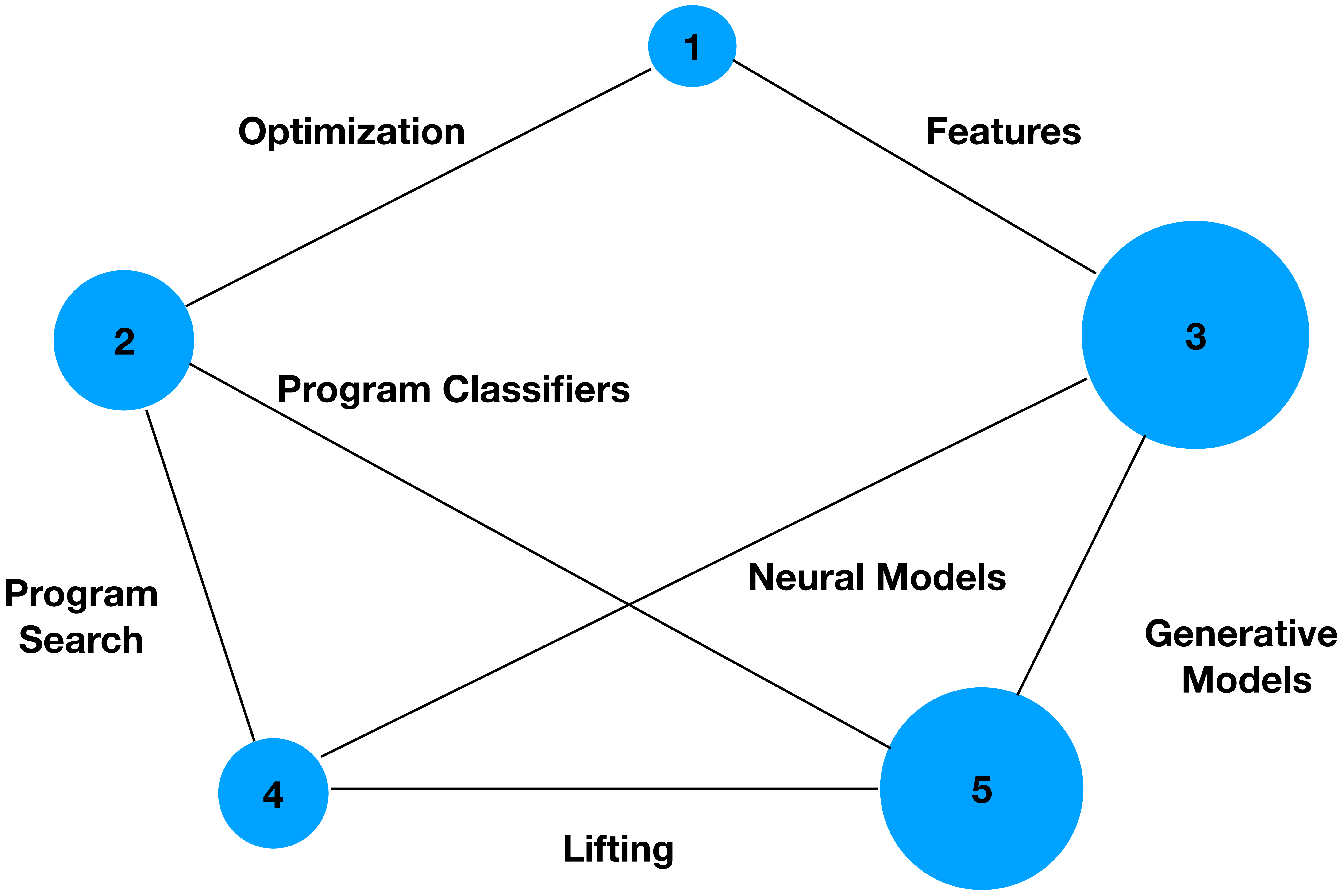
Rethinking compilation: L1

Alexander Braukmann, Jordi Armengol Estape, Jose Wesley Magalhaes.
Michael O'Boyle, Jackson Woodruff

Overview

- This lecture: Motivation and survey of auto-tuning/machine learning for compilers
- L2: Program rewriting schemes - e-graphs and equality saturation
- L3: Program embeddings and Graph Neural Networks
- L4: Program synthesis and neural synthesis
- L5: Neural Machine Translation,Transformers and Large language models


1
Optimization
Features
2
3
Program Classifiers
Neural Models
Program
Search
Generative
Models
4
5
Lifting

What is compilation

Why do we need new technicues

Automation

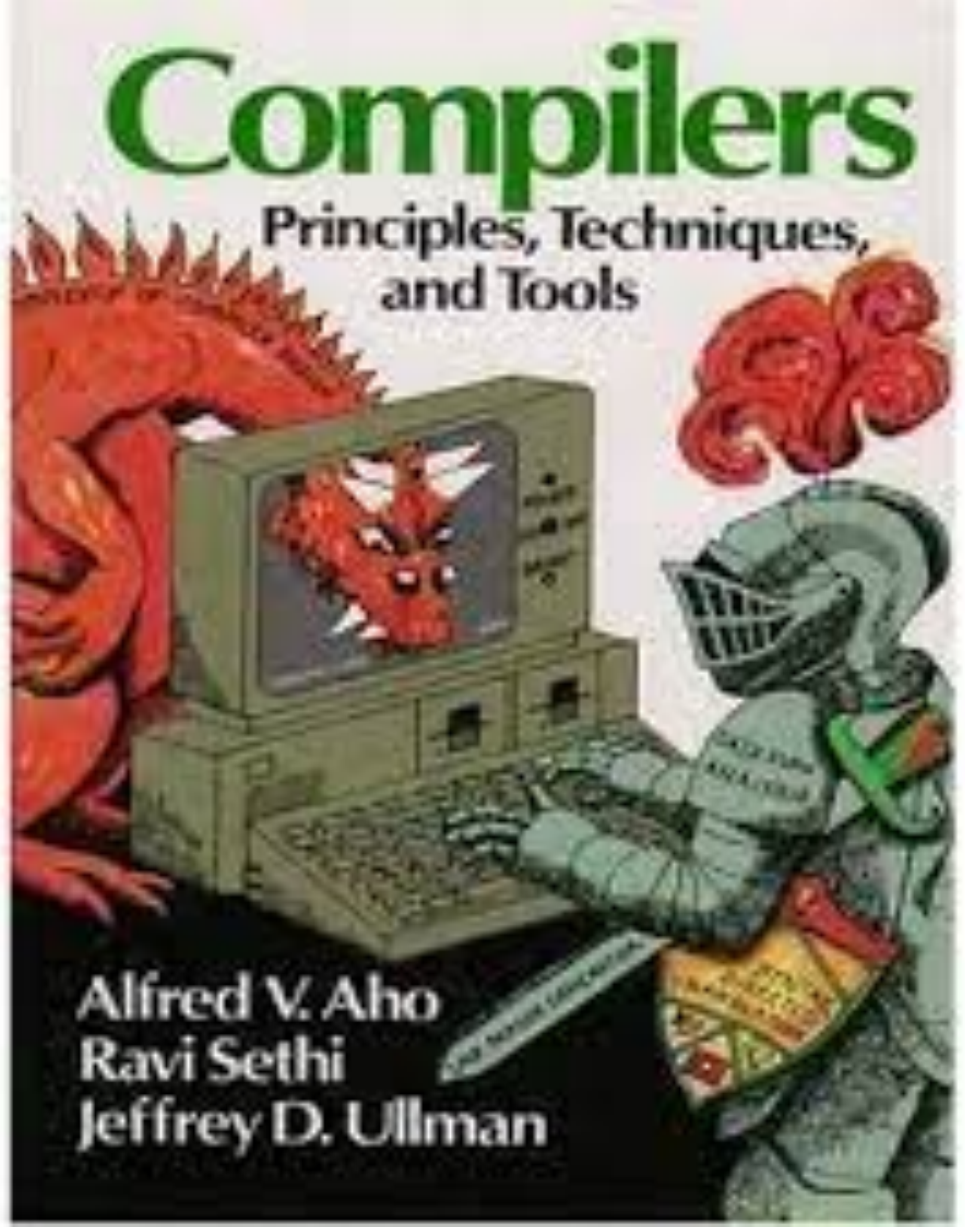
Search/ Auto-tuning/ Iterative compilation

ML for compilation

Features, models and applications

Summary

What is Compilation?

C →  → **x86**

Original Source

2

```
void add(int *list, int val, int n) {
  int i;
  for (i = 0; i < n; ++i) {
    list[i] += val;
  }
}
```

GCC O3:

4

```
 .globl add
 .type add, @function
add:
.LFB0:
 .cfi_startproc
 endbr64
 movq %rdi, %rcx
 testl %edx, %edx
 jle .L1
 leal -1(%rdx), %eax
 cmpl $2, %eax
 jbe .L6
 movq %rdi, %rax
```

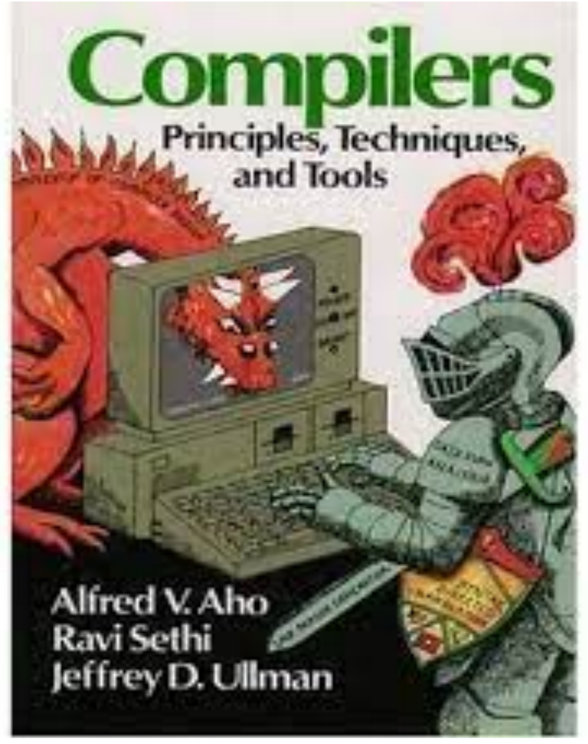
```
.L3:
 movslq %edi, %rax
 leal 1(%rdi), %r8d
 salq $2, %rax
 addl %esi, (%rcx,%rax)
 cmpl %r8d, %edx
 jle .L1
 addl $2, %edi
```

What is compiliation?

Translation - must be correct

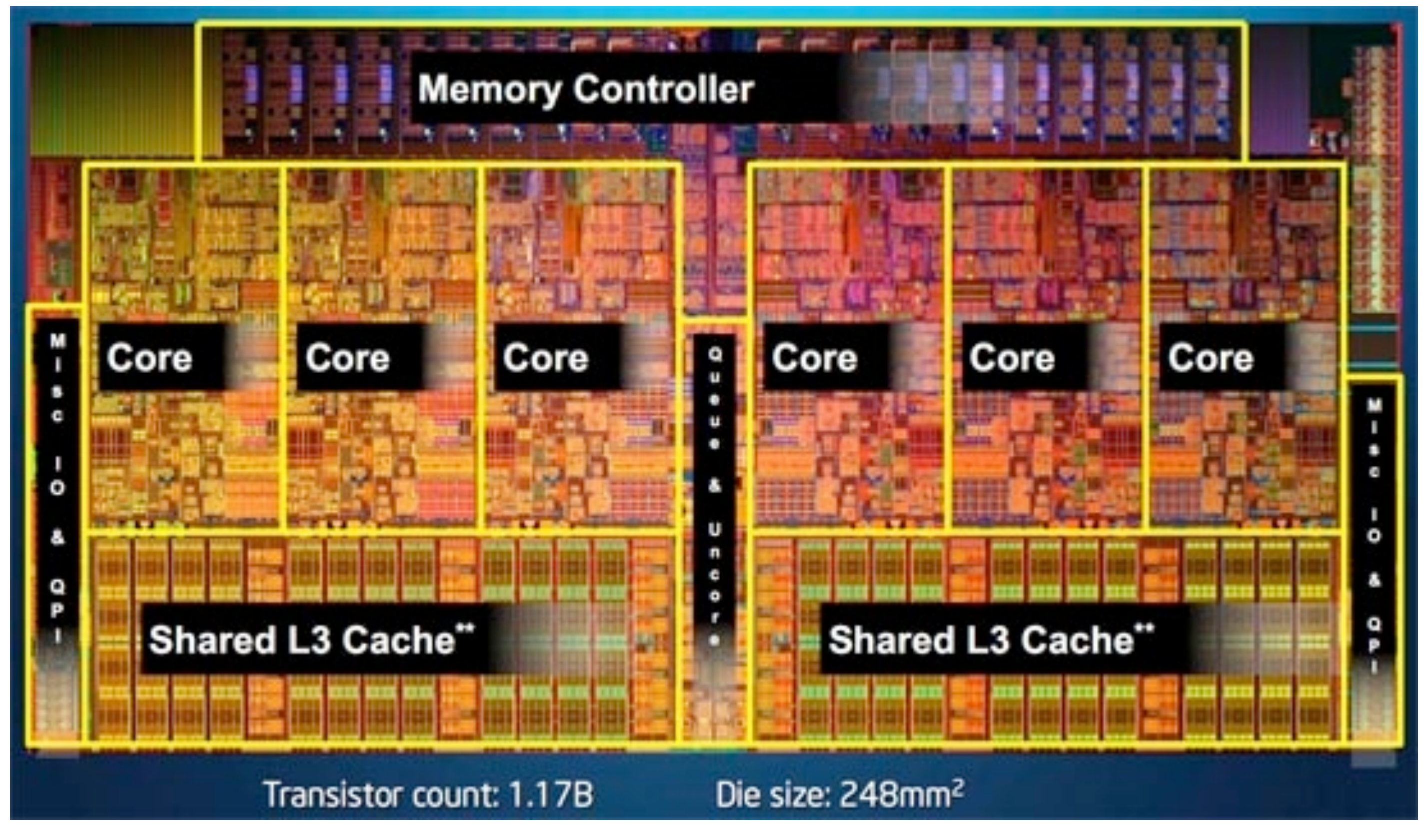
Optimisation: go faster, smaller, cooler.

- Hide complexity, machines are not Von Neumann

C →  → **x86**

Exploit architecture features

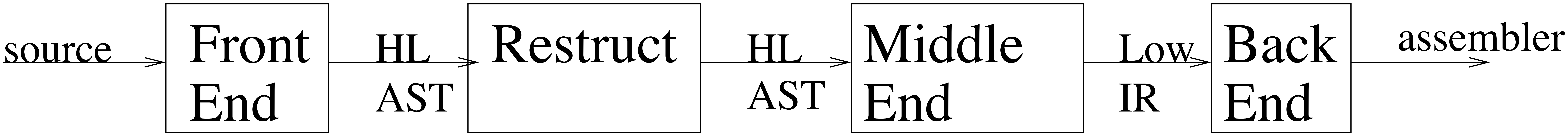
- Parallelism + Memory management



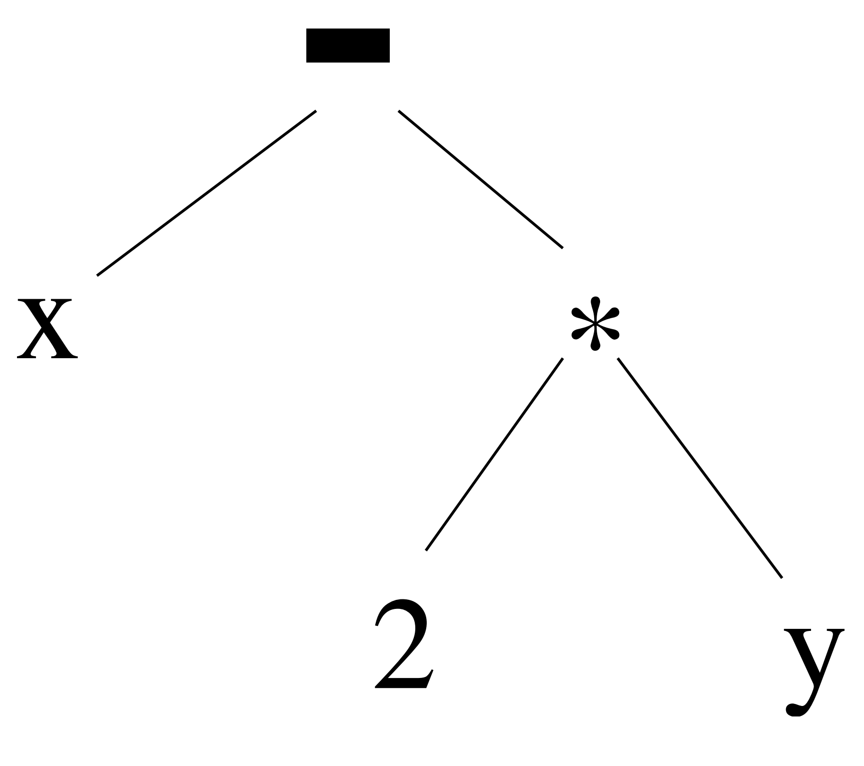
Gap between peak and actual performance widening

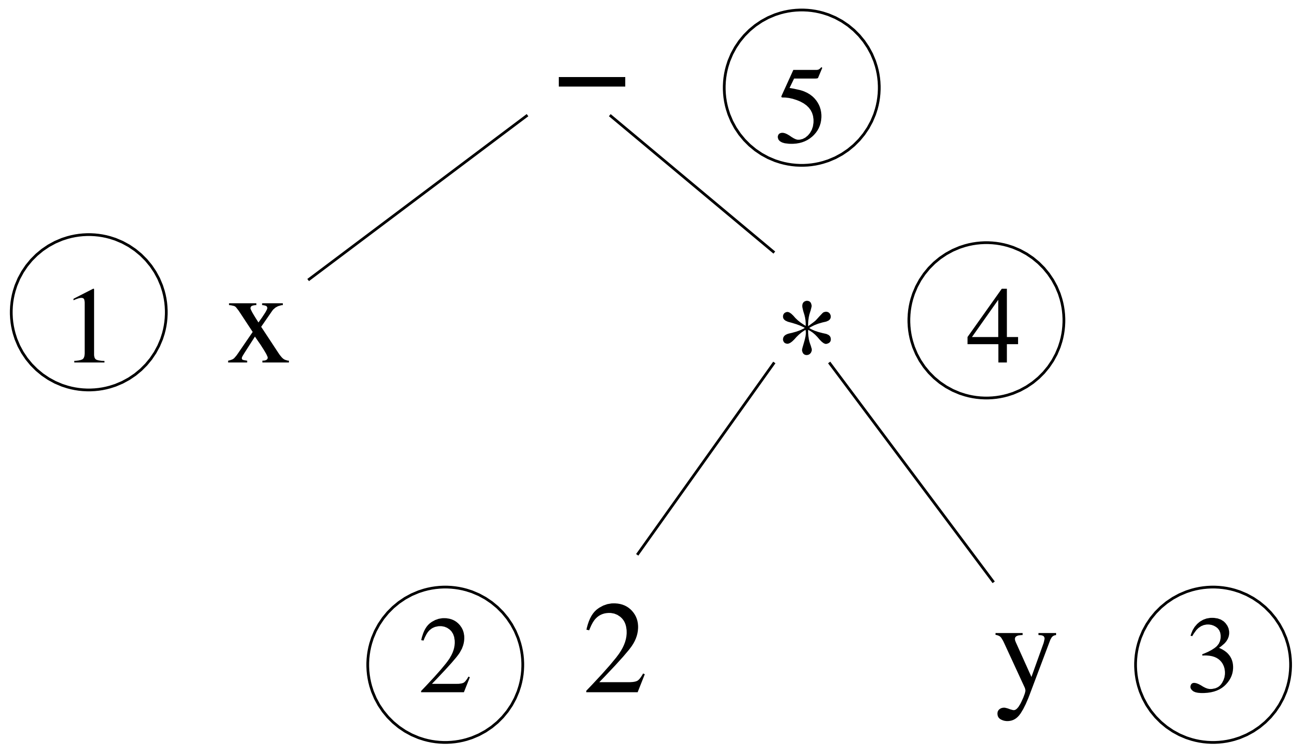
- can compilers help?

What is compiliation?



$$x - 2 * y$$






```
loadI @x -> r1          1
loadA0 r0,r1 -> r1      1
loadI 2 -> r2           2
loadI @y -> r3          3
loadA0 r0,r3 ->r3       3
mult r2,r3 -> r3        4
sub r1,r3->r3           5
```

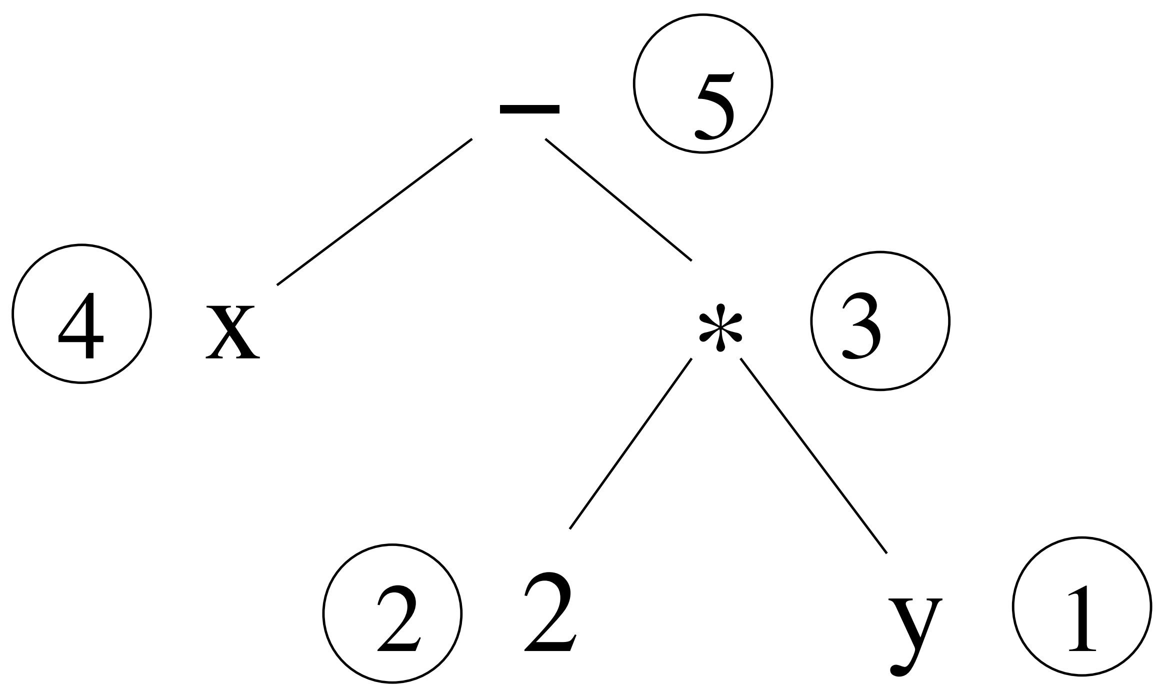
3 registers used

What is compilation?

Generate more efficient code -eliminate redundancy

a = b*c +d	t = b*c
e = 2-b*c	a = t +d
	e = 2- t

Different traversal - less registers




```
loadI @y -> r1
loadA0 r0,r1 -> r1
loadI 2 -> r2
mult r2,r1 -> r1
loadI @x -> r2
loadA0 r0,r2->r2
sub r2,r1->r2
```

[3]

What is compilation

Why do we need new techniques

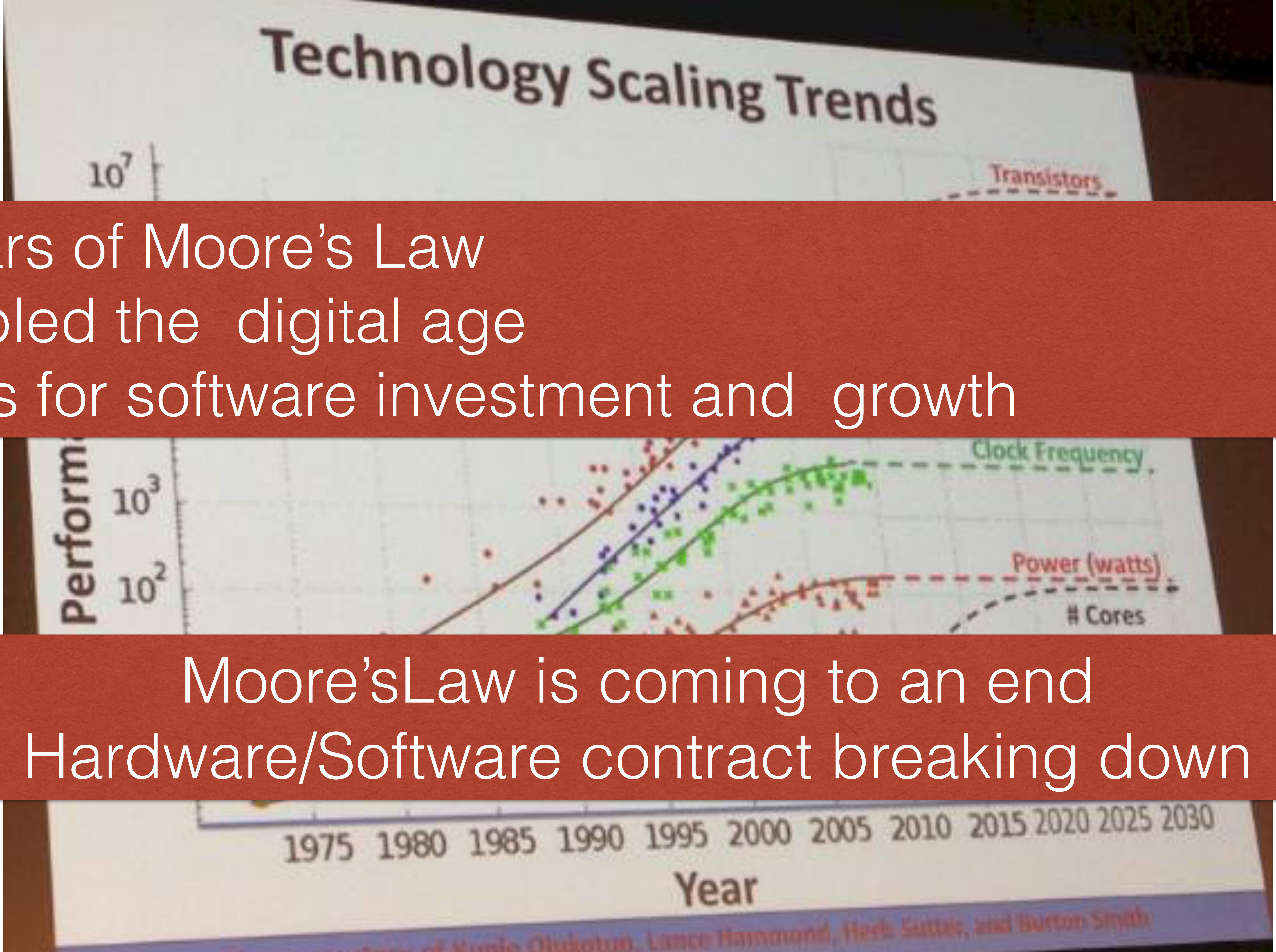
Automation

Search/ Auto-tuning/ Iterative compilation

ML for compilation

Features, models and applications

Summary



50 years of Moore’s Law

- Enabled the digital age
- Basis for software investment and growth

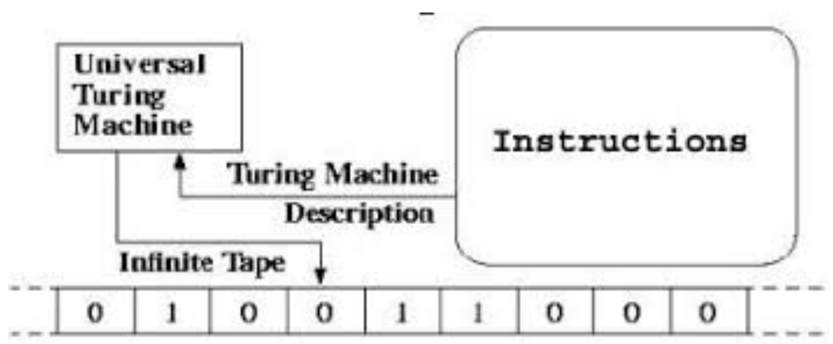
Moore’sLaw is coming to an end

Hardware/Software contract breaking down

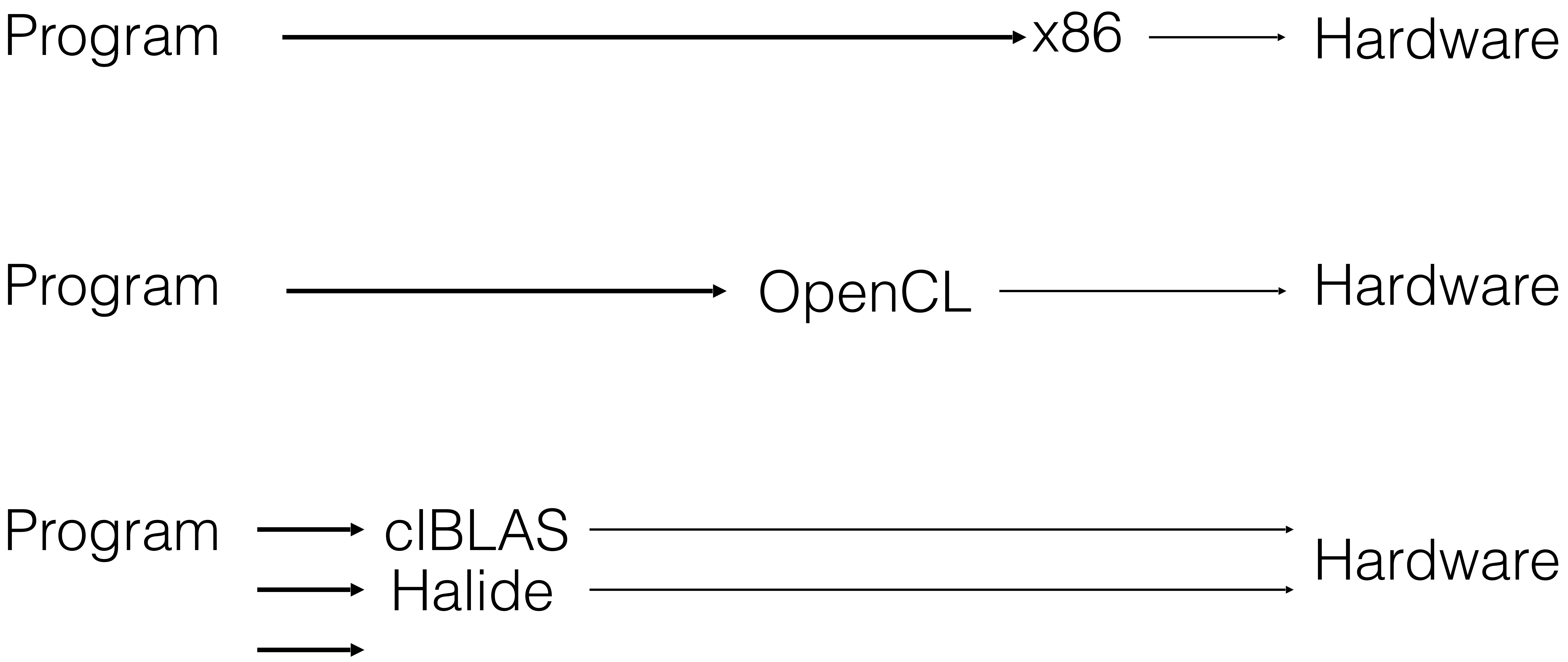
Hardware/software contract breaking down

Technology trends means
- Hardware specialised or heterogenous

Software cannot fit on new hardware



Heterogeneous crisis
- hardware stalls as software cannot fit



Constant change means any solution must work for any API, any DSL
On both sides of interface. **Need to automate**

What is compilation

Why do we need new techniques

Automation

Search/ Auto-tuning/ Iterative compilation

ML for compilation

Features, models and applications

Summary

Automation

1950s,

- auto-programming to auto-optimisation

2005 onwards:

- Software-gap due to multicores.

2010 onwards:

- Rapidly changing hardware

1990s to 2010s

- Auto-Tuning/ML due to poor compiler performance

Automation 1990s to 2010s
The case for evidence based approaches
including search and predictive models

Tiling and Unrolling. What are the best values?

UNROLLING

```
Do i = 1, 100
 a(i) = i
Enddo
```

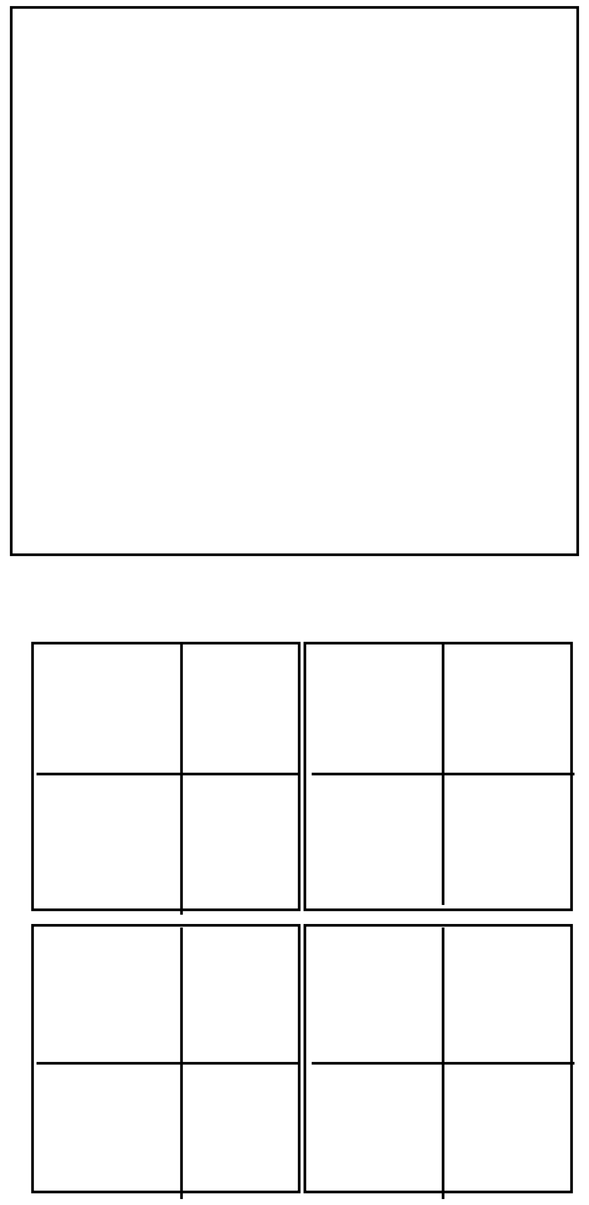
```
Do i = 1, 100, 3
 a(i) = i
 a(i+1) = i+1
 a(i+2) = i+2
Enddo

Do i = 100,100
 a(i) = i
Enddo
```

Tiling = Strip-Mine + Interchange

```
Do i =1, N
  Do j = 1,N
    a(i,j) = a(i,j) +b(i)
  Enddo
Enddo
```

```
Do i =1, N
  Do j = 1,N,s
    Do jj = j,j+s-1
      a(i,jj) = a(i,jj)+b(i)
    Enddo
  Enddo
Enddo
```

```
Do i =1, N,s
  Do j = 1,N,s
   Do ii = i, i+s-1
     Do jj = j,j+s-1
      a(ii,jj) = a(ii,jj) +b(ii)
     Enddo
   Enddo
  Enddo
Enddo
```



MxM: Tiling and Unrolling

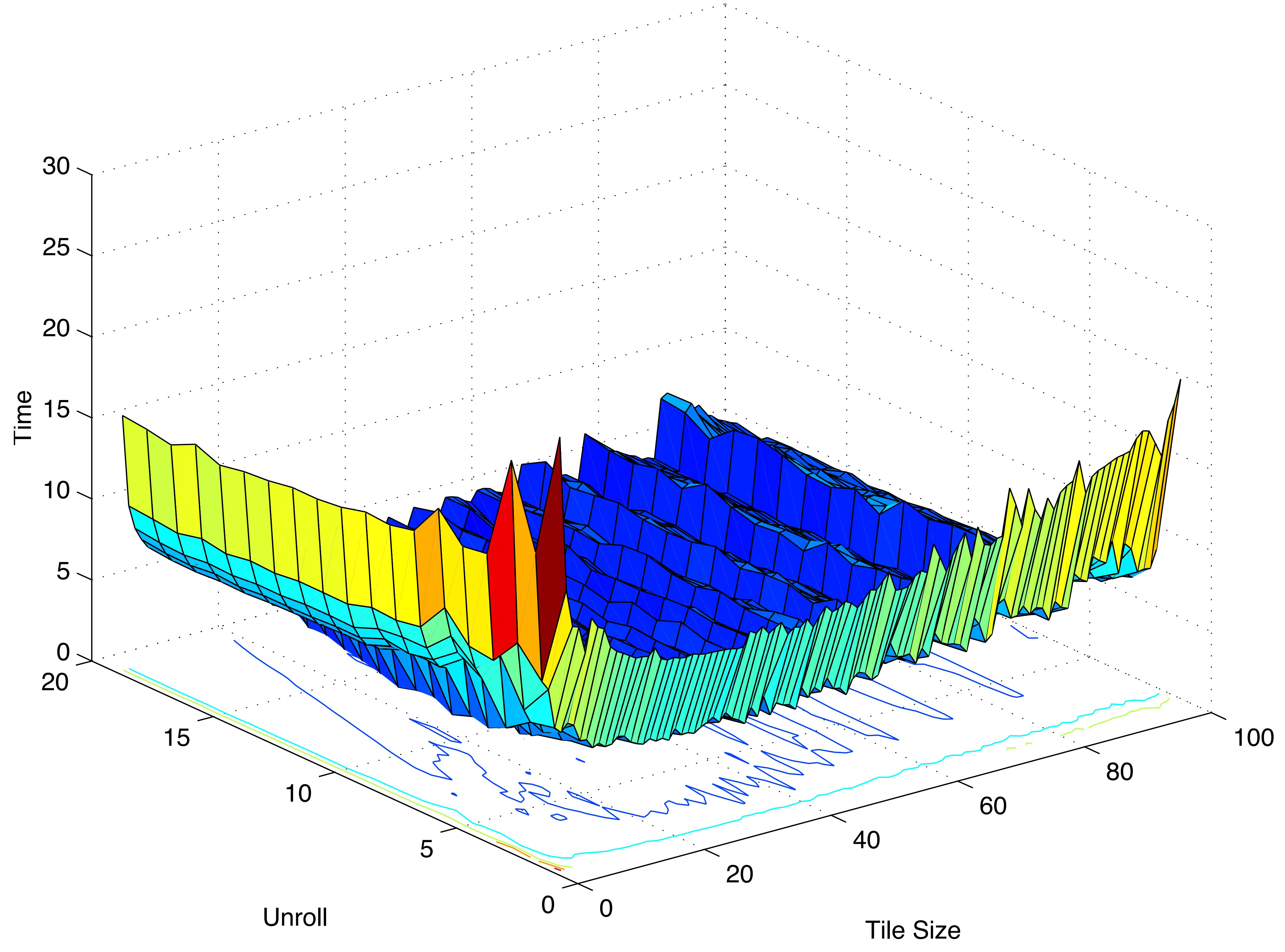
```
for ( i=0; i<N; i++) {
   for ( j=0; j<N; j++){
      for ( k=0; k<N; k++){
         C[i][j] += A[i][k]*B[k][j];
```

What is the best tile and unroll factors for MxM?

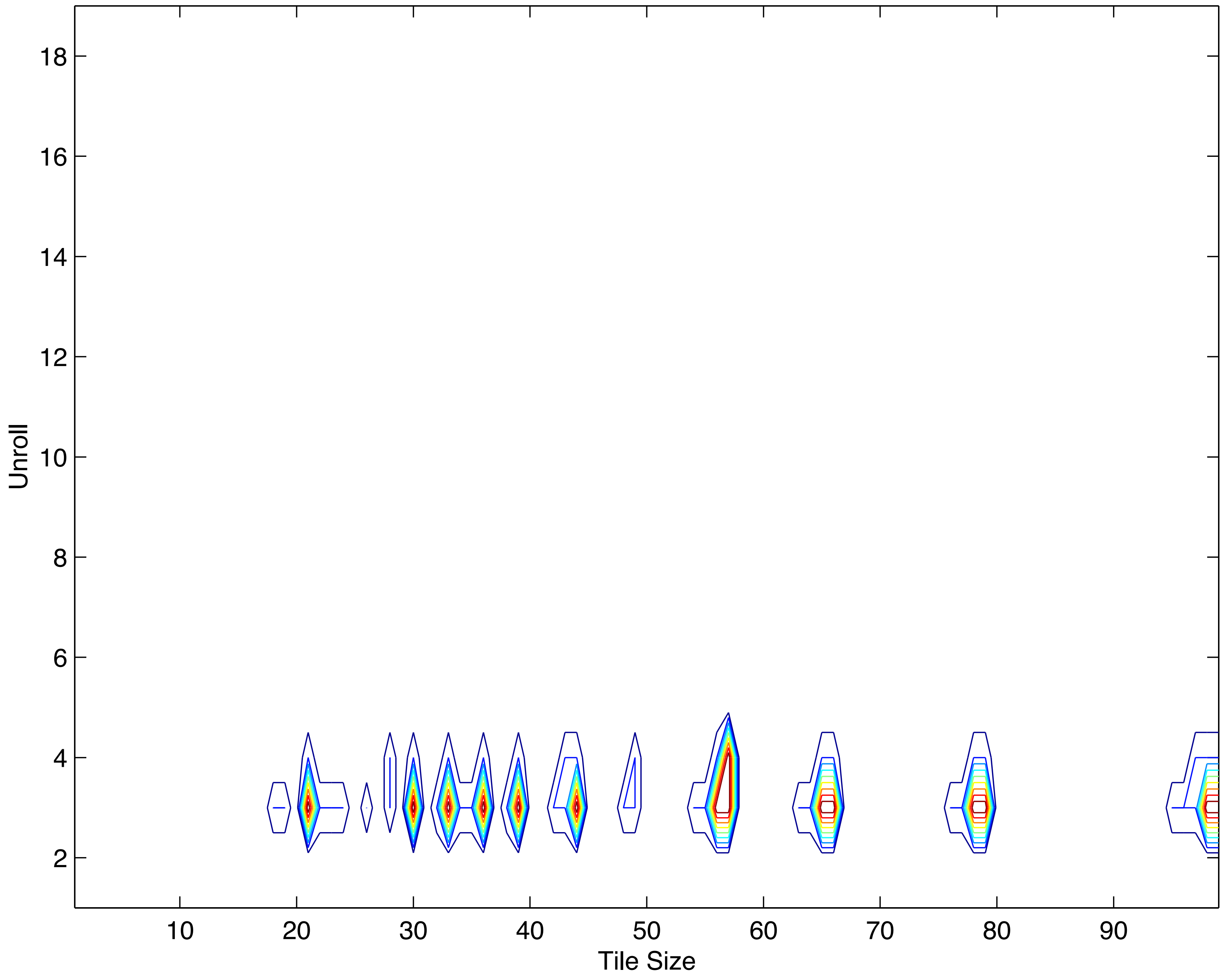
Many papers with definitive answers on either but not both

Empirical evaluation

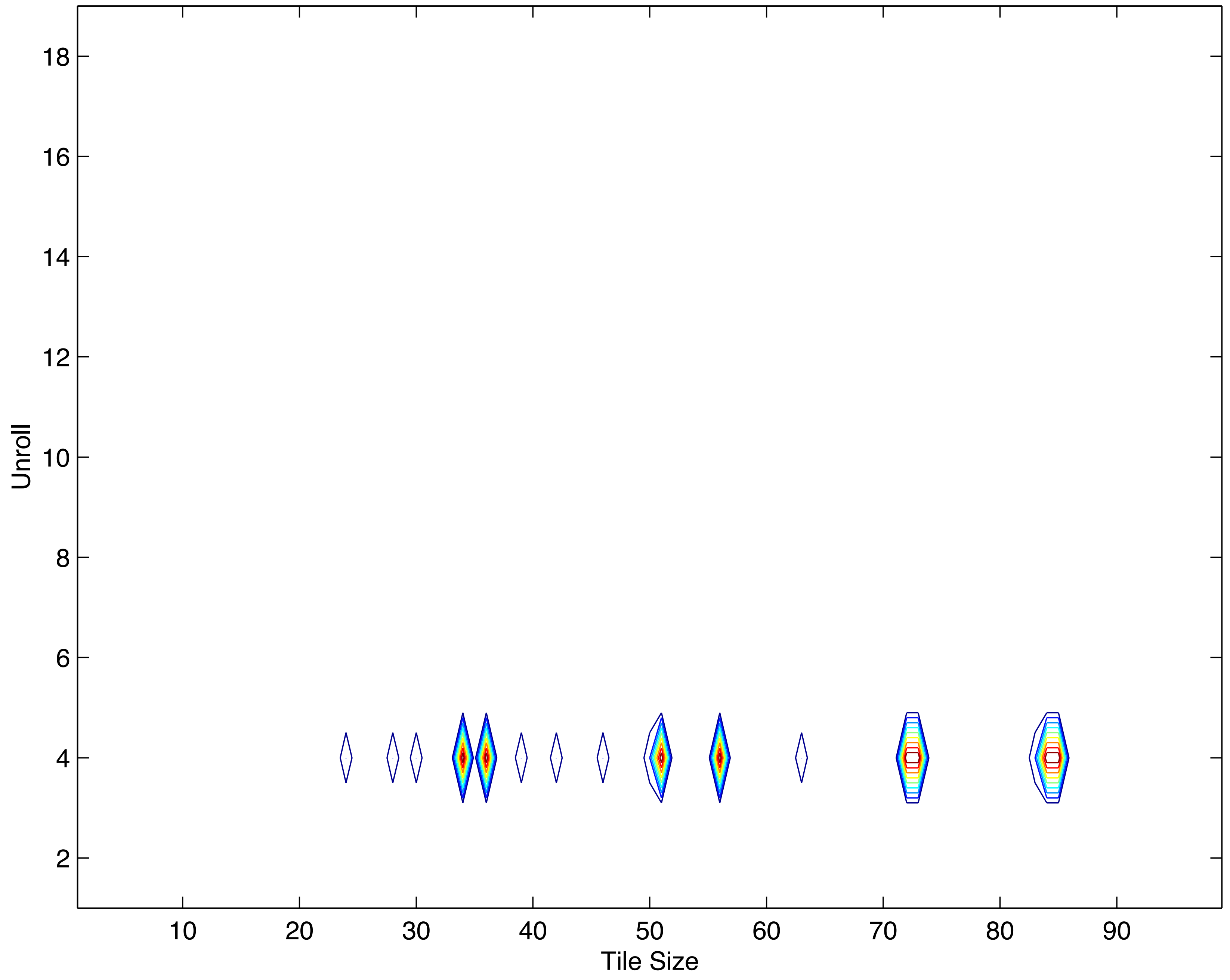
ATLAS [5] and Bodin [4]



UltraSparc: space within 20% of minimum $N = 400$



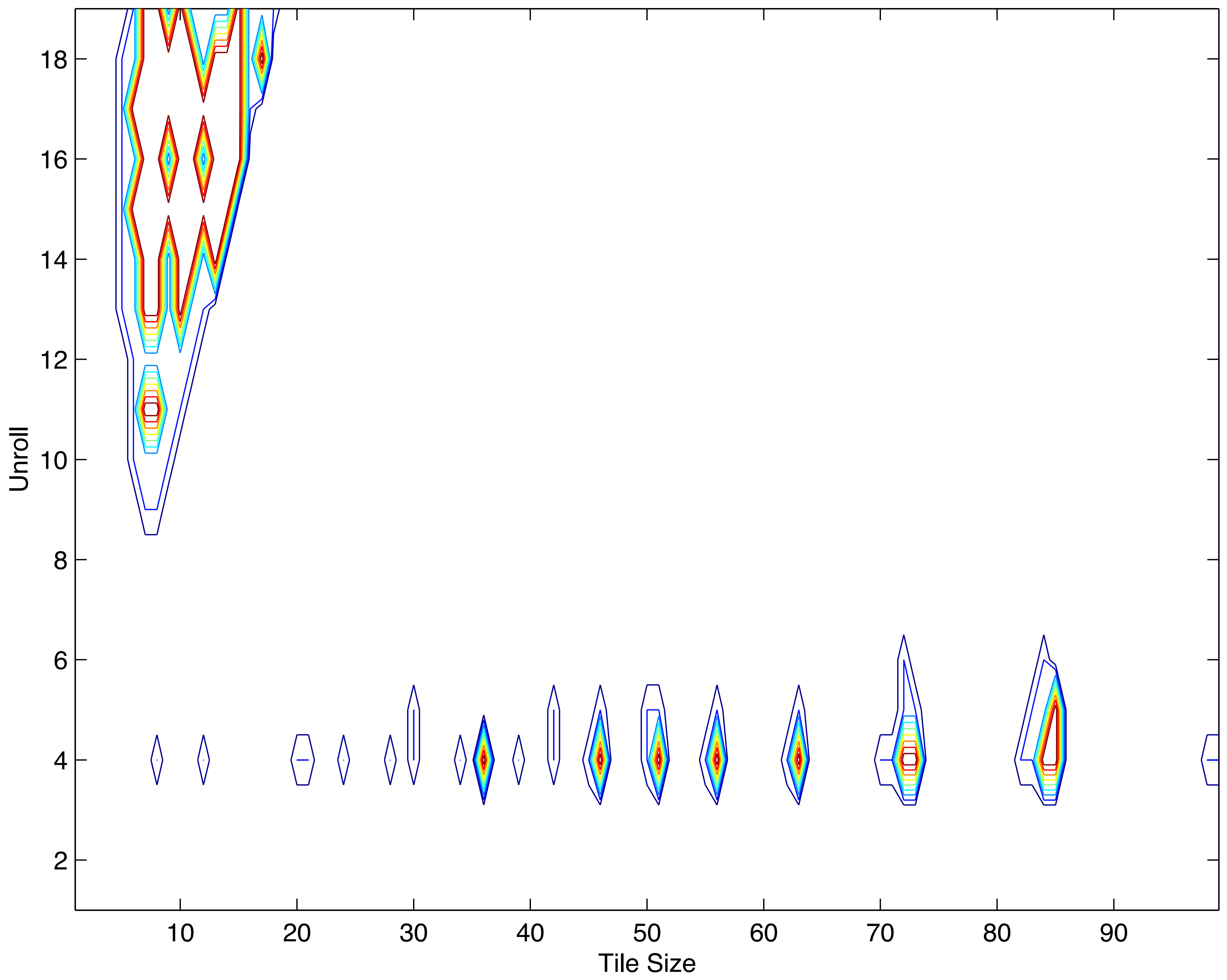
Alpha: space within 20% of minimum $N = 512$



Minimum at
- Unroll = 3, Tile =57
- 2.6% of space near minimum
- 10x between Original and Best

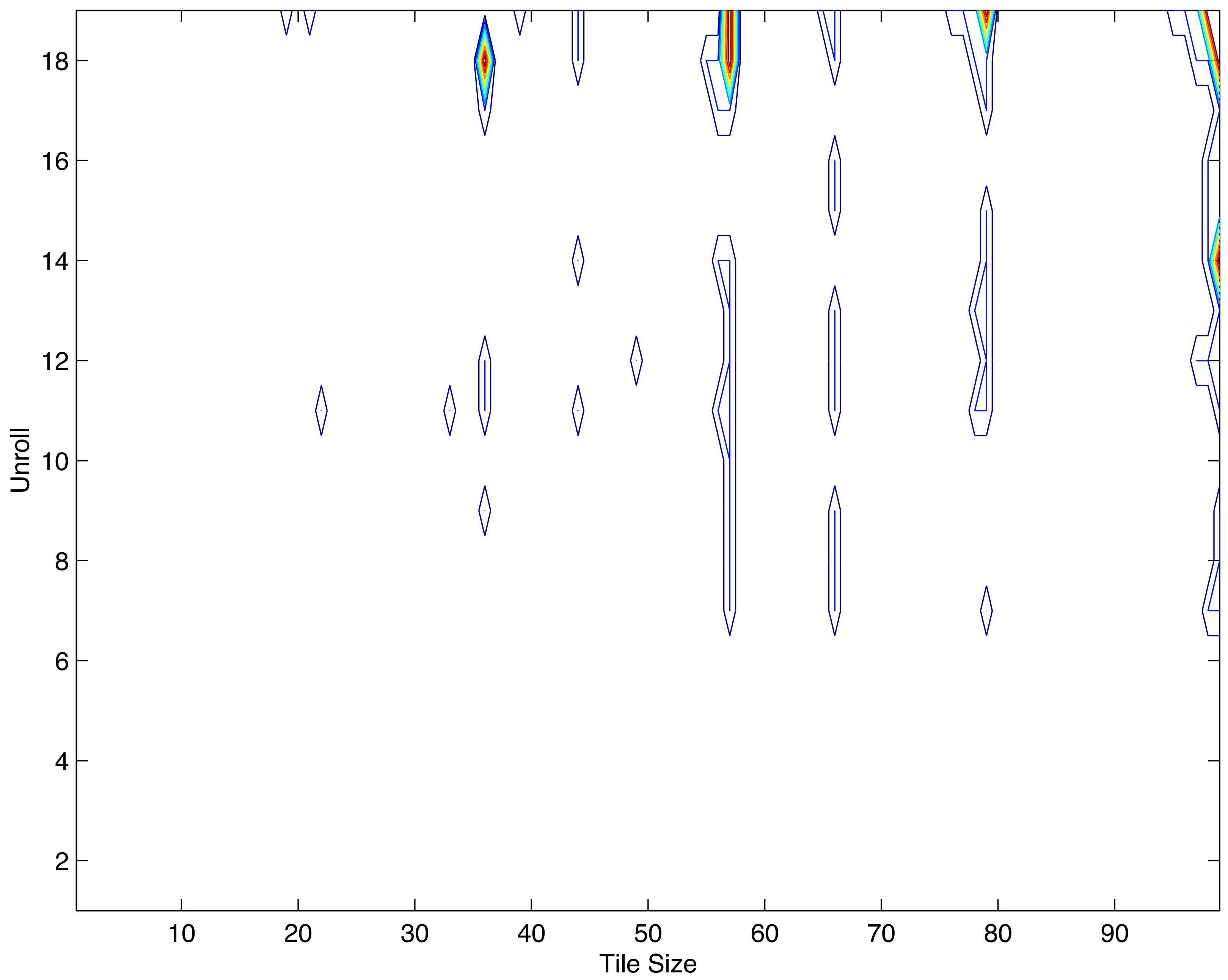
Minimum at
- Unroll = 4 Tile =85
- 0.9% of space near minimum
- 10x between Original and Best
- Worst: 3x slower

R10000: $N = 512$



Minimum at
- Unroll = 4, Tile =85
- 7.2% of space near minimum
- 2x between Original and Best

Pentium Pro: space within 20% of minimum $N = 400$



Minimum at
- Unroll = 19! Tile = 57
- 4.3% of space near minimum
- 3x between Original and Best

Despite 100s of papers, no prior scheme was correct!

Why compiler heuristics fail

50+ years

Fundamental reason is complexity and undecidability

- data to be read in

- processor architecture behaviour is complex
- O-O execution and cache have non-deterministic behaviour

The case for automation

Optimization space hard
- especially if hardware changes

All compiler analysis
- FAILED
- MxM: most studied benchmark

Empirical evidence
- rather than theory

2 ways forward
- Search : Auto-tuning
- Machine Learning: Automatic learning

What is compilation

Why do we need new techniques

Automation

Search/ Auto-tuning/ Iterative compilation

ML for compilation

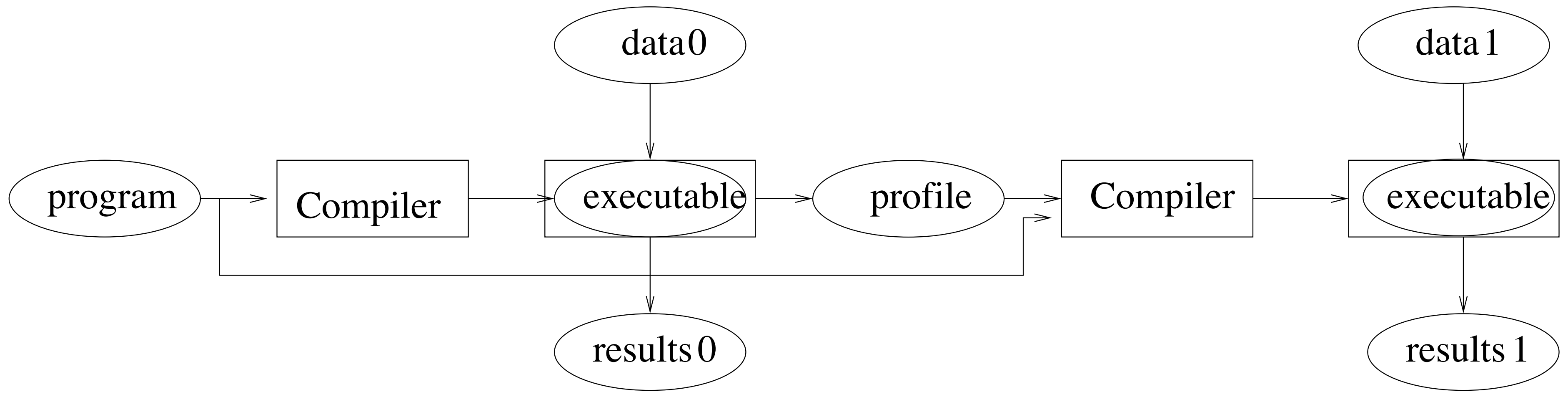
Features, models and applications

Summary

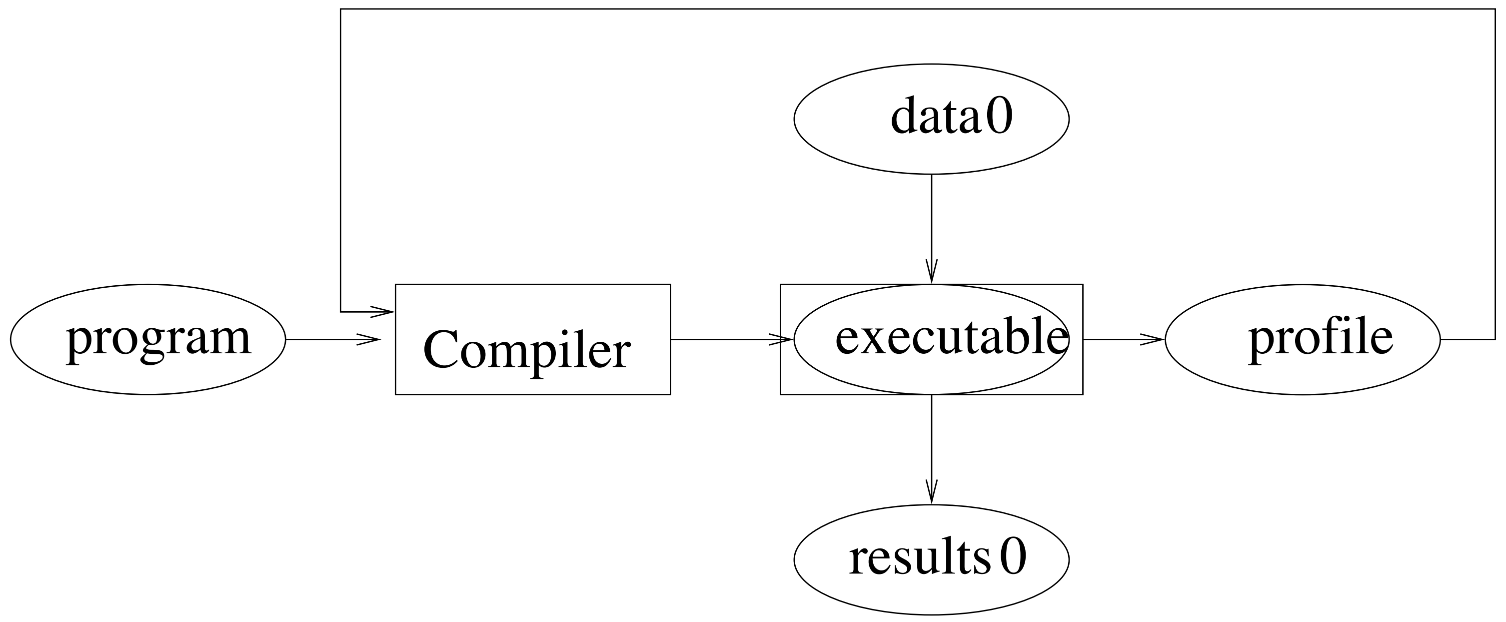
Profile-directed to Iterative Compilation

Profile directed compilation [6]

- Collect some information
- Use to improve program performance



PDC one compile

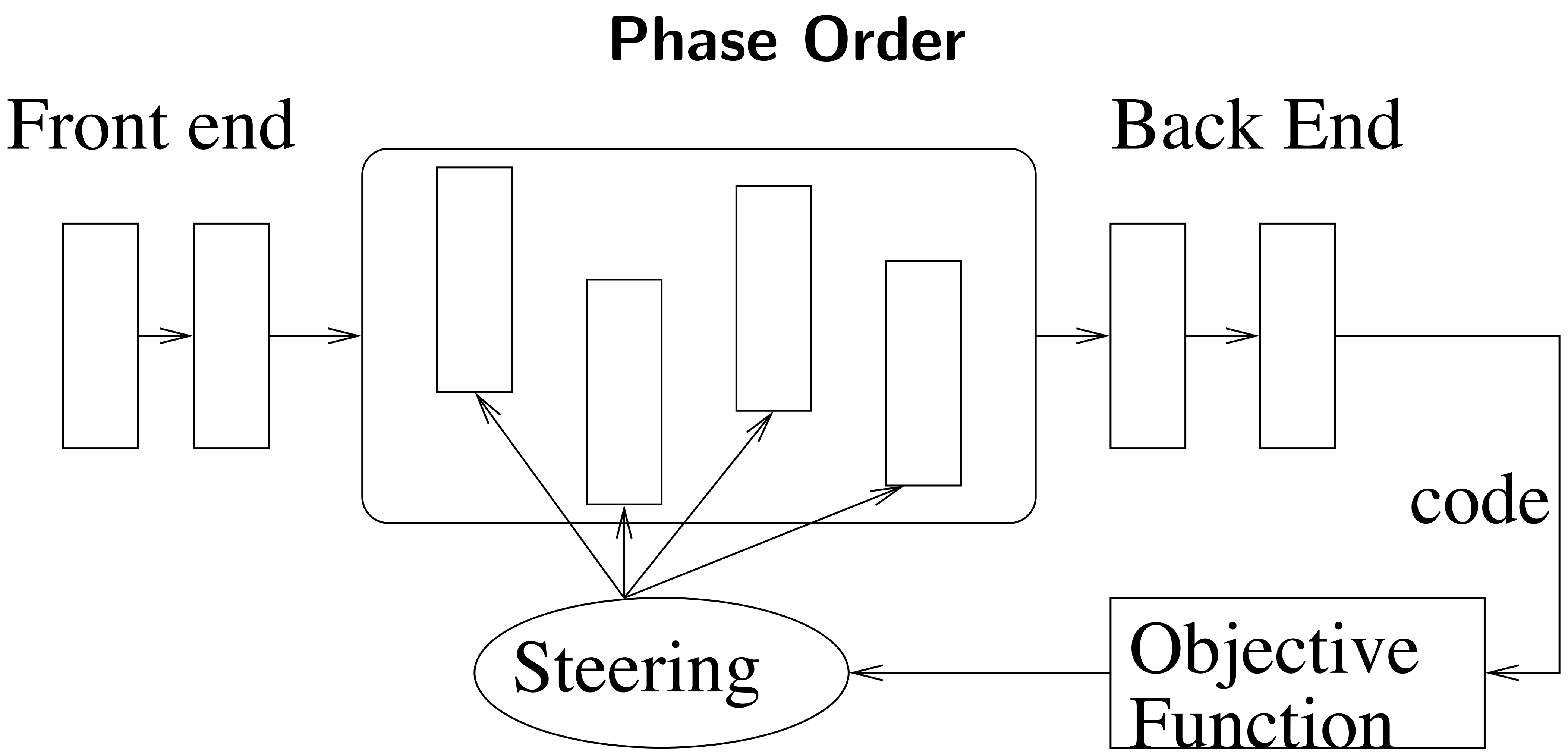


Iterative: multiple compiles

Performance gains modest

- Focuses on persistent control-flow
- All other information ignored

Search: Phase Ordering



Cooper [7] up to 25% over default..

- Focussed on code size
- Noise in execution time

- Leather[17[addressed this with raced profiles
- Look at problems where signal>noise

GA, Hill-climb, Gradient decent

- Many many papers!

Systematic evaluation:

- What about random?

Agakov 2006

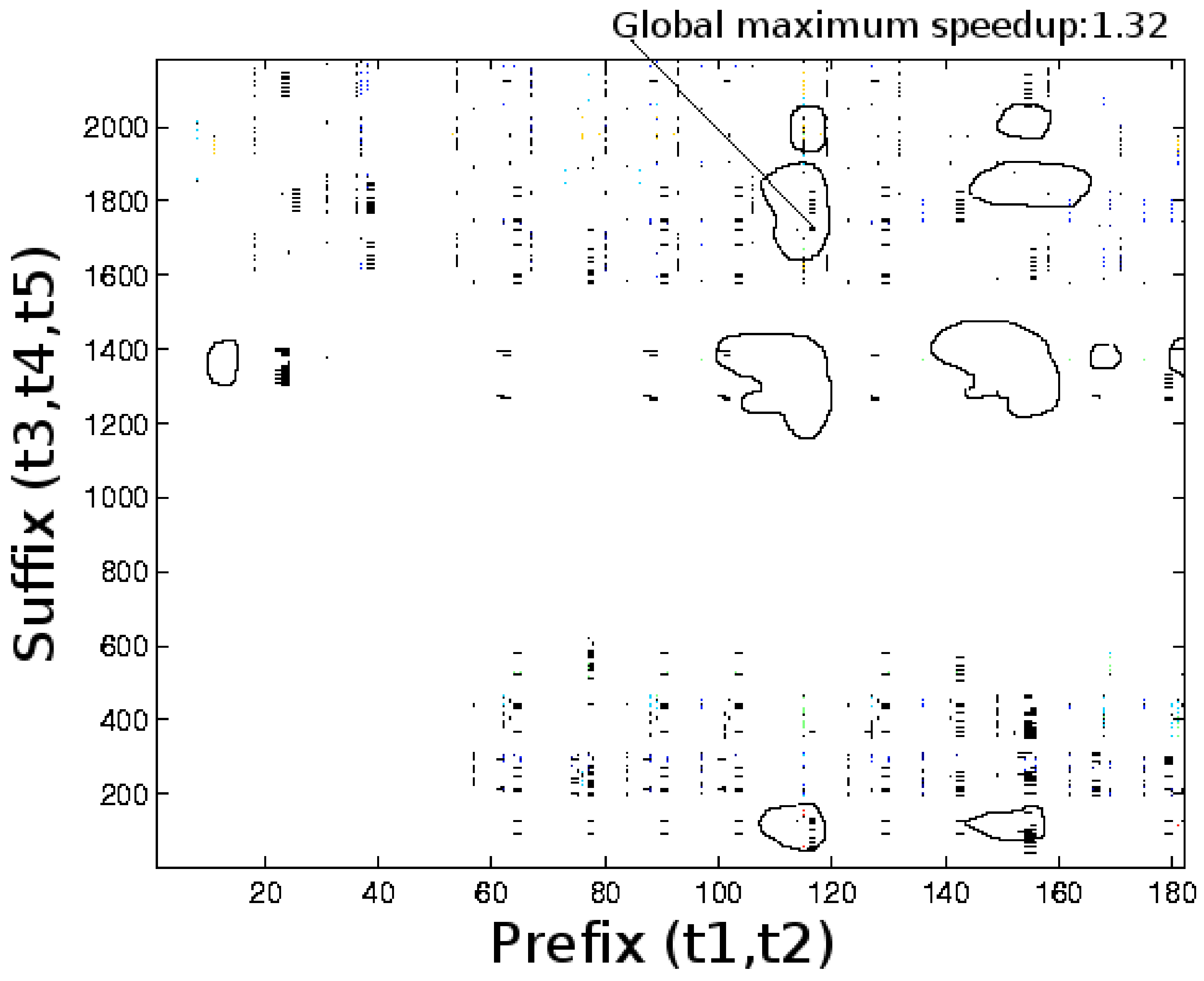
Random as good as any [8]

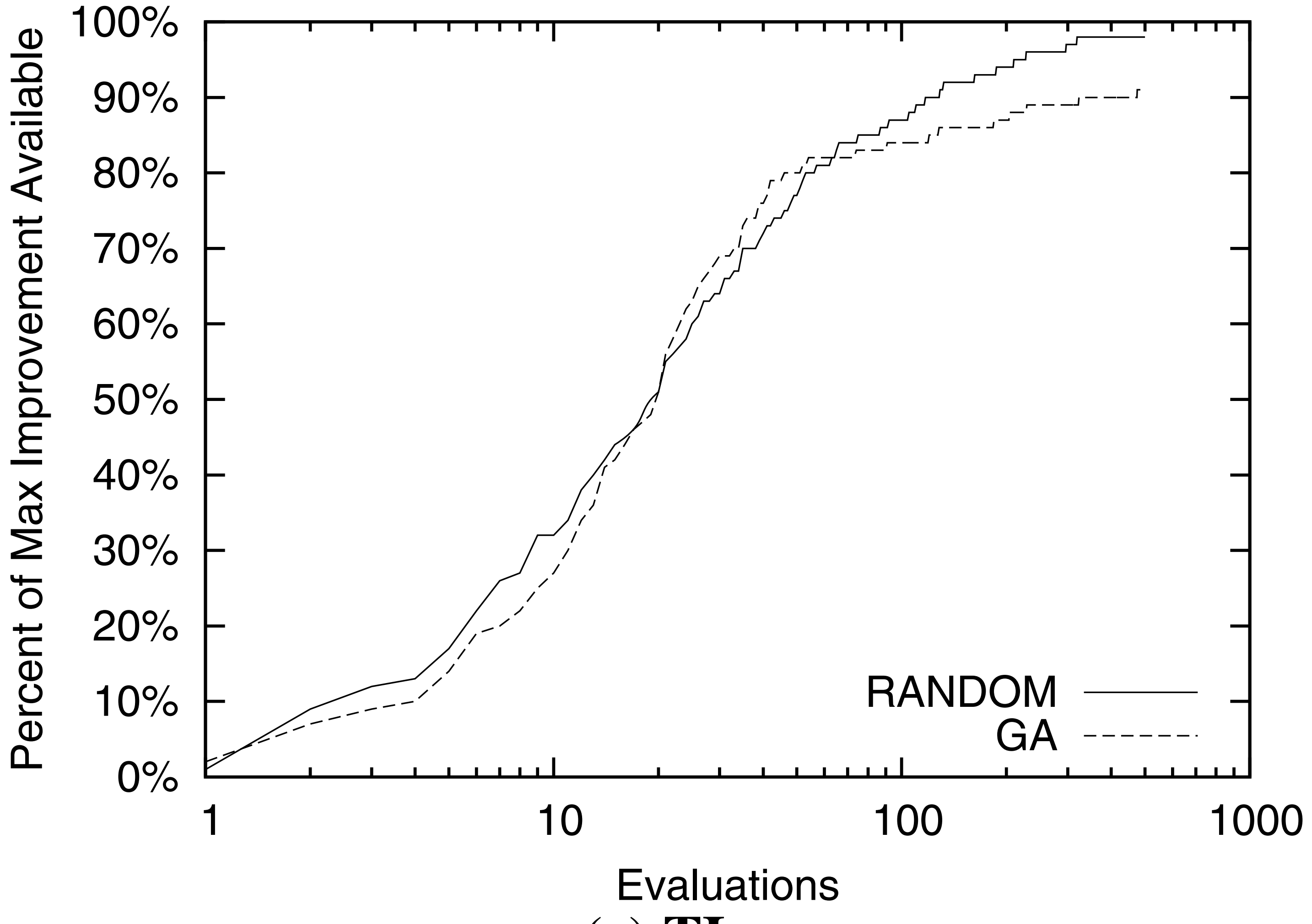
GCC: O(10^260)

- but large parts irrelevant

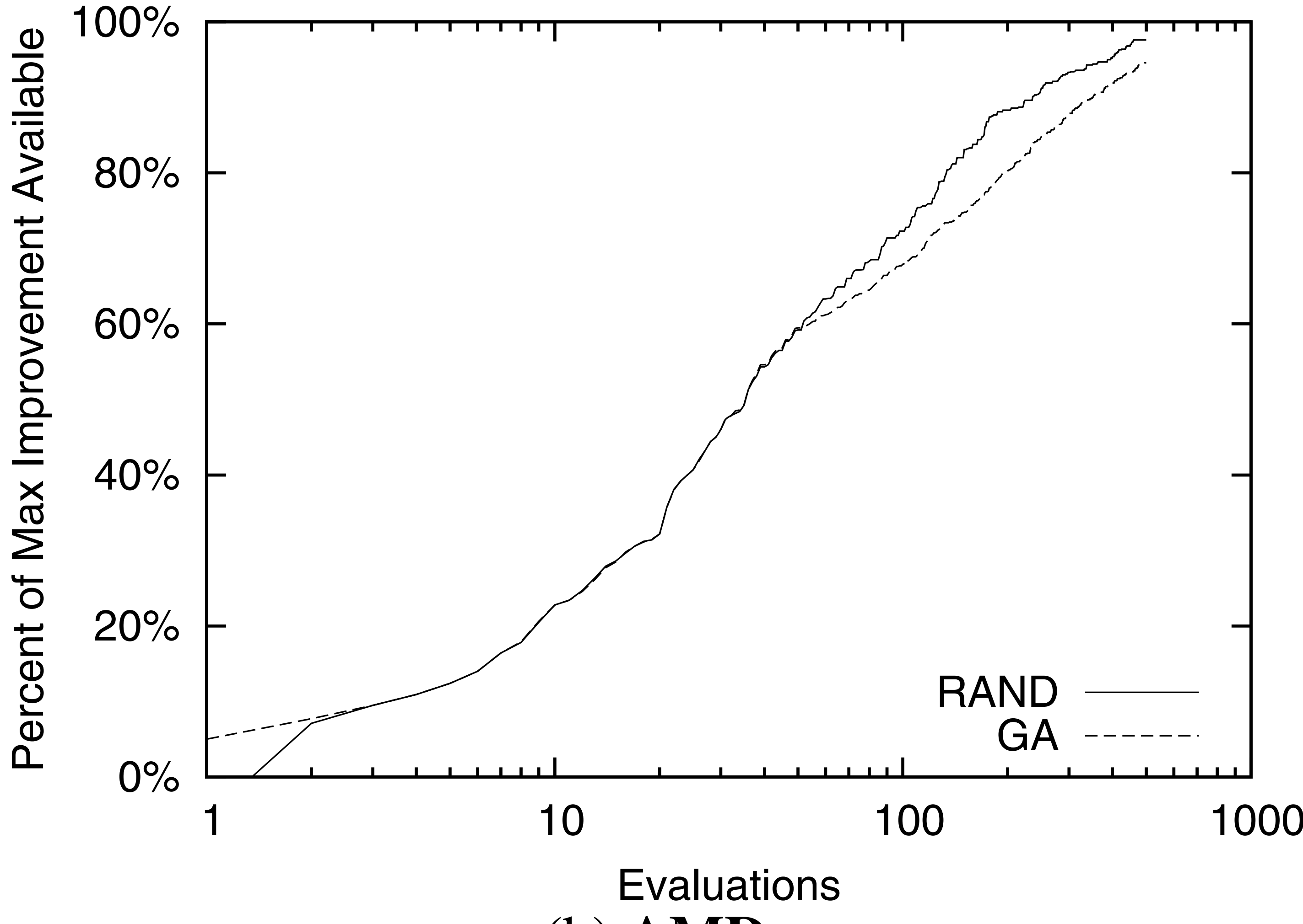
Need to have a useful space.

- Orthogonal - no repetition.
- Find good points quickly



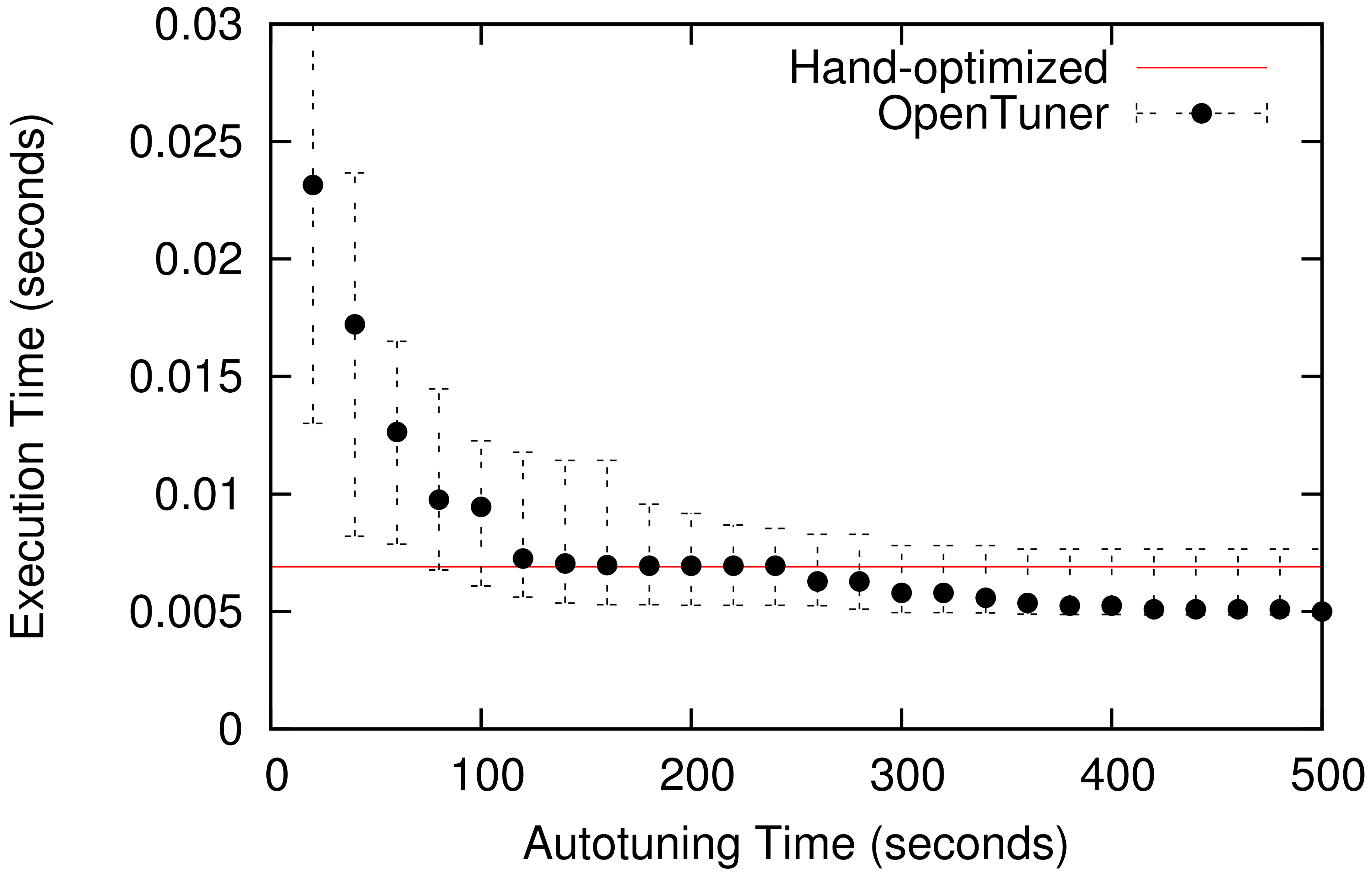


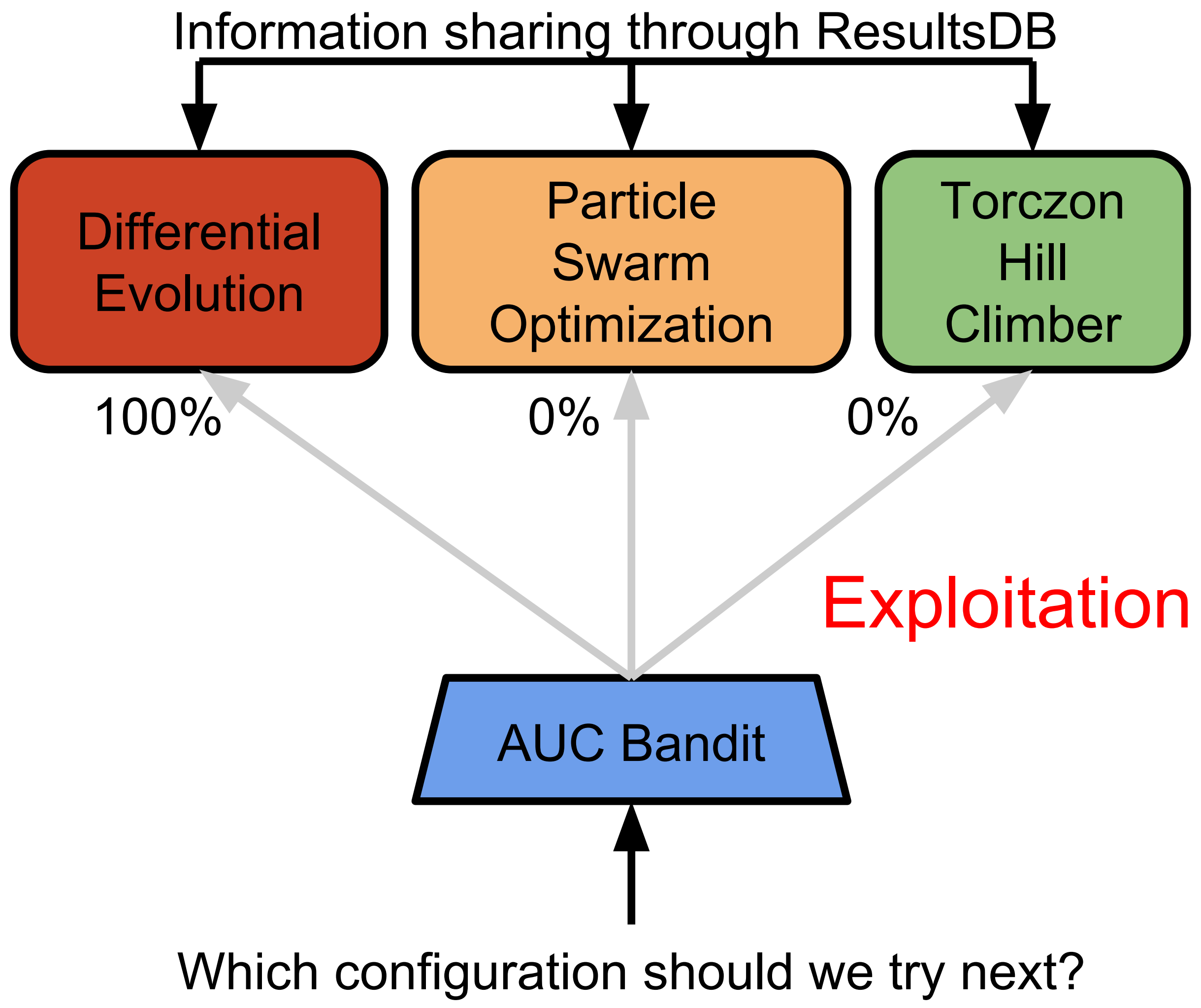
(a) TI



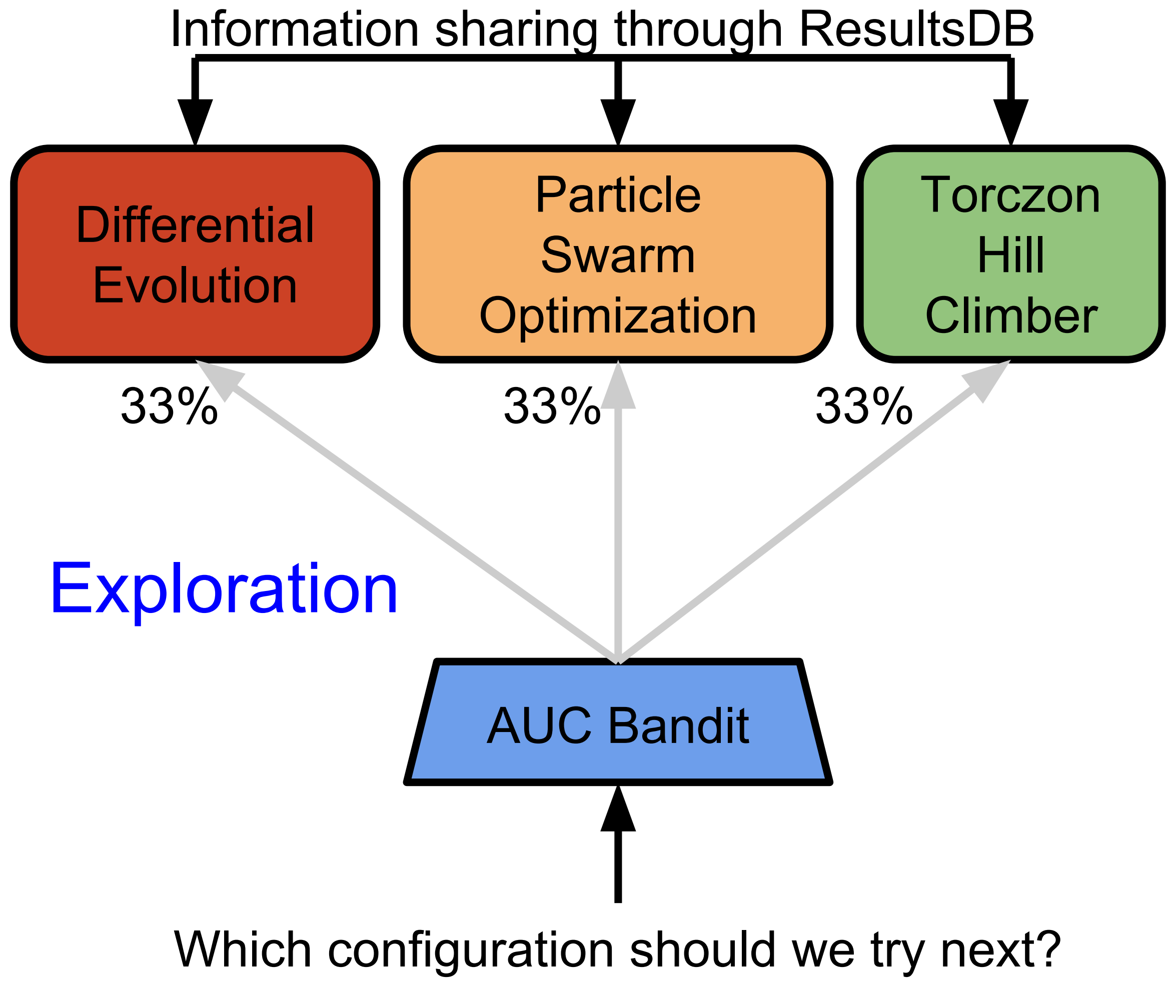
(b) AMD

OpenTuner [9]



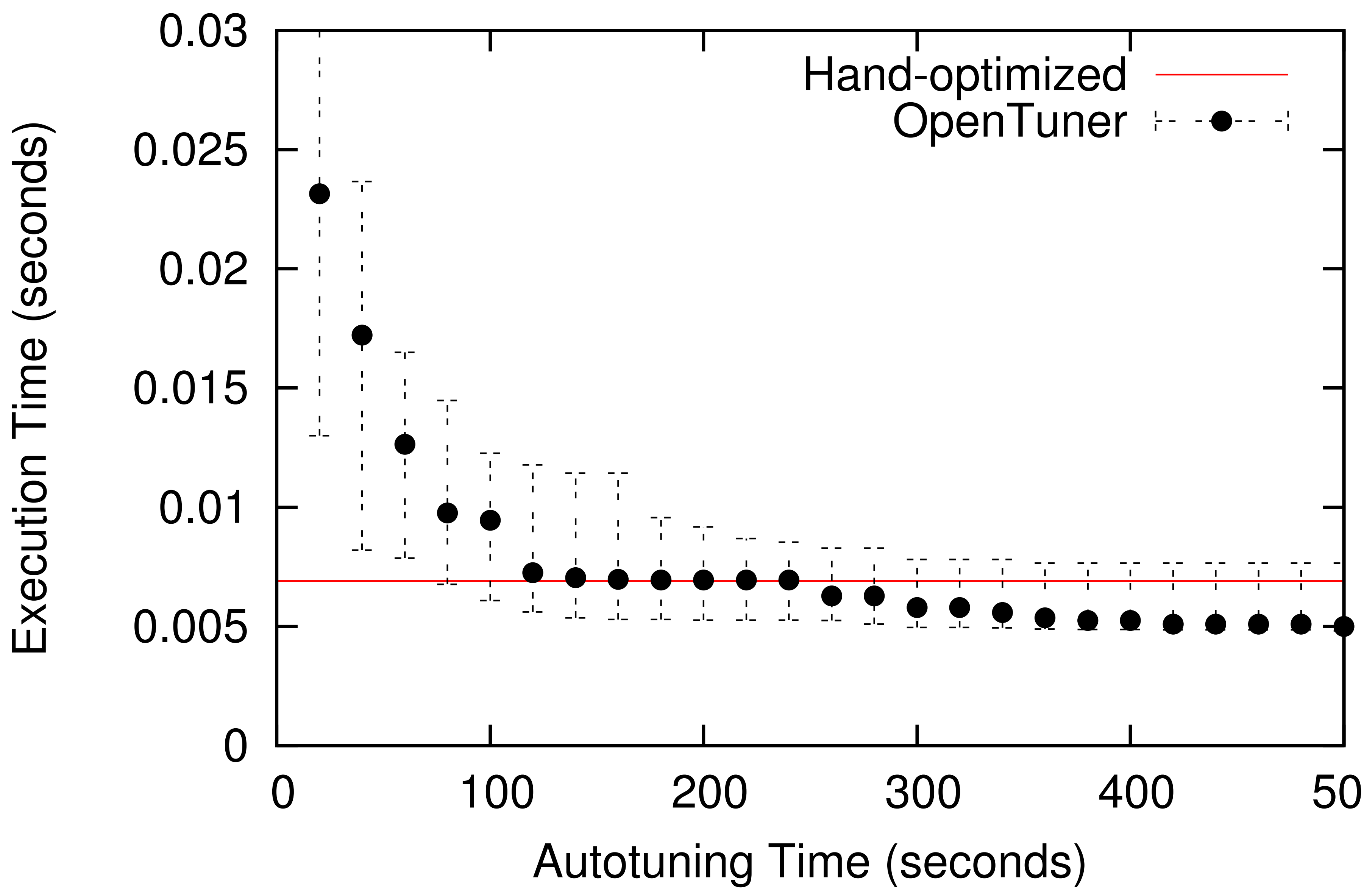


- Differential Evolution
- Genetic Algorithms
- Greedy Mutation
- Multi-armed Bandit
- Nelder Mead
- Partial Swarm Optimization
- Pattern Search
- Pseudo Annealing
- Torczon



Auto-tuning/Iterative Compilation

Space structure Cooper[7] says varies little Vuduc[10] disagrees

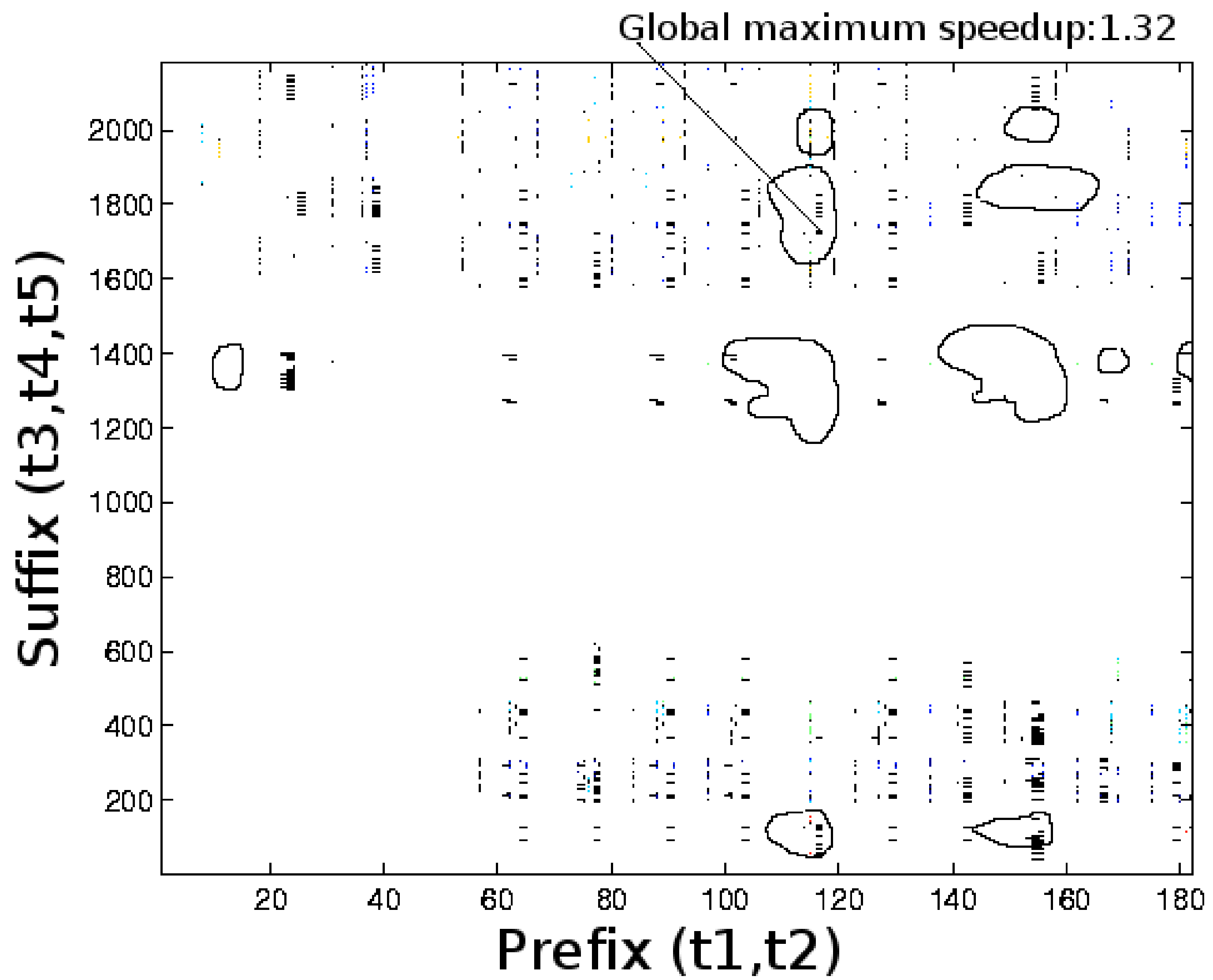


Application tuning is not portable

Useful for data independent programs (eg MxM)

Excessive compile time suitable for embedded or libraries.

Why not remember ??? Using Prior Knowledge in Search space: or ML



What is compilation

Why do we need new techniques

Automation

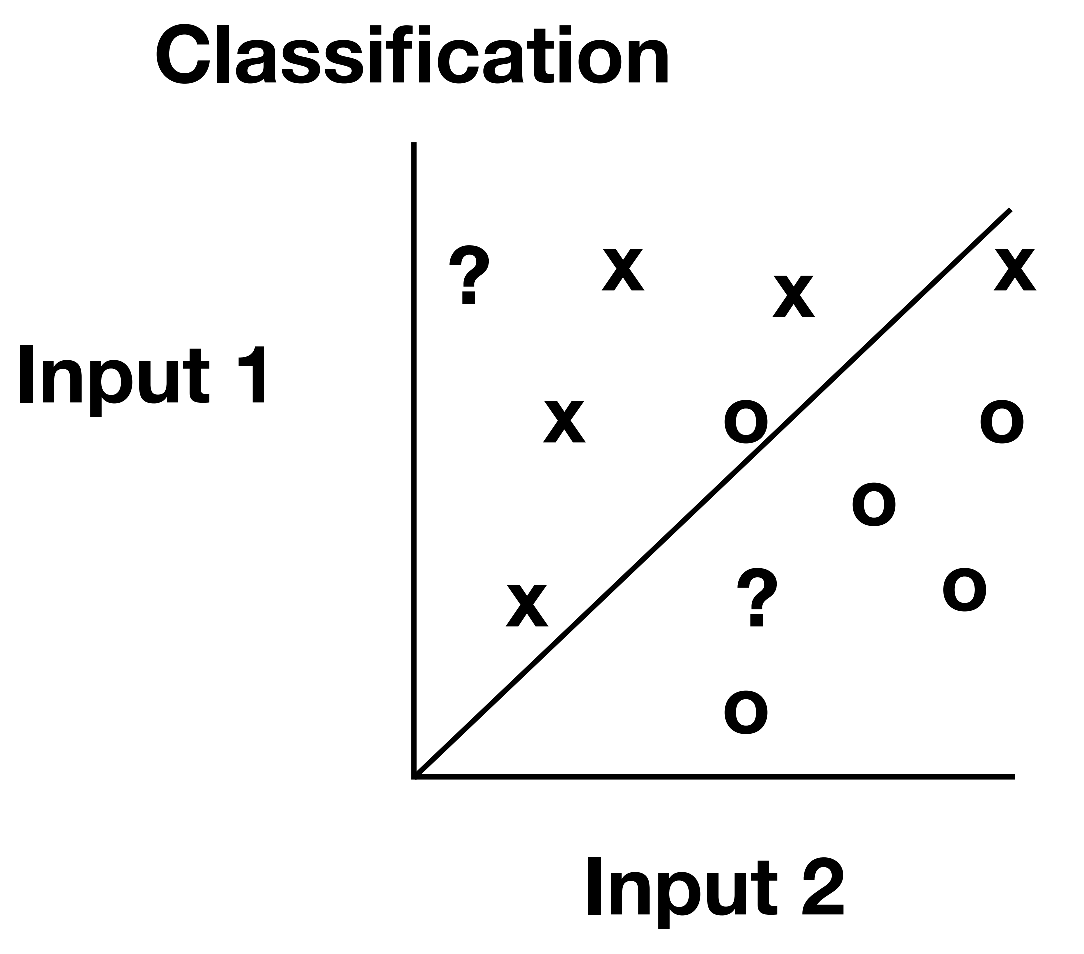
Search/ Auto-tuning/ Iterative compilation

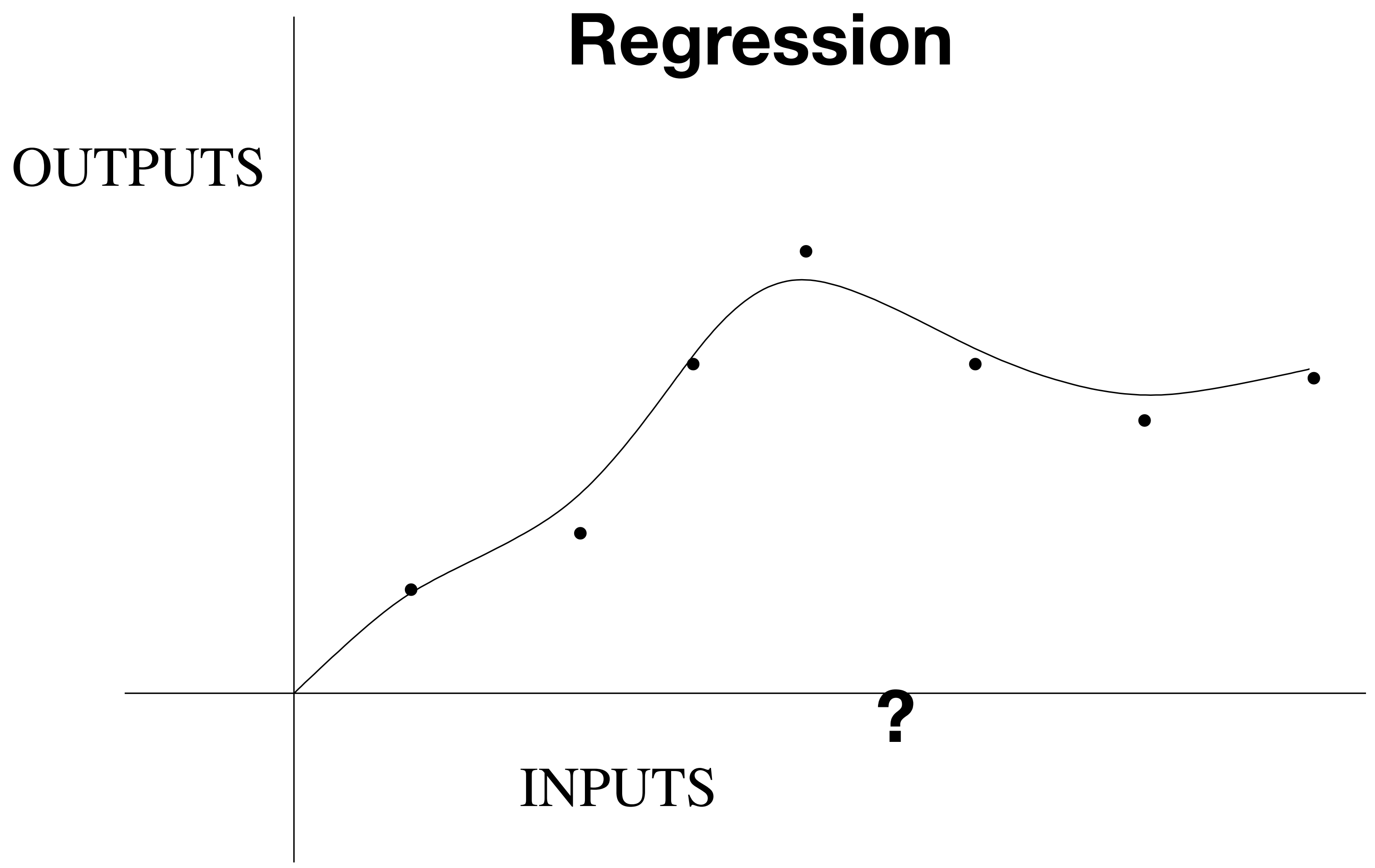
ML for compilation

Features, models and applications

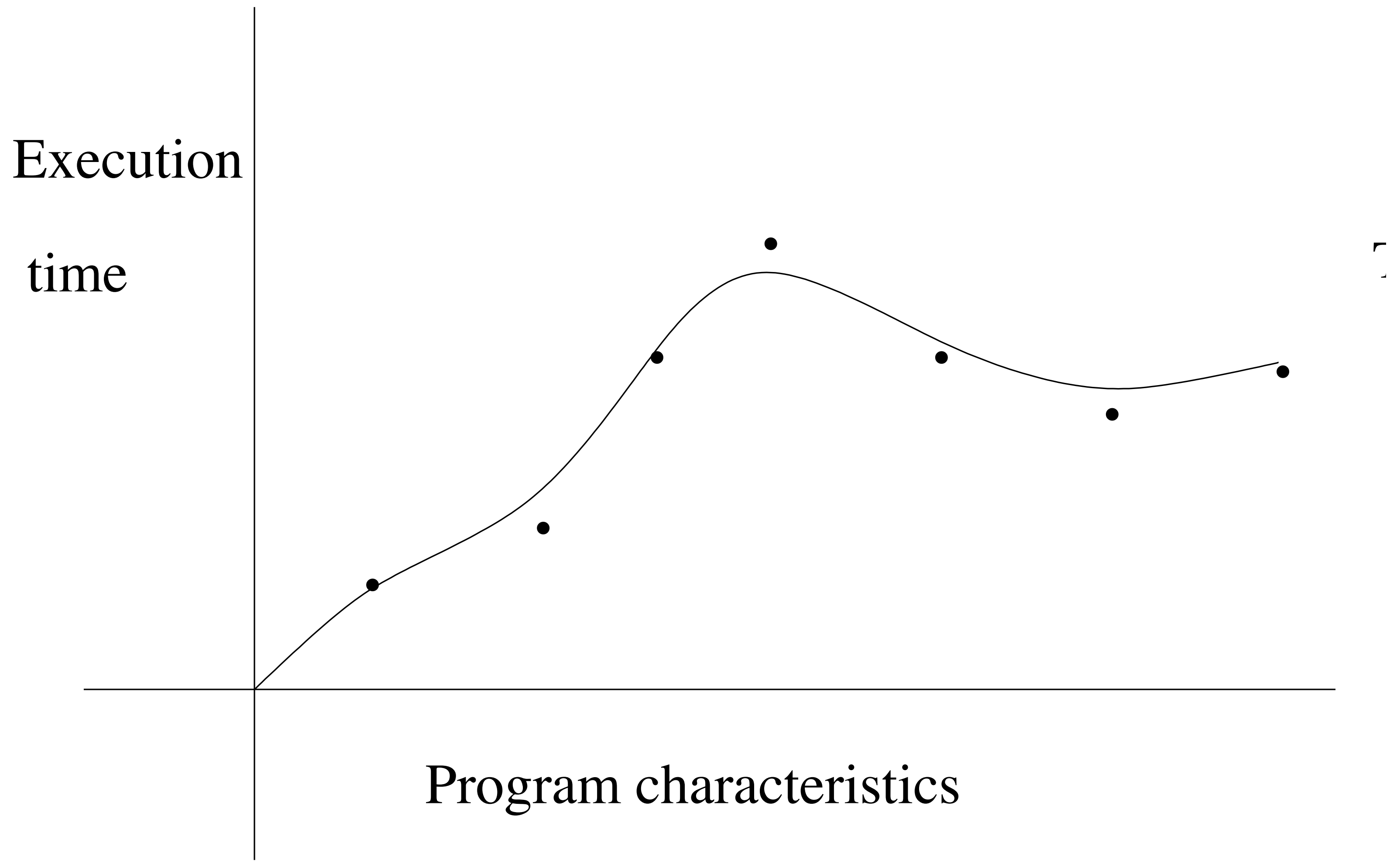
Summary

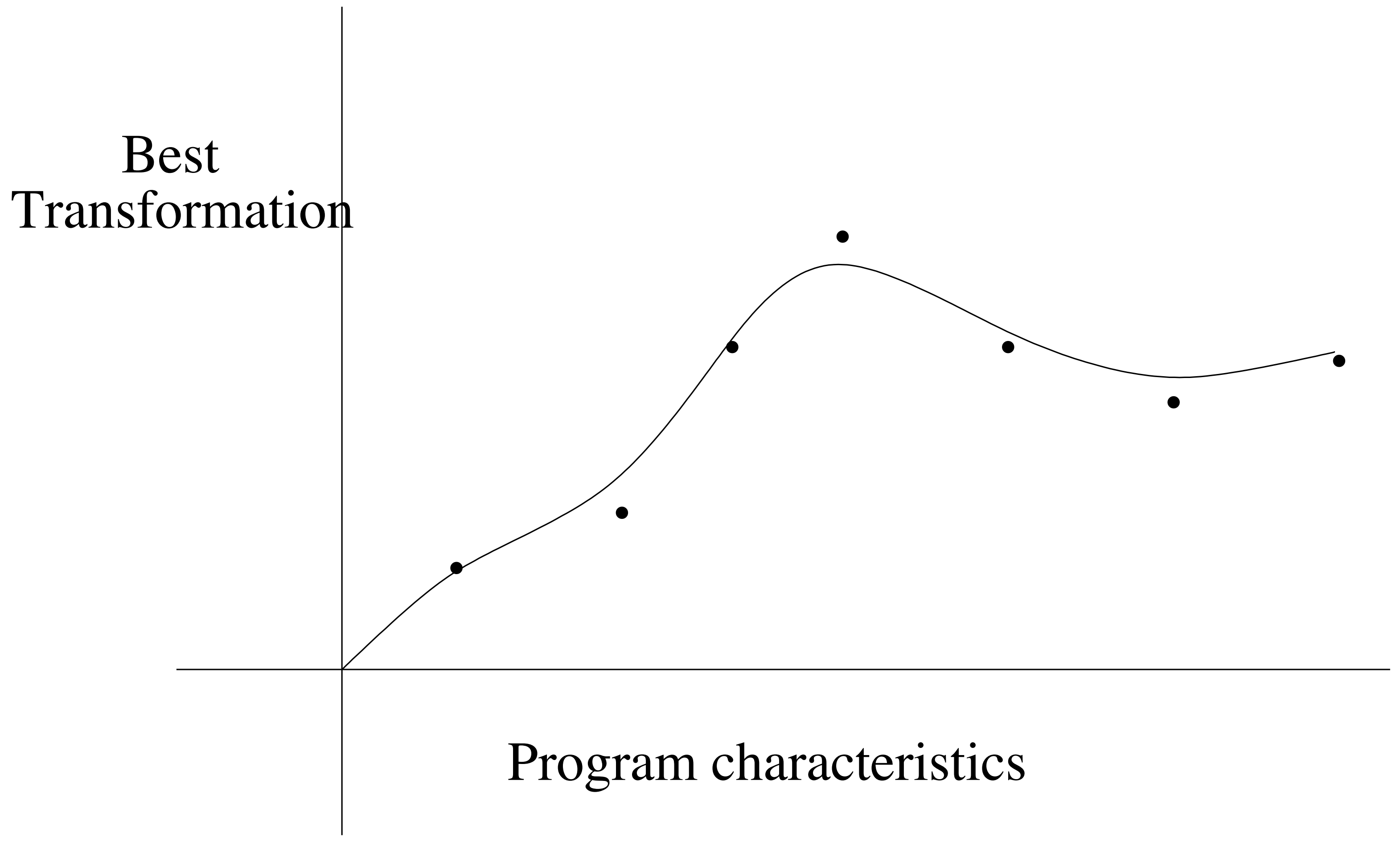
Brief intro to ML for compilers

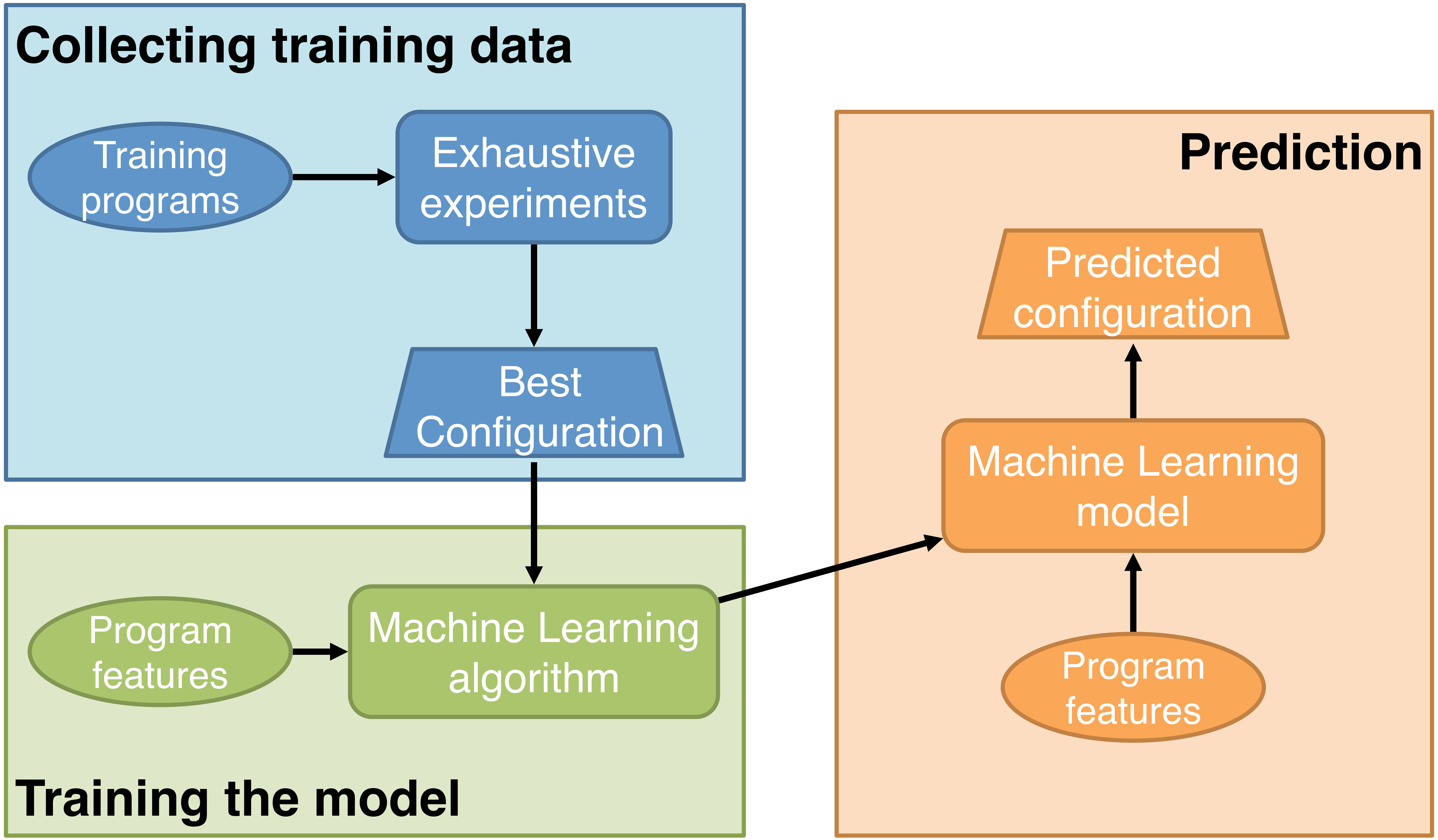




Sophisticated curve fitting? .

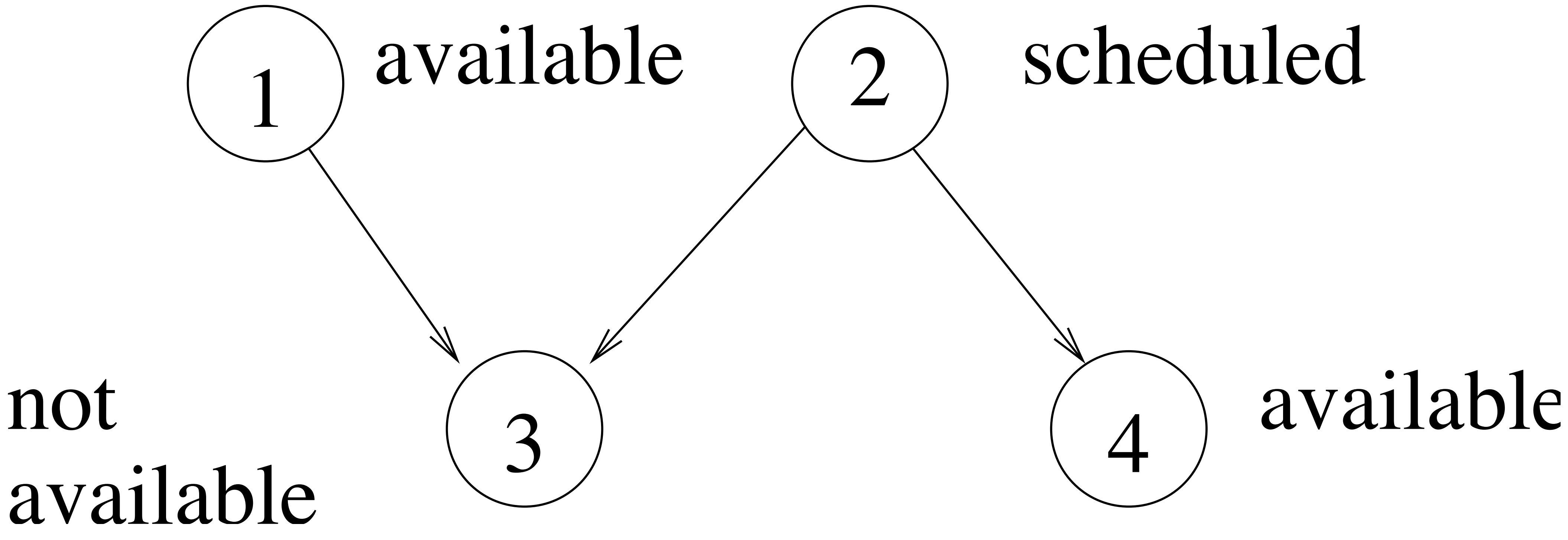






Learn a model that correlates outputs to inputs
Distinct train and test data - unlike most compiler papers!!

Learning to Schedule Moss, Cavazos [11]



Given partial schedule 2,

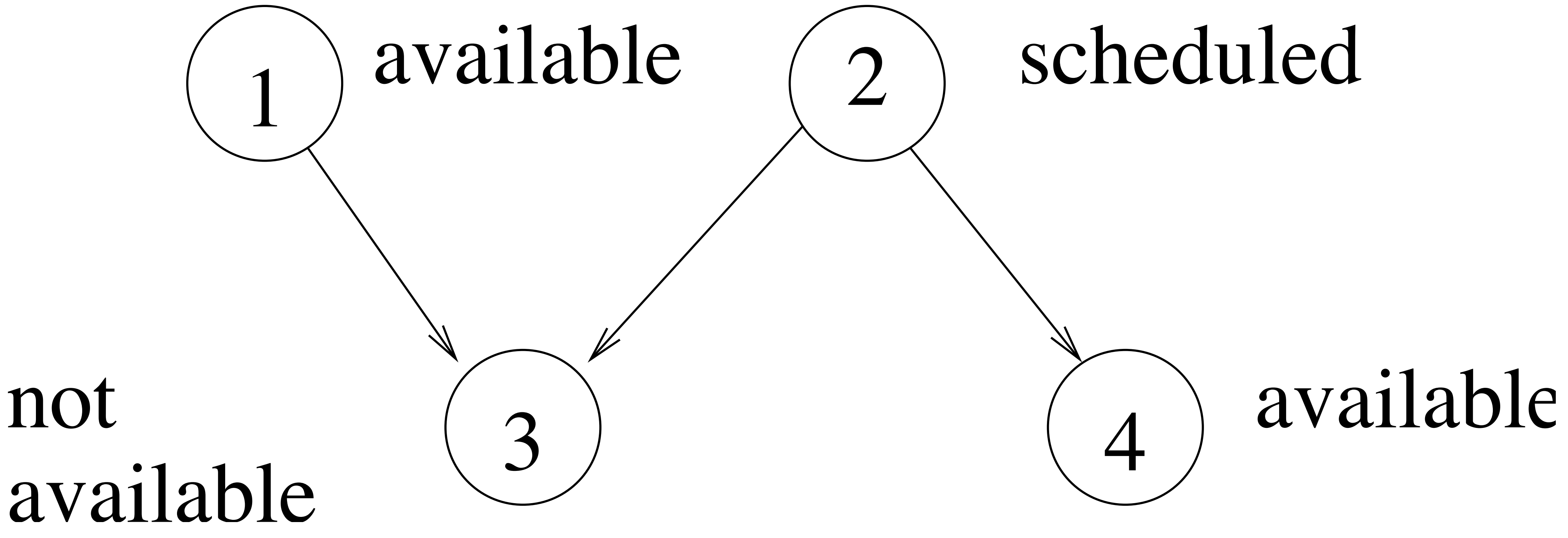
schedule next instruction 1 or 4?

First paper on ML for compiler optimisation

Appeared at NIPS '97

- not picked up by compiler community till later.

Learning to Schedule Moss, Cavazos



Train on many basic blocks, determine ALL possible schedules.

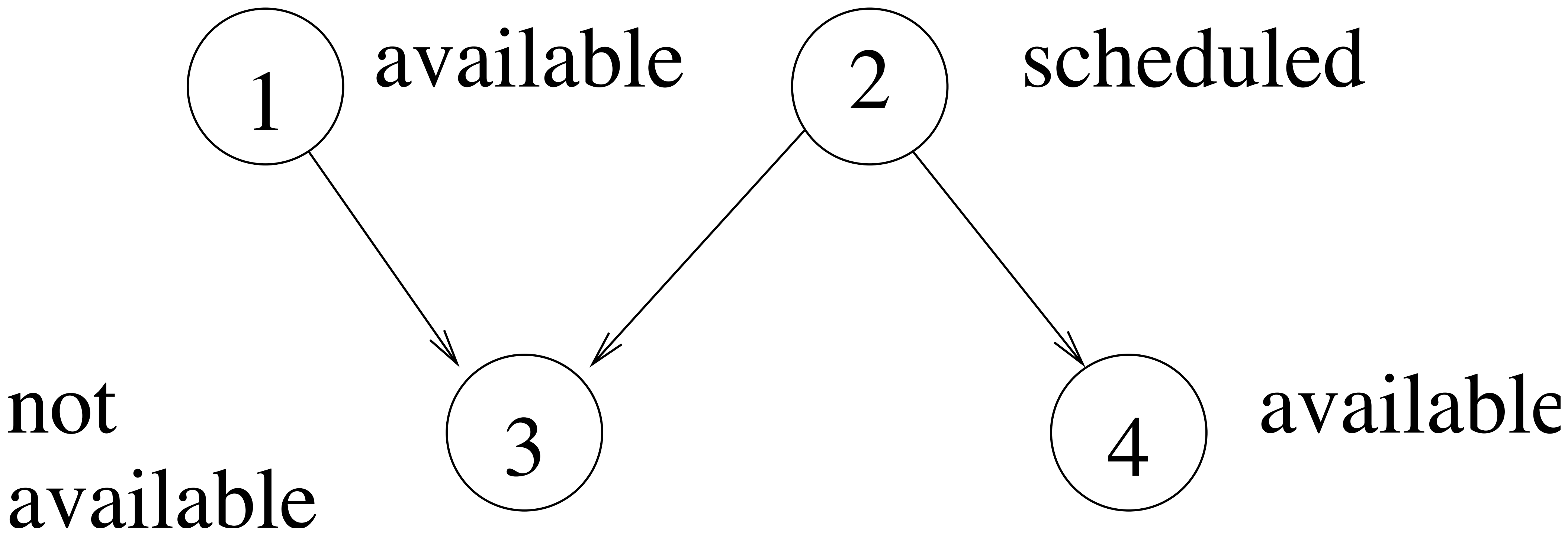
- Given two instructions to scheduled
- Select each in turn and determine which is best.

Record (P,I_i,I_j) P is a partial schedule, I_i. to be scheduled earlier first. Record TRUE as output.

- Record FALSE with (P,I_j,I_i)

Fixed length vector summary based on features.

Features and tuples



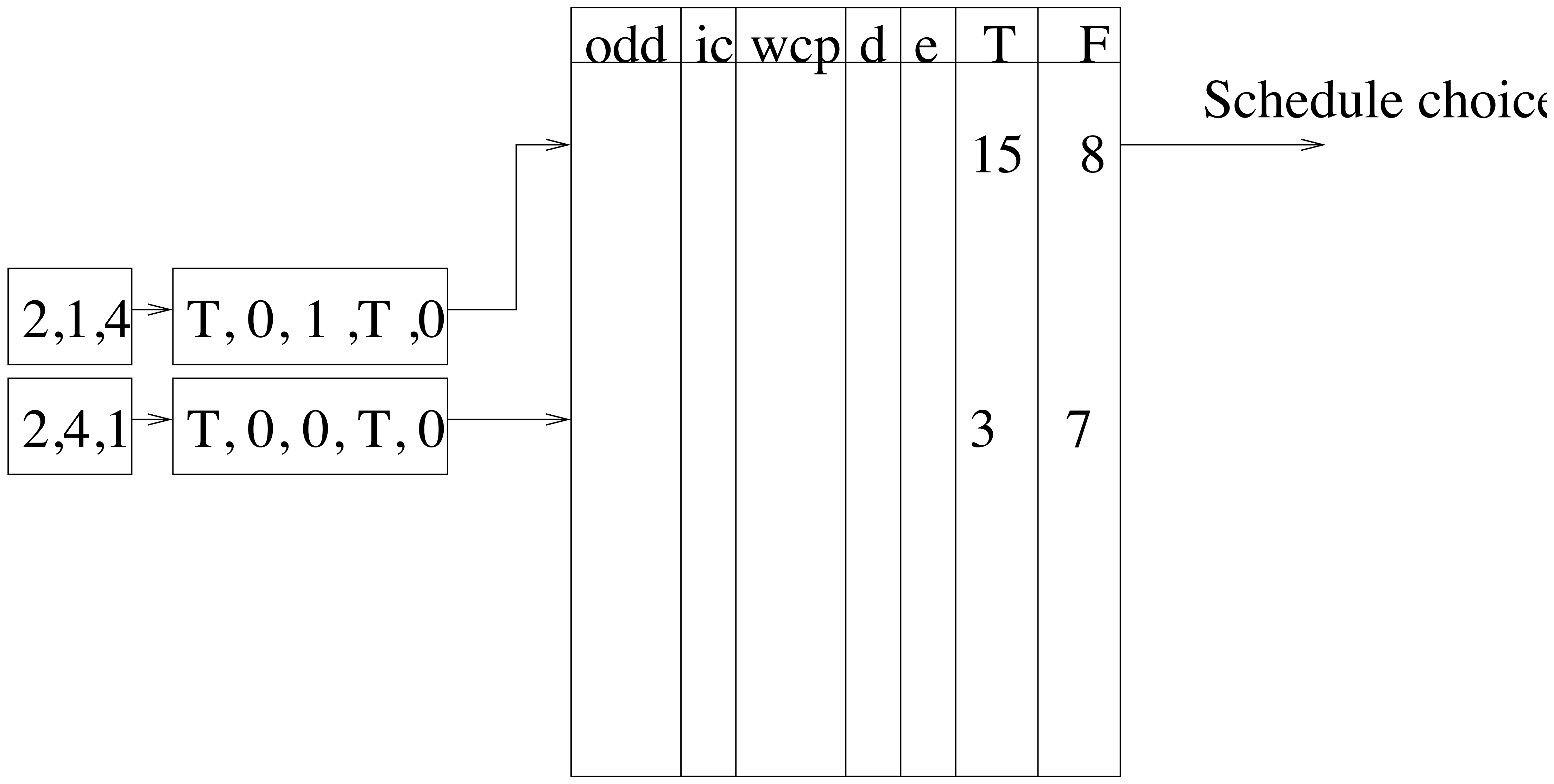
Tuple ({2}, 1, 4) : [odd:T, ic:0, wcp:1, d:T, e:0]: TRUE

Tuple ({2}, 4, 1) : [odd:T, ic:0, wcp:0, d:T, e:0]: FALSE

Feature selection can be a black art.

- Odd Partial (odd): odd or even length schedule
- Instruction Class (ic): which class corresponds to function unit
- Weighted critical path (wcp): length of dependent instructions
- Actual Dual (d): can this instruction dual issue with previous
- maxdelay (e): earliest cycle this instruction can go

Models: Lookup and Induction rule



If feature vector not stored, then find nearest example.

The first schedule is selected

- 15>8 vs 3<7

$e = second$
$e = same \wedge wcp = first$
$e = same \wedge wcp = same \wedge d = first \wedge ico = load$
$e = same \wedge wcp = same \wedge d = first \wedge ico = store$
$e = same \wedge wcp = same \wedge d = first \wedge ico = ilogical$
$e = same \wedge wcp = same \wedge d = first \wedge ico = fpop$
$e = same \wedge wcp = same \wedge d = first \wedge ico = iarith \wedge ic1 = load$ ...

Schedule the first I_j

- if the max time of the second is greater
- if the same, schedule the one with the greatest number of critical dependent instruction ...

Results

All techniques were very good

- 98% of the performance of the hand-tuned heuristic

Small basic blocks were good training data for larger blocks.

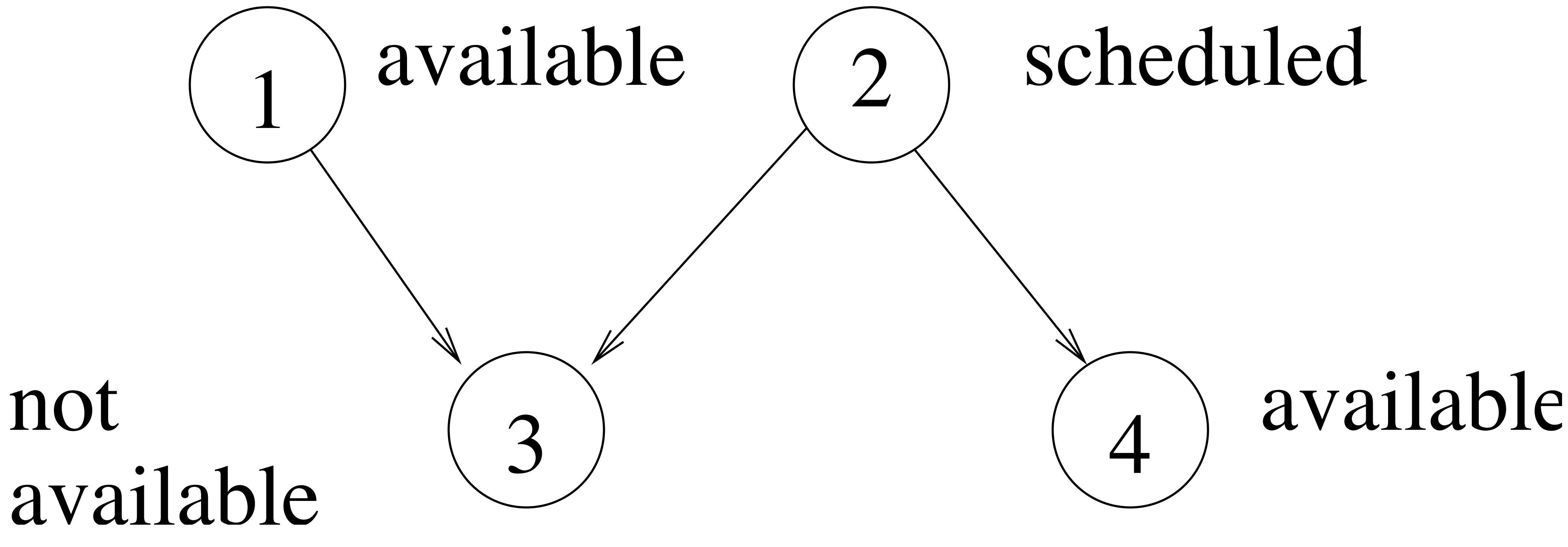
- Relied on unrealistic exhaustive search f

Technique relied on features that were machine specific

- Questionable portability

Little head room in basic bock scheduler

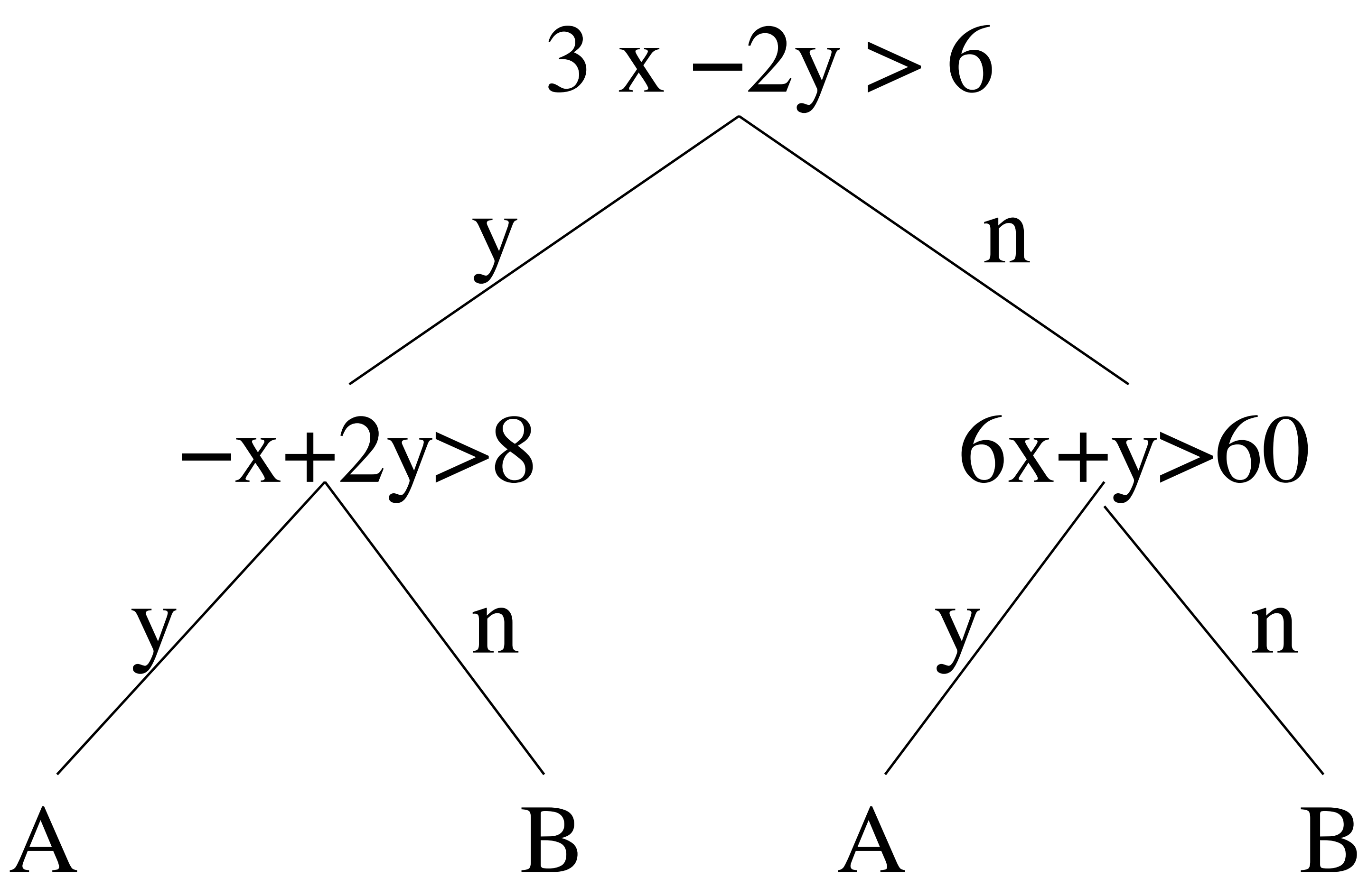
- Hard to see benefit over standard schemes.

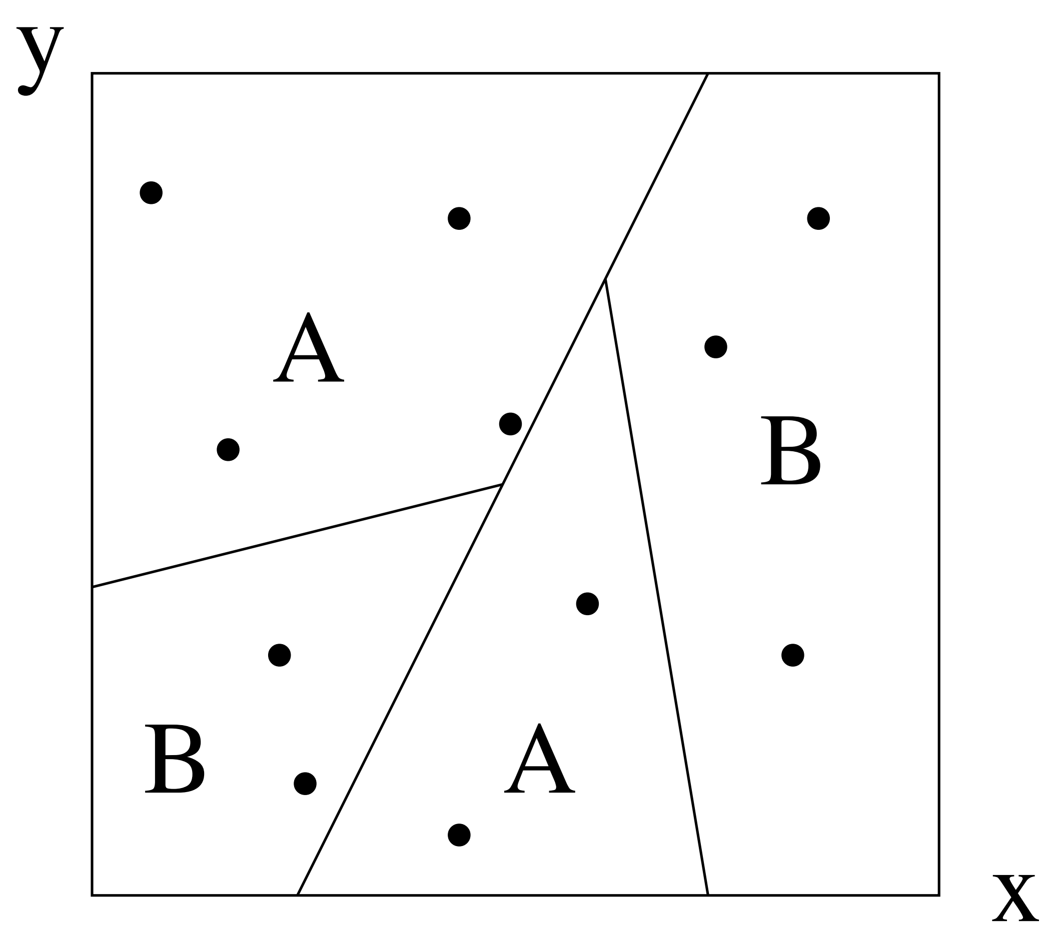


Monsifrot: Unroll? [12]

```
do i = 2, 100

    a(i) = a(i) + a(i−1) + a(i+1)

enddo
```

statements	1
aritmetic op	2
iterations	99
array access	4
resuses	3
ifs	0





Monsifrot Results

85% accuracy

- Better at picking negative cases due to bias in training set

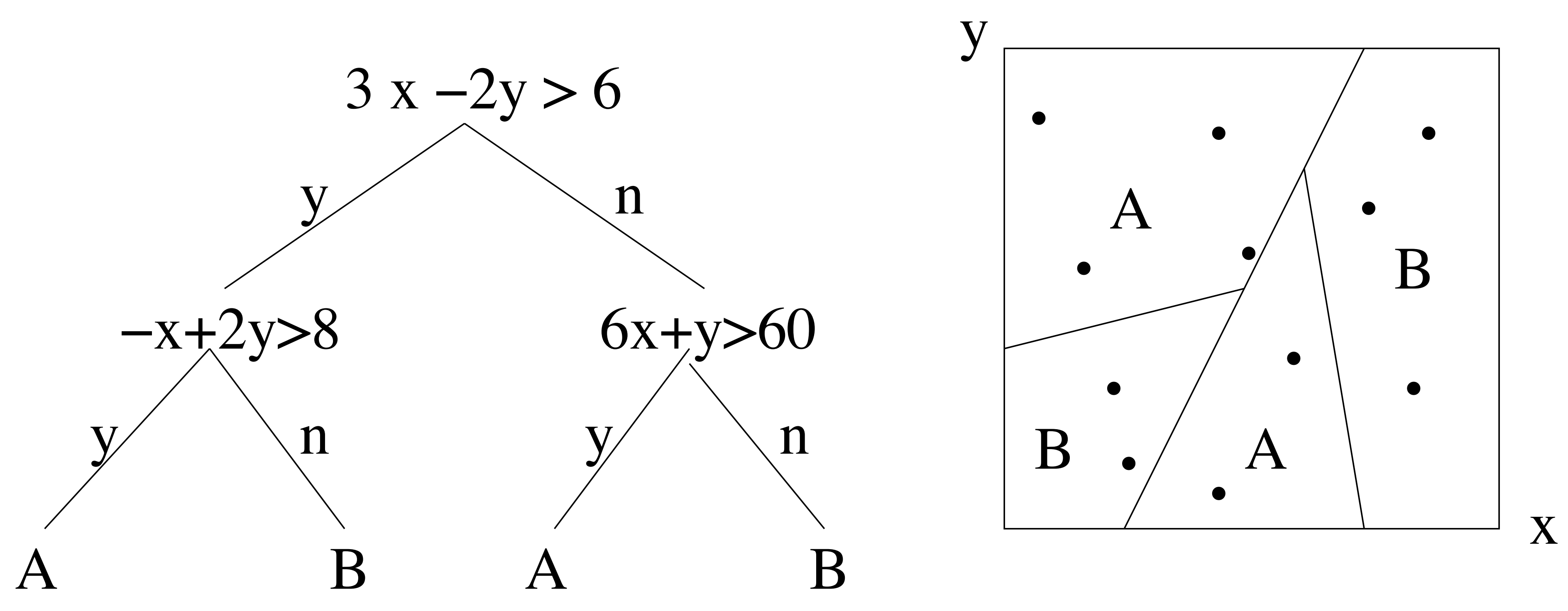
4% improvement over g77.

g77 is an easy compiler to improve upon.

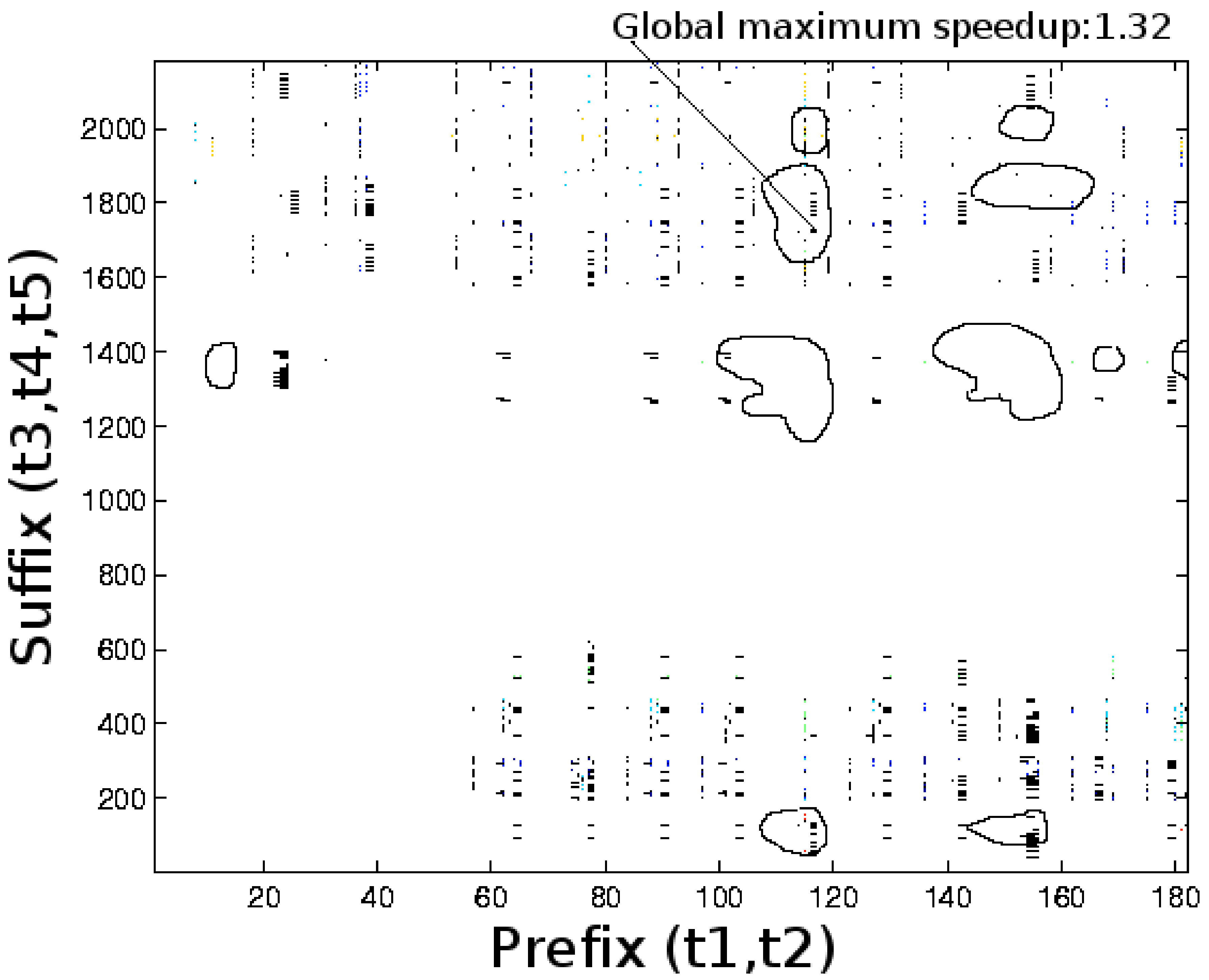
Only beneficial on 22% of benchmarks

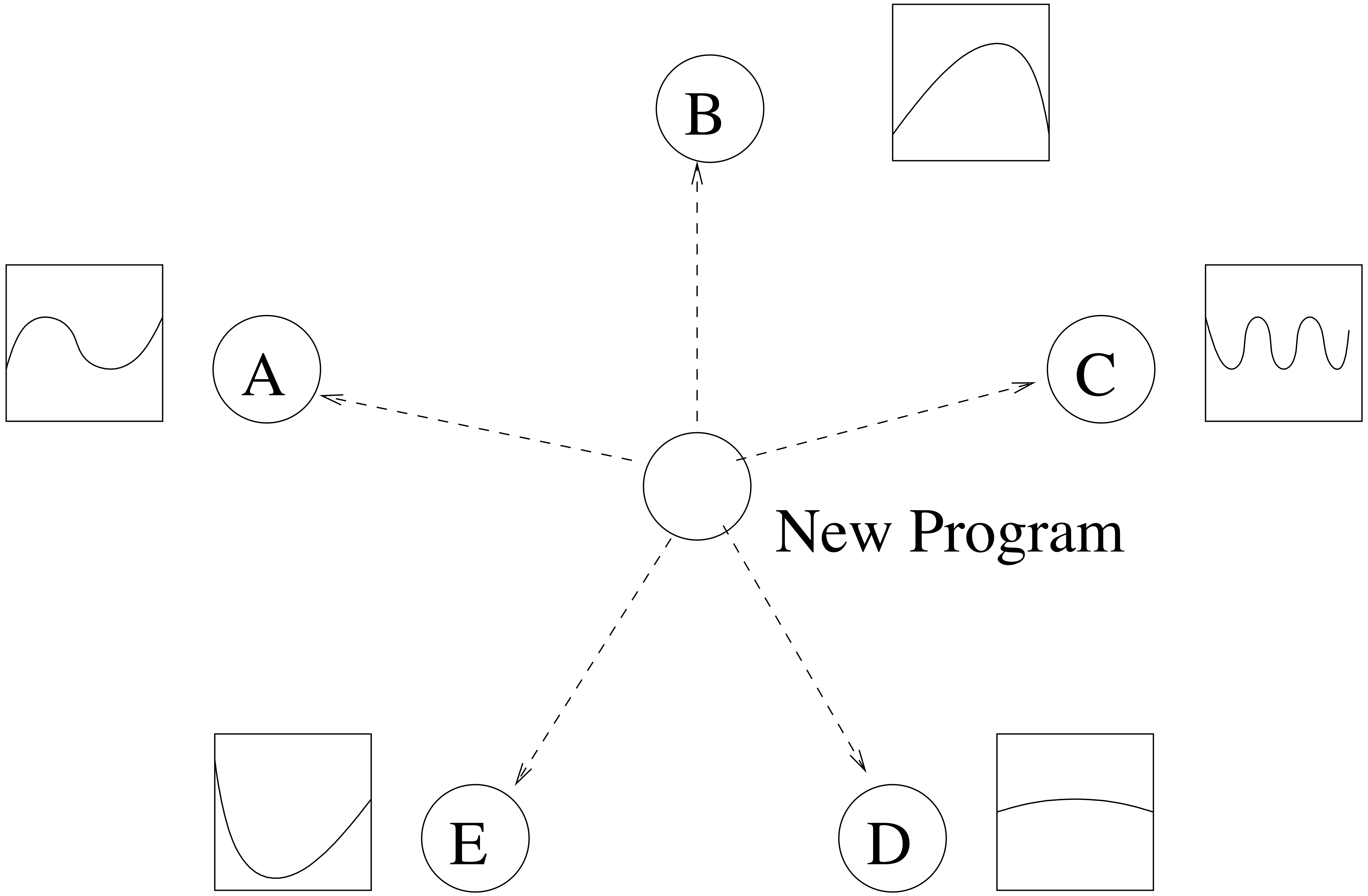
Basic approach - unroll factor not considered.

- Leather[16] looked at unrolll factor



Using ML to focus search [8]





Capture probability distribution of good transformations per benchmark
Then see if new program looks like existing ones and then use its distribution

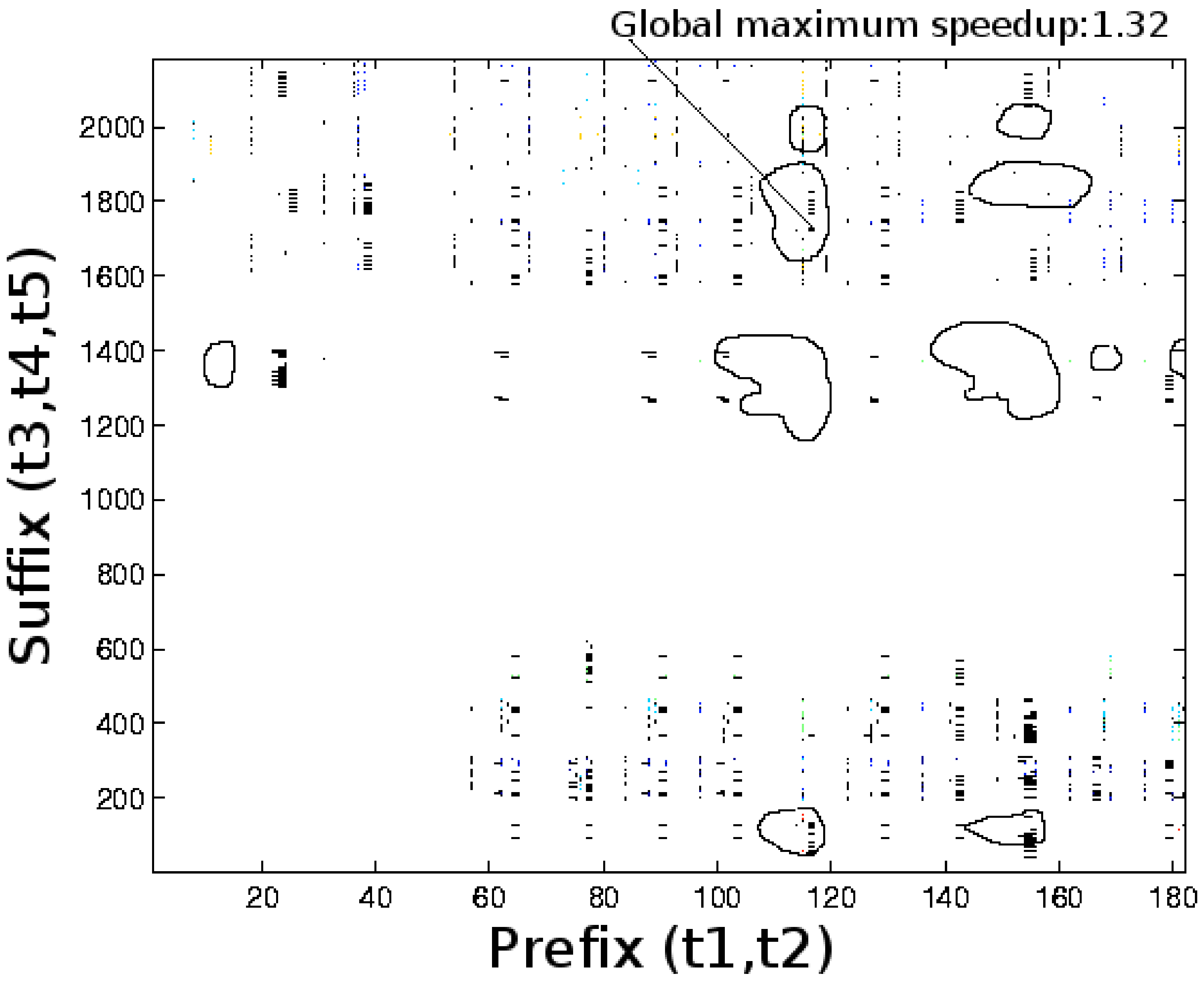
Features

Features
for loop is simple?
for loop is nested?
for loop is perfectly nested?
for loop has constant lower bound?
for loop has constant upper bound?
for loop has constant stride?
for loop has unit stride?
number of iterations in for loop
loop step within for loop
loop nest depth
no. of array references within loop
no. of instructions in loop
no. of load instructions in loop
no. of store instructions in loop
no. of compare instructions in loop
no. of branch instructions in loop
no. of divide instructions in loop
no. of call instructions in loop
no. of generic instructions in loop
no. of array instructions in loop
no. of memory copy instructions in loop
no. of other instructions in loop
no. of float variables in loop
no. of int variables in loop
both int and floats used in loop?
loop contains an if-construct?
loop contains an if statement in for-construct?
loop iterator is an array index?
all loop indices are constants?
array is accessed in a non-linear manner?
loop strides on leading array dimensions only?
loop has calls?
loop has branches?
loop has regular control flow?

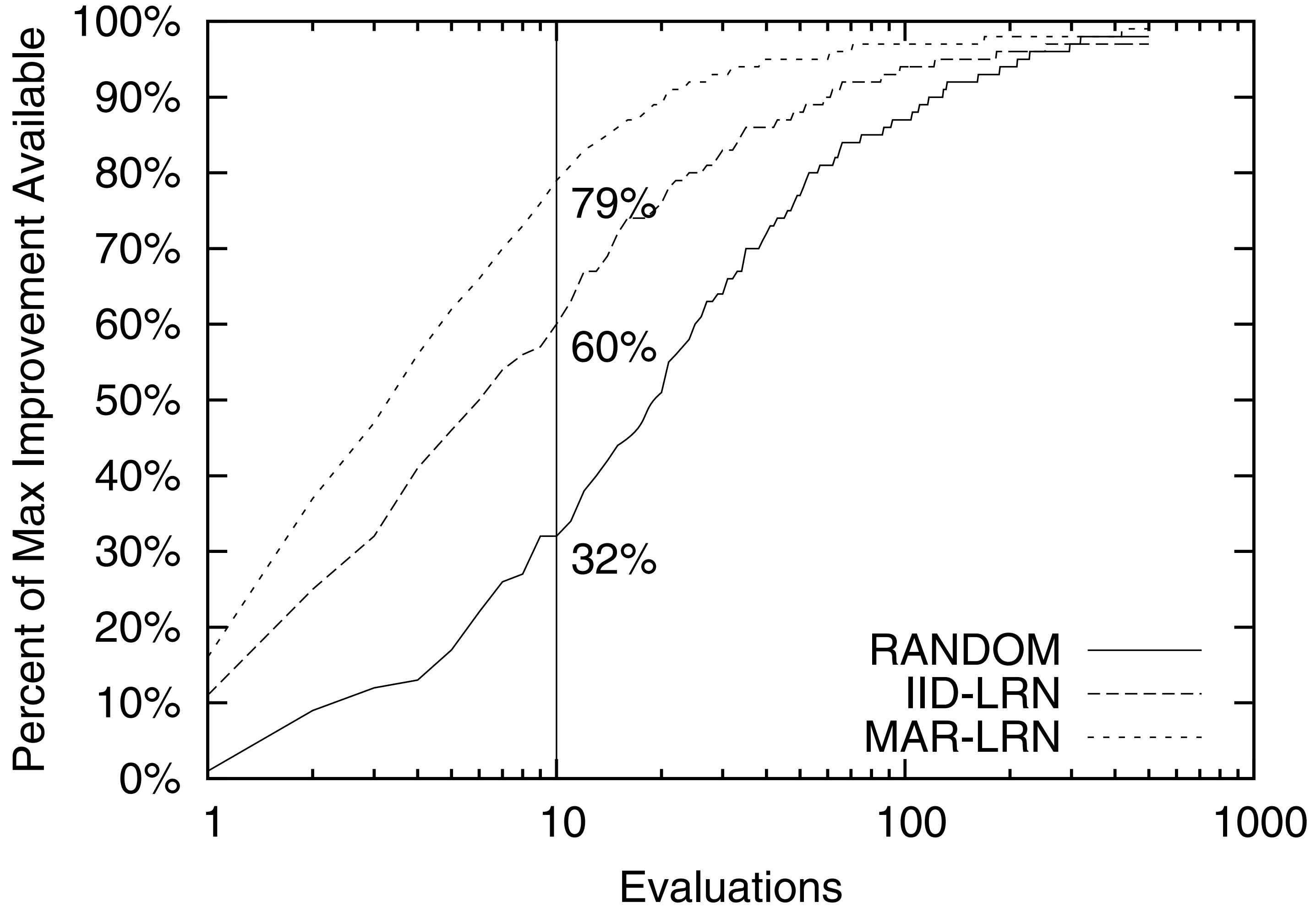
Probability models

$$P(s_1, s_2, \ldots, s_L) = \prod_{i=1}^{L} P(s_i).$$

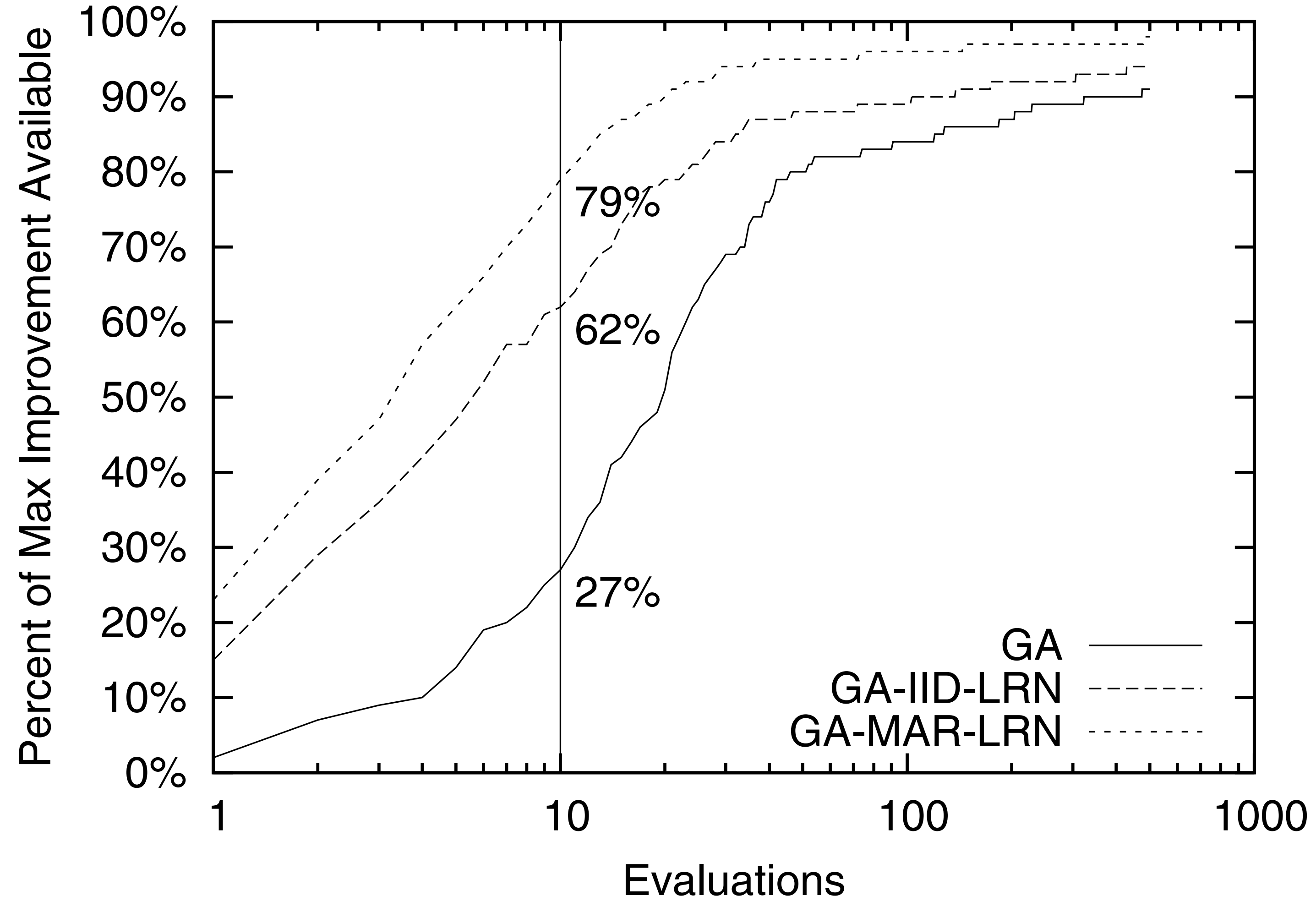
$$P(\mathbf{s}) = P(s_1) \prod_{i=2}^{L} P(s_i | s_{i-1}).$$



Search improvement based on models



(a) **TI: Random**



(b) **TI: GA**

What is compilation

Why do we need new technicues

Automation

Search/ Auto-tuning/ Iterative compilation

ML for compilation

Features, models and applications

Summary

Taxonomy of ML in compiler optimisation [2]

Approach	Problem	Application Domains	Models
Supervised learning	Regression	Useful for modelling continuous values, such as estimating execution time, speedup, power consumption, latency etc.	Linear/non-linear regression, artificial neural networks (ANNs), support vector machines (SVMs).
	Classification	Useful for predicting discrete values, such as choosing compiler flags, #threads, loop unroll factors, algorithmic implementations etc.	K-nearest neighbour (KNN), decision trees, random forests, logical regression, SVM, Kernel Canonical Correlation Analysis, Bayesian
Unsupervised learning	Clustering	Data analysis, such as grouping profiling traces into clusters of similar behaviour	K-means, Fast Newman clustering
	Feature engineering	Feature dimension reduction, finding useful feature representations	Principal component analysis (PCA), autoencoders
Online learning	Search and self-learning	Useful for exploring a large optimisation space, runtime adaption, dynamic task scheduling where the optimal outcome is achieved through a series of actions	Genetic algorithm (GA), genetic programming (GP), reinforcement learning (RL)

Features, Models, Applications

Features are critical to success

- Hand-coded vs automatic techniques Leather[2014] and later lectures

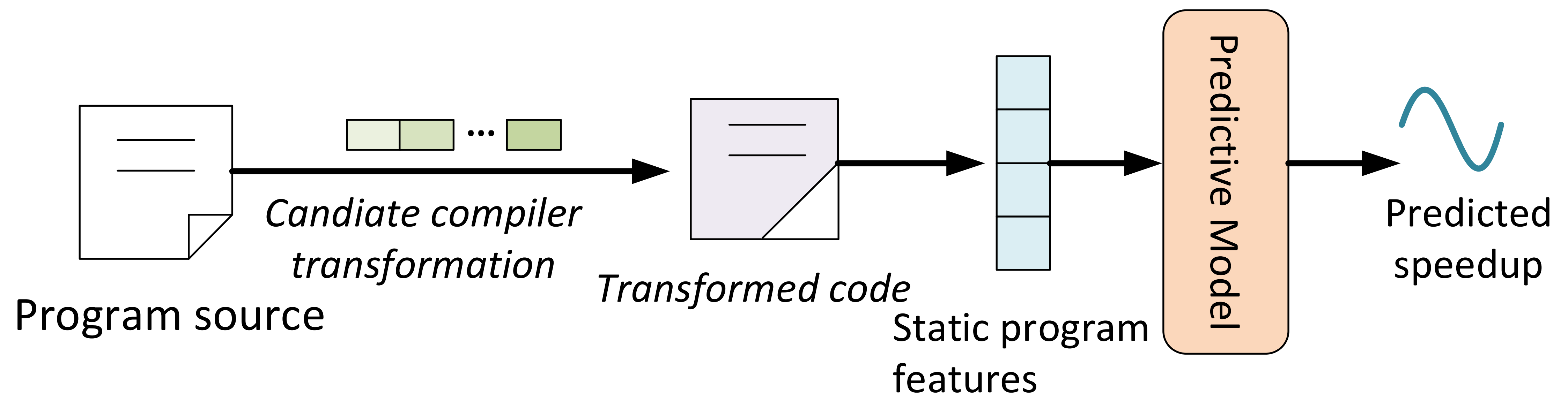
Model selection - less important than good data

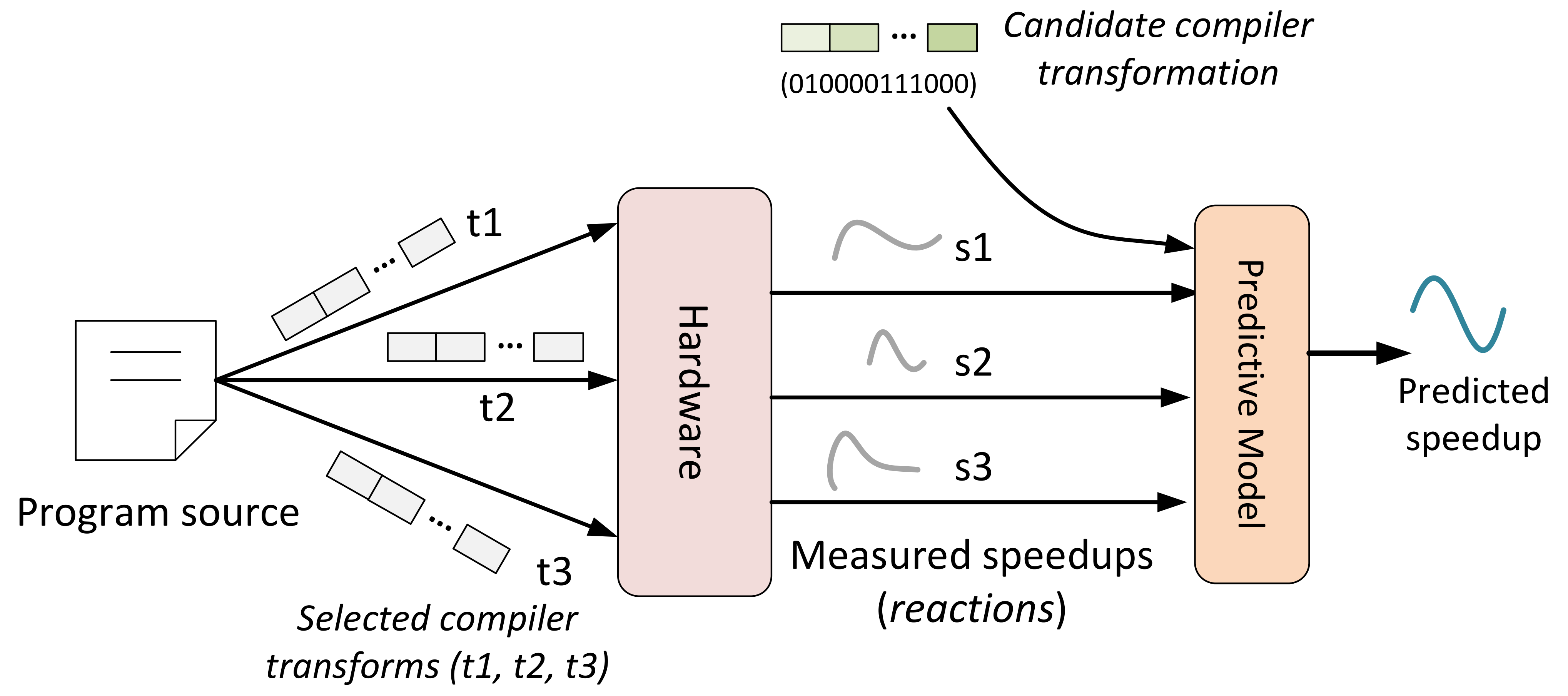
- Linear regression, KNN to SVM, GaussianProcesses, DNN
- Online vs Offline, active learning, (un) supervised vs reinforcement learning

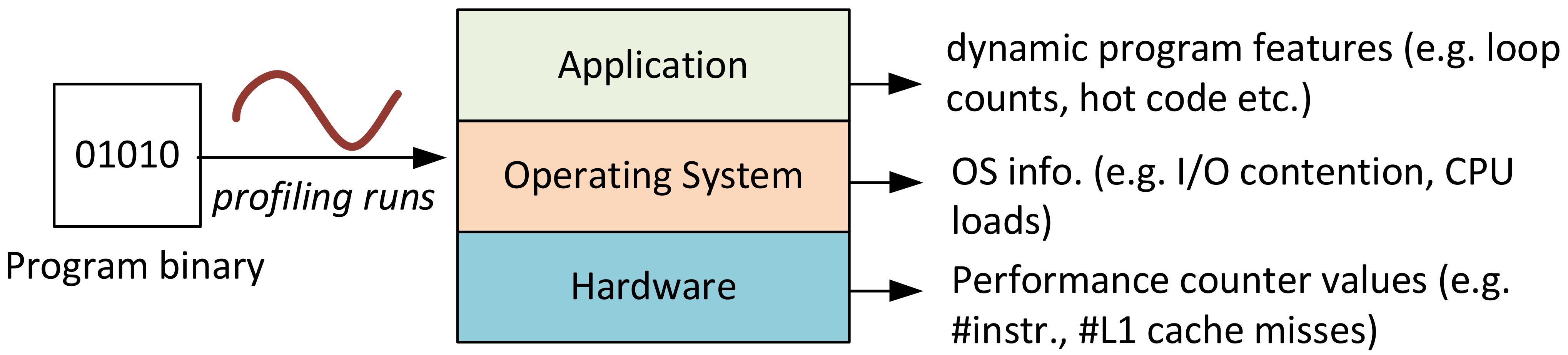
Applications

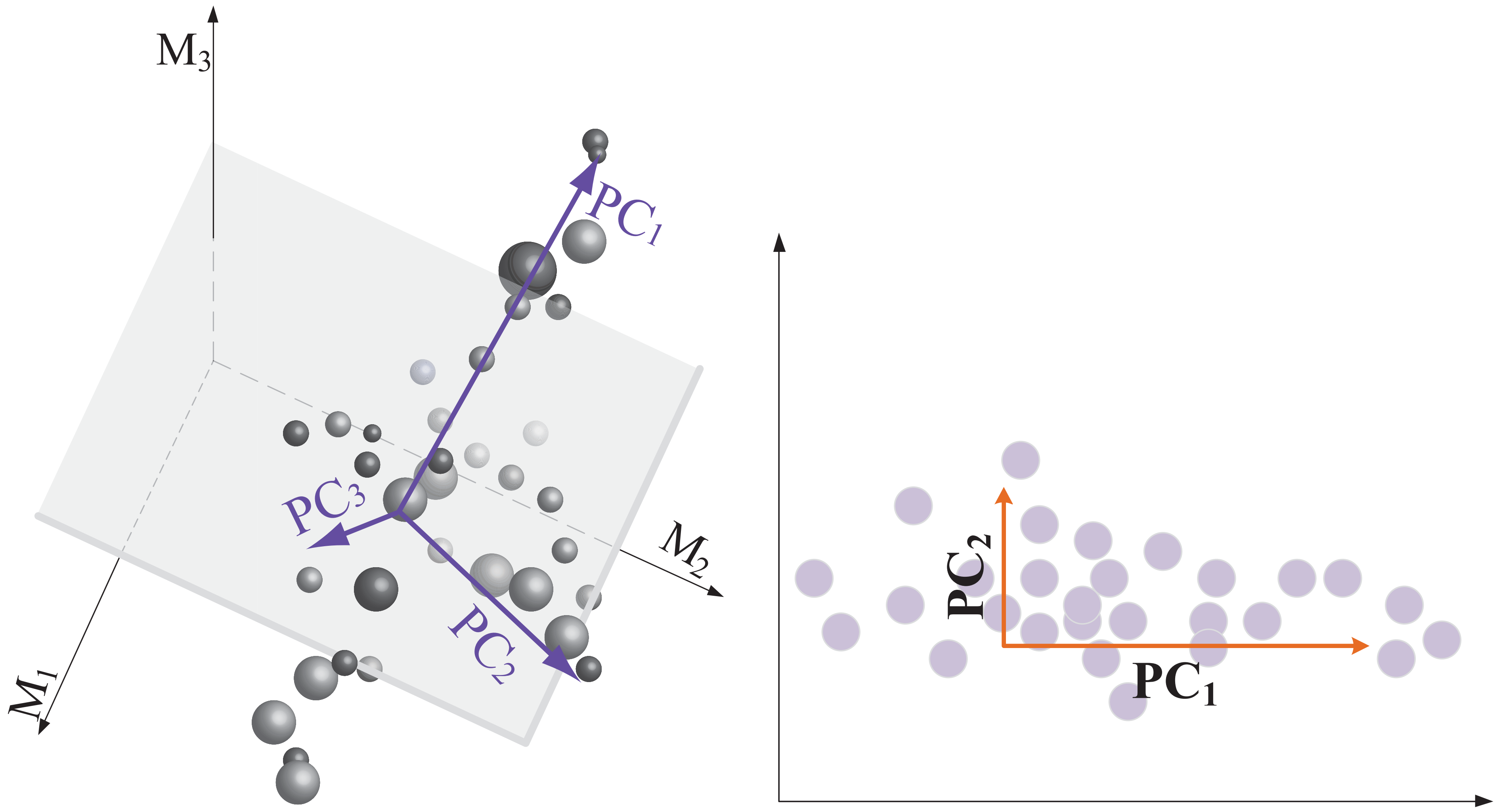
- Beyond flag selection: parallelisation, GPU opt, mapping

Features [2]



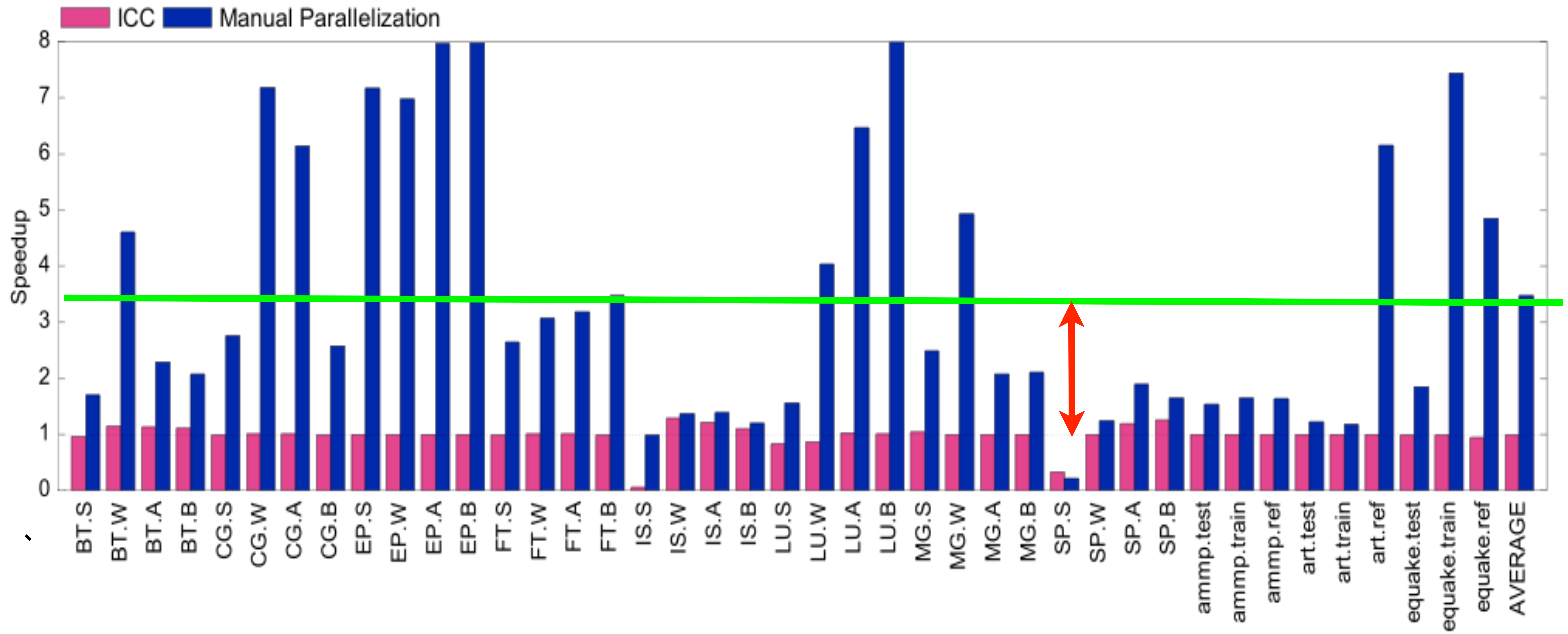






(a) Original feature space (b) Reduced feature space

Application: Automatic parallelization [13]



Manual: 3.4x speedup.   Automatic 0.9x !!!

Why poor performance?

equake **(75%)**

```
for (i = 0; i < nodes; i++) {
  Anext = Aindex[i];
  Alast = Aindex[i + 1];

  sum0 = A[Anext][0][0]*v[i][0] +
         A[Anext][0][1]*v[i][1] +
         A[Anext][0][2]*v[i][2];
  sum1 = ...

  Anext++;
  while (Anext < Alast) {
    col = Acol[Anext];

    sum0 += A[Anext][0][0]*v[col][0] +
            A[Anext][0][1]*v[col][1] +
            A[Anext][0][2]*v[col][2];
    sum1 += ...

    w[col][0] += A[Anext][0][0]*v[i][0] +
                 A[Anext][1][0]*v[i][1] +
                 A[Anext][2][0]*v[i][2];
    w[col][1] += ...
    Anext++;
  }
  w[i][0] += sum0;
  w[i][1] += ...
}
```

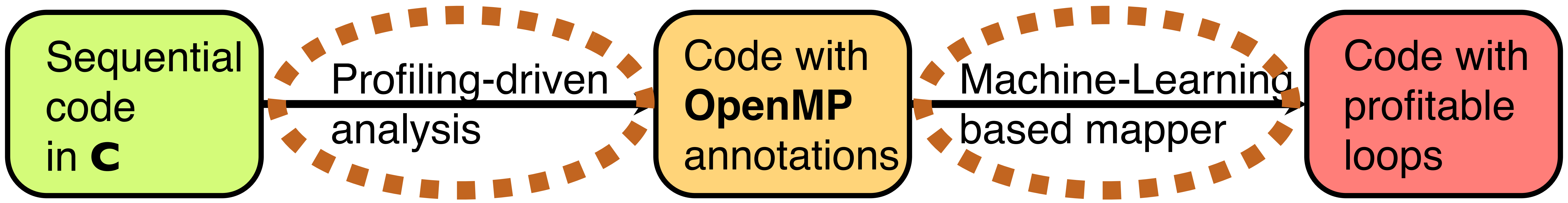
Static analysis finds no parallelism

Static analysis :
- indirect array accesses
- reductions
- pointer aliasing
- dynamic allocation

Profiling shows it is parallel

Profiling and ML mapping

- Key point: restrictions of static analysis can be overcome using precise, dynamic information
- How?
- Instrument the compiler representation
- Track all read/writes to memory
- Dynamically reconstruct precise view of control and data flow
  - Identify parallel loops
  - Unsafe so check with user

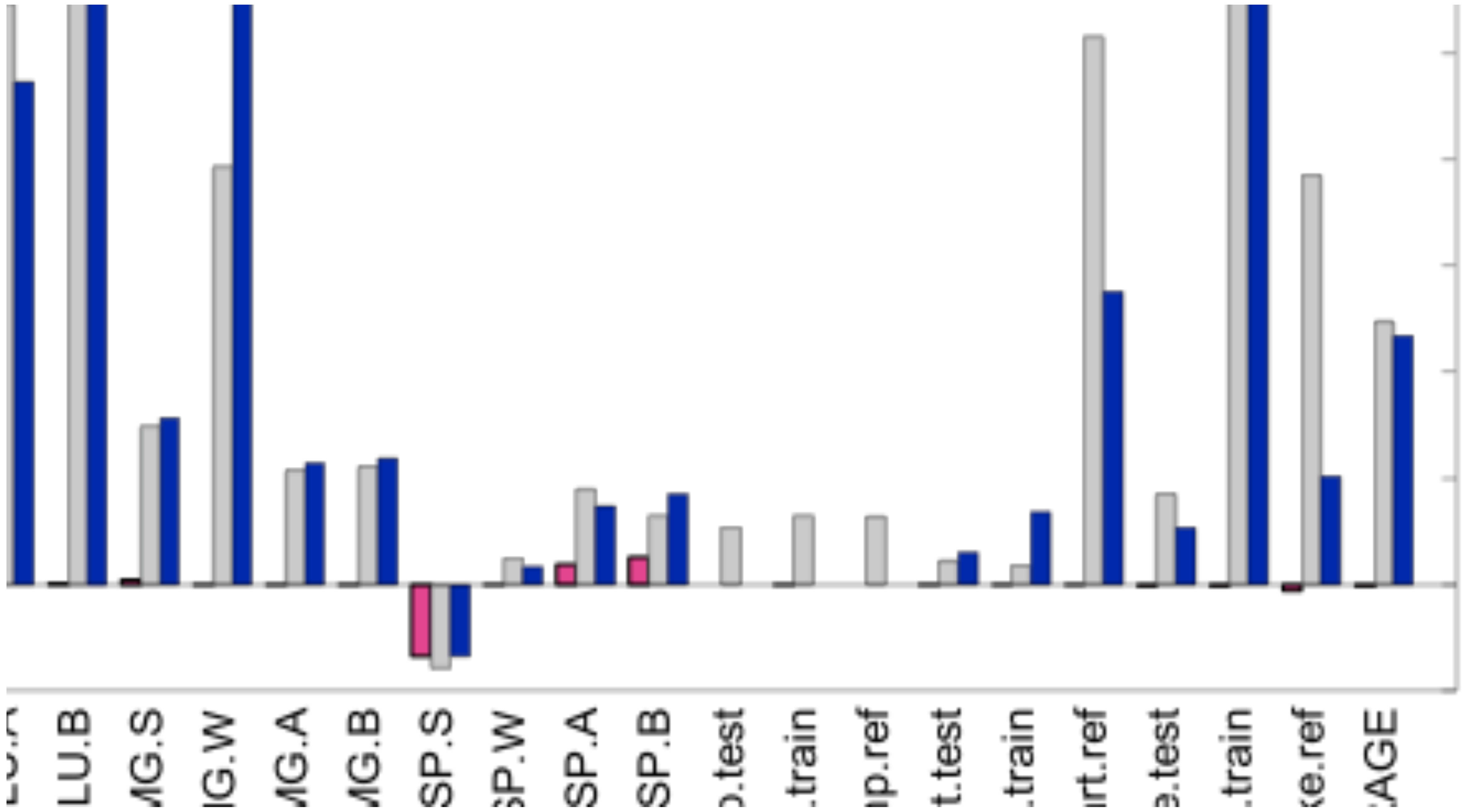


ML good profitability heuristic

ICC good number of loops
But poor sequential time coverage
– Majority of loops too short to be profitable

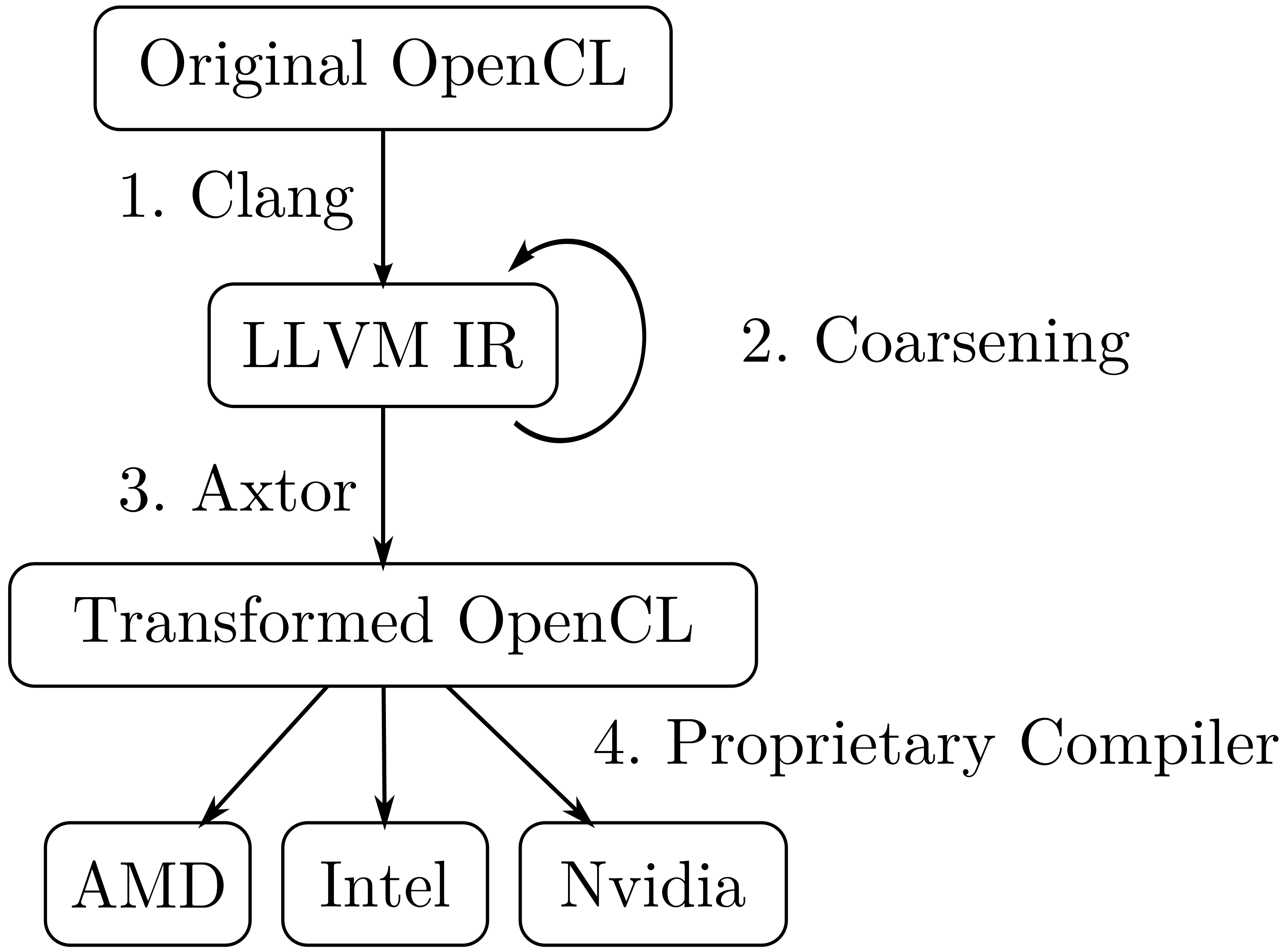
ML:96% of hand-parallelized

Application	icc #loops(%cov)		Profile-driven #loops(%cov)		Manual #loops(%cov)	
bt	72	(18.6%)	205	(99.9%)	54	(99.9%)
cg	16	(1.10%)	28	(93.1%)	22	(93.1%)
ep	6	(<1%)	8	(99.9%)	1	(99.9%)
ft	3	(<1%)	37	(88.2%)	6	(88.2%)
is	8	(29.4%)	9	(28.5%)	1	(27.3%)
lu	88	(65.9%)	154	(99.7%)	29	(81.5%)
mg	9	(4.70%)	48	(77.7%)	12	(77.7%)
sp	178	(88.0%)	287	(99.6%)	70	(61.8%)
equake	29	(23.8%)	69	(98.1%)	11	(98.0%)
art	16	(30.0%)	31	(85.6%)	5	(65.0%)
ammp	43	(<1%)	21	(1.40%)	7	(84.4%)



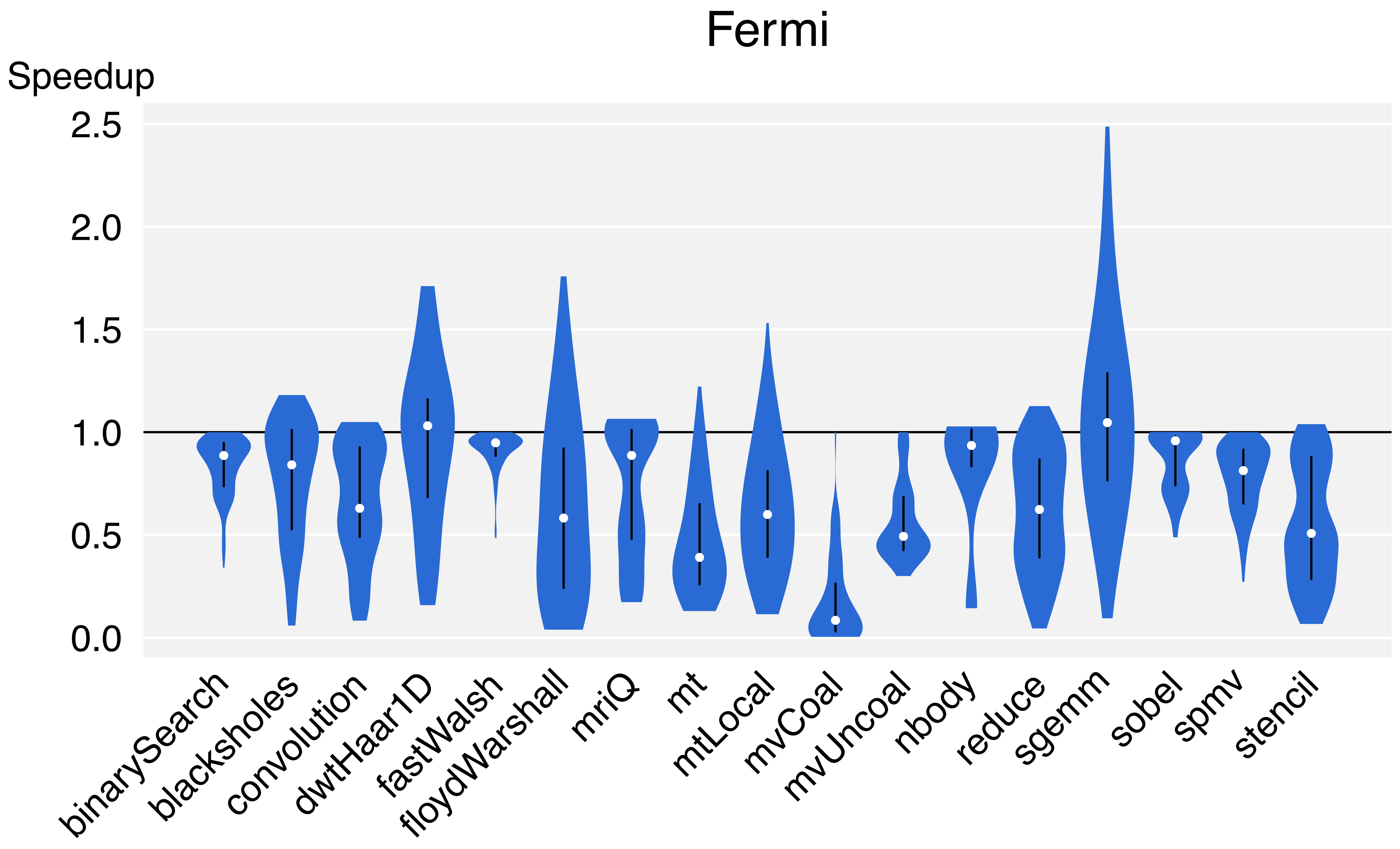
Using ML for GPU optimisation

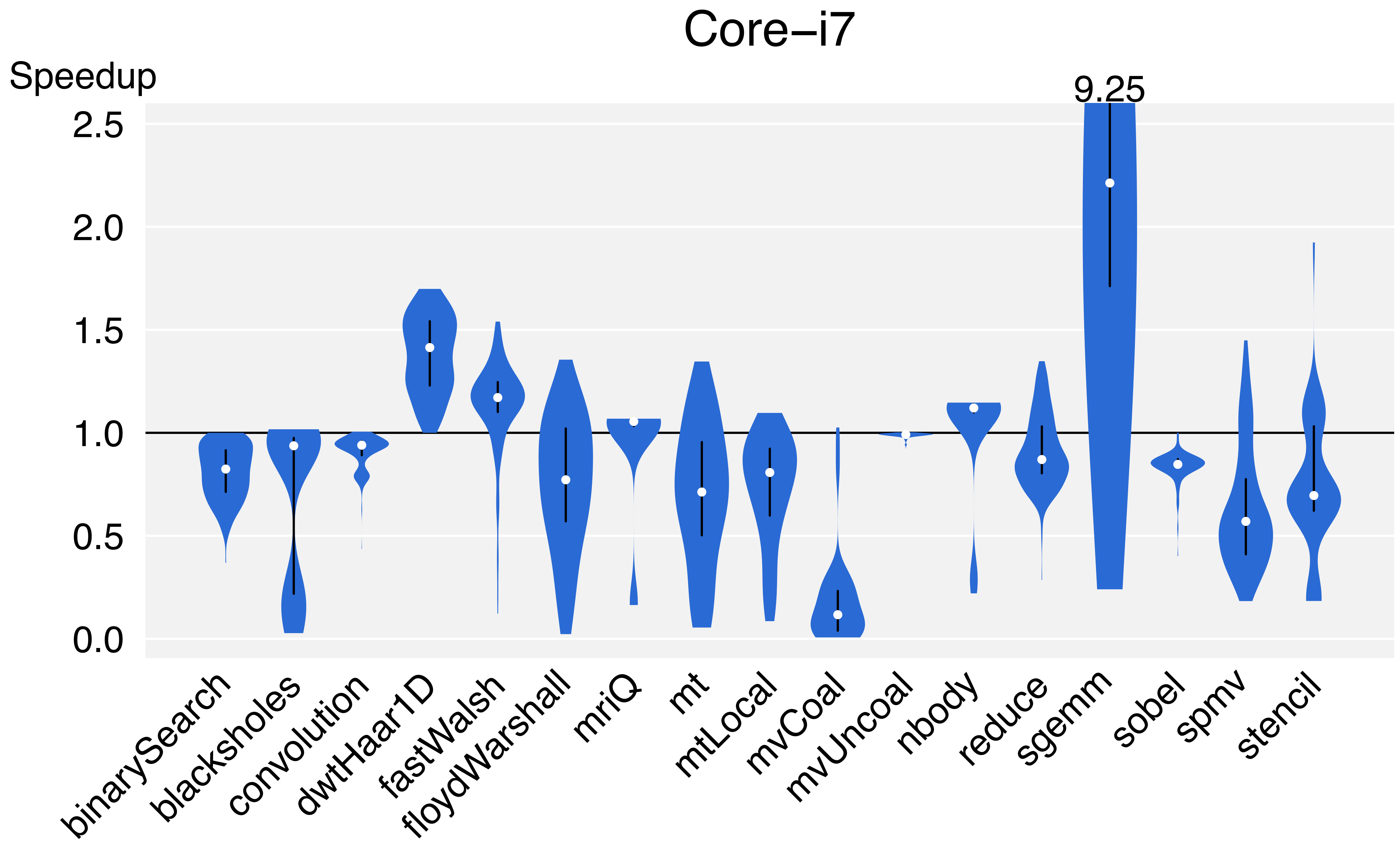
- 3 transformations
  - Thread coarsening: using divergence analysis
  - Stride optimisations
  - Work group size

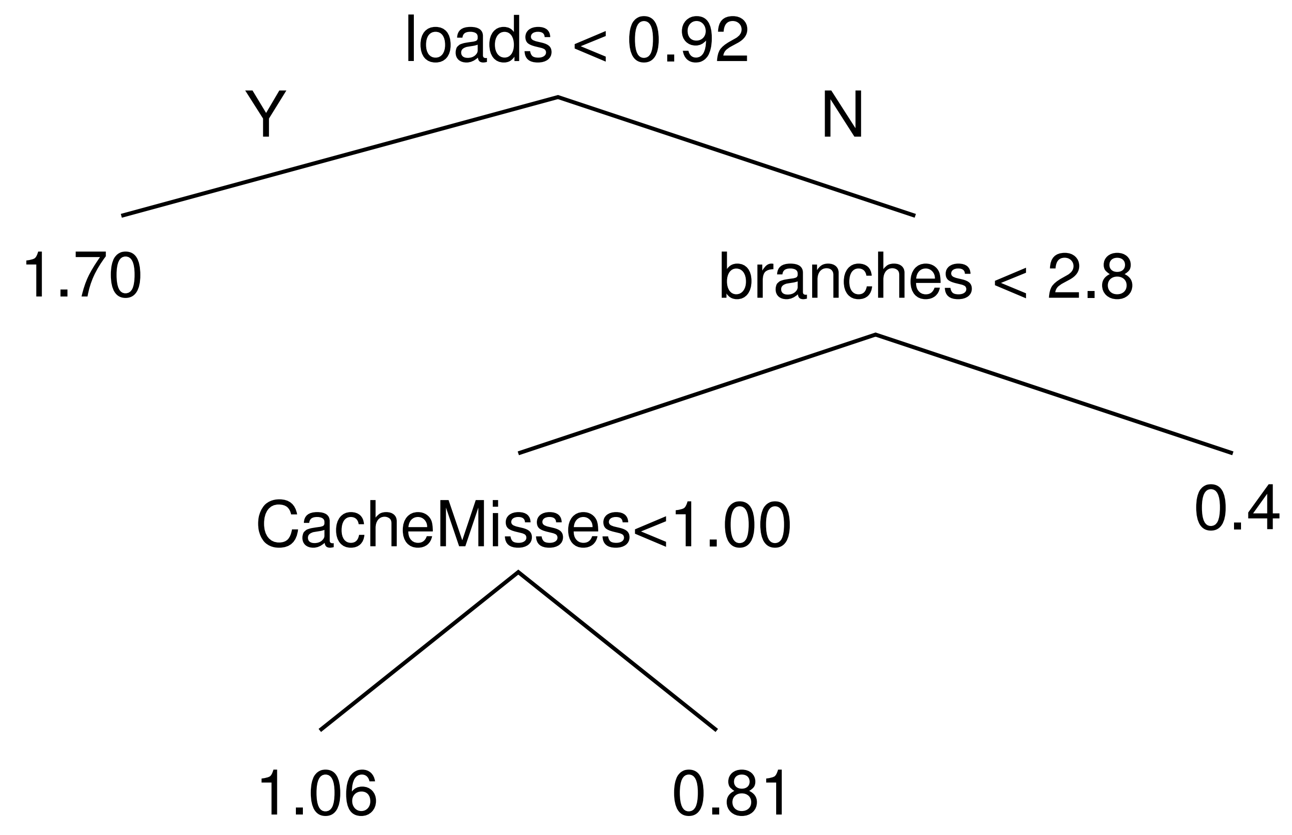


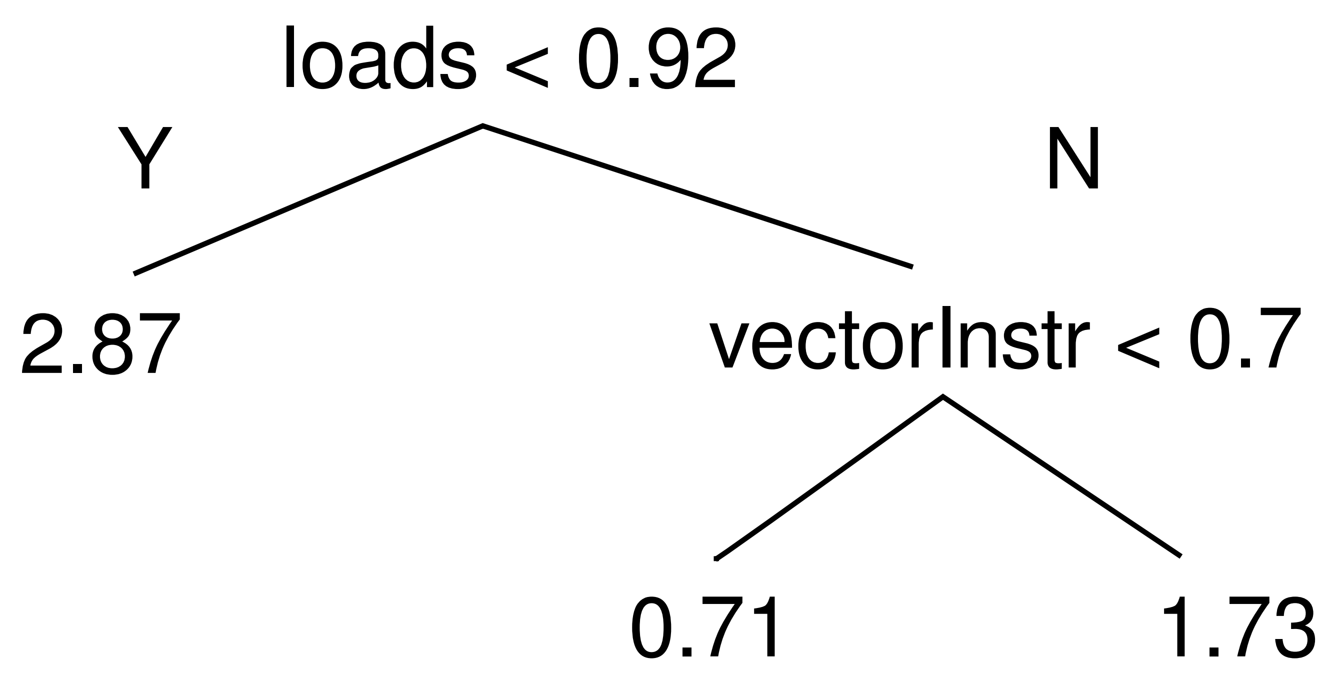
Can we use data driven approach to EXPLAIN behaviour? [14]
- later optimise [15]

Best optimisation hard to find









What is compilation

Why do we need new technicues

Automation

Search/ Auto-tuning/ Iterative compilation

ML for compilation

Features, models and applications

Summary

Old School ML for compilers

- Since Cavazos 1997 hundreds of paper
- Too many on flag selecting
  - Too much interest in models rather than data
- Excellent for profitability heuristics
  - Hand-written analytic heuristics usually pointless
  - Not used for anything involving correctness
- Key issue of transfer often missed
- Since 2010s
  - Interest in auto-feature selection
  - Beyond classification/regression - generative techniques for regression

Overview

- This lecture: Motivation and brief survey of auto-tuning/machine learning for compilers
- Next L2: Program rewriting schemes - e-graphs and equality saturation
- L3: Program embeddings and Graph Neural Networks
- L4: Program synthesis and neural synthesis
- L5: Neural Machine Translation,Transformers and Large language models

Bibliography

Surveys

- [1] AH Ashouri, W Killian, J Cavazos, G Palermo, C Silvano, A survey on compiler autotuning using machine learning, ACM Computing Surveys, 2018

- [2] Z Wang M O'Boyle, Machine Learning in Compiler Optimization, Proc IEEE 2018

Papers

- [3] K Cooper, L Torczon Engineering a compiler, Elsevier 2011

- [4] F Bodin, T Kisuki, P M W Knijnenburg, M F P O'Boyle, E Rohu, Iterative compilation in a non-linear optimisation space workshop on profile directed compilation 1998

- [5] R C Whaley, J J Dongarra, Automatically tuned linear algebra software SC 1998

- [6] N Gloy, Z Wang, C Zhang, B Chen, M Smith Profile directed optimisation with statistical profiles 1997

- [7] K Cooper, PJ Schiekle, D Subramanian. Optimizing for reduced code space using genetic algorithms, LCTES 1999

Bibliography

- [8] F Agakov, E Bonilla, J Cavazos, B Franke, G Fursin, MFP O'Boyle, J Thomson, M Toussaint, CKI Williams, Using machine learning to focus iterative compilation, CGO 2006.
- [9] J Ansel, S Kamil, K Veeramachaneni, J Ragan-Kelly, J Bosboom,U-M O'Reilly, S Amaraasinghe, Opentuner: An extensible framework for program autotuning PACT 2014
- [10] R Vuduc, JW Demmel, J Blimes, Statistical models for automatic performance tuning, ICCS 2001
- [11] J Moss, P Utgoff, J Cavazos, D Precup, D Stefanovic, C Drodley, D Scheed, Learning to schedule straight-line code, NIPS 1997
- [12] A Monsifrot, F Bodin, R Quinou, A machine learning approach to automatics production of compiler heuistics, AIMSA 2002
- [13] G Tournavitis, Z Wang, B Franke, MFP O'Boyle, Towards a holistic approach to auto-parallelization: integrating profile-directed parallelism detection and machine-learning based mapping
- [14] A Magni, C Dubach, MFP O'Boyle, A large scale cross architecture evaluation of thread coarsening SC13
- [15] A Magni, C Dubach, MFP O'Boyle, Automatic optimisation of thread-coarsening for graphics processors PACT14
- [16] H Leather, E Bonilla, M O'Boyle, Automatic feature generation for machine learning based optimising compilation CGO 09

Bibliography

- [16] H Leather, E Bonilla, M O'Boyle, Automatic feature generation for machine learning based optimising compilation CGO 09
- [17] H Leather, M O'Boyle B Worton,Raced profiles: efficient selection of competing compiler optimizations LCTES 09