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 big data analysis when some of kernel parameters are fixed during many kernel runs.

- Patterns in substring matching
-
-

Results

- Dataset for CFPQ evaluation is collected and published
  - Contains both graphs and queries
  - Contains both real-world and synthetic graphs
- Several CFPQ algorithms implementations are created, evaluated and published

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 [1]
- Evaluate on real-world examples

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,so we copy it into shared memory to reduce memory traffic

Partial Evaluation [2]

$$\underbrace{[\![handleData]\!]}_{handleData}[filterParams, data] = [\![\underbrace{[\![mix]\!][handleData, filterParams]}_{handleData_{mix}}]\!][da$$

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

```
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
}
```

We Need More Real-World Data

Graph: classical ontologies (RDFs)
Query: same-generation query over `type` and `SubClassOf` relations
Grammar: $S \rightarrow scor\ S\ sco \mid tr\ S\ t \mid scor\ sco \mid tr\ t$

RDF			Algorithms				
Name	#V	#E	Scipy	M4RI	GPU	CuSprs	CYK
atm-prim	291	685	3 ms	2 ms	1 ms	269 ms	8.5 min
biomed	341	711	3 ms	5 ms	1 ms	283 ms	7.1 min
pizza	671	2604	6 ms	8 ms	1 ms	292 ms	54 min
wine	733	2450	7 ms	6 ms	1 ms	294 ms	68 min

- 2019 (GPU) is 10^6 times faster than 2016 (CYK) on real-world data
  - Reasonable time even for CPU based implementations
- We should find bigger RDFs
- We should find other real-world cases for CFPQ
  - Both graphs and queries

We Should Do More Research on the Algorithms Scaling

	Graph	Scipy	M4RI	GPU	CuSprs
Sparse graphs are generated by GTgraph Query: $S \rightarrow a\ S\ b \mid a\ b$	G10k-0.001	37 s	2 s	0.2 s	35 s
	G10k-0.1	601 s	1 s	0.1 s	395 s
	G40k-0.001	-	97 s	8.1 s	-
	G80k-0.001	-	1142 s	65 s	-
Graph is a cycle Query: $S \rightarrow S\ S \mid a$	G25k	-	33 s	5 s	-
	G50k	-	360 s	44 s	-
	G80k	-	1292 s	190 s	-

- We can handle graphs with 80k vertices in a reasonable time by using GPGPU
  - Technical bound: GPGPU RAM does not fit bigger graphs
- We should evaluate multi-GPU systems
- We should evaluate distributed solutions
- We should implement a sparse boolean matrices library for GPGPU

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

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

[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