LAU
الجامعة اللبنانية الأميركية
Lebanese American University

CSC 447: Parallel Programming for Multi-Core and Cluster Systems

Designing Parallel Programs

Instructor: Haidar M. Harmanani

Spring 2021

1. Designing Parallel Algorithms

- Design a parallel algorithm :
  - What is the maximum theoretical scalability of my algorithm ?
  - Is my algorithm still interesting in a few years, when many-core machines will be standard ?

Some serial algorithms are not meant to be implemented because they'll never run fast enough.

2. Code Serial and Optimize Serial Performance

- Code and optimize serial performance:
  - Thoroughly debug your code, optimize for serial performance, use performance libraries.
- Collect performance data using a profiler:
  - See how they match your algorithm predictions,
  - Prepare parallelization of your code,
  - Evaluate if parallelization is still worth it.

Collect detailed performance data.

3. Introduce Parallelism

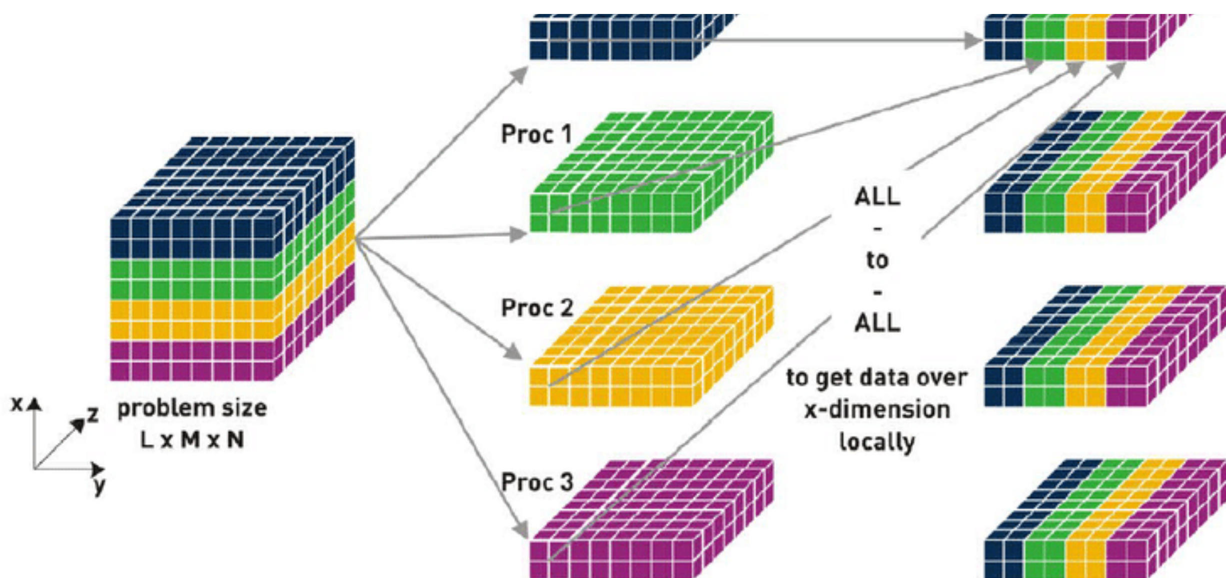
- Introduce parallelism :
  - Pick the right technology for you problem.
    - (OpenMP, OpenACC, MPI, CUDA, …)
  - Pick the right place in your code to introduce it.
  - Predict maximum scalability based on serial performance data.

4. Debug Parallel Code

- Debug parallel bugs :
  - A perfectly working serial code can give wrong results when ran in parallel if the parallelism was not introduced correctly.
  - A serial debug tool won't help.

5. Optimize Parallel Performance

- Optimize parallel performance :
  - When your serial performance problems are solved, you'll have a clear view of your parallel performance problems.
  - System tools can help you collect the right information.



Decomposition

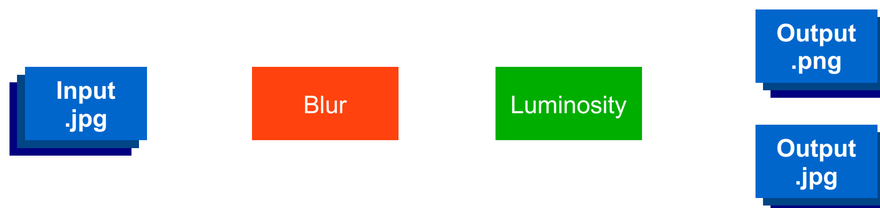
www.computerhistory.org

Decomposition

- To design a parallel algorithm or parallelize an existing code, you first have to understand if you have dependencies.
- If data can be processed or tasks executed independently, you can process them in parallel.

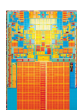
Example

- Suppose you want to apply two filters on a batch of images and save them in jpg and png formats.
  - Can you process different images in parallel ?
  - Can you process different pixels in parallel ?
  - Can you open/save files, apply filters in parallel ?

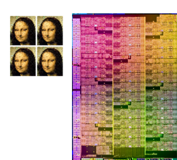


Data decomposition: Parallel Treatment of files

- In our case, data decomposition means you'll try to execute in parallel the treatment of different images or different pixels.
- Images are independent from each other, it's perfect and easy to implement. But if you have a low number of images to treat and a large number of cores available, you won't use them all.



A lot of images and a dual-core : easy to use all cores efficiently.



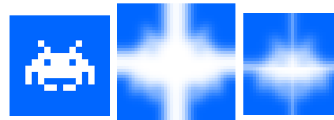
Few images and a many-core (50+) : data decomposition by file only won't be enough to use all cores.

Data decomposition: Parallel treatment of pixels

- For pixels, it's easy to do for a basic luminosity filter as each pixel is processed independently from the others. But the blur filter requires information from neighboring pixels.
- It can be done in parallel with some approximations or a lot of communication between threads (synchronization), impacting scalability.

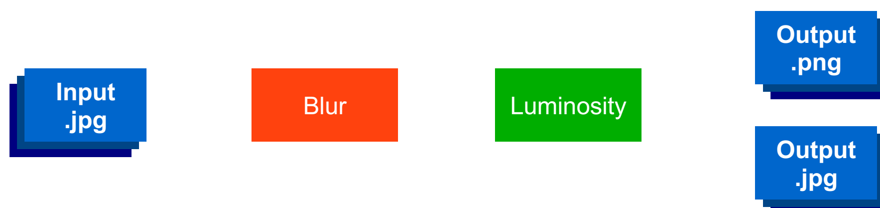
Luminosity adjustment can be done independently on each pixel.
Easy to parallelize efficiently.

Blur filter requires information from other pixels. Requires heavy synchronization OR flexibility regarding the result.



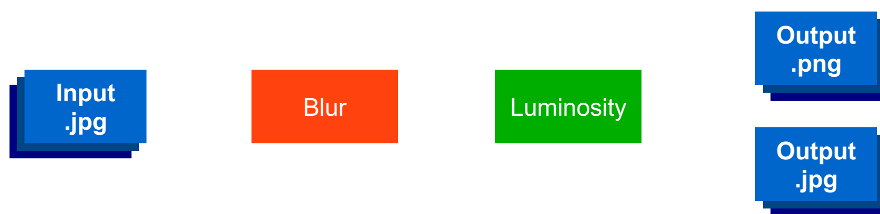
Task decomposition

- Another way to solve our problem is to execute different operations on the same data in parallel.
- Saving as .png and saving as .jpg are totally independent (and CPU intensive) operations.
  - Easily be parallelized.
- But luminosity and blur can't be done in parallel on the same image independently.



Data and task decompositions

- For our problem, each decomposition has pros and cons depending on the the number of files to process, precision required for the blur filter, number of cores, developer skills and time …
- A good real life solution would be to implement different levels of nested parallelism mixing data and task decomposition.



Methodology

- Study problem, sequential program, or code segment
- Look for opportunities for parallelism
- Try to keep all cores busy doing useful work

Ways of Exploiting Parallelism

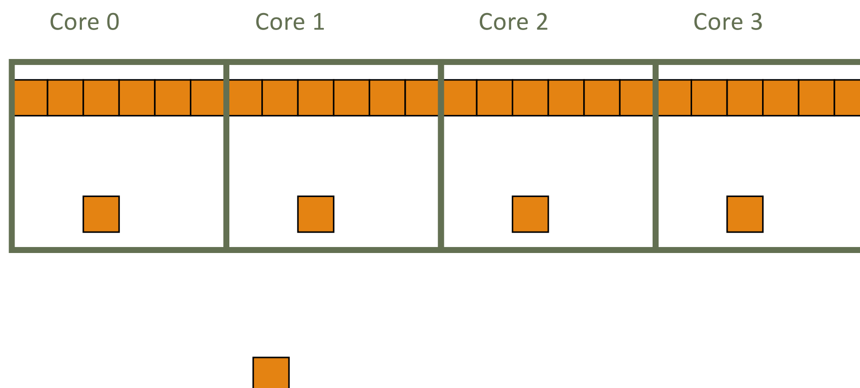
- Domain decomposition
- Task decomposition
- Pipelining

Domain Decomposition

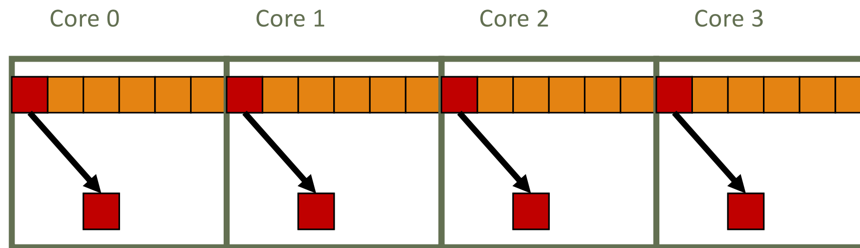
- First, decide how data elements should be divided among cores
- Second, decide which tasks each core should be doing
- Example: Vector addition

Domain Decomposition: Find the Largest Element of an Array

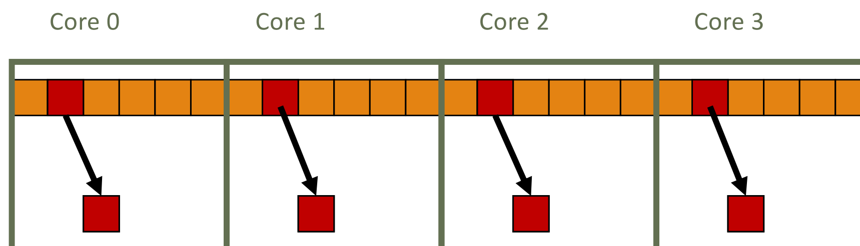
Domain Decomposition: Find the Largest Element of an Array



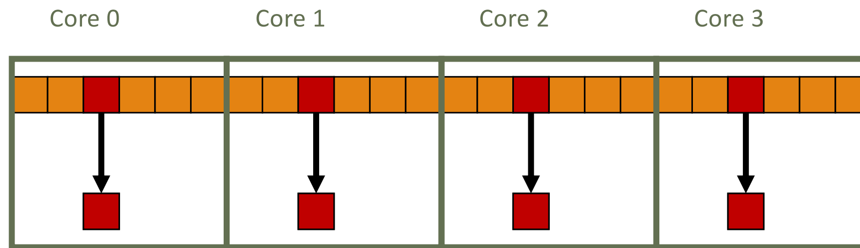
Domain Decomposition: Find the Largest Element of an Array



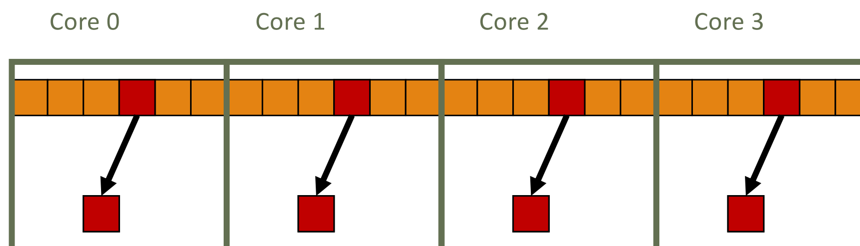
Domain Decomposition: Find the Largest Element of an Array



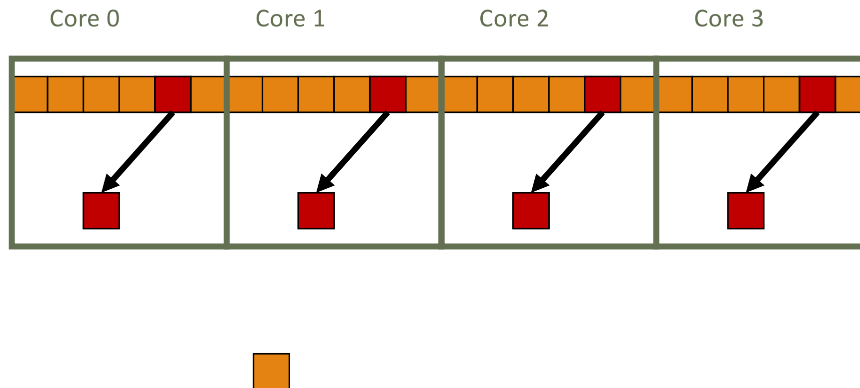
Domain Decomposition: Find the Largest Element of an Array



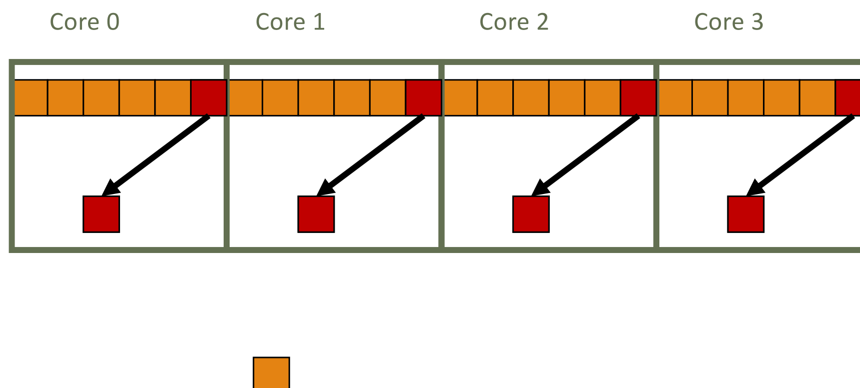
Domain Decomposition: Find the Largest Element of an Array



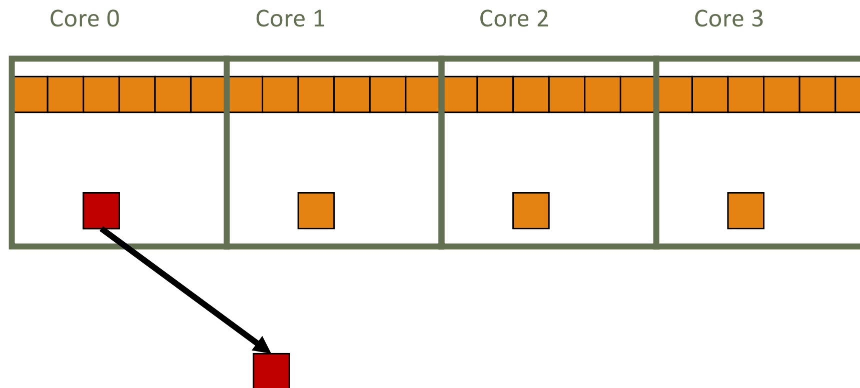
Domain Decomposition: Find the Largest Element of an Array



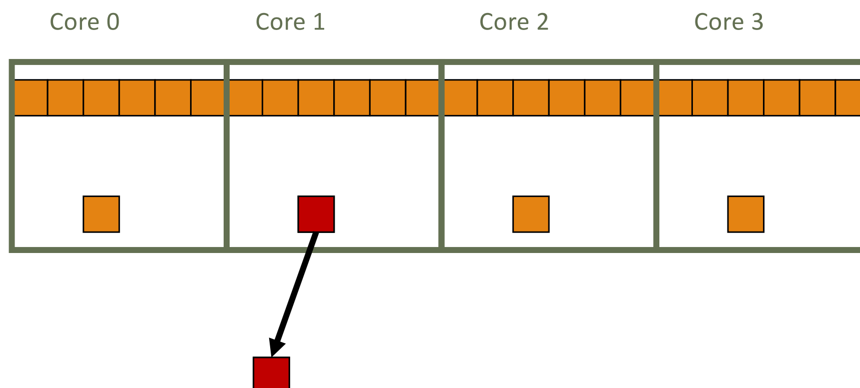
Domain Decomposition: Find the Largest Element of an Array



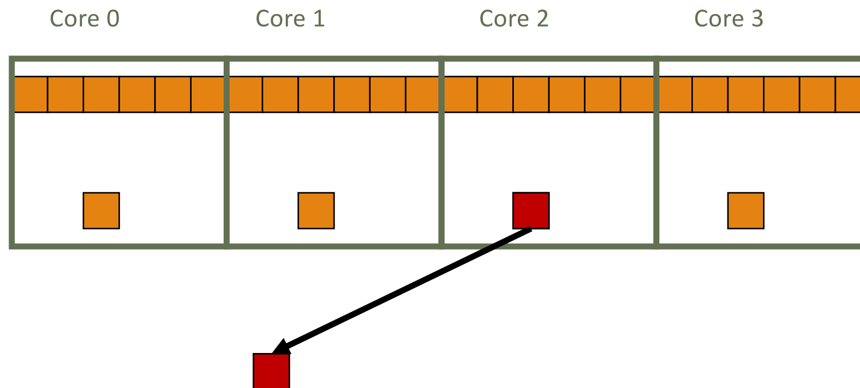
Domain Decomposition: Find the Largest Element of an Array



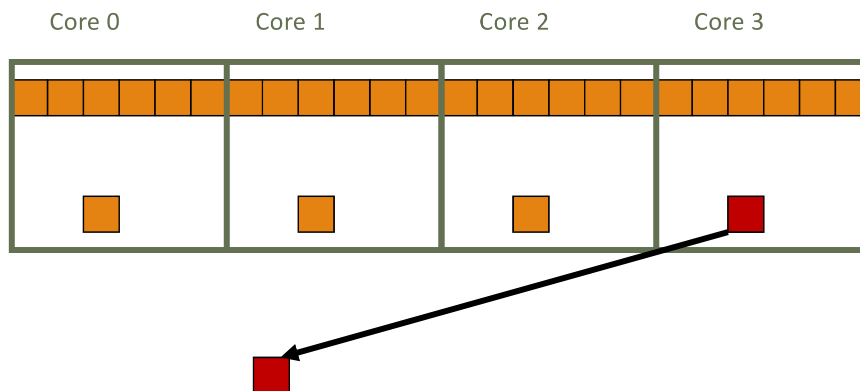
Domain Decomposition: Find the Largest Element of an Array



Domain Decomposition: Find the Largest Element of an Array



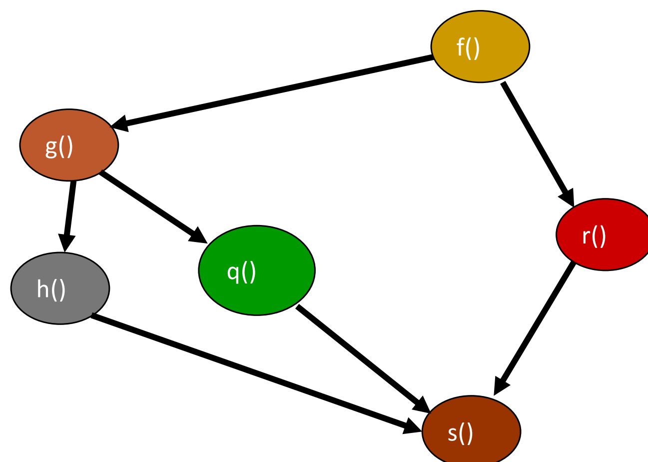
Domain Decomposition: Find the Largest Element of an Array



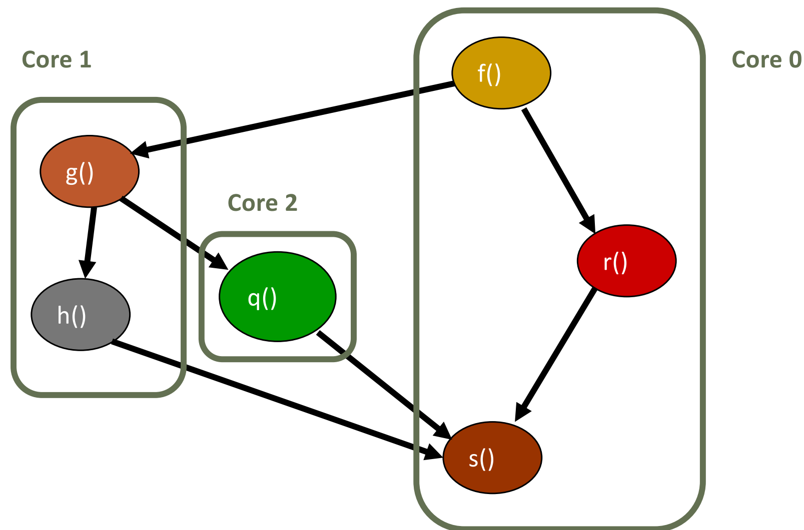
Task (Functional) Decomposition

- First, divide problem into independent tasks
- Second, decide which data elements are going to be accessed (read and/or written) by which tasks
- Example: Event-handler for GUI

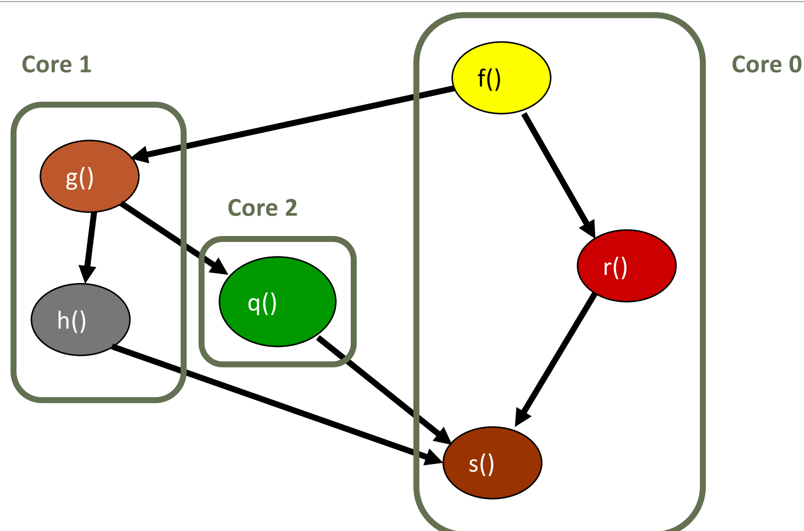
Task Decomposition



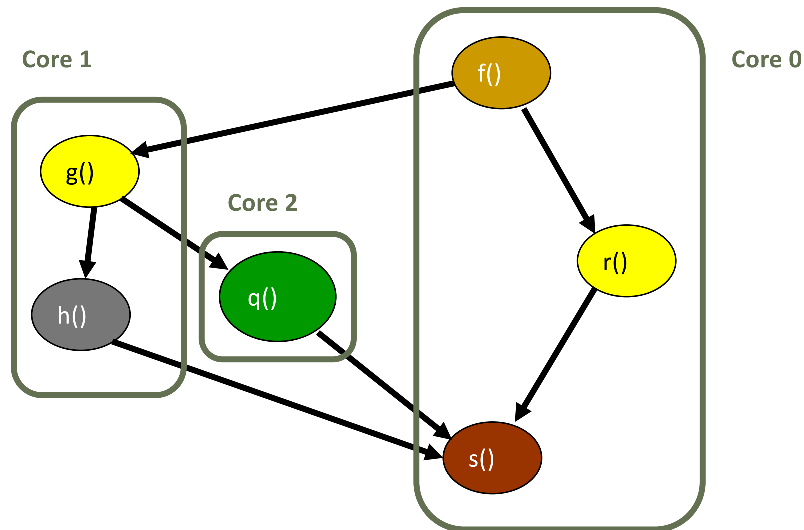
Task Decomposition



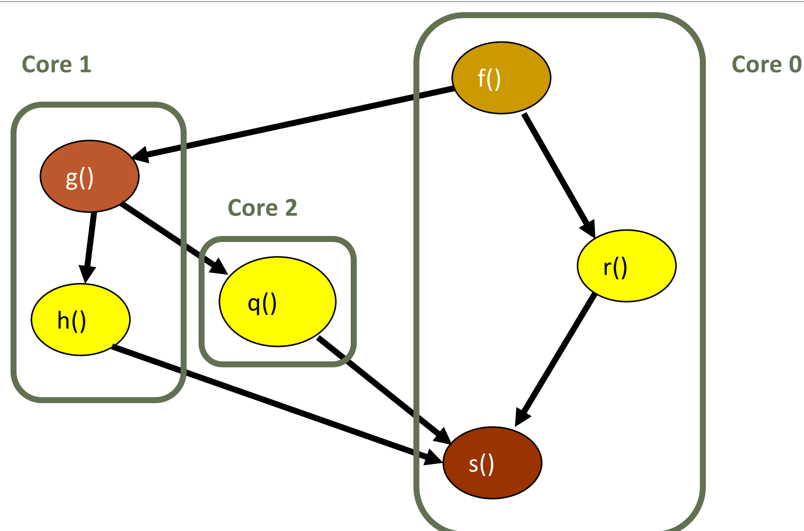
Task Decomposition



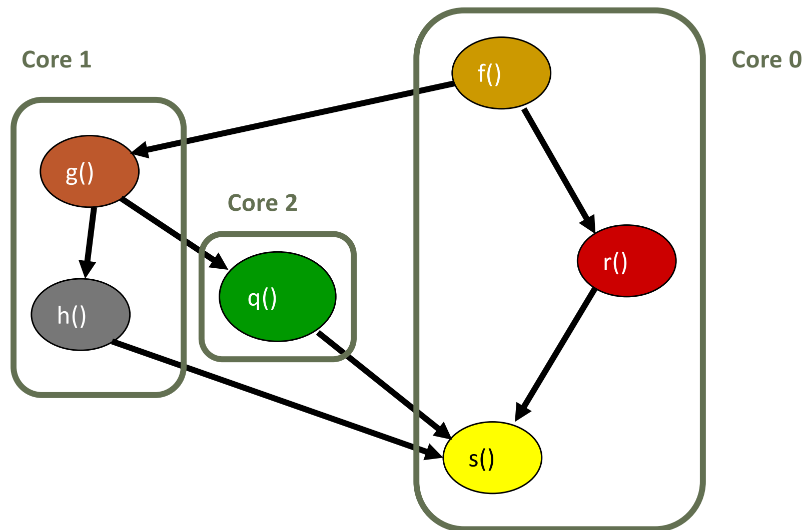
Task Decomposition



Task Decomposition

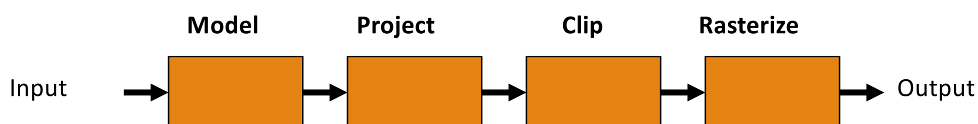


Task Decomposition

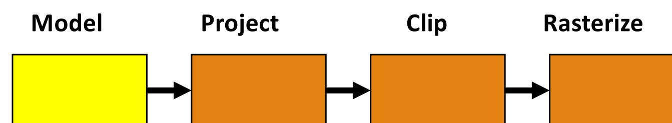


Pipelining

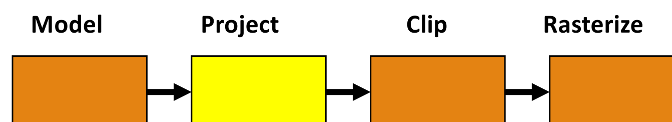
- Special kind of task decomposition
- "Assembly line" parallelism
- Example: 3D rendering in computer graphics



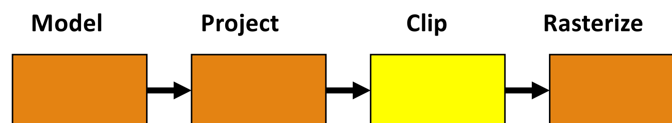
Processing One Data Set (Step 1)



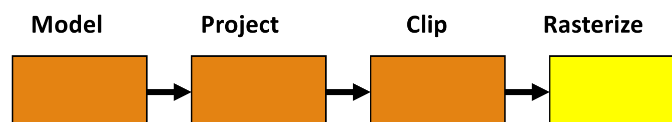
Processing One Data Set (Step 2)



Processing One Data Set (Step 3)

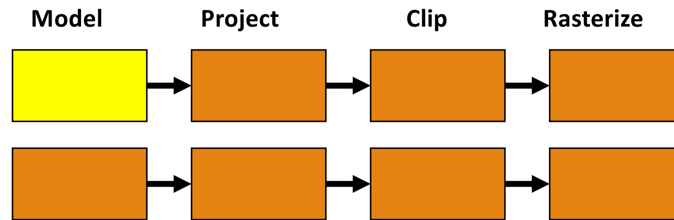


Processing One Data Set (Step 4)

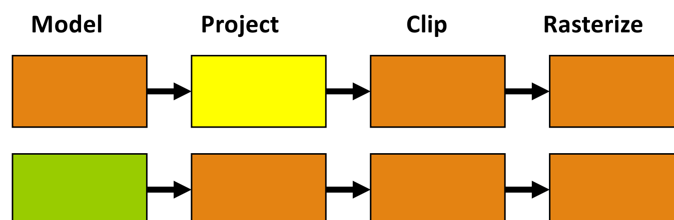


The pipeline processes 1 data set in 4 steps

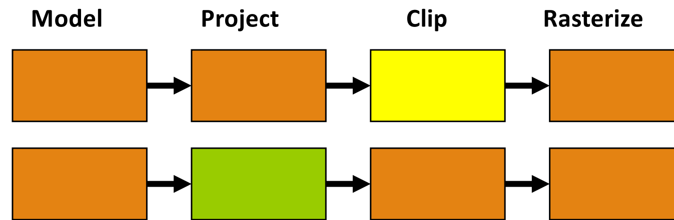
Processing Two Data Sets (Step 1)



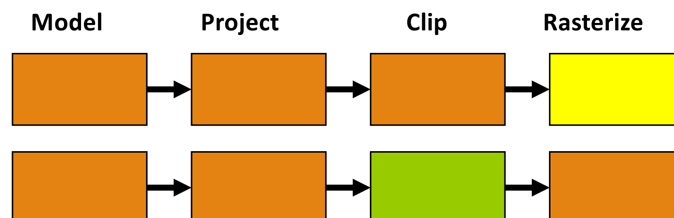
Processing Two Data Sets (Time 2)



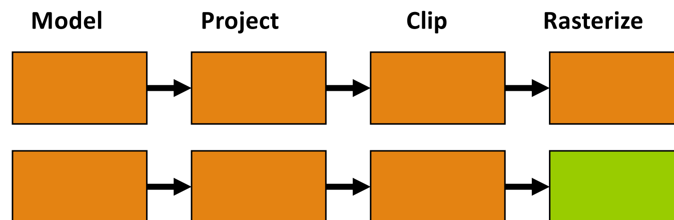
Processing Two Data Sets (Step 3)



Processing Two Data Sets (Step 4)

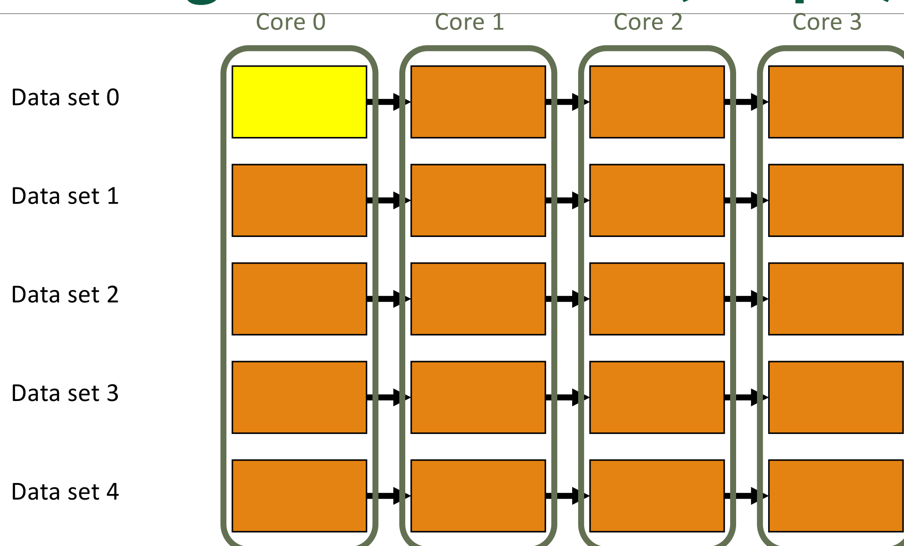


Processing Two Data Sets (Step 5)

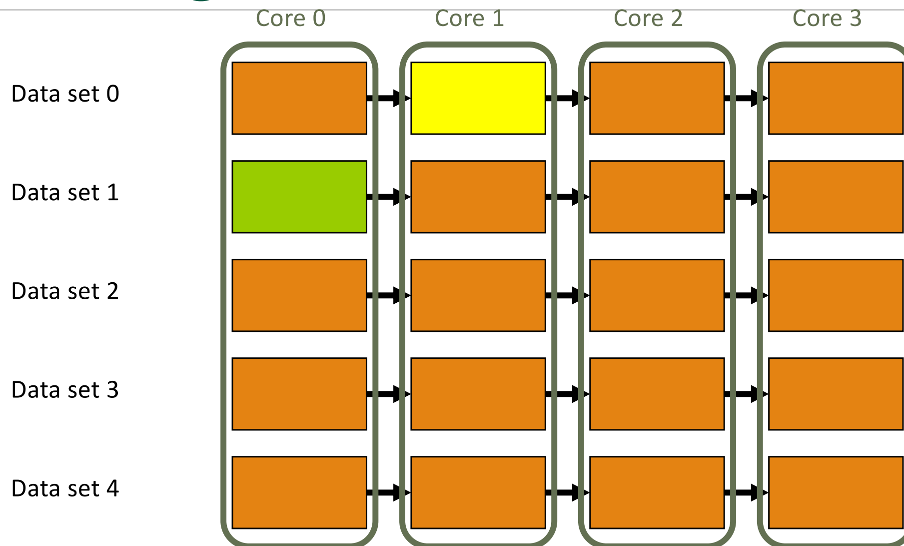


The pipeline processes 2 data sets in 5 steps

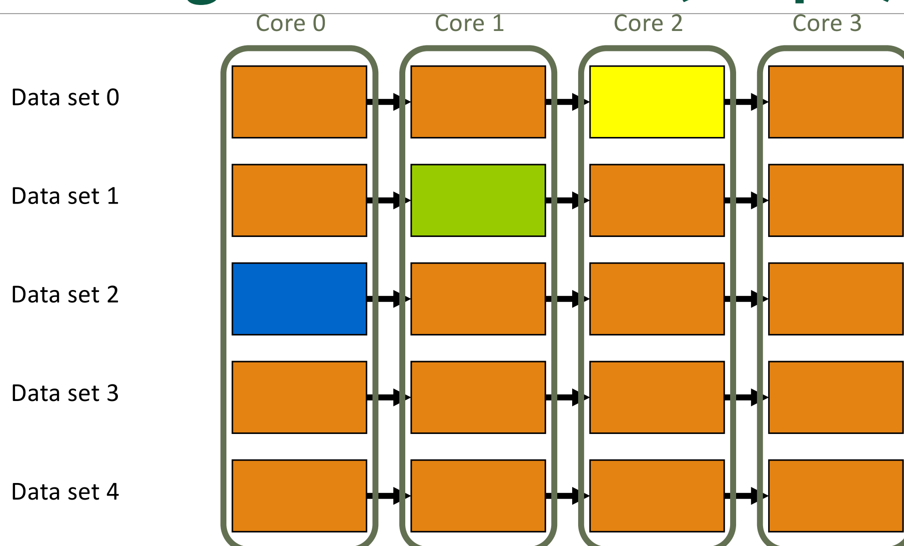
Pipelining Five Data Sets (Step 1)



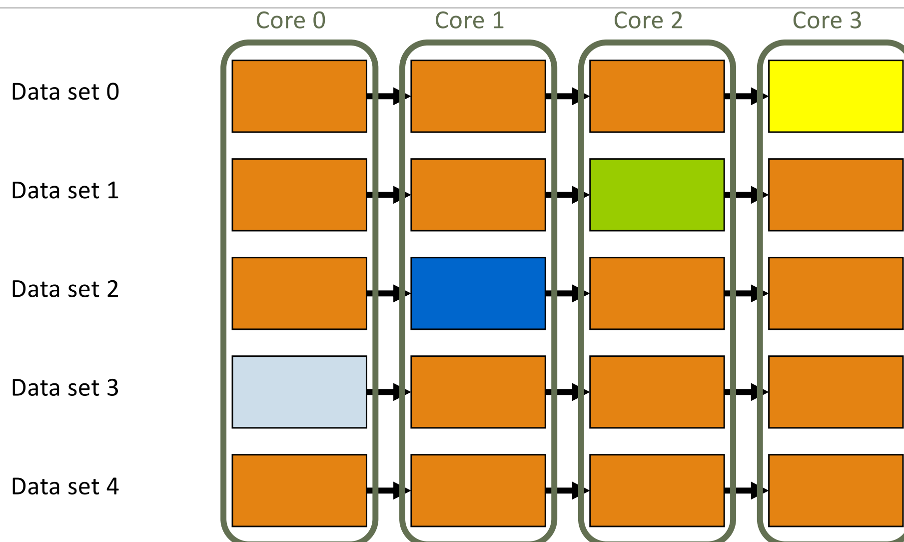
Pipelining Five Data Sets (Step 2)



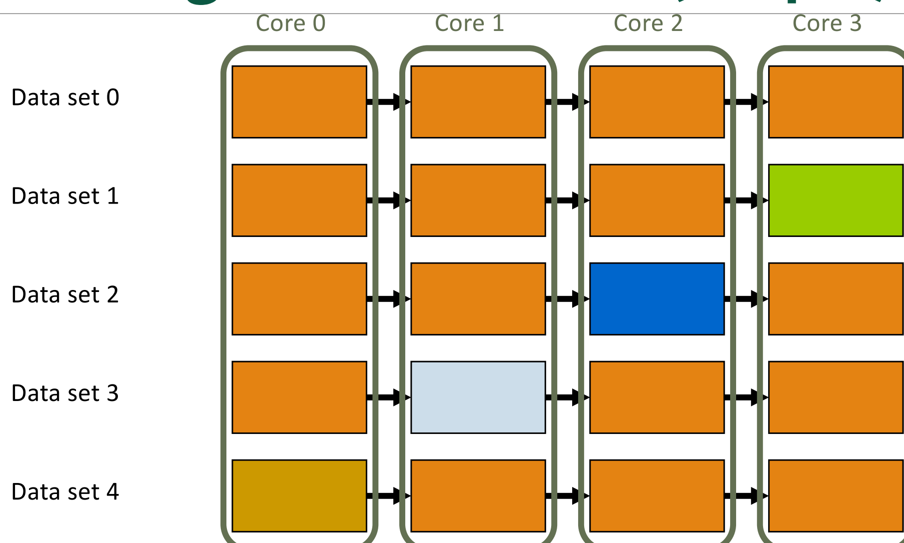
Pipelining Five Data Sets (Step 3)



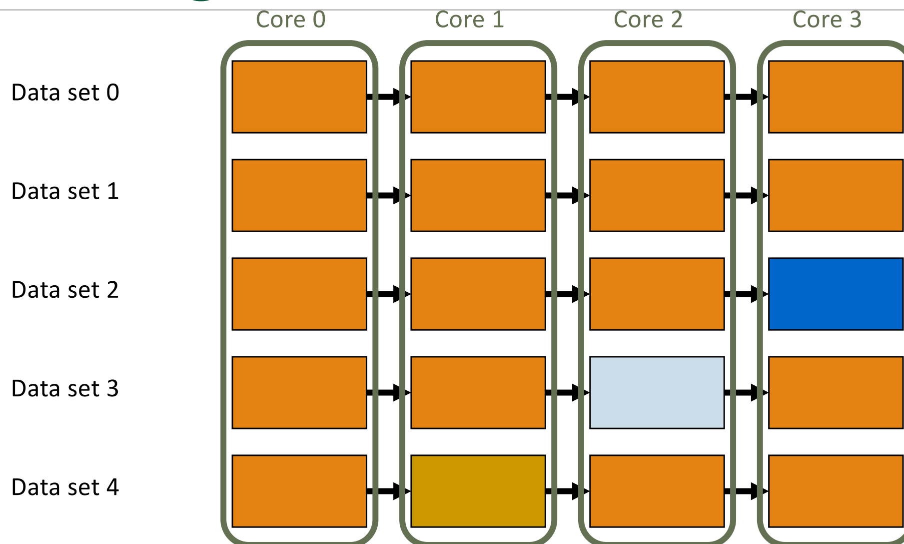
Pipelining Five Data Sets (Step 4)



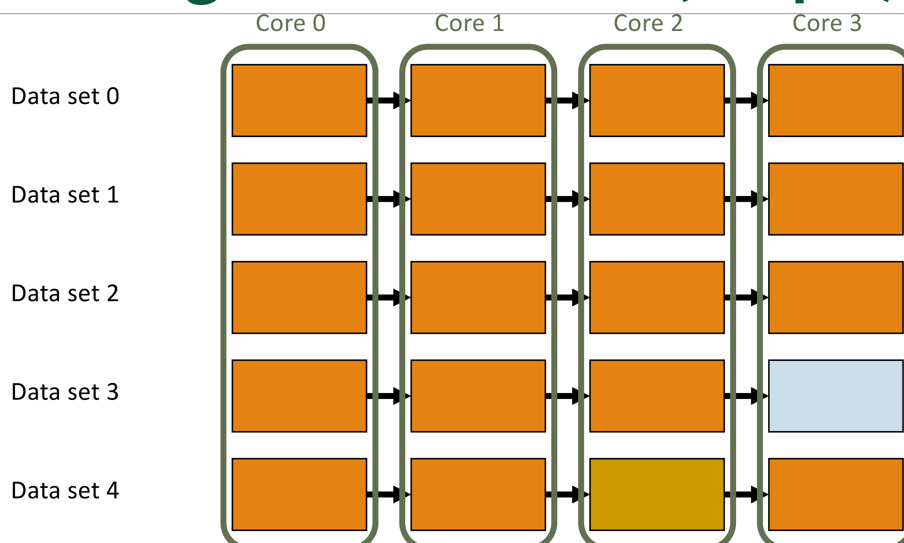
Pipelining Five Data Sets (Step 5)



Pipelining Five Data Sets (Step 6)



Pipelining Five Data Sets (Step 7)



Pipelining Five Data Sets (Step 8)