Accelerated Test Execution Using GPUs

Vanya Yaneva
Supervisors: Ajitha Rajan, Christophe Dubach

Mathworks
May 27, 2016

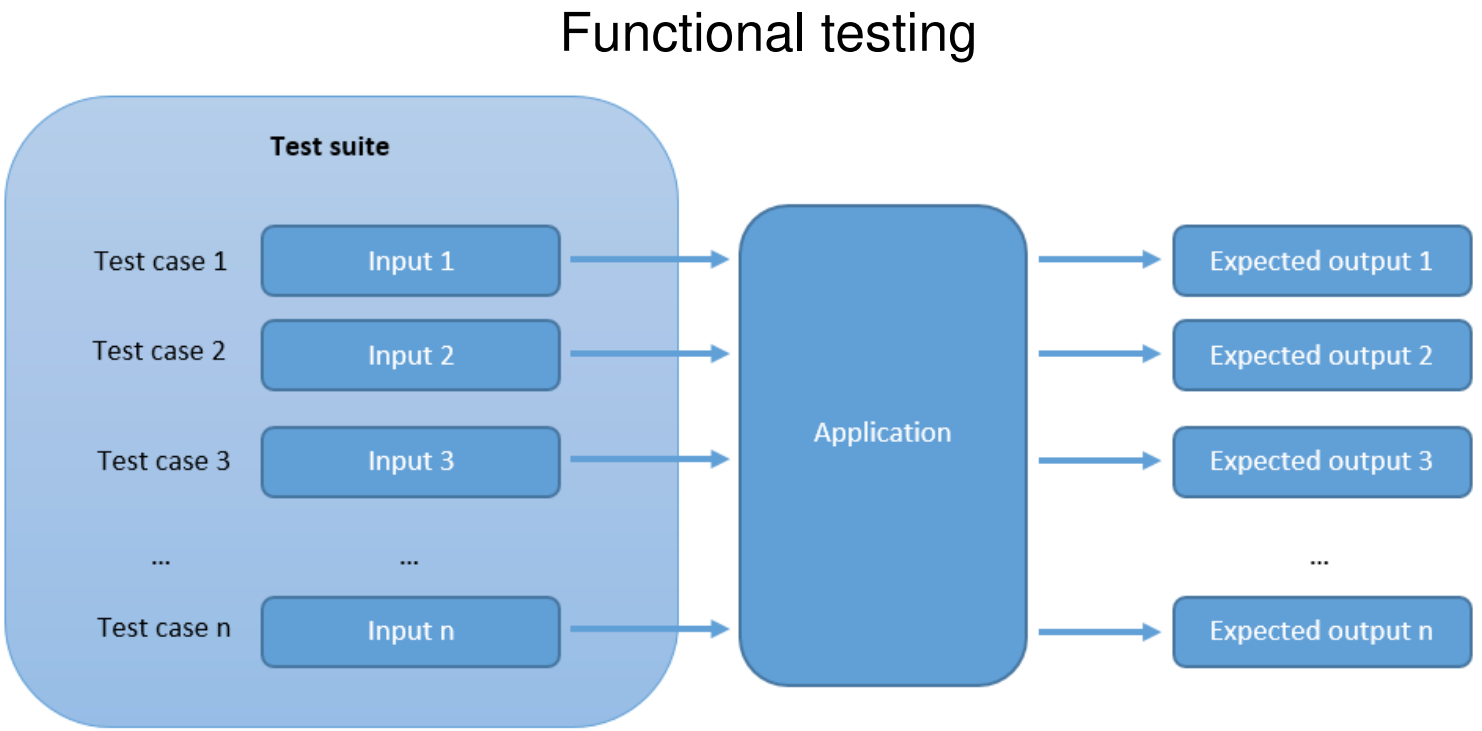
THE UNIVERSITY of EDINBURGH

EPSRC Centre for Doctoral Training in Pervasive Parallelism

EPSRC
Engineering and Physical Sciences Research Council

The Problem

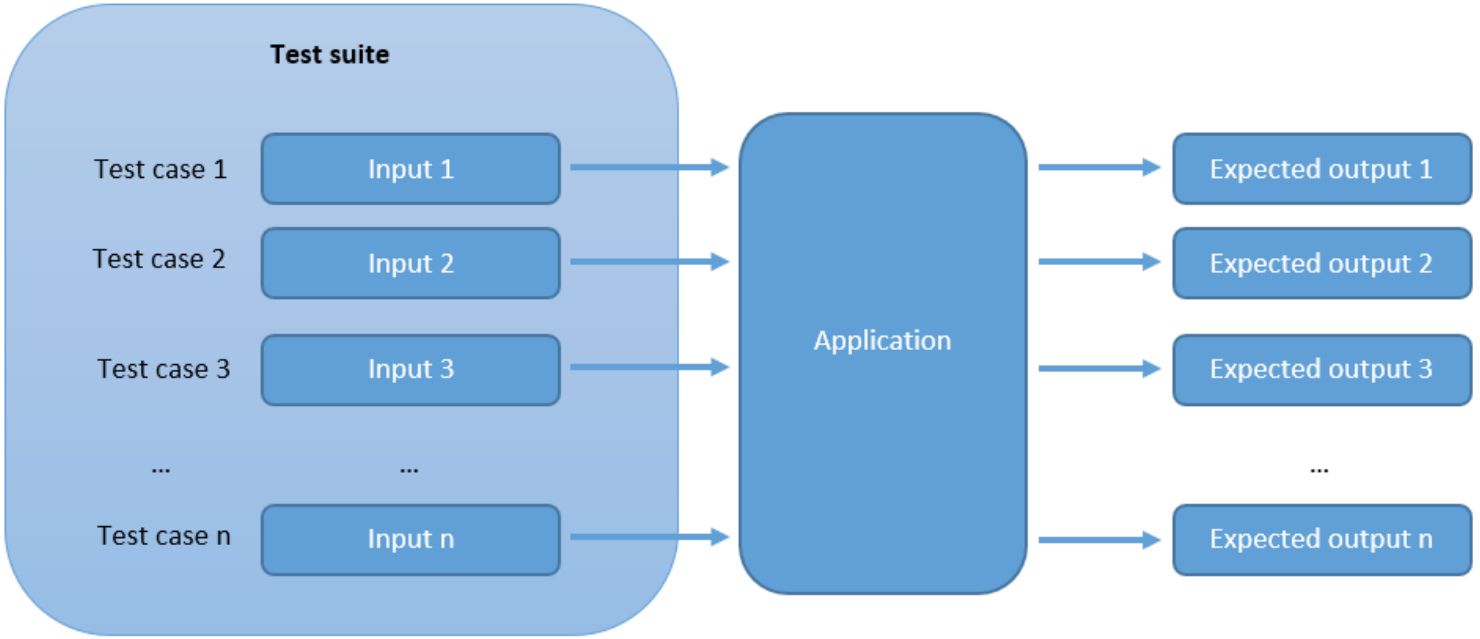
Software testing is time consuming



The Problem

Software testing is time consuming

Functional testing



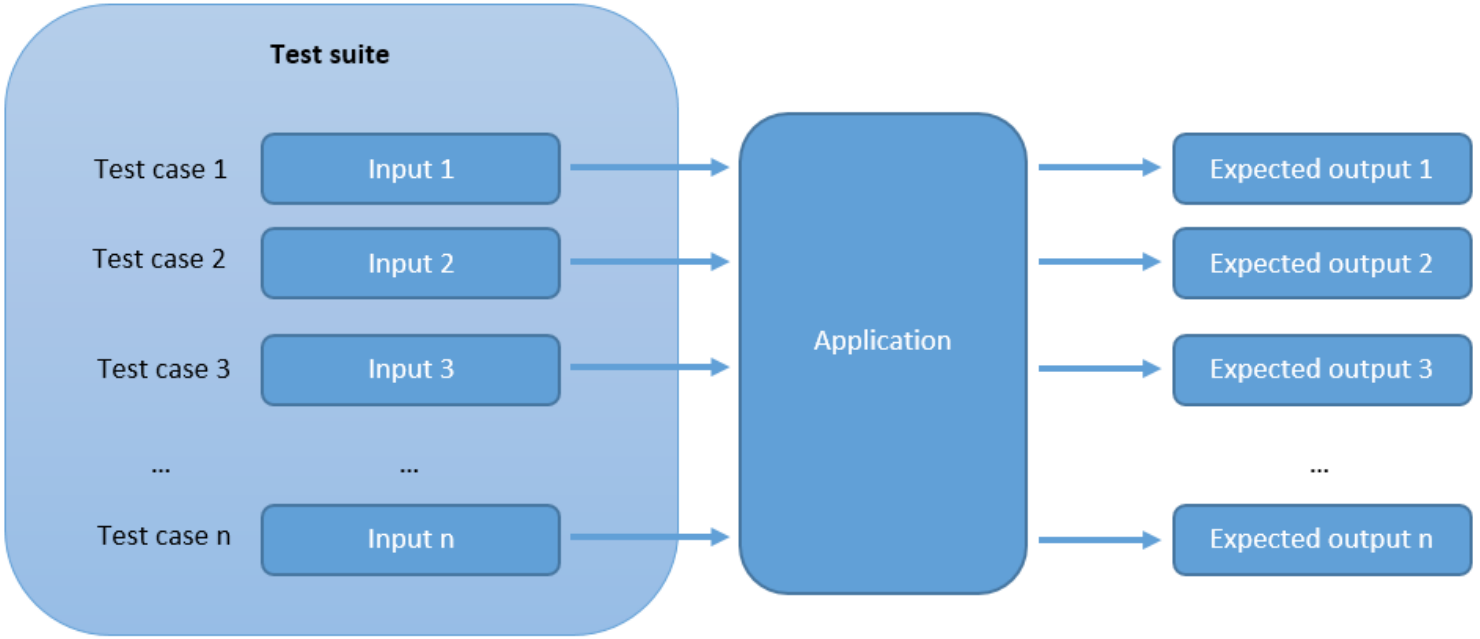
The test suite of a non-trivial system:

- could have **thousands** of test cases
- could take **hours**, **days** or even **weeks** to execute

The Problem

Software testing is time consuming

Functional testing



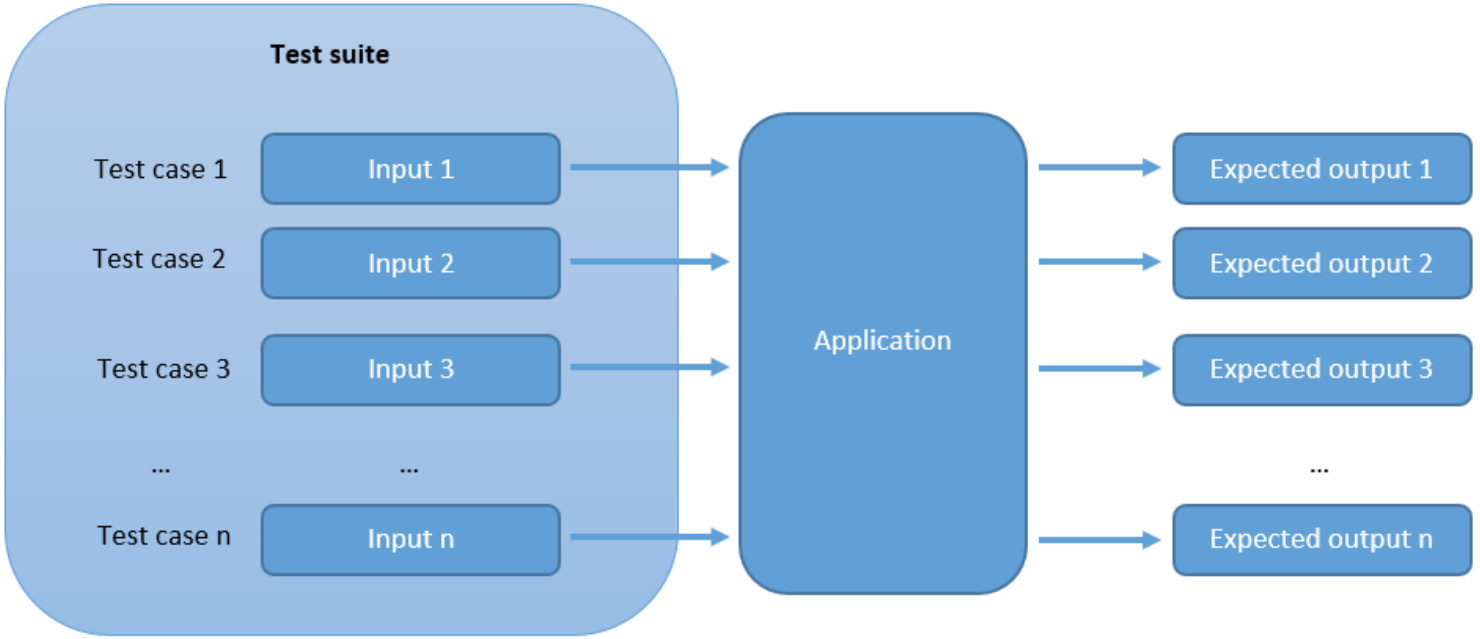
Characteristics:

1. Executions are independent

The Problem

Software testing is time consuming

Functional testing



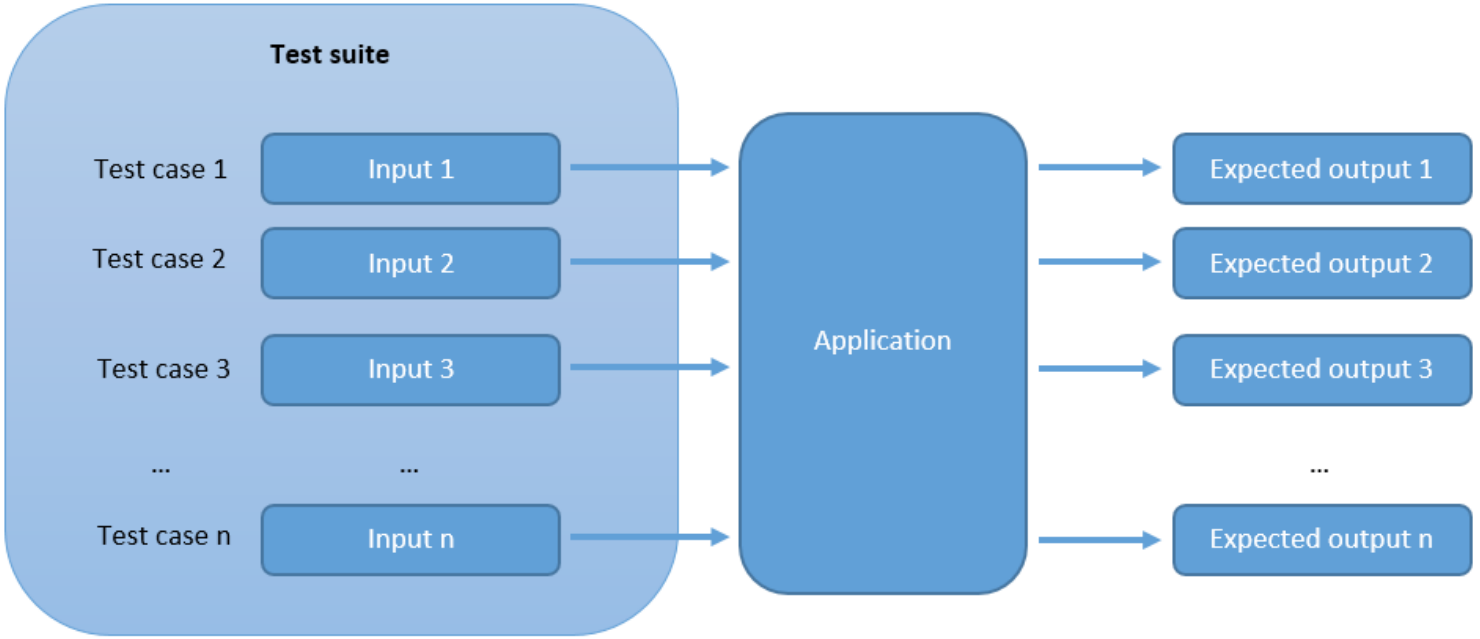
Characteristics:

1. Executions are independent
2. Executions are data parallel

The Problem

Software testing is time consuming

Functional testing



Characteristics:

1. Executions are independent
2. Executions are data parallel
3. Parallel executions are already employed in industry

CPU Servers vs. GPUs

CPU Servers

GPUs

- Comparatively expensive
- Do **not** scale to thousands of test cases
- Extremely under-utilised during testing (up to 90% stays idle)

CPU Servers vs. GPUs

CPU Servers

- Comparatively expensive
- Do **not** scale to thousands of test cases
- Extremely under-utilised during testing (up to 90% stays idle)

GPUs

- Cheap and widely available
- Large scale parallelism, thousands of threads
- SIMD architecure → suitable for testing

CPU Servers vs. GPUs

CPU Servers

GPUs

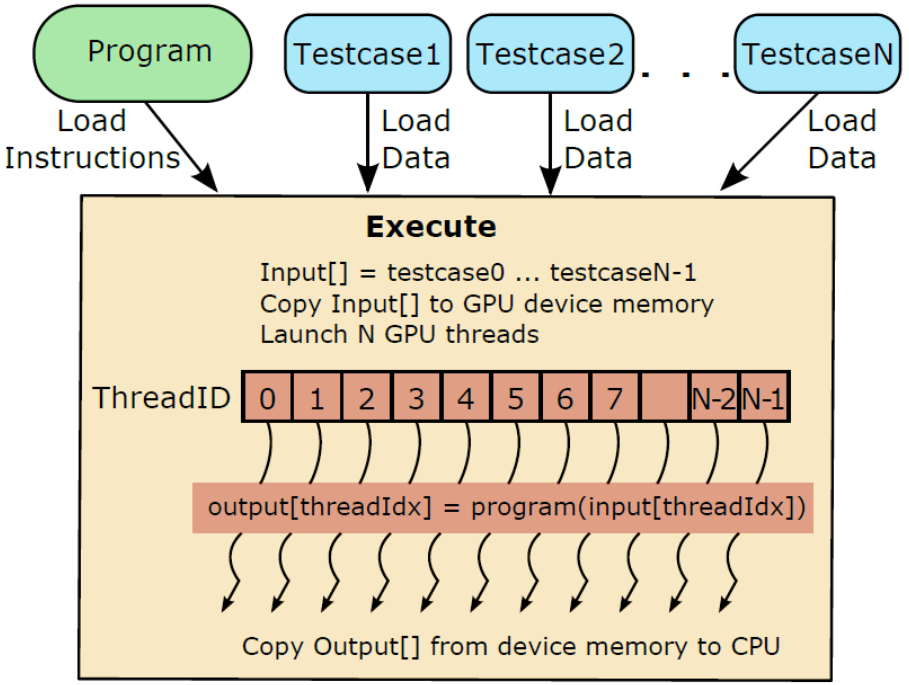
- Comparatively expensive
- Do **not** scale to thousands of test cases
- Extremely under-utilised during testing (up to 90% stays idle)

- Cheap and widely available
- Large scale parallelism, thousands of threads
- SIMD architecure → suitable for testing

GPUs have the potential to make a huge impact on **development schedules** and **costs** associated with testing.

Proposed Approach

Execute test cases in parallel on the GPU threads

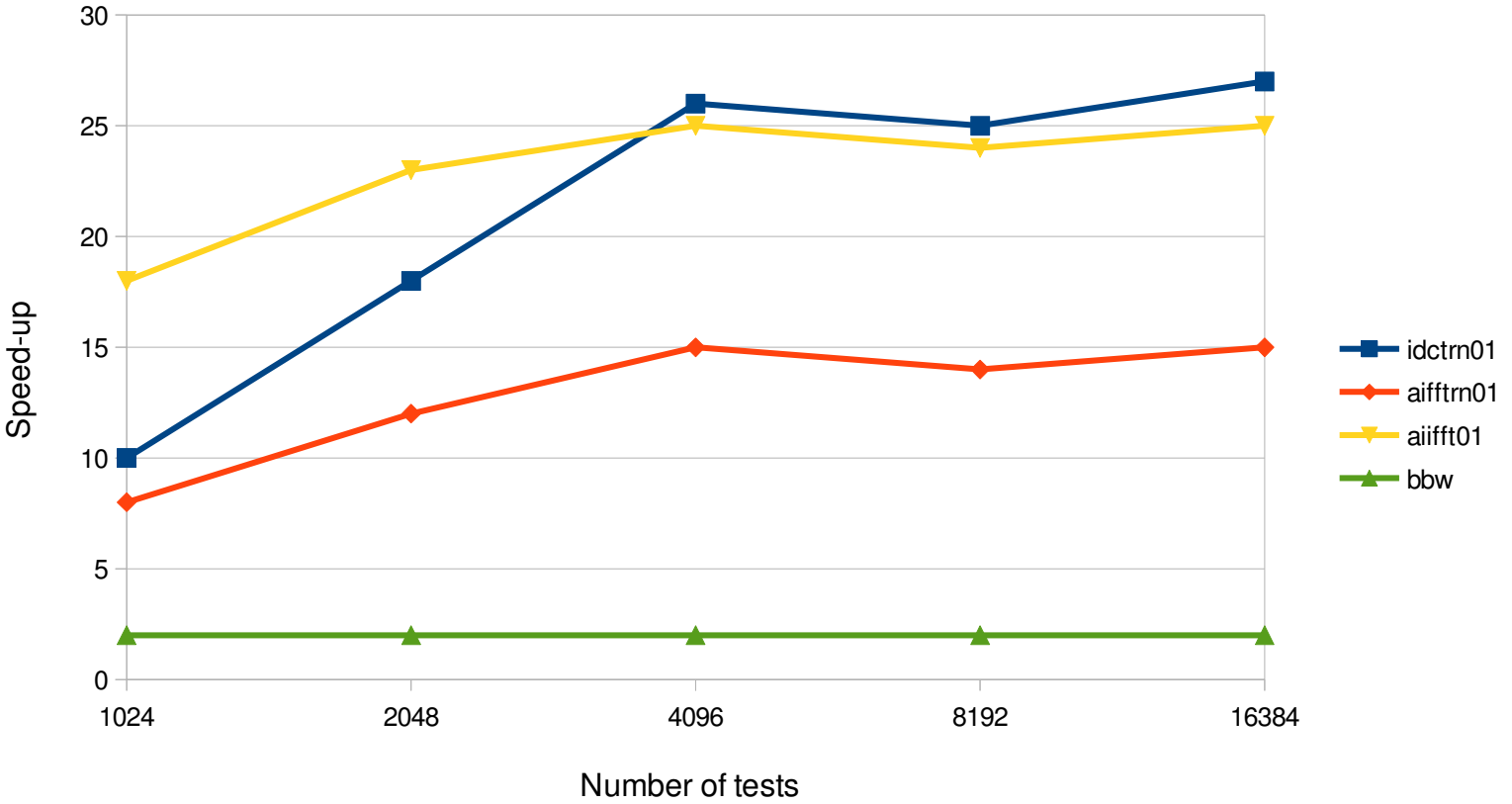


- Test cases are stored in an array and transferred to the GPU.
- Each GPU thread executes the same program on a single element of the array, i.e. on a single test case.
- Original program fuctionality is **unmodified**.

Source: A. RAJAN, S. SHARMA, P. SCHRAMMEL and D. KROENING, *Accelerated Test Execution Using GPUs* in ASE 2014

Previous Work

Performance



GPU used: Nvidia GeForce GTX 670

Source: A. RAJAN, S. SHARMA, P. SCHRAMMEL and D. KROENING, *Accelerated Test Execution Using GPUs* in ASE 2014

Challenges

1. GPUs use low-level programming models, eg. CUDA, OpenCL.
   - Test launching is not trivial to implement.
   - Not available to general programmers.

2. Only a subset of C/C++ features is supported for compilation on the GPU. Unsupported features include:

   recursion, dynamic memory allocation, standard library calls

Challenges

3. Performance is penalised by

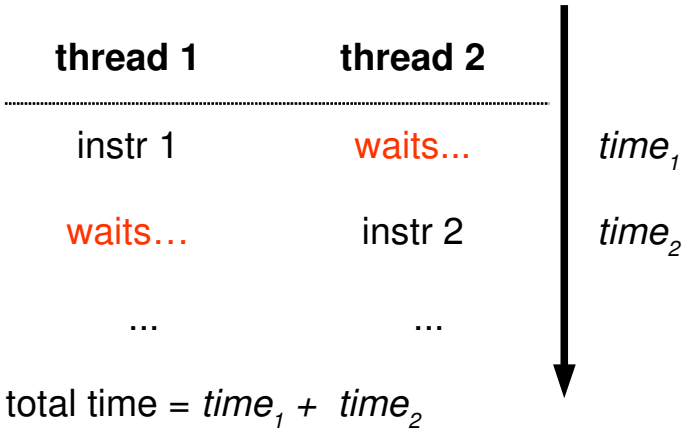
- control-flow divergence
- data transfers between the CPU and the GPU memory

Challenges

3. Performance is penalised by

- control-flow divergence

```
if (cond) {
  // instr 1
} else {
  // instr 2
}
```



total time = $time_1$ + $time_2$

- data transfers between the CPU and the GPU memory

Challenges

3. Performance is penalised by

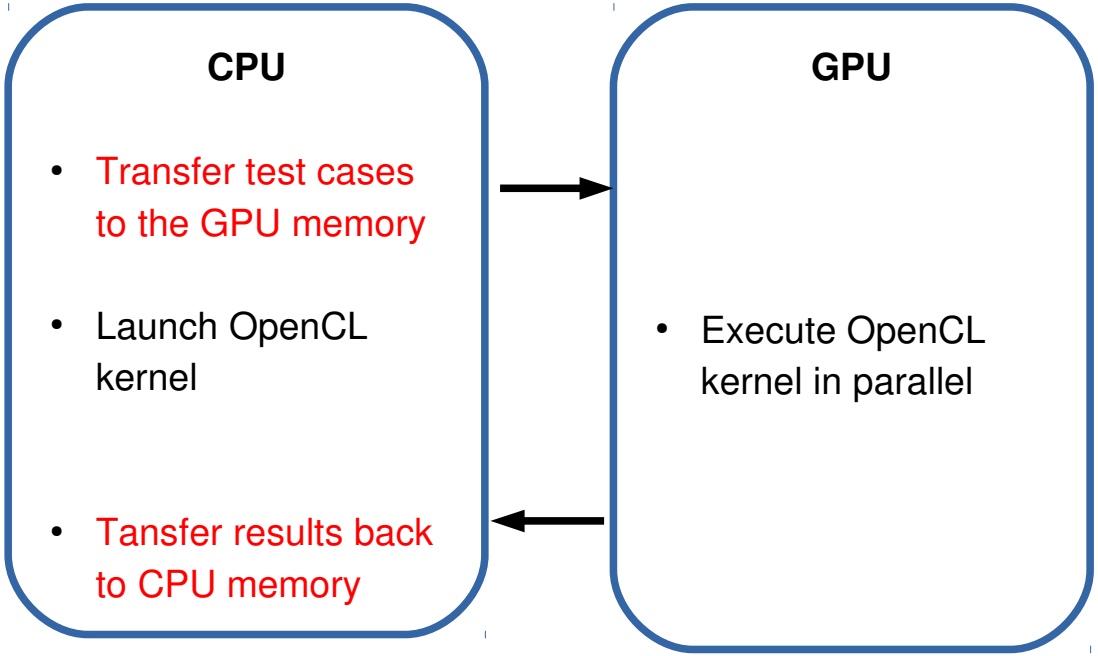
- control-flow divergence

```
if (cond) {
  // instr 1
} else {
  // instr 2
}
```

thread 1	thread 2	
instr 1	waits...	$time_1$
waits…	instr 2	$time_2$
...	...	

total time = $time_1$ + $time_2$

- data transfers between the CPU and the GPU memory

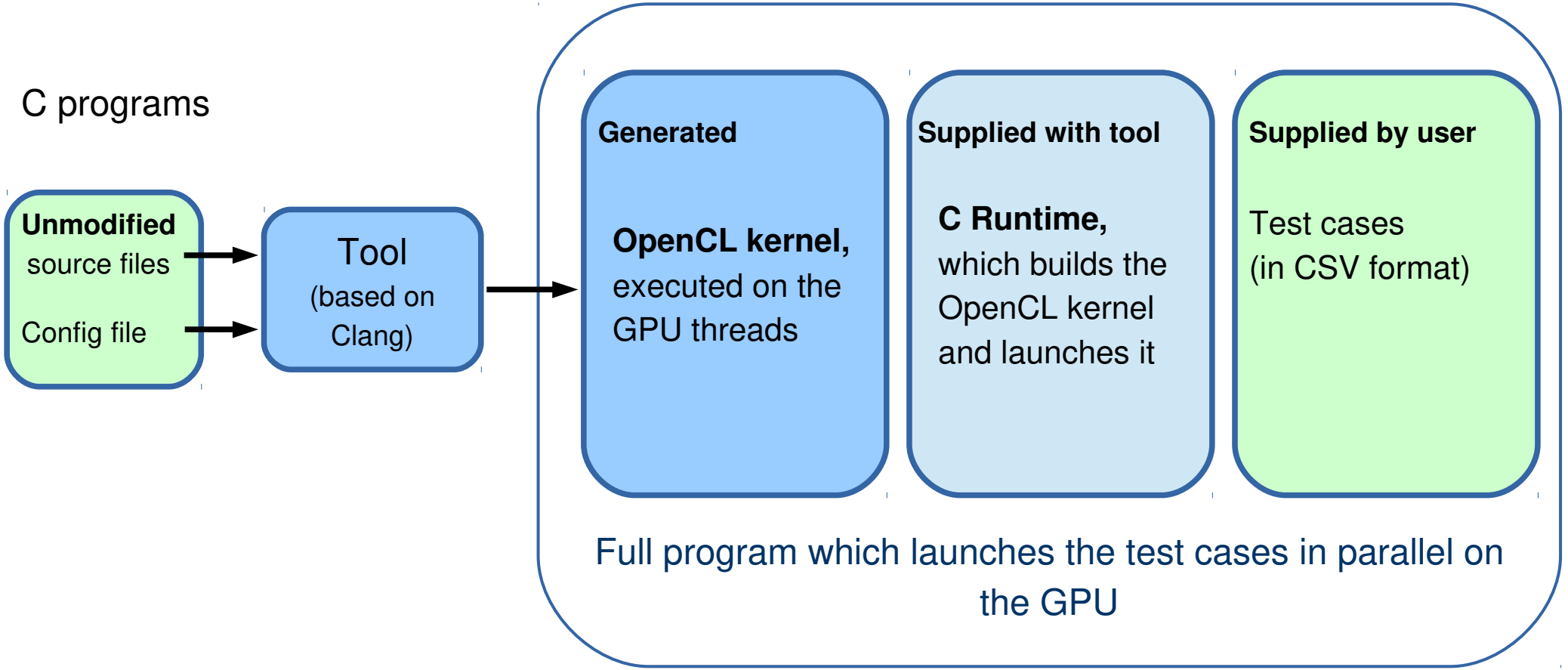


Current Work

Build a tool which generates OpenCL code to launch the test cases in parallel on the GPU.

Current Work

Build a tool which generates OpenCL code to launch the test cases in parallel on the GPU.



Example

Original code

```
int add(int a, int b) {

  return a+b;

}

int main(int argc, char* argv[]) {

  if(argc > 2)  {

   int a = atoi(argv[1]);

   int b = atoi(argv[2]);

   printf("%d + %d = %d\n", a, b,

           add(a, b));

  }

}
```

Example

Original code

```
int add(int a, int b) {

  return a+b;

}

int main(int argc, char* argv[]) {

  if(argc > 2)  {

   int a = atoi(argv[1]);

   int b = atoi(argv[2]);

   printf("%d + %d = %d\n", a, b,

           add(a, b));

  }

}
```

Generated OpenCL code

```
int add(int a, int b) {
  return a+b;
}
__kernel void main_kernel(
          __global struct input * inputs,
          __global struct result * results) {
  int idx = get_global_id(0);
  struct input input = inputs[idx];
  int argc = input.argc;
  results[idx].test_case_num = input.test_case_num;

  if(argc > 2)  {
   int a = input.a;
   int b = input.b;
   /*printf("%d + %d = %d\n", a, b, add(a, b));*/
   results[idx].result = add(a, b);
  }
}
```

Example

Original code

```
int add(int a, int b) {

  return a+b;

}

int main(int argc, char* argv[]) {

  if(argc > 2)  {

   int a = atoi(argv[1]);

   int b = atoi(argv[2]);

   printf("%d + %d = %d\n", a, b,

           add(a, b));

  }

}
```

Generated OpenCL code

```
int add(int a, int b) {

  return a+b;

}

__kernel void main_kernel(

          __global struct input * inputs,

          __global struct result * results) {

  int idx = get_global_id(0);

  struct input input = inputs[idx];

  int argc = input.argc;

  results[idx].test_case_num = input.test_case_num;


  if(argc > 2)  {

   int a = input.a;

   int b = input.b;

   /*printf("%d + %d = %d\n", a, b, add(a, b));*/

   results[idx].result = add(a, b);

  }

}
```

Example

Original code

```
int add(int a, int b) {

  return a+b;

}

int main(int argc, char* argv[]) {

  if(argc > 2)  {

   int a = atoi(argv[1]);

   int b = atoi(argv[2]);

   printf("%d + %d = %d\n", a, b,

           add(a, b));

  }

}
```

Generated OpenCL code

```
int add(int a, int b) {
  return a+b;
}
__kernel void main_kernel(
          __global struct input * inputs,
          __global struct result * results) {
  int idx = get_global_id(0);
  struct input input = inputs[idx];
  int argc = input.argc;
  results[idx].test_case_num = input.test_case_num;

  if(argc > 2)  {
   int a = input.a;
   int b = input.b;
   /*printf("%d + %d = %d\n", a, b, add(a, b));*/
   results[idx].result = add(a, b);
  }
}
```

Example

Original code

```
int add(int a, int b) {

  return a+b;

}

int main(int argc, char* argv[]) {

  if(argc > 2)  {

   int a = atoi(argv[1]);

   int b = atoi(argv[2]);

   printf("%d + %d = %d\n", a, b,

           add(a, b));

  }

}
```

Generated OpenCL code

```
int add(int a, int b) {

  return a+b;

}

__kernel void main_kernel(

          __global struct input * inputs,

          __global struct result * results) {

  int idx = get_global_id(0);

  struct input input = inputs[idx];

  int argc = input.argc;

  results[idx].test_case_num = input.test_case_num;


  if(argc > 2)  {

   int a = input.a;

   int b = input.b;

   /*printf("%d + %d = %d\n", a, b, add(a, b));*/

   results[idx].result = add(a, b);

  }

}
```

Example

Original code

```
int add(int a, int b) {

  return a+b;

}

int main(int argc, char* argv[]) {

  if(argc > 2)  {

   int a = atoi(argv[1]);

   int b = atoi(argv[2]);

   printf("%d + %d = %d\n", a, b,

          add(a, b));

  }

}
```

Generated OpenCL code

```
int add(int a, int b) {

  return a+b;

}

__kernel void main_kernel(

          __global struct input * inputs,

          __global struct result * results) {

  int idx = get_global_id(0);

  struct input input = inputs[idx];

  int argc = input.argc;

  results[idx].test_case_num = input.test_case_num;


  if(argc > 2)  {

   int a = input.a;

   int b = input.b;

   /*printf("%d + %d = %d\n", a, b, add(a, b));*/

   results[idx].result = add(a, b);

  }

}
```

Example

Original code

```
int add(int a, int b) {

 return a+b;

}

int main(int argc, char* argv[]) {

  if(argc > 2)  {

   int a = atoi(argv[1]);

   int b = atoi(argv[2]);

   printf("%d + %d = %d\n", a, b,

           add(a, b));

  }

}
```

Generated OpenCL code

```
int add(int a, int b) {

 return a+b;

}

__kernel void main_kernel(

          __global struct input * inputs,

          __global struct result * results) {

  int idx = get_global_id(0);

  struct input input = inputs[idx];

  int argc = input.argc;

  results[idx].test_case_num = input.test_case_num;


  if(argc > 2)  {

   int a = input.a;

   int b = input.b;

   /*printf("%d + %d = %d\n", a, b, add(a, b));*/

   results[idx].result = add(a, b);

  }

}
```

C Features

Out of the box

- simple data types, structs, vectors, one-dim arrays
- pure functions, function calls, double precision (for OpenCL 1.2)

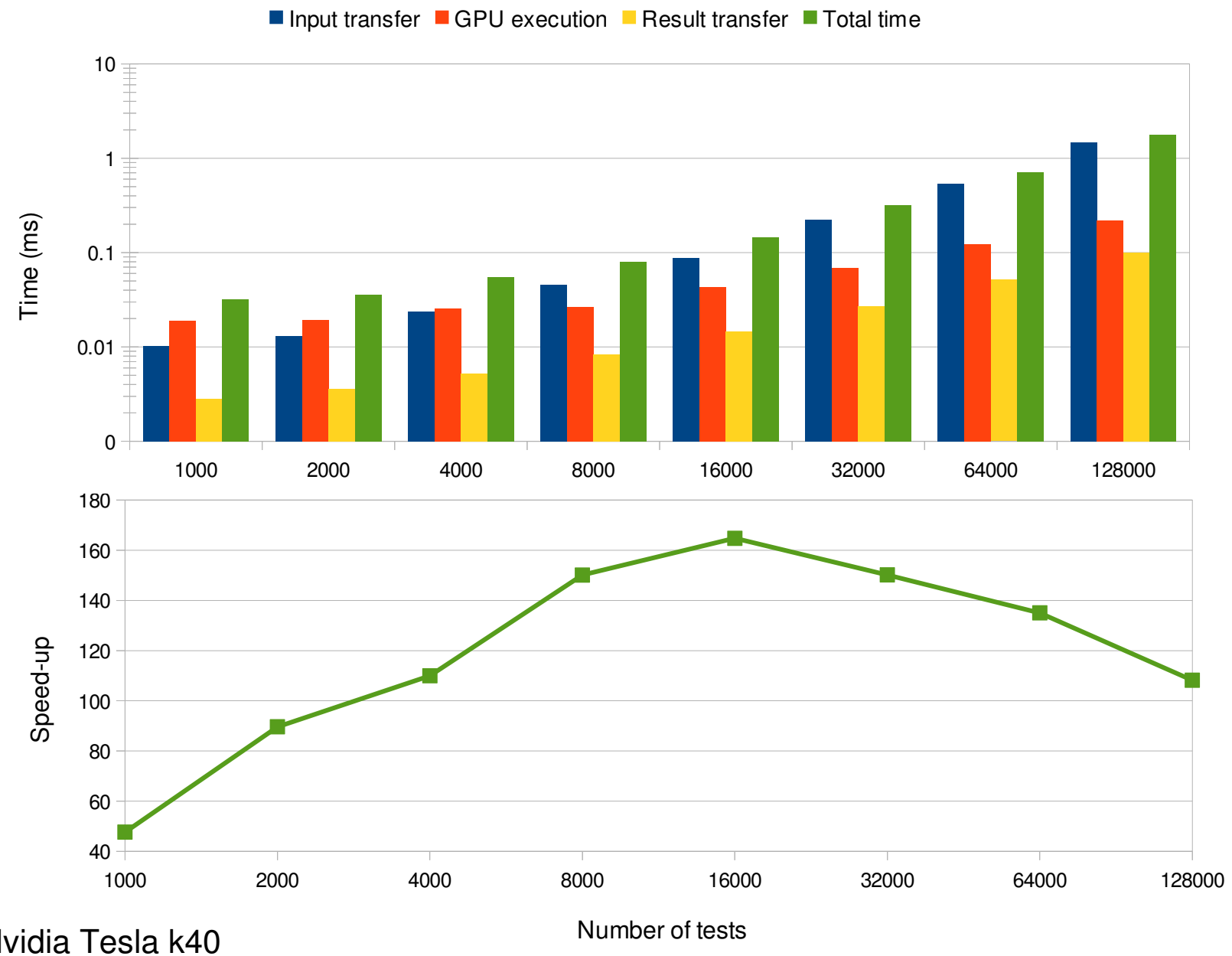
With transformations

- global scope variables: turn them into local variables and pass them as function arguments
- std in/out
- system calls: partition function and perform on the CPU

Currently unsupported

- library calls: study what library implementations exist for OpenCL; use my implementations
- multi-dimensional arrays
- dynamic memory allocation
- recursion: not possible in OpenCL (yet), but can be emulated with a stack

Preliminary Performance Results - tcas



GPU used: Nvidia Tesla k40

THE UNIVERSITY of EDINBURGH

EPSRC Centre for Doctoral Training in Pervasive Parallelism

EPSRC Engineering and Physical Sciences Research Council

Future Work

1. Evaluate, using benchmarks from different domains, eg. automotive, banking, embedded systems:
   - characterise the applications for which GPU testing is feasible

Future Work

1. Evaluate, using benchmarks from different domains, eg. automotive, banking, embedded systems:
   - characterise the applications for which GPU testing is feasible

2. Address performance issues:
   - data transfer
     - split test cases in groups and overlap transfer to the GPU with test case execution

Future Work

1. Evaluate, using benchmarks from different domains, eg. automotive, banking, embedded systems:
   - characterise the applications for which GPU testing is feasible

2. Address performance issues:
   - data transfer
     - split test cases in groups and overlap transfer to the GPU with test case execution
   - control-flow divergence
     - group test cases with similar control-flow paths together

Future Work

1. Evaluate, using benchmarks from different domains, eg. automotive, banking, embedded systems:
   - characterise the applications for which GPU testing is feasible

2. Address performance issues:
   - data transfer
     - → split test cases in groups and overlap transfer to the GPU with test case execution
   - control-flow divergence
     - → group test cases with similar control-flow paths together

3. Extend the tool to support additional C features