CS 6023 - GPU Programming

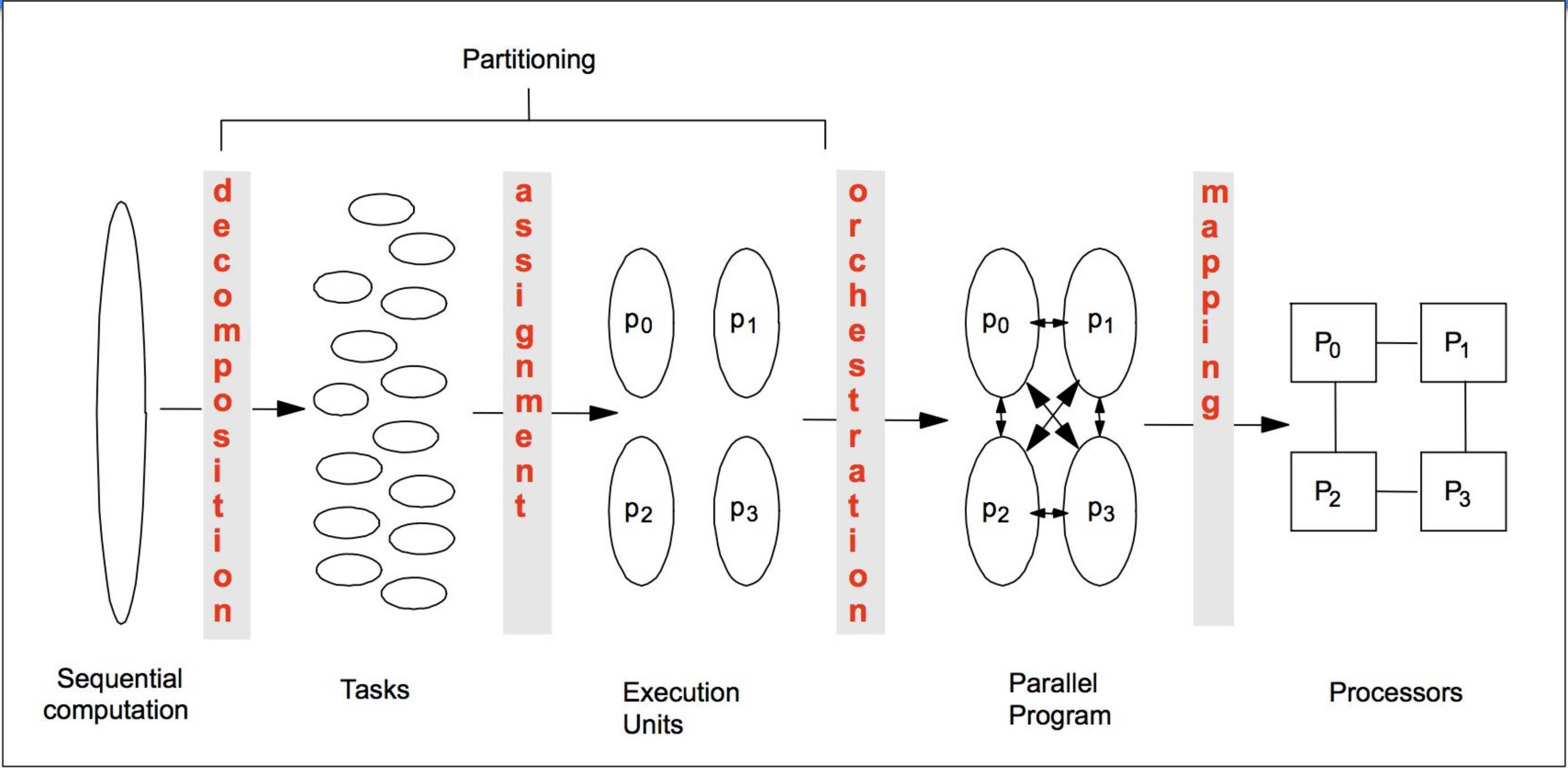
Parallelisation Thinking

23/09/2018

Setting and Agenda

- What are the patterns of parallel programming in general (ie. beyond GPUs)?
- What are the specific characteristics of GPU programming?
- How to approach a new parallelisation task

Steps in Parallelisation



Dr. Rodric Rabbah, 6.189 MIT

Parallelising Programs

Patterns for Parallel Programming. Mattson, Sanders, and Massingill (2005)

Identify the 4 aspects to parallelising programs:

1. **Algorithm expression**
   a. **Finding concurrency** - Expose concurrent tasks
   b. **Algorithm structure** - Map tasks to units of execution
2. **Software construction**
   a. **Supporting structures** - Code and data structuring patterns
   b. **Implementation mechanisms** - Low level programming mechanisms

Finding concurrency

- Types of concurrency:
  - Data
    - Geometric decomposition
    - Recursive data
  - Task
    - Linear task decomposition
    - Divide and conquer
  - Dataflow
    - Pipeline
    - Event-driven

Finding concurrency

- Types of concurrency:
  - Data
    - Geometric decomposition - matrices
    - Recursive data - tree, graph
  - Task
    - Linear task decomposition - ray tracing, particle simulation
    - Divide and conquer - merge sort
  - Dataflow
    - Pipeline - digital signal processing
    - Event-driven - process control

For GPUs ...

- The primary type of concurrency in GPUs is (geometric) data concurrency
  - Why? SPMD / SIMD semantics
  - Architecture evolved with graphics workload
- Applications can have range of data concurrency
  - Some can be embarrassingly parallel
    - Example $y_i = f(x_i)$ where f has no side-effects
  - Some can have control dependencies
    - Example: barrier synchronization, atomic operations
- Task parallelism also found, but limited in application
  - Example: ray tracing, particle physics

Algorithm structure

- Mapping application to units of execution
- This is a major challenge in distributed parallel programming (eg. cluster of PCs)
- However, with GPUs we are not concerned with mapping of tasks to SMs
  - Instead, we map tasks to threads, warps, blocks, and grids with well identified semantics
- Mapping decisions are driven by shared memory, registers, memory access patterns, synchronisation

Memory considerations in GPU

- We use the shared memory paradigm for GPUs, again inheriting form the graphics workload
  - Sharp contrast to message passing (eg. MPI) which could have point-to-point or broadcast communication patterns
- Memory shared between threads improves effective memory bandwidth
- But sharing memory between all threads can be a bottleneck too (atomics, serialization, large multi-ported memories)
- GPUs provide a mid-way solution: threads in the same block share faster memory, all threads share slower global memory

Software construction

- Code and data structuring patterns
  - In the non-GPU parallel world, these are significant challenges
  - For GPUs:
    - Code: careful alignment of warps to avoid control divergence
    - Data: patterns like tiling, privatisation, cached constant memory
- Low level programming mechanisms
  - Again, in non-GPU parallel world, these are major focus areas
  - For GPUs:
    - Memcopies, atomics, synchronization, streams (yet to cover)

Parallel paradigms other than SPMD

- Loop parallelism
  - Assign each iteration of a loop to a different unit
- Master / worker
  - Master maintains a list of tasks to complete and workers register for pending work when free
- Fork / join
  - Each process can fork and join when new work is created and completed dynamically

Which works well where

	Task-level decomposition		Data decomposition		Data-flow based	
	Linear tasks	Divide and conquer	Geometric	Recursive	Pipeline	Event-driven
SPMD	***	**	****	**	***	*
Loop	***	**	***			
Master/worker	****	**	*	*	****	*
Fork/join	**	****	**		****	****

Exercise: Add another dimension between shared memory and message passing

Modified version from Dr. Rodric Rabbah, 6.189 MIT