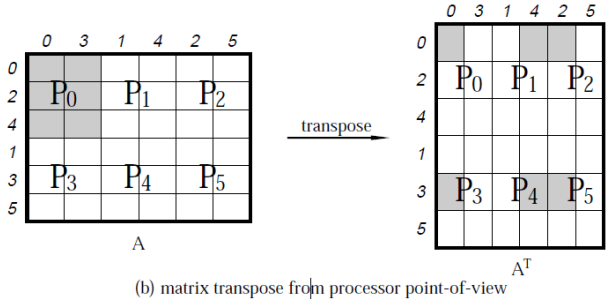


In the initial matrix, a processor would commit to transposing a particular row into a particular column in the target matrix. Whenever a process is done, it could go ahead and commit to the next available row/column pair until the entire matrix has been processed.
Evaluating communication complexity here, we can see that every processor, while performing their first pass, would have to launch a message to every other processor twice.

A second way of splitting the work would be in blocks. In this case, every processor would access a block and transpose the data it contains to the target block. This case would also require a lot of inter processor communication.



(a) matrix transpose from matrix point-of-view



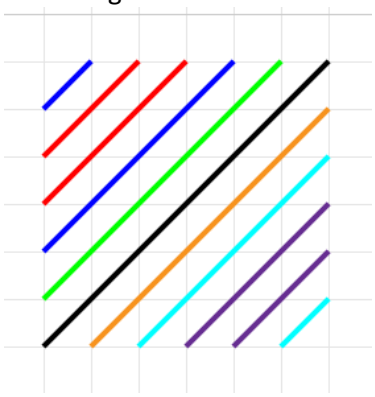
(b) matrix transpose from processor point-of-view

FIGURE TAKEN FROM: http://www.netlib.org/lapack/lawnspdf/lawn65.pdf

In this scenario, we can also see that each processor will have to send a message to every other processors.

The advantages of both of these methods are that they are simple to implement. Their principal inconvenient is the network contention they generate because of the storm of messages needing to travel.

If communication time becomes too much of an issue, another way of splitting the work would be by cross-diagonal.



The advantage of such a method is that besides synchronization, there is no need for communication between the processors. By looking at the figure, if we assume that every grid intersection is a matrix value, we can see that all processors (represented by different colors) share an equivalent work load (in this case, every processor will do 6 swaps, since the central elements on the diagonals don't need to move).

The biggest disadvantage of this scheme is that it might be a little harder to implement.

Does it matter whether you are programming a shared address space or message-passing machine?
Yes it does. If we are not in a shared address space, we can simply interchange the global row and column indices of the elements. If we are operating in a message passing machine, the data will need to be physically moved from one processor to another.

b- Why is interposes communication in a matrix transpose called all-to-all personalized communication?
It is called this way because every node of the system will need to send a message to every other node, hence the 'all-to-all'. This is a very dense communication pattern and will cause a lot of network congestion.

c- Write simple pseudo code for the parallel matrix transposition in a shared address space and in message passing (just the loops that implement the transpose).

Shared address space

```
For(i = 0 to n-1){
        If(i%PID){
                For(j = 0 to n-1)
                NewMatrix[j][i] = OldMatrix[i][j]
        }
}
```

Message passing:

```
For(i = 0 to n-1){
        If(i%PID){
                For(j = 0 to n-1){
                        Send {A[i][j], i, j} to {PID = j/P};
                        If(There is a message from others){
                                Extract {DATA, i, j};
                                newMatrix[j][i] = DATA;
                        }
                }
}
// Run a final check in case a processor is lagging
While(newMatrix !fully completed){
        If(There is a message from others){
                Extract {DATA, i, j};
                newMatrix[j][i] = DATA;
        }
}
```

3- Consider a bus-based shared memory multiprocessor system. It is constructed using processors with speed of 10^6 instructions/s, and a bus with a peak bandwidth of 10^5 fetches/s. The caches are designed to support a hit rate of 90%.
(a) What is the maximum number of processors that can be supported by this system?

With a 90% hit rate, the cashes have a 10% miss rate. If a processor runs 10^6 instructions/sec, and 10% of them will require a bus transfer because of cash miss, 1 processor will require 10% of 10^6 = 10^5 memory accesses via the bus/second.
With a peak BW of 10^5 fetches/sec, the bus can accommodate a single processor.

(b) What hit rate is needed to support a 20-processor system?
The bus can handle 10^5 fetches/second, that will need to be shared among every 20 processors, so every processor will have access at most to 10^5/20 = 5,000 fetches/s.
If every processor have access to 5,000 fetches/second, the cash miss rate cannot exceed 5,000 misses/s, so running 10^6 instructions/s, the cash will need at most a miss rate of:
5000/10^6 = 0.005.

So the hit rate required is 100% - 0.5% = 99.5%

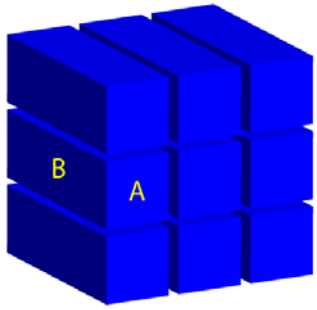
4- You are given a simple 3D Array of integer which consist of n^3 elements and p processors as a system specification. Using the Domain Decomposition technique, you can be able to varyingly parallelize operations of P processors on this 3D array. What are two different ways of decomposition and the concerning computation and computation overhead of each way.

Best domain decomposition will depend on information requirements. We will want to keep track of the communication to computation ratio (CCR) and keep it as low as possible, in this case expressed as the area to volume.

In either domain splitting scenario, the computational requirements are number of elements / number of processors on the task, i.e. n^3/p.

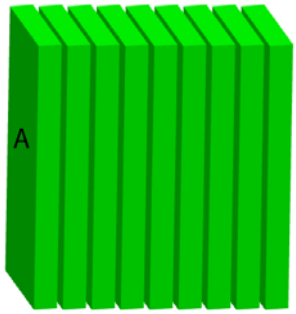
In a 3d Array, I can think of 3 way of splitting the domain.

The first is in ‘stick strips’.



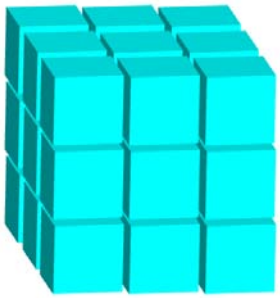
In this case, the area of every sub domain is composed of 2 sides (A) of n^2/p and 4 sides (B) of $n^2/\sqrt{p}$. It is important to observe that no matter how large n or p, the sides 'A' will never have to make any kind of communication between processor, the communicating areas is only composed of $4 * n^2/\sqrt{p}$
This scheme would yield a CCR of $[4 * n^2/\sqrt{p}] / [n^3/p] = 4\sqrt{p}/n$

The second possible domain split is in sheets:



In this case, every sub domain has 2 sides for communication area of size n^2, so the total communication area per processor is $2n^2$. This makes the CCR $[2n^2] / [n^3/p] = 2p/n$

The third possible domain split is in small cubes:



In this case, every sub domain would have 6 communicating sides, each one of size $n^2/p^{(2/3)}$. This would yield a CCR of $[6\ n^2/p^{(2/3)}]/ [n^3/p] = 6p^{1/3}/n$.

Name	Communication area	Computation	CCR ratio
Sticks	$4 * n^2/\sqrt{p}$	n^3/p	$4\sqrt{p}/n$
Planes	$2 * n^2$	n^3/p	$2p/n$
Cubes	$6\ n^2/p^{(2/3)}$	n^3/p	$6p^{1/3}/n$

Out of these 3 options, we can observe that the size of 'n' has equal impact on all domain splits. As 'p' grows, we can see that the domain split in cubes will only grow with a factor of $p^{1/3}$, so should be chosen when considering a large number of processors. The second best choice is the sticks, which yields a reduction by a factor of square root, and finally the most communication intensive choice is the planes, which yields at best a linear enhancement.

5-Consider a bus-based machine with 4 processors, each running at a 0.5 x 10^9 Instructions/S and running a workload that consists of: 60% ALU operations, 10% loads, 10% stores and 20% branches. Suppose that the cache miss rate at each processor is 1% for instruction cache and 2% for data cache, and that the cache sharing among 2 processors is 40% and zero otherwise. The system bus bandwidth is 8GB/s. Assuming that the cache line is 32 bytes large, and a snooping protocol, determine the bandwidth used. How many processors could the bus accommodate?

Total instructions / second
4P * 0.5GIPS = 2GIPS

Achievable cash updates / sec
8GB/s / (32B/cash line) = 250M cash lines/sec
So the bus can update 250M cash lines every second

Data cash misses:
Since we are provided a cash miss rate per processor and not per cash, the fact that 40% of the cash is shared between 2 processors is already accounted in those numbers and therefore need not be taken into account.

Instruction cash Misses (Assuming a 32 bit architecture):
Per Processor: 0.5 GIPS * 1% miss = 5M Bus Access/sec
For 4 Processors: 5M * 4 = 20M Bus Access/sec

Data cash misses:
LD:
Per P: 0.5GIPS * 10%LD * 2% miss rate = 1M Bus Access/sec
For 4P: 4M Bus Access/sec

STORE: Assuming that stores will always generate bus occupancy because of snooping
Per P: 0.5GIPS * 10%SD = 50M Bus Access/sec
For 4P: 200M Bus Access/sec

So the total number of cash misses per seconds amounts to
Per P: 5M + 1M + 50M = 56M/sec
For 4 P: 20M + 4M + 200M = 224M/sec

Since the bus can handle 250M transactions/sec, we are running at 89.6% capacity.

As calculated above, every processor will generate 56M bus requests/sec, so the maximum number of processors that the bus can accommodate is 4.