COMPILER-ASSISTED TEST ACCELERATION ON GPUS FOR EMBEDDED SOFTWARE

VANYA YANEVA

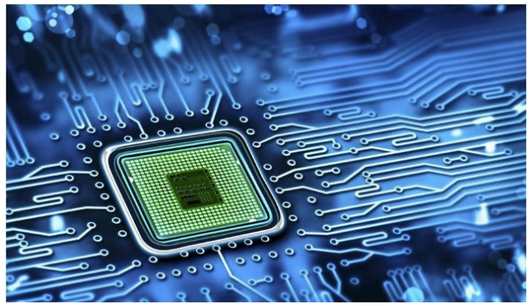
Ajitha Rajan, Christophe Dubach

ISSTA 2017
10 July 2017
Santa Barbara, CA

THE UNIVERSITY *of* EDINBURGH
informatics

EPSRC Centre for Doctoral Training in
Pervasive Parallelism

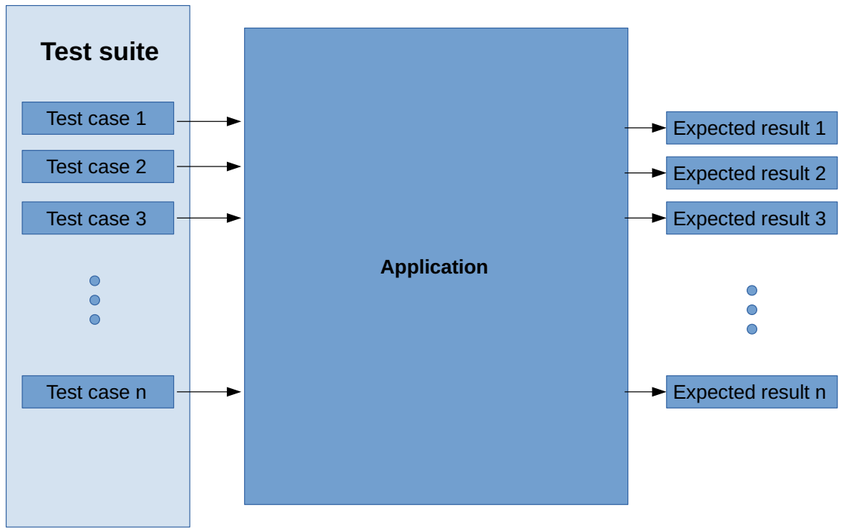
EMBEDDED SOFTWARE IS EVERYWHERE



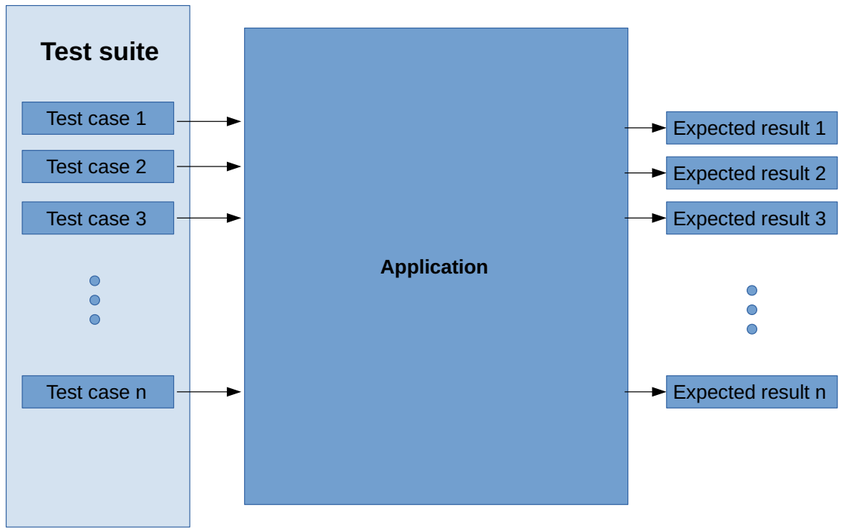
- ITS SAFETY AND CORRECTNESS ARE CRUCIAL
- FUNCTIONAL TESTING IS CRITICAL

FUNCTIONAL TESTING CAN BE EXTREMELY TIME CONSUMING

FUNCTIONAL TESTING CAN BE EXTREMELY TIME CONSUMING



FUNCTIONAL TESTING CAN BE EXTREMELY TIME CONSUMING



TESTING IS AN IDEAL CANDIDATE FOR PARALLELISATION

CPU SERVERS

- Expensive
- Do **not** scale easily as test suites grow
- Can be extremely underutilised

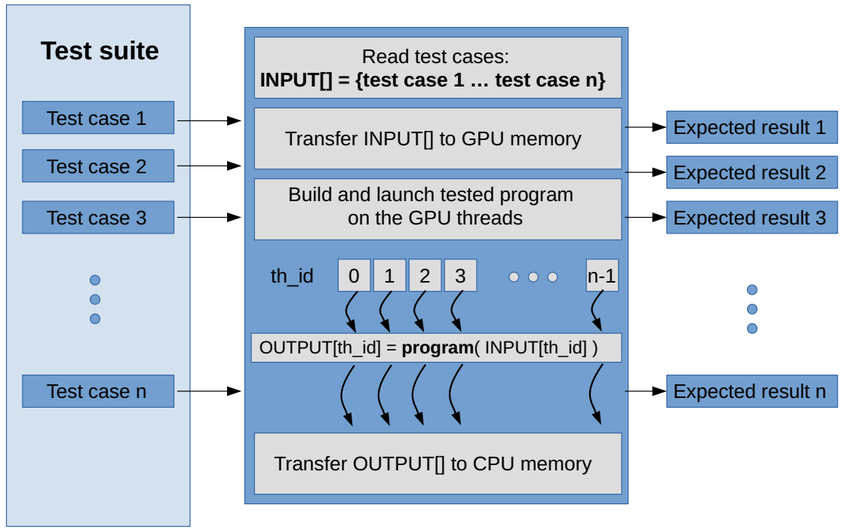
CPU SERVERS

- Expensive
- Do **not** scale easily as test suites grow
- Can be extremely underutilised

GPUS

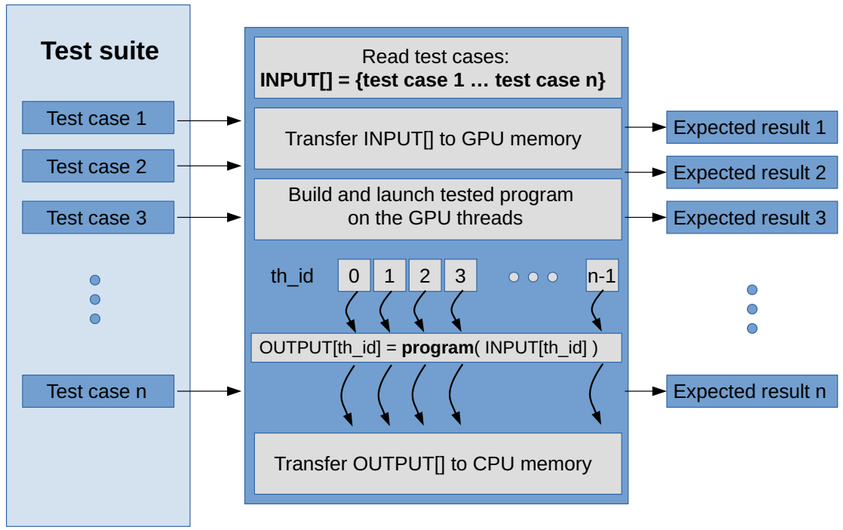
- Cheap and widely available
- Large-scale parallelism, thousands of threads
- SIMD architecture suited to functional testing

EXECUTE TESTS IN PARALLEL ON THE GPU THREADS



A. Rajan, S. Sharma, P. Schrammel, D. Kroening. ***Accelerated test execution using GPUs.*** *In proceedings of ASE 2014, pages 97-102, Sweden, Nov 2014.*

EXECUTE TESTS IN PARALLEL ON THE GPU THREADS

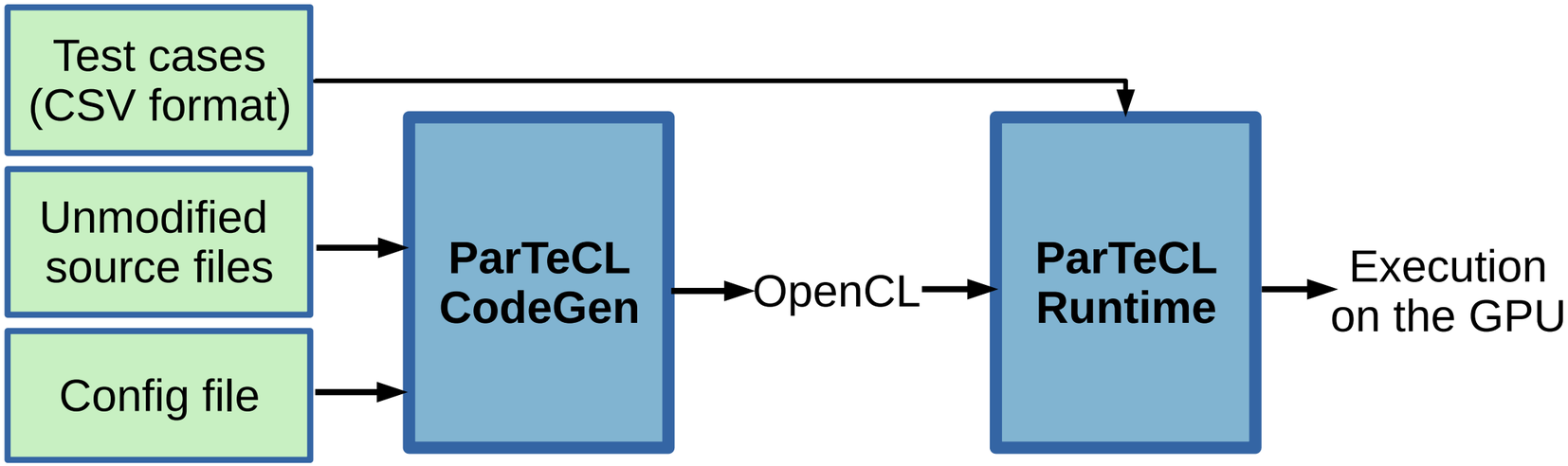


CHALLENGES

Usability	✘
Scope	✘
Performance	?

A. Rajan, S. Sharma, P. Schrammel, D. Kroening. ***Accelerated test execution using GPUs.*** *In proceedings of ASE 2014, pages 97-102, Sweden, Nov 2014.*

INTRODUCING PARTECL



INPUTS

Example:

```
#include <stdio.h>
#include <stdlib.h>

int c;

int addc(int a, int b){
  return a + b + c;
}

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

  int a = atoi(argv[1]);
  int b = atoi(argv[2]);
  c = 3;

  int sum = addc(a, b);

  printf("%d + %d + %c = %d\n", a, b, c, sum);
}
```

Configuration:

```
input: int a 1
input: int b 2
result: int sum variable: sum
```

Test cases:

```
1   13    7
2   50   22
3 1000    0
4    0 1000
5    0    0
```

PARTECL CODEGEN

Example:

```
#include <stdio.h>
#include <stdlib.h>

int c;

int addc(int a, int b){
  return a + b + c;
}

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

  int a = atoi(argv[1]);
  int b = atoi(argv[2]);
  c = 3;

  int sum = addc(a, b);

  printf("%d + %d + %c = %d\n", a, b, c, sum);
}
```

OpenCL:

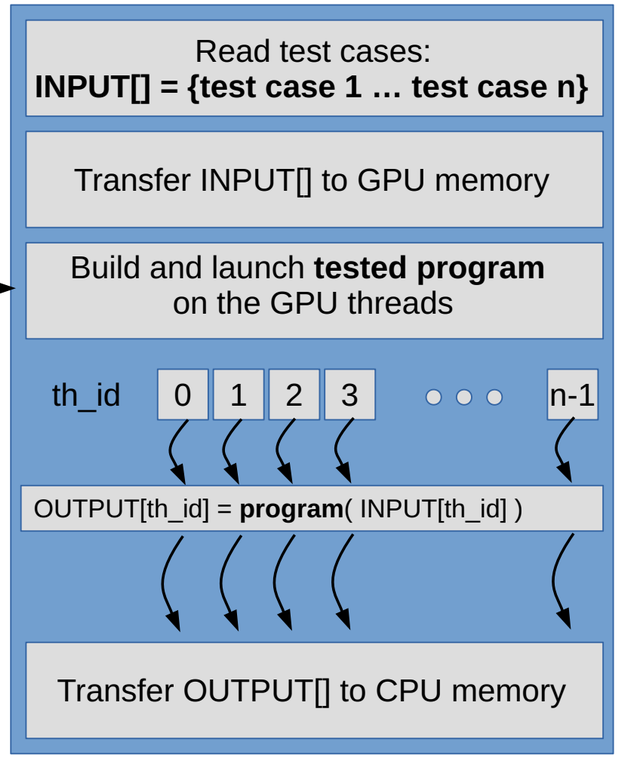
```
#include "structs.h"
//#include <stdio.h>
//#include <stdlib.h>

/*int c;*/
int addc(int a, int b, int *c){
  return a + b + (*c);
}

kernel void main_kernel(
    global struct test_input* inputs,
    global struct test_result* results){

  int idx = get_global_id(0);
  struct test_input input_gen = inputs[idx];
  global struct test_result *result_gen = &results[idx];

  int argc = input_gen.argc;
  result_gen->test_case_num = input_gen.test_case_num;

  int c;

  int a = input_gen.a;
  int b = input_gen.b;
  c = 3;

  int sum = addc(a, b, &c);

  /*printf("%d + %d + %c = %d\n", a, b, c, sum);*/
  result_gen->sum = sum;
}
```

CODE TRANSFORMATIONS

- global scope variables
- command line arguments
- standard in/out
- standard library (partial support): **clClibc**

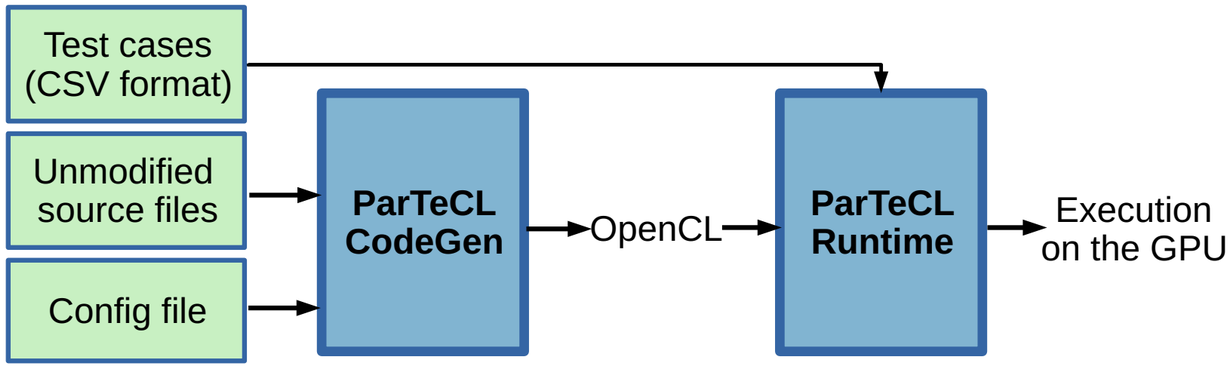
PARTECL RUNTIME

Automatically generated
OpenCL



CHALLENGES

Usability	✔
Scope	✔
Performance	?



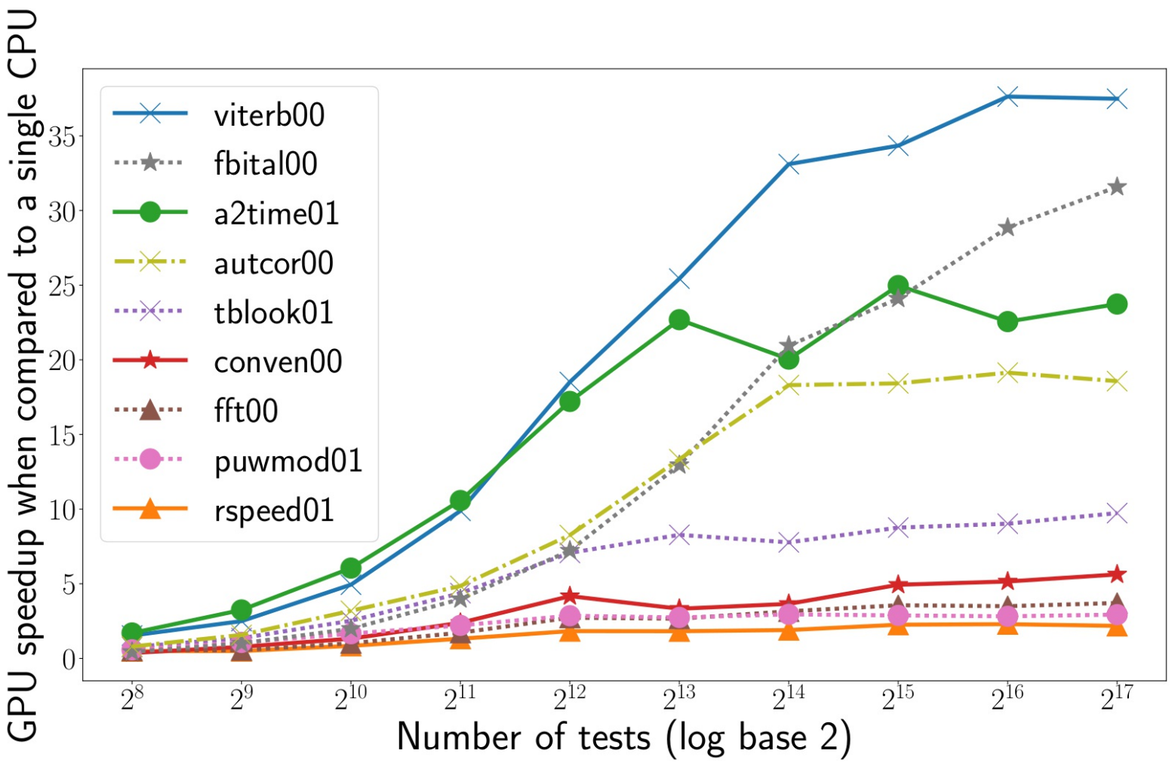
EVALUATION

1. Speedup against CPU
2. Data transfer overhead
3. Comparison to a multi-core CPU
4. Correctness

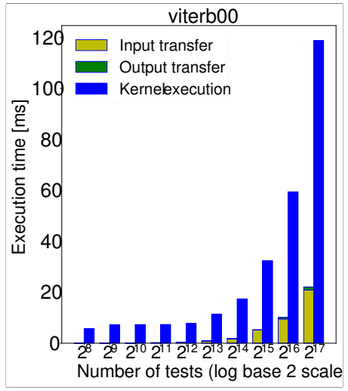
EXPERIMENT

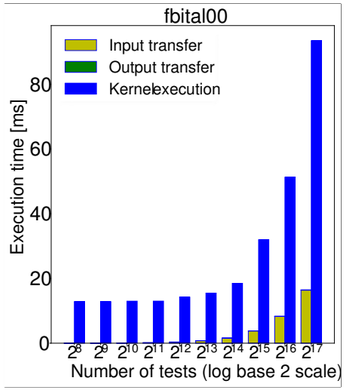
- **Subjects:** EEMBC - Industry-standard benchmark suite for embedded software
- **Hardware:** GPU - NVidia Tesla K40m; CPU - Intel Xeon, 8 cores
- **Test suite size:** 130K

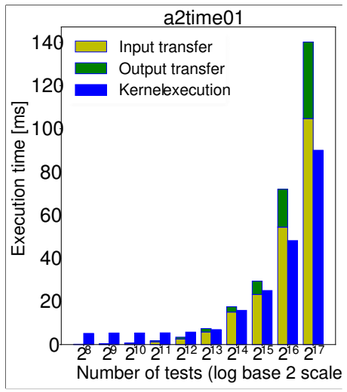
SPEEDUP AGAINST CPU

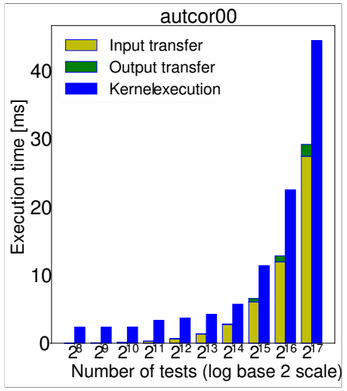


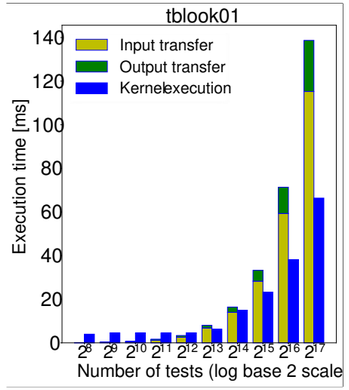
DATA TRANSFER OVERHEAD

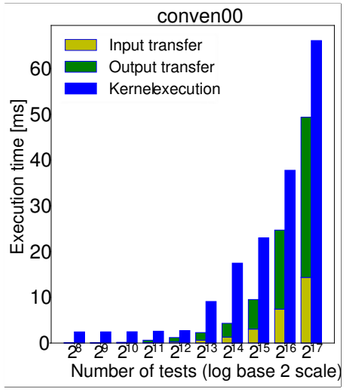


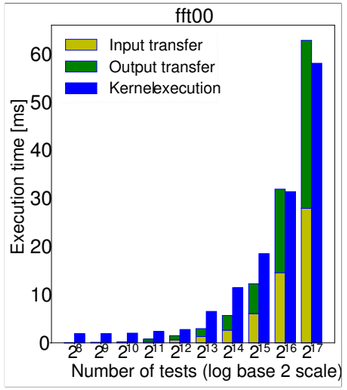


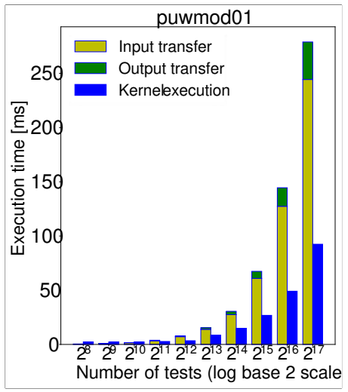


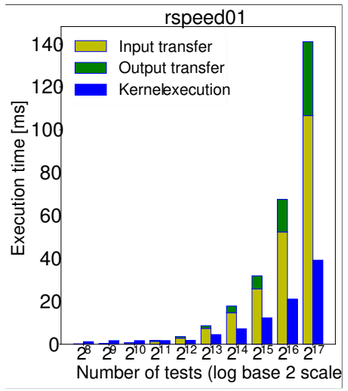




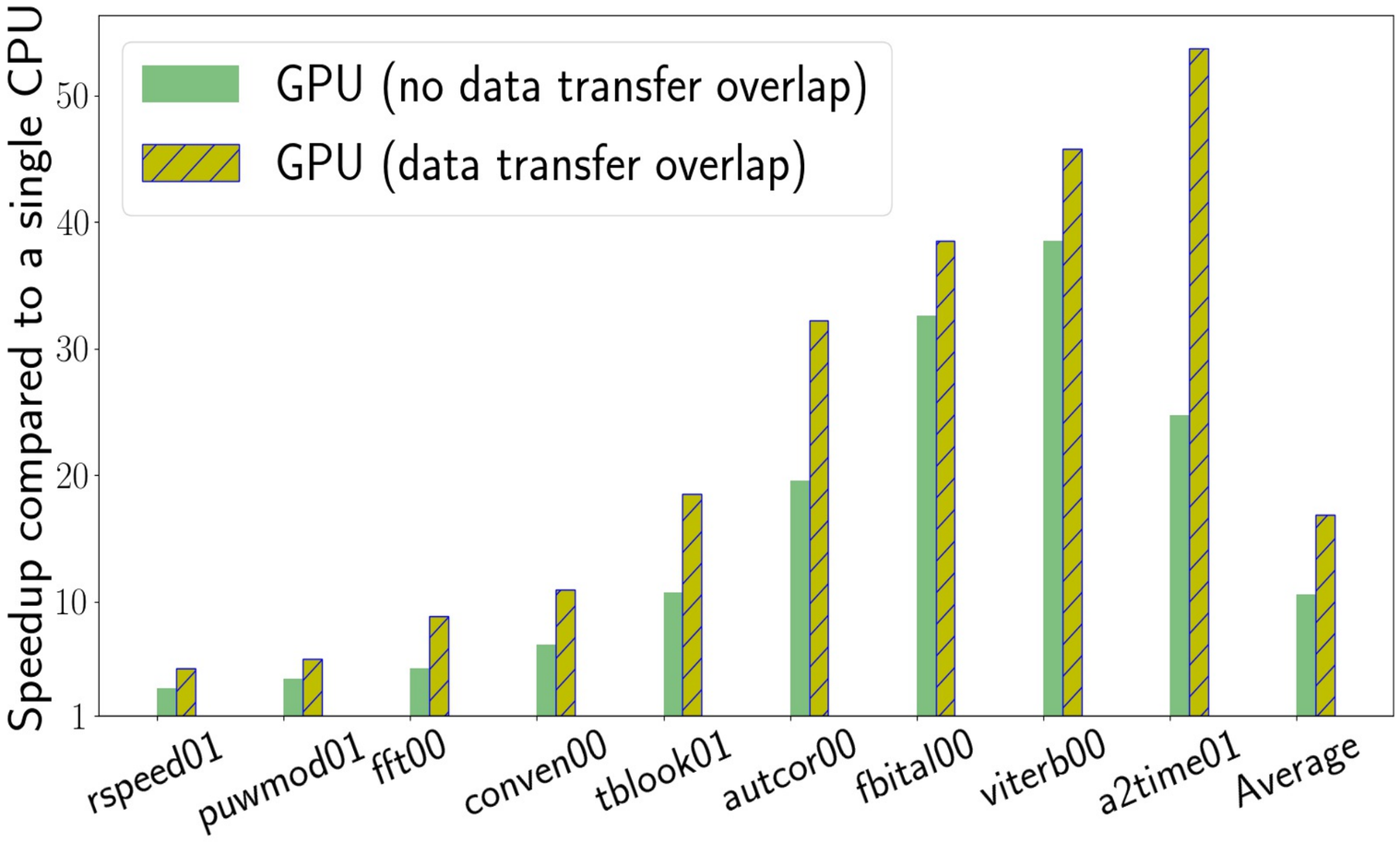




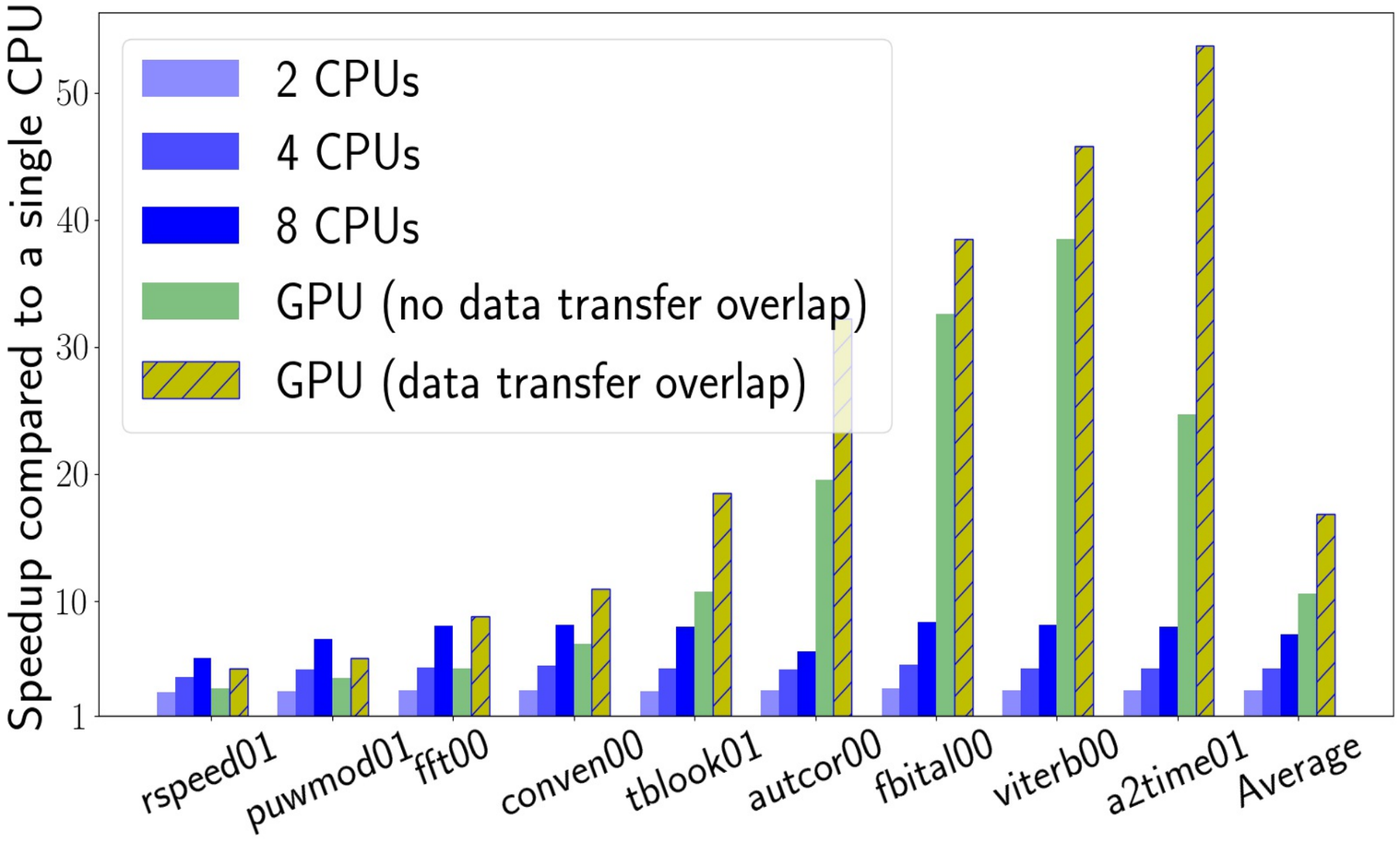




DATA TRANSFER OVERHEAD



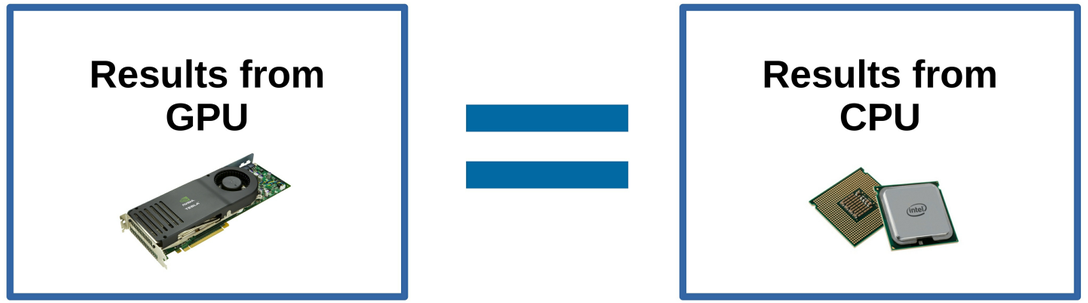
COMPARISON TO A MULTI-CORE CPU



CHALLENGES

Usability	✔
Scope	✔
Performance	✔

CORRECTNESS



For all 9 benchmarks, testing results from the GPU are **an exact match** to the testing results from the CPU.

SUMMARY

- Automatic GPU code generation
- Automatic test execution on the GPU threads
- Speedup of up to **53x** (avg 16x) on EEMBC benchmarks
- Correct testing results

SUMMARY

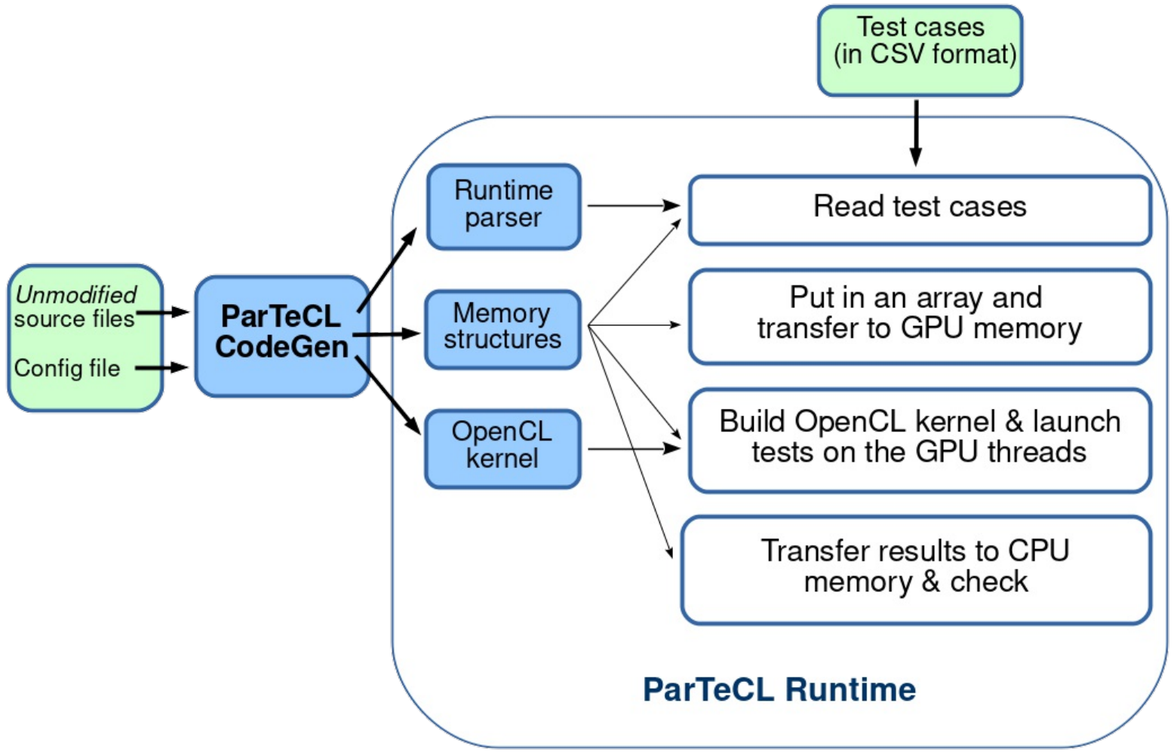
- Automatic GPU code generation
- Automatic test execution on the GPU threads
- Speedup of up to **53x** (avg 16x) on EEMBC benchmarks
- Correct testing results

FUTURE WORK

- Extend evaluation & scope
- Analyse & improve performance

THANKS

ParTeCL CodeGen	**github.com/wyaneva/partecl-codegen**
ParTeCL Runtime	**github.com/wyaneva/partecl-runtime**
clClibc	**github.com/wyaneva/clClibc**


Test cases
(in CSV format)
Unmodified
source files
Config file
ParTeCL
CodeGen
Runtime
parser
Memory
structures
OpenCL
kernel
Read test cases
Put in an array and
transfer to GPU memory
Build OpenCL kernel & launch
tests on the GPU threads
Transfer results to CPU
memory & check
ParTeCL Runtime

C FEATURES

- **Out of the box:**
  - pure functions, function calls, double precision (for OpenCL 1.2)

- **With transformations:**
  - standard in/out
  - global scope variables
  - standard library calls (partial support)

- **Unsupported (yet):**
  - dynamic memory allocation
  - file I/O
  - recursion