

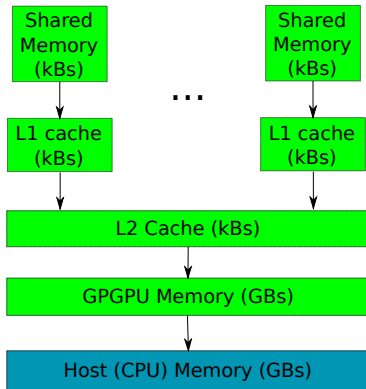
## POSTER: Optimizing GPU Programs by Partial Evaluation

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

JetBrains Research, Programming Languages and Tools Lab  
Saint Petersburg University

February 24, 2020

## GPGPU memory hierarchy

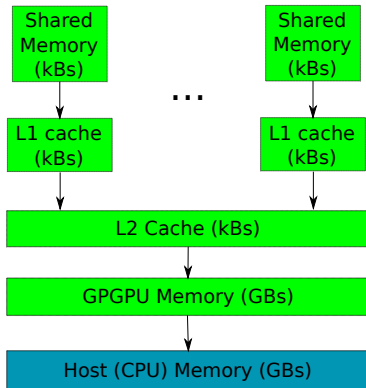


# GPGPU Architecture

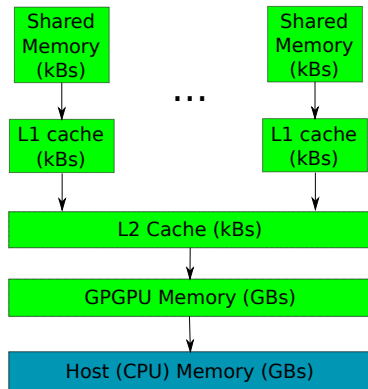
## GPGPU memory hierarchy

- Global memory

- 😊 Big
- 😞 Slow



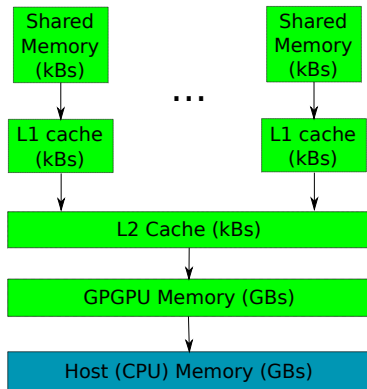
# GPGPU Architecture



## GPGPU memory hierarchy

- Global memory
  - 😊 Big
  - 😞 Slow
- Shared memory
  - 😊 Fast
  - 😞 Relatively small
  - 😞 Manual allocation management

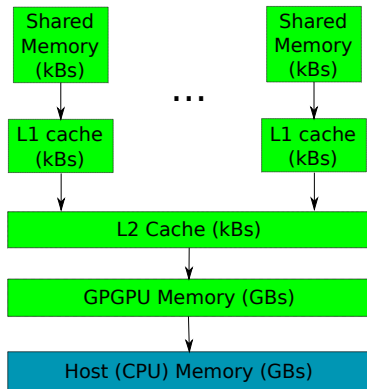
# GPGPU Architecture



## GPGPU memory hierarchy

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

# Data Processing

- Substring matching
- 2D convolution
- Filtering by using Hidden Markov Models (HMM)

# Data Processing

- Substring matching
- 2D convolution
- 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
- 2D convolution
- 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
- 2D convolution
- 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
- 2D convolution
- 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)
- 2D convolution  $\Rightarrow$  Image processing
- Filtering by using HMMs  $\Rightarrow$  Homology search (bioinformatics)

# Big Data Processing

- Substring matching  $\Rightarrow$  Data curving (cyber forensics)
- 2D convolution  $\Rightarrow$  Image processing
- Filtering by using HMMs  $\Rightarrow$  Homology search (bioinformatics)

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

# Big Data Processing

- Substring matching  $\Rightarrow$  Data curving (cyber forensics)
- 2D convolution  $\Rightarrow$  Image processing
- Filtering by using HMMs  $\Rightarrow$  Homology search

Many data chunks  $\Rightarrow$   
many procedure runs

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

# Big Data Processing

- Substring matching  $\Rightarrow$  Data curving (cyber forensics)
- 2D convolution  $\Rightarrow$  Image processing
- Filtering by using HMM

One filter for  
many data chunks

Many data chunks  $\Rightarrow$   
many procedure runs

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

# Big Data Processing

- Substring matching  $\Rightarrow$  Data curving (cyber forensics)
- 2D convolution  $\Rightarrow$  Image processing
- Filtering by using HMM

One filter for  
many data chunks

Many data chunks  $\Rightarrow$   
many procedure runs

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

filterParams is static during one data processing session



# Big Data Processing

- Substring matching  $\Rightarrow$  Data curving (cyber forensics)
- 2D convolution  $\Rightarrow$  Image processing
- Filtering by using HMM

One filter for  
many data chunks

Many data chunks  $\Rightarrow$   
many procedure runs

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

filterParams is static during one data processing session

How can we use this fact to optimize our procedure?

# Partial Evaluation or Specialization

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

# Partial Evaluation or Specialization

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

$\underbrace{\hspace{15em}}_{\text{handleData}_{\text{mix}}}$

```
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

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

$\underbrace{\hspace{15em}}_{\text{handleData}_{\text{mix}}}$

$\llbracket \llbracket \text{mix} \rrbracket[\text{handleData}, [2; 3]] \rrbracket$

```
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

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

$\text{handleData}_{\text{mix}}$

$\llbracket \text{mix} \rrbracket[\text{handleData}, [2; 3]]$

```
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
}
```

# Partial Evaluation or Specialization

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

$\text{handleData}_{\text{mix}}$

$\llbracket \text{mix} \rrbracket[\text{handleData}, [2; 3]]$

```
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

- We use AnyDSL framework for specialization
  - ▶ Special DSL which can be specialized and compiled
  - ▶ Ahead-of-time specialization

# Evaluation Setup

- We use AnyDSL framework for specialization
  - ▶ Special DSL which can be specialized and compiled
  - ▶ Ahead-of-time specialization
- Algorithms
  - ▶ Naïve multiple substring matching
  - ▶ 2D convolution

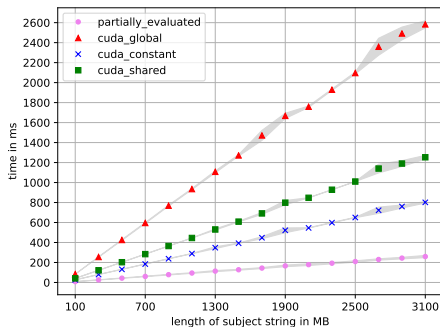


# Evaluation Setup

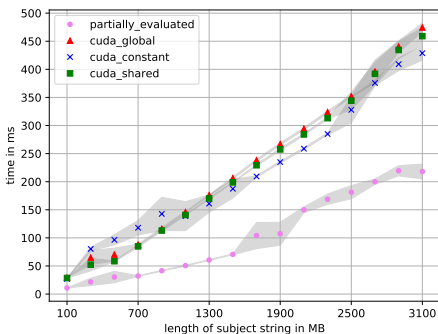
- We use AnyDSL framework for specialization
  - ▶ Special DSL which can be specialized and compiled
  - ▶ Ahead-of-time specialization
- Algorithms
  - ▶ Naïve multiple substring matching
  - ▶ 2D convolution
- Environment
  - ▶ **GTX-1070**: Pascal architecture, 8GB GDDR5, 1920 CUDA cores
  - ▶ **Tesla T4**: Turing architecture, 16GB GDDR6, 2560 CUDA cores

# Evaluation: Substring Matching

- Application: data curving
- Subject string: byte sequence from real hard drive
- Patterns: 16 file signatures from GCK's file signatures table<sup>1</sup>



Results for GTX-1070

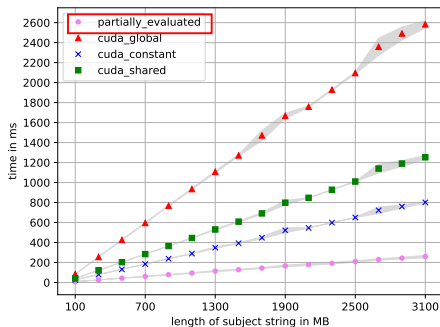


Results for Tesla T4

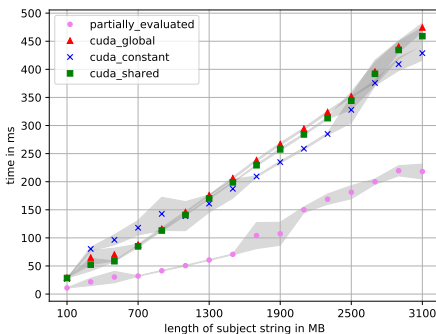
<sup>1</sup>[https://www.garykessler.net/library/file\\_sigs.html](https://www.garykessler.net/library/file_sigs.html)

# Evaluation: Substring Matching

- Application: data curving
- Subject string: byte sequence from real hard drive
- Patterns: 16 file signatures from GCK's file signatures table<sup>1</sup>



Results for GTX-1070

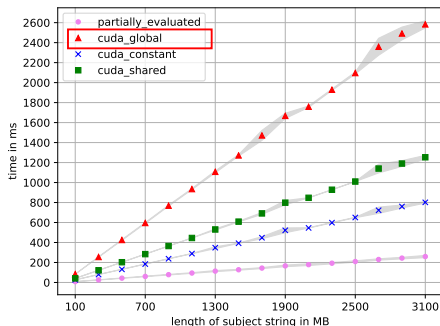


Results for Tesla T4

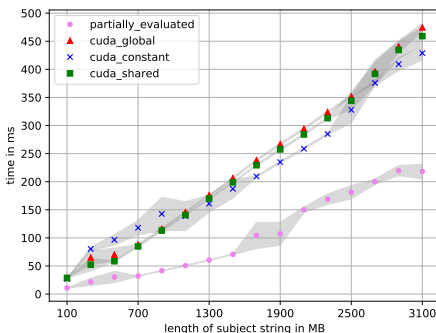
<sup>1</sup>[https://www.garykessler.net/library/file\\_sigs.html](https://www.garykessler.net/library/file_sigs.html)

# Evaluation: Substring Matching

- Application: data curving
- Subject string: byte sequence from real hard drive
- Patterns: 16 file signatures from GCK's file signatures table<sup>1</sup>



Results for GTX-1070

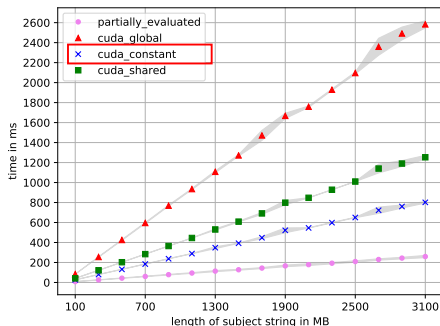


Results for Tesla T4

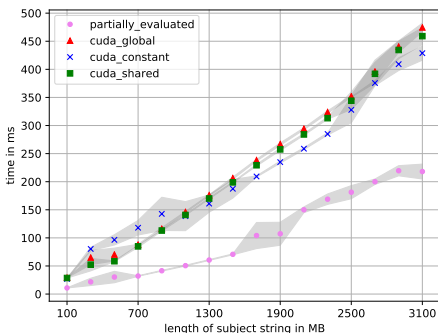
<sup>1</sup>[https://www.garykessler.net/library/file\\_sigs.html](https://www.garykessler.net/library/file_sigs.html)

# Evaluation: Substring Matching

- Application: data curving
- Subject string: byte sequence from real hard drive
- Patterns: 16 file signatures from GCK's file signatures table<sup>1</sup>



Results for GTX-1070

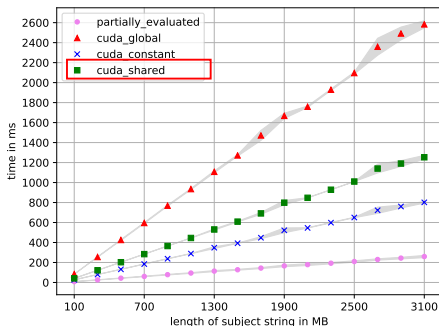


Results for Tesla T4

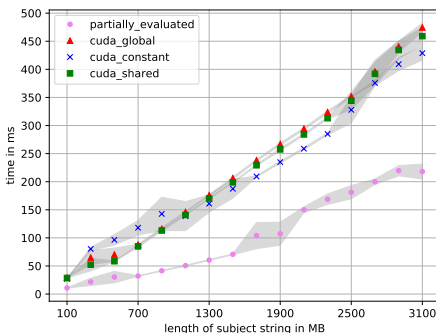
<sup>1</sup>[https://www.garykessler.net/library/file\\_sigs.html](https://www.garykessler.net/library/file_sigs.html)

# Evaluation: Substring Matching

- Application: data curving
- Subject string: byte sequence from real hard drive
- Patterns: 16 file signatures from GCK's file signatures table<sup>1</sup>



Results for GTX-1070

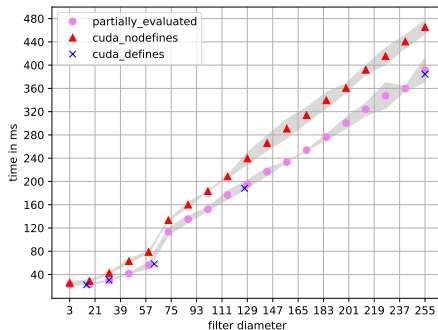


Results for Tesla T4

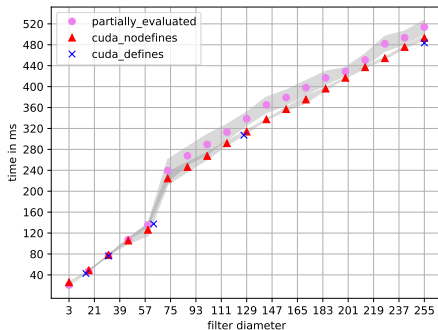
<sup>1</sup>[https://www.garykessler.net/library/file\\_sigs.html](https://www.garykessler.net/library/file_sigs.html)

# Evaluation: 2D Convolution

- Application: image processing
- Subject image: random image of size 1GB
- Filters: random square filters with diameter 3 to 255



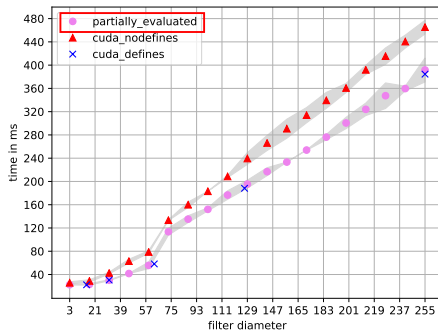
Results for GTX-1070



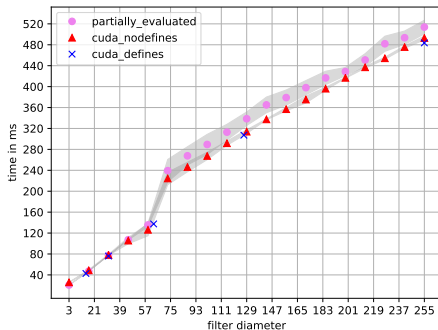
Results for Tesla T4

# Evaluation: 2D Convolution

- Application: image processing
- Subject image: random image of size 1GB
- Filters: random square filters with diameter 3 to 255



Results for GTX-1070

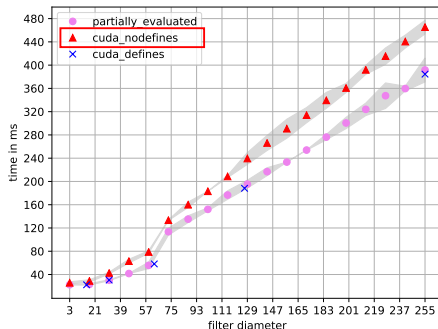


Results for Tesla T4

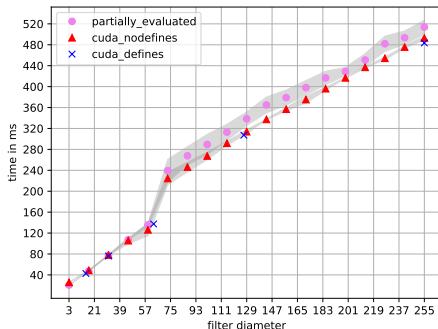


# Evaluation: 2D Convolution

- Application: image processing
- Subject image: random image of size 1GB
- Filters: random square filters with diameter 3 to 255



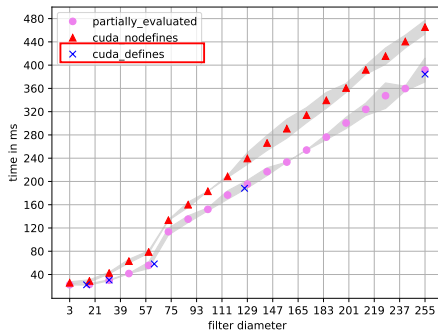
Results for GTX-1070



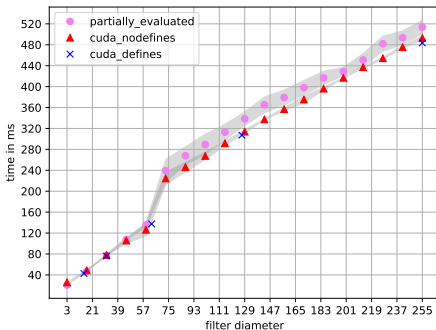
Results for Tesla T4

# Evaluation: 2D Convolution

- Application: image processing
- Subject image: random image of size 1GB
- Filters: random square filters with diameter 3 to 255



Results for GTX-1070



Results for Tesla T4

- Partial evaluation improves performance of GPGPU procedures
  - ▶ Effect depends on GPU architecture
    - ★ Dependencies should be carefully analyzed
  - ▶ Effect depends on the initial memory access pattern
    - ★ Irregular pattern — better performance improvement

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

# Future Research

- Migration to CUDA C partial evaluator
  - ▶ LLVM.mix: partial evaluator for LLVM IR
- Reduction of specialization overhead
  - ▶ To be applicable in run-time

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

- Migration to CUDA C partial evaluator
  - ▶ LLVM.mix: partial evaluator for LLVM IR
- Reduction of specialization overhead
  - ▶ To be applicable in run-time
- Integration with shared memory register spilling
  - ▶ “RegDem: Increasing GPU Performance via Shared Memory Register Spilling” (Putt Sakdhnagool et.al. 2019)
- Evaluation on real-world examples
  - ▶ Homology search in bioinformatics
  - ▶ Regular expression matching for traffic analysis, log processing
  - ▶ Graph database querying
  - ▶ Ray tracing, path tracing

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

Thanks!