Predictable GPU Sharing in Component-Based Real-Time Systems (Artifact)

Syed W. Ali
Department of Computer Science, University of North Carolina at Chapel Hill, NC, USA

Zelin Tong
Department of Computer Science, University of North Carolina at Chapel Hill, NC, USA

Joseph Goh
Department of Computer Science, University of North Carolina at Chapel Hill, NC, USA

James H. Anderson
Department of Computer Science, University of North Carolina at Chapel Hill, NC, USA

Abstract

This paper presents a real-time locking protocol whose design was motivated by the goal of enabling safe GPU sharing in time-sliced component-based systems. This locking protocol enables a GPU to be shared concurrently across, and utilized within, isolated components with predictable execution times. It relies on a novel resizing technique where GPU work is dimensioned on-the-fly to run on partitions of an NVIDIA GPU. This technique can be applied to any component that internally utilizes global CPU scheduling. The proposed locking protocol enables increased GPU parallelism and reduces GPU capacity loss with analytically provable benefits.

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

Keywords and phrases GPU locking protocols, real-time locking protocols, priority-inversion blocking, component-based systems

Digital Object Identifier 10.4230/DARTS.10.1.1

Funding Supported by NSF grants CPS 2038960, CPS 2038855, CNS 2151829, and CPS 2333120.

Related Article Syed W. Ali, Zelin Tong, Joseph Goh, and James H. Anderson, "Predictable GPU Sharing in Component-Based Real-Time Systems", in 36th Euromicro Conference on Real-Time Systems (ECRTS 2024), LIPIcs, Vol. 298, pp. 15:1–15:22, 2024. `https://doi.org/10.4230/LIPIcs.ECRTS.2024.15`

Related Conference 36th Euromicro Conference on Real-Time Systems (ECRTS 2024), July 9–12, 2024, Lille, France

1 Scope

There are two primary claims in the corresponding paper that are backed by this artifact. First, the worst-observed pi-blocking duration for the Streaming Multiprocessor Locking Protocol (SMLP) is up to 50% lower than the worst-observed pi-blocking duration for the coarse-grain OMLP [2]. Second, GPU kernels do not need to utilize every compute unit in a GPU, thus enabling the improved parallelism of the SMLP. The first claim is backed by a simulation study. The second claim is supported by testing a simple GPU kernel and reporting completion times when pinned to a varying number of compute units.

2 Content

The artifact package includes:

- Experiment 1: Simulation code
- Experiment 2: NVIDIA GPU partition testing code

- Pre-built simulation results for Experiment 1
- Graph generation tools for Experiment 1

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). In addition, the artifact is also available at: `https://www.cs.unc.edu/~swali/ecrts24/`.

4 Tested platforms

Experiment 1

Experiment 1 requires a machine with many processors as it can run multiple simulations simultaneously on separate threads. Experiment 1 was performed on the below machine:

CPU:	Intel i9-13900k (32 logical processors)
RAM:	32GB, DDR5 6000MHz
OS:	Windows 11, Ubuntu Linux 23.10

Compiling instructions and optimizations are detailed in Sec. A.

Experiment 2

Experiment 2 requires a machine with an NVIDIA GPU that is compatible with `libsmctrl` [1]. Experiment 2 was performed on:

CPU:	Intel i7-12700k
RAM:	128GB, DDR4 3600MHz
OS:	Ubuntu Linux 23.10
GPU:	NVIDIA GeForce RTX 4080

Experiment 2 was also confirmed to work on an i7-3770k CPU, 16GB RAM, Ubuntu 23.10, and a 3060 Ti. Additionally, this experiment requires a version of CUDA compatible with `libsmctrl` [1]. Details on how to build and run the experiments are given in Sec. B.

5 License

The artifact is available under the GPLv3 license.

6 MD5 sum of the artifact

59602553fb34de203b253b31a96c9cf4

7 Size of the artifact

11.2 MiB

A Experiment 1

Either extract from the 7z archive or clone the GitHub repository at `https://github.com/swali-unc/SMLP-ECRTS24`. If you want to compile this experiment, please see the instructions on `https://www.cs.unc.edu/~swali/ecrts24/`, also briefly shown in Sec. A.5, A.6. A Windows pre-built binary is in the `bin` folder of the archive and is also available as a release on GitHub. All Experiment 1 files are in the `SMLP-ECRTS24` folder.

This experiment has three modes of operation. (i) A task set can be generated and analyzed in a CSV file. (ii) A task set can be used as input for a single simulation instance. (iii) The full simulation is run multiple times using a range of values. If you run the binary without any parameters, the accepted parameters are listed. Detailed below are the command-line parameters.

A.1 Generating Task Sets

Table 1 Command-line Parameters: Task Generation.

`-t`	Generate and output a task set
`-o filename`	Output CSV filename of the task set
`-m corecount`	Number of processors in the component
`-tmin minPeriod`	Minimum period of each task
`-tmax maxPeriod`	Maximum period of each task
`-n numTasks`	The max number of tasks in a task set
`-u targetUtil`	The target normalized utilization

This will generate a task set and store the data in a CSV file. A task is generated using the techniques described in [3]. The generated task set can then be used in a single simulation instance described below.

A.2 Single Simulation

Table 2 Command-line Parameters: Single Simulation.

`-s`	Run a single simulation instance
`-i filename`	Input CSV filename of the task set
`-o filename`	Output filename for observed results
`-Hmin hmin`	Lower bound of tested $\mathcal{H}$ values
`-Hmax hmax`	Upper bound of tested $\mathcal{H}$ values
`-Hstep hstep`	Value by which $\mathcal{H}$ changes every scenario
`-Tmin thetaMin`	Lower bound of Θ
`-Tmax thetaMax`	Upper bound of Θ
`-Tstep thetaStep`	Value by which Θ changes every scenario
`-p prob`	Probability a task is selected to issue a request
`-m corecount`	Number of processors in the component

The above will take an already generated task set (Sec. A.1) as input. It then simulates the system and outputs the observed data to the specified output file.

A.3 Full Simulation

The results presented in the paper were conducted using the following values:

```
-a -Mmin 4 -Mmax 16 -Mstep 4 -tminset 3 10 50 -tmaxset 33 100 200 -n 150
-taskSets 1000 -utilMin 0.2 -utilMax 0.9 -utilStep 0.1 -Hmin 8 -Hmax 64
-Hstep 8 -Tmin 1.5 -Tmax 3 -Tstep 0.5 -p 0.5 -o p0.5.csv
```

The above parameters will ensure up to 40 simulation threads are running concurrently. This is due to there being 4 Theta values and `PARALLEL_SETS=5`. With both the OMLP and SMLP tested, this results in up to 40 concurrent simulation threads. This can be modified to better match the parallelism capacity of your system by tuning `PARALLEL_SETS` or generating data over a smaller range of Θ.

Table 3 Command-line Parameters: Full Simulation.

`-a`	Run multiple simulations
`-o filename`	Output filename for observed results
`-Mmin corecount`	Lower bound of tested $\mathcal{M}$ values
`-Mmax corecount`	Upper bound of tested $\mathcal{M}$ values
`-Mstep corecount`	Value by which $\mathcal{M}$ changes every scenario
`-tminset tmin1, tmin2, ...`	Set of minimum periods
`-tmaxset tmax1, tmax2, ...`	Set of maximum periods
`-n numTasks`	The max number of tasks in a task set
`-taskSets numSets`	The number of task sets to generate
`-utilMin minUtil`	Lower bound of normalized utilization values
`-utilMax maxUtil`	Upper bound of normalized utilization values
`-utilStep ustep`	Value by which utilization changes every scenario
`-Hmin hmin`	Lower bound of tested $\mathcal{H}$ values
`-Hmax hmax`	Upper bound of tested $\mathcal{H}$ values
`-Hstep hstep`	Value by which $\mathcal{H}$ changes every scenario
`-Tmin thetaMin`	Lower bound of Θ
`-Tmax thetaMax`	Upper bound of Θ
`-Tstep thetaStep`	Value by which Θ changes every scenario
`-p prob`	Probability a task is selected to issue a request

A.4 Validating Results

The CSV files can take a while to generate. You can also use pre-generated data available here: `https://www.cs.unc.edu/~swali/ecrts24/p0.5.7z` (also available in the `prebuilt` folder in the archive). The CSV files can also be very large. For example, when fully simulating with `-n 10000`, the CSV file reached 3.8 GiB. As such, most CSV viewers are unable to render or process the data. Instead, we provide a Jupyter notebook here: `https://www.cs.unc.edu/~swali/ecrts24/smlp-eval.7z` (also in the `prebuilt` folder if using the archive).

1. Open Jupyter Notebook `https://jupyter.org/`.
2. Create a folder for this project.
3. Extract the notebook contents into this folder.
4. Ensure your CSV results are in this folder.
5. Open and run `smlp_eval.ipynb`.
6. (Optional) Modify the code to generate graphs for other data sets.

A.5 Compiling – Windows

To compile on Windows, we suggest using Microsoft Visual Studios 2022 or newer (`https://visualstudio.microsoft.com/vs/`).

1. Clone the GitHub repo or extract the 7z archive.
2. Open the `.sln` file in Visual Studios.
3. Ensure the build configuration is set to Release.
4. Compile.

A.6 Compiling – Linux

To compile on Linux, we tested using `g++14` specifically, and it should work on later versions.

1. Clone the GitHub repo or extract the 7z archive.
2. Ensure `g++` is installed (Version 14 or newer).
3. Compile: `g++ -o smlpsim main.cpp util.cpp task_gen.cpp gedf_multisim.cpp gedf_sim.cpp gedf_sim_thread.cpp`

B Experiment 2

Either extract from the 7z archive or clone the GitHub repository at `https://github.com/swali-unc/SMLP-ECRTS24`. All Experiment 2 files are in the `libsmctrlTest` folder. This experiment requires access to an NVIDIA GPU capable of using `libsmctrl` [1]. Additionally, as of this writing, `libsmctrl` only works on CUDA versions from 8.1 to 12.2. This experiment was specifically run on CUDA 12.0.r12 on Ubuntu 22.04. Note, there is currently no Windows version of `libsmctrl`.

First, get the latest version of `libsmctrl` [1] from `http://rtsrv.cs.unc.edu/cgit/cgit.cgi/libsmctrl.git/about/`. Compile using `make`, then store the generated shared object (`.so`) file ideally in a location referenced by `LD_LIBRARY_PATH`.

In the artifact files, run `make` in the folder with the Makefile. Note that you may have to modify the Makefile to point to where `libsmctrl` is located by modifying the `LDFLAGS` parameter.

This program only generates one pass of results. It can be easily modified to generate multiple runs and take the worst-observed values. Adding some delay to allow the GPU to "cool off" is advisable to prevent unnecessary damage.

B.1 Validating Results

This experiment, by default, assumes your GPU has 19 partitionable TPCs (the 3060 Ti has 19 TPCs). This can be modified by first changing `TOTAL_TPCs` in `main.cu`, then recompiling. The goal is to see kernel execution durations that exhibit a step-graph behavior, thus showing that kernels do not always need all of the full compute capacity of a GPU.

References

1 Joshua Bakita and James H. Anderson. Hardware compute partitioning on NVIDIA GPUs*. In *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 54–66, 2023. doi:10.1109/RTAS58335.2023.00012.

2 Bjorn B. Brandenburg and James H. Anderson. Optimality results for multiprocessor real-time locking. In *2010 31st IEEE Real-Time Systems Symposium*, pages 49–60, 2010. doi:10.1109/RTSS.2010.17.

3 P. Emberson, R. Stafford, and R.I. Davis. Techniques for the synthesis of multiprocessor tasksets. *WATERS'10*, January 2010.