PPoPP 2020

Optimizing GPU Programs by Partial Evaluation

Semyon Grigorev
JetBrains Research, Saint Petersburg University, Russia
s.v.grigoriev@spbu.ru, Semyon.Grigorev@jetbrains.com

Problem Statement

Memory traffic is a bottleneck of GPGPU programs. In the data analysis, part of kernel parameters is fixed during many kernel runs.

- Patterns in substring matching
- HMM in homology search
- Query in graph database querying

Known parameters still increase memory traffic. **Can we automatically optimize procedure when some parameters are known?**

Results

- It is possible to optimize procedures with partially known parameters by means of **partial evaluation** [1].
  - Optimized procedure for substring matching is up to 2 times faster.
  - Automatically optimized procedure for 2D convolution is comparable with the one optimized manually.
  - Optimization effect depends on GPU architecture and initial memory access pattern.

Future Research

- Utilization of LLVM.mix, a partial evaluator for LLVM IR, to use CUDA C instead of DSL.
- Reduction of specialization overhead to make it applicable in run-time.
- Integration with shared memory register spilling [2].
- Real-world examples evaluation.
  - Homology search in bioinformatics.
  - Regular expression matching.

Motivating Example

Filter parameters are fixed during one data processing session which runs procedure multiple times.

```
__global__ void handleData
                (int* filterParams, int* data, ...)
{
  __shared__ int cachedFilterParams[size];

  /*some code to load filterParams
    to cachedFilterParams*/
    ...
}
```

Data is split into multiple chunks, thus the procedure is run many tims

Filter params are read only and the same for all threads, so they are copied into shared memory to reduce memory traffic. In some cases this data can be placed in the constant memory.

Can we automatically create an optimal procedure for partially known parameters?

Partial Evaluation [1]

$$[\![\underbrace{handleData}_{handleData}]\!][filterParams, data] = [\![\overbrace{mix}^{\text{partial evaluator}}]\!][\underbrace{handleData, filterParams}_{handleData_{mix}}][data]$$

$$[\![mix]\!][handleData, [|2;3|]]$$

```
handleData (filterParams, data)
{
  res = new List()
  for d in data
    for e in filterParams
      if d % e == 0
      then res.Add(d)
  return res
}
```

```
handleData (data)
{
  res = new List()
  for d in data
    if d % 2 == 0 ||
       d % 3 == 0
    then res.Add(d)
  return res
}
```

Implementation

We use AnyDSL [4] framework for ahead-of-time partial evaluation.

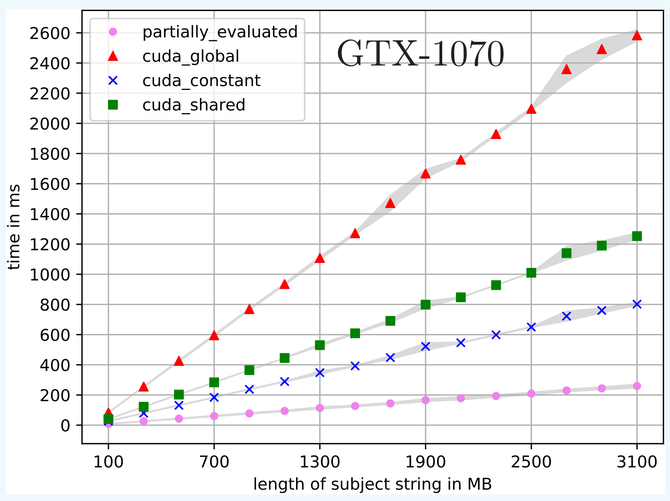
- Substring matching: partially evaluated naïve implementation and naïve implementation with different locations of patterns.
- 2D convolution: versions from CUDA SDK examples with manually unrolled loops for a number of filters' size and without such optimization, partially evaluated version which equals to SDK example without unrolling.

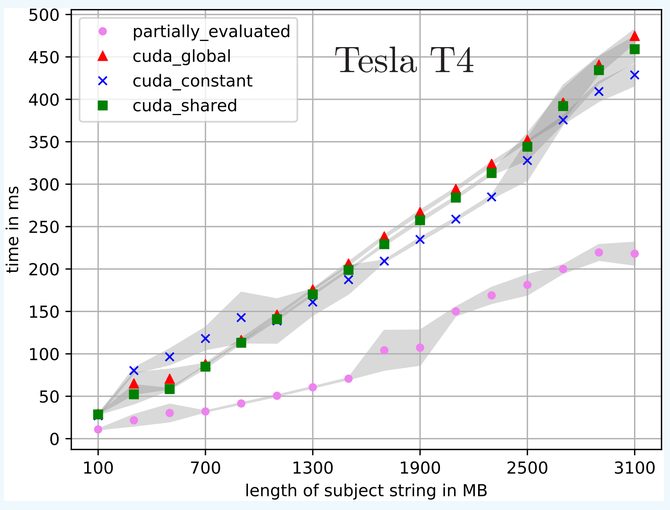
Hardware

- GTX-1070: Pascal architecture, 8GB GDDR5, 1920 CUDA cores.
- Tesla T4: Turing architecture, 16GB GDDR6, 2560 CUDA cores.

Evaluation: Substring Matching

Application: data curving in cyber forensics.
Subject string: byte array from real hard drive.
Patterns: 16 file signatures from GCK's file signatures table [3].





Evaluation: 2D Convolution

Application: data curving in cyber forensics.
Subject image: random image of size 1GB.
Filters: random square filters with diameter 3 to 255.

Contact Us

Our team:

- Semyon Grigorev: s.v.grigoriev@spbu.ru
- Aleksey Tyurin: alekseytyurinspb@gmail.com
- Daniil Berezun: daniil.berezun@jetbrains.com

Acknowledgments

This research was supported by Russian Foundation For Basic Research grant 18-01-00380, and a grant from JetBrains Research.

References

[1] Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[2] Putt Sakdhnagool, Amit Sabne, and Rudolf Eigenmann. Regdem: Increasing GPU performance via shared memory register spilling. *CoRR*, abs/1907.02894, 2019.

[3] Gary C Kessler. Gck's file signatures table. `https://www.garykessler.net/library/file_sigs.html`. Accessed February 4, 2020.

[4] Roland Leissa, Klaas Boesche, Sebastian Hack, Arsène Pérard-Gayot, Richard Membarth, Philipp Slusallek, André Müller, and Bertil Schmidt. Anydsl: A partial evaluation framework for programming high-performance libraries. *Proc. ACM Program. Lang.*, 2(OOPSLA):119:1–119:30, October 2018.