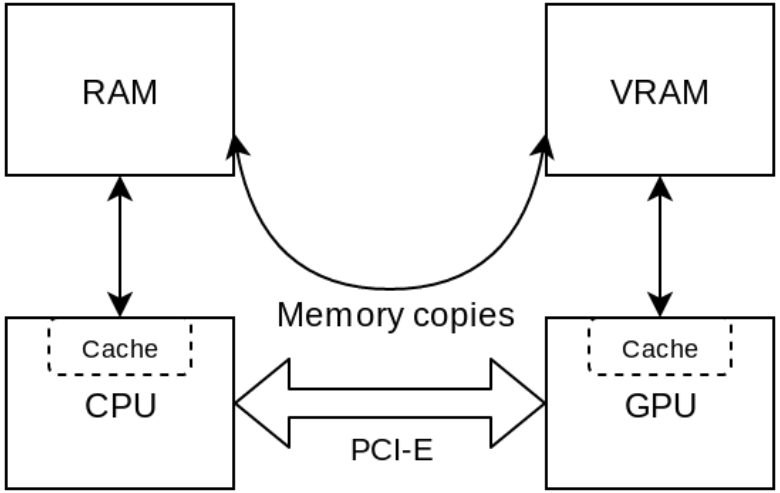
Heterogeneous CPU-GPU Processors Pipeline Optimisation Opportunities

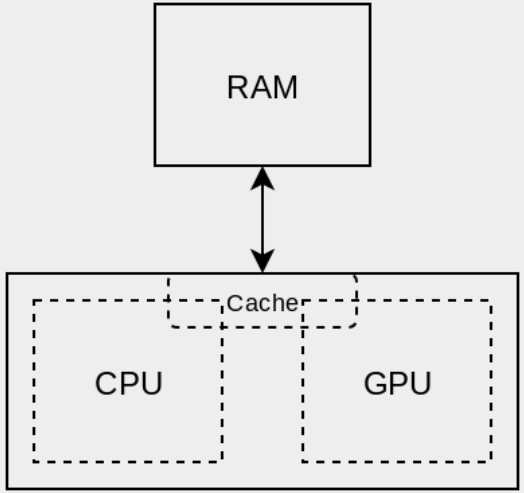
Comparison with Discrete GPU Computing

Heterogeneous System Architecture

Discrete GPU



Heterogeneous CPU-GPU Processor



Optimisations Targets

- run-time improvements with eliminated memory copies
- increased parallelism with in-memory synchronisation mechanisms
- chunked work to improve cache usage

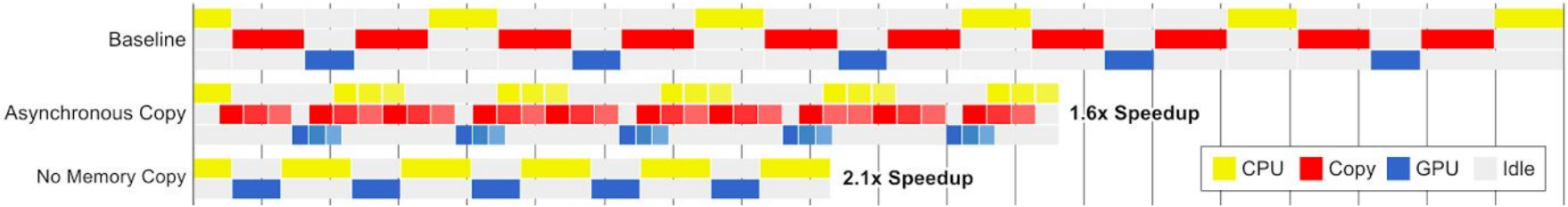
Benchmarks’ Measurements

“Copy” and “Limited-Copy” versions

- memory footprint - *access reason*
- memory accesses - *components memory usage*
- run-time - *components relative activity*

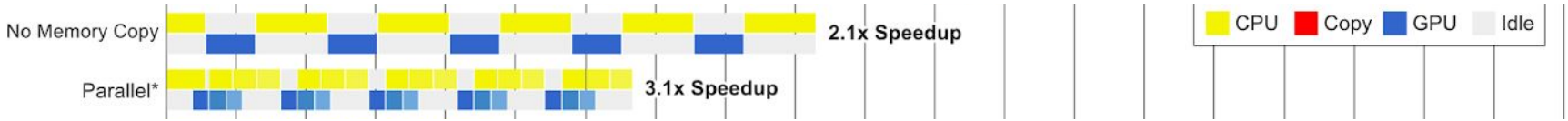
Memory Copies

- 70% of the data is used on the GPU
  - copied at some point
  - asynchronous page synchronisation for linked data-structures
- 10% of total memory accesses
  - eliminating copies brings little cache improvements
- 10% run-time improvement
  - PCI-E engine copy time



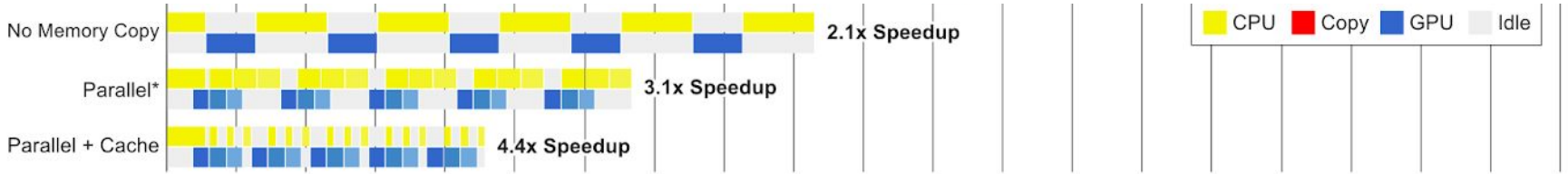
Pipeline

- weak CPU-GPU activity overlapping
- overlap control code and launch latency
  - asynchronous streams
  - in-memory synchronisation
- 30% run-time improvement by shifting work to underutilized cores
  - difficult to implement



Cache

- bulk-synchronized pipelines don’t benefit from cache
  - long reuse timespan
- 2% performance loss due to cache spills
  - inter-stage data exceeds cache capacity
- 60% memory access due to cache contention
  - stage dataset exceeds cache capacity
  - reach memory bandwidth limit



Conclusions

- need to increase programmer control over cache
  - help identify key data for concurrency
  - avoid cache contention
- easier programming models for heterogeneous processors
  - load-balancing
  - shared memory synchronization
  - cache