High-level Techniques for Kernel Fusion and Other Optimizations

The AI technology in Information and Communications Technology domain sharing

Semyon Grigorev

Saint Petersburg State University

December 3, 2025

Problem Statement

- ✚ Library of basic functions is highly optimized
  - — But algorithms that utilize it are not

Problem Statement

- ✚ Library of basic functions is highly optimized
  - ━ But algorithms that utilize it are not
- Intermediate data structures problem

Highly-optimized function from linear algebra library

$$\underbrace{M_1 + M_2}_{M_4} + M_3 \rightsquigarrow add(\underbrace{add(M_1, M_2)}_{\text{Intermediate results allocation}}, M_3)$$

Problem Statement

- + Library of basic functions is highly optimized
  - − But algorithms that utilize it are not
- Intermediate data structures problem

$$\underbrace{M_1 + M}_{M_4} \quad \underbrace{, M_2)}_{\text{intermediate results allocation}}, M$$

One filter for many data chunks

Many data chunks ⇒ many procedure runs

- Parametrization

$$\texttt{\_\_global\_\_ void } \underbrace{\texttt{handleData}}_{\texttt{Generic function}} \; (\boxed{\texttt{int* filterParams}}, \; \boxed{\texttt{int* data,}} \; \ldots) \;\; \{\ldots\}$$

`filterParams` is static during one data processing session

How can we use this fact to optimize `handleData` procedure?

High-Level Techniques for Code Optimization

- Source-to-source transformations
  - Partial Evaluation[1]
  - Deforestation[2]
  - Supercompilation[3]
  - Distillation[4]

[1] Jones, N.D., Gomard, C.K., Sestoft, P.: Partial Evaluation and Automatic Program Generation.

[2] Wadler, P.: Deforestation: Transforming Programs to Eliminate Trees.

[3] Turchin, V.F.: The Concept of a Supercompiler.

[4] Hamilton, G.W.: The Next 700 Program Transformers

High-Level Techniques for Code Optimization

- Source-to-source transformations
  - Partial Evaluation[1]
  - Deforestation[2]
  - Supercompilation[3]
  - Distillation[4]

✓ Can be used to solve problems described above

? Well-suited for functional-first programming languages

☹ Not for Python or C

[1] Jones, N.D., Gomard, C.K., Sestoft, P.: Partial Evaluation and Automatic Program Generation.

[2] Wadler, P.: Deforestation: Transforming Programs to Eliminate Trees.

[3] Turchin, V.F.: The Concept of a Supercompiler.

[4] Hamilton, G.W.: The Next 700 Program Transformers

High-Level Languages For High-Performance Computing (HLL for HPC)

- Functional, functional-first programming languages for
  - GPGPU programming
  - FPGA programming (program specific processors)
  - Hardware synthesis
- Expressivity, high-level composable primitives
- Type safety, static code checks
- Specific optimizations
  - Fusion (stream fusion)
  - Partial evaluation
  - Deforestation
- Specific hardware

HLL for HPC: Projects

- **AnyDSL**: A partial evaluation framework for programming high-performance libraries
  - Saarland University, German Research Center for Artificial Intelligence (DFKI)
- **Futhark**: high-performance purely functional data-parallel array programming
  - University of Copenhagen
- **LIFT**: high-level functional data parallel language for portable HPC
  - University of Edinburgh, University of Glasgow
  - Supported by HIRP FLAGSHIP
- **Haflang**: special purpose processor for accelerating functional programming languages
  - Heriot Watt University
  - Supported by Xilinx and QBayLogic
- ...

Kernel Fusion

- ✓ Manual optimizations for frequent cases
  - Matrix multiply-add
  - ...
- ✓ Stream Fusion — for linear data
- ✓ XLA — for dense data
- ⚙ MLIR[5]
- ? For general sparse computations
  - Sparse attention
  - Graph neural networks
  - For GPGPU and other accelerators

[5] E.g. mlir-graphblas

Distillation for Sparse Linear Algebra Kernels (Work in Progress)

- Distillation[6]
  - High-level program transformation technique
  - Includes kernel-fusion-like optimization
- Special hardware
  - Reduceron[7]
    - Lambda-processor
    - Migration to Haflang
  - FHW[8]
    - Functional program to hardware translator
    - Program-specific accelerator

[6] https://github.com/YaccConstructor/Distiller
[7] https://github.com/tommythorn/Reduceron
[8] https://github.com/sedwards-lab/fhw

Preliminary Evaluation: Input

- A set of functions for sparse matrices manipulation
  - `addMask m1 m2 m3 = mask (mtxAdd m1 m2) m3`
  - `kronMask m1 m2 m3 = mask (kron m1 m2) m3`
  - `addMap m1 m2 = map f (mtxAdd m1 m2)`
  - `kronMap m1 m2 = map f (kron m1 m2)`
  - `seqAdd m1 m2 m3 m4 = mtxAdd (mtxAdd (mtxAdd m1 m2) m3) m4`
- All matrices are in quad-tree format

Preliminary Evaluation: Results (In Hardware Emulator)

Function	Matrix size				Interpreter		Reduceron	FHW
	m1	m2	m3	m4	Steps	Reads	Ticks	Ticks
seqAdd	64 × 64	64 × 64	64 × 64	64 × 64	2.7	1.9	1.8	1.4
addMask	64 × 64	64 × 64	64 × 64	–	2.1	1.8	1.4	1.4
kronMask	64 × 64	2 × 2	128 × 128	–	2.2	1.9	1.4	2.7
addMap	64 × 64	64 × 64	–	–	2.5	1.7	1.7	1.5
kronMap	64 × 64	2 × 2	–	–	2.9	2.2	1.8	2.0

Table: Evaluation results: original program to distilled one ratio of measured metrics

Partial Evaluation or Specialization

Utilization of partially known data to generate optimized code

- Convolution (1D, 2D, 3D) ⇒ Data (e.g. image) processing, including AI
  - Generic function for almost arbitrary kernel and data
→ optimized functions to apply specific (fixed) kernel to variable data
- Substring matching ⇒ Data curving (cyber forensics)

Partial Evaluation: Deatils

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

Partial Evaluation: Deatils

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

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

$\underline{[\![[\![mix]\!][handleData, [2;3]]]\!]}$

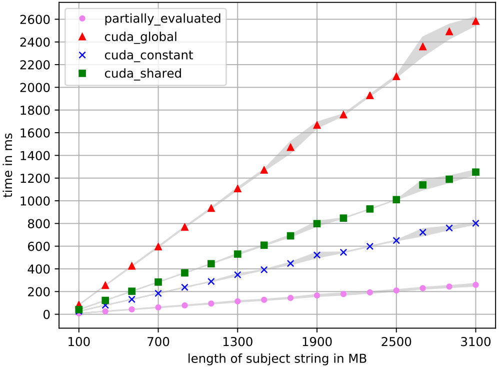
```
handleData_mix (data)
{
  res = new List()
  for d in data
    if d % 2 == 0 ||
       d % 3 == 0
    then res.Add(d)
  return res
}
```

Evaluation Setup

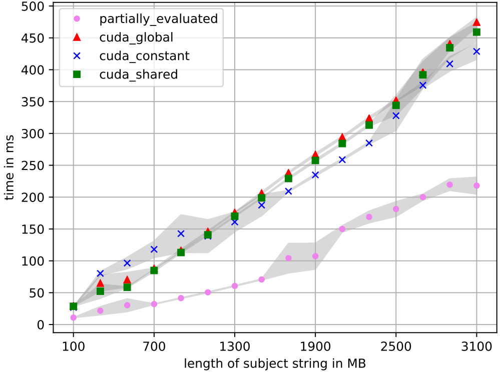
- We use **AnyDSL** framework for specialization
  - Special DSL which can be specialized and compiled
  - Ahead-of-time specialization
- Algorithms
  - Naïve multiple substring matching
  - 2D convolution
- Hardware
  - **GTX-1070**: Pascal architecture, 8GB GDDR5, 1920 CUDA cores
  - **Tesla T4**: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

Evaluation: Substring Matching

- Application: data curving
- Subject string: byte sequence from real hard drive
- Patterns: 16 file signatures from GCK's file signatures table[9]



Results for GTX-1070

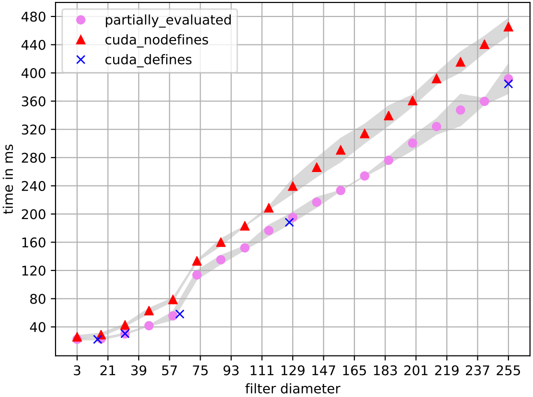


Results for Tesla T4

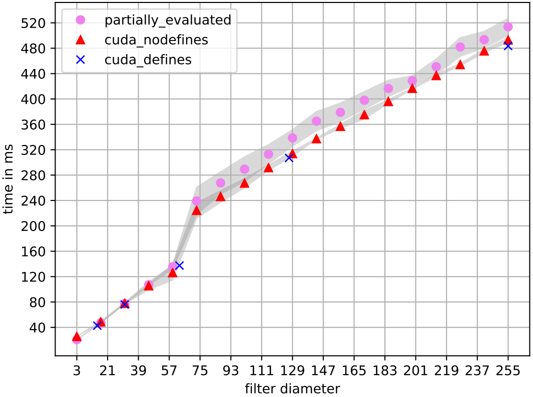
[9] https://www.garykessler.net/library/file_sigs.html

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

Conclusion

- Distillation can (partially) solve kernel fusion problem
- Partial evaluation can optimize code utilizing partially known data
- Functional-first programming languages can inspire optimization techniques useful in HPC and AI

Semyon Grigorev

- Associate professor at St. Petersburg State University
- Head of research group
- Research area
  - **High-performance generic linear algebra**
    - **Advanced optimization techniques** including software-hardware co-design
    - **GPGPU-powered** sparse linear algebra libraries
  - High-performance graph analysis

- Email: s.v.grigoriev@mail.spbu.ru
- GitHub: gsvgit
- Google Scholar: Semyon Grigorev
- DBLP: Semyon V. Grigorev