Complexity Analysis of Code Generation for the SCAD Machine

Hi, I am Markus Anders.

Complexity Analysis of Code Generation for the SCAD Machine

1. **Motivation**
2. Reduction for Bounded Buffers
3. Reduction for Unbounded Buffers

Motivation

The complexity of Code Generation for SCAD architectures is (or was) not known.

→Justify the use of heuristics

Motivation

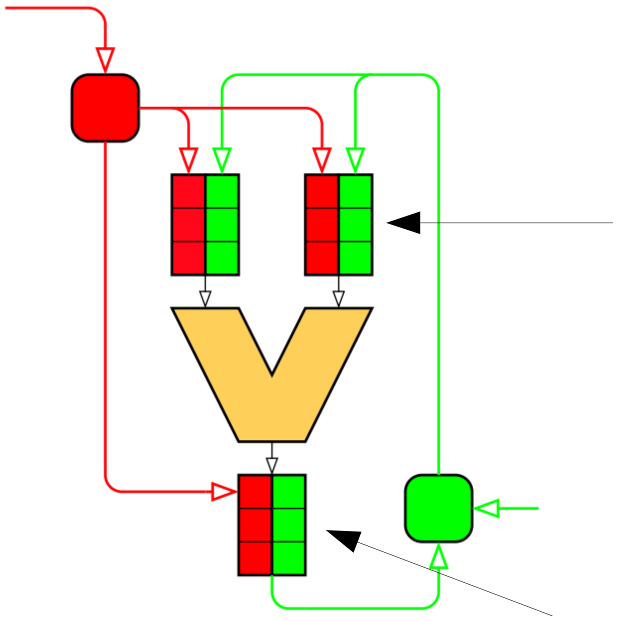
Most traditional code generation problems are based on registers.

→ Find relationships to known problems if possible

Complexity Analysis of Code Generation for the SCAD Machine

1. Motivation
2. **Reduction for Bounded Buffers**
3. Reduction for Unbounded Buffers

Bounded Buffers vs. Unbounded Buffers



Input and Output Buffers can either be bounded or unbounded

Bounded Buffer Implications

- Space in the processor is limited
- Spilling values to memory may be necessary

Bounded Buffer Implications

- Space in the processor is limited
- Spilling values to memory may be necessary

→ Optimize the amount of spill code
→ Queue operation (*dup*, *swap*) overhead will be ignored

Bounded Buffer Code Generation Problem

INSTANCE: Expression DAG D, positive integer k, positive integer n

QUESTION: Which move code M evaluates and stores all roots of D with the least amount of memory operations on k PUs in which all buffers are bounded by n?

Non-Commutative One-Register Optimal Code Generation

Instruction set

(1) $r \leftarrow m$ (load)
(2) $m \leftarrow r$ (store)
(3) $r \leftarrow r + m$ (memop)

Non-Commutative One-Register Optimal Code Generation

INSTANCE: Expression DAG *D*

QUESTION: What is the shortest machine program *M* that evaluates and stores all roots of *D*?

Reduction Idea

- Buffers in the SCAD machine will be bounded to 1
- 1 PU, will also act as the LSU
- Give a polynomial-time transformation from optimal register code to optimal SCAD code...
- … and vice-versa
- If transformations do not change the amount of memory operations, optimality follows

Register Code to SCAD Code

- For each of the register instructions equivalent SCAD code is needed
- SCAD code and register code have to use the same amount of memory operations
- Many details are omitted on these slides

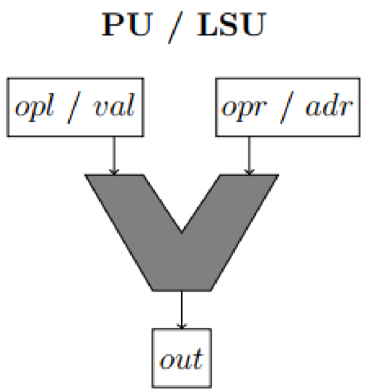
Register Code to SCAD Code

Register code:

```
r ← m
```

SCAD code:

```
load → opc
m → opr
out → opl
```



Register Code to SCAD Code

<u>Register code:</u>

```
m ← r
```

<u>SCAD code:</u>

```
store → opc
m → opr
```

Register Code to SCAD Code

Register code:

```
r ← r + m
```

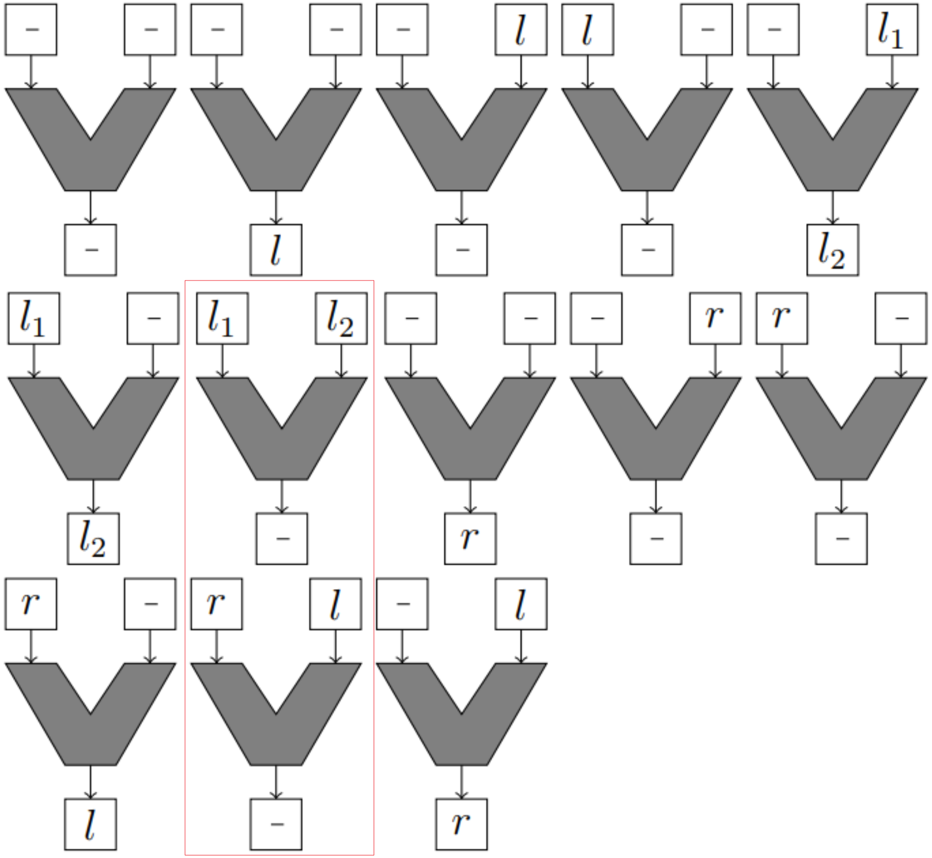
SCAD code:

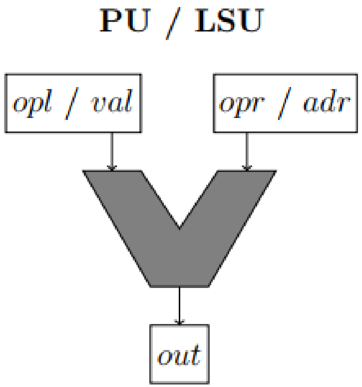
```
load→ opc
m → opr
out → opr
+ → opc
out → opl
```

SCAD Code to Register Code

- Queue overhead and moves are not important for optimality
- Track where data originally came from (*loaded* from memory or *result* of an operation)
- Generate register code when an operation is actually fired using the tracked information

SCAD Code to Register Code





Conclusions for Code Generation with Bounded Buffers

- → NP-completeness of the restricted problem for 1 PU and buffers bounded by 1
- → NP-hardness for the general problem
- → Relationship to a known register machine problem

Complexity Analysis of Code Generation for the SCAD Machine

1. Motivation
2. Reduction for Bounded Buffers
3. **Reduction for Unbounded Buffers**

Reduction with Unbounded Buffers

- All values can be kept inside the processor
- Spill code not necessary, queue overhead is

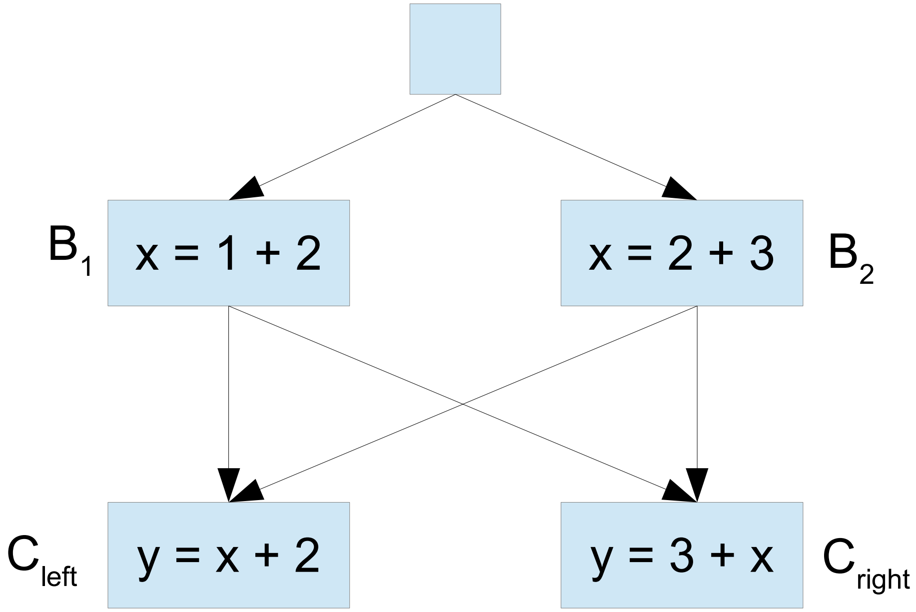
→ Optimize the amount of queue overhead (*dup*, *swap*)

Unbounded Buffer Code Generation Problem

INSTANCE: Program *P*, positive integer *k*

QUESTION: Is it possible to compile P without using *dup* or *swap* operations on *k* PUs?

Control Flow Implications



If no queue overhead is assumed, x has to be produced on the same PU

Reducing Graph Coloring

- Graph Coloring will be reduced
- A graph *G* is transformed into program *P*
- *G* is colorable with *k* colors iff P is is schedulable on *k* PUs without *dup* and *swap*

Reduction Idea

- Every vertex becomes a variable (v_1, …, v_n)
- Every edge becomes a basic block (B_1, …, B_m)
- PU assignment of a variable is the color of the vertex

Edge Basic Block

Vertices connected by an edge should not be colored with the same color

→

Corresponding variables should not be produced by the same PU

Edge Basic Block

B_j for edge $e_j = \{v_x, v_y\}$

$v_1 = \_ + \_$

$v_2 = \_ + \_$

…

$v_x = a + b$

…

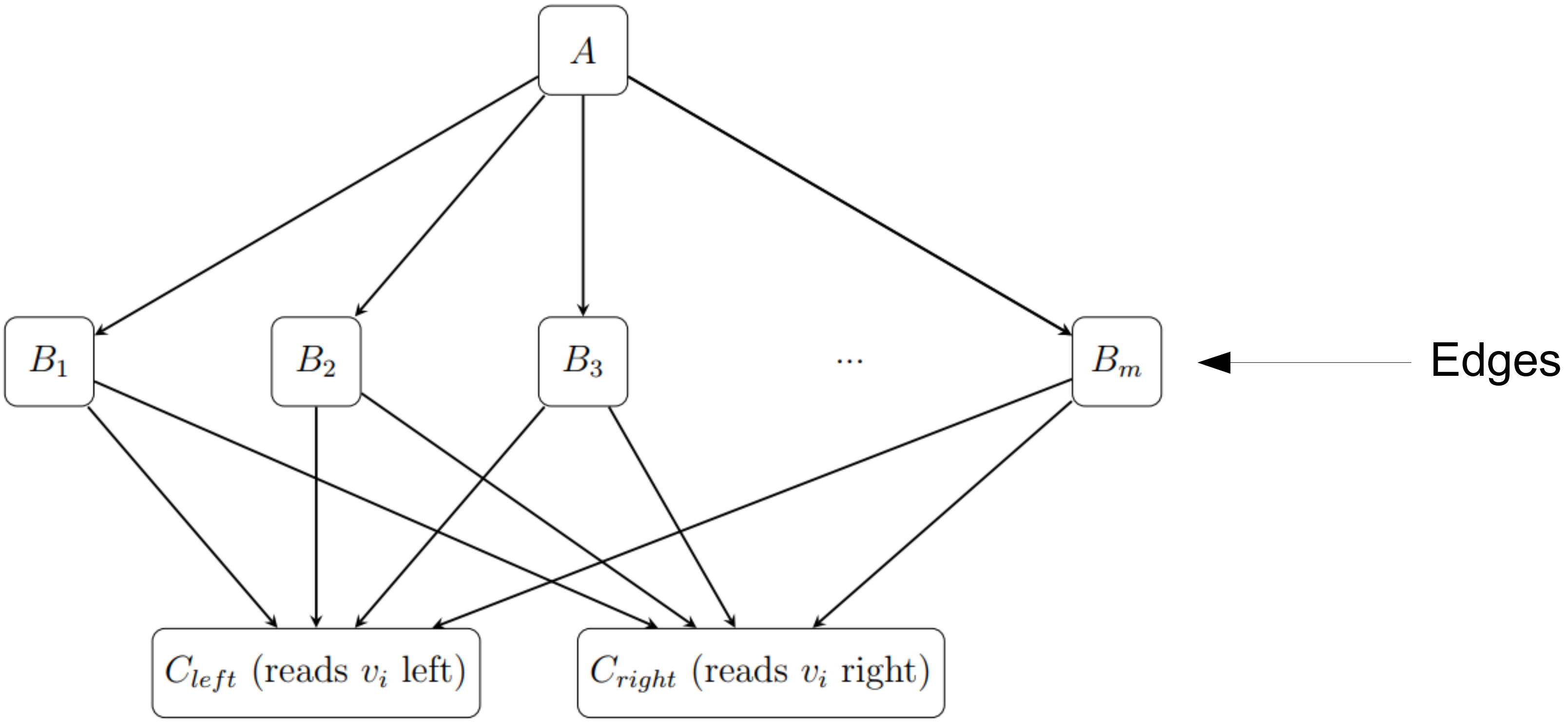
$v_y = b + a$

…

$v_i = \_ + \_$

with „a“ and „b“ being some load variable
→ can not be produced by the same PU

Reduction Control Flow



Conclusions for Code Generation with Unbounded Buffers

- → NP-hardness of the given problem
- → Reduction looks similar to register allocation

Thank you for your attention!

Sources

[1] F. Yazdanpanah, C. Alvarez-Martinez, D. Jimenez-Gonzalez, and Y. Etsion, “Hybrid dataflow/von-Neumann architectures,” IEEE Transactions on Parallel and Distributed Systems, vol. 25, pp. 1489–1509, June 2014.

[2] S. Swanson, K. Michelson, A. Schwerin, and M. Oskin, “WaveScalar,” in Microarchitecture (MICRO), (San Diego, California, USA), pp. 291–302, IEEE Computer Society, 2003.

[3] R. Tomasulo, “An efficient algorithm for exploiting multiple arithmetic units,” IBM Journal of Research and Development, vol. 11, no. 1, pp. 25–33, 1967.anpur, India), pp. 143–152, IEEE Computer Society, 2016..

[4] A. Bhagyanath and K. Schneider, “Exploring the potential of instruction-level parallelism of exposed datapath architectures with buffered processing units,” in Application of Concurrency to System Design (ACSD) (A. Legay and K. Schneider, eds.), (Zaragoza, Spain), pp. 106–115, IEEE Computer Society, 2017.

[5] A. Bhagyanath and K. Schneider, “Exploring different execution paradigms in exposed datapath architectures with buffered processing units,” in International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), (Samos, Greece), IEEE Computer Society, 2017.

[6] M. Garey and D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, 1979.

[7] A. Aho, S. Johnson, and J. Ullman, “Code generation with common subexpressions,” Journal of the ACM (JACM), vol. 24, no. 1, pp. 146–160, 1977.