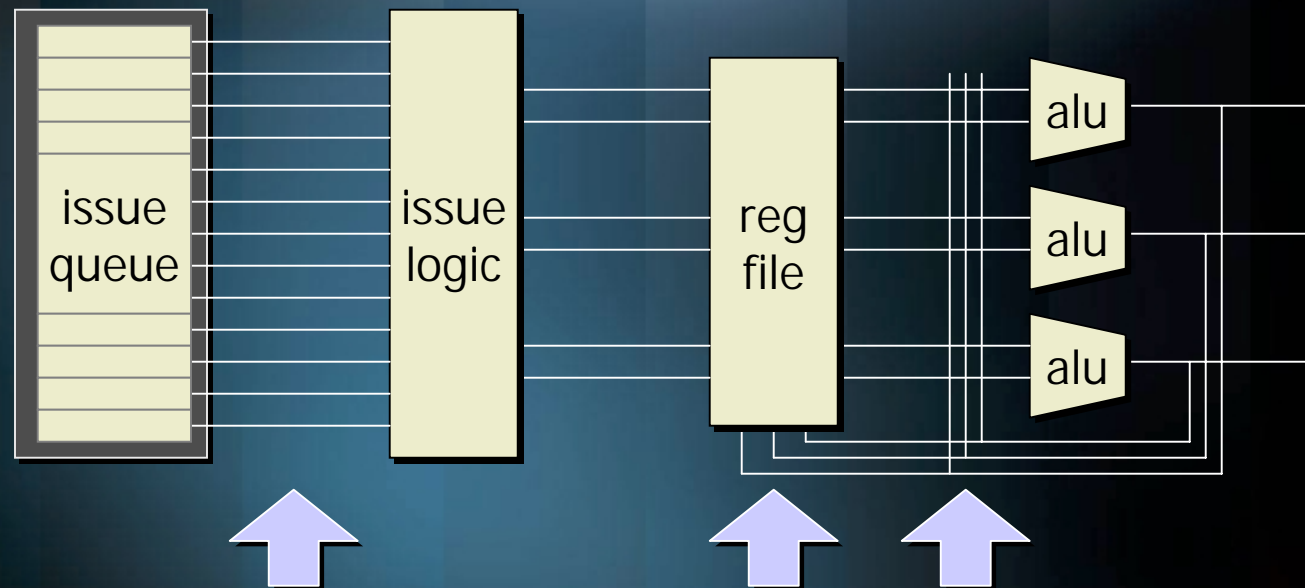


On the Extraction and Analysis of Prevalent Dataflow Patterns

Workshop on Workload Characterization (WWC-7)
Austin, TX
October 25, 2004

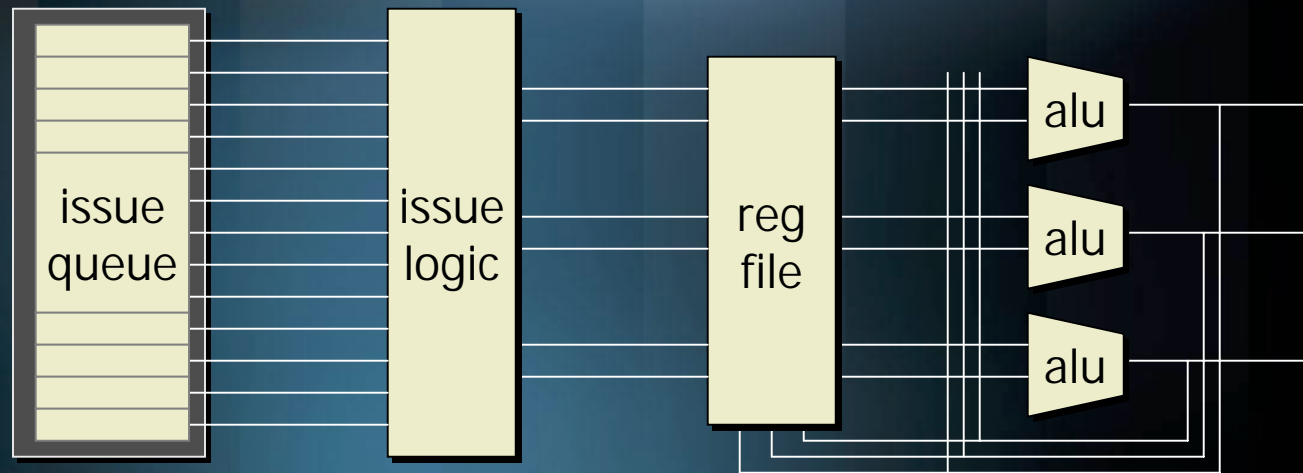
Peter G. Sassone
D. Scott Wills
Georgia Institute of Technology
{sassone, scott.wills} @ ece.gatech.edu

Motivation



- Modern out-of-order processors are designed for the **worst-case**.
 - Issue logic checks every value in every waiting instruction for readiness.
 - Register files need enough ports for every issuing instruction to read both values.
 - Every ALU output is bypassed to every ALU input.

Motivation



- The logic and communication that once were free now restricts maximum clock frequency.
- Thankfully, architectural research has recently focused more on common-case performance.

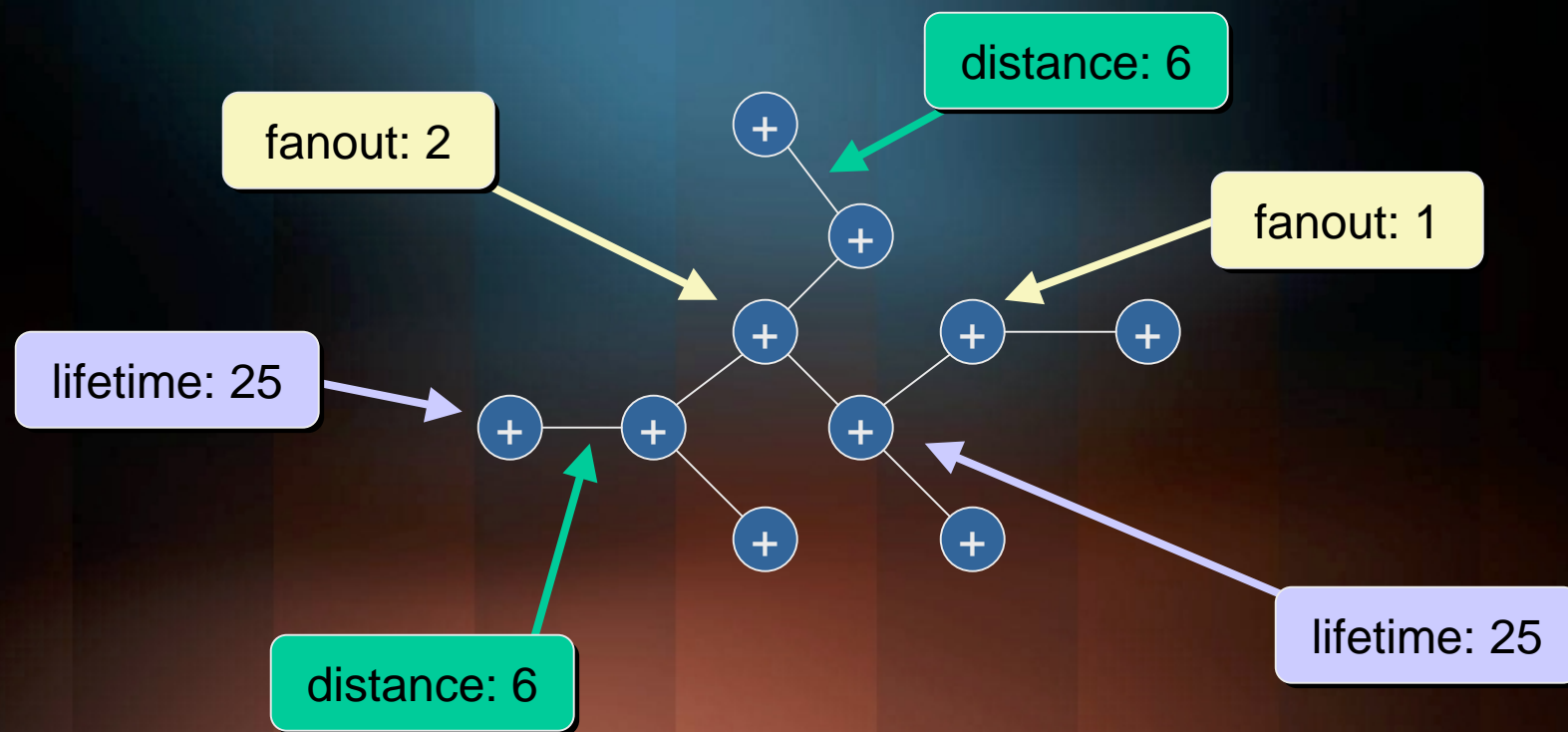
But what is the common case?

Outline

- ➔ • Motivation
- Communication Patterns
- Related Work
- Pattern Extraction
 - Intractability of problem
 - Extraction Insights
 - CPX tool
- Most Popular Patterns
 - Spec2000int
 - MediaBench
 - All benchmarks
 - Coverage
- Conclusion

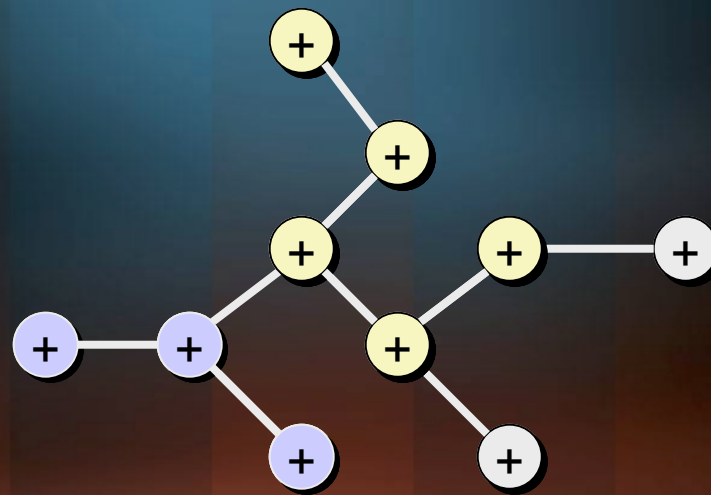
Communication patterns

- The common case of operand movement is traditionally determined circumstantially.
 - Producer-consumer distance
 - Functional unit fan-out
 - Operand lifetime



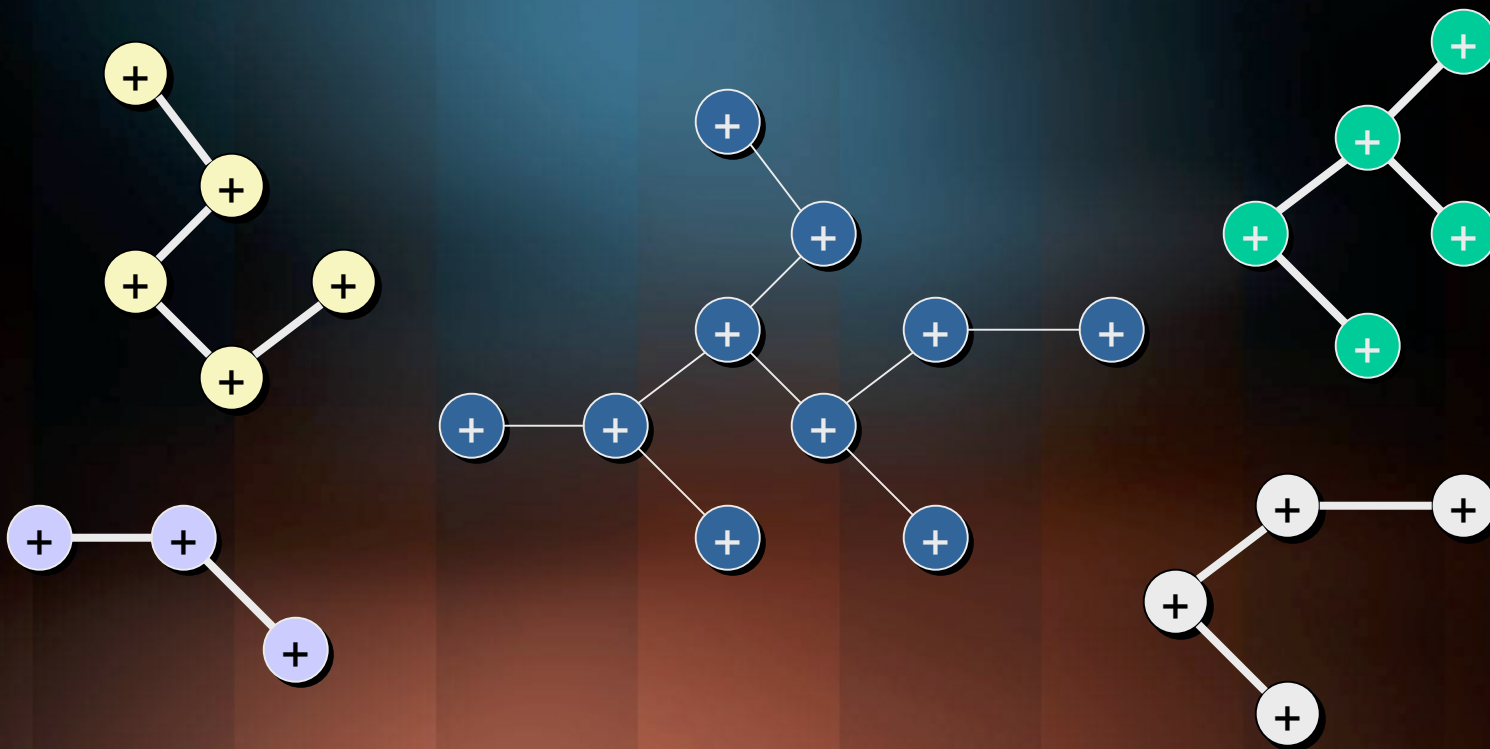
Communication patterns

- Instead, we focus on discovering the common case communication patterns themselves.
- Allows architects to visualize functional unit communication.
 - And provide benefits to other fields



Communication Patterns

- In the end, we can reduce 90% of Spec2000int and 75% of MediaBench dynamic insns to a set of only ten communication primitives!



Outline

- Motivation
- ➔ • Communication Patterns
- Related Work
- Pattern Extraction
 - Intractability of problem
 - Extraction Insights
 - CPX tool
- Most Popular Patterns
 - Spec2000int
 - MediaBench
 - All benchmarks
 - Coverage
- Conclusion

Pattern Applicability

- Researchers in compilers and code compression have long identified repeated code segments, usually termed idioms.
- These repeatable patterns are the result of:
 - Source code repetition
 - Iterative nature of source code
 - Limited instruction sets
 - Compiler transformations
- Proposed uses:
 - Code compression
 - ISA design
 - Cluster steering heuristics

Patterns of a Different Sort

- Spadini et al. identified idioms for each of the Spec2000int suite.
 - Only 10 idioms (with 5 instructions or less) per benchmark could cover over 25% of the dynamic instructions.
- Arrujo et al. divided binaries into operation patterns and input sets.
 - Removing entropy reduced Spec95 binary size by over 40%.
- However, these previously identified idioms are instruction-specific, not functional-unit-specific.

Outline

- Motivation
- Communication Patterns
- ➔ • Related Work
- Pattern Extraction
 - Intractability of problem
 - Extraction Insights
 - CPX tool
- Most Popular Patterns
 - Spec2000int
 - MediaBench
 - All benchmarks
 - Coverage
- Conclusion

Difficulties of the problem

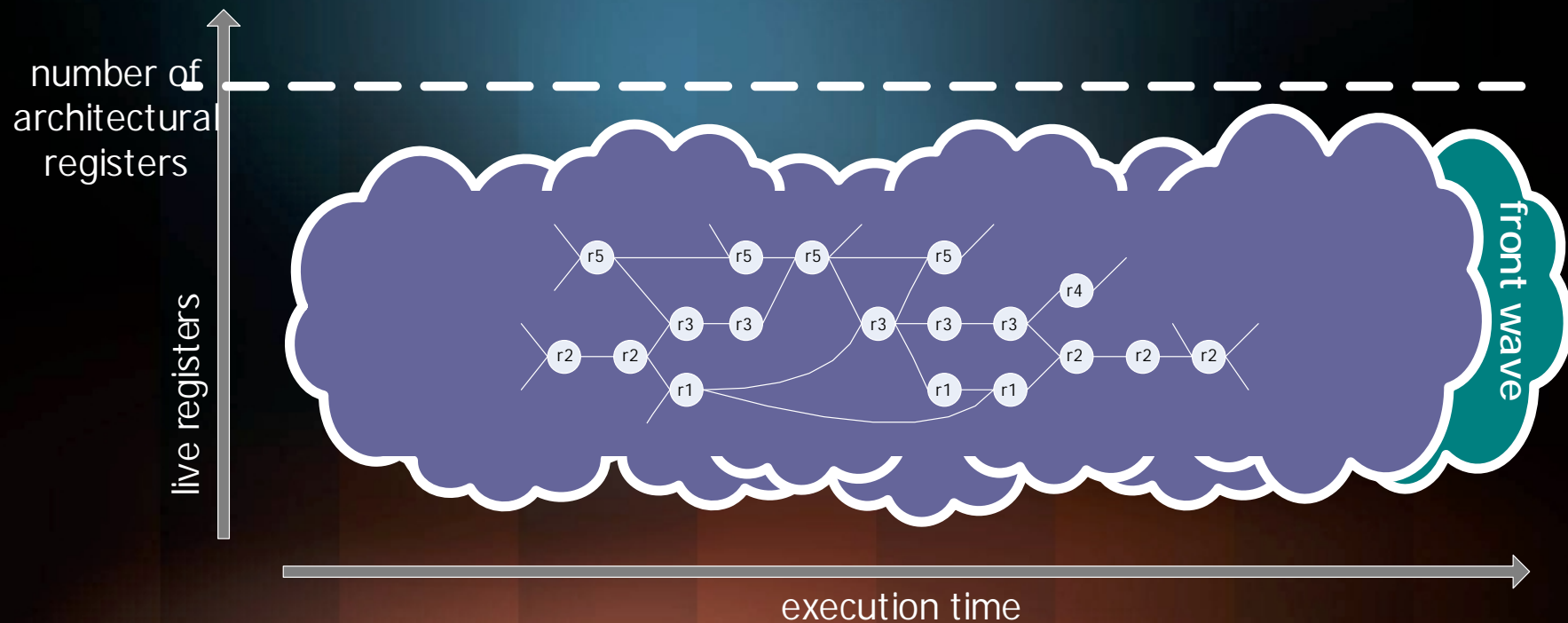
- Problem 1:
 - The size of the dataflow graphs are **tremendous**—several hundred million nodes minimum.
- Problem 2:
 - Any algorithm for identifying common patterns reduces to subgraph isomorphism—**NP complete**.
- Problem 3:
 - There is **no easy metric**—would we prefer less common but bigger patterns, or smaller but more common patterns?

Difficulties of the problem

- Problem 1:
 - The size of the dataflow graphs are **tremendous**—several hundred million nodes minimum.
- Observations:
 - Dataflow graphs are not arbitrarily connected—they are quite **narrow**.
 - The maximum width is only the number of architectural registers (ie., 32).
 - The entire DFG does not need to be held in memory, only **front wave**.

Difficulties of the problem

- Problem 1:
 - The size of the dataflow graphs are **tremendous**—several hundred million nodes minimum.

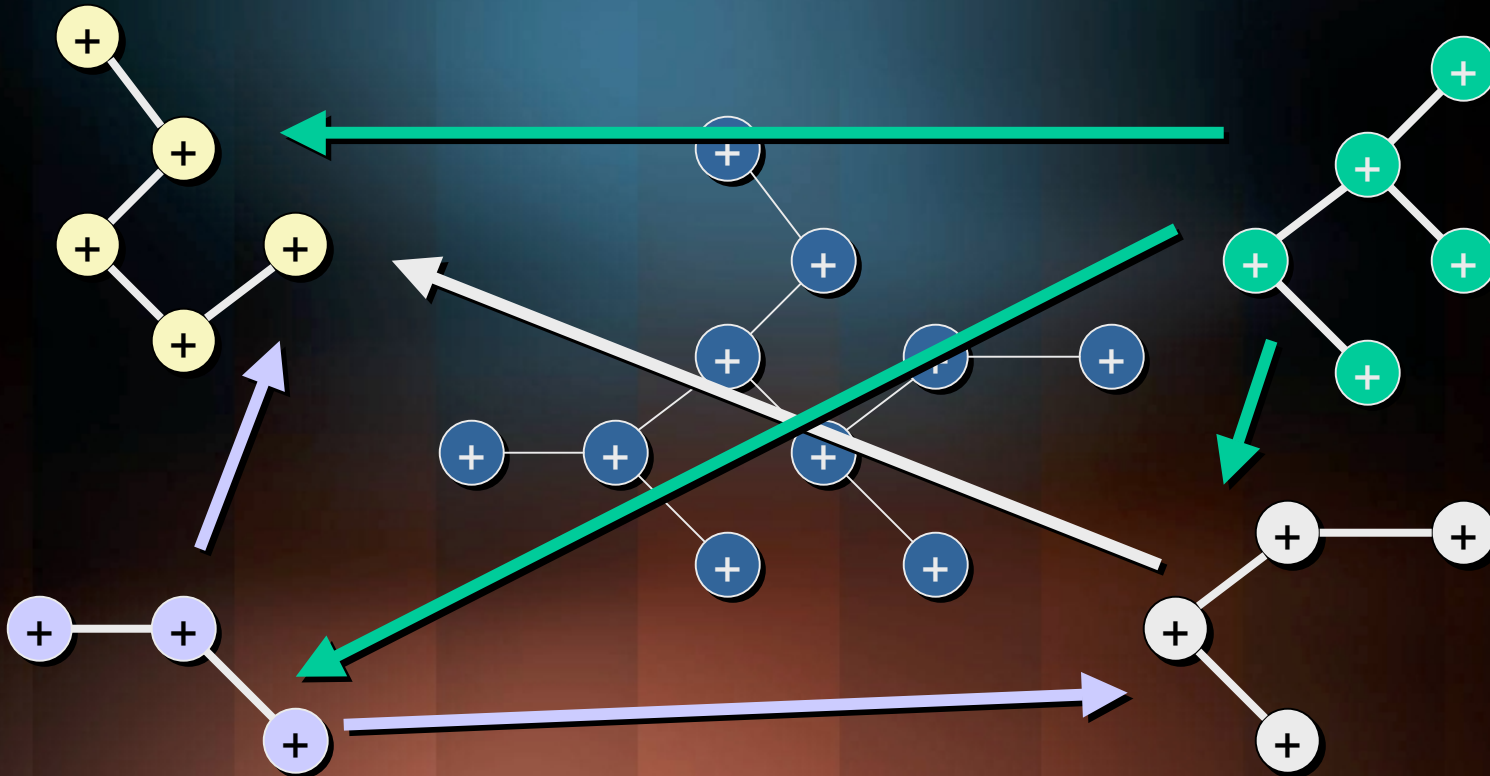


Difficulties of the problem

- Problem 2:
 - Any algorithm for identifying common patterns reduces to subgraph isomorphism—NP complete.
- Observations
 - Have to brute force, but be smart about it.
 - The interesting patterns are small (ten or fewer insns).
 - Sampling is simple and effective.
 - Avoid pair-wise matching.

Difficulties of the problem

- Problem 2:
 - Any algorithm for identifying common patterns reduces to subgraph isomorphism—NP complete.



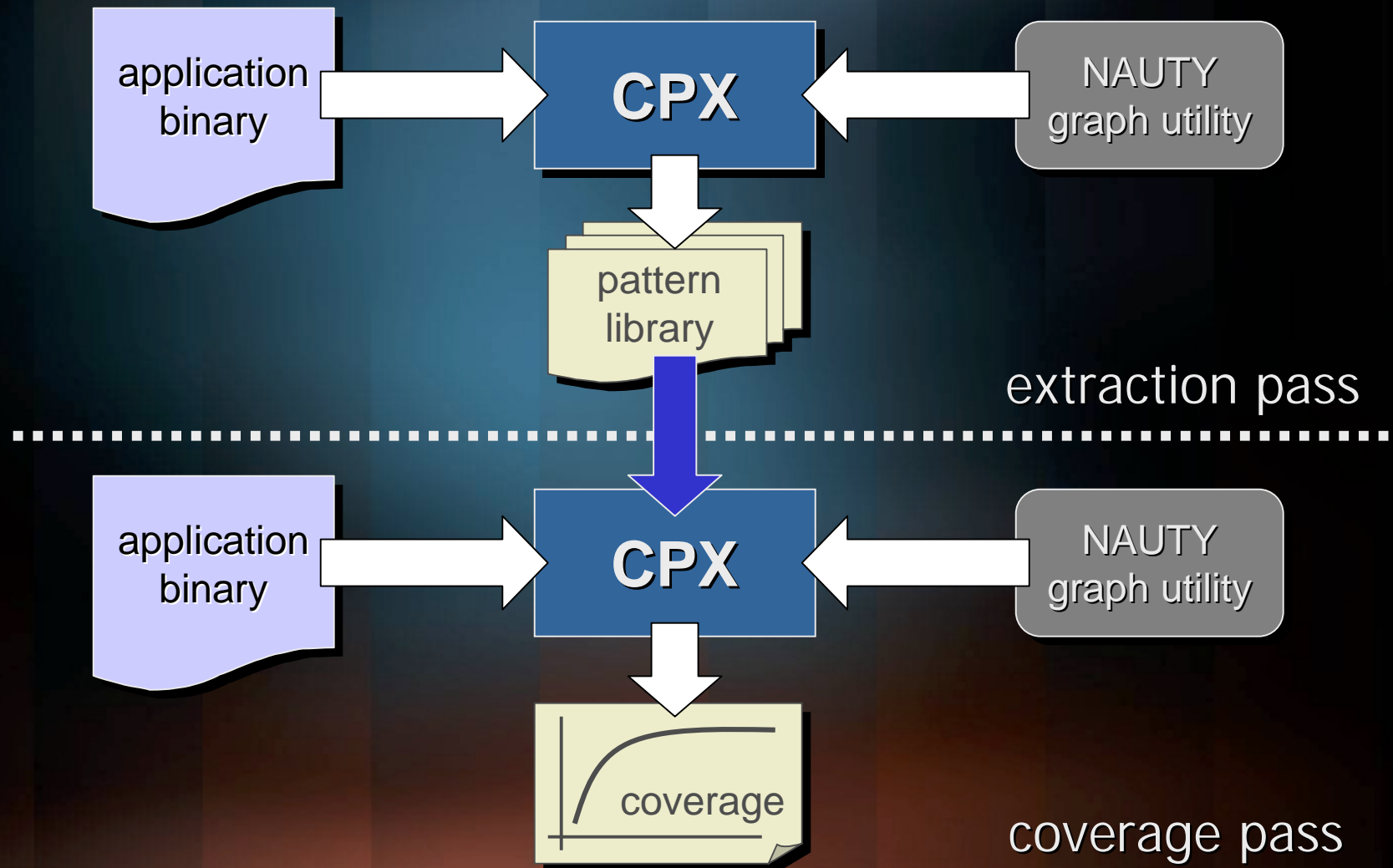
Difficulties of the problem

- Problem 3:
 - There is **no easy metric**—would we prefer less common but bigger patterns, or smaller but more common patterns?
- Observations:
 - We are really interested in the patterns that incorporate the **most instructions**.
 - Use **popularity**: pattern frequency * size
 - Most popular patterns are those which are instructions are most likely to be a part of.

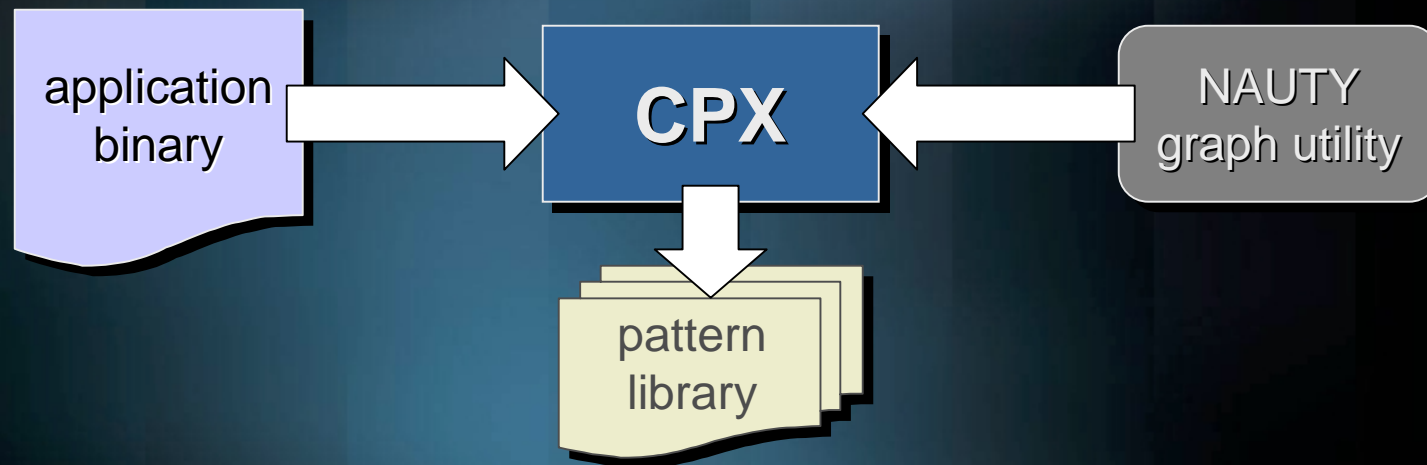
CPX

- Using these observations, we developed CPX, Communication Pattern Extractor
- **Rapid**: analyzes over 100K patterns per second on our testbed, and does not slow down over time.
- **Small**: memory footprint less than 10MB.
- **Balanced**: uses popularity metric to balance size and frequency of patterns.

CPX algorithm

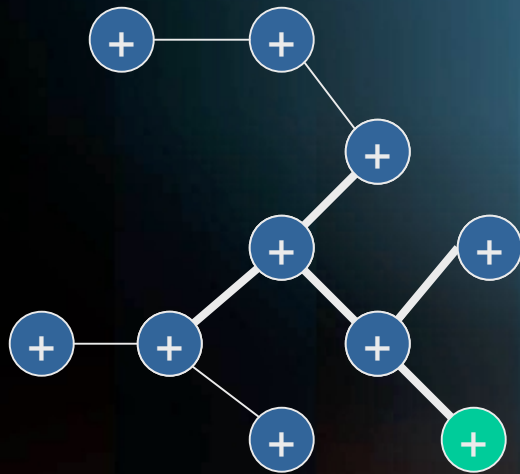


CPX extraction pass



- Searches for patterns *while simulating app.*
- For each instruction simulated:
 - Enqueue new instruction to DFG front wave
 - Dequeue old instruction from DFG front wave
 - Enumerate all patterns which include newest inst
 - Determine hash code for each pattern (or sample)
 - Check pattern against library hashtable
- Produces list of found patterns as *output.*

CPX extraction example

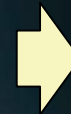


DFG front wave



new instruction

Graph Hasher

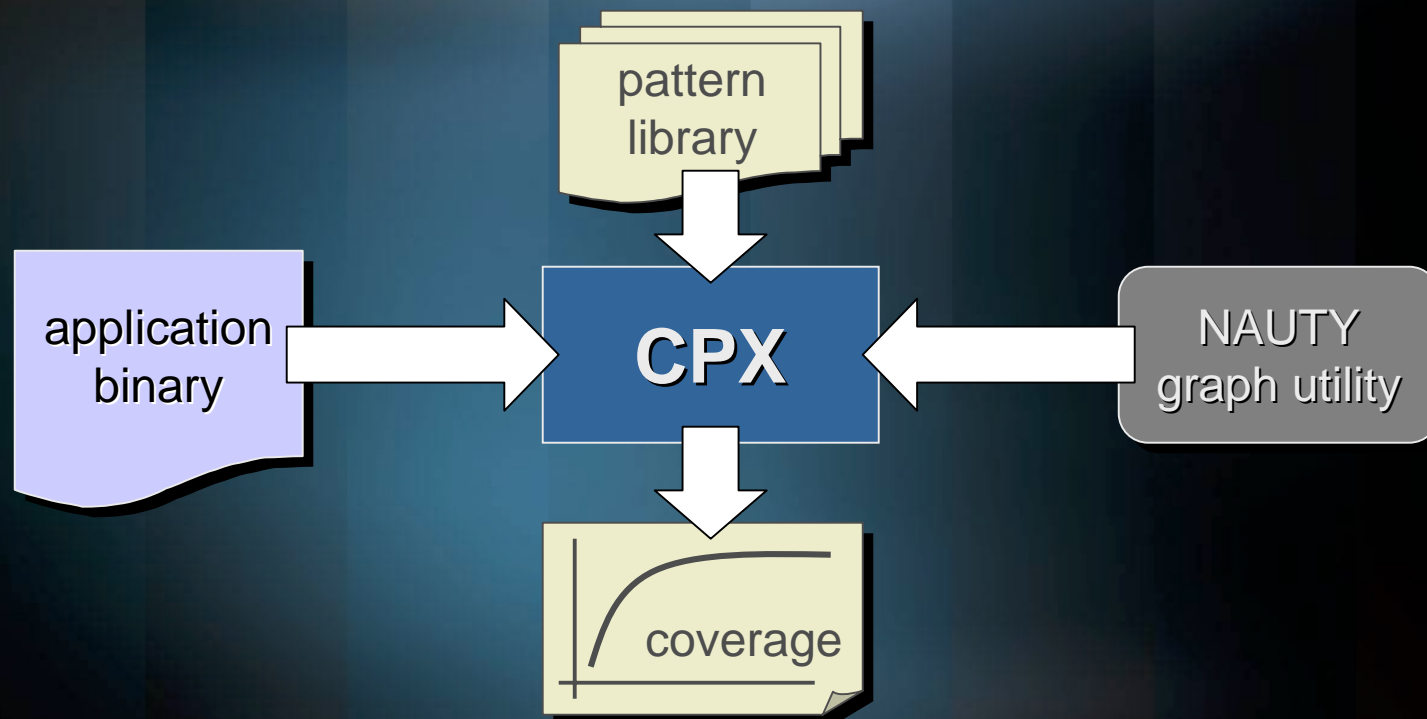


47A7DB2331C22

hash code	count
23B3E87AA81D9	8
A391BF934112E	10
FE826A712C090	21
BB2781CA73803	2
721B7CD271F01	5
47A7DB2331C22	17

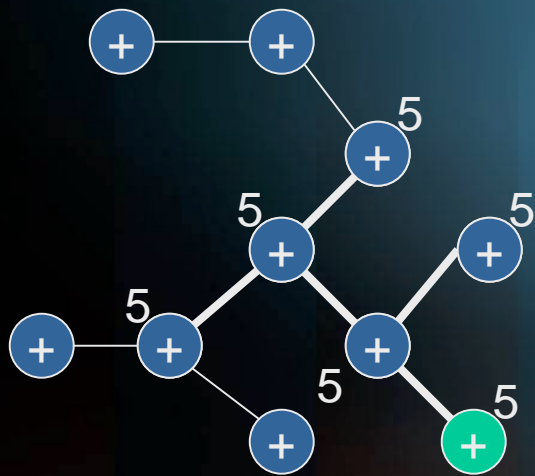
pattern library

CPX coverage pass



- Similar to extraction pass, but takes library in as input.
- Each enumerated pattern is checked against the library.
- Instruction is covered by the most popular pattern which can incorporate it.
- Coverage CDF is final output.

CPX coverage example



DFG front wave

Graph Hasher

47A7DB2331C22

hash code count rank

hash code	count	rank
932CD7210DA01	52	1
62CC820EF0128	44	2
C701037DDD326	29	3
FE826A712C090	21	4
47A7DB2331C22	17	5
A391BF934112E	10	6
23B3E87AA81D9	8	7
721B7CD271F01	5	8
BB2781CA73803	2	9

pattern library



new instruction

Outline

- Motivation
- Communication Patterns
- Related Work
- ➔ • Pattern Extraction
 - Intractability of problem
 - Extraction Insights
 - CPX tool
- Most Popular Patterns
 - Spec2000int
 - MediaBench
 - All benchmarks
 - Coverage
- Conclusion

Experimental setup

- CPX is implemented w/ SimpleScalar 3.0.
- NAUTY 2.2 graph library is used for pattern hashing.
- Runtime results are on an Intel Xeon 2.4GHz, 512MB, Redhat Linux.
- Extraction parameters:
 - Patterns are 8 or less instructions.
 - Input and output ordering does not matter.
 - Patterns do not span basic blocks.
 - Sampling rate is set to 1% for extraction.

Benchmarks

	Benchmark	CPU Hours	Total Patterns	Unique Patterns
Spec2000int	164.gzip	5	74 billion	1066
	175.vpr	4	70 billion	4617
	176.gcc	10	254 billion	4333
	181.mcf	3	46 billion	3319
	197.parser	6	120 billion	3089
	255.vortex	21	528 billion	3484
	256.bzip2	3	54 billion	867
	Total		52	1145 billion
MediaBench	adpcm-decode	9	88 billion	1874
	adpcm-encode	16	170 billion	1007
	jpeg-decode	8	100 billion	2891
	jpeg-encode	7	104 billion	2524
	epic-decode	3	38 billion	3497
	epic-encode	5	92 billion	770
	g721-decode	4	54 billion	1678
	g721-encode	5	74 billion	1968
	mpeg2-decode	10	84 billion	2448
	mpeg2-encode	13	132 billion	5298
	pegwit-decode	16	284 billion	1242
	pegwit-encode	22	314 billion	2779
	Total		117.6	1534 billion
All	Total	169.4	2679 billion	8615

Instruction Types

iALU

integer ALU instruction

branch

branch predicate

EA

effective address computation

load

load word (without EA)

store

store word (without EA)

found in top patterns

integer multiply

integer division

floating point addition

floating point multiplication

floating point division

floating point square root

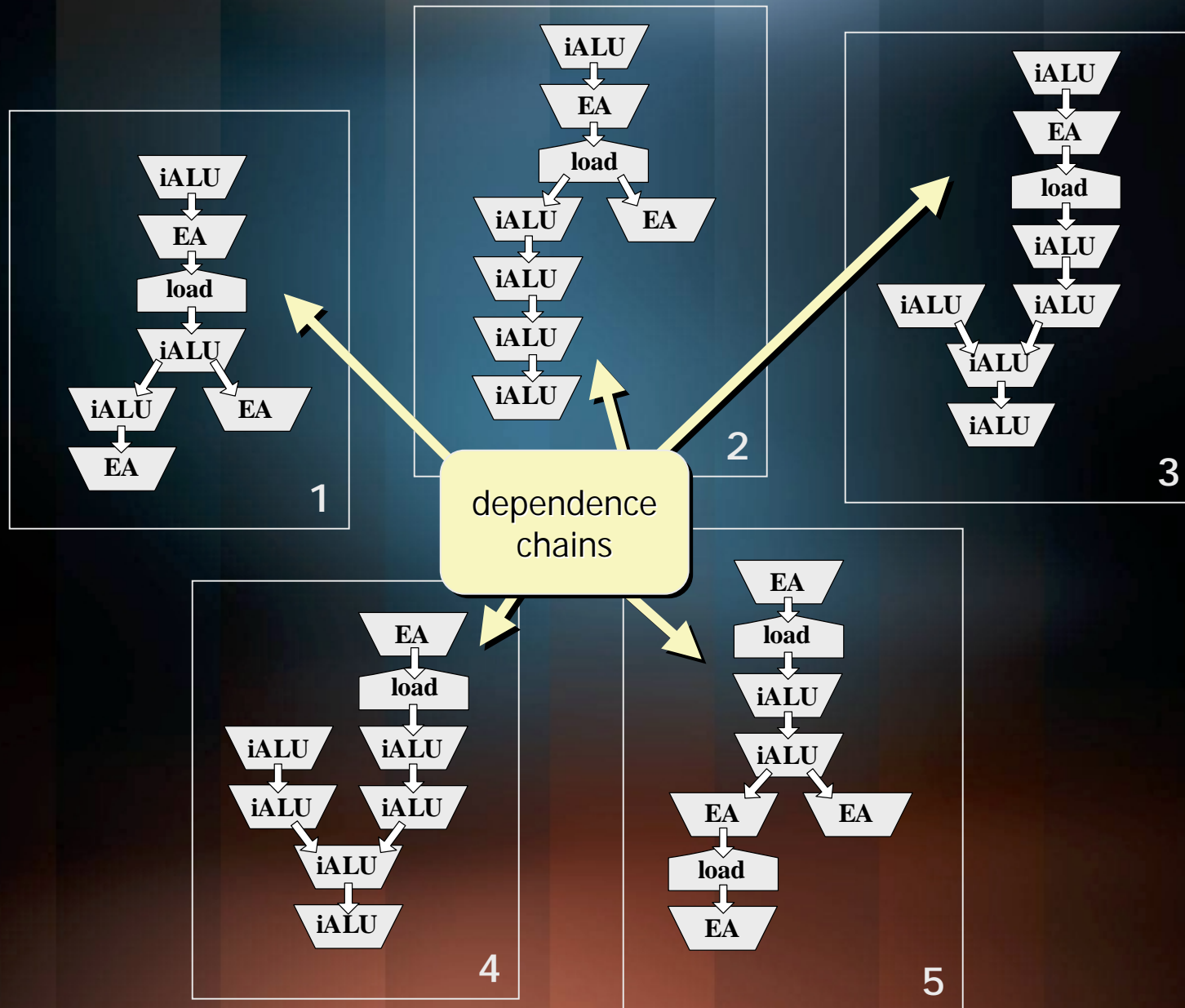
floating point comparison

floating point conversion

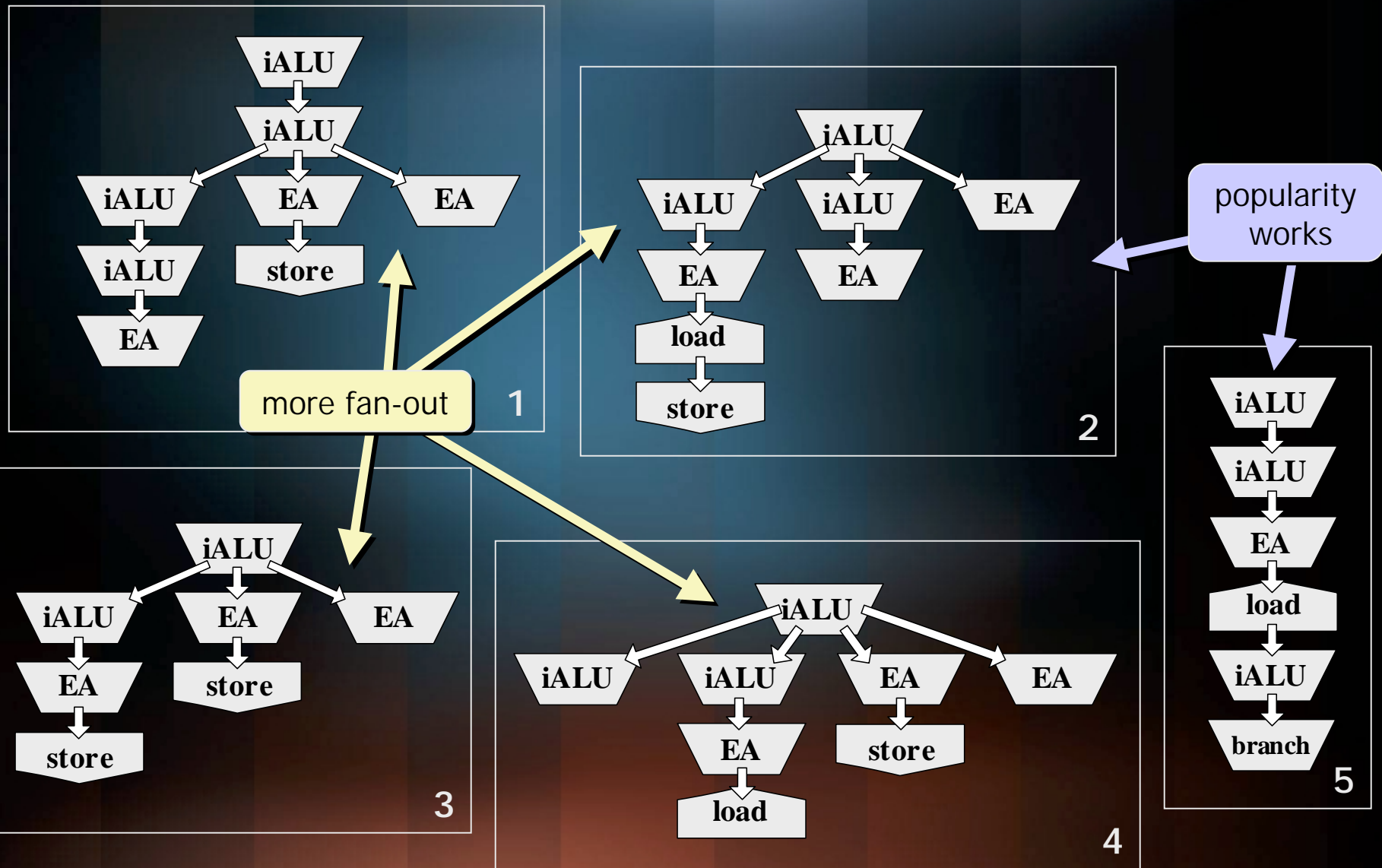
control jump

not in top patterns

MediaBench patterns



Top Spec2000int patterns



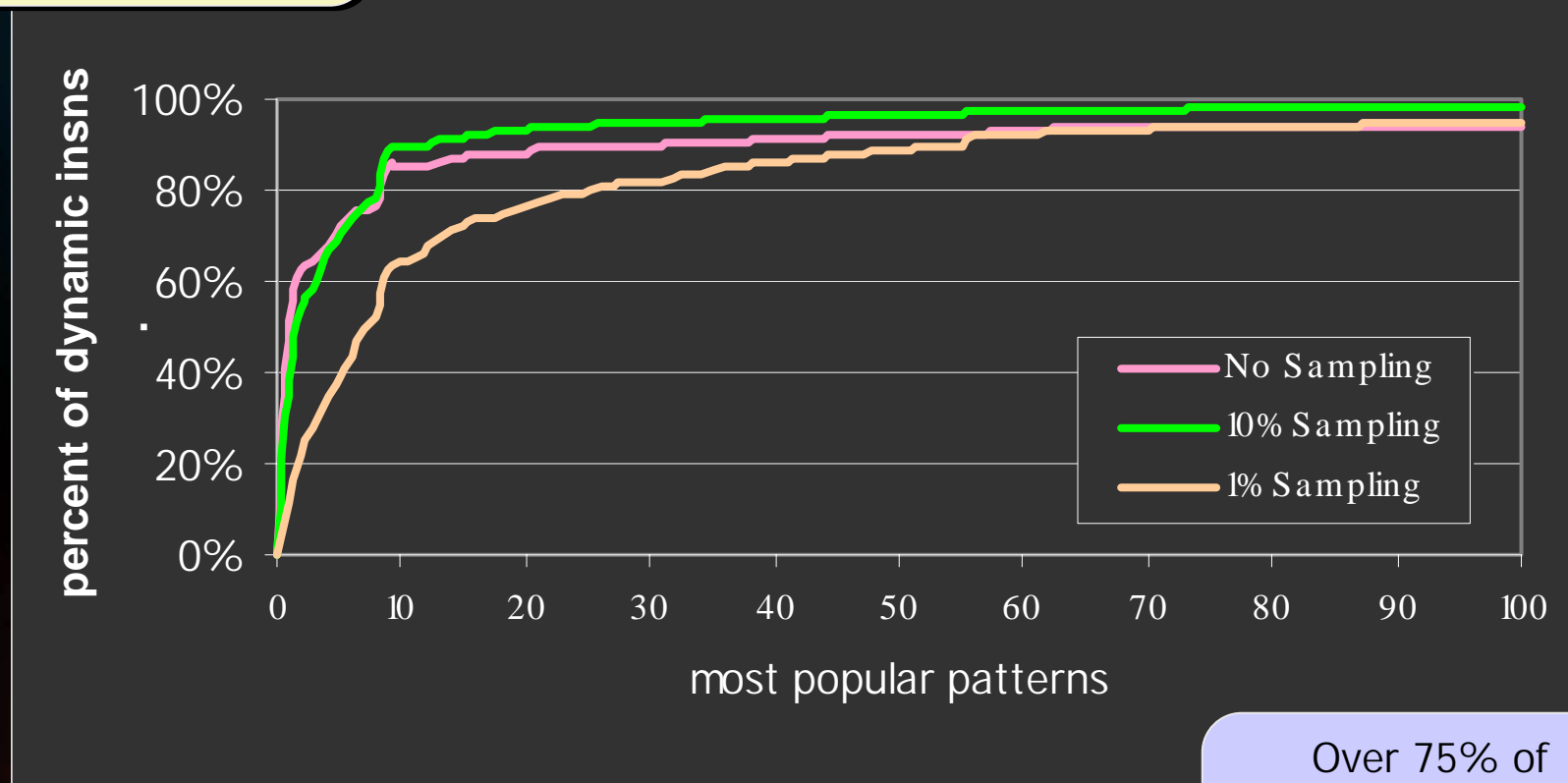
Coverage results

- By feeding the top ten overall patterns (shown in paper) into CPX, we determine coverage.
- We also vary the sampling rate to determine the effect on the coverage CDF.
 - When sampling, CPX uses the assumption that patterns **randomly overlap** to estimate actual coverage rates.
 - As some patterns are **highly correlated**, coverage sampling is prone to some error.
 - However, this phenomenon does not affect extraction sampling—rates from 1% to 100% produce the **same top patterns** shown earlier.

Coverage results

Over 90% of Spec2000 dynamic insns covered by top 10 patterns

Pattern Coverage CDF



Over 75% of MediaBench dynamic insns covered by same 10 patterns

Outline

- Motivation
- Communication Patterns
- Related Work
- Pattern Extraction
 - Intractability of problem
 - Extraction Insights
 - CPX tool
-  • Most Popular Patterns
 - Spec2000int
 - MediaBench
 - All benchmarks
 - Coverage
- Conclusion

Benefits

- Architecture research can benefit from **visualizing** the average case in operand movement.
- Quantify the motivation for research changing **worst-case** processor design.
- **Code compression** by removing pattern entropy.

Future Work

- Study the effect of compilers on patterns.
 - Compilers repetitively use the same transformations for code statements (ie, for accessing an array element).
- Hardware pattern detection, providing dynamic cache compression or reconfigurable execution opportunities.
- Hardware pattern prediction, allowing reconfiguration before whole patterns have been seen.

Thank You