Graph Coloring using GPUs

Submission No: 38

Overview Document

1 Getting Started Guide

Our paper 'Graph Coloring using GPUs' runs on a single GPU. The evaluation of the artifact requires

1. sudo apt-get install unzip
2. g++ version - 4.8.5,
3. CUDA version - 9.1

2 Step-by-Step Instructions

The section provides a step-by-step procedure to reproduce results of SIRG, ChenGC, csrcolor and other SIRG variants as mentioned in the paper. We have used Tesla P100 GPU, with 12GB memory. We evaluated our codes on 8 input graphs (details mentioned in the paper).

The repository contains four source code folders (`sirg/`, `chenGC/`, `csrcolor/`, `otherSIRGVariants/`)one output folder (`outputs/`) and one sample output folder(`sample_outputs/` - contains the sample results obtained on running the various programs on different input graphs).

We provide scripts to run our programs on all the input graphs. Note that one run of each program on all input graphs takes about $1\frac{1}{2}$ hours. For compiling and running separately (without a script) on other input graphs, please check README files in respective source code folders.

Follow the given instructions to get results mentioned in the paper.

```
$ cd GraphColoringGPUs_Artifact
$ export root="$(pwd)"
```

Before evaluating any program, create a directory `inputs/` and include all input graphs you want to use for evaluation, in this directory. Please note that the input graphs should be in '.mtx' format. Also set PATH variable to include <path/to/cuda-9.1-bin> and LD_LIBRARY_PATH to include <path/to/cuda-9.1-lib64>.

```
$ export PATH=$PATH:<path/to/cuda-9.1-bin>
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<path/to/cuda-9.1-lib64>
```

Section 6.1 of the paper involves comparison of SIRG (Scalable and Improved RIG algorithm for GPUs) with ChenGC and csrcolor. We provide steps to compile and run SIRG, ChenGC and csrcolor on the input

graphs and reproduce results. The following steps are provided assuming a GPU is connected to the machine on which the steps are being executed. In case you have to submit a job, add the steps to the job file and submit the job.

2.1 Compiling and Running SIRG

Follow the steps to evaluate SIRG. The output of running SIRG on all the input graphs will be stored in output_sirg.out

```
$ cd "$root"
$ cd sirg/src
$ ./sirg.sh
```

2.2 Compiling and Running ChenGC

Follow the steps to evaluate ChenGC. The output of running chenGC on all the input graphs will be stored in output_chengc.out

```
$ cd "$root"
$ cd chenGC/src/code
$ ./chengc.sh
```

2.3 Compiling and Running csrcolor

Follow the steps to evaluate NVIDIA's cuSPARSE library function csrcolor. The output of running csrcolor on all the input graphs will be stored in output_csrcolor.out

```
$ cd "$root"
$ cd csrcolor/src
$ ./csrcolor.sh
```

Section 6.2 of the paper shows the effect of the proposed optimizations Opt1 (Section 4.1 of paper) and Opt2 (Section 4.2 of paper) over baseline approach (Section 3 of paper). We provide the required steps to run baseline code and baseline with Opt1. Baseline with Opt1 and Opt2 is SIRG (as described above in Section 2.1).

2.4 Compiling and Running Baseline Code

Follow the steps to evaluate the baseline code. The output of running baseline code on all the input graphs will be stored in output_baseline.out

```
$ cd "$root"
$ cd otherSIRGVariants/src
$ ./othervar.sh baseline
```

2.5 Compiling and Running Baseline + Opt1 Code

Follow the steps to evaluate the baseline with Opt1 code. The output of running baseline code on all the input graphs will be stored in output_baselineOpt1.out

```
$ cd "$root"
$ cd otherSIRGVariants/src
$ ./othervar.sh baselineOpt1
```

Section 6.3 of the paper discusses the effect of our design decision of choosing adjColors per thread (SIRG) instead of per vertex (SIRGPerVertex). We provide steps to run 'SIRGPerVertex'.

2.6 Compiling and Running SIRGPerVertex Code - adjColors per vertex

Follow the steps to evaluate the SIRGPerVertex code. The output of running baseline code on all the input graphs will be stored in output_sirgpervertex.out

```
$ cd "$root"
$ cd otherSIRGVariants/src
$ ./othervar.sh sirgpervertex
```

All the `.out` files will be stored in `outputs/` folder.

2.7 Generating results - speedup and memory usage

To obtain speedup and memory usage results mentioned in the paper, run the following steps.

```
$ cd "$root"
$ ./6.1.sh
$ ./6.2.sh
$ ./6.3.sh
```

The output values corresponding to every graph presented in the paper are generated in `outputs/` folder.

- `outputs/results_6_a.out` → Fig 6(a)
- `outputs/results_6_b.out` → Fig 6(b)
- `outputs/results_7.out` → Fig 7
- `outputs/results_8.out` → Fig 8
- `outputs/results_9.out` → Fig 9

3 Frequently Asked Questions:

Q1: How to resolve this error?
`error:"array" is ambiguous`
- Use GCC version 4.8.5. Other versions of GCC cause this error.

Q2: What to do when facing the following error on running scripts after logging out and logging into the server?
`nvcc fatal : Path to libdevice library not specified`

- After logging out and logging into the system, run the following commands before running any scripts. Go to the directory where you have extracted GraphColoingGPUs_Artifact.zip file and run the following:

```
$ cd GraphColoringGPUs_Artifact
$ export root="$(pwd)"
$ export PATH=$PATH:<path/to/cuda-9.1-bin>
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<path/to/cuda-9.1-lib64>
```