Florida State University

Exhaustive Optimization Phase Order Space Exploration

Prasad A. Kulkarni
David B. Whalley
Gary S. Tyson
Jack W. Davidson

Florida State University

Optimization Phase Ordering

- Optimizing compilers apply several optimization phases to improve the performance of applications.
- Optimization phases interact with each other.
- Determining the best order of applying optimization phases has been a long standing problem in compilers.

Florida State University

Exhaustive Phase Order Enumeration... is it Feasible ?

- A obvious approach to address the phase ordering problem is to exhaustively evaluate all combinations of optimization phases.
- Exhaustive enumeration is difficult
  - compilers typically contain many different optimization phases
  - optimizations may be successful multiple times for each function / program

Optimization Space Properties

- Phase ordering problem can be made more manageable by exploiting certain properties of the optimization search space
  - optimization phases might not apply any transformations
  - many optimization phases are independent
- Thus, many different orderings of optimization phases produce the same code.

Florida State University

Re-stating the Phase Ordering Problem

- Rather than considering all attempted phase sequences, the phase ordering problem can be addressed by enumerating all distinct *function instances* that can be produced by combination of optimization phases.
- We were able to exhaustively enumerate 109 out of 111 functions, in a few minutes for most.

Florida State University

Outline

- Experimental framework
- Algorithm for exhaustive enumeration of the phase order space
- Search space enumeration results
- Optimization phase interaction analysis
- Making conventional compilation faster
- Future work and conclusions

Florida State University

Experimental Framework

- We used the VPO compilation system
  - established compiler framework, started development in 1988
  - comparable performance to gcc –O2
- VPO performs all transformations on a single representation (RTLs), so it is possible to perform most phases in an arbitrary order.
- Experiments use all the 15 available optimization phases in VPO.
- Target architecture was the StrongARM SA-100 processor.

Florida State University

VPO Optimization Phases

ID	Optimization Phase	ID	Optimization Phase
b	branch chaining	l	loop transformations
c	common subexpr. elim.	n	code abstraction
d	remv. unreachable code	o	eval. order determin.
g	loop unrolling	q	strength reduction
h	dead assignment elim.	r	reverse branches
i	block reordering	s	instruction selection
j	minimize loop jumps	u	remv. useless jumps
k	register allocation		

Disclaimers

- Did not include optimization phases normally associated with compiler front ends
  - no memory hierarchy optimizations
  - no inlining or other interprocedural optimizations
- Did not vary how phases are applied.
- Did not include optimizations that require profile data.

Florida State University

Benchmarks

- Used one program from each of the six MiBench categories.
- Total of 111 functions.

Category	Program	Description
auto	bitcount	test processor bit manipulation abilities
network	dijkstra	Dijkstra's shortest path algorithm
telecomm	fft	fast fourier transform
consumer	jpeg	image compression / decompression
security	sha	secure hash algorithm
office	stringsearch	searches for given words in phrases

Florida State University

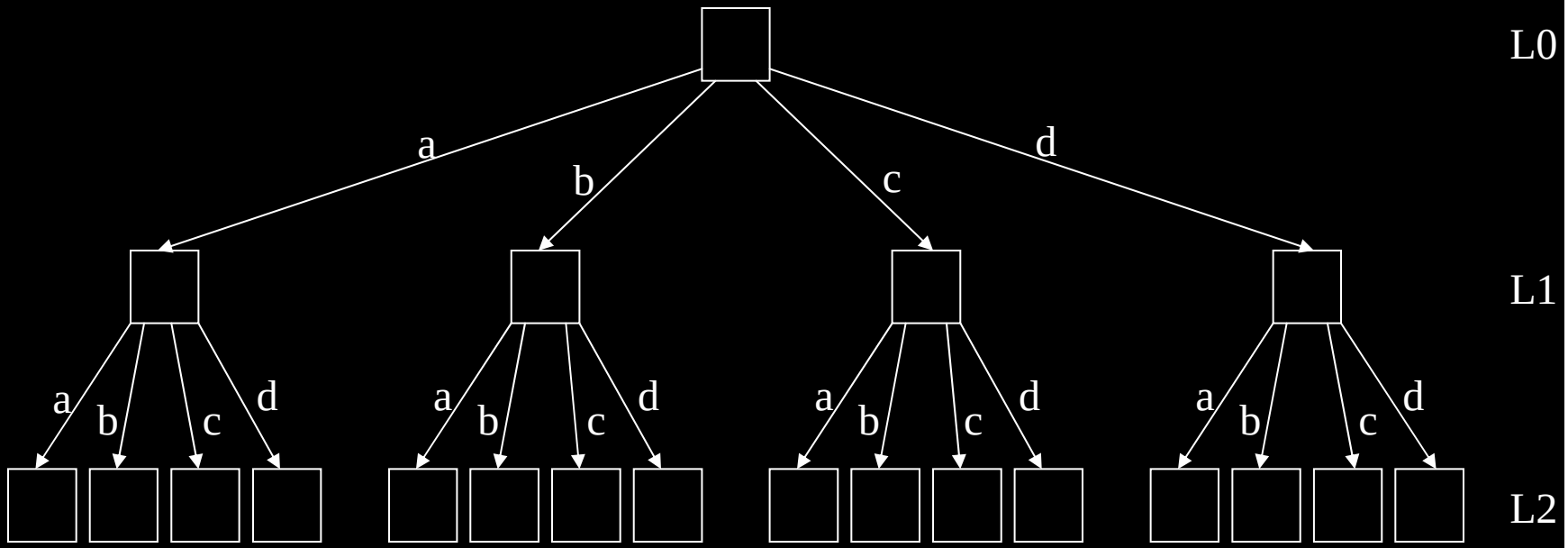
Outline

- Experimental framework
- Exhaustive enumeration of the phase order space.
- Search space enumeration results
- Optimization phase interaction analysis
- Making conventional compilation faster
- Future work and conclusions

Florida State University

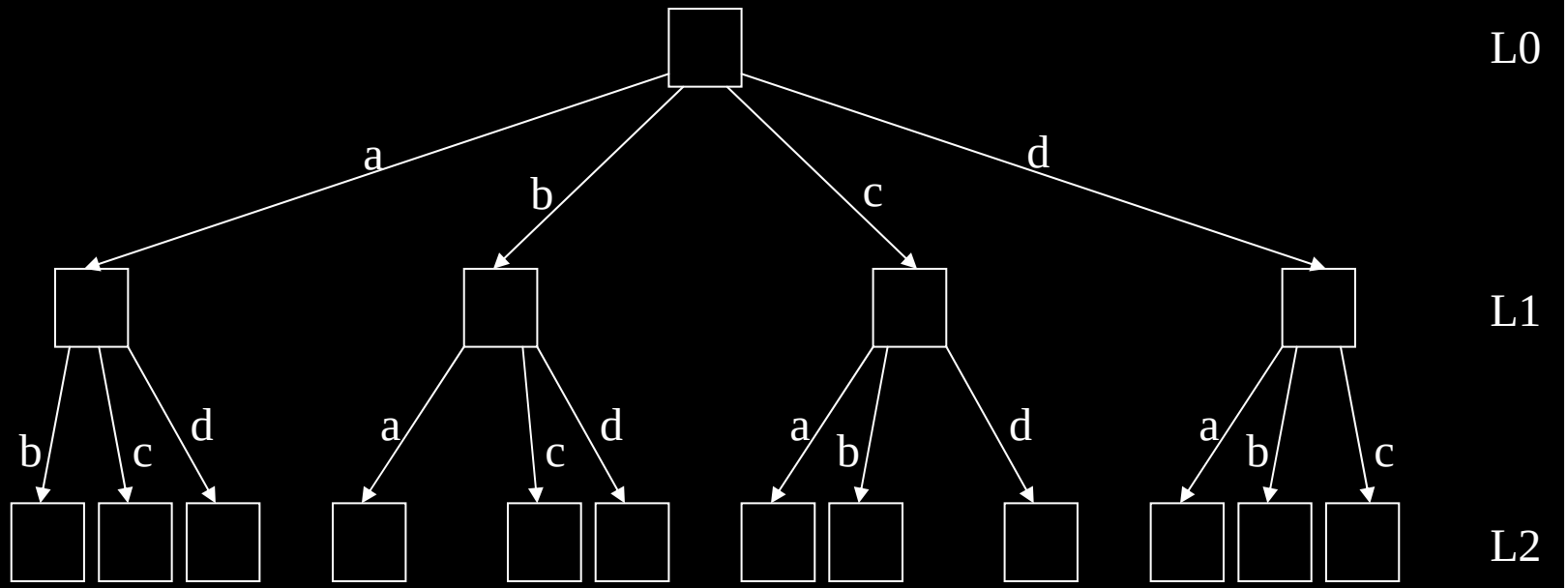
Naïve Optimization Phase Order Space Exploration

- All combinations of optimization phase sequences are attempted.



Eliminating Consecutively Applied Phases

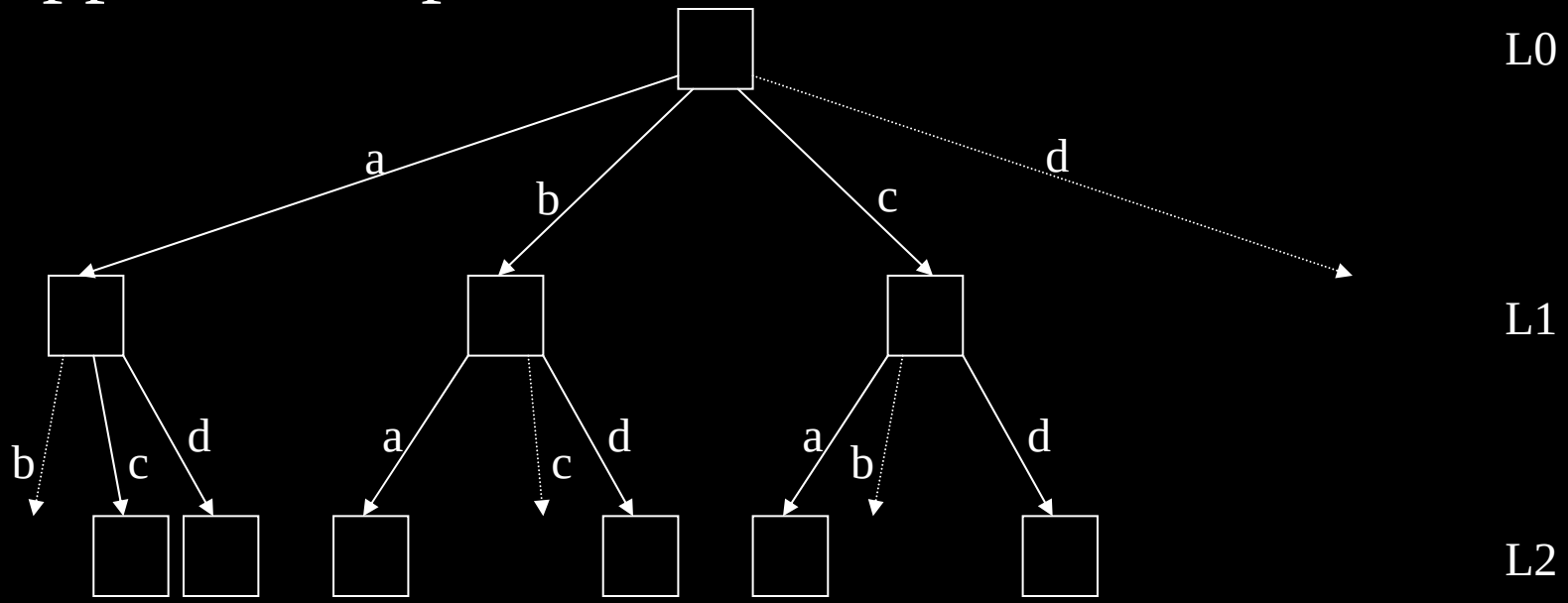
- A phase just applied in our compiler cannot be immediately active again.



Florida State University

Eliminating Dormant Phases

- Get feedback from the compiler indicating if any transformations were successfully applied in a phase.



Florida State University

Detecting Identical Function Instances

- Some optimization phases are independent
  - example: branch chaining & register allocation
- Different phase sequences can produce the same code

```
r[2] = 1;
r[3] = r[4] + r[2];
```

⇒instruction selection

```
r[3] = r[4] + 1;
```

```
r[2] = 1;
r[3] = r[4] + r[2];
```

⇒constant propagation

```
r[2] = 1;
r[3] = r[4] + 1;
```

⇒dead assignment elimination

```
r[3] = r[4] + 1;
```

Florida State University

Detecting Equivalent Function Instances

```
sum = 0;
for (i = 0; i < 1000; i++ )
     sum += a [ i ];
```

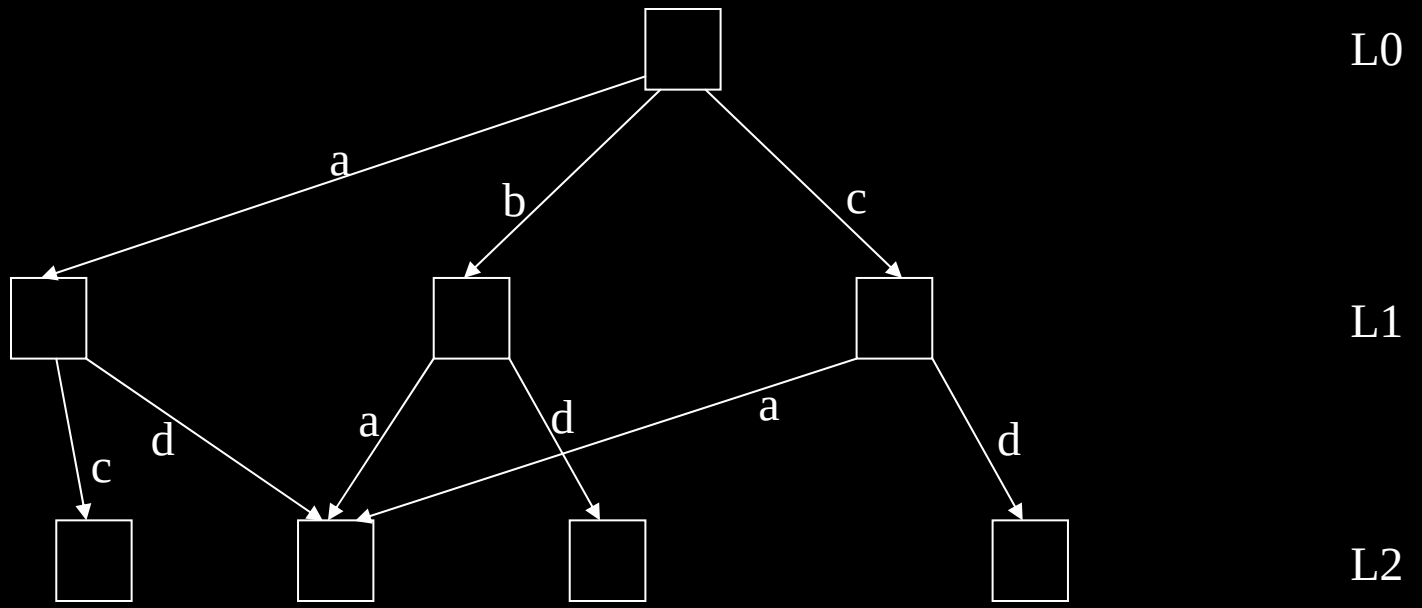
Source Code

Register Allocation before Code Motion	Code Motion before Register Allocation	After Mapping Registers

r[10]=0;
r[12]=HI[a];
r[12]=r[12]+LO[a];
r[1]=r[12];
r[9]=4000+r[12];
L3
r[8]=M[r[1]];
r[10]=r[10]+r[8];
r[1]=r[1]+4;
IC=r[1]?r[9];
PC=IC<0,L3;
	r[11]=0;
	r[10]=HI[a];
	r[10]=r[10]+LO[a];
	r[1]=r[10];
	r[9]=4000+r[10];
	L5
	r[8]=M[r[1]];
	r[11]=r[11]+r[8];
	r[1]=r[1]+4;
	IC=r[1]?r[9];
	PC=IC<0,L5;
		r[32]=0;
		r[33]=HI[a];
		r[33]=r[33]+LO[a];
		r[34]=r[33];
		r[35]=4000+r[33];
		L01
		r[36]=M[r[34]];
		r[32]=r[32]+r[36];
		r[34]=r[34]+4;
		IC=r[34]?r[35];
		PC=IC<0,L01;

Resulting Search Space

- Merging equivalent function instances transforms the tree to a DAG.



Florida State University

Efficient Detection of Unique Function Instances

- Even after pruning there may be tens or hundreds of thousands of unique instances.
- Use a CRC (cyclic redundancy check) checksum on the bytes of the RTLs representing the instructions.
- Used a hash table to check if an equivalent function instance already exists in the DAG.

Florida State University

Techniques to Make Searches Faster

- Kept a copy of the program representation of the unoptimized function instance in memory to avoid repeated disk accesses.
- Also kept the program representation after each active phase in memory to reduce the number of phases applied for each sequence.
- Reduced search time by at least a factor of 5 to 10.
- Out of 111 functions in our benchmark suite we were able to completely enumerate all instances for 109 functions.

Florida State University

Outline

- Experimental framework
- Exhaustive enumeration of the phase order space.
- Search space enumeration results
- Optimization phase interaction analysis
- Making conventional compilation faster
- Future work and conclusions

Florida State University

Search Space Statistics

Function	Insts	Blk	Loop	Instances	Phases	Len	CF	Leaves
start_inp...(j)	1,371	88	2	74,950	1,153,279	20	153	587
parse_swi...(j)	1,228	198	1	200,397	2,990,221	18	53	2365
start_inp...(j)	1,009	72	1	39,152	597,147	16	18	324
start_inp...(j)	971	82	1	64,571	999,814	18	47	591
start_inp...(j)	795	63	1	7,018	106,793	15	37	52
fft_float(f)	680	45	4	N/A	N/A	N/A	N/A	N/A
main(f)	624	50	5	N/A	N/A	N/A	N/A	N/A
sha_trans...(h)	541	33	6	343,162	5,119,947	26	95	2964
read_scan...(j)	480	59	2	34,270	511,093	15	57	540
LZWRea...(j)	472	44	2	49,412	772,864	20	41	159
main(j)	465	40	1	33,620	515,749	17	12	153
dijkstra(d)	354	30	3	86,370	1,361,960	20	18	1168
....	....	....	....	....	....	....	....	....
average	**166.7**	**16.9**	**0.9**	**25,362.6**	**381,857.7**	**12**	**27.5**	**182.9**

Florida State University

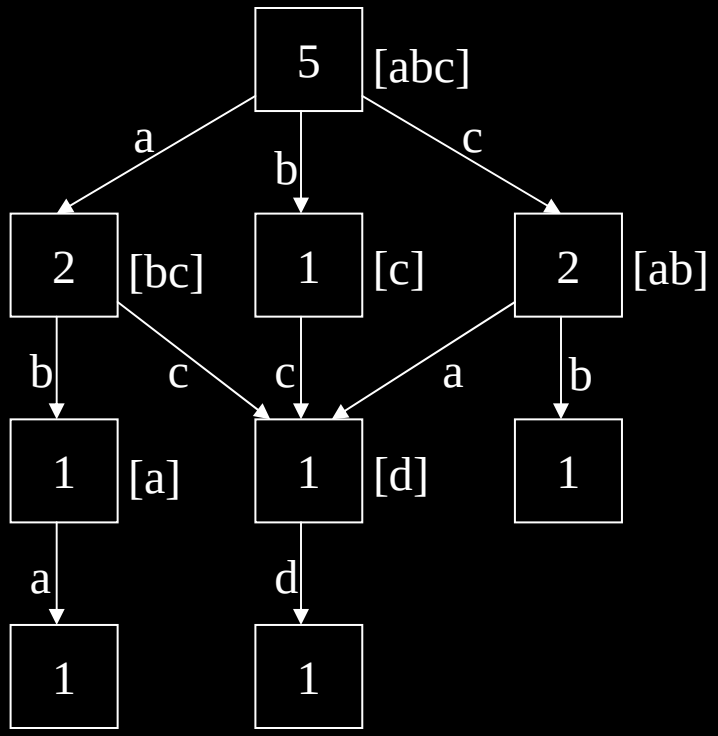
Outline

- Experimental framework
- Exhaustive enumeration of the phase order space.
- Search space enumeration results
- Optimization phase interaction analysis
- Making conventional compilation faster
- Future work and conclusions

Florida State University

Weighted Function Instance DAG

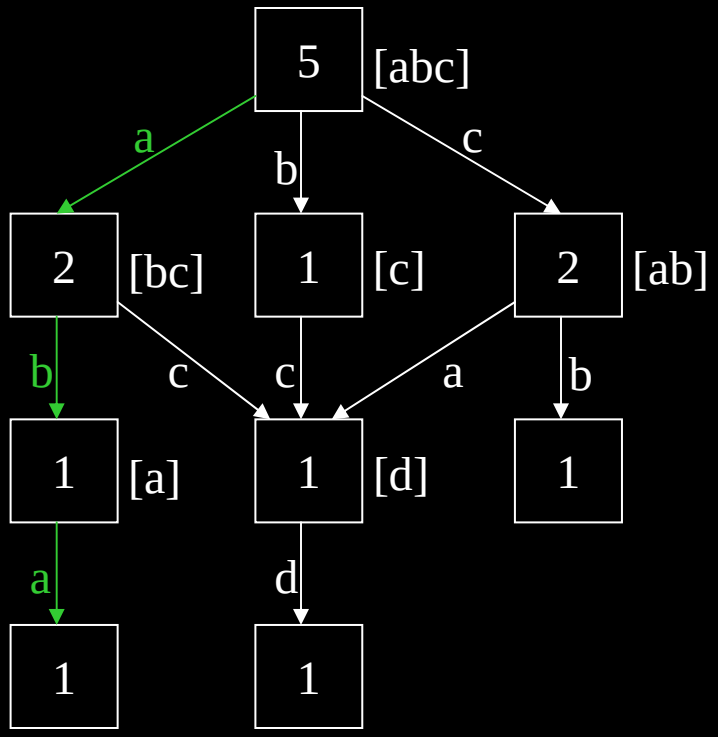
- Each node is weighted by the number of paths to a leaf node.



Florida State University

Enabling Interaction Between Phases

- b enables a along the path a-b-a.



Florida State University

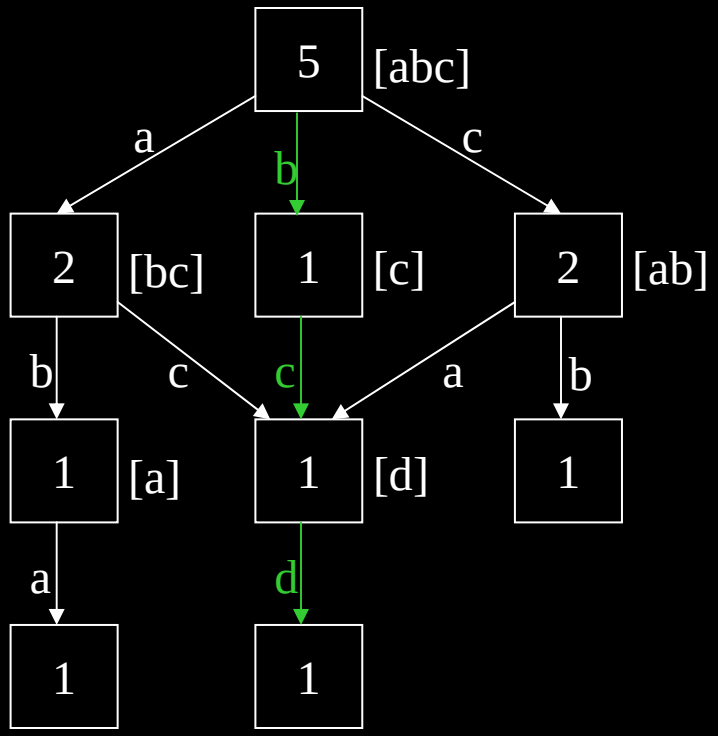
Enabling Probabilities

Ph	St	b	c	d	g	h	i	j	k	l	n	o	q	r	s	u
b	0.62		0.01		0.15	0.06										
c	1.00	0.02			0.23	0.14		0.12	**0.99**	0.72	0.38	0.33	1.00	0.05	0.32	
d																
g		0.01							0.18							0.01
h	0.06		0.7		0.02				0.01	0.03					0.46	
i	0.61	0.01	0.01					0.61								
j	0.03	0.01			0.13											
k			0.01			0.11									0.81	
l	0.59		0.06		0.02	0.01			0.03						0.06	
n	0.42		0.04		0.22	0.01	0.04			0.01		0.01		0.03	0.05	0.03
o	0.87														0.01	
q			0.16												0.08	
r	0.45	0.02			0.15					0.05	0.01					
s	1.00		0.29		0.16	0.23			**0.97**	0.53	0.2		1.00			
u		0.73			0.03											

Florida State University

Disabling Interaction Between Phases

- b disables a along the path b-c-d.



Florida State University

Disabling Probabilities

Ph	b	c	d	g	h	i	j	k	l	n	o	q	r	s	u
b	1.00			0.15		0.02				0.08			0.05		0.31
c		1.00			0.02									0.15	
d			1.00												
g	0.35			1.00		0.19	0.02							0.03	
h		0.01			1.00										
i	0.08			0.06		1.00	0.14						0.14		0.55
j						0.13	1.00						0.49		0.14
k		0.04		0.01				1.00							
l		0.71			0.07			0.30	1.00					0.73	
n	0.33	0.49		0.09	0.25		0.07	0.31	0.53	1.00	0.02			0.33	
o		**1.00**			**1.00**			**1.00**	**1.00**	**1.00**	1.00			0.21	
q												1.00		0.12	
r	0.05			0.01	0.03	0.53							1.00		
s		0.11												1.00	
u	0.08					1.00	0.20								1.00

Florida State University

Disabling Probabilities

Ph	b	c	d	g	h	i	j	k	l	n	o	q	r	s	u
b	1.00			0.15		0.02				0.08			0.05		0.31
c		1.00			0.02									0.15	
d			1.00												
g	0.35			1.00		0.19	0.02							0.03	
h		0.01			1.00										
i	0.08			0.06		1.00	0.14						0.14		0.55
j						0.13	1.00						0.49		0.14
k		0.04		0.01				1.00							
l		**0.71**			0.07			0.30	1.00					**0.73**	
n	0.33	0.49		0.09	0.25		0.07	0.31	0.53	1.00	0.02			0.33	
o		1.00			1.00			1.00	1.00	1.00	1.00			0.21	
q												1.00		0.12	
r	0.05			0.01	0.03	0.53							1.00		
s		0.11												1.00	
u	0.08					1.00	0.20								1.00

Florida State University

Outline

- Experimental framework
- Exhaustive enumeration of the phase order space.
- Search space enumeration results
- Optimization phase interaction analysis
- Making conventional compilation faster
- Future work and conclusions

Florida State University

Faster Conventional Compiler

- We modified the VPO compiler to use enabling and disabling probabilities to decrease compilation time.

```
# p[i]    - current probability of phase i being active
# e[i][j] - probability of phase j enabling phase i
# d[i][j] - probability of phase j disabling phase i

For each phase i do
  p[i] = e[i][st];
While (any p[i] > 0) do
  Select j as the current phase with highest probability of being active
  Apply phase j
  If phase j was active then
    For each phase i, where i != j do
            p[i] += ((1-p[i]) * e[i][j]) - (p[i] * d[i][j])
    p[j] = 0
```

Florida State University

Probabilistic Compilation Results

Function	Old Compilation		Prob. Compilation		Prob. / Old		
	Attempted	Active	Attempted	Active	Time	Size	Speed
start_inp...(j)	233	16	55	14	0.469	1.014	N/A
parse_swi...(j)	233	14	53	12	0.371	1.016	0.972
start_inp...(j)	270	15	55	14	0.353	1.010	N/A
start_inp...(j)	233	14	49	13	0.420	1.003	N/A
start_inp...(j)	231	11	53	12	0.436	1.004	1.000
fft_float(f)	463	28	99	25	0.451	1.012	0.974
main(f)	284	20	73	18	0.550	1.007	1.000
sha_trans...(h)	284	17	67	16	0.605	0.965	0.953
read_scan...(j)	233	13	43	10	0.342	1.018	N/A
LZWReadByte(j)	268	12	45	11	0.325	1.014	N/A
main(j)	270	12	57	14	0.375	1.007	1.000
dijkstra(d)	231	9	43	9	0.409	1.010	1.000
....	....	....	....	....	....	....	....
average	**230.3**	**8.9**	**47.7**	**9.6**	**0.297**	**1.015**	**1.005**

Outline

- Experimental framework
- Exhaustive enumeration of the phase order space.
- Search space enumeration results
- Optimization phase interaction analysis
- Making conventional compilation faster
- Future work and conclusions

Florida State University

Future Work

- Study methods to find more equivalent performing function instances to further reduce the optimization phase order space.
- Evaluate approaches to find the dynamically optimal function instance.
- Improve non-exhaustive searches of the phase order space.
- Study additional methods to improve conventional compilers.

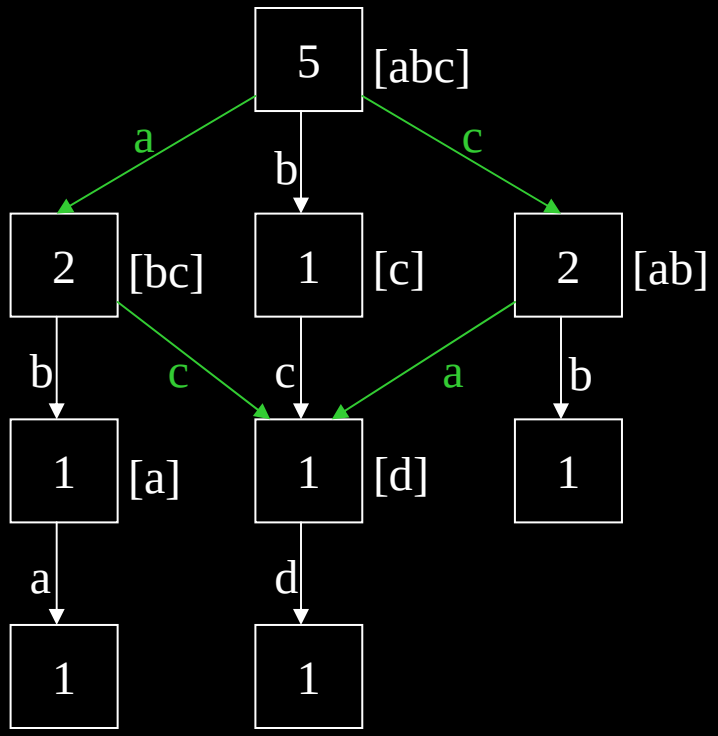
Conclusions

- First work to show that the optimization phase order space can often be completely enumerated (at least for the phases in our compiler).
- First analysis of the entire phase order space to capture various phase probabilities.
- Used phase interaction information to achieve a much faster compiler that still generates comparable code.

Florida State University

Optimization Phase Independence

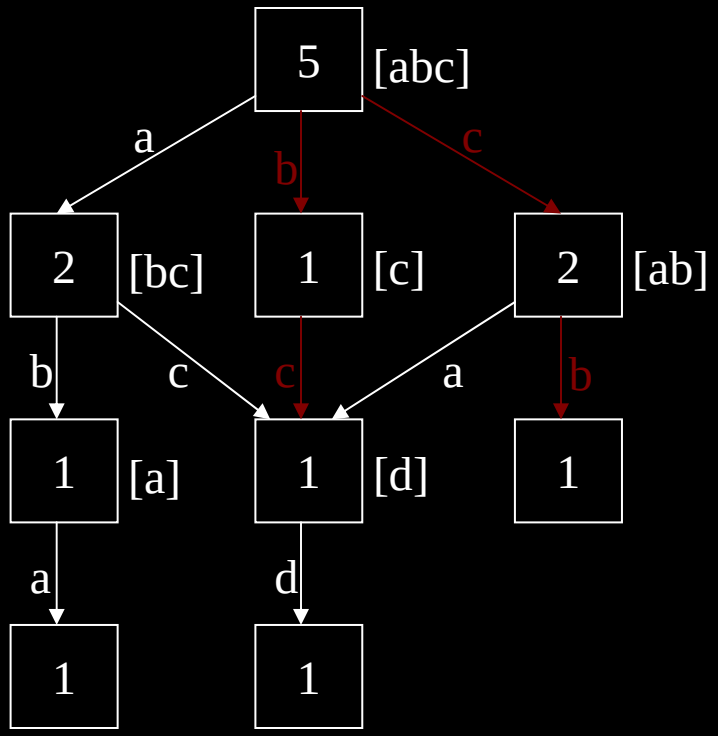
- a-c and c-a are independent.



Florida State University

Optimization Phase Independence

- b-c and c-b are not independent.



Florida State University

Independence Probabilities

Ph	b	c	d	g	h	i	j	k	l	n	o	q	r	s	u
b				0.84		0.94		0.97	0.95	0.82			0.96		0.95
c				0.96	0.91			0.45	0.44	0.65	0.12	0.98	0.99	0.22	
d															
g	0.84	0.96			0.98	0.84			0.96	0.98				0.96	
h		0.91		0.98				0.79	0.95	0.88	0.59		0.98	0.96	
i	0.94			0.84			0.98	0.97	0.96				0.71		0.5
j						0.98							0.97		0.98
k	0.97	0.45			0.79	0.97			0.87	0.81	0.30		0.99	0.82	0.97
l	0.95	0.44		0.96	0.95	0.96		0.87		0.78	0.45			0.45	
n	0.82	0.65		0.98	0.88			0.81	0.78		0.58			0.61	
o		0.12			0.59			0.30	0.45	0.58				0.39	
q		0.98												0.89	
r	0.96	0.99			0.98	0.71	0.97	0.99						0.94	
s		0.22		0.96	0.96			0.82	0.45	0.61	0.39	0.89	0.94		
u	0.95					0.5	0.98	0.97							

Florida State University

Independence Probabilities

Ph	b	c	d	g	h	i	j	k	l	n	o	q	r	s	u
b				0.84		0.94		0.97	0.95	0.82			0.96		0.95
c				0.96	0.91			0.45	0.44	0.65	0.12	0.98	0.99	0.22	
d															
g	0.84	0.96			0.98	0.84			0.96	0.98				0.96	
h		0.91		0.98				0.79	0.95	0.88	0.59		0.98	0.96	
i	0.94			0.84			0.98	0.97	0.96				0.71		0.5
j						0.98							0.97		0.98
k	0.97	0.45			0.79	0.97			0.87	0.81	0.30		0.99	0.82	0.97
l	0.95	0.44		0.96	0.95	0.96		0.87		0.78	0.45			0.45	
n	0.82	0.65		0.98	0.88			0.81	0.78		0.58			0.61	
o		0.12			0.59			0.30	0.45	0.58				0.39	
q		0.98												0.89	
r	0.96	0.99			0.98	0.71	0.97	0.99						0.94	
s		0.22		0.96	0.96			0.82	0.45	0.61	0.39	0.89	0.94		
u	0.95					0.5	0.98	0.97							

VPO Optimization Phases (cont...)

- Register assignment (assigning pseudo registers to hardware registers) is implicitly performed before the first phase that requires it.
- Some phases are applied after the sequence
  - fixing the entry and exit of the function to manage the run-time stack
  - exploiting predication on the ARM
  - performing instruction scheduling