Compiling with the Regionalized Value State Dependence Graph

Nico Reissmann[1] Helge Bahmann[2] Jan Christian Meyer[1] Magnus Själander[1]
[1]{firstname.lastname}@ntnu.no [2]hcb@chaoticmind.net

1 Introduction

- Regionalized Value State Dependence Graph (RVSDG) is an intermediate representation (IR) for optimizing and parallelizing compilers
- Models flow of data with state edges to sequentialize side-effecting operations
- Enforces strict static single assignment (SSA) form
- Exposes hierarchical structure of programs
- Single unified IR that normalizes program representation

2 Regionalized Value State Dependence Graph

Expressions

```
*x += 4;
*y += 5;
```

Conditionals

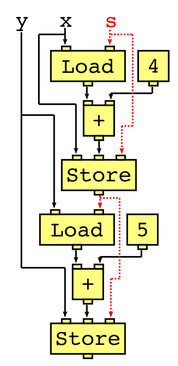
```
switch (x) {
  case 0: y = 1; break;
  case 1: y = 0; break;
  default: y = 2; break;
}
```

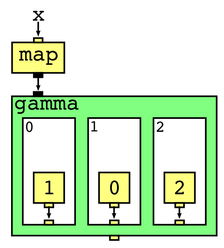
Loops

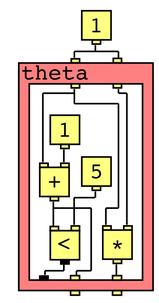
```
int r = 1, n = 1;
do {
  r = n*r;
  n++;
} while (n < 5);
```

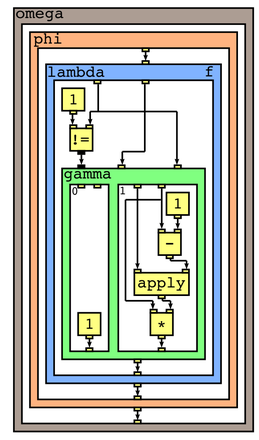
Functions

```
unsigned int
f (unsigned int x){
  if (1 != x)
    return x * f(x-1);
  return 1;
}
```









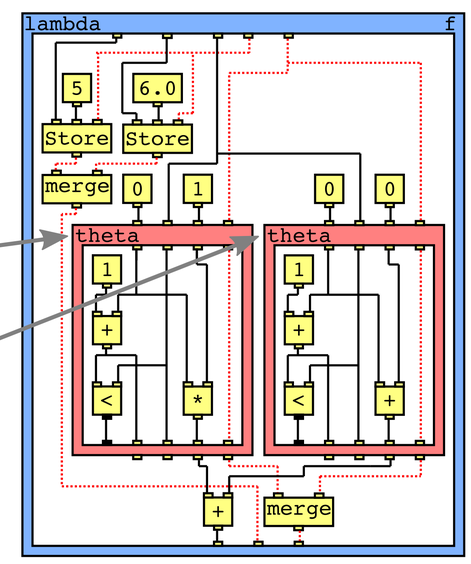
3 Motivation

- Raises IR abstraction level by enforcing desirable properties and relaxing the overly strict order of input programs
- Eliminates many helper passes* of conventional CFG-based compilers
- Explicitly exposes parallelism in programs

LLVM Optimization (-O3)	# Invocations
1. Alias Analysis	16
2. **Dominator Tree Construction***	**14**
3. Basic Alias Analysis	13
4. Scalar Evolution Analysis	10
5. **Natural Loop Canonicalization***	**9**
6. Redundant Instruction Combinator	8
7. **Loop-Closed SSA Form***	**8**
8. **Loop-Closed SSA Form Verifier***	**8**
9. **CFG Simplifier***	7
10. **Natural Loop Information***	**6**
Total	99
SSA Restoration*	**12**

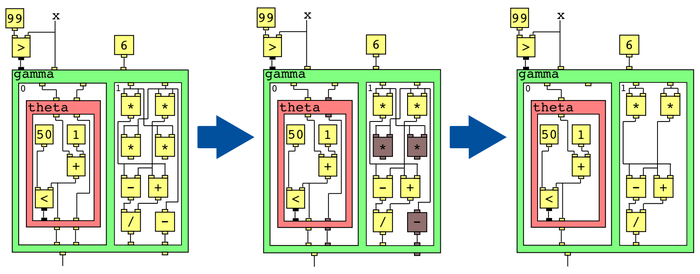
```
int f(int* x, float* y, int k)
{
  *x = 5;
  *y = 6.0;

  int i=0, f=1;
  int sum=0, fac=1;
  do {
   fac += f;
   f++;
  } while(f < k);

  do {
    sum *= i;
    i++;
  } while(is < k);

  return fac+sum;
}
```

Exposed Parallelism

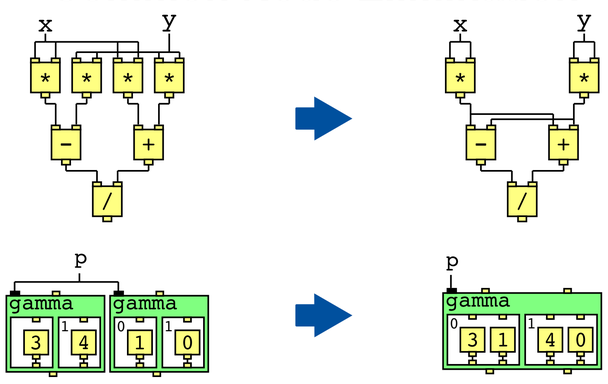


4 Optimizations

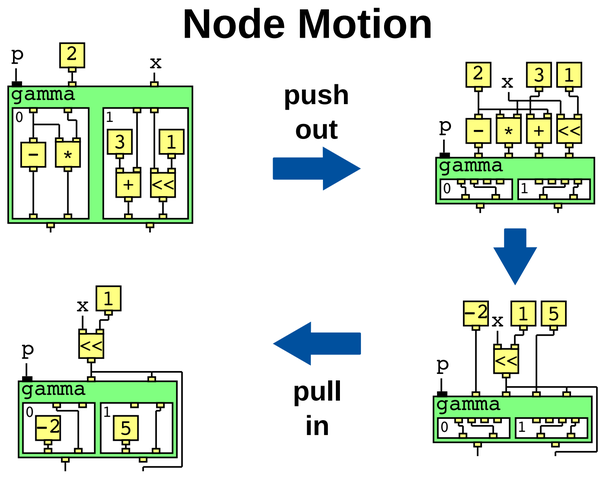
Dead Node Elimination



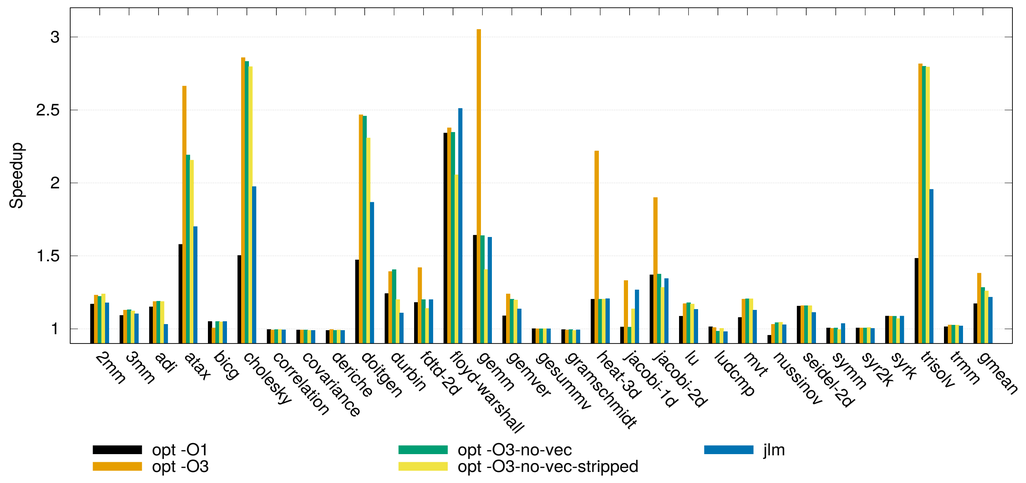
Common Node Elimination



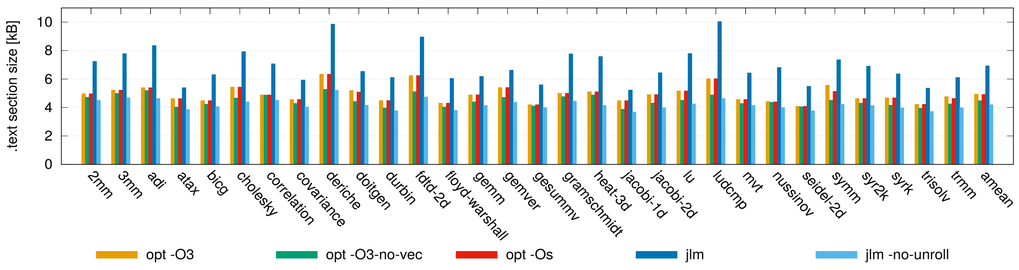
Node Motion



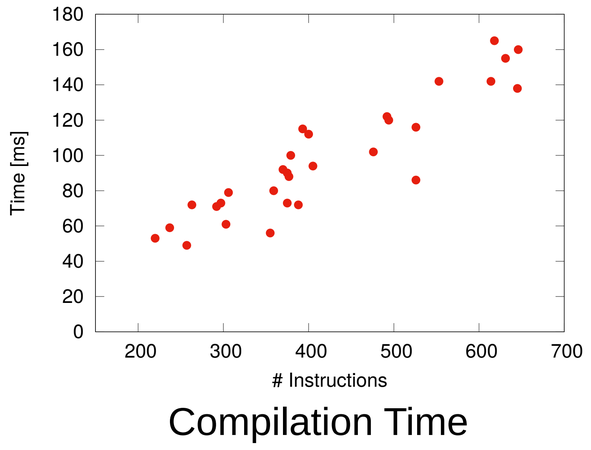
5 Preliminary Results



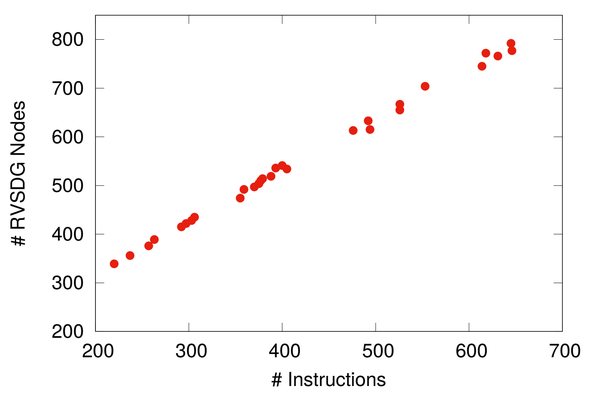
Speedup relative to LLVM -O0



Codesize



Compilation Time



Representational Overhead

6 Further Information

1. **Perfect Reconstructability of Control Flow from Demand Dependence Graphs**
   Transactions on Architecture and Code Optimization (TACO), 2015
2. **RVSDG: An Intermediate Representation for the Multi-Core Era**
   Nordic Workshop on Multi-Core Computing (MCC), 2018
3. **RVSDG API implementation:** `https://github.com/phate/jive`
4. **LLVM-based compiler using RVSDG:** `https://github.com/phate/jlm`
5. **RVSDG Viewer:** `https://github.com/phate/rvsdg-viewer`