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 programms. There are cases of data analysis when some of kernel parameters are fixed during many kernel runs.

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

Known parameters are still increase memory traffic. **Can we automatically opimize procedire with partially known parameters?**

Results

- It is possible to optimize procedures with partially known parameters by using **partial evaluation** [1]
  - Optimized procedure for substring matching is up to 2 times faster
  - !!!

Future Research

- Switch to CUDA C partial evaluator.
  - LLVM.mix: partial evaluator for LLVM IR.
- Reduce specialization overhead to make it applicable in run-time.
- Integrete with shared memory register spilling [2].
- Evaluate on real-world examples.
  - Homology search in bioinformatics.
  - Graph processing.

Example

Parameters of filter are fixed during one data processing session which may contains many procedure runs.

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

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

In real-world cases we have a huge number of data chunks. Thus we have multiple procedure runs.

Filter params are read only and common for all threads, so we usually copy it into shared memory to reduce memory traffic. In some cases this data can be placed in the constant memory

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 [3] framework for ahead-of-time partial evaluation.

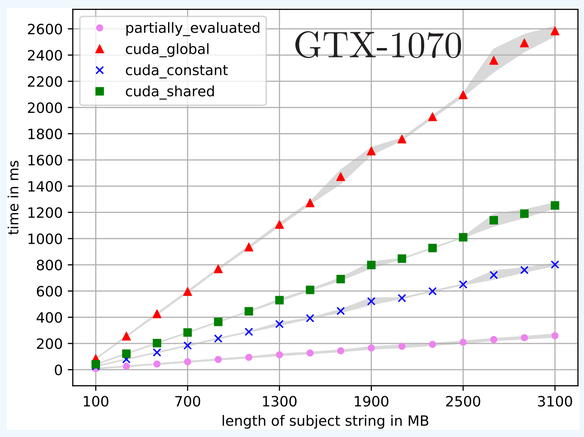
- Substring matching: partially evaluated naïve implementation and naïve implementation with different locations of patterns.
- 2D convolution: partially evaluated !!!!! from nvidia examples !!!! with unrolled loops by using predefined filter dianetr

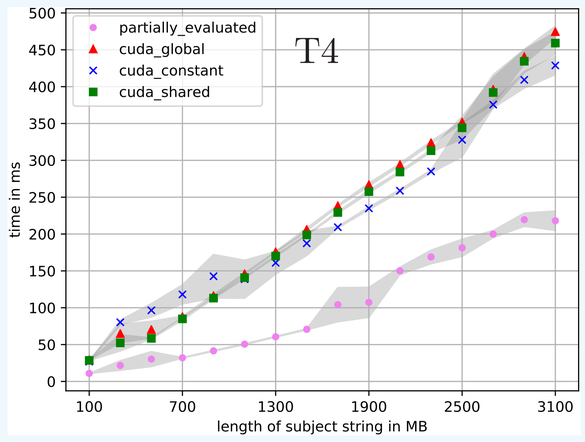
Hardware

- GTX-1070:
- T4:

Evaluation: Substring Matching

Application: data curving in cyber forensics
Subject string: byte array from real hard drive
Patterns: !!!!





Evaluation: 2D Convolution

Application: data curving in cyber forensics
Subject string: byte array from real hard drive
Patterns: !!!!

Contact Us

Our team:

- Semyon Grigorev: s.v.grigoriev@spbu.ru
- Nikita Mishin: mishinnikitam@gmail.com
- Iaroslav Sokolov: sokolov.yas@gmail.com

Both dataset and implementations are available on GitHub:

`https://github.com/SokolovYaroslav/CFPQ-on-GPGPU`

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] 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.

Acknowledgments

The research is supported by the JetBrains Research grant and the Russian Science Foundation grant 18-11-00100