HPC and modeling

Chapter 2 – Shared memory and Patterns

M2 – MSIAM

12 octobre 2016

Agenda

Parallel platforms : part 2

Some Examples

Models of parallelism

Specific patterns

Performances

Flynn classification

- SISD – Single Instruction, Single Data
- SIMD – Single Instruction, Multiple Data
- MISD – Multiple Instruction, Single Data
- MIMD – Multiple Instruction, Multiple Data

Most modern architectures are based on MIMD principles.

Main architectures (1)

- Multiple processing units : all the processing unit shares the same global memory.
  - Scaling is complex from the algorithm point of view but also from the technical point of view.
  - Intuitive programming since most of modern programming tools manage memory accesses automatically.
  - Local programming
- Cluster : aggregation of processing unit link through a high speed network. Memories are locals and each unit has its own memory. There is no global memory access.
  - Scaling only depends on the number of resources that are allocated to the cluster.
  - Specific communication protocol must be used for the processors to interact.
  - Optimization of the ratio computation/communication requires careful design.

Main architectures (2)

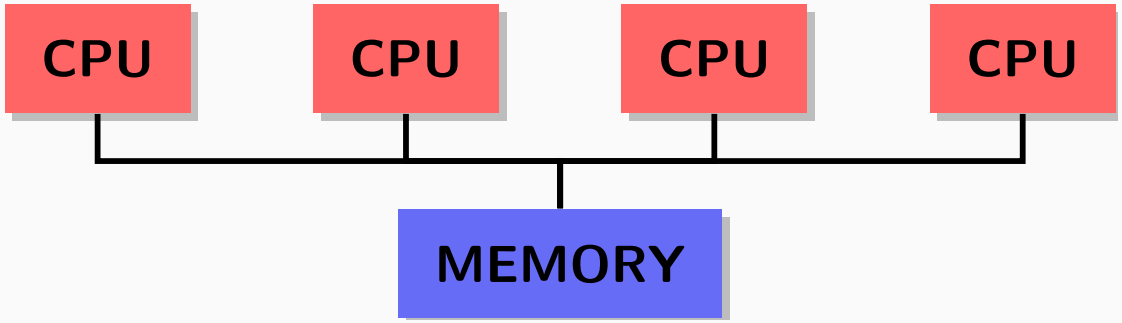
- Hybrids : heterogeneous collection of processing unit with a level of distributed memory. Each node can itself be a shared memory or a distributed memory architecture.
  - Programming is hard : at least two parallel paradigms must be use
  - Optimisation is complex.
  - Performance gain is better.
  - Some resources must be virtualized.

Emerging architectures

- Grid : heterogeneous processing unit link through low speed network (LAN/WAN).
  - Low network
  - No administration required.
  - Only for applications having low network requirements (SETI@HOME)
- Cloud : virtualization of hardware resources.
  - Low performances compared to standard architectures.
  - Cost effective for low usage.

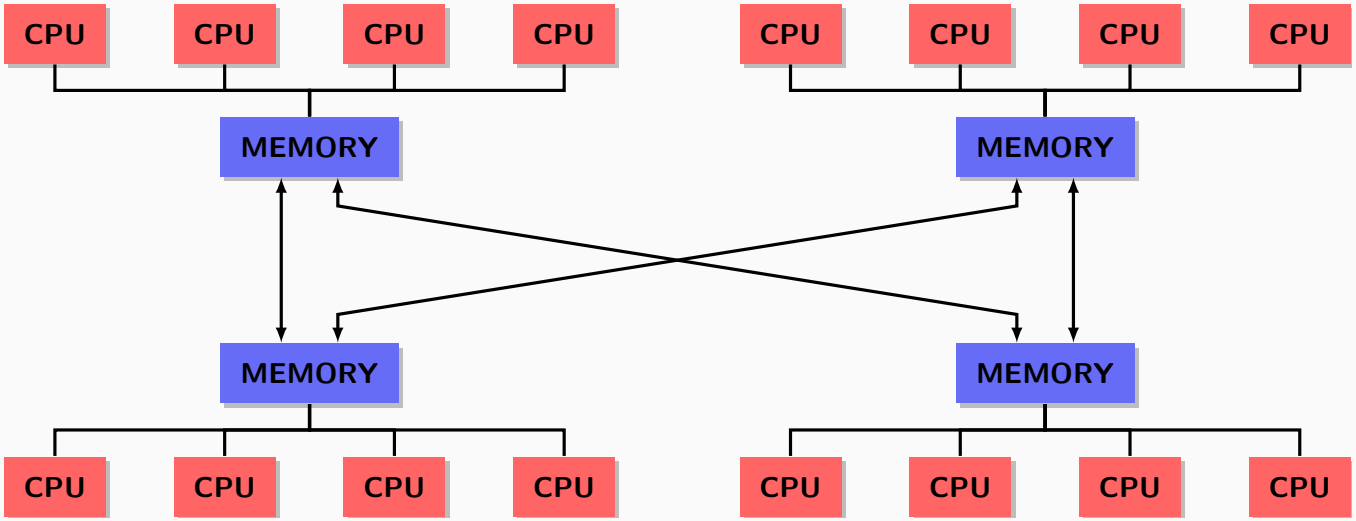
Shared memory (1)

- All the processors shares the same memory space. They communicate using reading and writing shared variables.
- Each processing unit carry out its task independently but modification of shared variables are instantaneous.
- Two kind of shared memories
  - SMP (Symmetric MultiProcessor) – All the processors share a link to the memory. Access to the memory is uniform.



Shared memory (2)

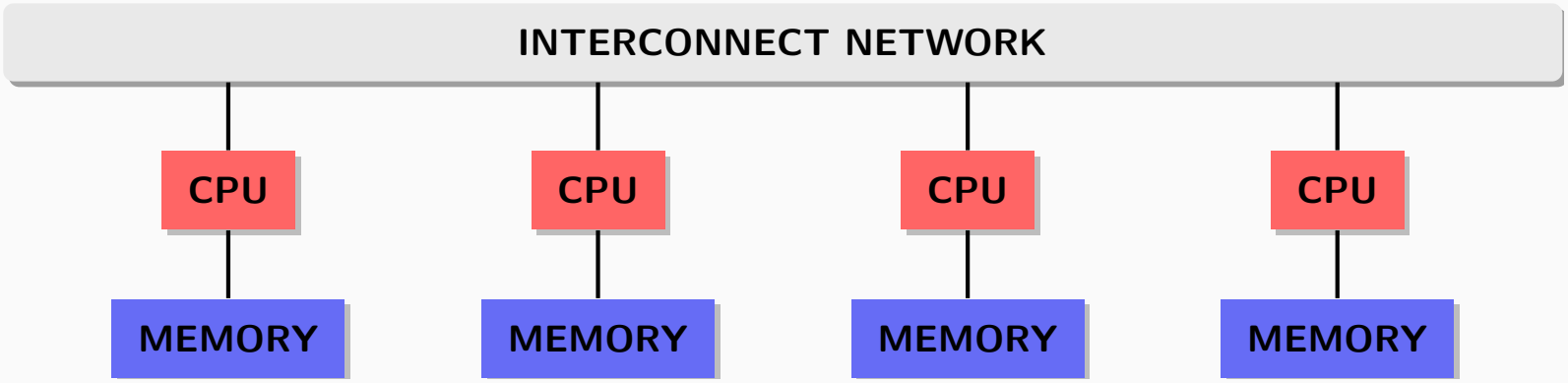
- NUMA (NonUniform Memory Access) – All the processors can access to the memory but not uniformly. Each processor has a preferred access to some memory part.



  - Decrease the risk of bottleneck to memory access.
  - Local memory cache on each processor to mitigate the effect of non-uniform access.

Distributed memory

- Each processor has its own memory. There is no global memory space.
- Each processor communicate with the others using messages.
  - Modification of variables are local and only the processor managing the memory can access it.
  - Each processor work independently on its own set of variables.
  - The speed of the resolution depends on the architecture : network, topology, processors.
  - Can scale easily.



Agenda

Parallel platforms : part 2

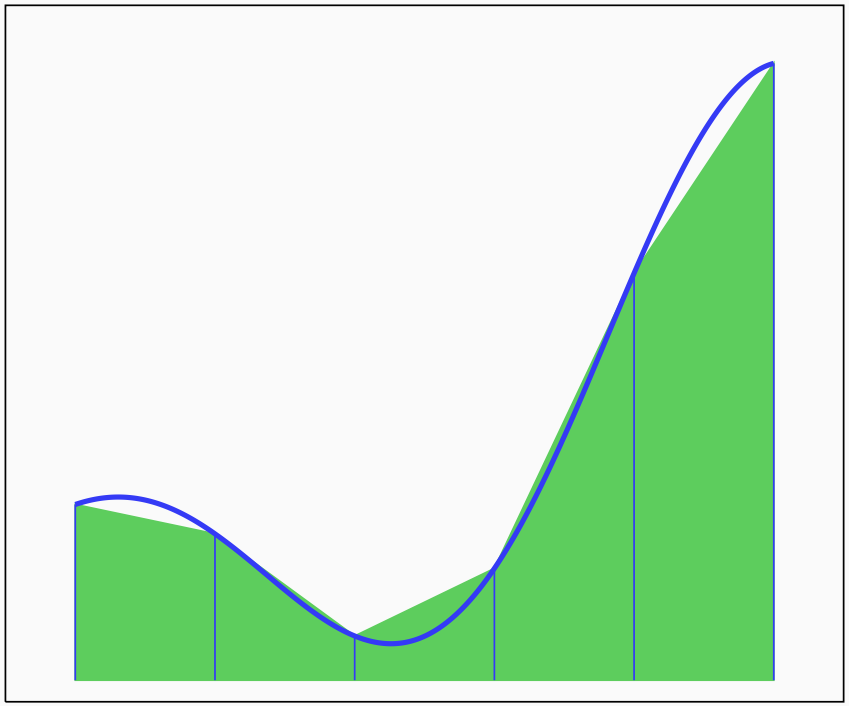
Some Examples

Models of parallelism

Specific patterns

Performances

Example 1 : Integral Computation



Trapezium formula is given by $\forall f \in C^2([a;b]), \exists \xi \in [a;b]$

$$I = \frac{h}{2}\left(f(a) + 2\sum_{k=1}^{n-1} f(a+kh) + f(b)\right) - (b-a)\frac{h^2}{12} f^{(2)}(\xi)$$

Example 1 : Pseudo-code

```
1     Input b, a, n
2     h = (b-a)/n
3     I = (f(a) + f(b))/2.0;
4     for (i = 0; i<= n-1; i++)
5     {
6       x_i = x_i + h
7       I += f(x_i)
8     }
9     I = h*I
```

Example 1 : Parallel Pseudo-code

```
1     Find b, a, n
2     h = (b-a)/n
3     local_n = n/n_p
4     local_a = a + id * local_n*h
5     local_b = local_a + local_n*h
6     local_I = Trap(local_a, local_b, local_n)
7
8     If (id == 0)
9     {
10      I = local_I;
11      for (i = 1; i<= n_p; i++)
12      {
13        I += local_I;
14      }
15      Display I;
16    }
```

Example 1 : Parallel Code

```
1   /* Compute the size of intervals */
2   d = 1.0/(double) n;
3
4   /* Start the threads */
5   #pragma omp parallel default(shared) private(nthreads, tid, x)
6   {
7     /* Get the thread number */
8     tid = omp_get_thread_num();
9
10    /* The master thread checks how many there are */
11  #pragma omp master
12    {
13      nthreads = omp_get_num_threads();
14      printf("Number of threads = %d\n", nthreads);
15    }
16
17    /* This loop is executed in parallel by the threads */
18  #pragma omp for reduction(+:sum)
19    for (i=0; i<n; i++) {
20      x = d*(double)i;
21      sum += f(x) + f(x+d);
22    }
23  }  /* The parallel section ends here */
24
25  pi = d*sum*0.5;
```

Example 2 : Matrix Multiply

Let $A \in \mathbb{R}^{n \times m}$ and $B \in \mathbb{R}^{m \times p}$. Then $C = A \times B, C \in \mathbb{R}^{m \times p}$ is defined by

$$\forall 1 \leq i \leq n, 1 \leq j \leq p, c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}$$

Example 2 : Pseudo-code

```
1     Input A, B, n, m, p
2     for(i = 1; i<= n;i++)
3     {
4       for(j = 1; j<= p;j++)
5       {
6         for(k = 1; k<= m;k++)
7         {
8           C[i][j]= C[i][j]+ A[i][k] x B[k][j]
9         }
10      }
11    }
```

Example 2 : Parallel Pseudo-code

```
1     Input A, B, n, m, p
2     l_n = n/nn_p
3     for(i = id*l_n +1; i<= (id+1)*l_n;i++)
4     {
5       for(j = 1; j<= p;j++)
6       {
7         for(k = 1; k<= m;k++)
8         {
9           C[i][j]= C[i][j]+ A[i][k] x B[k][j]
10        }
11      }
12    }
```

Example 2 : Parallel Code

```
#pragma omp parallel shared(a,b,c,nthreads,chunk) private(tid,i,j,k)
{
  tid = omp_get_thread_num();
  /* Initialize matrices */
#pragma omp for schedule (static, chunk)
  for (i=0; i<NRA; i++)
    for (j=0; j<NCA; j++)
      a[i][j]= i+j;
#pragma omp for schedule (static, chunk)
  for (i=0; i<NCA; i++)
    for (j=0; j<NCB; j++)
      b[i][j]= i*j;
#pragma omp for schedule (static, chunk)
  for (i=0; i<NRA; i++)
    for (j=0; j<NCB; j++)
      c[i][j]= 0;

  /* Do matrix multiply sharing iterations on outer loop */
#pragma omp for schedule (static, chunk)
  for (i=0; i<NRA; i++)
    for(j=0; j<NCB; j++)
      for (k=0; k<NCA; k++)
        c[i][j] += a[i][k] * b[k][j];
}
```

Example 3 : Gaussian Elimination – Pseudo-code

```
1     for k = 1 ... m:
2       Find pivot for column k:
3       i_max  := argmax (i = k ... m, abs(A[i, k]))
4       if A[i_max, k] = 0
5         error "Matrix is singular!"
6       swap rows(k, i_max)
7       Do for all rows below pivot:
8       for i = k + 1 ... m:
9         Do for all remaining elements in current row:
10        for j = k ... n:
11          A[i, j]  := A[i, j] - A[k, j] * (A[i, k] / A[k, k])
12        Fill lower triangular matrix with zeros:
13        A[i, k]  := 0
```

Example 3 : Parallel Code

```
1   for(pivot = 1; pivot < n; pivot++)
2   {
3   #pragma omp parallel for private(xmult) schedule(runtime)
4     {
5       for(i = pivot + 1; i < n; i++)
6       {
7         xmult = a[i][pivot] / a[pivot][pivot]
8           for(j = pivot + 1; j < n; j++)
9           {
10            a[i][j] -=  (xmult * a[pivot][j])
11          }
12        b[i] -=  (xmult * b[pivot])
13      }
14    }
15  }
```

Agenda

Parallel platforms : part 2

Some Examples

Models of parallelism

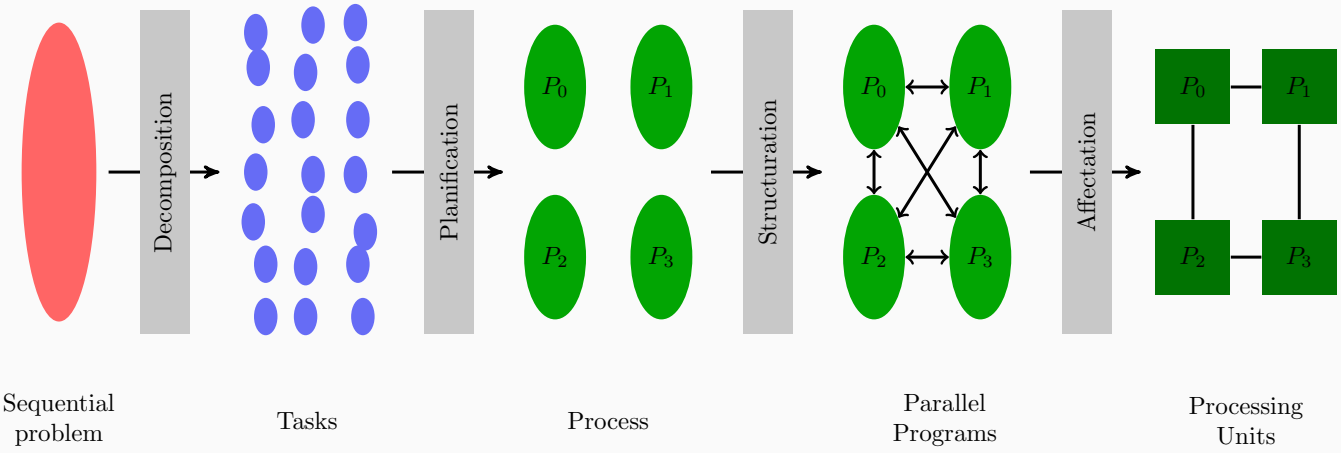
Specific patterns

Performances

Choice criterion

- Three parameters may influence the choice of a kind of parallelism
  - Flexibility : support of different programming constraints. Should adapt to different architectures.
  - Efficiency : better scalability.
  - Simplicity : allows to solve complex problem but with a low maintenance cost.
- For each model of parallelism, we will expose the strengths and weaknesses for each elements.

Parallel decomposition



Dependency Design Space

- The analysis is made on three elements
  - Grouping the data

    time dependency
    collection of data
    independence
  - Scheduling

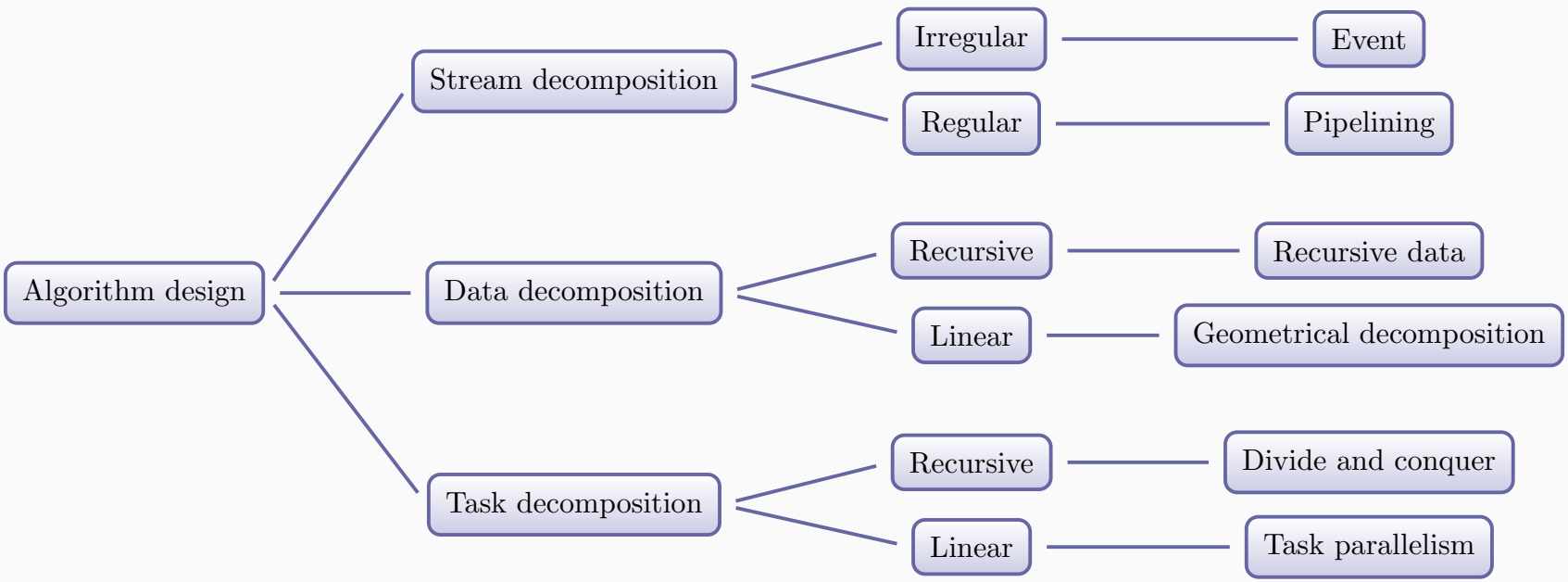
    Identify which data are requires for executing a specific task.
    Identify the tasks creating the different data.
  - Sharing the data

    Identify the data shared between tasks.
    Manage access to data.

Decomposition

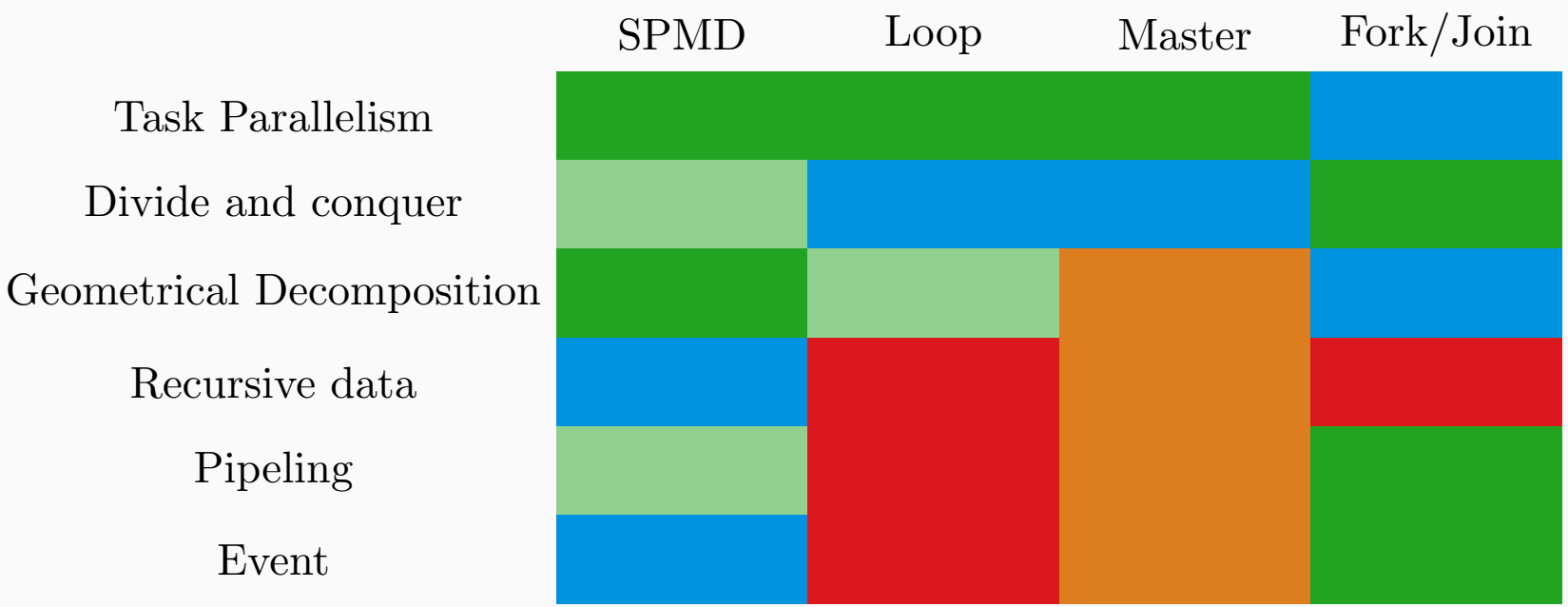
- Identify the elements that allows for a parallel processing.
- The question to answer is
  How to decompose the code in order to achieve maximum parallelism ?
- Task decomposition : the problem is a sequence of instructions that can be divided into elementary tasks. Each task can be executed independently.
- Data decomposition : data of the problem are split into elementary blocks of independent data.
- Pipeline decomposition : the problem is a sequence of task apply to blocks of independent data.

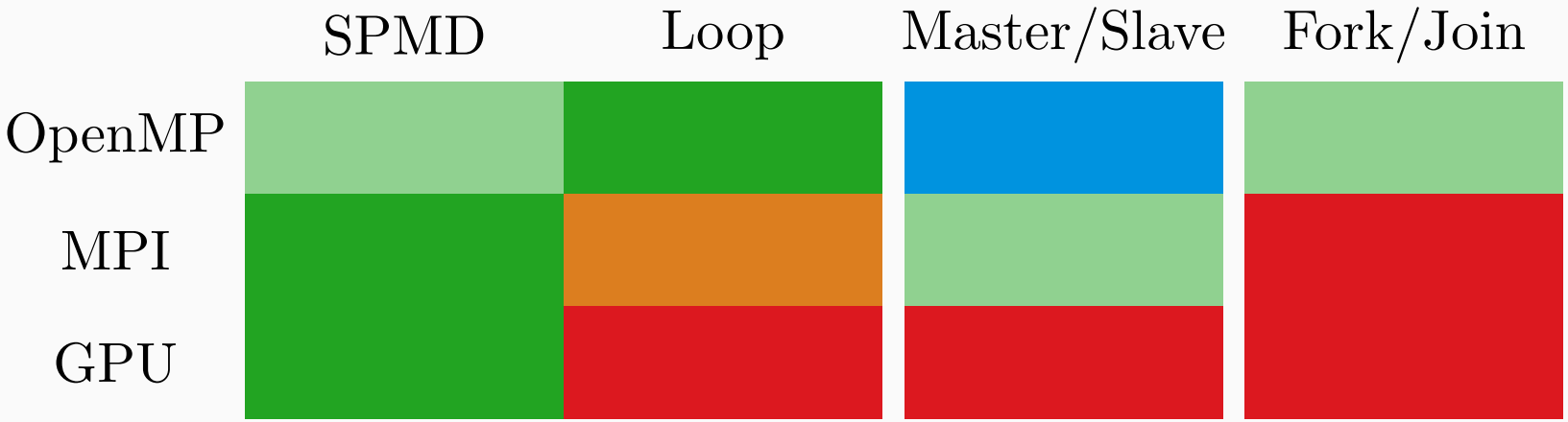
Parallel Algorithms



Parallel Patterns

- SPMD
- Master/Slave
- Master/Worker
- Queue
- Join/Fork
- Loop


SPMD
Loop
Master
Fork/Join
Task Parallelism
Divide and conquer
Geometrical Decomposition
Recursive data
Pipeling
Event


SPMD
Loop
Master/Slave
Fork/Join
OpenMP
MPI
GPU

Agenda

Parallel platforms : part 2

Some Examples

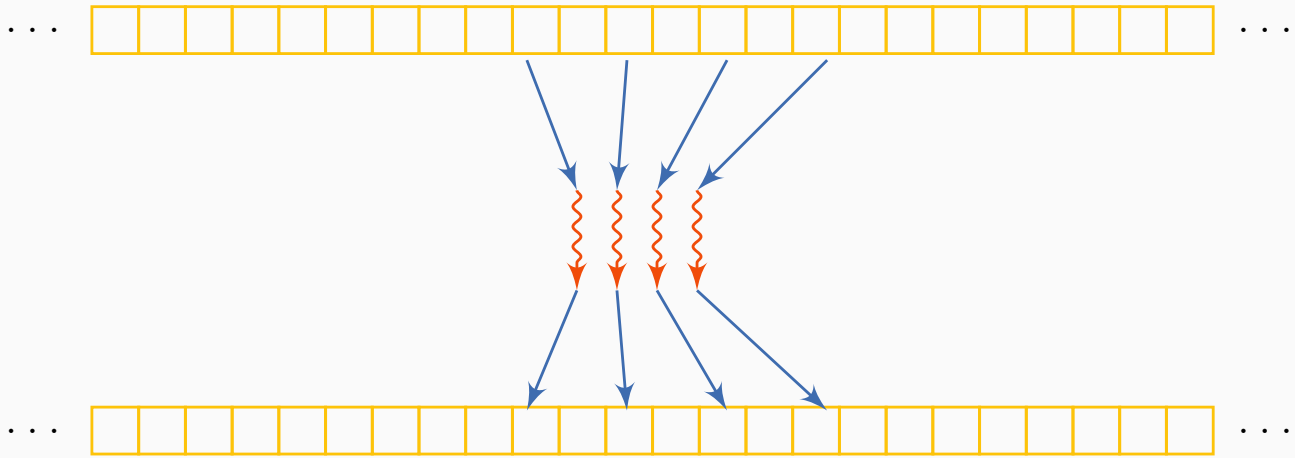
Models of parallelism

Specific patterns

Performances

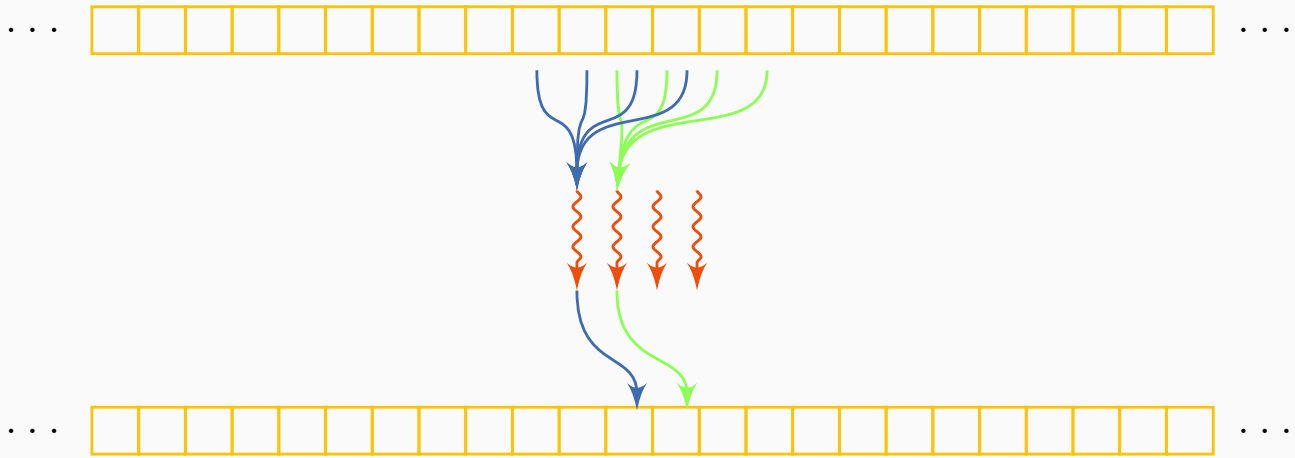
Map pattern

All the threads read and write data from specific and distincts places.

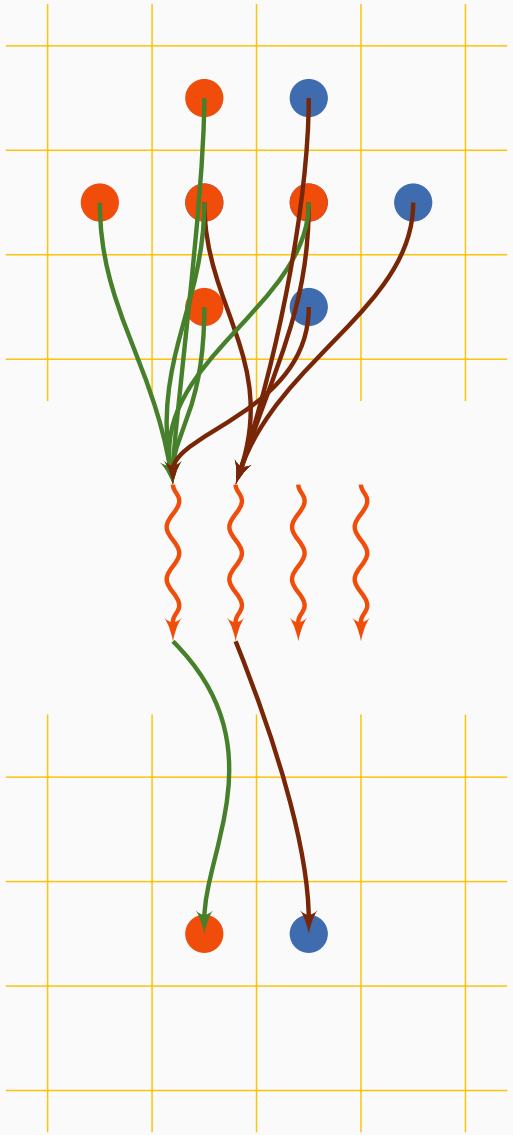


Gather pattern

All the threads read data from specific and distincts places. Some operation is realized on the data. One thread write the result in a unique place.

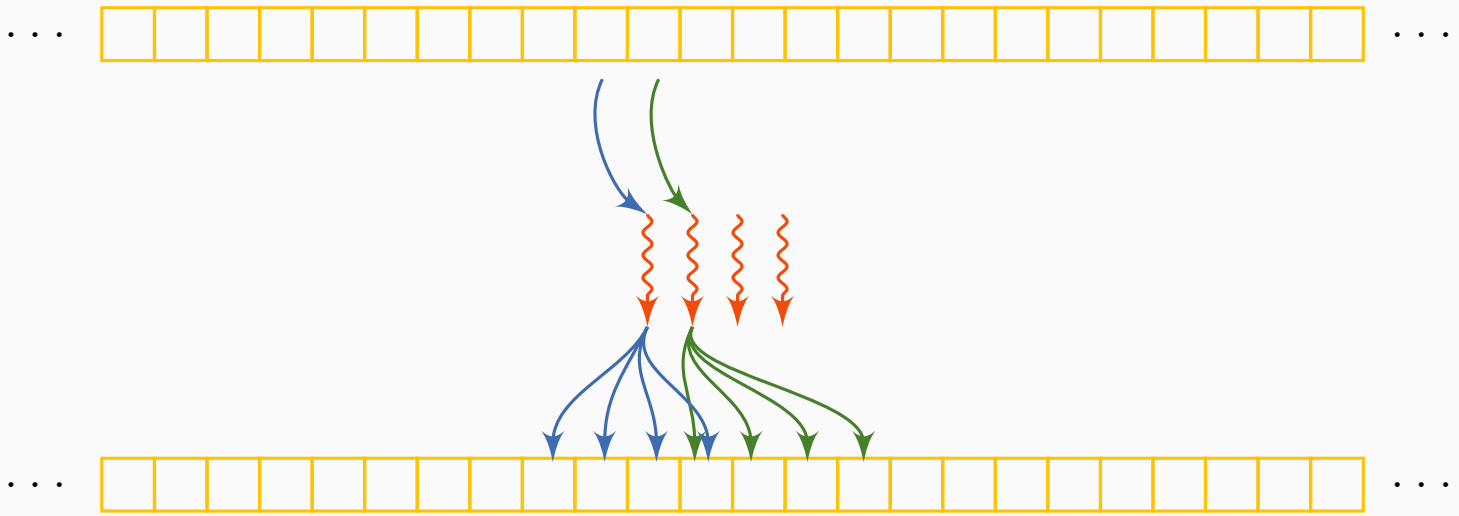


Gather pattern

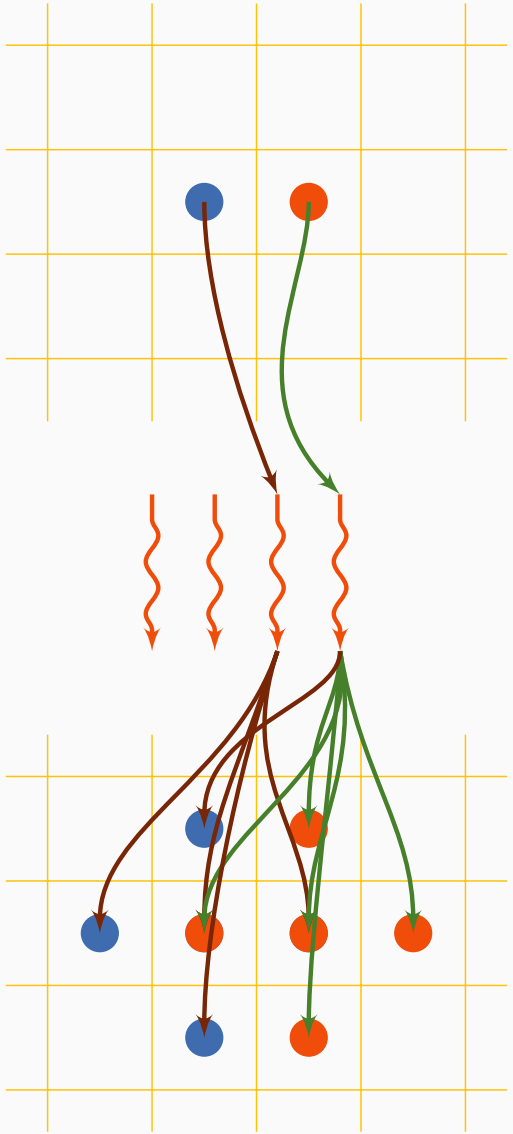


Scatter pattern

All the threads compute the memory space to which the output will be written.

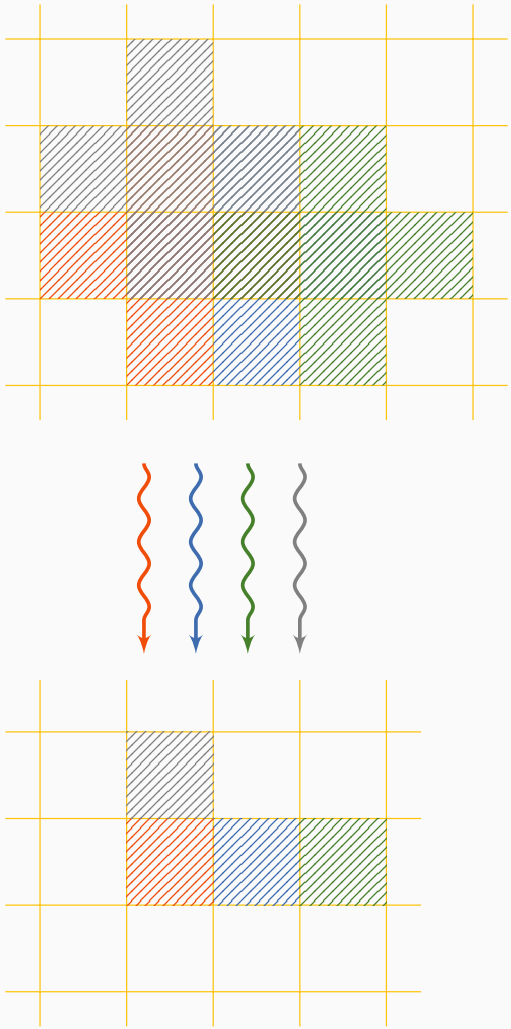


Scatter pattern (2)



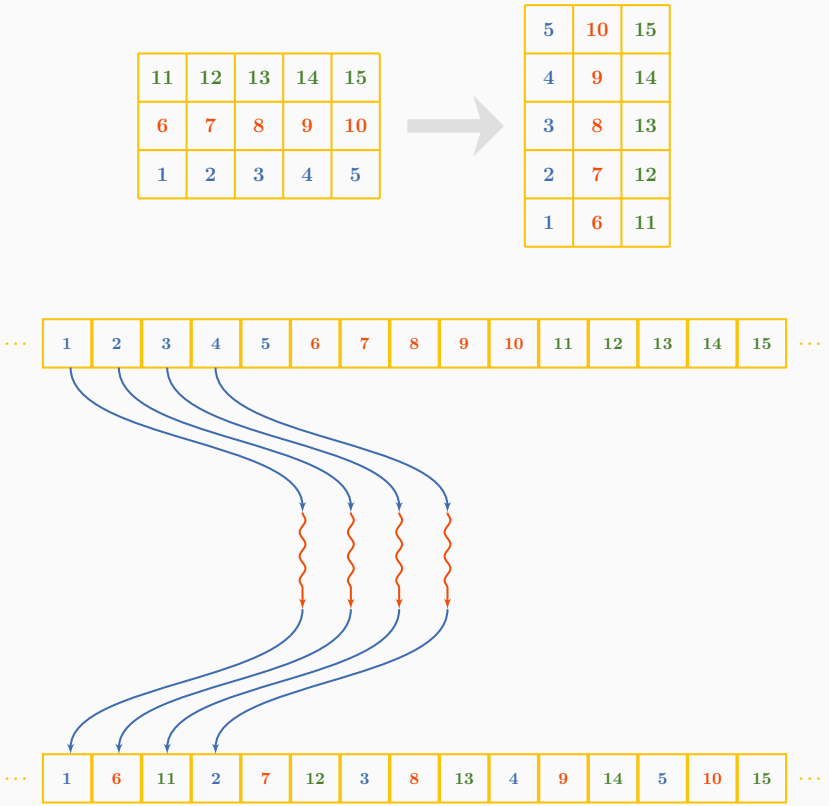
Stencil pattern

All the threads compute a part of an array using the neighbours. All the threads use the same stencil.



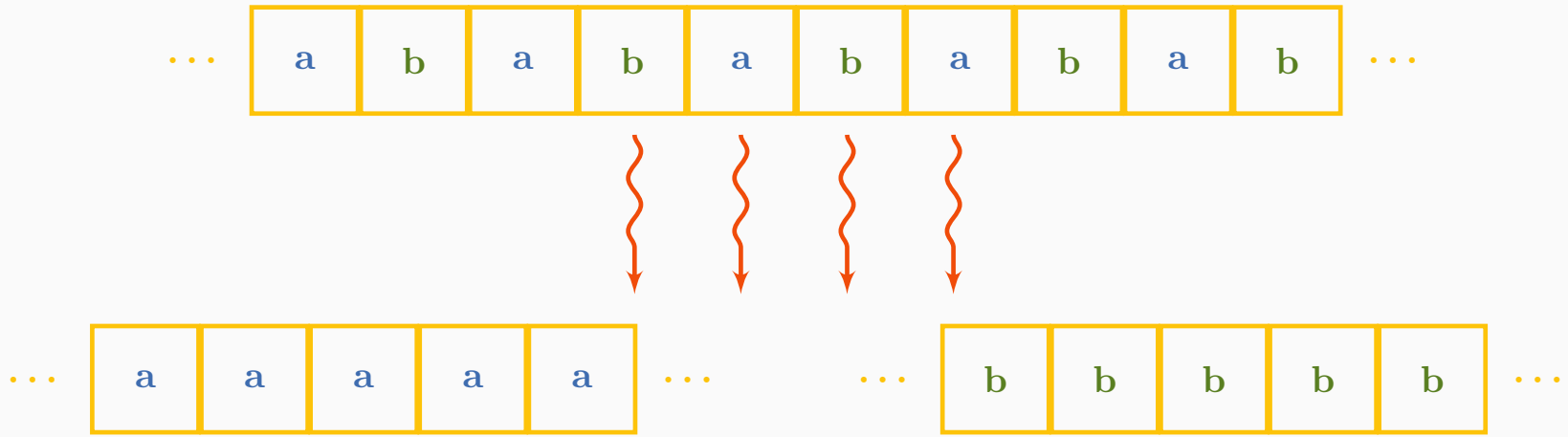
Transposition pattern (1)

All the threads read data in some array and rewrite it to some other part of the array. The position is well defined.

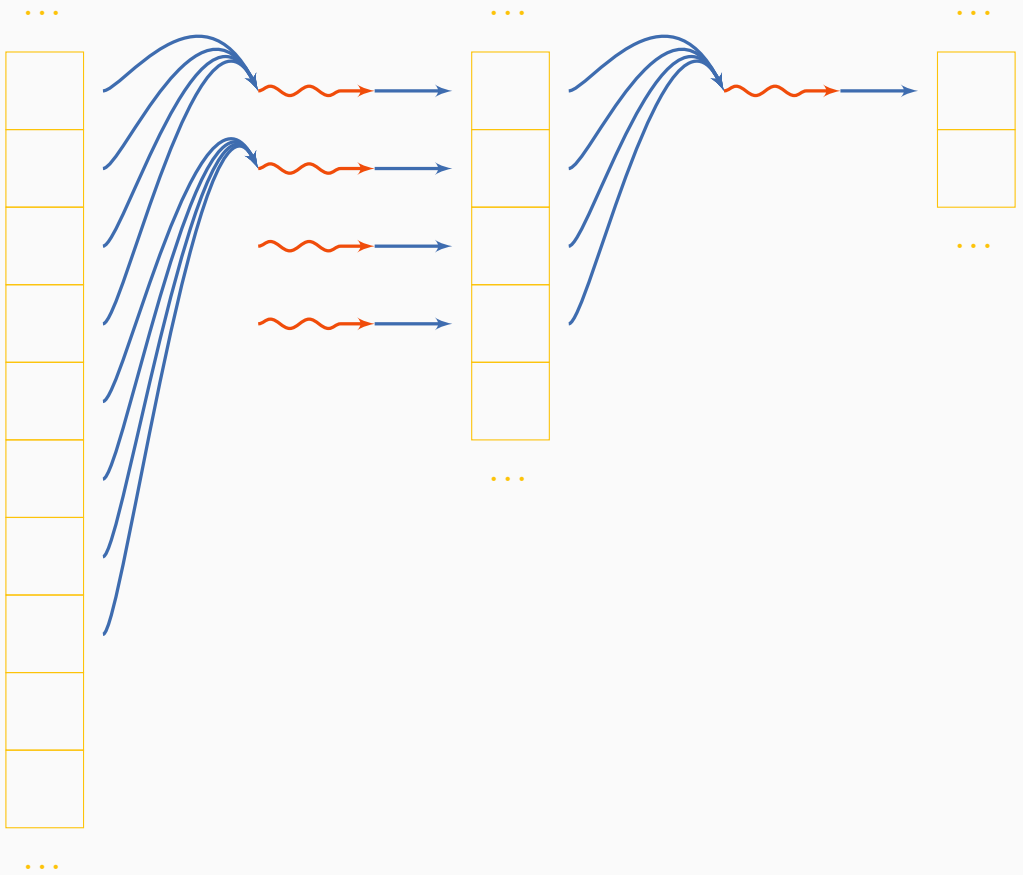


Transposition pattern (2)

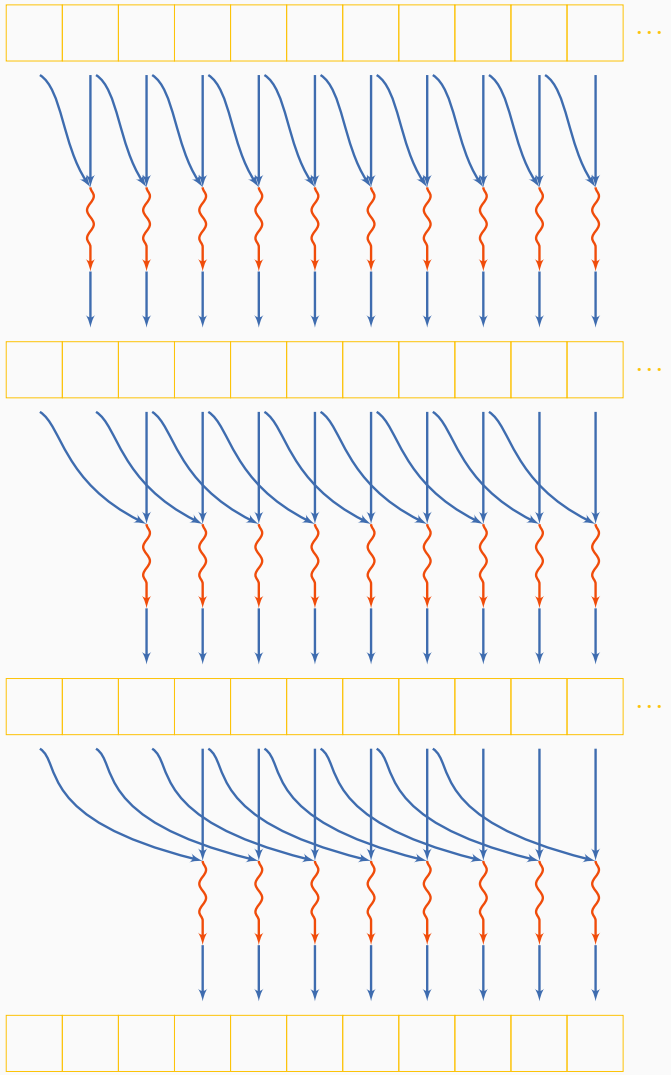
We can also use the concept of transposition for an arrays of structures to build a structure of array.



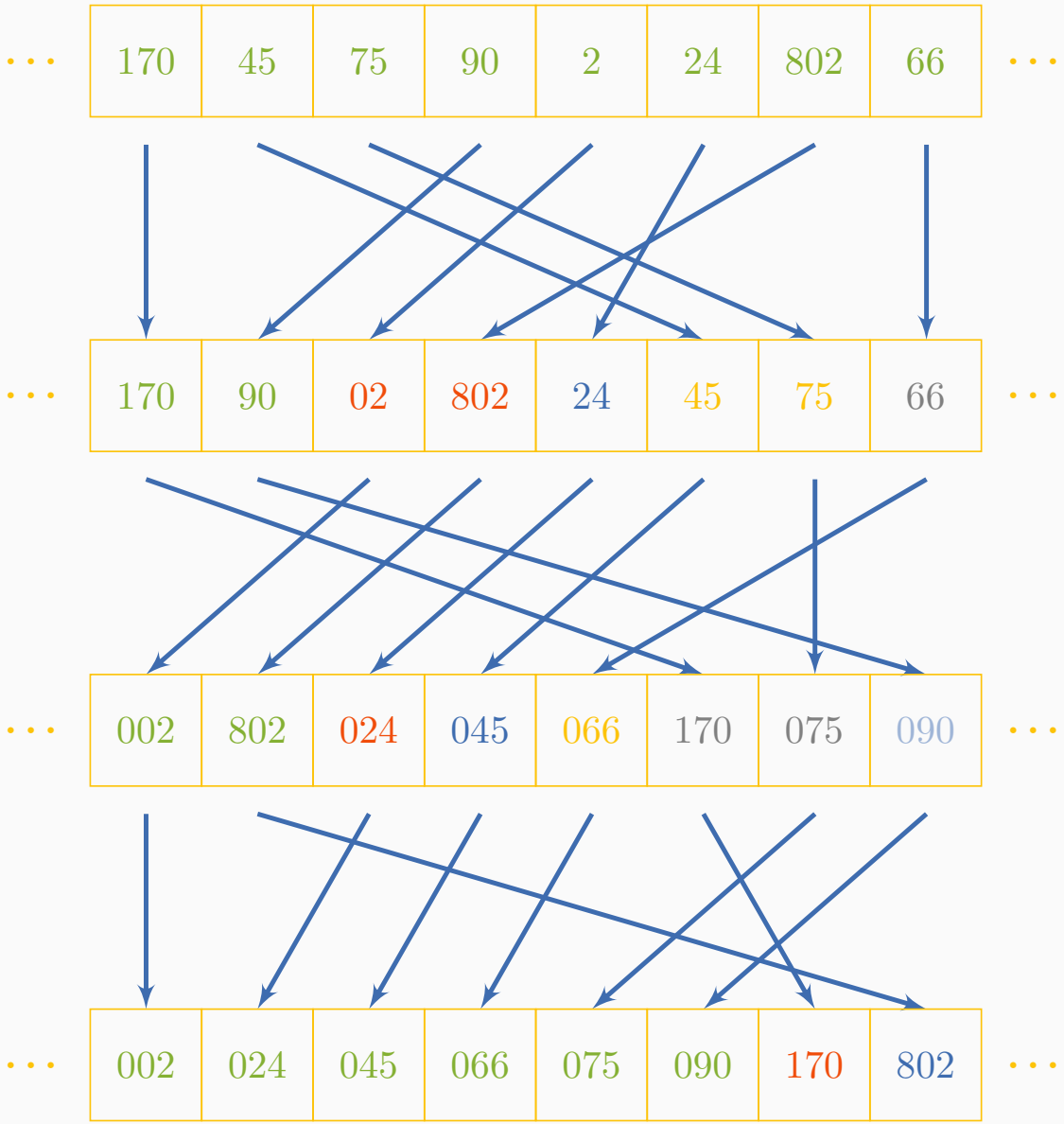
Réduction



Scan



Radix sort


170 45 75 90 2 24 802 66
170 90 02 802 24 45 75 66
002 802 024 045 066 170 075 090
002 024 045 066 075 090 170 802

Agenda

Parallel platforms : part 2

Some Examples

Models of parallelism

Specific patterns

Performances

- A sequential computation has 3 phases : setup, processing and completion.

$$T_{total} = T_{init} + T_{calcul} + T_{comp}$$

- If the computation is made using P units

$$T_{total}(P) = T_{init} + \frac{T_{calcul}}{P} + T_{compl}$$

Définition

We can define the speedup

$$S(P) = \frac{T_{total}}{T_{total}(P)}$$

and the efficiency

$$E(P) = \frac{S(P)}{P}$$

Amdhal's law (1)

- The part of the code that cannot be executed simultaneously represent a certain percentage of the total time

$$\gamma = \frac{T_{init} + T_{compl}}{T_{total}(1)}$$

Definition 1

Using γ, we can write the Amdhal's law

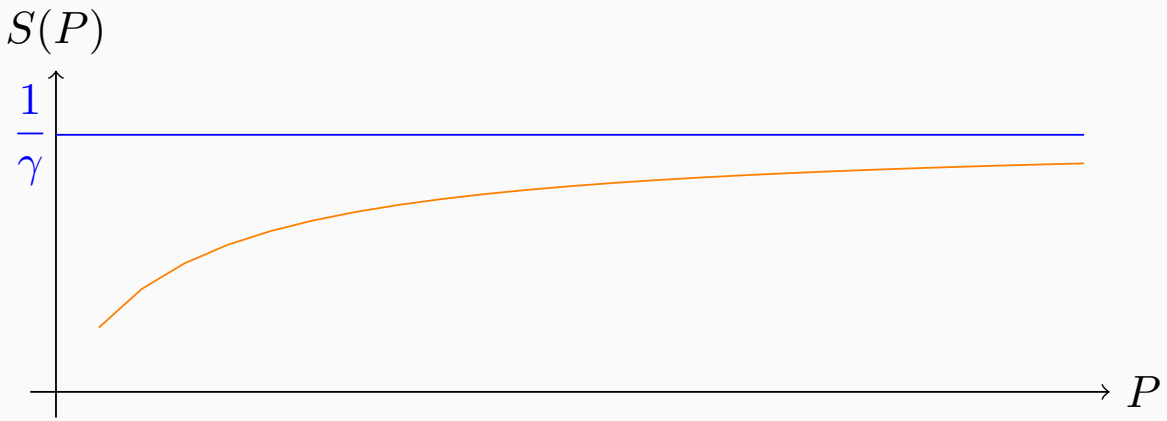
$$S(P) = \frac{T_{total}(1)}{\left(\gamma + \frac{1-\gamma}{P}\right) T_{total}(1)} = \frac{1}{1 + \frac{1-\gamma}{P}}$$

- Programming constraints are ignored : threads management, bottleneck, …
- Main hypothesis : parallelism does not have any intrinsic cost.

Amdhal's law (2)

- The scalar part γ contains all the timing corresponding to thread managements, scalar computations, os operations.
- The parallel part $1 - \gamma$ is completely parallel
- The upper bound for the speedup is given by

$$S(P) < \frac{1}{\gamma}$$



- If the workload does not increase, it is unnecessary to increase the number of processor.

- γ is normalized with respect to the number of processing units.

$$\gamma(P) = \frac{T_{init} + T_{compl}}{T_{total}(P)}$$

Definition 2

Gustafson's law writes

$$S(P) = P + (1 - P)\gamma(P)$$

- Using this law, we can study the impact of the number of processing unit on the computation.

Next time...

Introduction to distributed memory computing