Artifacts of "End-to-end Scheduling of Real-time Task Pipelines on Multiprocessors"

Constraint Solvers

1. `CoPi`: Our heuristic constraint solver algorithm.
2. `GEKKO`: The APOPT Solver: https://machinelearning.byu.edu/

Most experiments compare the above two. We have also experimented with two other MINLP solvers but only for the first experiment:

1. `pyomo`: http://www.pyomo.org/
2. `scipy`: https://docs.scipy.org/doc/scipy/reference/optimize.html

Requirements

Please follow the installation procedure of `GEKKO`, `pyomo`, `scipy`, `pickle` (for reading and writing raw dataset). We have tested the code should work for python version 3.6. If not sure, please use `virtualenv` to install python 3.6 .

Algorithms and Files

The CoPi algorithms are at the root location of the source directory. The MINLP solver implementations are in `ilp/` directory.

CoPi files are: 1. `copi_e2e.py`: For only E2E Delay Constraint implementation. 2. `copi_all.py`: For all the constraints. 3. `copi_lib.py`: Contains the CoPi algorithm that solves both constraints. 4. `multi_pipeline.py`: For the multiprocessor experiments with CoPi.

MINLP Solvers: 1. `ilp/ilp_gekko.py`: `GEKKO` constraint optimization solution. 2. `ilp/ilp_pyomo.py`: `pyomo` constraint optimization solution. 3. `ilp/ilp_scipy.py`: `scipy` constraint optimization solution.

Datasets Random pipelines are generated using UUnifast algorithm. All the dataset files are `pickle` files. 1. `dataset_*` files are for uniprocessor and runtime experiments. 2. `pipelines_*` files are for multiprocessor experiments. 3. `watersdata_*` files are for WATERS 2015 Workshop paper dataset.

Helper function files: 1. `utility.py`: Some helper functions for utilization, RMS bound and other calculations. 2. `pipeline.py`: Most functions related to a pipeline are here. For example, end-to-end delay, loss-rate, etc.

Figures The paper figures are provided in the `Figures` directory.

All the figures could be generated as PDF files by:

```
cd Figures
make
```

Experiments

The next subsections correspond to the subsections in the Evaluation section of the paper.

Uniprocessor Acceptance Ratio Experiments

Only for End-to-end Constraint

1. Run all the experiments for `CoPi`:

```
./run_exp1_copi.sh
```

2. Run all the experiments for `GEKKO (APOPT)`:

```
./run_exp1_gekko.sh
```

3. Run all the experiments for `pyomo (IPOPT)`:

```
./run_exp1_pyomo.sh
```

4. Run all the experiments for `scipy (trust-constr)`:

```
./run_exp1_scipy.sh
```

Result All the data will be written to `accepted_sets_<solver>.txt` in space separated format for $LBG = \{11, 12, 14, 15, 16, 18\}$. For example, `accepted_sets_copi.txt` is for `CoPi`.

Paper results are in `Figures/accept.csv`.

1st line is Latency Budget Hap (LBG). 2nd line is the total number of tasksets for each LBG.

3rd, 5th, 6th, 7th lines are respectively the accepted number of tasksets for CoPi, GEKKO, scipy and pyomo. (Ignore 4th line; it is not used anymore.) Match each line with the corresponding results in `accepted_sets_<solver>.txt` file.

In the paper, Figure 9a is generated by `python3 accept_models.py` with the above result.

The results may vary a bit depending on the changes in the random dataset, but the main trend in the result should remains same as the result in the paper and in `Figures` folder.

Both End-to-end Delay and Loss-rate Constraints

1. Generate CoPi data points by the following script:

```
./run_loss_rate_copi.sh
```

The `accepted_lr_copi.txt` will have the number of accepted (schedulable) pipelines in space separated format for {0, 25, 50, 75}% loss-rates.

2. Generate GEKKO data points by the following script (result will be appended to `accepted_sets_gekko.txt`, so if you can remove the file if you do not want to see previous results):

```
./run_gekko_lr.sh
```

All the data will be appended to `accepted_sets_gekko.txt`.

3. Generate GEKKO (with BAC) data points by the following script:

```
./run_gekko_lr_bac.sh
```

All the data will be appended to `accepted_sets_gekko.txt`.

Paper results are in `Figures/accept_models_lr.csv`.

First two lines are same as the previous result.

1st line is Latency Budget Gap (LBG).

2nd line is the total number of tasksets for each LBG.

3rd, 4th, 5th lines are respectively: CoPi, GEKKO, GEKKO (with BAC). Match these paper results with the results retrieved from running the above shell scripts.

In the paper, Figure 9b is generated by `python3 accept_models_lr.py` with the above result.

The exact data of this experiment and the figure in the submitted version of the paper may differ depending on the changes in the dataset. However, the main trend in both the results remains same.

Solver Runtime Overhead

1. Runtime vs pipeline length experiment:

```
./time_measure_copi_pipelength.sh
./time_measure_gekko_pipelength.sh
```

The results will be available in `scheduled_times_copi.txt`, `failed_times_copi.txt`, `accepted_time_gekko.txt` and `failed_time_gekko.txt` in space separated format.

Paper results are in `Figures/runtime_pipe_length.csv`. (1st, 2nd, 3rd, 4th, 5th line: pipeline length, CoPi times for schedulable, CoPi times for unschedulable, GEKKO times for schedulable and CoPi times unschedulable). Match these paper results with the results from the above scripts.

In the paper, Figure 10c is generated by `python3 runtime_pipe_length.py` with the above result.

Exact results may vary depending on a machine's performance, but the main trend in the results should remain the same.

2. Run `CoPi` experiments for Figure 9a and 9b:

```
./run_solver_exp2_copi.sh
```

3. Run `GEKKO` experiments for Figure 9a and 9b:

```
./run_solver_exp2_gekko.sh
```

The results will be available in `scheduled_times_copi.txt` and `failed_times_copi.txt` for `CoPi`, and `accepted_time_gekko.txt` and `failed_time_gekko.txt` for `GEKKO`, in space separated format.

Paper results are in `Figures/lbg_time.csv` and `Figures/lbg_time_failed.csv`, respectively for schedulable and unschedulable pipelines. In each file: (1st, 2nd line, 3rd line: LBG, times for CoPi and times for GEKKO). Match these paper results with the results from the above scripts. `scheduled_times_copi.txt` and `accepted_time_gekko.txt` files should match with `Figures/lbg_time.csv`. `failed_times_copi.txt` and `failed_time_gekko.txt` should match with `Figures/lbg_time_failed.csv`.

In the paper, Figure 10a and 10b are generated respectively by `python3 lbg_time.py` and `python3 lbg_time_failed.py` with the above results.

CoPi Performance Insight

CoPi's AR vs. pipeline length and NLBG

```
./run_perform_insight.sh
```

The results will be written to `accepted_sets_copi.txt` file in space separated format.

Paper results will be available in `Figures/ increasing_task.csv`:

1st line is NLBG.

2nd line is the number of total tasks.

3rd, 4th, 5th, 6th, 7th lines are for experiments with 3, 5, 10, 15 and 20 tasks.

In `accepted_sets_copi.txt`, every five consecutive entries corresponds to one line in `Figures/increasing_task.csv` starting from 3rd line until 7th line.

In the paper, Figure 11 is generated by `python3 increasing_task.py` with the above result.

Exact result may vary depending on the change of datasets, but the main trend of the result should be same.

Multiprocessor Experiments

Multiprocessor experiments are run by the following command:

Run the processor based experiments with the following script:

```
./run_multiproc_2cores.sh
./run_multiproc_4cores.sh
./run_multiproc_8cores.sh
```

Results will be written to `accepted_multiproc_<number of cores>_<number of tasks in a pipeline>.txt`.

Paper results are in `Figures/accept_multiprocessor_2core.csv`, `Figures/accept_multiprocessor_4core` `Figures/accept_multiprocessor_8core.csv`. In all these files, the format is following:

1st line is the pipeline length.

2nd line is number of schedulable pipelines.

3rd line is number of schedulable pipelines only with RPO.

4th line is number of schedulable pipelines only with migration.

5th line is number of schedulable pipelines only with RPO+migration.

1st column is for pipeline length 3, 2nd column is for pipeline length 5, 3rd column is for pipeline lngth 10.

Match these results with the above shell script results in `accepted_multiproc_*` files.

In the paper, Figure 13a, 13b and 13c are generated respectively by `python3 accept_multiprocessor_2core.py`, `python3 accept_multiprocessor_4core.py` and `python3 accept_multiprocessor_8core.py` with the above results.

Just for convenience, lower level python script:

```
python multi_pipeline.py -p <number of pipelines> -t <number of tasks in each Pipeline> -c <numb
```

Utilization Experiments and Results:

```
./run_util.sh
```

The result is written to `multi_util_result.txt`.

In the paper, the Table 4 results are retrieved from the above result.

Migration Experiments:

`./run_mig.sh`

The result is written to `migrations_result.txt`.

In the paper, the Table 5 results are retrieved from the above result.

WATERS 2015 Workshop Paper Experiments:

For the experimental results with the dataset provided in the WATERS 2015 workshop paper by Bosch (*Kramer, Simon, Dirk Ziegenbein, and Arne Hamann. "Real world automotive benchmarks for free." In 6th WATERS Workshop. 2015.*), run the following script:

`./run_waters.sh`

The results of the CoPi runtimes will be in `accepted_waters_copi.txt` file. The results of the derived E2E delay for schedulable pipelines will be populated in `accepted_waters_avge2e.txt` file.

Paper results are in `Figures/waters_ar.csv` and `Figures/waters_avge2e.csv`. The format is:

1st line is NLBG.

2nd line is total number of tasks.

3rd to 8th line is for pipeline length for 3, 5, 8, 10, 12, 15.

Match these results with the results retrieved by running the above shell script.

The exact result may vary a bit for the runtimes experiment depending on the machine where the code is being run. However, the overall trend should match.

In the paper, Figure 14a, and 14b are generated respectively by `python3 waters_ar.py` and `python3 waters_avge2e.py` with the above results.

Figure 14c can be generated using `python3 waters_wcet_cdf.py` in the main source directory.