PPoPP 2020

POSTER: Optimizing GPU Programs By Partial Evaluation

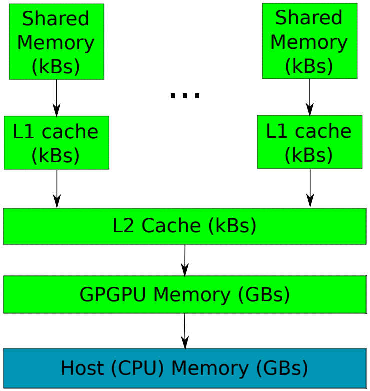
Aleksey Tyurin, Daniil Berezun, **Semyon Grigorev**

JetBrains Research, Programming Languages and Tools Lab
Saint Petersburg University

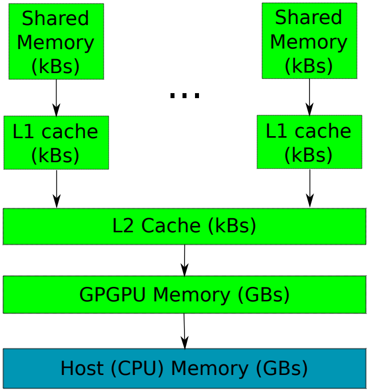
February 24, 2020

GPGPU Architecture

GPGPU memory hierarchy



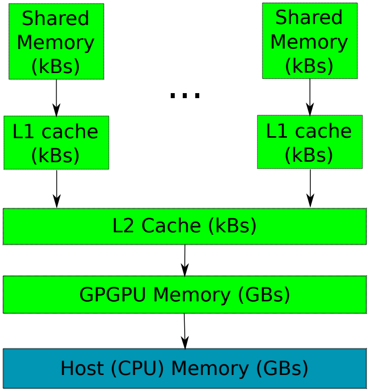
GPGPU Architecture



GPGPU memory hierarchy

- Global memory
  - ☺ Big
  - ☹ Slow

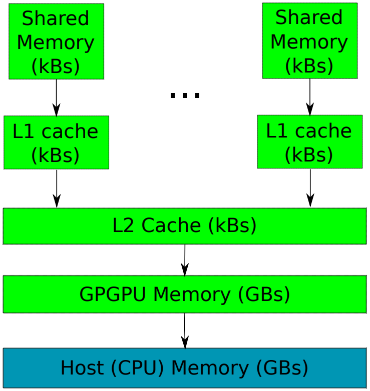
GPGPU Architecture



GPGPU memory hierarchy

- Global memory
  - ☺ Big
  - ☹ Slow
- Shared memory
  - ☺ Fast
  - ☹ Relatively small
  - ☹ Manual allocation mamagement

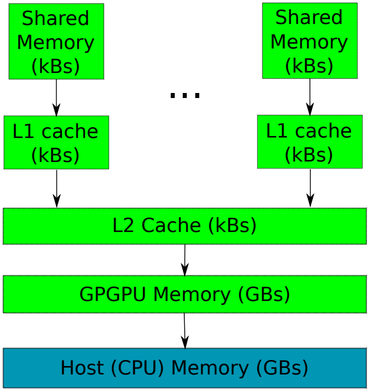
GPGPU Architecture



GPGPU memory hierarchy

- Global memory
  - ☺ Big
  - ☹ Slow
- Shared memory
  - ☺ Fast
  - ☹ Relatively small
  - ☹ Manual allocation mamagement
- Constant memory
  - ☺ Fast
    - ☹ Only for appropriate access pattern
  - ☹ Small
  - ☹ Static allocation

GPGPU Architecture



GPGPU memory hierarchy

- Global memory
  - ☺ Big
  - ☹ Slow
- Shared memory
  - ☺ Fast
  - ☹ Relatively small
  - ☹ Manual allocation mamagement
- Constant memory
  - ☺ Fast
    - ☹ Only for appropriate access pattern
  - ☹ Small
  - ☹ Static allocation
- **Memory traffic is a bottleneck**

Data Processing

- Substring matching
- Filtering by using Hidden Markov Models (HMM)

Data Processing

- Substring matching
- Filtering by using Hidden Markov Models (HMM)

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

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

Data Processing

- Substring matching
- Filtering by using Hidden Markov Models (HMM)

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

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

Data Processing

- Substring matching
- Filtering by using Hidden Markov Models (HMM)

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

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

Data Processing

- Substring matching
- Filtering by using Hidden Markov Models (HMM)

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

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

Big Data Processing

- Substring matching $\Rightarrow$ Data curving (cyber forensics)
- Filtering by using Hidden Markov Models (HMM) $\Rightarrow$ Homology search (bioinformatics)

Big Data Processing

- Substring matching ⇒ Data curving (cyber forensics)
- Filtering by using Hidden Markov Models (HMM) ⇒ Homology search (bioinformatics)

```
__global__ void handleData
                    (int* filterParams, int* data, ...)
{
    ...
}
```

Big Data Processing

- Substring matching ⇒ Data curving (cyber forensics)
- Filtering by using Hidden Markov Models (HMM
  (bioinformatics)

Many data chunks
⇒ many runs
of procedure

```
__global__ void handleData
                        (int* filterParams, int* data, ...)
{
    ...
}
```

Big Data Processing

- Substring matching ⇒ Data curving (cyber forensics)
- Filtering by using Hidden Markov Models (HMM) (bioinformatics)

One filter for many data chunks

Many data chunks ⇒ many runs of procedure

```
__global__ void handleData
                    (int* filterParams, int* data, ...)
{
    ...
}
```

Big Data Processing

- Substring matching ⇒ Data curving (cyber forensics)
- Filtering by using Hidden Markov Models (HMM) (bioinformatics)

One filter for many data chunks

Many data chunks ⇒ many runs of procedure

```
__global__ void handleData
                    (int* filterParams, int* data, ...)
{
    ...
}
```

`filterParams` is a static during one data porcessing session.

Big Data Processing

- Substring matching ⇒ Data curving (cyber forensics)
- Filtering by using Hidden Markov Models (HMM) (bioinformatics)

One filter for many data chunks

Many data chunks ⇒ many runs of procedure

```
__global__ void handleData
                    (int* filterParams, int* data, ...)
{
    ...
}
```

`filterParams` is a static during one data porcessing session.

How can we use this fact to optimize our procedure?

Partial Evaluation or Specialization

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

Partial Evaluation or Specialization

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

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

Partial Evaluation or Specialization

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

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

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

Partial Evaluation or Specialization

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

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

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

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

Partial Evaluation or Specialization

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

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

```
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
}
```

Evaluation Setup

- AnyDSL framework for specialization
  - Special DSL which can be specialized and comiled
  - Ahead-of-time specialization

Evaluation Setup

- AnyDSL framework for specialization
  - Special DSL which can be specialized and comiled
  - Ahead-of-time specialization
- Algorithms
  - Naïve multiple substring matching
  -

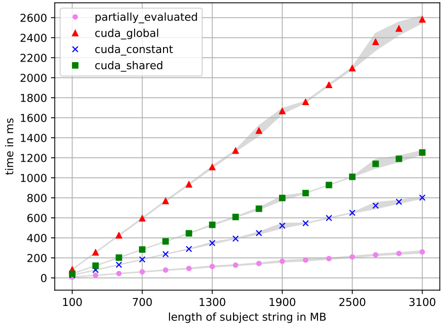
Evaluation Setup

- AnyDSL framework for specialization
  - Special DSL which can be specialized and comiled
  - Ahead-of-time specialization
- Algorithms
  - Naïve multiple substring matching
  -
- Environment

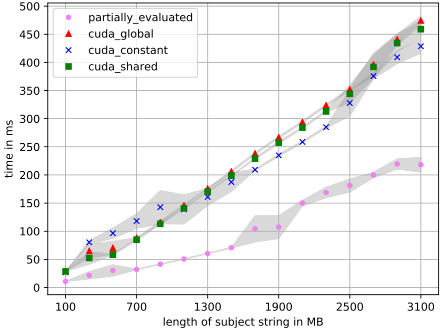
GTX-1070	Environment
T4	

Evaluation: Substring Matching

- Application: data curving
- Subject string: byte sequence from real hard drive
- Patterns: 16 file signatures from GCK's file signatures table[1]



Results for GTX-1070

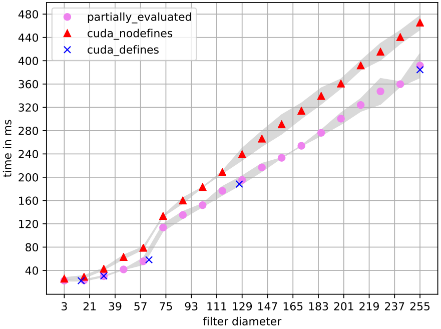


Results for T4

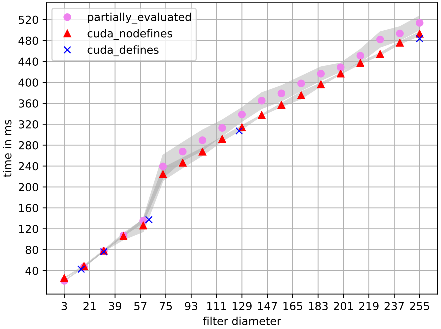
[1]https://www.garykessler.net/library/file_sigs.html

Evaluation: 2D Convolution

- Application: image processing
- Subject image: random image (16384 * 16384) 1Gb size
- Filters: random sqare filters with fiameter 13 to 255



Results for GTX-1070



Results for T4

Conclusion

- Partial evaluation improves performance of GPGPU procedures
  - !!!
  - !!!

Future Research

- Switch to CUDA C partial evaluator
  - LLVM.mix: partial evaluator for LLVM IR

Future Research

- Switch to CUDA C partial evaluator
  - LLVM.mix: partial evaluator for LLVM IR
- Reduce specialization overhead
  - To be applicable in run-time

Future Research

- Switch to CUDA C partial evaluator
  - LLVM.mix: partial evaluator for LLVM IR
- Reduce specialization overhead
  - To be applicable in run-time
- Integrete with shared memory register spilling
  - “RegDem: Increasing GPU Performance via Shared Memory Register Spilling” (Putt Sakdhnagool et.al. 2019)

Future Research

- Switch to CUDA C partial evaluator
  - LLVM.mix: partial evaluator for LLVM IR
- Reduce specialization overhead
  - To be applicable in run-time
- Integrete with shared memory register spilling
  - "RegDem: Increasing GPU Performance via Shared Memory Register Spilling" (Putt Sakdhnagool et.al. 2019)
- Evaluate on real-world examples
  - Homology search in bioinformatics
  - Graph processing
  - Graph database querying

Contact Information

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

- Dataset and algorithm implementations: https://github.com/SokolovYaroslav/CFPQ-on-GPGPU

Thanks!