Distilled Sparse Linear Algebra in Hardware

Speaker: Aleksey Tyurin

Supervisors: Daniil Berezun and Semyon Grigorev

JetBrains Research, Programming Languages and Tools Lab
SPbU

17.12.2021

Sparse linear algebra

- In some circumstances data contains lots of "nil" values
  - Inefficient explicit storage
  - In case of linear algebra such data is represented with matrices
  - To mitigate the inefficiencies matrices are implemented with sparse data structures
    - ⋆ Store only needed elements by inducing pointer chasing
- Extensively used in
  - Graph analysis
  - Computational biology
  - Machine learning
  - ...
- *GraphBLAS* standard

GraphBLAS

- Standard, that defines sparse linear algebra building blocks to build graph analysis algorithms
- Optimized blocks $\rightarrow$ optimized algorithms
- Proposes several particular optimizations
  - Heuristic-driven load-balancing
  - Fusion-like optimizations

Fusion-like optimizations

- Eliminate intermediate data structures
- Ad-hoc implementations
  - Manually coded for particular operations, e.g. masking
  - Based on rewrite rules that require properties like commutativity, associativity, etc., which is not often the case in practice
  - Automatic fusion suffers from indexing induced by sparsity

Distillation

- A generalization of positive supercompilation
- Represents all possible execution paths of a program with a finite process graph
- Provides fusion
  - Since it performs a transitive closure of such a graph
- And other nice features
  - Positive information propagation
  - Specialization

Specialized hardware

- General purpose devices are inefficient for sparse applications
  - Underutilization
  - Power consumption
- Extensively used for sparse applications
- Power consumption is dominated by external memory accesses
- Fusion reduces memory accesses

The ultimate purpose

Purpose

Compile sparse linear algebra programs into specialized hardware with fusion in mind

- Distillation provides fusion for free for a functional language
- Functional language $\rightarrow$ specialized hardware?

Hardware synthesis

- Modern high-level synthesis tools do not handle arbitrary recursion
- We utilize FHW — Haskell to hardware compiler
  - Experimental
  - Arbitrary recursion
  - Parallel and pipelined dataflow representation
    - ⋆ Easily extensible with, e.g., custom memory
  - Hardware garbage collection (not present yet)
  - External Core GHC feature as a frontend
    - ⋆ Removed from GHC $>$ 7.6.3
  - No external memory support
    - ⋆ Data lives in code

Data structures

- Fusion result depends on the underlying data structure
- Widely used CSR, COO, etc. do not fit
- We use purely functional quad-tree representation

```
data QTree a = QNone
             | QVal a
             | QNode (QTree a) (QTree a)
                     (QTree a) (QTree a)
```

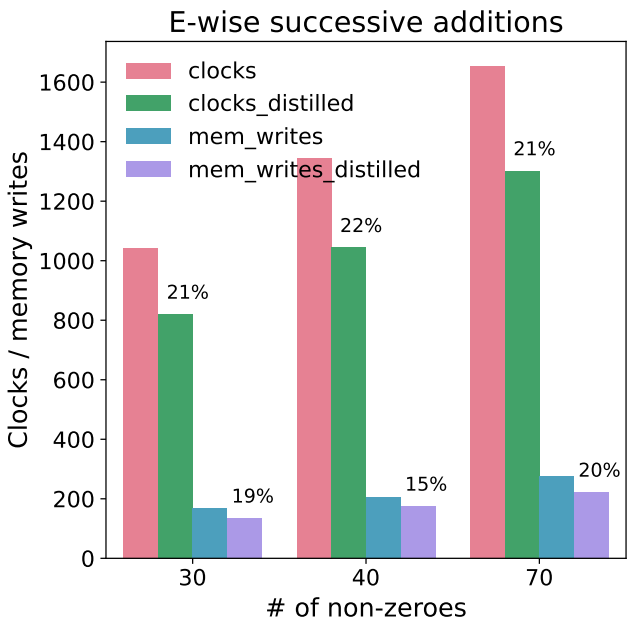
Plan

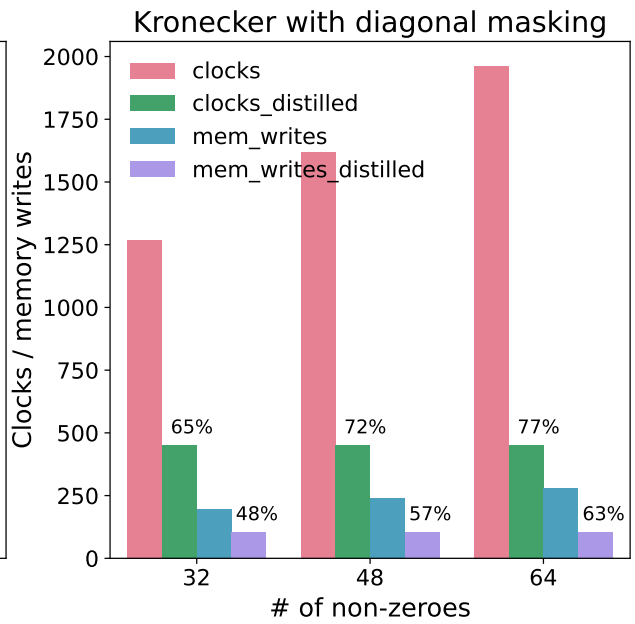
- Get FHW compiler to work
- Carry out first experiments
- Mitigate GHC 7.6.3 dependency
- Add support for external memory
- Run full-fledged experiments

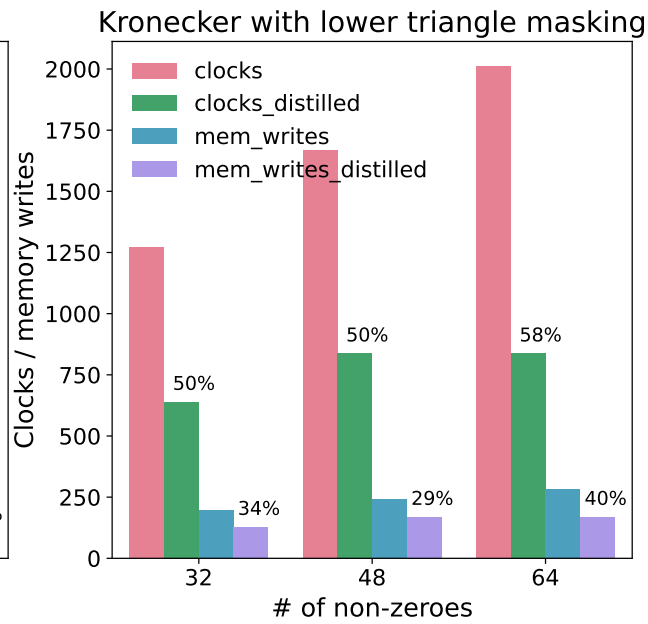
Get FHW compiler to work

- Involved a lot of dataflow and System Verilog debugging
- Fixed bugs
- Added support for the datatypes of arbitrary size
- Automated test workflow

Compare distilled and non-distilled programs in hardware (small matrices)







Compare distilled and non-distilled programs in hardware (larger matrices)

Function	m_1	m_2	m_3	clock ratio	time ratio
E-wise additions	64	64	64	0.94 $\pm$0.04	0.94 $\pm$0.04
	128	128	128	0.94 $\pm$0.04	0.95 $\pm$0.04
Masked addition	64	64	64	0.6 $\pm$0.17	0.9 $\pm$0.07
	128	128	128	0.6 $\pm$0.17	0.9 $\pm$0.09
	256	256	256	0.6 $\pm$0.17	0.9 $\pm$0.08
Masked kron	64	2	128	0.16 $\pm$0.03	0.35 $\pm$0.06
	64	4	256	0.27 $\pm$0.03	0.37 $\pm$0.05
	128	2	256	0.19 $\pm$0.01	0.38 $\pm$0.03
Map addition	64	64	64	0.93 $\pm$0.15	0.92 $\pm$0.06
	128	128	128	0.94 $\pm$0.14	0.92 $\pm$0.07
	128	2	256	0.94 $\pm$0.16	0.92 $\pm$0.07

Move from GHC 7.6.3

- We use GRIN framework as a middleware between frontend language of the distiller and dataflow backend
- It offers whole program optimization
- CPS and defunctionalization are more convenient

Results and plans

- ✔ We have the first evaluation
- ✔ Working dataflow backend[1]
- ✔ GRIN support for the frontend language[2]
- ✔ ICFP 2021 SRC poster (and few rejected ones)
- ⚙ GRIN $\rightarrow$ dataflow
- ⚙ External memory support
- ⚙ More mature evaluation
- ⚙ GrApl or YArch

[1] `https://github.com/sedwards-lab/fhw`
[2] `https://github.com/Tiltedprogrammer/SparseLinAlgHardware`