Bounding the WCET of a GPU Thread Block with a Multi-Phase Representation of Warps Execution (Artifact)

Louison Jeanmougin
Université de Toulouse, IRIT, CNRS, France

Thomas Carle
Université de Toulouse, IRIT, CNRS, France

Christine Rochange
Université de Toulouse, IRIT, CNRS, France

Abstract

This paper proposes to model the Worst-Case Execution Time (WCET) of a GPU thread block as the Worst-Case Response Time (WCRT) of the warps composing the block. Inspired by the WCRT analyzes for classical CPU tasks, the response time of a warp is modeled as its execution time in isolation added to an interference term that accounts for the execution of higher priority warps. We provide an algorithm to build a representation of the execution of each warp of a thread block that distinguishes phases of execution on the functional units and phases of idleness due to operations latency. A simple formula relying on this model is then proposed to safely upper bound the WCRT of warps scheduled under greedy policies such as Greedy-Then-Oldest (GTO) or Loose Round-Robin (LRR). We experimented our approach using simulations of kernels from a GPU benchmark suite on the Accel-Sim simulator. We also evaluated the model on a GPU program that is likely to be found in safety critical systems : SGEMM (Single-precision GEneral Matrix Multiplication). This work constitutes a promising first building block of an analysis pipeline for enabling static WCET computation on GPUs.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Computer systems organization → Embedded systems

Keywords and phrases GPU, WCET analysis

Digital Object Identifier 10.4230/DARTS.11.1.3

Funding *Louison Jeanmougin*: This work was supported by a grant overseen by the French National Research Agency (ANR) as part of the CAOTIC ANR-22-CE25-0011 project.
Thomas Carle: This work was supported by a grant overseen by the French National Research Agency (ANR) as part of the MeSCAliNe ANR-21-CE25-0012 project. Our work has also benefited from the AI Interdisciplinary Institute ANITI under the Grant agreement n°ANR-23-IACL-0002.

Related Article Louison Jeanmougin, Thomas Carle, and Christine Rochange, "Bounding the WCET of a GPU Thread Block with a Multi-Phase Representation of Warps Execution", in 37th Euromicro Conference on Real-Time Systems (ECRTS 2025), LIPIcs, Vol. 335, pp. 11:1–11:26, 2025.
`https://doi.org/10.4230/LIPIcs.ECRTS.2025.11`

Related Conference 37th Euromicro Conference on Real-Time Systems (ECRTS 2025), July 8–11, 2025, Brussels, Belgium

1 Scope

The artifact allows to reproduce the results of all experiments presented in the article. Namely, one can expect to obtain the content of Tables 2, 3 and 4. Table 2 presents the impact of the assumptions through an ablation study. Table 3 shows the overestimation of our model compared

to the effective execution time of kernels from the Rodinia Benchmark. Table 4 lists the results given by the model when tested against a GPU Kernel that is likely to be found in Real-Time Systems.

2 Content

The artifact package includes:

- A folder comprising all necessary to reproduce Tables 2 an 3 (accel-sim-framework_model).
- A folder with everything needed to reproduce Table 4 (accel-sim-framework_ablation-study).
- The results of the Rodinia benchmark simulations for the model (pre-run_simulations).
- The results of the SGEMM programs simulation (sgemm).
- The results of the Rodinia simulations for the ablation study (ablation-study_simulations).

Note that all our experiments make use of the Accel-Sim GPU simulator [1] to generate execution traces.

3 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS).

3.1 Before you start

All the files needed to reproduce the experiments are contained in the downloadable artifact. We will use **Python 3.11** as well as **pip3** so ensure that they are installed. The experiments also require the use of a container so you will either need **docker** or **podman** installed.
If you decide to make use of a Virtual Machine, we recommend you set it up with at least 60GB of disk space.

3.2 Deriving the WCET of a thread block based on a single path CFG

Our model allows to find a safe upper-bound of the WCET For a GPU thread block using a single path CFG as an input. We use a GPU simulator in order to obtain the path taken by the program for each warp of the block, then perform our analysis on the obtained traces before comparing them to the output simulation duration. This comparison allows us to assess the precision of our model.

3.3 Reproducing the Rodinia benchmark analysis for a full block of threads

For the first part, uncompress the artifact file and place yourself in the "accel-sim-framework_model" folder:

```
$ cd accel-sim-framework_model
```

In order to have the simulator up and working without too much effort, we will use docker. Simply follow the upcoming steps and you will be all set :

```
$ sudo docker pull accelsim/ubuntu-18.04_cuda-11
$ sudo docker run -v $(pwd):/home/runner/accel-sim -i -t 924b9731d807 /bin/bash
```

From now on, there will be commands to run from the docker you just started and other commands you will have to run from another terminal. The difference between the two will be specified.
In docker:

```
$ chmod +x full_simulations.sh
$ ./full_simulations.sh
```

The *full_simulation.sh* script will:

- Install the libraries required
- Build the rodinia benchmark
- Build the simulator
- Run the simulations

The simulation execution can be tracked running the "top" command in the docker. It can happen that the simulations do not start. This is noticeable in "top" as no kernel names are on display and only a bunch of python3 processes are present. If that happens, clean the content of "accel-sim-framework/util/job_launching/procman" and "accel-sim-framework/util/job_launching/logfiles" with:

```
$ rm -rf util/job_launching/procman
$ rm -rf util/job_launching/logfiles
```

Running the simulations takes about an hour on an i7-1370P. If you are not willing to wait that time, you can copy the content of the "pre-run_simulations" folder inside the "accel-sim-framework_model" folder. After all the simulations are done you can find the trace dump for each kernel instance that has been executed on the simulator. These traces are located under "accel-sim-framework/*config*/*bench_name*/*data_src_name*/*config_name*/*instance_name*.trace". When opening one you can observe a series of PTX instructions (PTX being the Nvidia virtual assembly) prefixed with the warp that issued the instruction.

Now we will use those thread block execution traces to generate a trace for each warp. In a new terminal, go to the "accel-sim-framework_model/TraceParser" and run the analysis:

```
$ cd accel-sim-framework_model
$ pip3 install -r requirements.txt
$ cd accel-sim-framework_model/TraceParser
$ chmod +x block_stats.sh
$ ./block_stats.sh
```

This will give you the results for GTO and LRR from **Table 3** in the paper. What it did was parsing the execution traces given by the simulation and splitting them into files representing the CFG path taken by each warp. Then our model produced execution profile for each warp before combining them, giving us an upper-bound for the WCET of the thread block. Note that there is a slight difference between the results in the article and the results you will obtain. We decided to remove one of the rodinia programs that was prohibitively long to simulate. There is, however, no significant difference between the experimental from full simulation and the quicker one.

3.4 Reproducing the Rodinia benchmark analysis for a single warp

The last section should also have produced the simulation results for a single warp in isolation. Note that since this experiment is on a single warp, the warp scheduler does not matter. This is why we do not need to test on both GTO and LRR. Getting the results from **Table 3** for a single warp is then obtained via terminal outside of the docker:

```
$ cd accel-sim-framework_model/TraceParser
$ chmod +x 1w_stats.sh
$ ./1w_stats.sh
```

3.4.1 Reproducing the SGEMM analysis

We extended the evaluation of our model with the case study of two SGEMM (Single-precision GEneral Matrix Multiplication) implementations. The SGEMM algorithm is likely to be used in machine learning applications such as computer vision or autonomous driving. Two implementations of SGEMM were implemented and tested with our model. A naïve one that doesn't take advantage of the latency masking GPUs generally exploit, and an implementation of a double buffering technique allowing the algorithm to issue arithmetic instructions on data already present in shared memory while waiting for the next data to be available. All the versions were simulated with all the configurations and with 4, 8, 16 as well as 32 warps. There is no need to test it above 32 warps since a thread block can't exceed 1024 threads (32 warps x 32 threads = 1024). To run the simulation, inside the docker, run the script :

```
$ ./sgemm_exp.sh
```

This should compile all the sgemm programs and run them. We provide the simulations results for this experiment as well so you can just copy the "sgemm" folder into "accel-sim-framework_model" if you want. You can then obtain the content of **Table 4** from the paper by following the next steps in a new terminal:

```
$ cd accel-sim-framework_model/TraceParser
$ chmod +x sgemm_stats.sh
$ ./sgemm_stats.sh
```

3.5 Ablation study on the baseline simulator

Determining the WCET for a block of threads is a fairly complex task and has to be built step by step. As our work does not include cache or memory analysis, nor the dual issue feature of GPUs, some modifications have been made on the baseline GPU simulator. All modifications are presented in **Section 4.1** of our paper under **Modifications**.

3.5.1 Reproduce the results from the ablation study

The results from **Table 2** are obtained following the upcoming steps. Start a new docker in "accel-sim-framework_ablation-study" folder:

```
$ cd accel-sim-framework_ablation-study
$ sudo docker pull accelsim/ubuntu-18.04_cuda-11
$ sudo docker run -v $(pwd):/home/runner/accel-sim -i -t 924b9731d807 /bin/bash
```

Then run :

```
$ ./ablation_exp.sh
```

Note that the results of the simulations can be found in the "abaltion-study_simulations" folder so you can copy them in the "accel-sim-framework_ablation-study/sim_results" folder if you're not willing to run them. When simulations are over do the following in a new terminal:

```
$ cd accel-sim-framework_ablation-study/sim_results
$ pip3 install -r requirements.txt
$ python3.11 results_parser.py .
```

This will give you the results from **Table 2** present in the paper.

4 Tested platforms

This artifact has been tested on an Ubuntu 22.04 distribution running on an i7-1370P.

5 License

The Accel-Sim GPU simulator is available under BSD 2-Clause license.
The rest of the artifact is available under the CeCILL Free Software License Agreement v1.0.

6 MD5 sum of the artifact

83f8cddda1e1ee16c87fb381700f516d

7 Size of the artifact

13.7 GiB

References

1 Mahmoud Khairy, Zhesheng Shen, Tor M. Aamodt, and Timothy G. Rogers. Accel-sim: An extensible simulation framework for validated GPU modeling. In *47th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2020, Virtual Event / Valencia, Spain, May 30 - June 3, 2020*, pages 473–486. IEEE, 2020. doi:10.1109/ISCA45697.2020.00047.