Compiler-Assisted Test Acceleration Using GPUs

Vanya Yaneva
Supervisor: Dr Ajitha Rajan
University of Edinburgh, UK
vanya.yaneva@ed.ac.uk

ABSTRACT

Software testing is a crucial part of the software development process, but is often extremely time consuming, expensive, manual and error prone. This has resulted in a crucial need for test automation and acceleration. We propose using GPUs for the acceleration of test execution, by running individual functional tests in parallel on the GPU threads. We provide a fully automatic framework, called ParTeCL, which generates GPU code from sequential programs and executes their tests in parallel on the GPU. Current evaluation on 9 programs from the EEMBC industry standard benchmark suite show that ParTeCL achieves an average speedup of 16× when compared to a single CPU for these benchmarks.

KEYWORDS

Functional testing, Automated testing, GPUs, Compilers

ACM Reference Format:
Vanya Yaneva. 2018. Compiler-Assisted Test Acceleration Using GPUs. In *ICSE '18 Companion: 40th International Conference on Software Engineering Companion, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3183440.3190337

1 RESEARCH PROBLEM & MOTIVATION

Rigorous testing of any non-trivial system involves the generation and execution of a huge number of individual test cases. This can often take hours, days or even weeks and puts an enormous pressure on the software development schedule. Standard practices like test-driven development and overnight test runs maintain the quality of the developed system, but rely on an exhaustive test suite and regular test executions. As a result, there is a crucial need to accelerate testing, allowing for both faster development and the production of higher quality software, as more tests can be executed at a time.

This problem has led to a significant interest in test automation both in academic literature and industry, resulting in advanced algorithms for test case generation, metrics for test suite coverage and methods for the acceleration of both test generation and execution.

Our research focuses on *the acceleration of test execution*, proposing the use of GPGPUs (General Purpose Computing on Graphics Processing Units) to execute tests in parallel, reducing total testing times. This idea is based on the intuition that in the majority of testing, test executions can be performed independently and thus, in parallel. Indeed, testing is often performed in parallel in industry, but the degree to which this can be done is limited by the cost of procuring and maintaining testing infrastructure. GPUs, on the other hand, provide massive degrees of parallelism, capable of executing thousands of test cases simultaneously. Originally designed for graphics processing, they have been the focus of research in various general purpose domains in recent years.

We currently target functional testing of *embedded software*. This is an important problem, as embedded systems are ubiquitous, featuring in consumer electronics and safety critical systems such as car sensors, breaking systems, medical monitoring devices and telecommunication systems. This makes safety concerns a top priority when developing and approving embedded software and functional testing is an crucial part of this process. What is more, embedded software exhibits few of the restrictions of GPGPUs, due to the restrictions of embedded hardware.

2 BACKGROUND & RELATED WORK

Functional software testing, also known as black-box testing, is the type of testing which verifies that the developed system behaves as intended. It involves the execution of the tested functionality with different inputs, which constitute the test cases, and checking that the results are as expected.

Using GPUs in software testing has gotten relatively little attention in literature. Existing approaches are mainly concerned with reducing the size of the test suite and/or prioritising test cases. For example, [10] proposes a parallel algorithm to accelerate test case *generation* using GPUs, while [3, 9] present parallel algorithms for test-suite minimisation and test case prioritisation. Both approaches aim to accelerate the testing process by focusing on the test suite - the first on its generation and the second on reducing its size.

In contrast, our work focuses on using GPUs for test *execution*, leaving the test suite unchanged. This idea is first proposed in [6] and is extended in our existing work [7, 8].

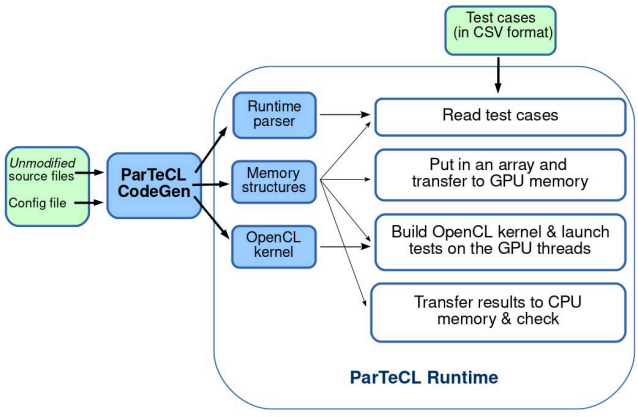

Figure 1: Design of ParTeCL

This work was supported by grant $EP/L01503X/1$ from EPSRC..

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05…$15.00
https://doi.org/10.1145/3183440.3190337

3 PARTECL: PARALLEL TESTING IN OPENCL

Programming GPUs requires the use of specialist low-level programming models, such as CUDA [4] and OpenCL[2], making executing tests on the GPU unavailable to the general programmer. To address this, we present **ParTeCL** - a compiler framework, which automatically generates OpenCL code from sequential C embedded applications and launches their tests on the GPU. It consists of two components, illustrated in Figure 1: (1) **ParTeCL CodeGen** is a tool to generate OpenCL code executable on the GPU and (2) **ParTeCL Runtime** is a system to execute the tests.

ParTeCL CodeGen wraps the embedded C code into an OpenCL kernel, abstracting away low level GPU details, making the approach accessible to all programmers, even those unfamiliar with OpenCL. What differentiates it from other code-generation tools targeted at GPUs is that it does not parallelise the application. Instead, it launches different instances of the sequential program in parallel, mapping the test cases to the GPU threads.

ParTeCL also allows the automatic transformations of program features typically unsupported on the GPU, improving the scope of the approach. Such features currently supported by ParTeCL are standard input and output, assignment to global scope variables and standard library calls. In addition, usability is further improved by ParTeCL Runtime, which launches test executions on the GPU, thus completely automating the testing process.

ParTeCL's implementation uses the Clang compiler's LibTooling library [1] to perform the code generation. The source code is hosted on https://github.com/wyaneva/ParTeCL-CodeGen and https://github.com/wyaneva/ParTeCL-Runtime. Thorough presentation of the tool's functionality and current limitations can be found in our existing papers [7, 8].

4 EXPERIMENTS & RESULTS

To check the feasibility and performance of our approach on C programs from the embedded systems domain, we perform empirical evaluation on 9 applications from the automotive and telecom domains of the EEMBC industry-standard benchmarks for embedded systems [5]. For each benchmark, we randomly generate 131,072 unique test inputs. We assess three aspects in our evaluation: speedup over single and multi-core CPU execution, overhead of using a GPU, and correctness.

We use an Intel(R) Xeon(R) CPU with 8 cores at 2.60 GHz and 16 GB RAM, compiling all programs with GCC using the highest optimization level (-O3). The GPU we use is the NVidia Tesla K40m with 15860 work items, spread across 15 compute units, operating at 745 MHz and has 12 GB global memory and 50 KB local memory.

Speedup. Figure 2 shows the speedup achieved by our approach when compared to a single CPU over increasing sizes of the test suite. As expected, since the GPU is able to utilise more threads as test cases are added, the speedup increases with the size of the test suite, reaching up to 37× when compared to a single GPU, for some benchmarks. The figure also shows that there is a large variety in the degree of speedup achieved by the different benchmarks, the reasons for which are discussed in [7].

Overhead. We measured the overhead of transferring test case inputs and results between CPU and GPU memory and implemented strategy for transferring test data in chunks, overlapping data transfer and test execution times. Figure 3 shows that this strategy improves the achieved speedup in all benchmarks, bringing the highest value from 37× to up to 53×. It also shows the average speedup achieved by our approach: it is 16× over single thread performance across all benchmarks, which is significantly better than an 8 core CPU which achieves an average speedup of 7×.

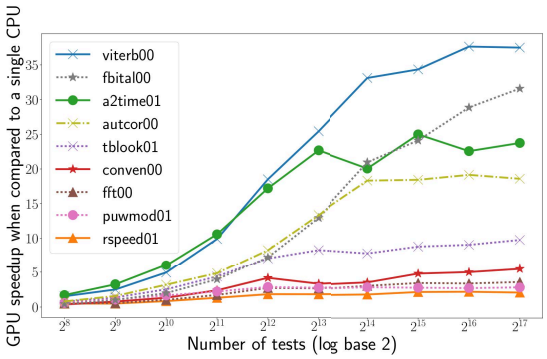

Figure 2: Speedup on the GPU vs single thread execution for 10 different test suite sizes.

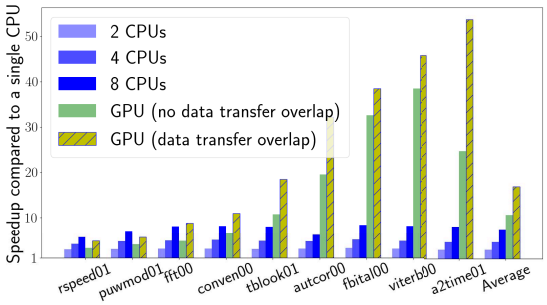

Figure 3: Speedup of GPU and multi-core CPUs over single CPU core.

Correctness. For each subject program, we collected the test case outputs from the CPU and GPU executions across all test suites. Each test suite was executed 100 times on the GPU and CPU. We found that for all 9 subject programs, with 131,072 test cases each, the test case outputs between the CPU and GPU executions were an **exact match**. We can safely conclude that our framework for executing tests on the GPU *preserves correctness of program execution* for all 9 embedded system benchmarks and test suites in our experiment.

Detailed discussion of all experiments and results can be found in our paper [7].

5 FUTURE WORK

There are multiple ways in which this work can be extended.

Extend evaluation. While our speedup results are very promising, the EEMBC benchmark applications are relatively small, executing within seconds. To better demonstrate the impact of our approach, we plan to extend our evaluation to larger and longer-running C programs.

Apply to other domains. We plan to extend our evaluation to domains outside of embedded systems, adding support in our tools for features such as dynamic memory allocation and recursion.

Build a classifier. As seen in our evaluation, some programs achieve faster testing on the GPU than others. As part of our related work, we plan to build a classifier for programs suitable for testing on the GPU, based on program and test features.

REFERENCES

[1] Clang. 2018. Clang 6 LibTooling Documentation. (2018). http://clang.llvm.org/docs/LibTooling.html

[2] The Khronos Group. 2018. OpenCL. (2018). https://www.khronos.org/opencl/

[3] Zheng Li, Yi Bian, Ruilian Zhao, and Jun Cheng. 2013. A Fine-Grained Parallel Multi-objective Test Case Prioritization on GPU. In *SSBSE*. Springer.

[4] NVidia. 2018. CUDA Programming Guide. (2018). http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[5] Jason A. Poovey, Thomas M. Conte, Markus Levy, and Shay Gal-On. 2009. A Benchmark Characterization of the EEMBC Benchmark Suite. *IEEE Micro* 29, 5 (2009), 18–29. https://doi.org/10.1109/MM.2009.74

[6] Ajitha Rajan, Subodh Sharma, Peter Schrammel, and Daniel Kroening. 2014. Accelerated test execution using GPUs. In *ACM/IEEE ASE'14*. 97–102.

[7] Vanya Yaneva, Ajitha Rajan, and Christophe Dubach. 2017. Compiler-assisted Test Acceleration on GPUs for Embedded Software. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2017)*. ACM, New York, NY, USA, 35–45. https://doi.org/10.1145/3092703.3092720

[8] Vanya Yaneva, Ajitha Rajan, and Christophe Dubach. 2017. ParTeCL: Parallel Testing Using OpenCL. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2017)*. ACM, New York, NY, USA, 384–387. https://doi.org/10.1145/3092703.3098227

[9] Shin Yoo, Mark Harman, and Shmuel Ur. 2011. Highly Scalable Multi Objective Test Suite Minimisation Using Graphics Cards. In *SSBSE*. Springer, 219–236. http://dl.acm.org/citation.cfm?id=2042243.2042271

[10] Zhao Yu, Jae-Han Cho, Byoung-Woo Oh, and Lee-Sub Lee. 2013. Parallel Algorithm for Generation of Test Recommended Path using CUDA. *International Journal of Engineering Science & Technology* 5, 2 (2013).