CERN School of Computing

Development of High Performance Computing Applications Across Heterogeneous Systems

*Lecture **1***

Scalable Parallel Computing

André Pereira

LIP-Minho/University of Minho

Inverted CERN School of Computing, 23-24 February 2015

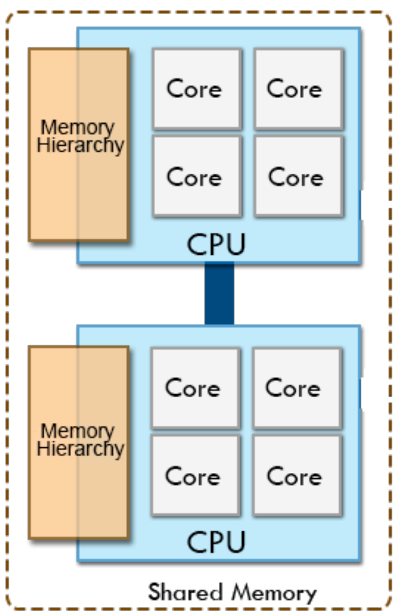
Agenda

- **Motivation**
- Heterogeneous Platforms (HetPlats)
- Levels of Parallelism
- Performance Scalability
- Performance Portability

Motivation

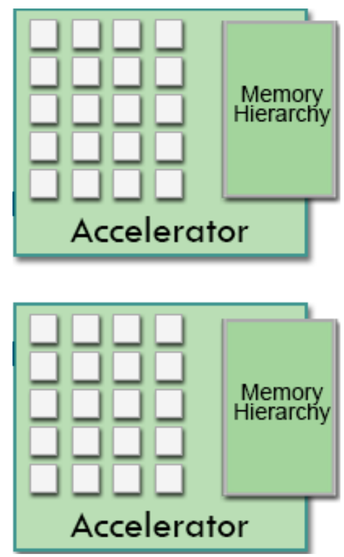
Common Parallel Approach

- CPUs have multiple complex computing cores
  - Use processes/threads to parallelise the code



Common Parallel Approach

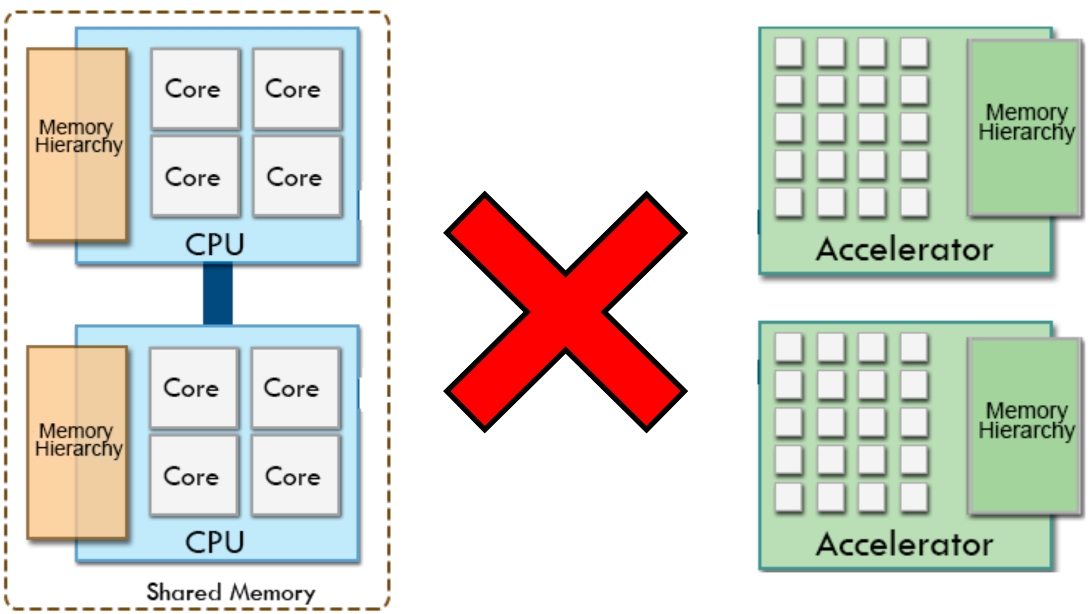
- CPUs have multiple complex computing cores
  - Use processes/threads to parallelise the code
- GPUs support a very high number of simultaneous threads
  - Offload data intensive computations to the device



Common Parallel Approach

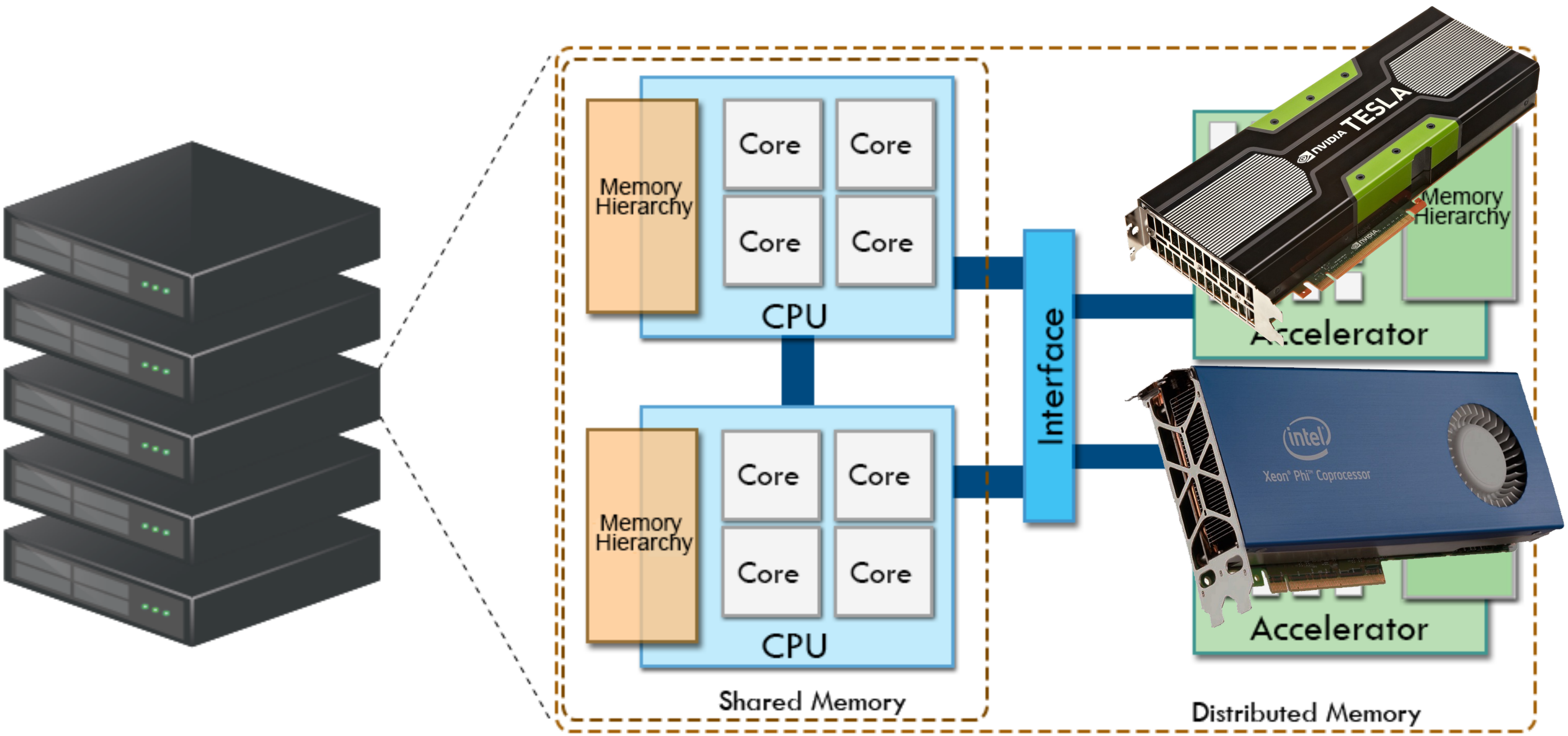
- Both devices coexist on the same system, but…
  - The processing power of the CPU is wasted when offloading code to the GPU, and vice-versa

- **Why not simultaneously use both devices?**

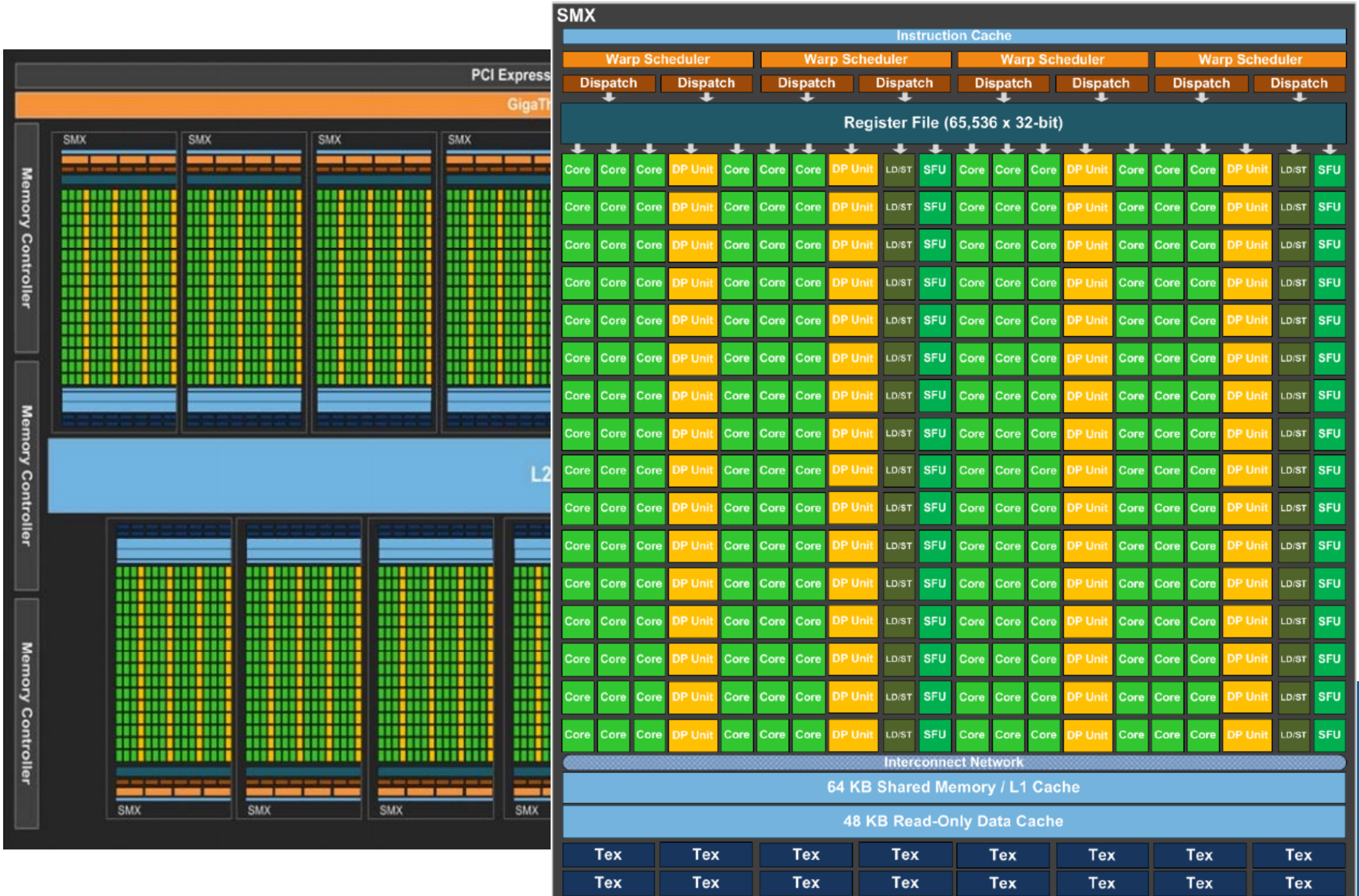


HetPlat Architecture

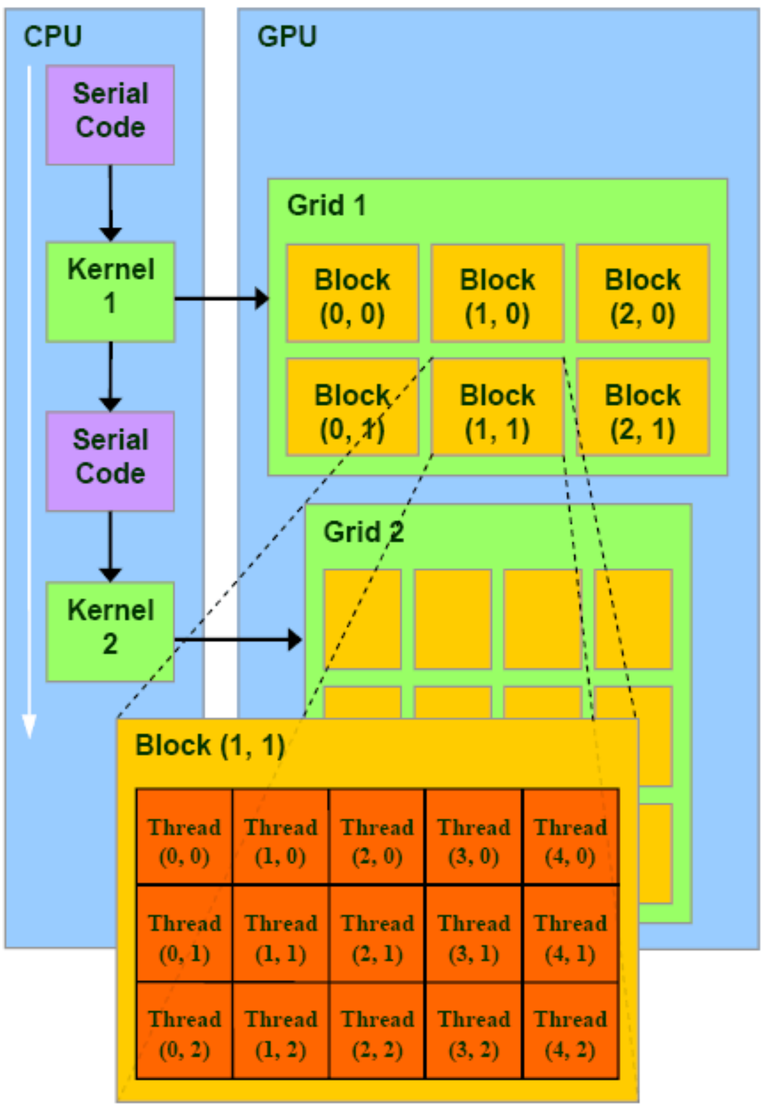
Motivation
Heterogeneous Platforms (HetPlats)
Levels of Parallelism
Performance Scalability
Performance Portability

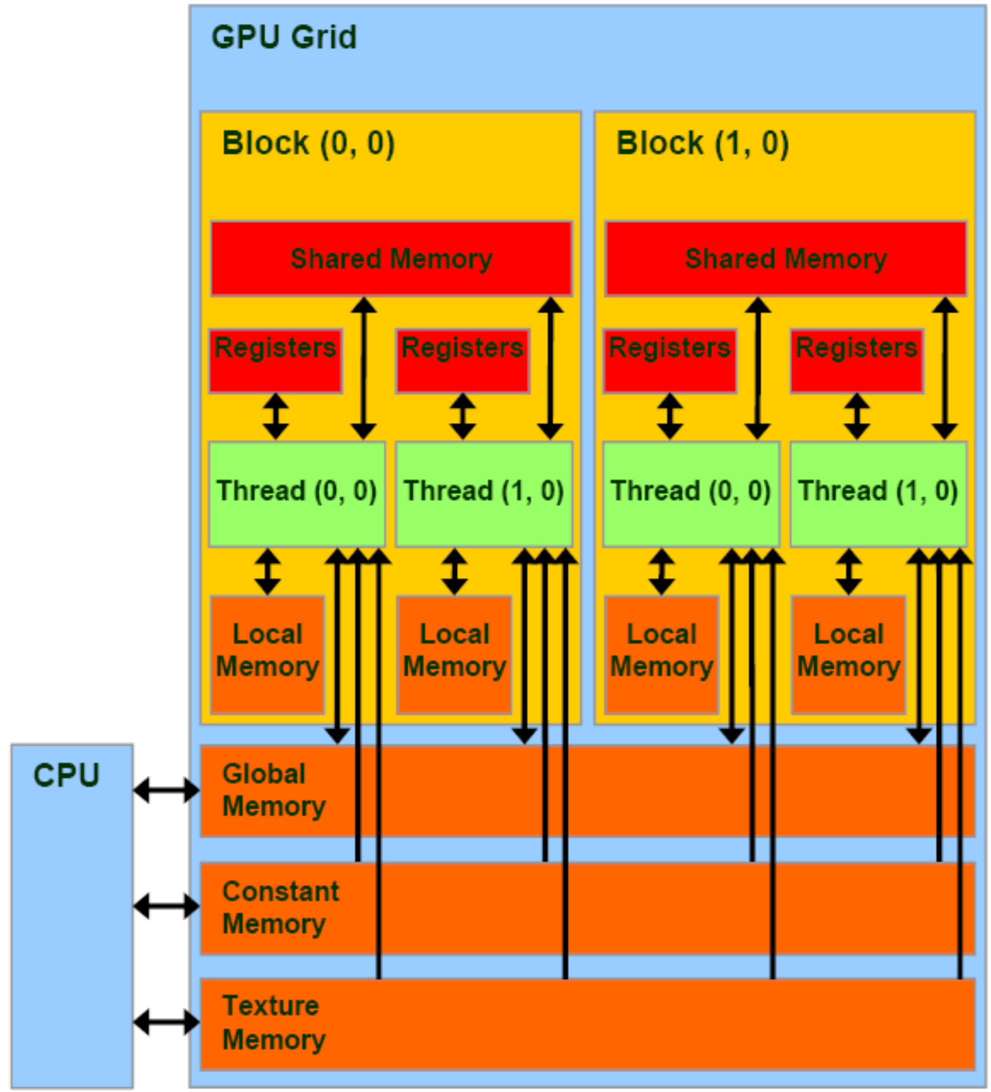


GPU Kepler Architecture

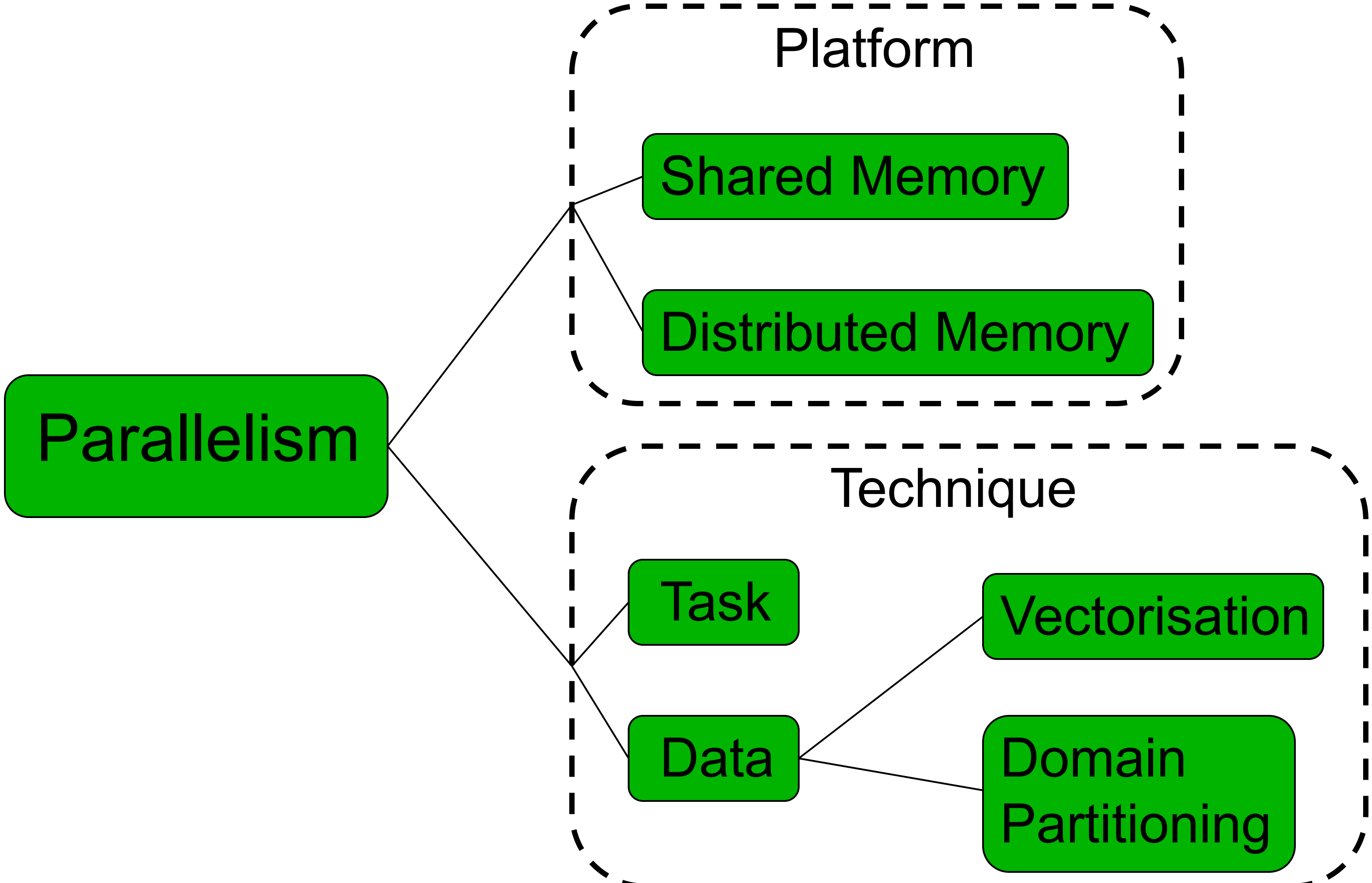


CUDA Programming Model

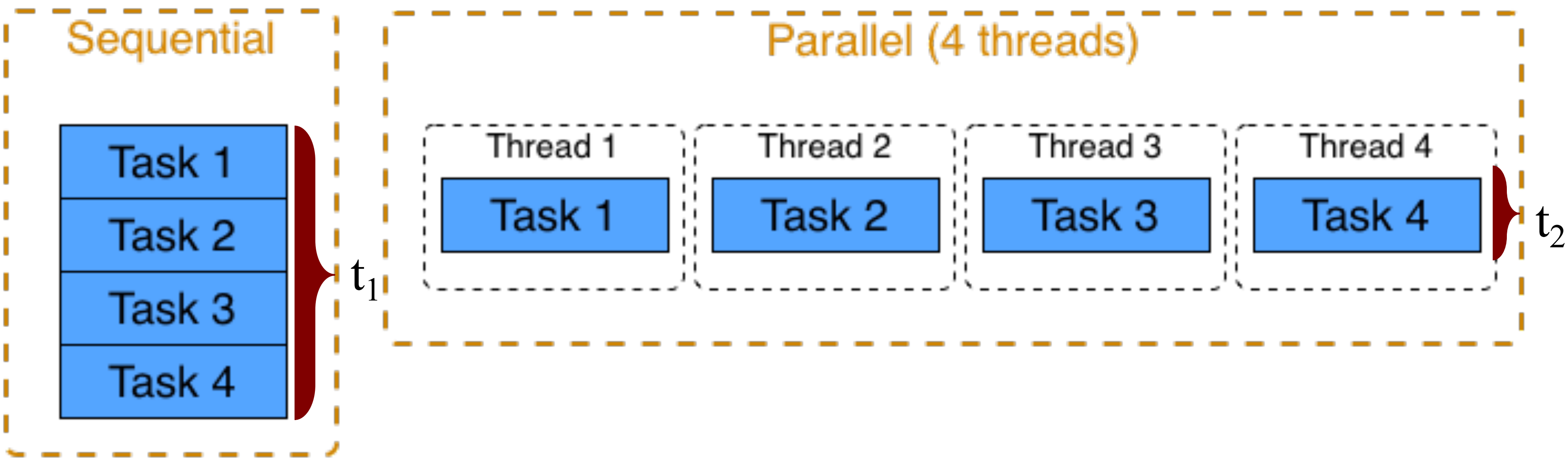




Levels of Parallelism



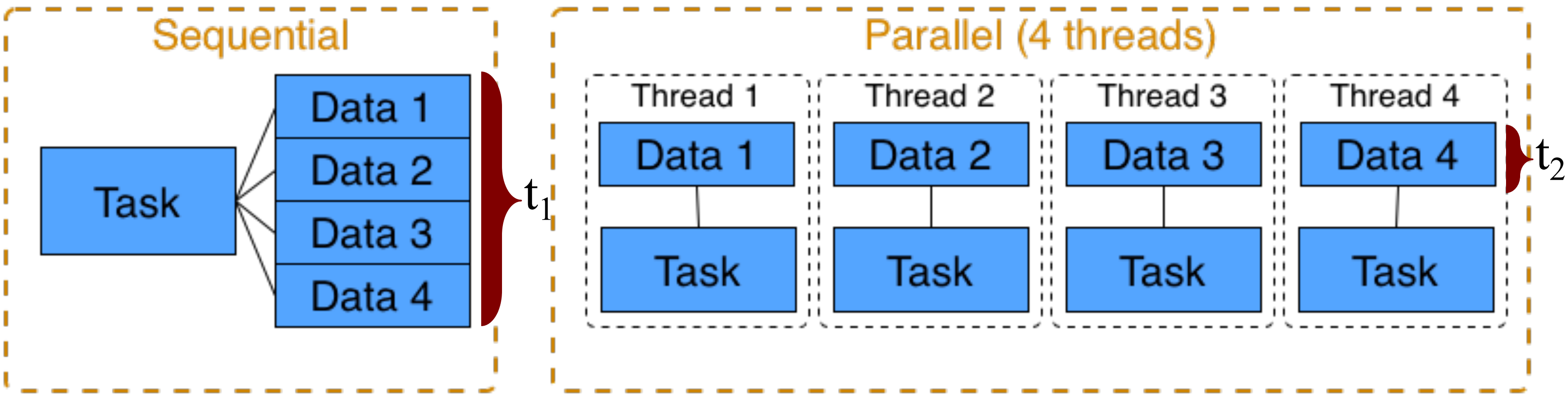
CPU Task Parallelism



$$t_2 \approx t_1 / 4$$

Levels of Parallelism

CPU Data Parallelism



$$t_2 \approx t_1 / 4$$

CERN School of Computing

A Small Example

Sequential

```
void stencil_1d(int *in, int *out) {
   int result = 0;

   for (int i = 0; i < SIZE; i++) {
      for (int offset = -RADIUS ; offset <= RADIUS ; offset++)
            result += in[i + offset];
      // Store the result
      out[i] = result;
   }
}
```

Parallel - CPU

```
void stencil_1d(int *in, int *out) {
   int result = 0;
   #pragma omp parallel for
   for (int i = 0; i < SIZE; i++) {
      for (int offset = -RADIUS ; offset <= RADIUS ; offset++)
            result += in[i + offset];
      // Store the result
      out[i] = result;
   }
}
```

A Small Example

Parallel - CPU

```
void stencil_1d(int *in, int *out) {
   int result = 0;
   #pragma omp parallel for
   for (int i = 0; i < SIZE; i++) {
      for (int offset = -RADIUS ; offset <= RADIUS ; offset++)
               result += in[i + offset];
      // Store the result
      out[i] = result;
   }
}
```

Parallel - GPU

```
void stencil_1d(int *in, int *out) {
   int result = 0;
   int i = threadIdx.x + blockIdx.x * blockDim.x;

   for (int offset = -RADIUS ; offset <= RADIUS ; offset++)
      result += in[i + offset];
   // Store the result
   out[i] = result;
}
```

However, it is highly inefficient…

A Small Example

GPU – Optimised

```
void stencil_1d(int *in, int *out) {
   int result = 0;
   int gindex = threadIdx.x + blockIdx.x * blockDim.x;
   int lindex = threadIdx.x + RADIUS;

   // Read input elements into shared memory
   temp[lindex] = in[gindex];
   if (threadIdx.x < RADIUS) {
      temp[lindex - RADIUS] = in[gindex - RADIUS];
      temp[lindex + BLOCK_SIZE] = in[gindex + BLOCK_SIZE];
   }
   // Synchronize (ensure all data is available)
   __syncthreads();

   for (int offset = -RADIUS ; offset <= RADIUS ; offset++)
      result += temp[i + offset];
   // Store the result
   out[i] = result;
}
```

With simple optimisations the code complexity starts to increase considerably….

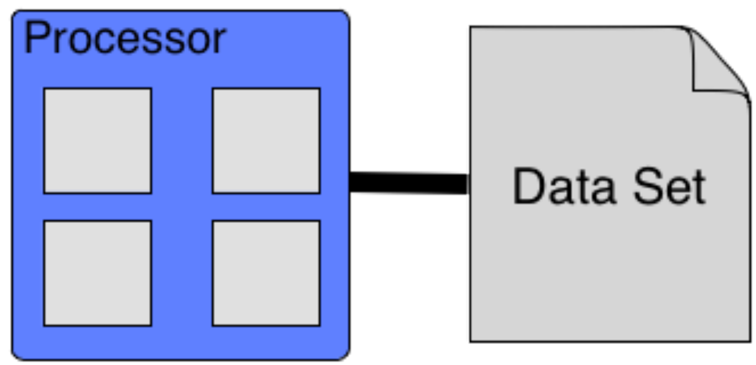
Challenges in Heterogeneous Computing

- **Different architectures**
  - Distinct designs of parallelism
  - Distinct memory hierarchies
- **Different programming paradigms**
  - Distinct code for efficient algorithms among devices
- **Workload management**
  - High latency communication between CPU and device
  - Different throughputs among devices

Performance Scalability

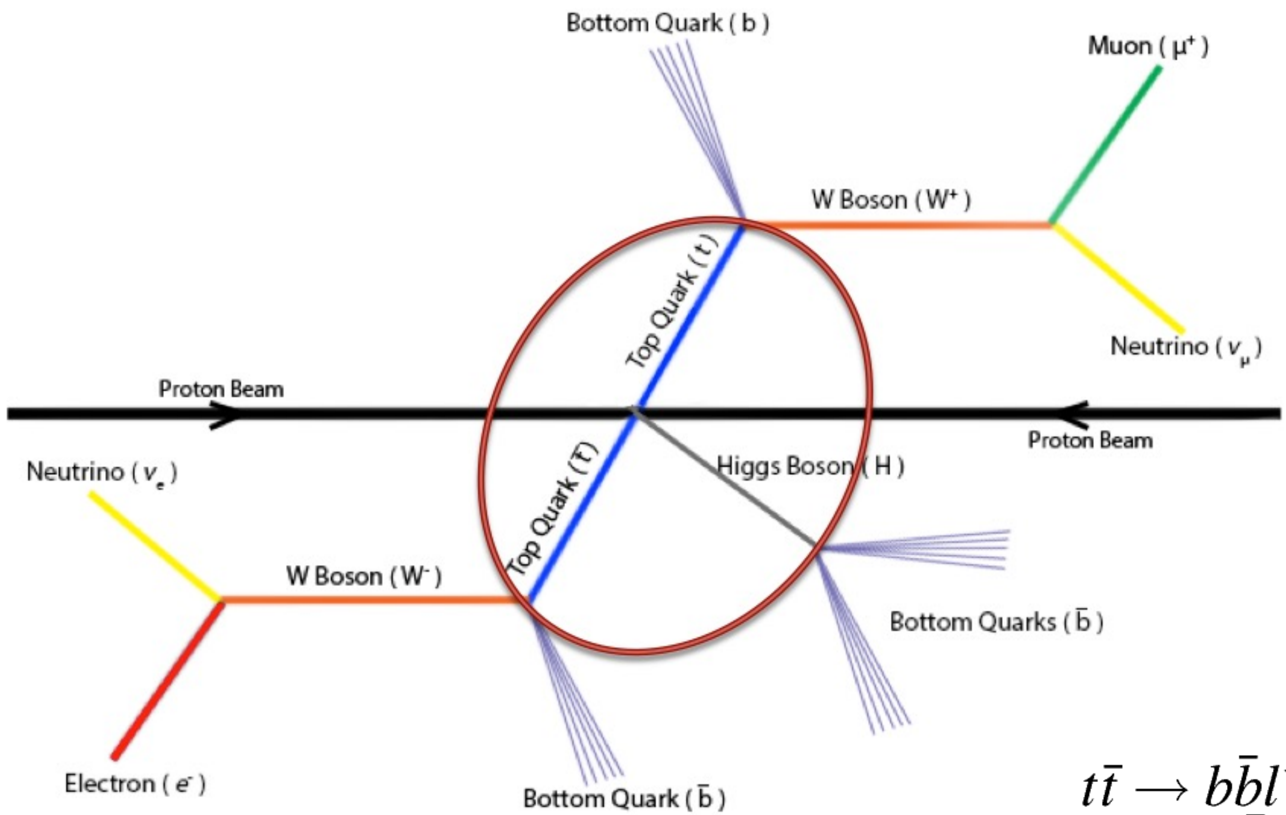
- Consider an efficient algorithm, with optimised parallel code

- Does the performance scale?
  - With the data set size
  - **With the increased number of cores**



Performance Scalability

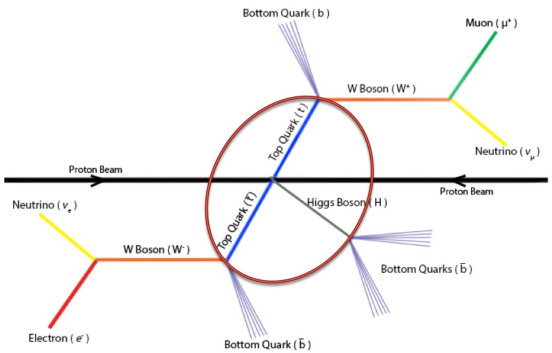
A Real Case Study

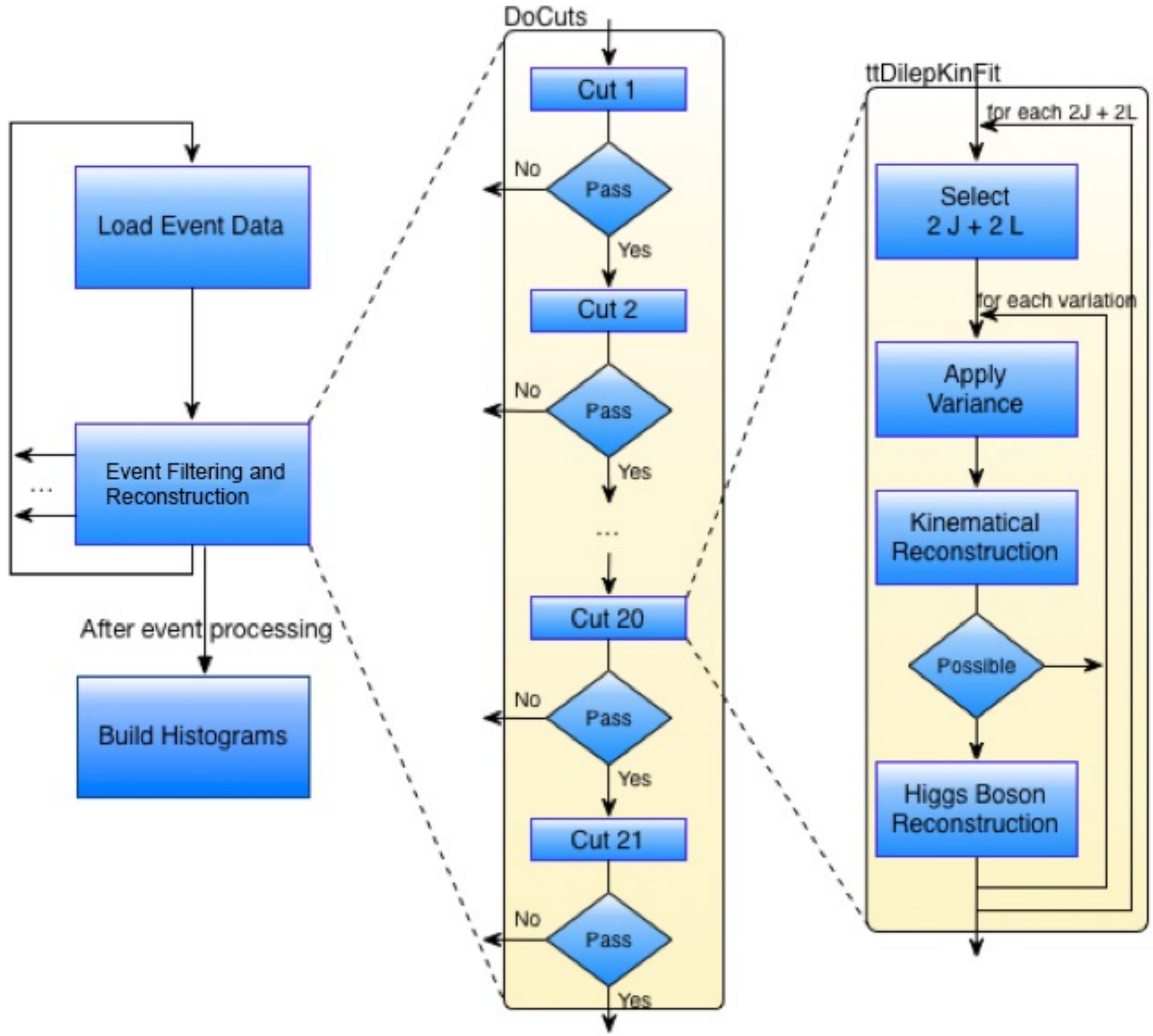


$$t\bar{t} \rightarrow b\bar{b}l^{+}l^{-}\nu\bar{\nu}$$
$$H \rightarrow b\bar{b}$$

Performance Scalability

A Real Case Study

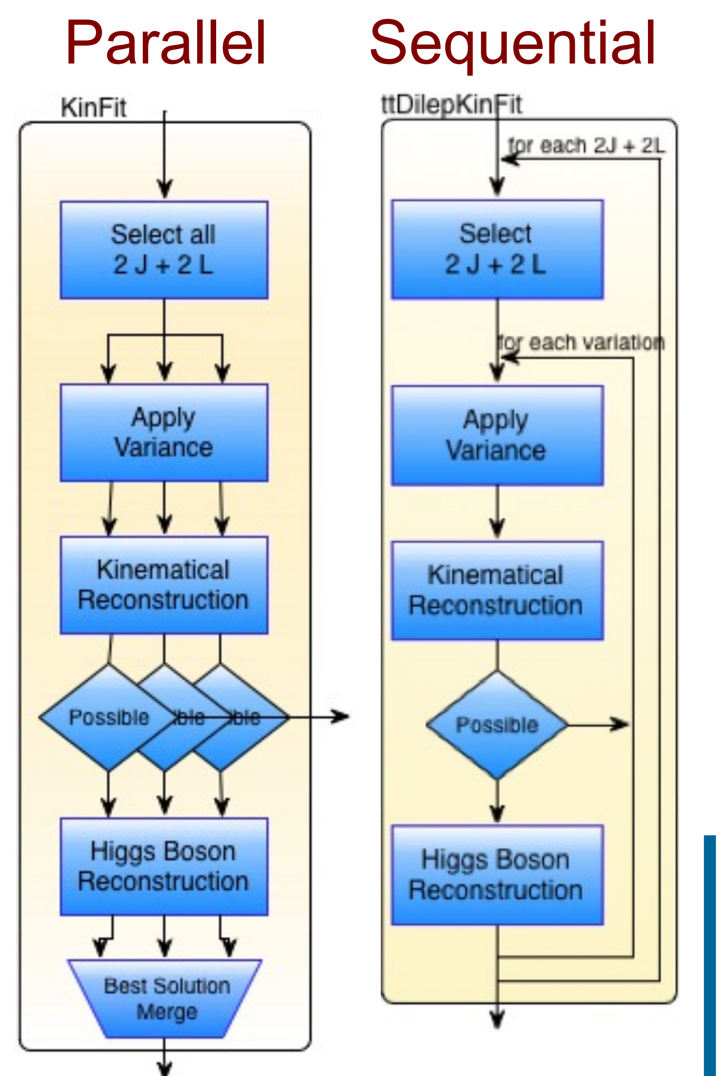




Performance Scalability

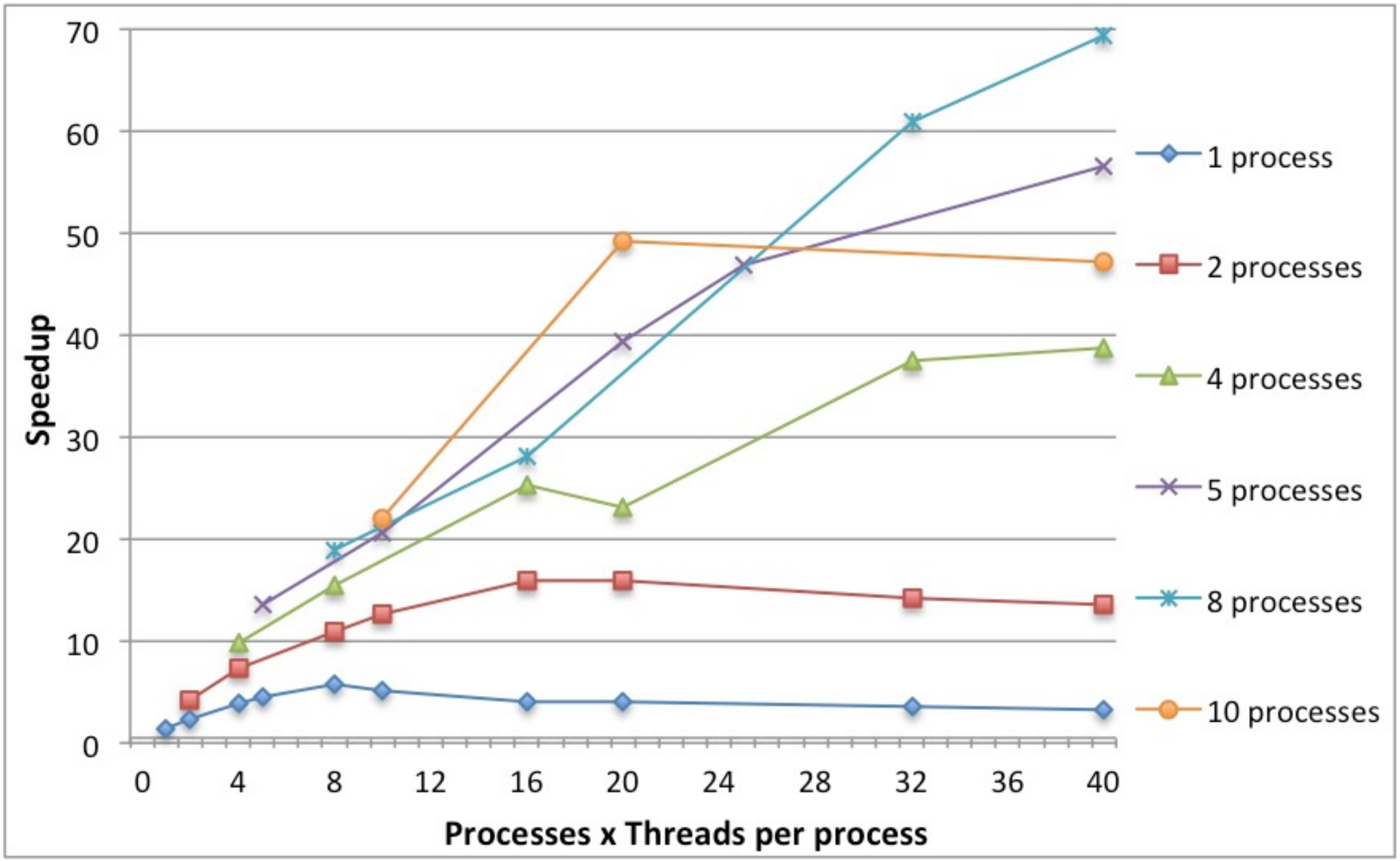
A Real Case Study

- Optimisation on a dual socket homogeneous system
  - KinFit takes 99.8% overall execution time
- Execution Parallelism
  - Exploring multi-core devices
- Runtime inefficiencies
  - Multithreading inefficiencies



Performance Scalability

A Real Case Study



2x 10-core Intel Xeon E5-2670v2, with Hyperthreading, 64GB RAM

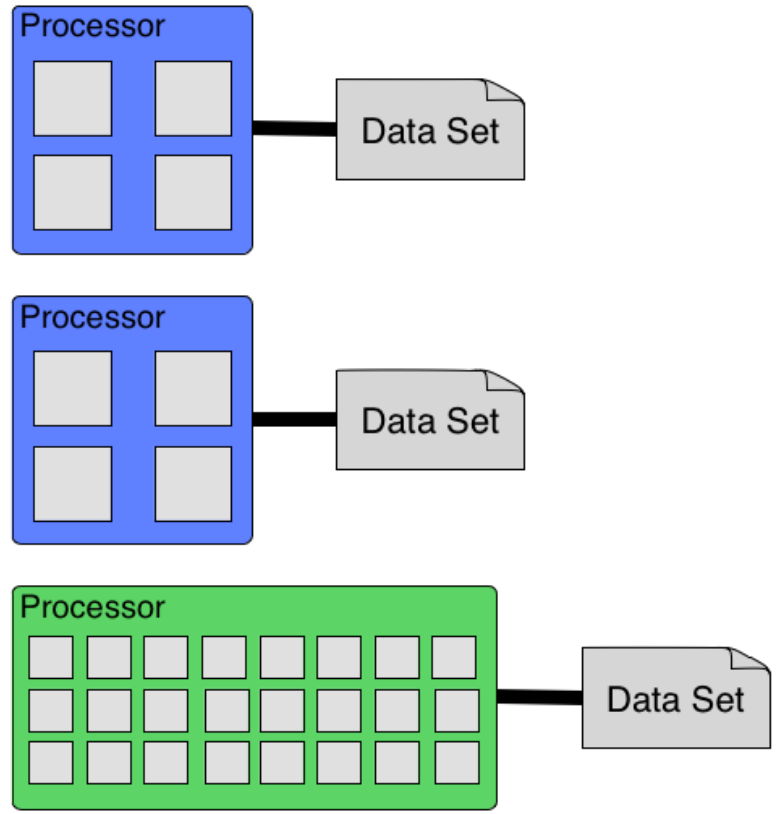
CERN School of Computing

However…

- The code either uses the CPU or the GPU
  - Each requires different code
- Either CPU or GPU processing… Only a factor of the system processing potential is being used
  - It would be great to get both codes to simultaneously work sharing the same data set

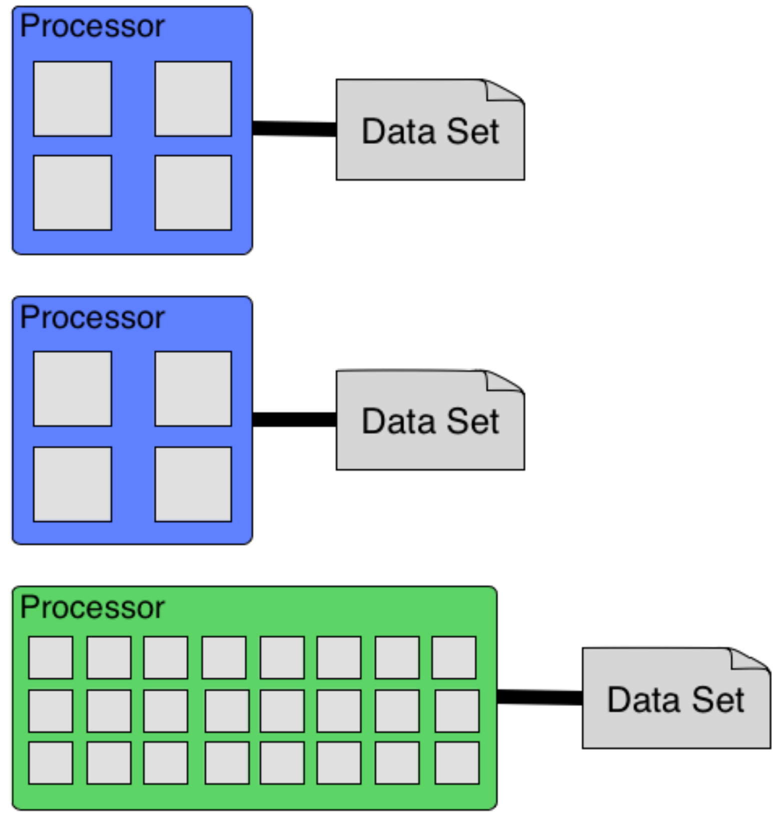
Performance Portability

- **Several challenges arise**
  - How is data partitioned?
  - How is data balanced among devices?
  - The code needs synchronisation?
  - Will the code scale?



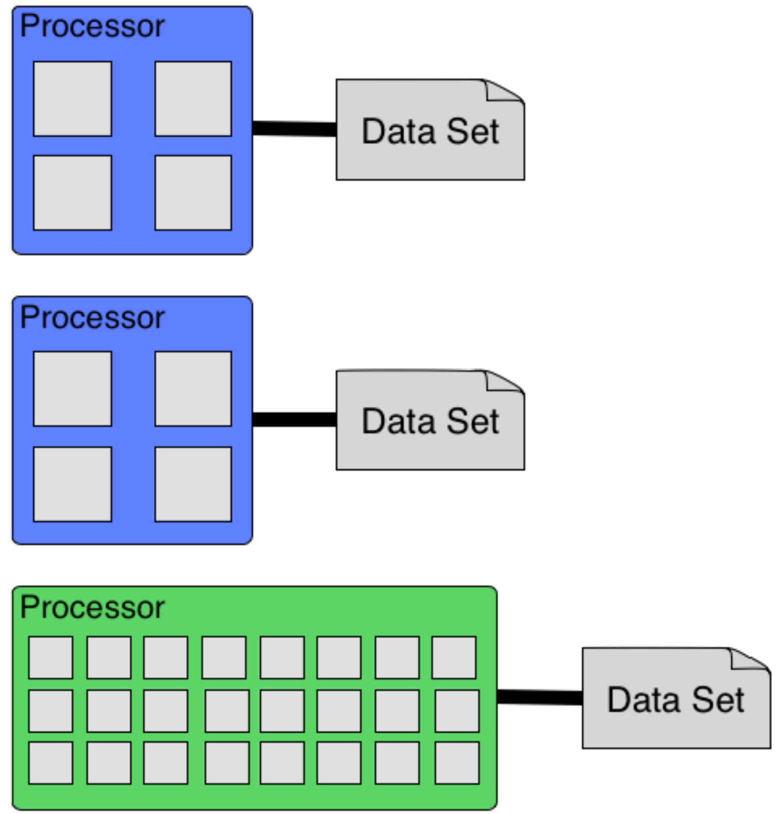
Performance Portability

- **Requires very complex coding**
  - Code the algorithm for the CPU
    - Efficiently manage the parallel workload
    - Ensure its performance scalability
  - Code the algorithm for the accelerator
    - Efficiently manage the parallel workload
    - Ensure its performance scalability



Performance Portability

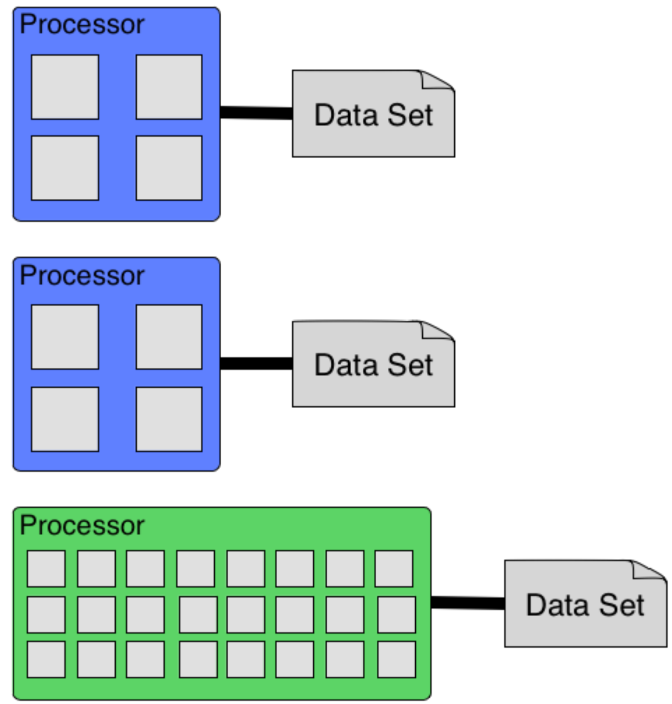
- **Requires very complex coding**
  - Manage the workload among different devices
    - Not trivial to code
    - Data transfers use a low bandwidth connection
    - Computing devices have different processing throughputs
    - What is the best data chunk size for each?
    - What is the best scheduling technique?



HetPlats Challenges

- **"I spent months optimising my code for HetPlats, I bet it will be super fast on this new system I just bought"**
  - No! You need to re-tune the code for each system…

- How is it possible to
  - achieve code scalability in each device?
  - simultaneously use both computing devices?
  - write the code once and guarantee its performance across different HetPlats?



CERN School of Computing

Conclusions

- Current computing platforms are heterogeneous
  - Multicore CPUs coupled with GPUs
  - Current parallel code uses either the CPU or the GPU
- Performance scalability is not linear
  - Having more cores does not always mean faster code
- Performance portability is very complex to achieve
  - Each different computing platform requires specific hand tuning
  - Workload balancing is very complex when using CPUs and GPUs simultaneously

CERN School of Computing

Development of High Performance Computing Applications Across Heterogeneous Systems

*Lecture **1***

Scalable Parallel Computing

André Pereira

LIP-Minho/University of Minho

Inverted CERN School of Computing, 23-24 February 2015