Your Paper

You

December 3, 2021

Contents

1 Exercise 1

Describe the basic steps in designing a parallel algorithm. What should we look out for in each of these stages?

1.1 Solution

There are four basic principles that we must take into consideration to design a parallel algorithm. These different principles are meticulously described below.

1.1.1 Problem Partitioning

A problem must be split into smaller processes that can run concurrently. However we must be careful since concurrent and parallel code is not always faster.

- **The number of processes must be equal to or larger than the available CPUs.** Each processing core can execute more than one threads at a time since multiple ALUs are available. When I/O operations are performed a context switch might occur and the next ready process starts running. When more processes than the total available cores are ready we can save time when performing I/O related tasks. Spawning too many threads however produces an overhead in performance. Therefore, the number of threads is recommended to be around the number of total available CPU's logical cores.
- **Unnecessary computations and memory operations like stores and loads must be avoided.** As much processing power is available must be harvested without wasting any resources. Unnecessary calculations, most times, make the overall execution slower. In addition, unnecessary memory operations may might result in a lot of cache misses
- **When more CPU cores are available the number of processes must increase accordingly.** The execution must depend on the available cores in order to take full advantage of the available hardware. In other words, the application must scale to handle more computational load.
- **The computational size of each process must not increase with the same rate that the problem does.** Few large processes might take a lot more CPU time than more smaller ones. This can be attributed to current CPU scheduling policies and the way the load is distributed to the processing units.

1.1.2 Process Communication

Processes must communicate with each other in order to synchronize and share data. The communication process may take some time since in many cases it is performed as an I/O operation. In order to improve performance we must apply the following process communication principles.

- **Same frequency and load**
- **Asynchronous with calculations**
- **Synchronous between other communications**
- **Local (when possible)**

Communication can be also categorized as

- **Local or general**
- **Structured or not-structured**
- **Static or dynamic**
- **Synchronous or asynchronous**

1.1.3 Process Merging

In order to avoid process communication costs many small processes can be merged into a single bigger process. To do this we must be very careful since a lot more problems may arise.

- **The degree of parallelism might decrease significantly.** Many less operations can be simultaneously executed by each core since each process must now run sequentially.
- **Small processes can not run in parallel since they must be merged together.** Communication cost must be greater than the cost of running sequential code instead of parallel to observe any significant performance boost.
- **Communication between processes that perform same calculations must be avoided.** Process communication takes and lot of time since a context switch might occur. Therefore, processes that perform the same calculation should not try to communicate with each other.

1.2 Process-CPU Pairing

When pairing processes to CPU cores we must have in mind a few more things.

- **Communication between cores must be minimized.** Each time a core wants to share data with another core a lot of time is lost. Cache Misses will most probably occur since the L1 and L2 caches of each core do not contain the same data that the other cores have. In addition, cache coherence must be preserved and cache lines travel through the memory bus more frequently. This creates congestion in the memory bus. Finally, the MMU of each core must recalculate the physical page frame from the virtual page in order to store it into the TLB and actually access the physical memory.
- **Processes that communicate must occupy the same core.** In order to avoit above we can group processes that communicate with each other and put them on the same CPU core. This partially solves the above problem.
- **Increasing the degree of parallelism is a priority.** By breaking code in more processes that can run concurrently the OS's scheduler can make a few jobs run in parallel. Thus, a lot of waiting time can be saved.
- **Reduction in run-time will probably be observed.** More processing power combined with efficient code can provide an immense performance boost.

2 Exercise 2

Which are the basic properties of mapping two graphs? Describe what each of these reveals with respect the workload and the communication needs, considering that the first graph represents the individual processes of a parallel algorithm and the second one represents a system of computing nodes with their wiring while the embedding shows the assignment of processes to computing nodes.

2.1 Solution

To represent the individual processes of a parallel algorithm and the wiring between nodes we use two graphs. $G(V, E)$ is used to represent the architecture of the algorithm and it's processes. $G'(V', E')$ is used to represent the computational nodes and the wiring between them. Vertices and edges, V and E, can be matched with one or more V' and E' respectively.

2.1.1 Graph Conjunction Properties

- **Congestion:** When too many edges E are matched with a single edge E' congestion will occur and messages will be transmitted at a slower rate.
- **Expansion:** The computational load of each vertex depends on V/V'
  - When V' is more loaded $V/V' < 1$
  - When V is more loaded $V/V' > 1$
- **Dilation:** The ammount of edges E' that are required in order edge E to be matched. Large dilation on a certain edge means that nodes that use that edge communicate a lot slower. Dilation is like congestion since communication speed is affected.

3 Exercise 3

Assume the symmetric matrix A from Figure 1 below and the problem of matrix-vector multiplication Ab = c, where b and c are vectors of 8 elements. Assume that we divide the problem by block of lines of matrix A and elements of b and c, into 2 processors in the obvious way of Fig. 1. Create the dependence graph of the rows/columns of matrix A. How will the specific multiplication operation be performed in the 2 processors and what are the communication requirements? Study the dependence graph of the rows/columns in matrix A that you made before and present another (non- obvious) partition of the problem with less communication. Which rows/columns will be assigned to each processor? What are the communication requirements?

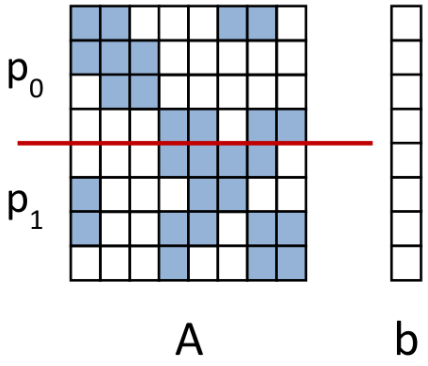
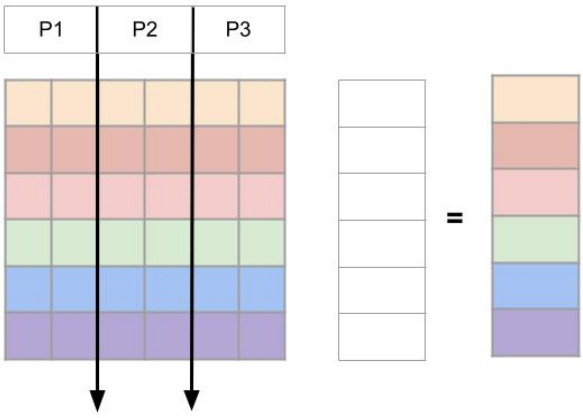


Figure 1. Sparse symmetric matrix A.

3.1 Solution

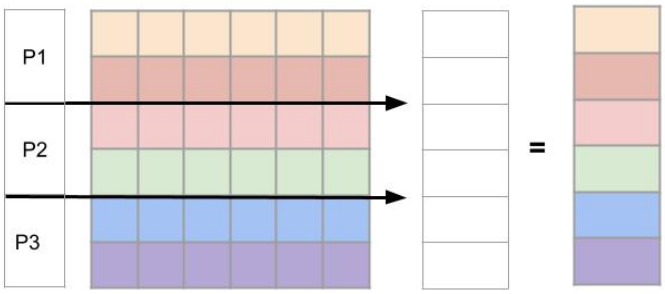
3.1.1 Assigning Columns to Processors

Each thread is responsible to calculate the inner product a block of columns with the vector B. Threads must communicate to accumulate their results into a single cell in vector C ($C[i] = P_1 + P_2 + P_3$). The communication process is shown bellow.



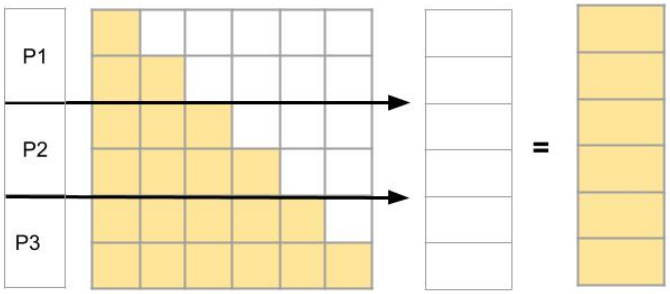
3.1.2 Assigning Rows to Processors

Each threads is responsible to calculate the inner product of a block of rows with the vector B. Communication is not required between threads since each thread changes different cells in vector C.



3.1.3 Taking Advantage of the Symmetric Nature of Array

A lot less memory will be used since almost half of the array is stored. The upper half is not important since it is the same as the lower half and must not be stored. Threads have access to the elements below then the main diagonal.



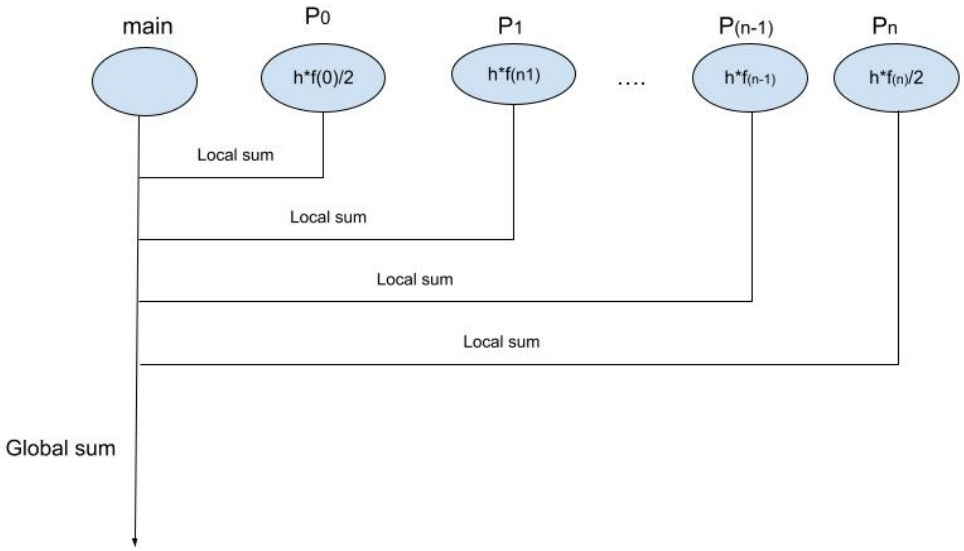
To calculate correct results we must find the corresponding positions i and j on the symmetric matrix.

4 Exercise 4

Graph the solution to the following problem: If (x_i, y_i), i = 1,2,..., n points from the graph of a function, calculate its integral using the trapezoidal rule. Explain in details what task each node of the graph is working on and use the directional arrows to indicate process dependency and data exchange. Calculate maximum and average degree of parallelism for general n, while creating your graphs for n = 4. Try writing a parallel algorithm for this problem running on p processors. What is the relation between p and n so there is uniform work load? Assume this relation for your study.

4.1 Solution

4.1.1 Graph



4.1.2 Maximum and Average Degree of Parallelism

The maximum degree of parallelism is N, where N is the maximum number of trapezoids. The number of trapezoids N is recommended to be equal to the number of processes P that the CPU can execute concurrently.
The average degree of parallelism is

$$\frac{N + \frac{N}{2} + \frac{N}{4} + ... + 1}{h},\ h = 1 + log_2(n) + 1$$

4.1.3 Relation Between P and N

When N increases P should also increase.The upper limit of P must be around the available CPU threads. If Ps are more than the available CPU threads and no I/O tasks are performed only some processes will be executed and the others will wait for a CPU to be available.

5 Exercise 5

Assume 2 graphs G(V, E) and G'(V', E'). G is a 2-dimensional grid of 16 vertices as in Fig. 2, and G' is a 4th dimension hypercube. Calculate the mapping between the nodes of the 2 graphs so that all three properties of mapping are equal to 1. For the 2D grid, express the nodes' id of the Cartesian coordinate type, while for the hypercube, consider to express the ids of the nodes in binary format, as shown in slides. a) (10) Write the formula that maps the ids. b) (15) Write a general mapping that works for general cases, where the hypercube is of k dimensions and the 2D grid has 2 k nodes while k is an even number.

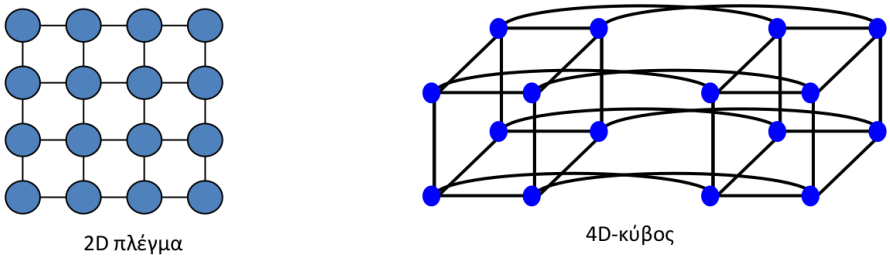


Figure 2. 2D grid, G(V,E), hypercube 4th dimension, G' (V', E).

5.1 Solution

We have to do a match-up between the IDs of two graph's vertices and therefore *congestion* = *expansion* = *dilation* and equal to 1. In order to achieve this we must apply Gray Coding and match the 2D and 4D graphs up.

5.1.1 Mapping

Mapping a $grid(2^k \times 2^k)$ in a $hypercube 2(k+k)$ requeres mapping of $grid(i,j)$ vertex on $hypercube(i, k-1)||(j, k-1)$. Each row uses the 2 most significant bits while each column the 2 least significant bits.

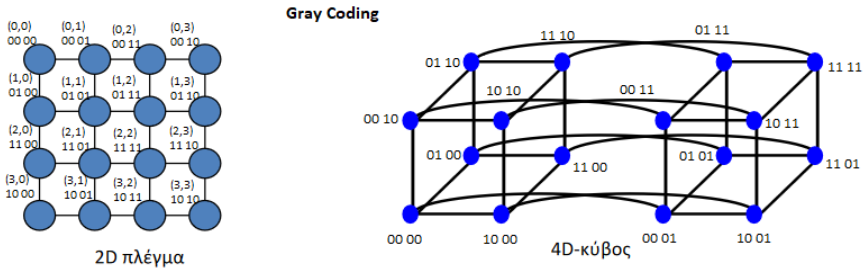
5.1.2 Mathematical Formula

The mathematical formula that matches the IDs is shown below.

$$G(0,1) = 0 \tag{1}$$

$$G(1,1) = 1 \tag{2}$$

$$G(i, x+1) = \begin{cases} G(i,x), i < 2^x \\ 2^x + G(2^{x+1} - 1 - i, x), i \geq 2^x \end{cases} \tag{3}$$



6 Exercise 6

Consider the problem of computing the norm 1 of an n by n matrix A, i.e.,

$$\|A\|_1 = \max_{1 \le j \le n} \sum_{i=1}^{n} |a_{ij}|$$

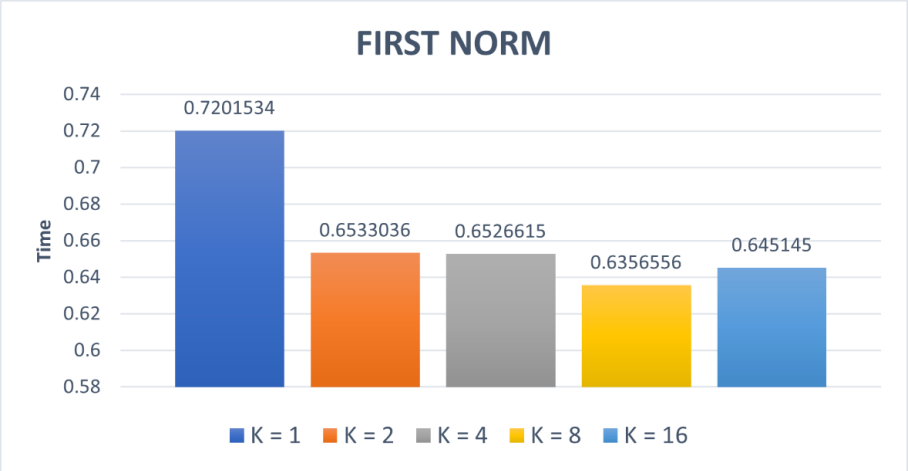
The 1-norm can be calculated with 2 loops. To study the effect of the loop step, we will perform the loop of the sums with different steps (k = 1, 2, 4, 8, 16) with k separate commands.
a. (20) Write corresponding functions that calculate the sums of the elements of each column with a different step k and measure the execution time.
b. (10) Graph the execution time of norm calculation with respect to k.
c. (10) What is the number of operations (addition or multiplication) with floating point numbers in your program?
d. (5) What is the maximum speed of your program in FLOPS (Floating Point Operations Per Second)?
e. (5) Which k or ks are considered critical for efficiency improvement?
f. (10) What is the reason of expecting an improvement in efficiency?
g. Make sure your operations and results are correct. Answer each sub-question separately and clearly justify your remarks.

6.1 Solution

6.1.1 Code and Execution Time (a)

Code is included in the submission files.

6.1.2 Graph of Execution Time (b)



6.1.3 Number of Operations (c)

The number of FLOP is N(N-1), where N is the dimensions of the array. Elements of the array are accessed by column. For this particular program we use $N = 10^4$, so the total FLOP are equal to 99990000 operations.

6.1.4 Floating Point Operations per Second (d)

The speed of our program in FLOPS (Floating Point Operations Per Second) is the division of operations to execution time of the program. But our program runs for $N = 10^4$ and k = 1,2,4,8,16.

$$S = \frac{Operations}{Time} = \frac{N(N-1)}{t} \tag{1}$$

$$k = 1 \xrightarrow{(1)} \begin{cases} time = 0.7201534 \\ S = 99990000/0.7201534 = 138845418.212\ FLOPS \end{cases} \tag{2}$$

$$k = 2 \xrightarrow{(1)} \begin{cases} time = 0.6533036 \\ S = 99990000/0.6533036 = 153052883.835\ FLOPS \end{cases} \tag{3}$$

$$k = 4 \xrightarrow{(1)} \begin{cases} time = 0.6526615 \\ S = 99990000/0.6526615 = 153203459.987\ FLOPS \end{cases} \tag{4}$$

$$k = 8 \xrightarrow{(1)} \begin{cases} time = 0.6356556 \\ S = 99990000/0.6356556 = 157302161.737\ FLOPS \end{cases} \tag{5}$$

$$k = 16 \xrightarrow{(1)} \begin{cases} time = 0.645145 \\ S = 99990000/0.645145 = 154988413.457\ FLOPS \end{cases} \tag{6}$$

So, the maximum speed of our program is for k = 8 with speed 154988413.457 FLOPS.

6.1.5 Critical K for Efficiency Improvement (e)

Increasing k to 2 is considered critical for improvement since if we look at the graph we can observe that the execution time decreases with significantly greater rate than with any other instance when k's value is altered.

6.1.6 Reason of Expecting Improvement (f)

The reason of expecting an improvement in efficiency is because we use the technique of loop unrolling. When Loop Unrolling is applied the body of the loop contains many more instructions, but the iterations are a lot less. This might cause a lot more Cache Misses in the L1 Instruction Cache. On the other hand, in each iteration there is at least one conditional statement that needs to be checked in order for the loop to be executed. By reducing the iterations, the conditional statements decrease as well. A conditional statement is somewhat "expensive" since each one takes at least 2 CPU Cycles to complete (Assuming that Branch Prediction is Disabled or False).

7 Exercise 7

Consider the problem of computing the max norm of an n by n matrix A, i.e.,

$$\|A\|_\infty = \max_{1\le j\le n} \sum_{j=1}^{n} |a_{ij}|$$

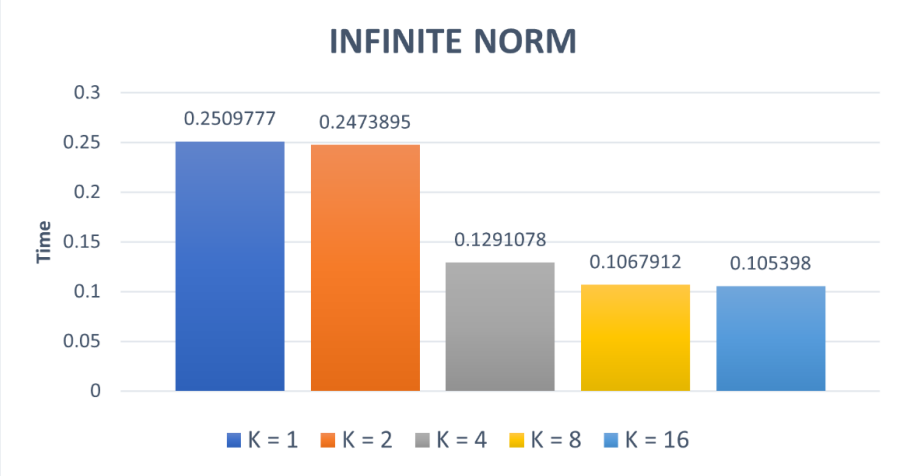
The max-norm can be calculated with 2 loops. To study the effect of the loop step, we will perform the loop of the sums with different steps (k = 1, 2, 4, 8, 16) with k separate commands.
a. (20) Write corresponding functions that calculate the sums of the elements of each line with a different step k and measure the execution time.
b. (10) Graph the execution time of norm calculation with respect to k.
c. (10) What is the number of operations (addition or multiplication) with floating point numbers in your program?
d. (5) What is the maximum speed of your program in FLOPS (Floating Point Operations Per Second)?
e. (5) Which k or ks are considered critical for efficiency improvement?
f. (10) What is the reason of expecting an improvement in efficiency?
g. Make sure your operations and results are correct. Answer each sub-question separately and clearly justify your remarks.

7.1 Solution

7.1.1 Code and Execution Time (a)

Code is included in the submission files.

7.1.2 Graph of Execution Time (b)



7.1.3 Number of Operations (c)

The number of FLOP is N(N-1), where N is the dimensions of the array. Elements of the array are accessed by column. For this particular program we use $N = 10^4$, so the total FLOP are equal to 99990000 operations.

7.1.4 Floating Point Operations per Second (d)

The speed of our program in FLOPS (Floating Point Operations Per Second) is the division of operations to execution time of the program. But our program runs for $N = 10^4$ and k = 1,2,4,8,16.

$$S = \frac{Operations}{Time} = \frac{N(N-1)}{t} \tag{1}$$

$$k = 1 \xrightarrow{(1)} \begin{cases} time = 0.2509777 \\ S = 99990000/0.2509777 = 398301929.733\ FLOPS \end{cases} \tag{2}$$

$$k = 2 \xrightarrow{(1)} \begin{cases} time = 0.2473893 \\ S = 99990000/0.2473893 = 404180779.039\ FLOPS \end{cases} \tag{3}$$

$$k = 4 \xrightarrow{(1)} \begin{cases} time = 0.1291078 \\ S = 99990000/0.1291078 = 774469087.073\ FLOPS \end{cases} \tag{4}$$

$$k = 8 \xrightarrow{(1)} \begin{cases} time = 0.1067912 \\ S = 99990000/0.1067912 = 93613104.451\ FLOPS \end{cases} \tag{5}$$

$$k = 16 \xrightarrow{(1)} \begin{cases} time = 0.105398 \\ S = 99990000/0.105398 = 948689728.458\ FLOPS \end{cases} \tag{6}$$

So, the maximum speed of our program is for k = 16 with speed 948689728.458 FLOPS

7.1.5 Critical K for Efficiency Improvement (e)

Increasing k to 4 is considered critical for improvement since if we look at the graph we can observe that the execution time decreases with significantly greater rate than with any other instance when k's value is altered.

7.1.6 Reason of Expecting Improvement (f)

The reason of expecting an improvement in efficiency is because we use the technique of loop unrolling. When Loop Unrolling is applied the body of the loop contains many more instructions, but the iterations are a lot less. This might cause a lot more Cache Misses in the L1 Instruction Cache. On the other hand, in each iteration there is at least one conditional statement that needs to be checked in order for the loop to be executed. By reducing the iterations, the conditional statements decrease as well. A conditional statement is somewhat "expensive" since each one takes at least 2 CPU Cycles to complete (Assuming that Branch Prediction is Disabled or False).

8 Exercise 8

Is there any difference in execution time between the 2 programs of exercises 6 and 7? Justify your answer.

8.1 Solution

The difference between exercise 6 and 7 is quite obvious because of: Arrays and Multi-Dimensional Arrays in C are stored horizontally. When a Memory Address is accessed (Read or Write) the CPU searches L1, L2 and L3 Caches and then the Main Memory (RAM or in the worst-case scenario the Disk). If the requested data is not on the Cache, then many more CPU Cycles are needed to access the data in the Main Memory. When the data is retrieved from Main Memory it is stored into the Cache in order to be retrieved more quickly next time. This is called a Cache Miss, since the data was not in the Cache and a Main Memory access was necessary. On a Cache Miss the CPU also retrieves a block of data that was next to the requested data since this block might be used later (Cache Locality). On the other hand, a Cache Hit can occur, and the data can be easily retrieved from Cache. A Cache Hit is always faster than a Cache Miss. So, should you iterate multi-dimensional arrays vertically or horizontally? If you iterate arrays vertically many Cache Misses will occur since there will be a lot less Cache Locality (a Cache Miss will pull a block of the following Columns into the Cache, but the program tries to access a whole new Row which might not be into the Cache). If you iterate arrays horizontally fewer Cache Misses will occur since data is already in 2 Cache.

Cache Misses and Hits with Red and Green color respectively.

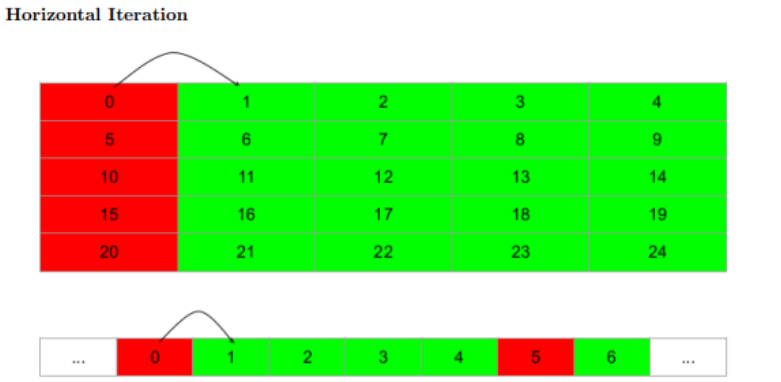


Figure 2: 2D Array in Memory with Horizontal Iteration (Jump to Nearby Address $\Longrightarrow$ *CacheHits*)

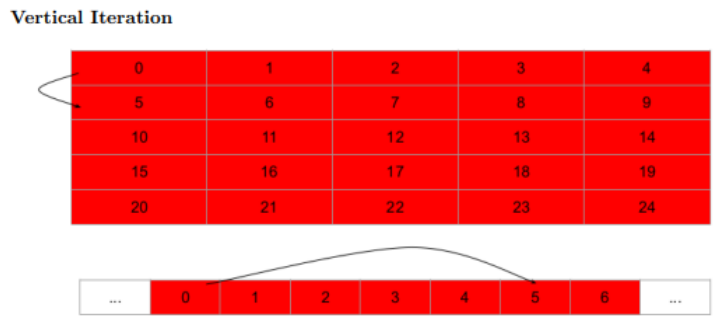


Figure 1: 2D Array in Memory with Vertical Iteration (Jumb in Memory $\Longrightarrow$ *CacheMisses*)