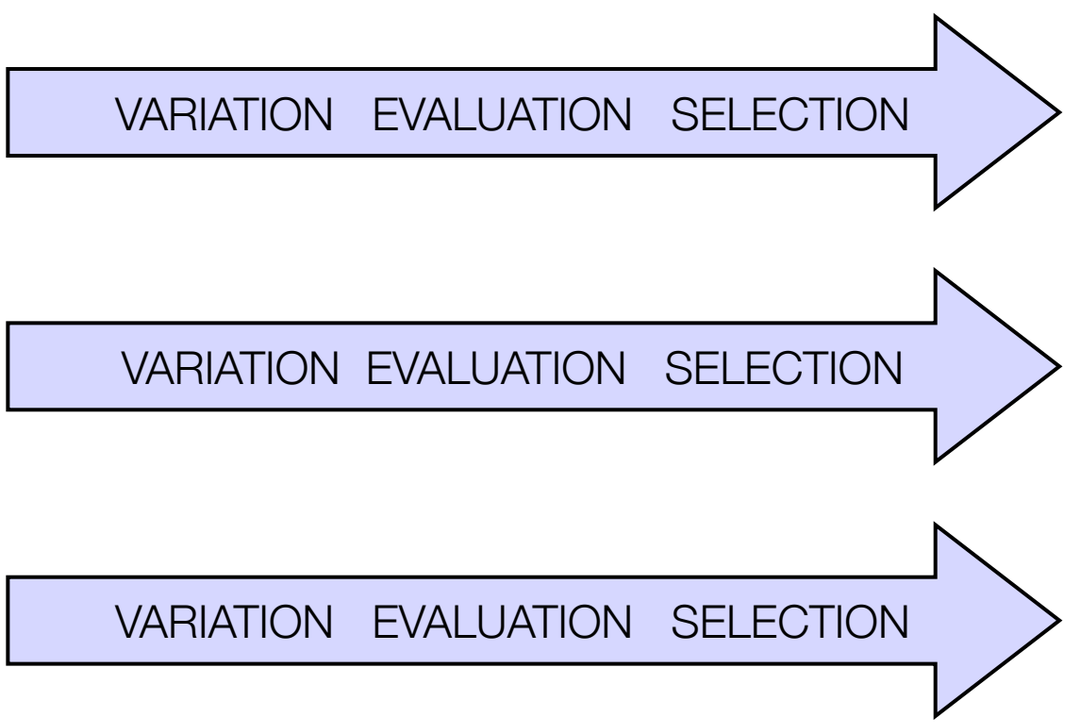
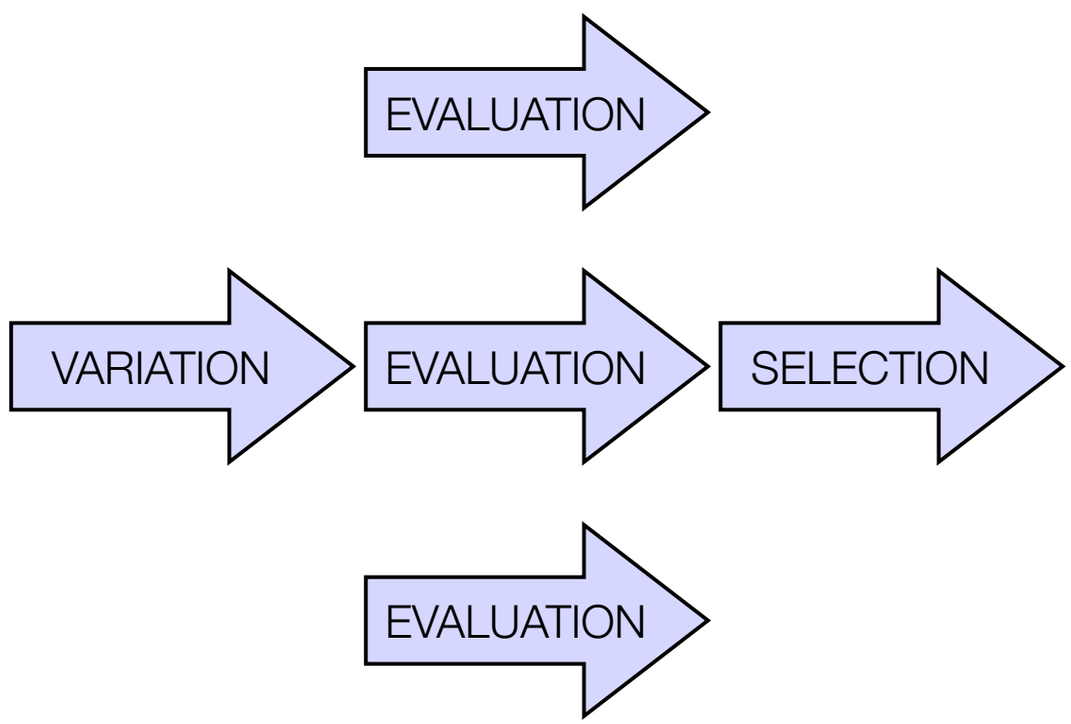
# Tutorial: High Performance SBSE Using Commodity Graphics Cards

Simon Poulding, University of York, UK
SSBSE, September 2012

# SBSE and High Performance Computing

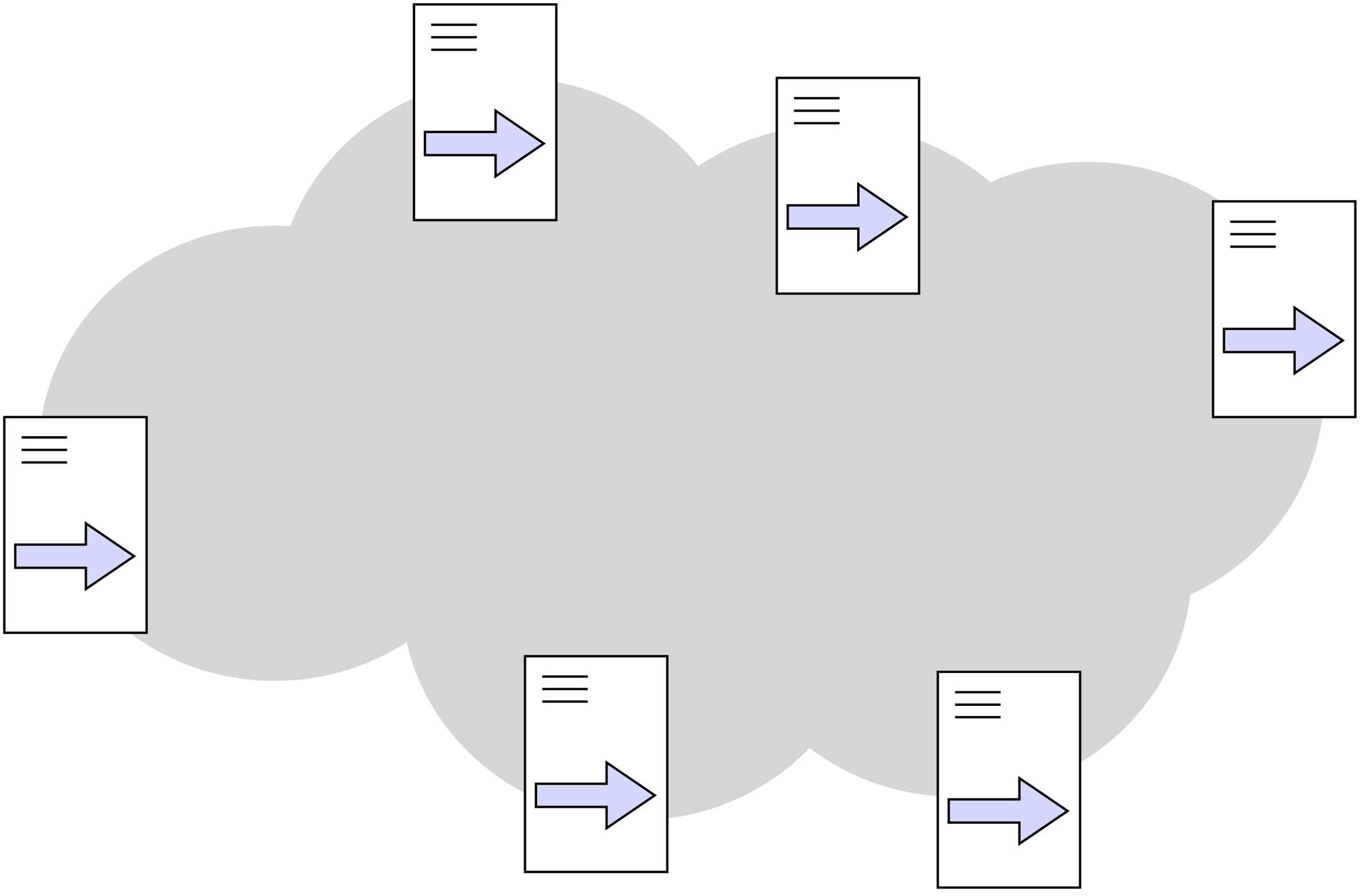



entire search
algorithm parallelisable

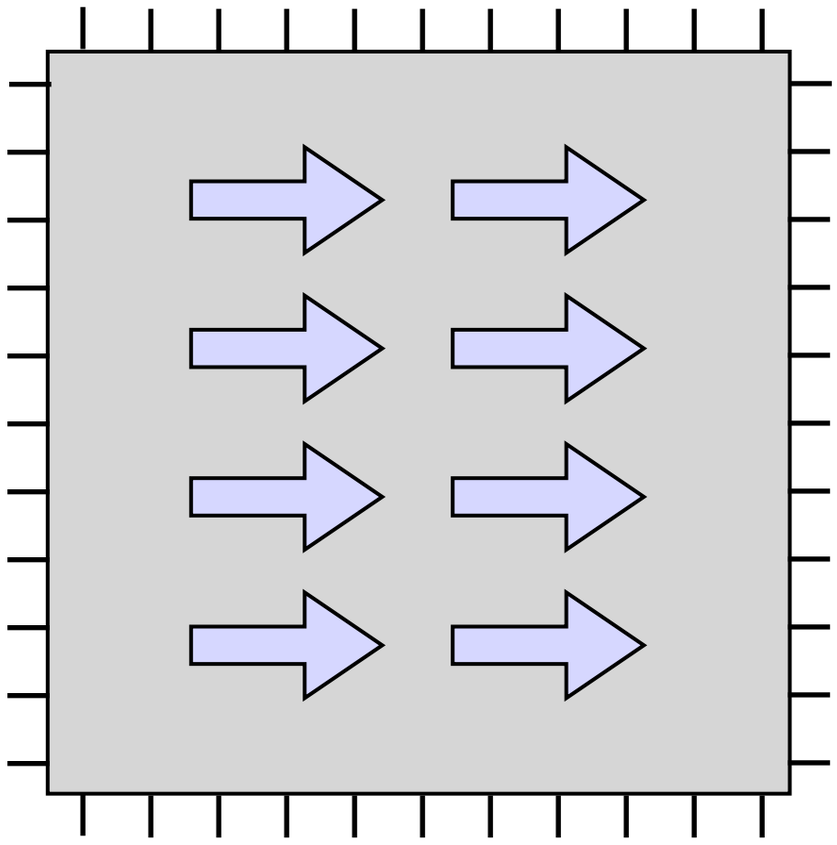



operations within
algorithm parallelisable

# Distributed Computing



# Multicore Computing



General Purpose Computing on GPUs (GPGPU)

CUDA Architecture

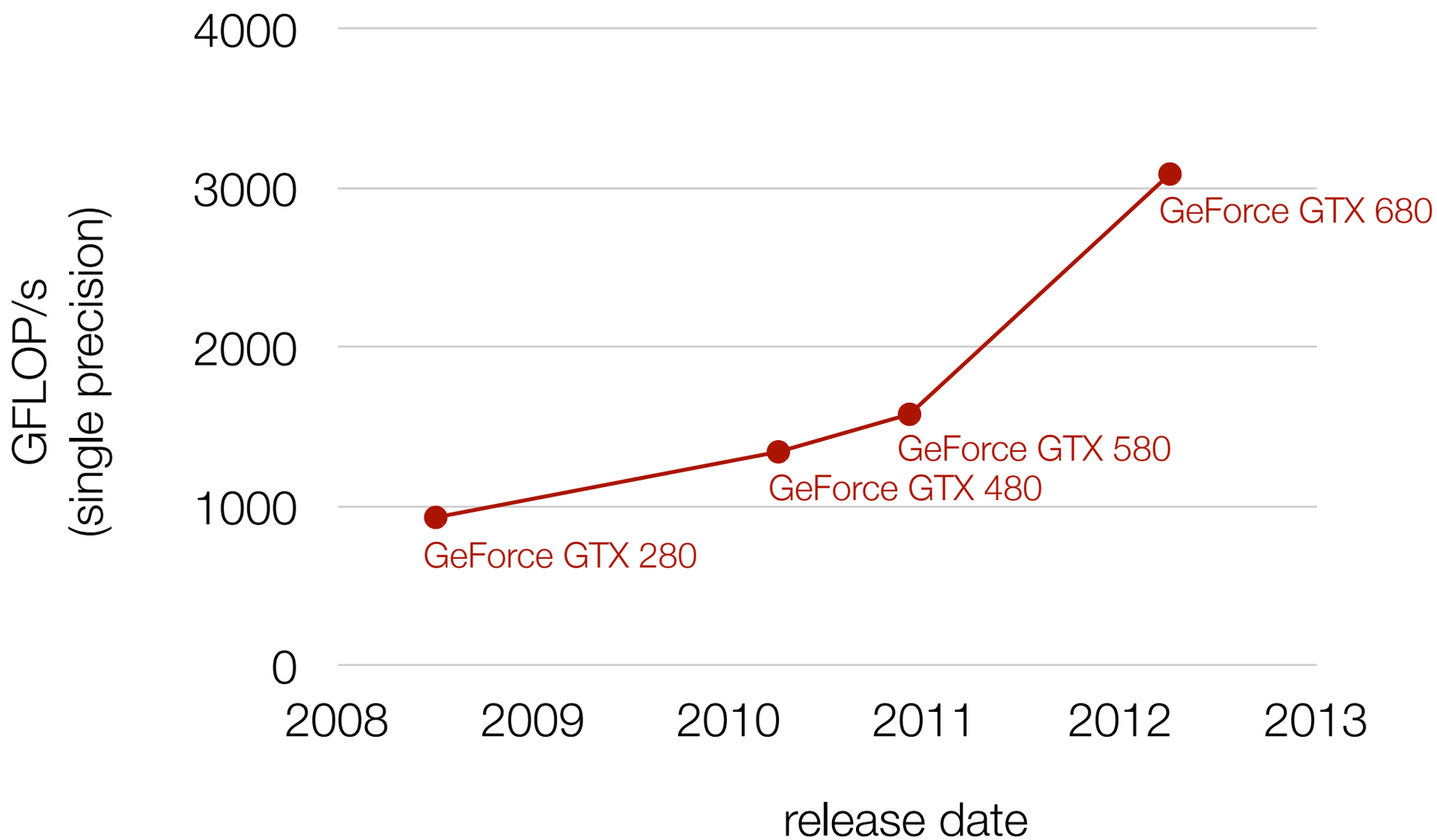
Developing CUDA Applications

Case Studies

- Parallelising Search Algorithm
- Parallelising Fitness Evaluation
- Parallelising Software Execution

# GPU Cards

# Technical Innovation




Adapted from “CUDA C Programming Guide”, NVIDIA, July 2012

# General Purpose Computing for GPUs (GPGPU)


NVIDIA GPUs
(most since 2009)


NVIDIA GPUs

AMD GPUs

Intel HD GPUs

Intel Core CPUs

other vendors ...

General Purpose Computing on GPUs (GPGPU)

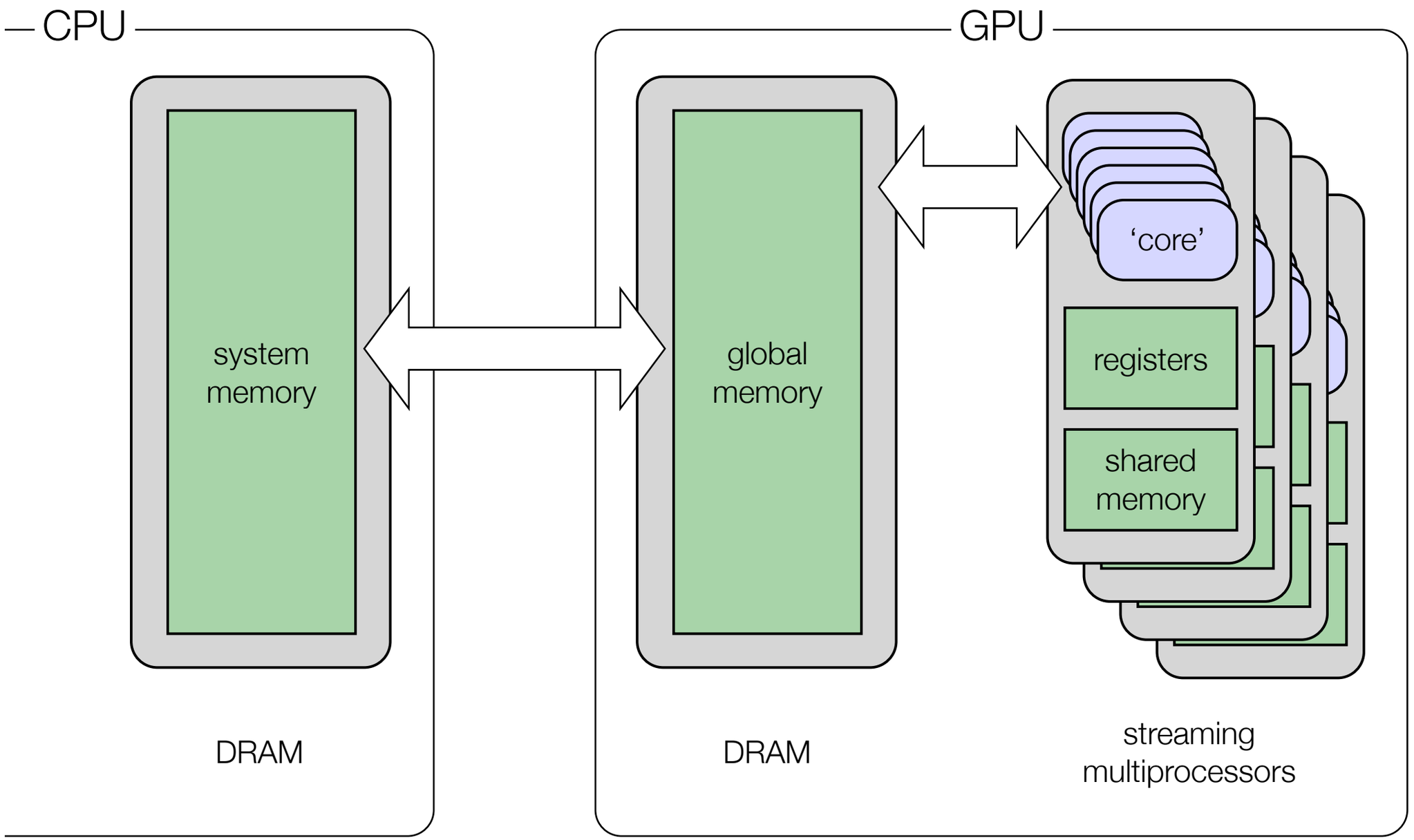
CUDA Architecture

Developing CUDA Applications

Case Studies

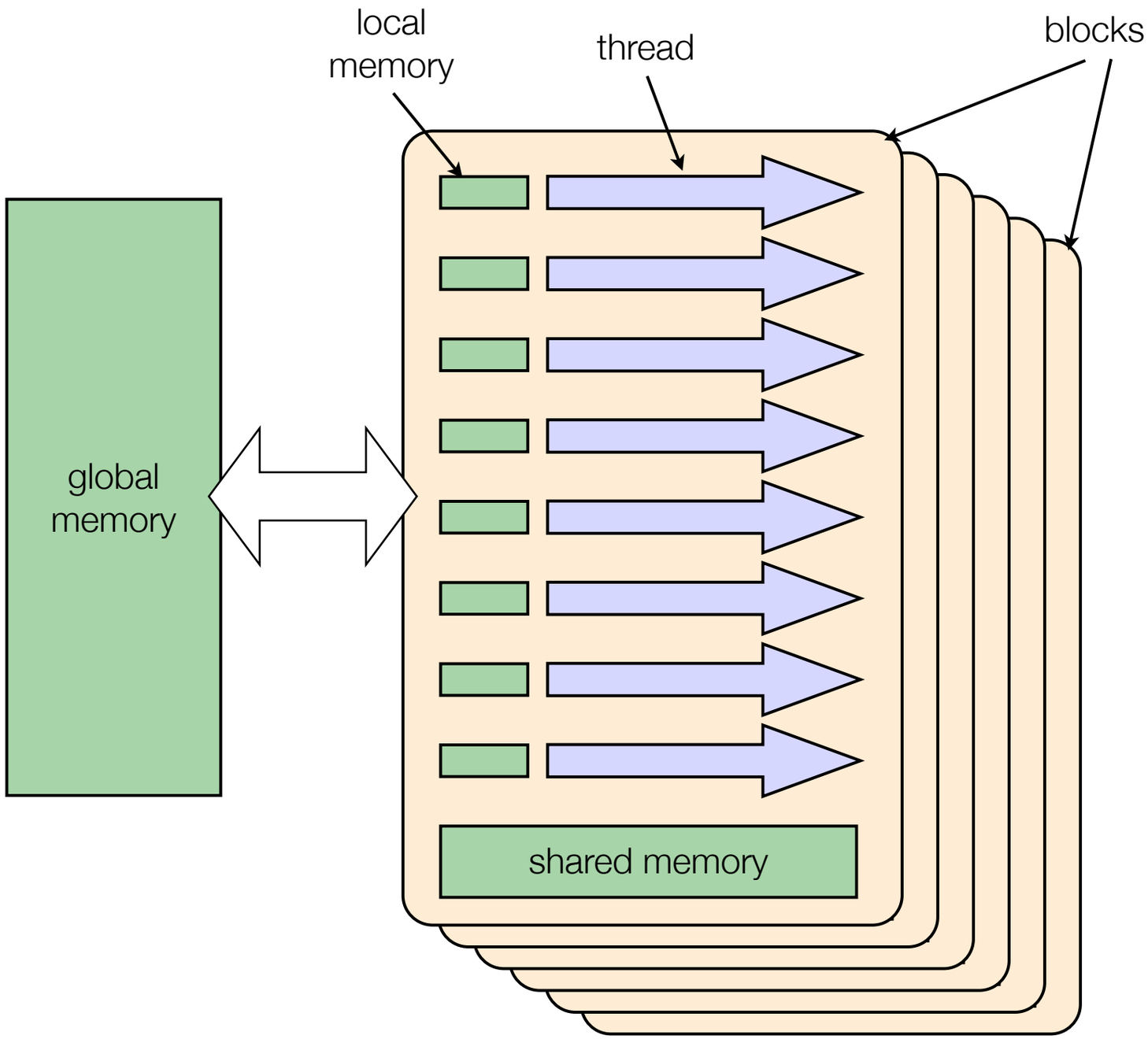
- Parallelising Search Algorithm
- Parallelising Fitness Evaluation
- Parallelising Software Execution

# Physical Architecture

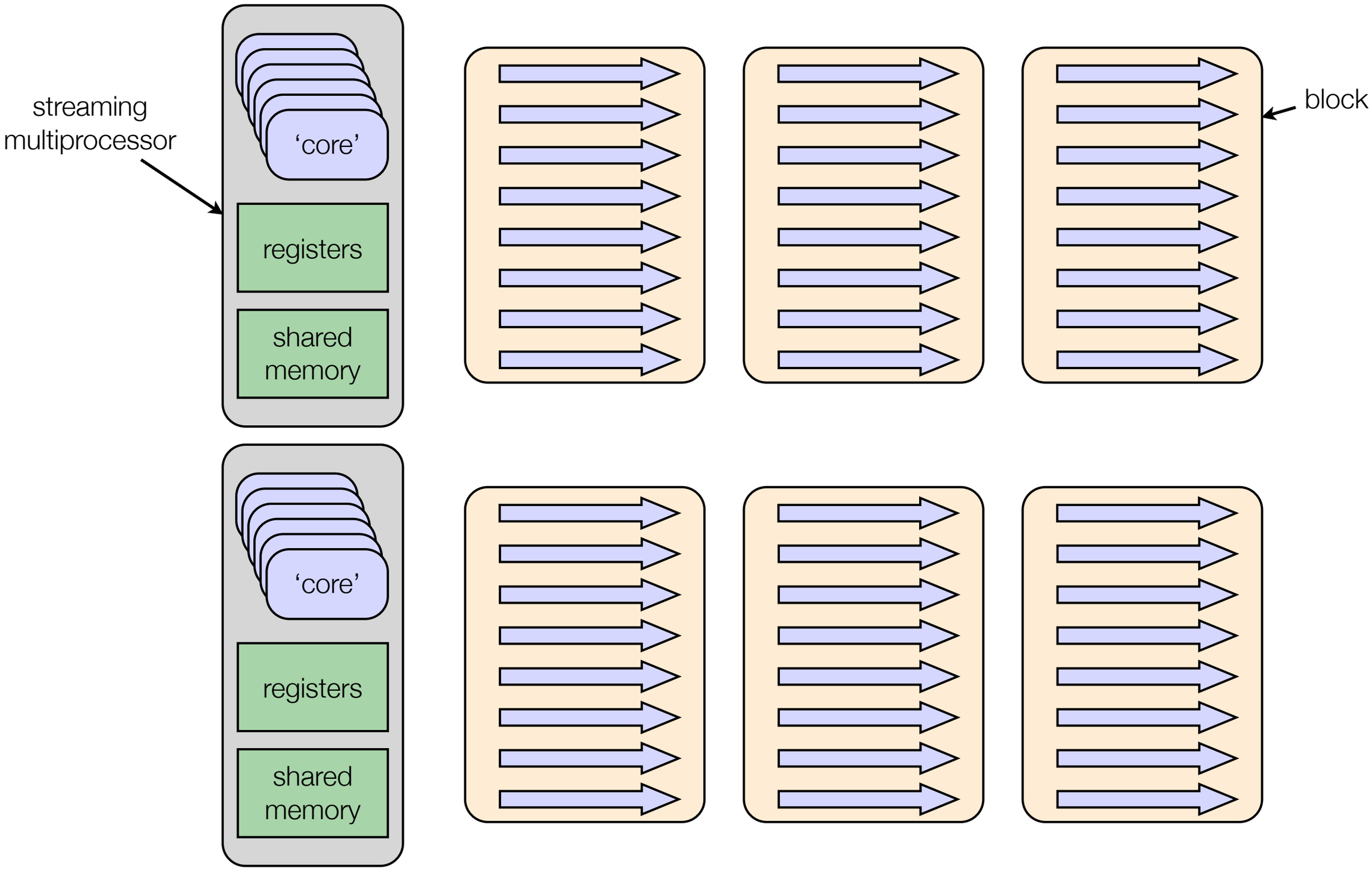



Adapted from "CUDA C Best Practices Guide", NVIDIA, May 2012

# Logical Architecture



# Mapping Logical to Physical



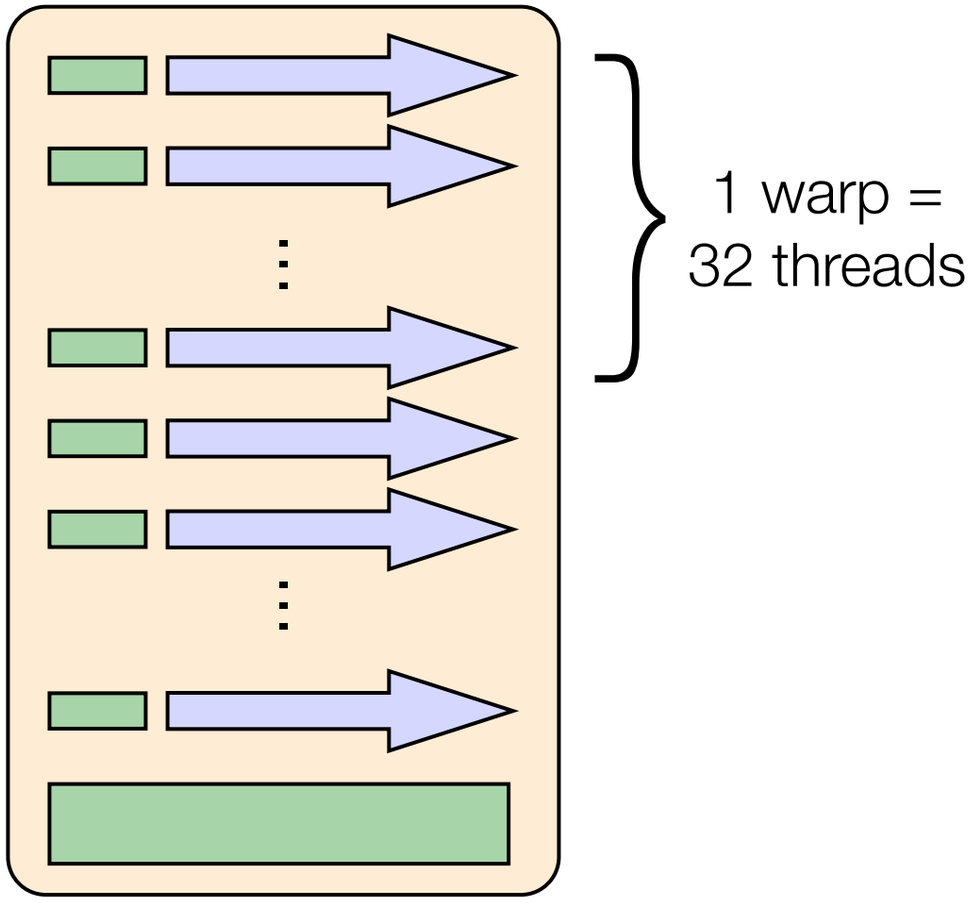
# CUDA Performance Features

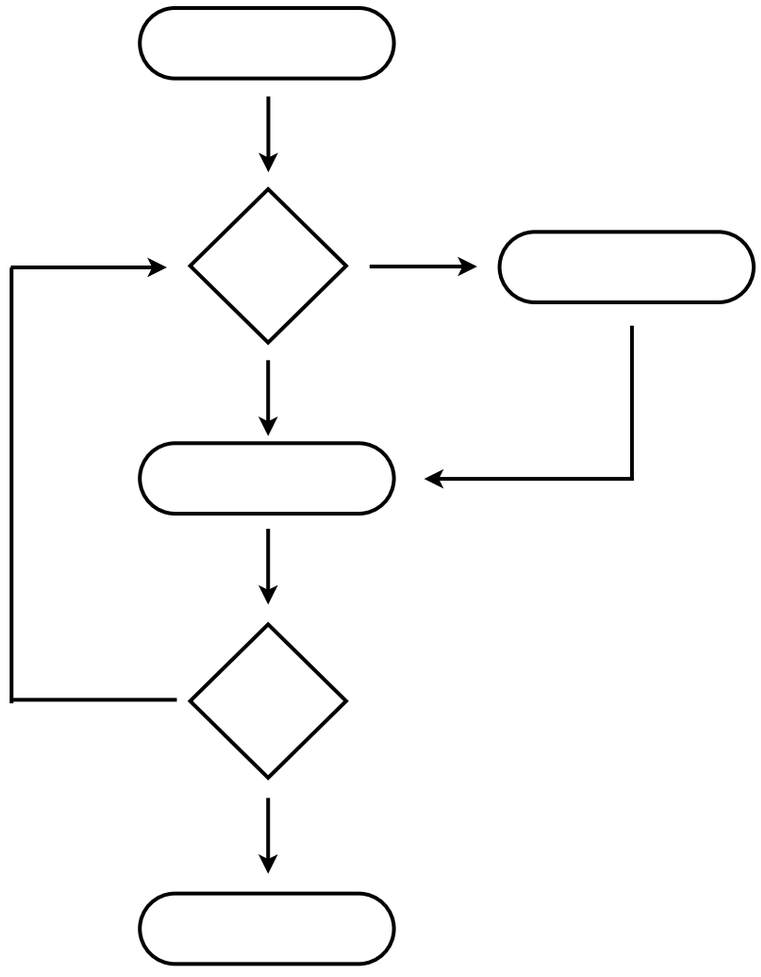
single-instruction multiple-thread

hardware multithreading

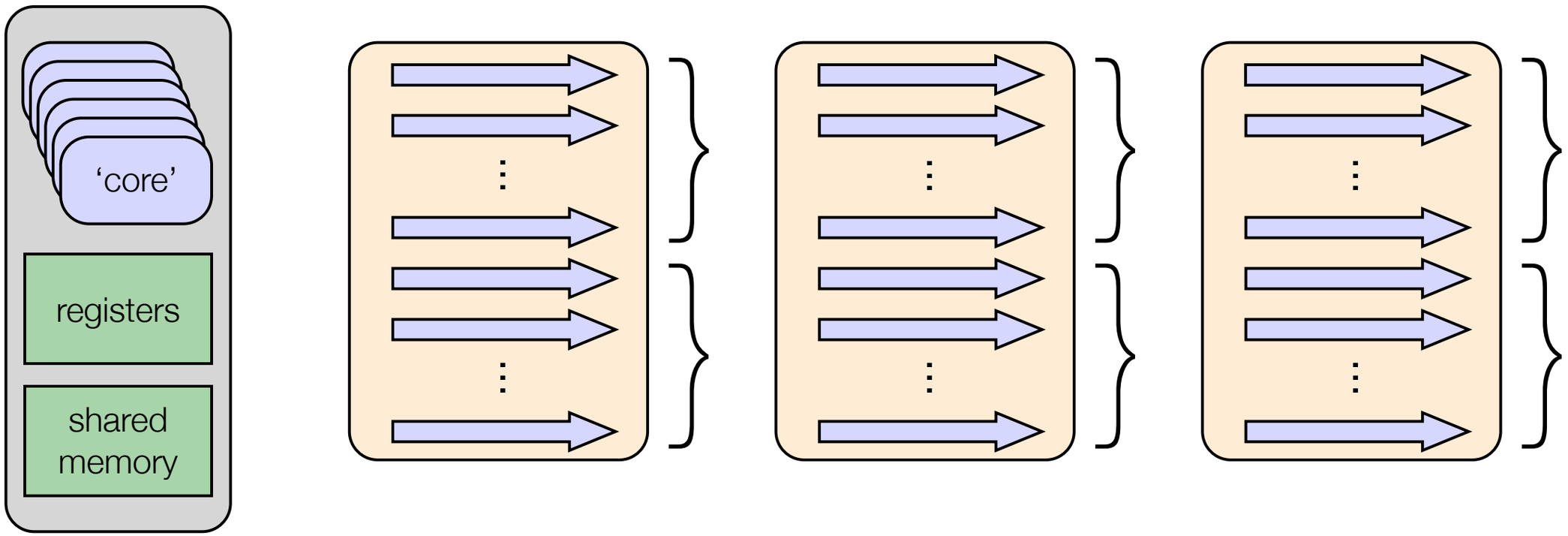
coalesced memory access

# Single-Instruction Multiple-Thread

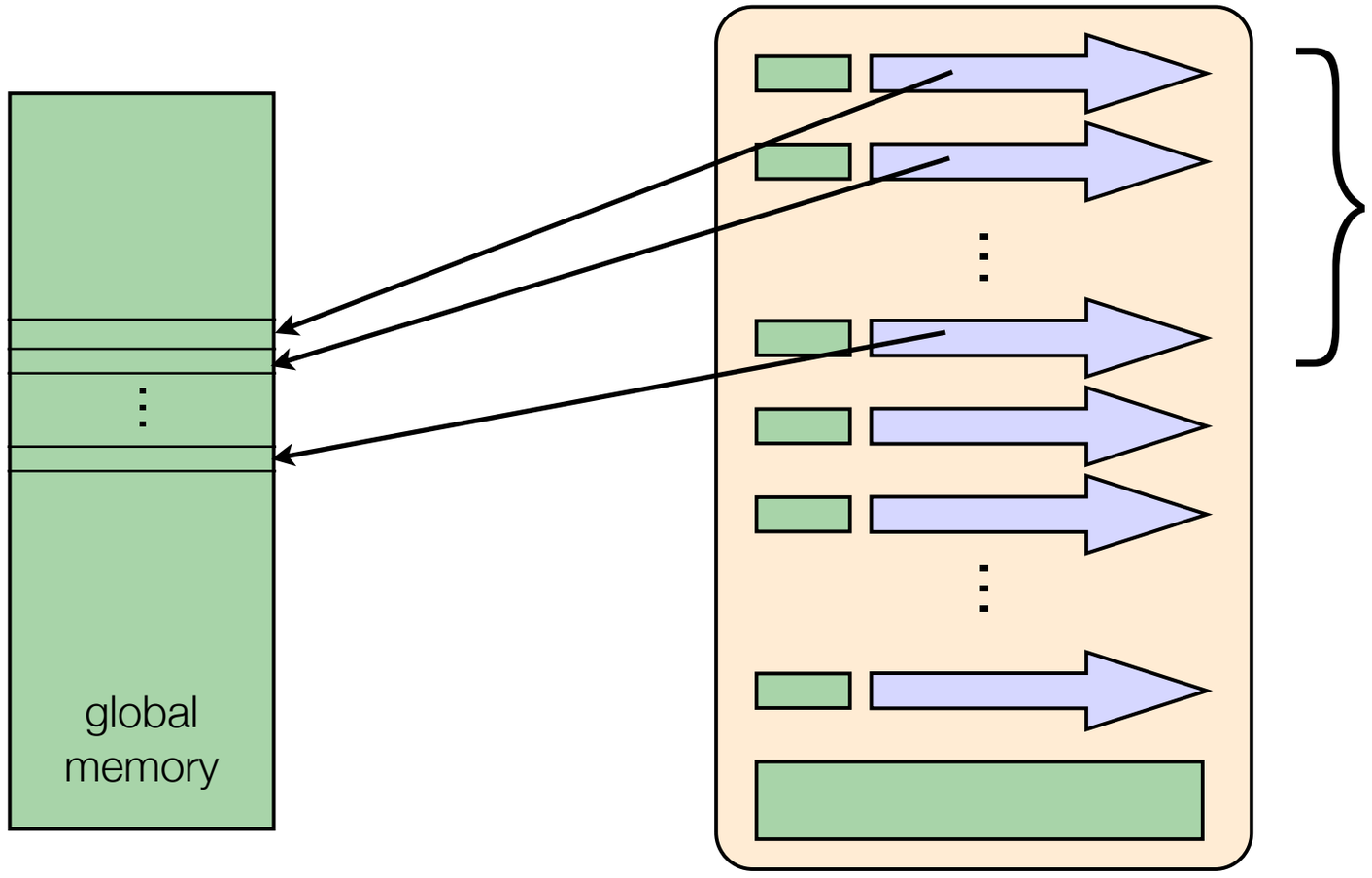


# Hardware Multithreading & Occupancy



# Coalesced Memory Access



General Purpose Computing on GPUs (GPGPU)

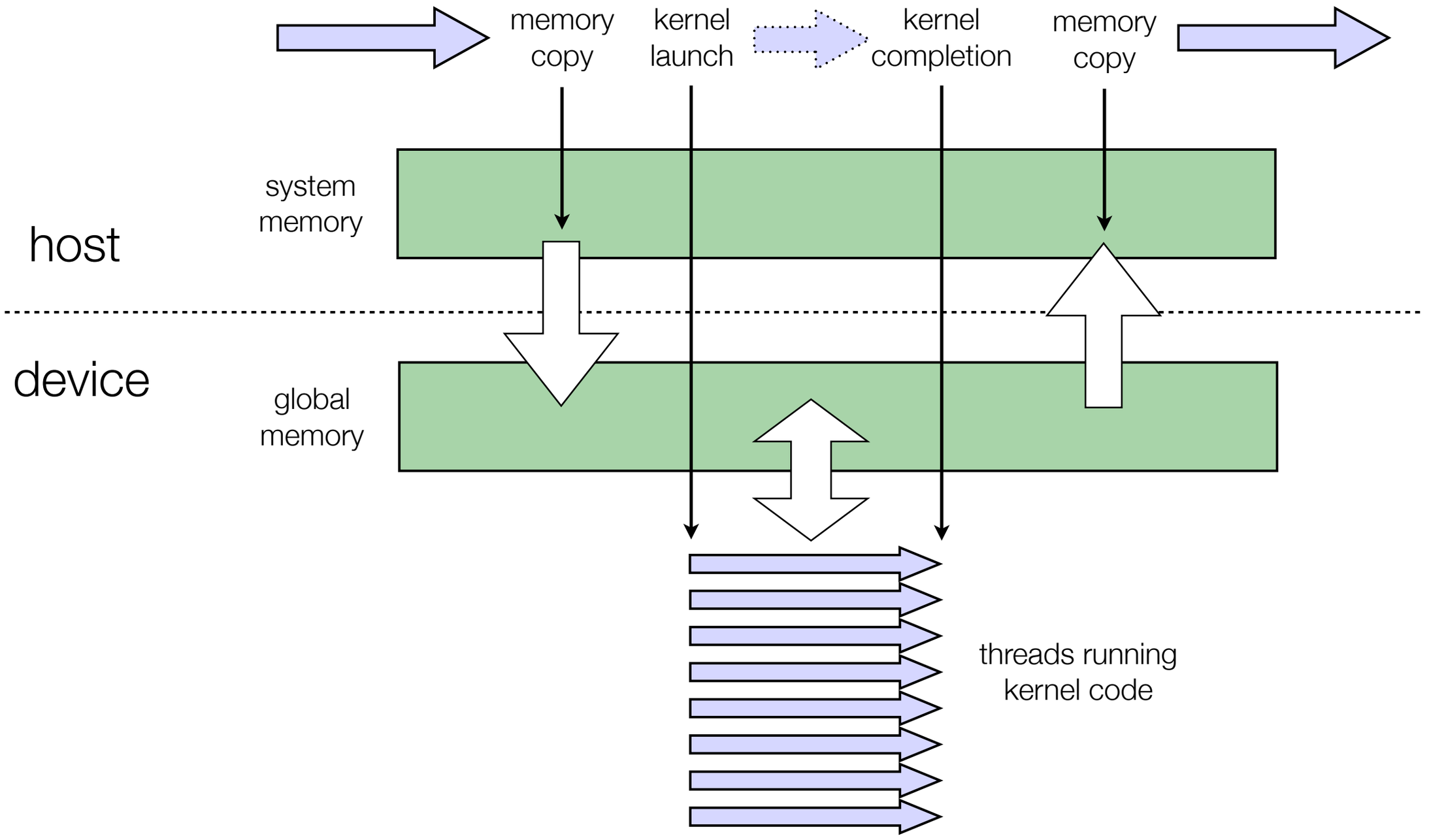
CUDA Architecture

Developing CUDA Applications

Case Studies

- Parallelising Search Algorithm
- Parallelising Fitness Evaluation
- Parallelising Software Execution

# Typical CUDA Application Pattern



# Example Problem

| a | b | c = a * b |
|---|---|---|
| 382 | 17 | ? |
| 1124 | 17 | ? |
| ⋮ | | |
| 30 | 17 | ? |
| 2781 | 98 | ? |
| 824 | 98 | ? |
| ⋮ | | |
| 4510 | 98 | ? |
| 4088 | 31 | ? |
| ⋮ | | |

256 x 64

# Kernel Code (device-side)

```
__global__ void exampleKernel(int * a, int * b, int * c) {

  __shared__ int sb;

  const unsigned int thread = threadIdx.x;
  const unsigned int block = blockIdx.x;
  const unsigned int gThread = block * blockDim.x + thread;

  if (thread == 0) {
    sb = b[block];
  }

  __syncthreads();

  c[gThread] = a[gThread] * sb;

}
```

# Launching a Kernel (host-side)

```
const unsigned int numThreads = 256;
const unsigned int numBlocks = 64;

dim3  gridD(numBlocks, 1, 1);
dim3  blockD(numThreads, 1, 1);

exampleKernel<<<gridD,blockD>>>(a,b,c);
```

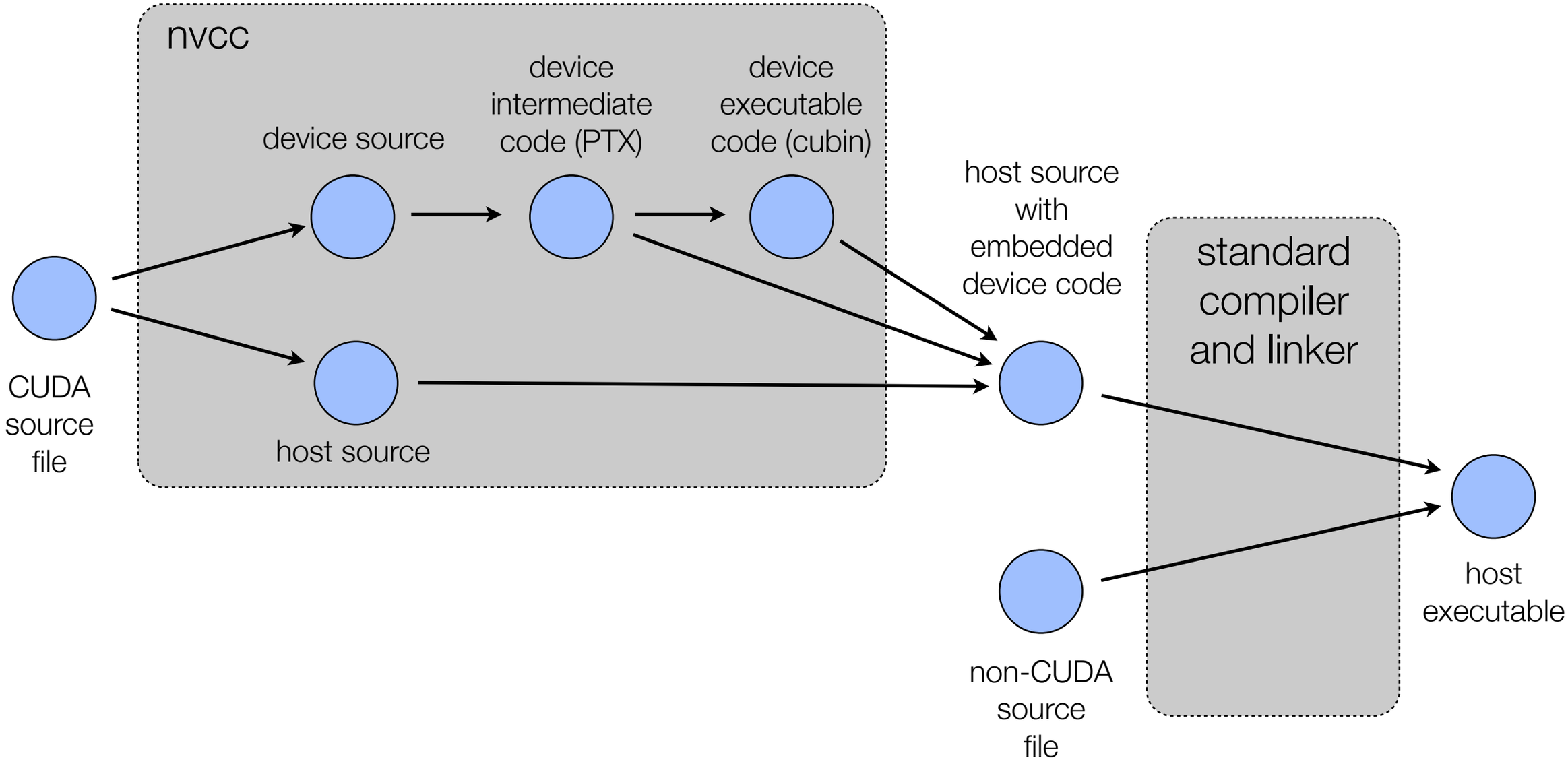
# Allocating and Copying Memory (host-side)

```
const unsigned int numThreads = 256;
const unsigned int numBlocks = 64;

int * a,b,c;

cudaMalloc((void **)&a, numThreads * numBlocks * sizeof(int));
cudaMalloc((void **)&b, numBlocks * sizeof(int));
cudaMalloc((void **)&c, numThreads * numBlocks * sizeof(int));

cudaMemcpy(a, inputA, numThreads * numBlocks * sizeof(int),
                                    cudaMemcpyHostToDevice);
cudaMemcpy(b, inputB, numBlocks * sizeof(int),
                                    cudaMemcpyHostToDevice);
```

# Putting It All Together

```
__global__ void exampleKernel(int * a, int * b, int * c) {
...
}

int main(...) {
...
cudaMalloc(...);
cudaMemcpy(...);
...
exampleKernel<<<gridD,blockD>>>(a,b,c);
...
cudaMemcpy(...);
...
}
```

# Build Process




Adapted from “CUDA Compiler Driver NVCC”, NVIDIA, May 2012

# Compute Capability

| compute capability | 1.0 | 1.1 | 1.2 | 1.3 | 2.x | 3.0 | 3.5 |
|---|---|---|---|---|---|---|---|
| atomic functions (global memory) | No | Yes | | | | | |
| atomic functions (shared memory) | No | | Yes | | | | |
| warp vote functions | No | | Yes | | | | |
| double precision floating point | No | | | Yes | | | |
| additional fence and sync functions | No | | | | Yes | | |
| max number threads per block | 512 | | | | 1024 | | |
| number register per multiprocessor | 8K | | 16K | | 32K | | 64K |
| max shared memory per multiprocessor | 16KB | | | | 48KB | | |
| local memory per thread | 16KB | | | | 512KB | | |
| max number instructions per kernel | 2 million | | | | 512 million | | |

Adapted from “CUDA C Programming Guide”, NVIDIA, July 2012

# Additional Tools and Libraries

## Development Tools

debugger

memory checker

profiler

## CUDA Libraries

linear algebra (CUBLAS)

sparse matrices (CUSPARSE)

random number generation (CURAND)

fast Fourier transform (CUFFT)

Thrust

General Purpose Computing on GPUs (GPGPU)

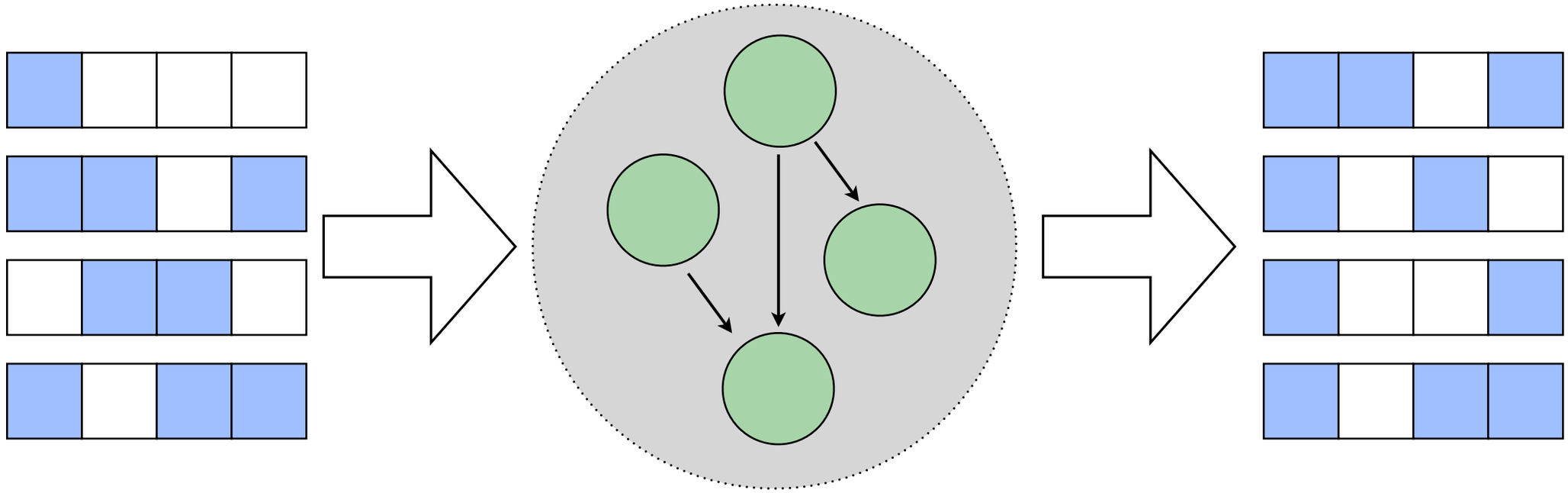
CUDA Architecture

Developing CUDA Applications

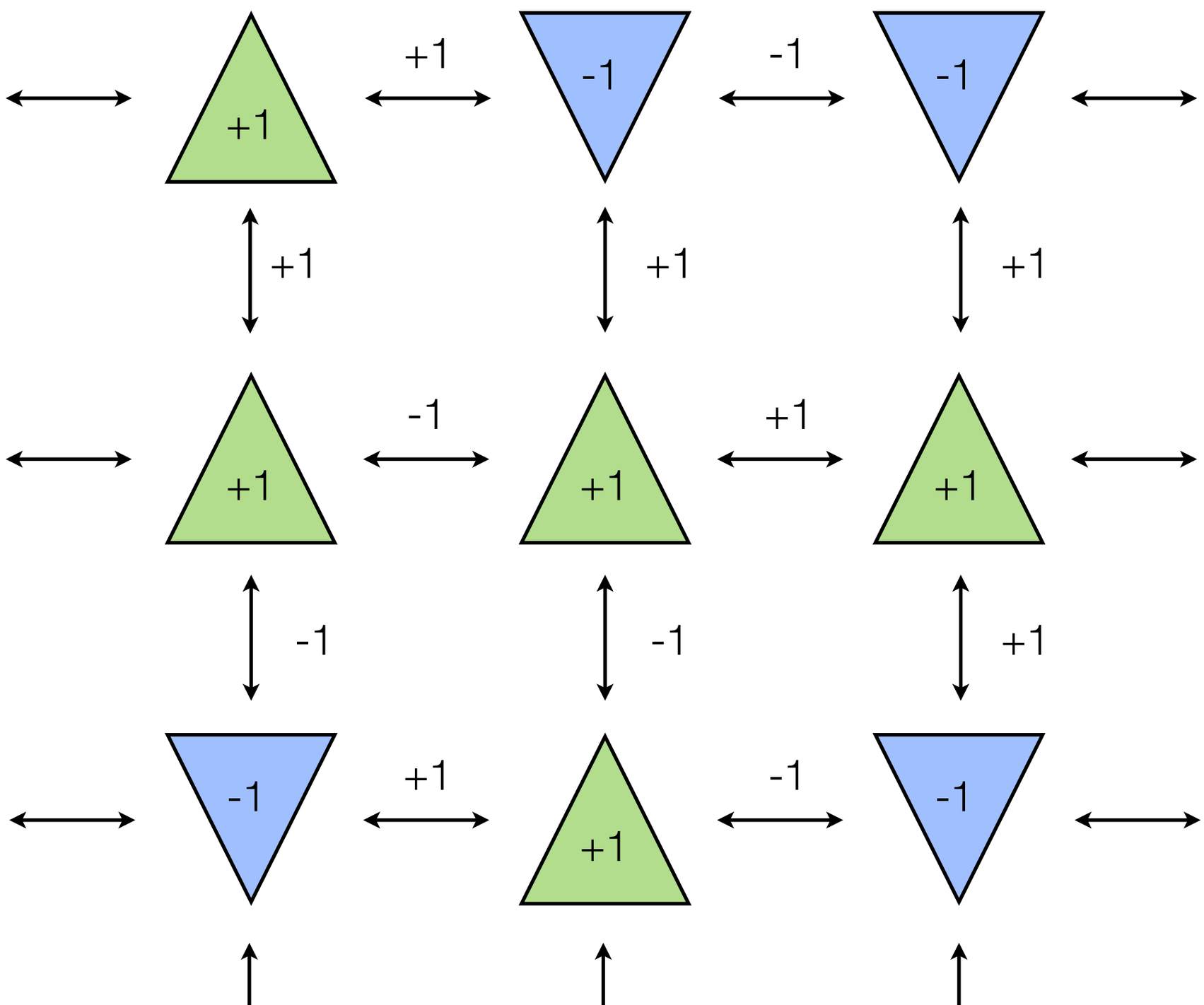
Case Studies

- Parallelising Search Algorithm
- Parallelising Fitness Evaluation
- Parallelising Software Execution

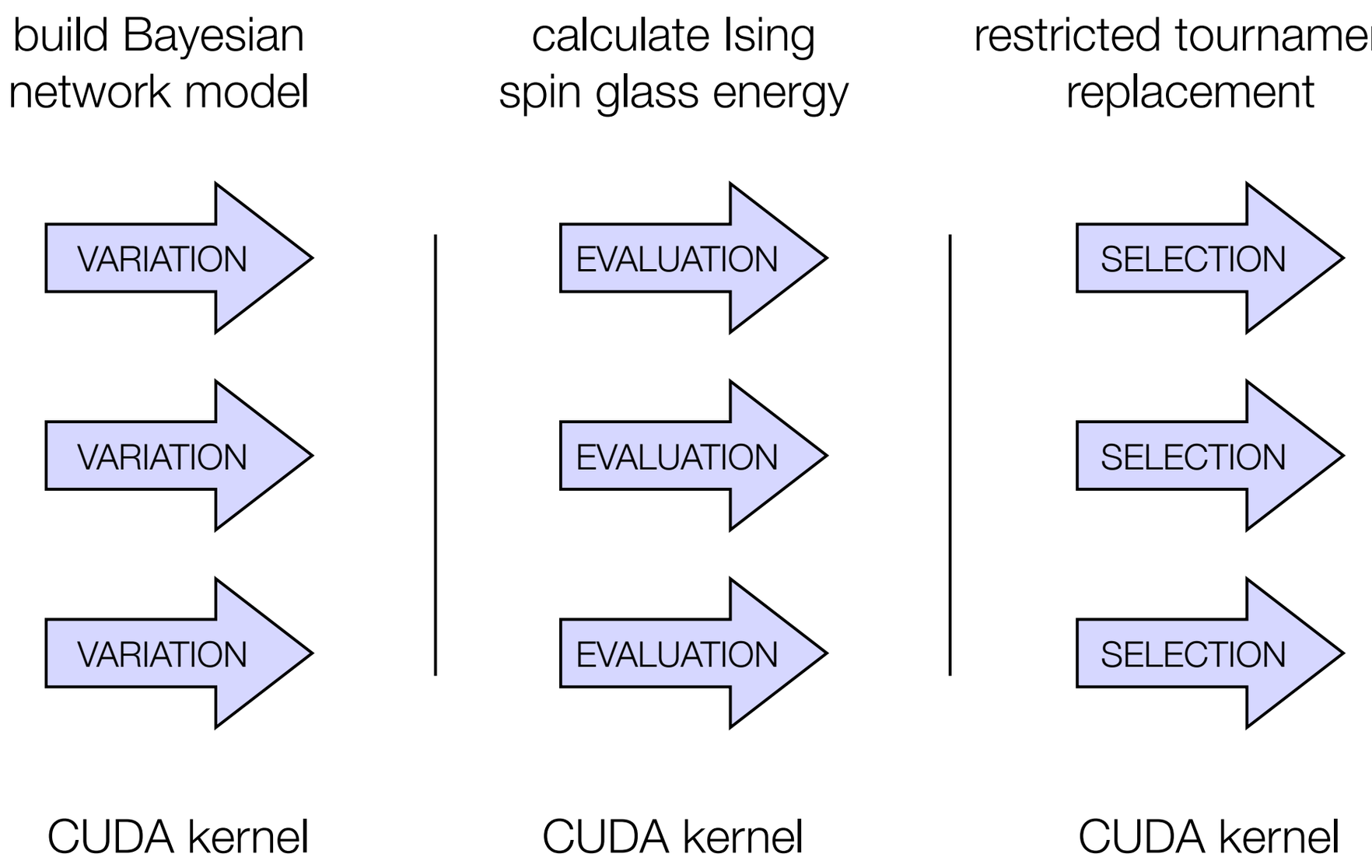
# Bayesian Optimisation Algorithm



# Ising Spin Glass

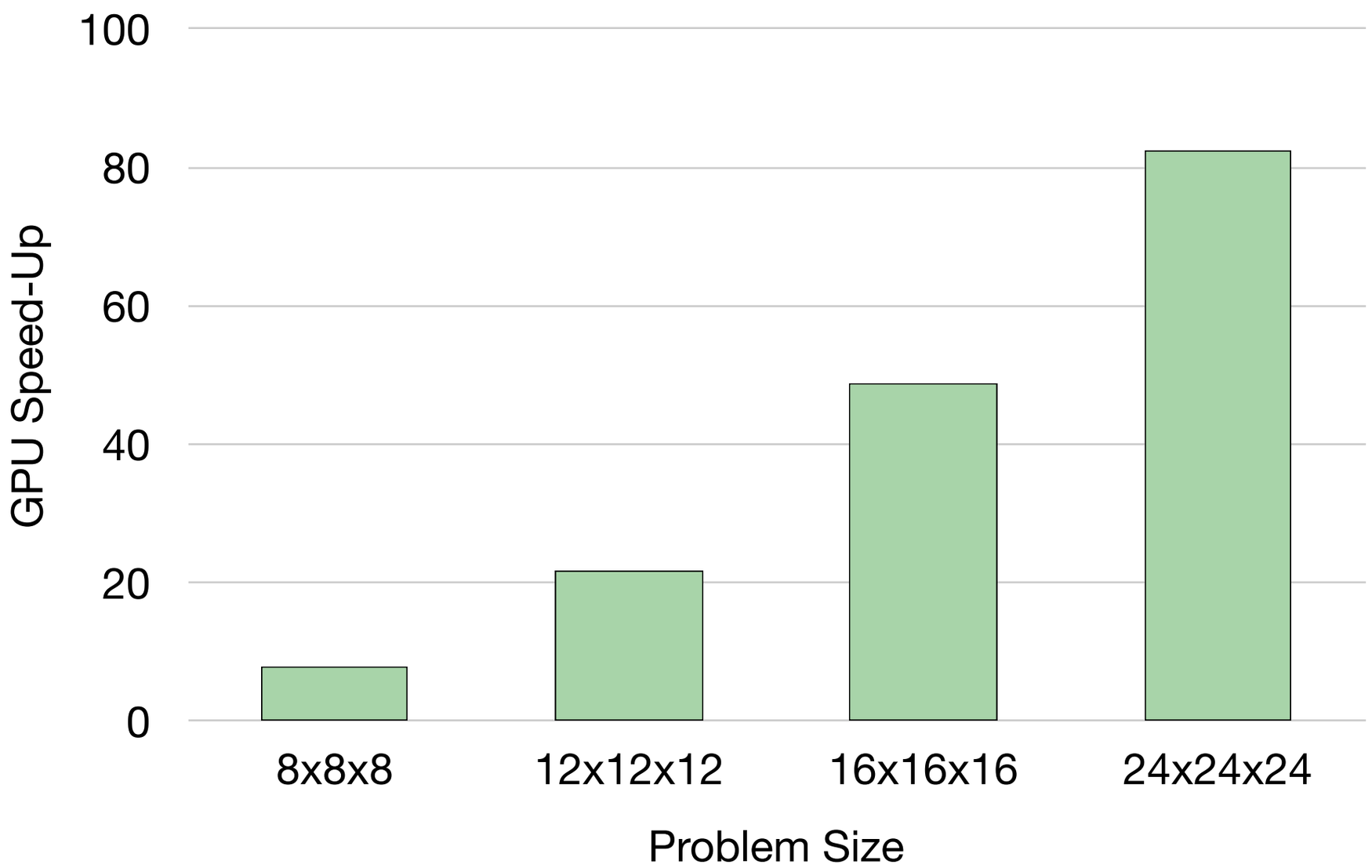


# Implementation




Poulding, Staunton, Burles, “Full Implementation of an Estimation of Distribution Algorithm on a GPU”, CIGPU Competition Entry, GECCO 2011

# Results




Poulding, Staunton, Burles, “Full Implementation of an Estimation of Distribution Algorithm on a GPU”, CIGPU Competition Entry, GECCO 2011

General Purpose Computing on GPUs (GPGPU)

CUDA Architecture

Developing CUDA Applications

Case Studies

- Parallelising Search Algorithm
- Parallelising Fitness Evaluation
- Parallelising Software Execution

# Multi-Objective Test Suite Minimisation

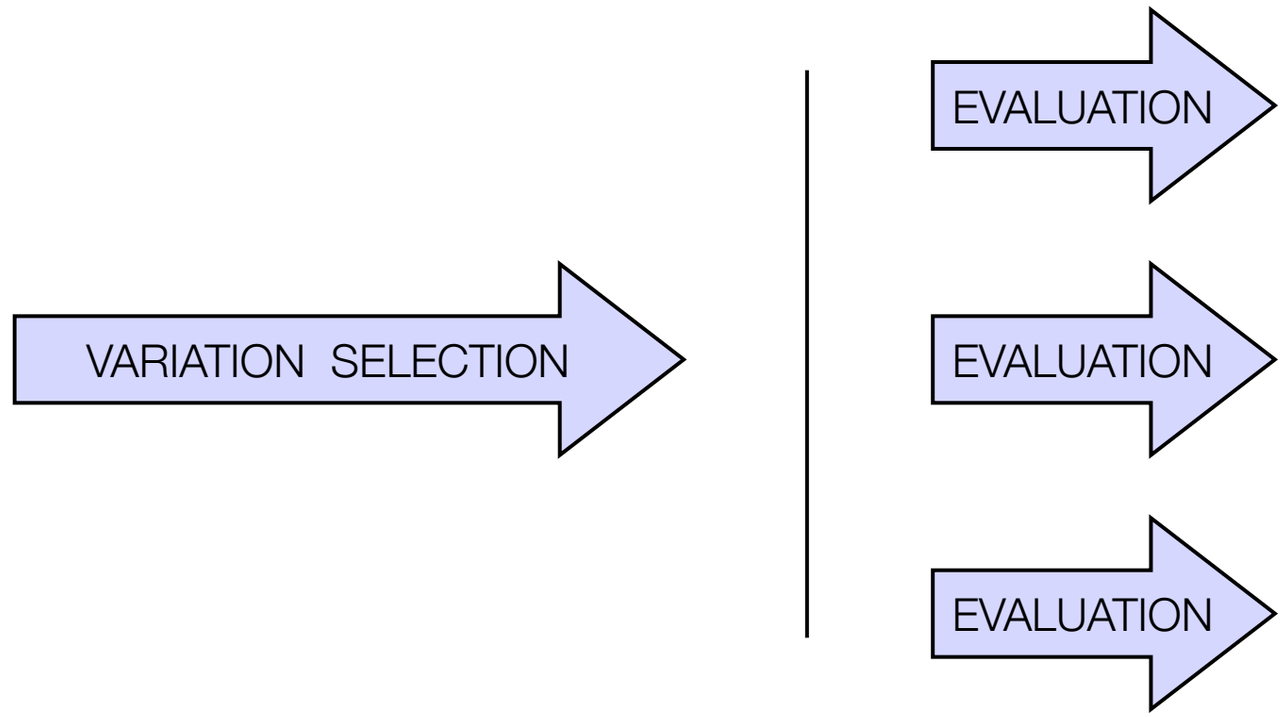
| | | test cases | | | | |
|---|---|---|---|---|---|---|
| | | $t_1$ | $t_2$ | $t_3$ | ... | $t_l$ |
| requirements | $r_1$ | 1 | 0 | 1 | ... | 0 |
| | $r_2$ | 1 | 0 | 0 | ... | 1 |
| | $r_3$ | 0 | 1 | 1 | ... | 1 |
| | ⋮ | ⋮ | ⋮ | ⋮ | | ⋮ |
| | $r_m$ | 1 | 1 | 0 | ... | 0 |
| | cost | 9 | 7 | 4 | | 6 |

Yoo, Harman, Ur, “Highly Scalable Multi Objective Test Suite Minimisation Using Graphics Cards”, SSBSE 2011

# Implementation

MO algorithm NSGA-II

calculation of coverage
and cost by
matrix multiplication

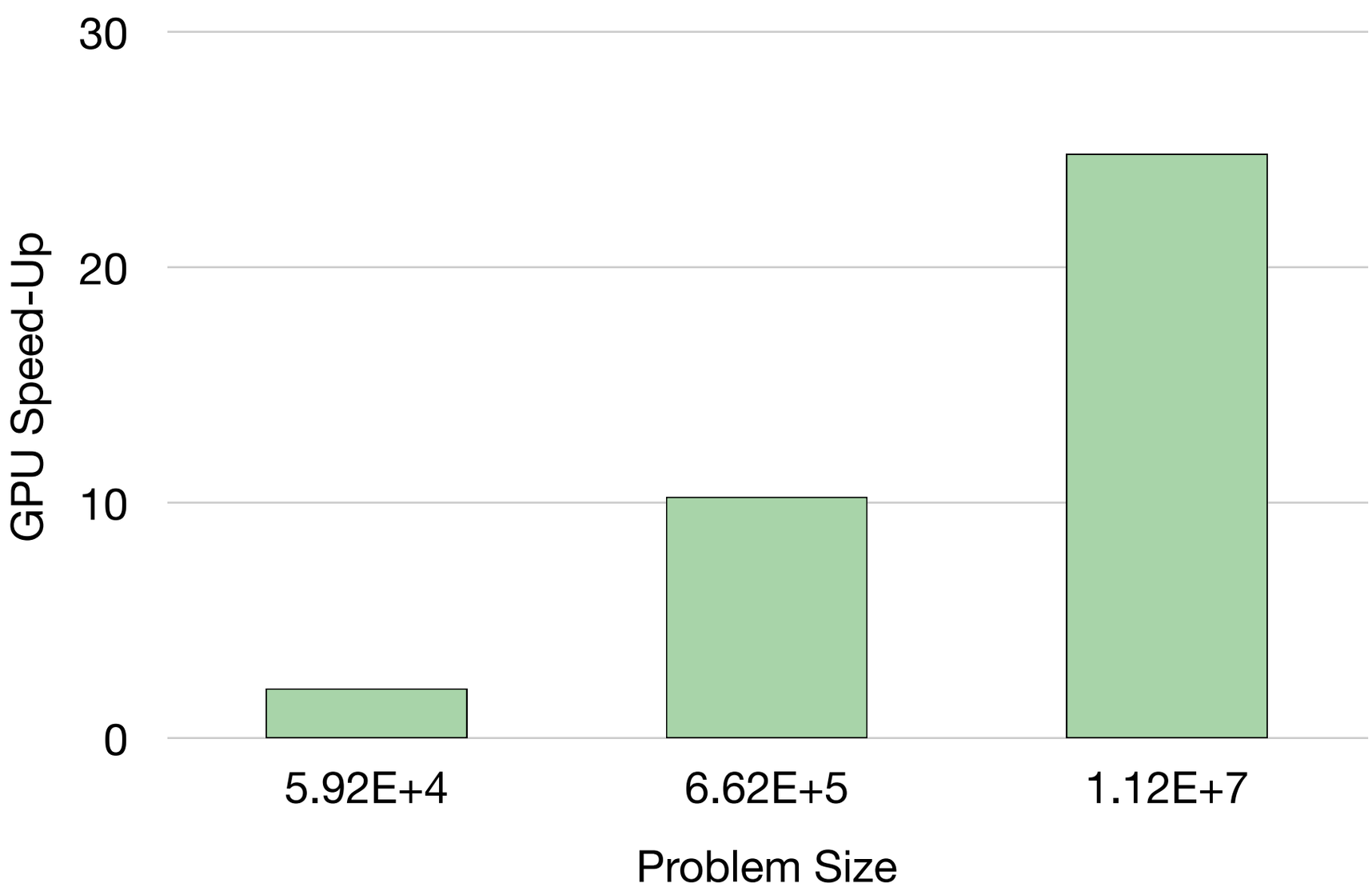



Java jMetal MOEA library

openCL using
JavaCL wrapper

Yoo, Harman, Ur, “Highly Scalable Multi Objective Test Suite Minimisation Using Graphics Cards”, SSBSE 2011

# Results




Yoo, Harman, Ur, “Highly Scalable Multi Objective Test Suite Minimisation Using Graphics Cards”, SSBSE 2011

General Purpose Computing on GPUs (GPGPU)

CUDA Architecture

Developing CUDA Applications

Case Studies

- Parallelising Search Algorithm
- Parallelising Fitness Evaluation
- Parallelising Software Execution

# Implementation

execute instrumented software
with test inputs




CUDA kernel

research funded by the MOD Centre for Defence Enterprise (CDE)

# Language Compatibility

large subset of C++

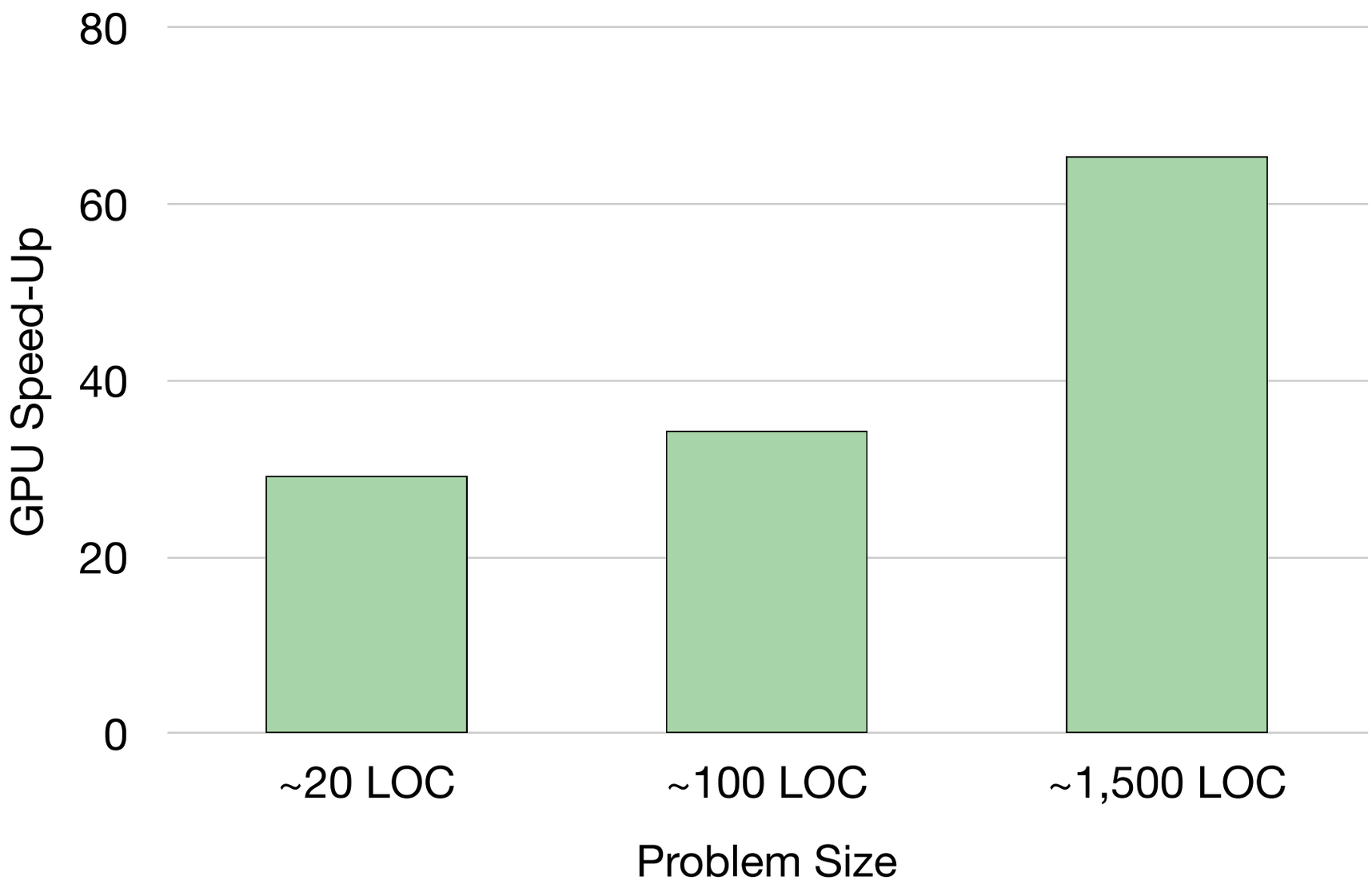
including:
- OO features
- templates
- math library
- IEEE 754 floating point compliance

missing:
- Standard Template Library
- runtime type information
- network and file IO
- rand()

only in compute capability 2.0+:
- dynamic memory allocation
- function pointers
- function recursion
- multiple source code files

research funded by the MOD Centre for Defence Enterprise (CDE)

# Results




research funded by the MOD Centre for Defence Enterprise (CDE)

# Resources

NVIDIA CUDA Zone

C Programming Guide

C Best Practices Guide

CUDA SDK samples - ‘template’ application