High-performance GPU code generation for high-order stencils: Alleviating register pressure[†]

Prashant Singh Rawat, Aravind Sukumaran-Rajam,
Atanas Rountev, P. Sadayappan

The Ohio State University
Dept. of Comp. Sci. and Engg.

Overview

Goal

- Achieve high performance for high-order multi-statement stencil computations on GPU

Problem

- Higher-order stencils have high arithmetic intensity, but exhibit high per-thread register pressure
- Mature compilers like NVCC unable to perform well

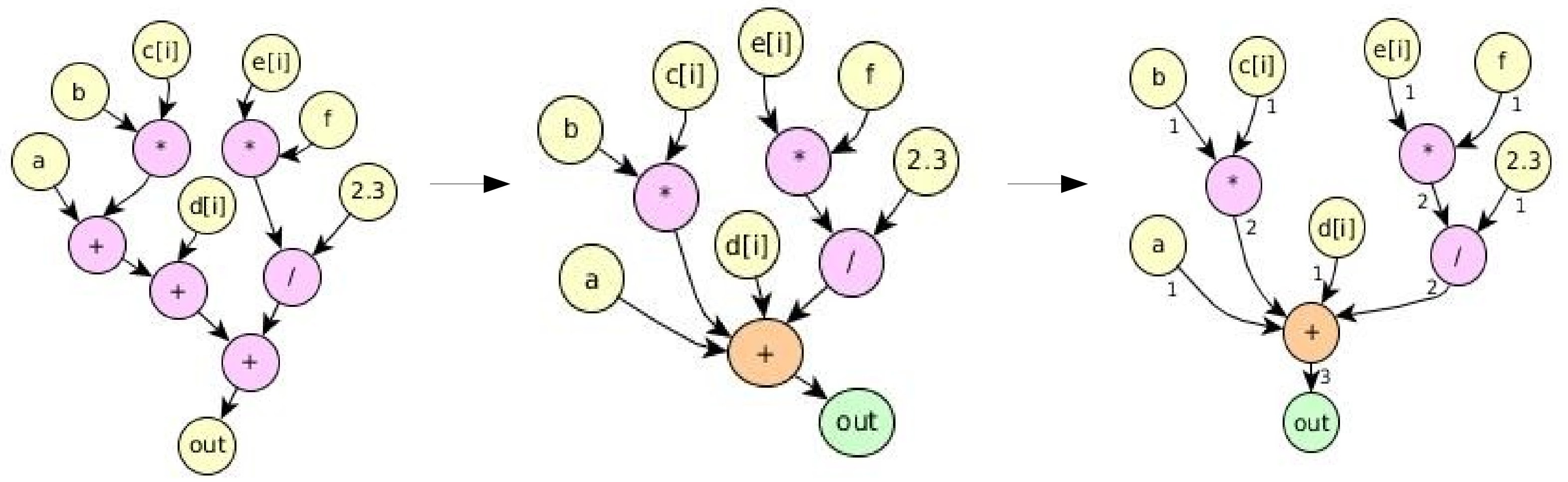
Solution Approach

- Spill-free minimal-register instruction scheduling for trees is known (Sethi-Ullman, 1970)
- A new abstraction that models a multi-statement stencil as a DAG of expression trees
- The many-to-many reuse within/across stencil ops captured via shared leaves
- Extend Sethi-Ullman register allocation to a DAG of expression tree with data sharing

Instruction Reordering: Stencils

Sethi-Ullman Scheduling

- Gives optimal evaluation of a tree *without* data reuse
- Computes Sethi-Ullman number (*SU*) for a node, which is the MAXLIVE for a subtree rooted at it
- For a binary node, prioritize evaluation of 'heavier' child first for optimality: better reuse of registers



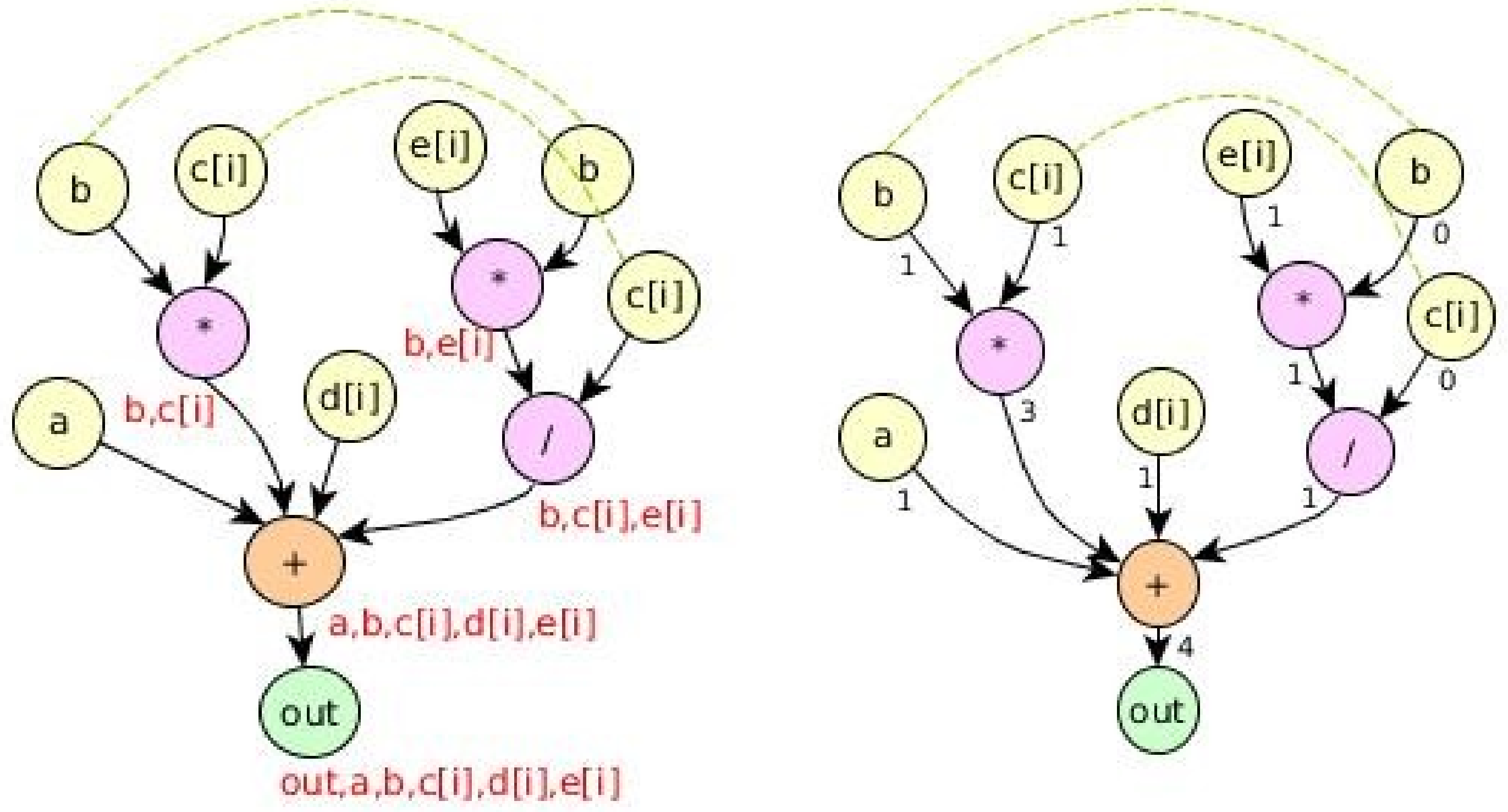
Expression Tree

Accumulation Form

MAXLIVE: SU Numbering

Scheduling a tree with sharing

- Modified Sethi-Ullman algorithm with 'context' of live-in and live-out values at each node
- For node *n*, try all permutations of children for evaluation. Select one with minimum MAXLIVE
- Optimal under atomic evaluation, but intractable

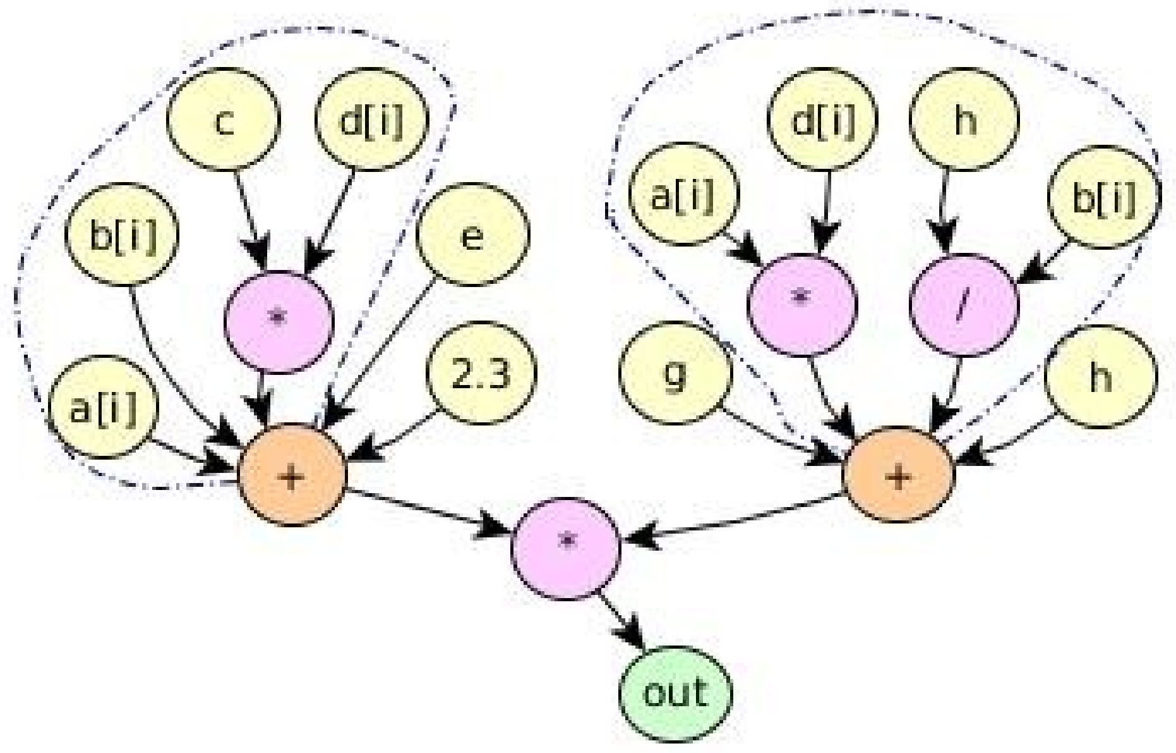


Optimizations for tractability

- Prune the evaluations with several heuristics:
  - Use *SU* for independent subtrees, exploration restricted to dependent subtrees
  - Stop exploration if the register requirement is close to the computed lower bound on *SU*
  - Memoize MAXLIVE with context at each subtree

Interleaving within subtrees

- Go beyond the restriction of atomic subtree evaluation – interleave computations to further reduce MAXLIVE
- Must be performed within and across trees
- Example: Bring uses of a[i],b[i],d[i] together



Scheduling a DAG of trees

- Generate versions with varying degree of splits, increase register-level reuse via unrolling
- For all the trees within a split,
  - fix an evaluation order that preserves dependences
  - perform computation interleaving across trees
  - perform scheduling and interleaving within a tree

Experimental Evaluation

- Evaluation on rhs4center_dev routine of sw4lite code (developer branch) K40c device with NVCC-8.0
- *restrict* keyword for texture cache, register pressure varied to get optimal performance
- Unrolling enhances register-level reuse, but better instruction order required to alleviate register pressure

Benchmarks	Domain	GFlops
loh1	301^2x171	150.59
		254.28
Cartesian	128^2x512	146.32
		235.39
Cartesian skinny	96^2x1600	132.60
		205.40
Pointsource	*201^3*	151.65
		249.49

🟩 *Original Code*
▨ *Register Optimized Code*

[†] Funded in part by DOE-ECP project: "Exascale Code Generation Toolkit", PI: Dan Quinlan