Euro-Par 2022 Artifact - Overview document

This code, inputs, and scripts are a companion to the paper entitled "*Exploring scheduling algorithms for parallel task graphs: a modern game engine case study*" by M. Regragui, B. Coye, L. L. Pilla, R. Namyst, and D. Barthou, accepted for publication in Euro-Par 2022.

1. Getting Started Guide

In order to compile and reproduce the results of the article, you may execute `run_experiments.sh`. It will compile the code using `make` and `g++`, and then run the experiments with five different input files in order. In a conventional computer, the whole process takes about 3 hours and 30 minutes.

Result analysis can be directly reproduced using the script `run_analysis.sh` in directory `analysis_scripts/`, which calls three different Python 3 scripts. A series of Jupyter Notebook files are also available to help visualize the results. A list of required Python 3 modules is provided later in this document. In order to install them automatically, you can run `pip3 install -r requirements.txt` in `analysis_scripts/` (this happens automatically when `run_analysis.sh` is executed).

If you have any issues compiling the simulator, please check that you have g++ on your platform and that its version is at least 9.3.0. If you have any error messages when analyzing the results, please check if your Python 3 version is at least 3.8.10. Older versions may not be compatible with some module versions, which end up not being installed by pip and result in error messages.

If one does not want to rerun the experiments, a copy of the generated dataset is available on Zenodo (https://zenodo.org/record/6532252). This dataset can be used to validate the results achieved by the simulator or to avoid running the whole set of experiments (e.g., validating the results for only one of the input files).

1.1. Simple list of commands to reproduce the experiments

```
./run_experiments.sh
```

```
cd analysis_scripts
```

```
./run_analysis.sh
```

2. Step-by-Step Instructions

Reproducing these experiments takes about 3 hours and 30 minutes in a conventional computer.

2.1. Input format

Our simulator expects a text file containing information regarding the simulation parameters (e.g., `input_scenario_1.txt`). They are organized as follows:

- A line with `_nbFrame` is followed by the number of frames to simulate.
- A line with `_nbWorkers` is followed by multiple lines containing the number of resources to be used in the simulation. A line with `0` finishes the list.
- A line with `_nbStartSeed` is followed by the first RNG seed to be used. A line with `_nbEndSeed` provides the value after the last RNG seed to be used. For instance, if values `1` and `51` are used, RNG seeds from 1 to 50 will be employed by the simulator.
- A line with `_IsDivided` is followed by a value of `0` if we simulate Scenarios 1 and 2. A value of `1` is used for Scenario 3.
- A line with `_SortingSteps` is followed by a value of `0` for Scenario 1 and by `1` for Scenarios 2 and 3.
- A line with `Methods` is followed by a list of scheduling strategy names to be used in the simulation.

2.2. Expected outputs

The execution of `run_experiments.sh` will generate five different directories:

1. `Result_1/` contains the results for Scenario 1 generated using file `input_scenario_1.txt`.
2. `Result_2/` contains the results for Scenario 2 generated using file `input_scenario_2.txt`.
3. `Result_3/` contains the results for Scenario 3 generated using file `input_scenario_3.txt`.
4. `Result_CP_1/` contains the results for the critical path of Scenarios 1 and 2 generated using file `input_CP_scenario_1.txt`.
5. `Result_CP_3/` contains the results for the critical path of Scenario 3 generated using file `input_CP_scenario_3.txt`.

One may also choose to run simulations individually. After compiling the code with the command `make`, you can run the simulator followed by a directory renaming to keep the same names expected by the analysis scripts. For instance, to generate the results for Scenario 1, one may execute the two following commands:

1. `./engine_simulator input_scenario_1.txt`
2. `mv Result/ Result_1/`

Each result file (e.g., `HLF_NonSorted_Random_1_200_10.txt`) contains 200 lines representing information of the 200 frames that were simulated. Each line contains four values: the frame number, the duration of the frame (in microseconds), a critical path estimation for the previous frame (in microseconds), and the load parameter (value between 0 and 1).

For a simple and incomplete visual verification, you can run the command `head`

`Result_1/FIFO/12/200/TXT/FIFO_NonSorted_Random_1_200_12.txt` to get the results for the first 10 frames simulated for FIFO on Scenario 1, 12 resources, and RNG seed 1. The expected output is:

```
0 8445.85 0 0

1 7200.85 7962.45 0.01

2 8277.29 6639.1 0.02

3 7273.17 6520.63 0.03

4 7297.32 6596.38 0.04

5 8285.28 5964.44 0.05

6 6935.89 7347.61 0.06

7 8896.13 6449.06 0.07

8 7925.11 7796.88 0.08

9 7540.43 6383.42 0.09
```

A more detailed verification can be done using the dataset available on https://zenodo.org/record/6532252.

2.2.1 Relation between outputs and information in the article

The outputs of the analysis scripts include some PDF files representing the figures in the accepted article and some CSV files representing the values shown in tables. The CSV columns are split using ampersands to make it easier to reuse the results in LaTeX tables. Here is the mapping of structures in the article to files:

- Fig. 2(a): `results-1-sf.pdf`
- Fig. 2(b): `results-1-df.pdf`
- Fig. 2(c): `results-1-cs.pdf`
- Fig. 3(a): `results-2-lpt.pdf`
- Fig. 3(b): `results-2-wt.pdf`
- Fig. 3(c): `results-2-cg.pdf`
- Fig. 3(d): `results-2-dcp.pdf`
- Fig. 4: `results-3-resources-df.pdf`
- Fig. 5: `results-4-frame-duration.pdf`
- Fig. 6(a): `results-5-resources-sf.pdf`
- Fig. 6(b): `results-5-resources-df.pdf`
- Fig. 6(c): `results-5-resources-cs.pdf`
- Fig. 7: `results-7-frame-duration.pdf`
- Table 2: `table_2___average_metrics_12_resources.csv`
- Table 3 average metrics: `table_3___average_metrics_12_resources.csv`
- Table 3 % change: `table_3___diff_metrics_12_resources.csv`
- Table 4 average metrics: `table_4___average_metrics_12_resources.csv`

- Table 4 % change: `table_4___diff_metrics_12_resources.csv`

The standard output shows the p-values computed in parts of the statistical analysis.

2.3. Software and hardware information

The simulation results were generated on an Intel Core i7-1185G7 processor, with 32 GB of LPDDR4 RAM (3200 MHz). The machine ran on Ubuntu 20.04.3 LTS (5.14.0-1034-oem). GNU Make 4.2.1, GNU bash 5.0.17(1)-release, and g++ 9.4.0 were used for the simulator's compilation (-O3 flag) and execution. The compiled file `engine_simulator` references the following libraries:

- linux-vdso.so.1
- libstdc++.so.6
- libm.so.6
- libgcc_s.so.1
- libc.so.6
- /lib64/ld-linux-x86-64.so.2

The results were analyzed using Python 3.8.10, pip 20.0.2 and jupyter-notebook 6.0.3. The following modules and their respective versions were used:

- pandas 1.3.2
- numpy 1.21.2
- matplotlib 3.4.3
- seaborn 0.11.2
- scipy 1.7.1
- pytz 2019.3
- python-dateutil 2.7.3
- kiwisolver 1.3.2
- pyparsing 2.4.7
- cycler 0.10.0
- Pillow 7.0.0
- six 1.14.0

2.4. Simulation information

Simulation results were generated from 4 to 20 resources. Each configuration was run with 50 different RNG seeds (1 up to 50).

Each simulation is composed of 200 frames. The load parameter (lag) starts at zero and increases by 0.01 with each frame up to a value equal to 100% in frame 101. After that, the load parameter starts to decrease in the same rhythm down to 0.01 in frame 200.

2.5. Algorithms abbreviation in presentation order

FIFO serves as the baseline for comparisons.

1. **FIFO:** First In First Out.
2. **LPT:** Longest Processing Time First.
3. **SPT:** Shortest Processing Time First.
4. **SLPT:** LPT at a subtask level (input method name: SLRT).
5. **SSPT:** SPT at a subtask level (input method name: SSRT).
6. **HRRN:** Highest Response Ratio Next.
7. **WT:** Longest Waiting Time First.
8. **HLF:** Hu's Level First with unitary processing time of each task
9. **HLFET:** HLF with estimated times (input method name: Hu).
10. **CG:** Coffman-Graham's Algorithm (input method name: Coffman).
11. **DCP:** Dynamic Critical Path Priority (input method name: Priority).

2.6. Metrics

- SF: slowest frame (maximum frame execution time)
- DF: number of delayed frames (with 16.667 ms as the due date)
- CS: cumulative slowdown (with 16.667 ms as the due date)