Parallel Programming: Scheduling and Partitioning

Felice Pantaleo
CERN – Experimental Physics Department

felice@cern.ch

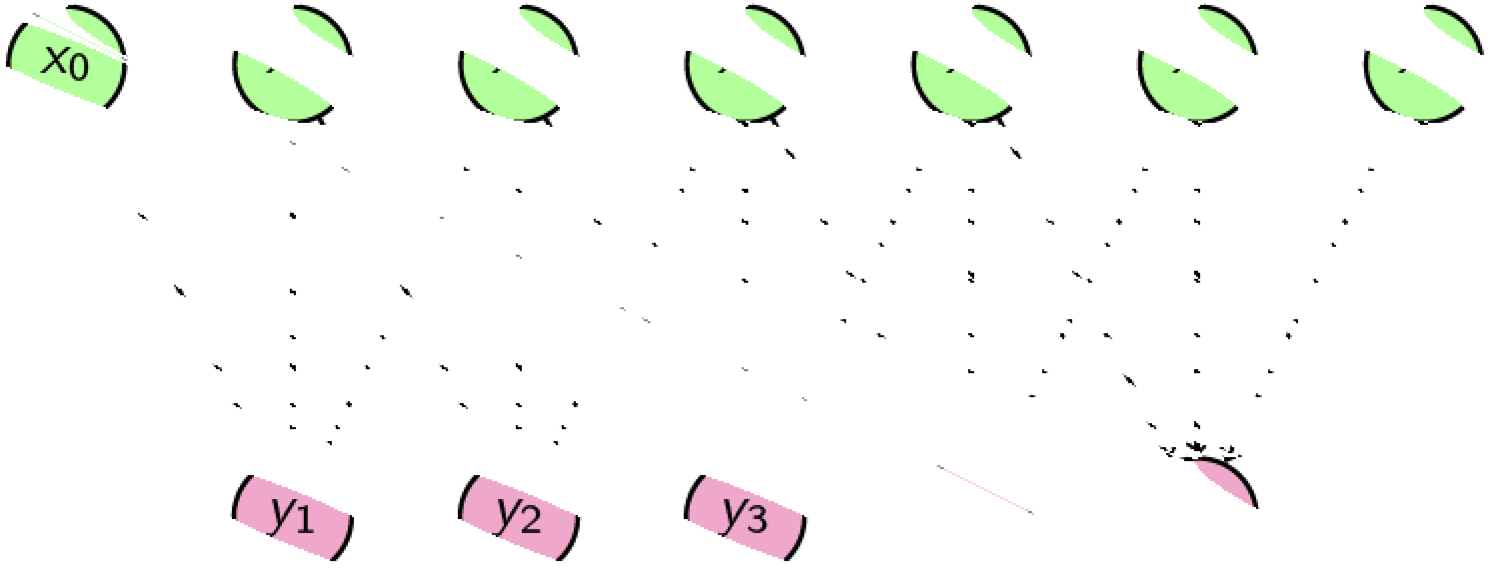
Mother-child parallelism

When thinking about possible parallel solutions:

- How to partition the problem
- How to share information

Data Partitioning

$$y_i = f_i(range(x_i, \delta))$$



Partitioning

- Static:
  - all information available before computation starts
  - use off-line algorithms to prepare before execution time
  - Run as pre-processor, can be serial, can be slow and expensive
- Dynamic:
  - information not known until runtime
  - work changes during computation (e.g. adaptive methods)
  - locality of objects can change (e.g. particles move)
  - use on-line algorithms to make decisions mid-execution
  - must run side-by-side with application
  - should be parallel, fast, scalable.
  - Incremental algorithm preferred (small changes in input result in small changes in partitions)

Why? In order to minimize idle time.

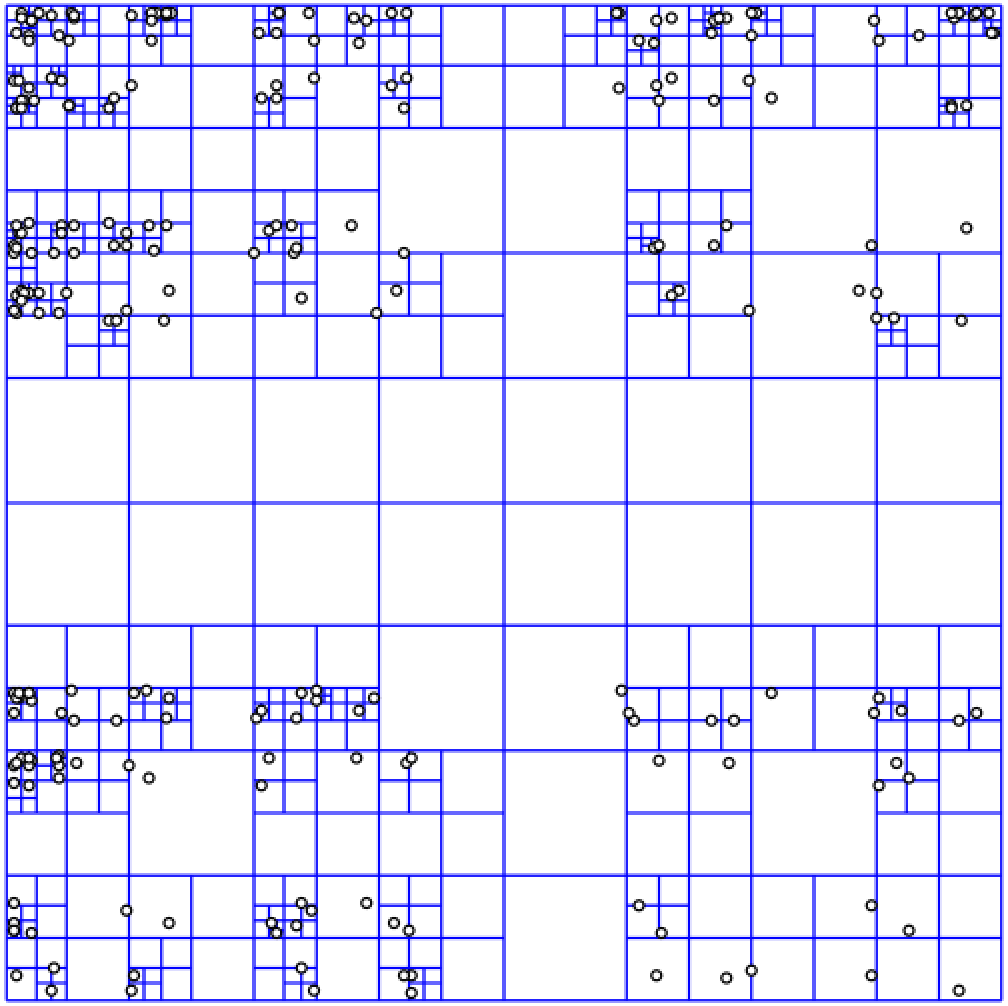
Load balancing

Sometimes dividing the input data in two does not mean that the load has been also divided in two.

Example:

Total load: 100

- If 5 workers take 20 each
  - Speedup 5
- If 1 worker takes 50
  - Speedup 2



Partitioning and Load Balancing

- Assignment of application data to processors for parallel computation
- Applied to grid points, elements, matrix rows, particles

Non-uniform data distributions

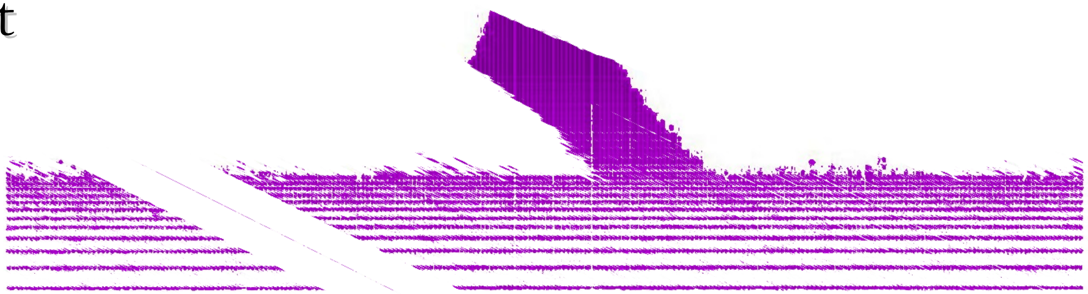
- Highly concentrated spatial

data areas

- Astronomy, medical imaging,

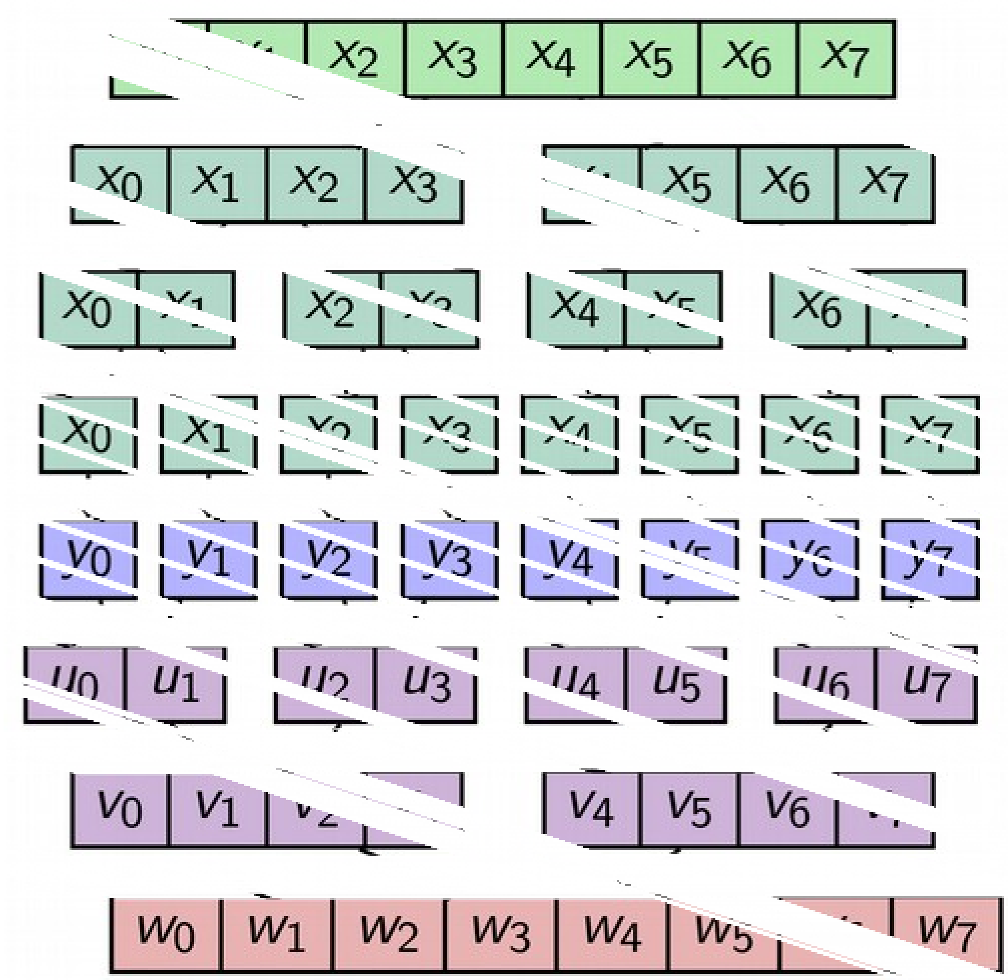
computer vision, rendering

If each thread processes the input data of a given spatial volume unit, some will do a lot more work than others



Divide et Impera

When you don't have any idea on how to approach the parallelization of a problem, try *Divide et Impera*



Load Imbalance

Sometimes load imbalance could also be caused by some underestimated consideration

- Example:

```
int N = 1000;
for(int i=0; i<N; ++i){
...
}
```

Load Imbalance

Sometimes load imbalance could also be caused by some underestimated consideration

- Example:

```
i_start = my_id * (N/num_threads);
i_end = i_start + (N/num_threads);
if (my_id == (num_threads-1))
   i_end = N;
for (i = i_start; i < i_end; i++) {
...
}
```

Load Imbalance

- The last thread executes the remainder

```
i_start = my_id * (N/num_threads);
i_end = i_start + (N/num_threads);
if (my_id == (num_threads-1))
  i_end = N;
for (i = i_start; i < i_end; i++) {
...
}
```

- If the number of threads is 32, each thread will execute 31 instructions
- The last thread will execute 8 more instructions
- Try to extrapolate to a bigger number of iterations and of threads!

CUDA Dynamic parallelism

- Enables a CUDA kernel to create and synchronize new nested work.
- A child CUDA kernel can be called from within a parent CUDA kernel
  - optionally synchronize on the completion of that child CUDA Kernel