Rapidly Selecting Good Compiler Optimizations Using Performance Counters

John Cavazos[1] Grigori Fursin[2]

Felix Agakov[1] Edwin Bonilla[1]

Michael O'Boyle[1] Olivier Temam[2]

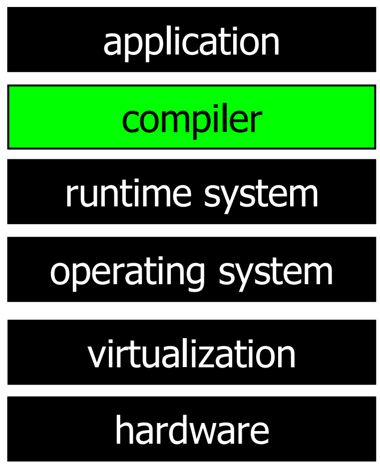
[1]*University of Edinburgh, UK*

[2]*INRIA Futurs, France*

Members of HiPEAC

Traditional Compilers

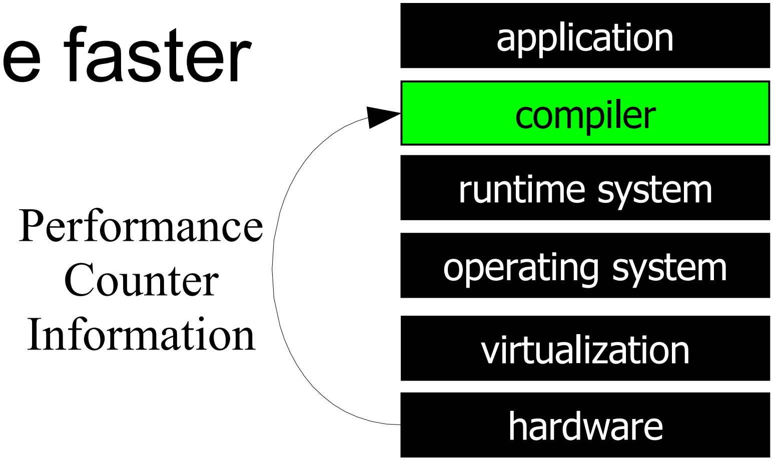
- "One size fits all" approach
- Tuned for average performance
- Aggressive opts often turned **off**
- Need to "understand" all layers below
  - Hard to model analytically



Solution

- Use performance counter characterization
  - Train model off-line
  - Counter values are “features” of program
  - Out-performs highest optimization setting in production quality compiler
  - 2 orders of magnitude faster

    than pure search

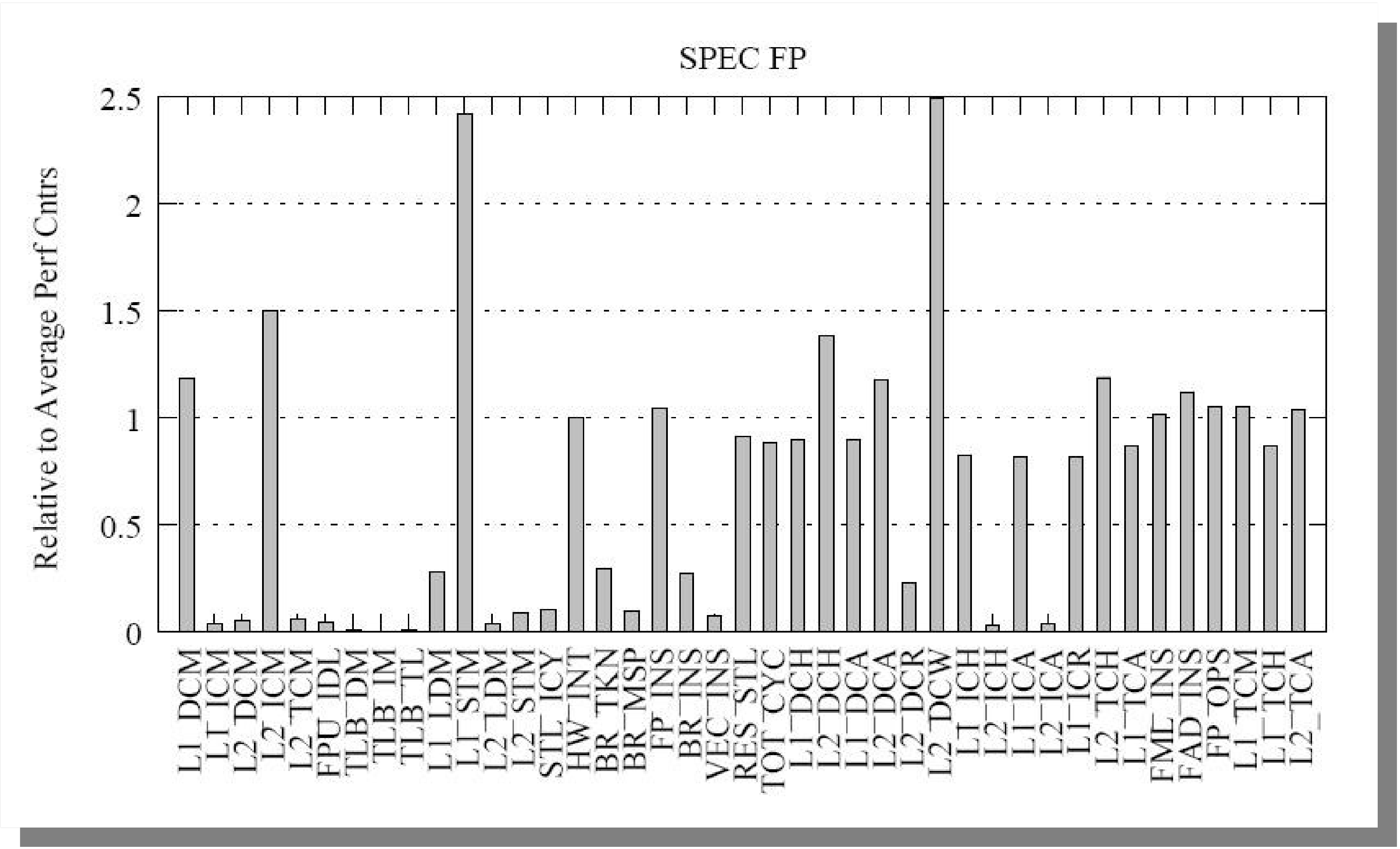


Performance Counters

- 60 counters available
- 5 categories
  - Floating point, Branch, L1 cache, L2 cache, TLB, Others
  - Examples:

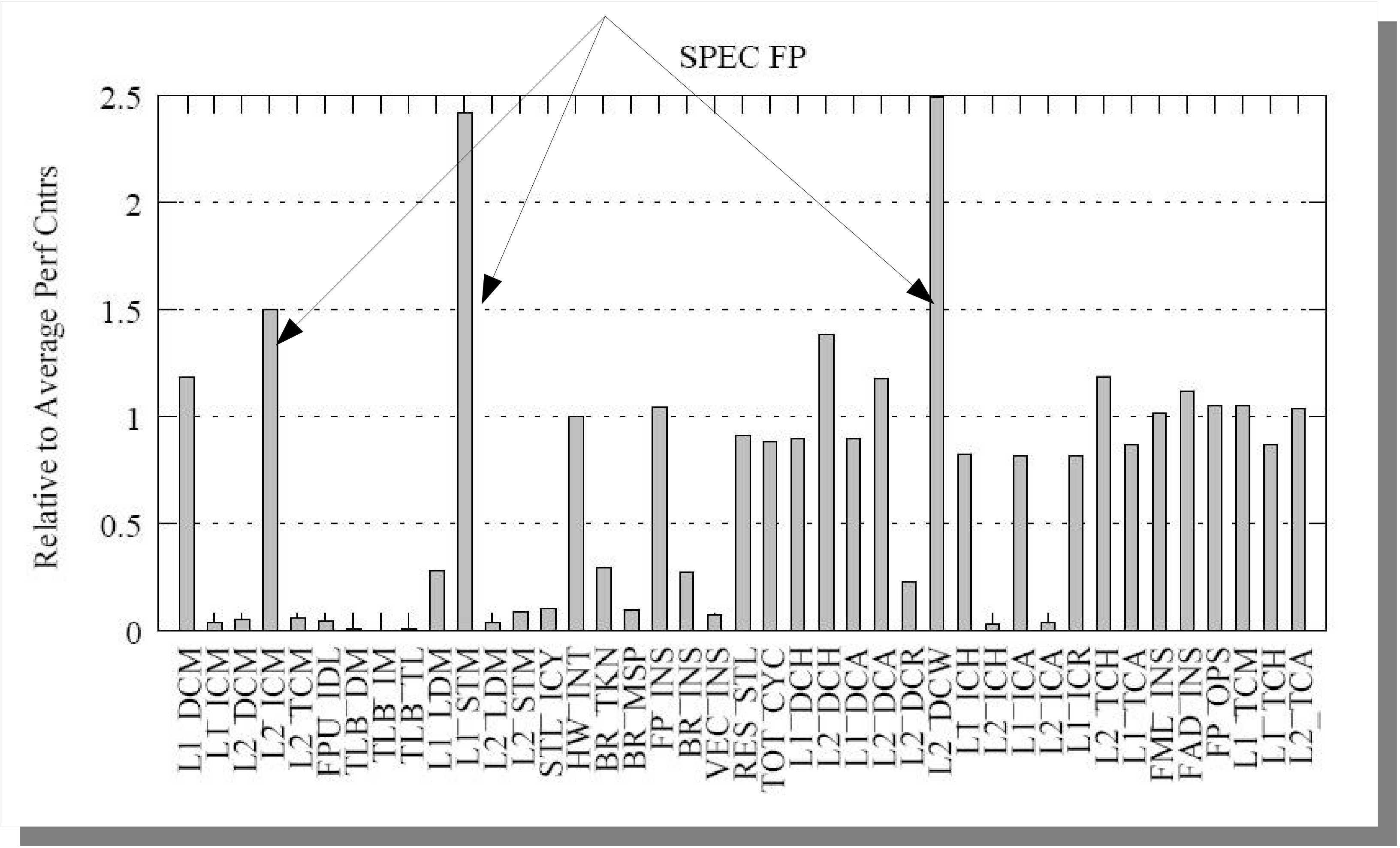
Mnemonic	Description	Avg Values
FPU_IDL	(Floating Unit Idle)	0.473
VEC_INS	(Vector Instructions)	0.017
BR_INS	(Branch Instructions)	0.047
L1_ICH	(L1 Icache Hits)	0.0006

Characterization of SPEC FP



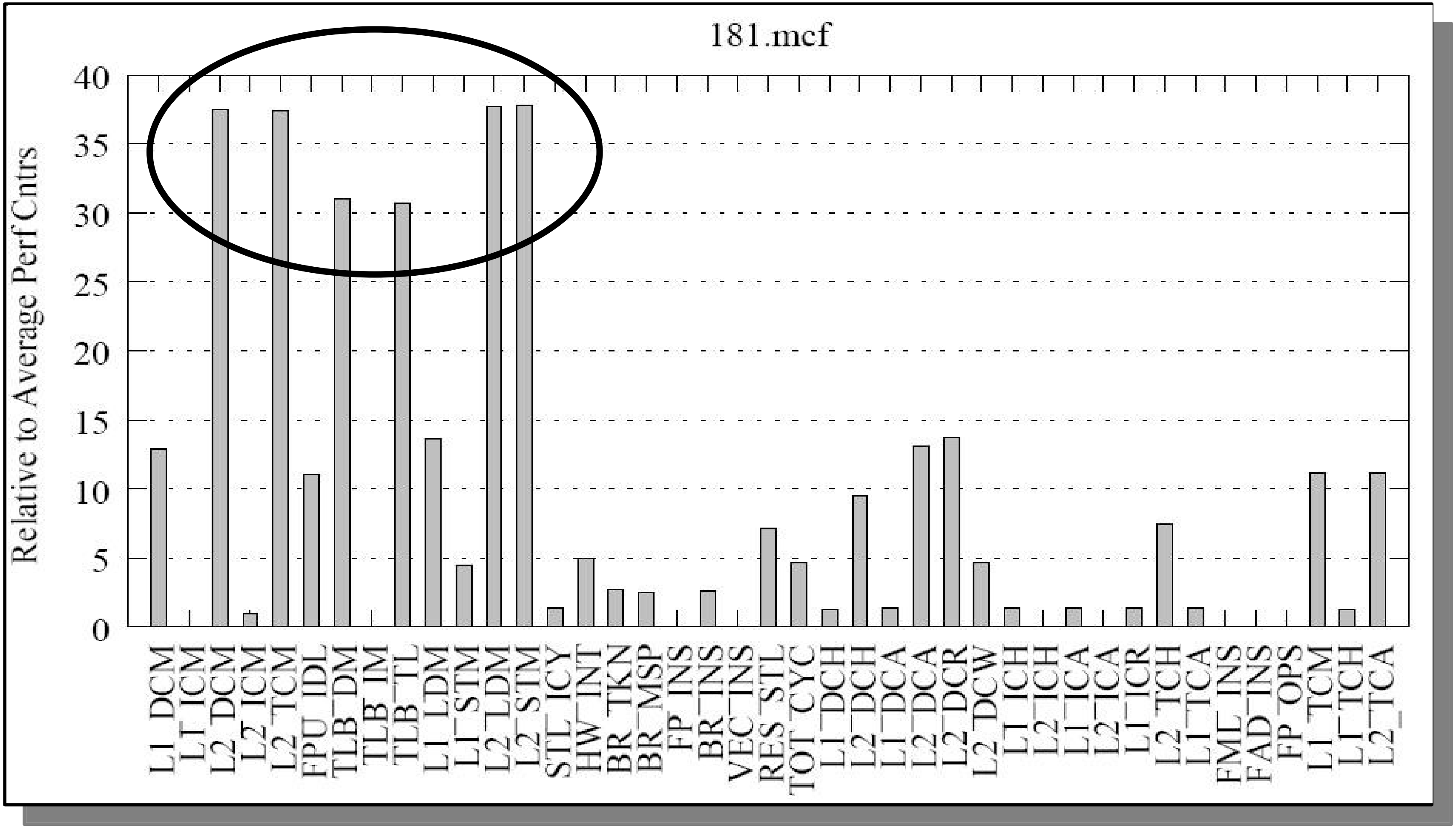
Characterization of SPEC FP

Larger number of L1 icache misses, L1 store misses and L2 D-cache writes



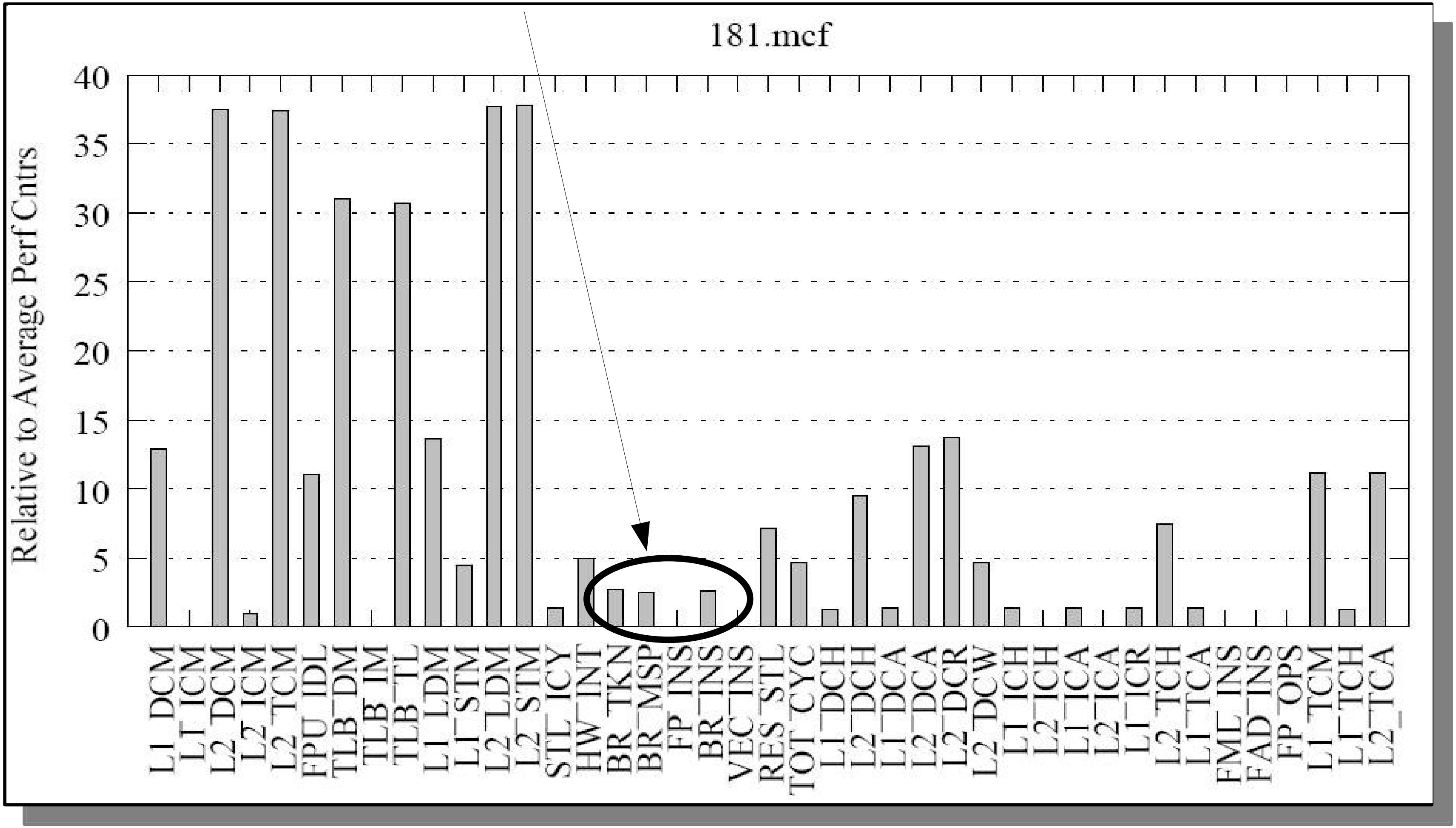
Characterization of 181.mcf

Problem: Greater number of memory accesses per instruction than average



Characterization of 181.mcf

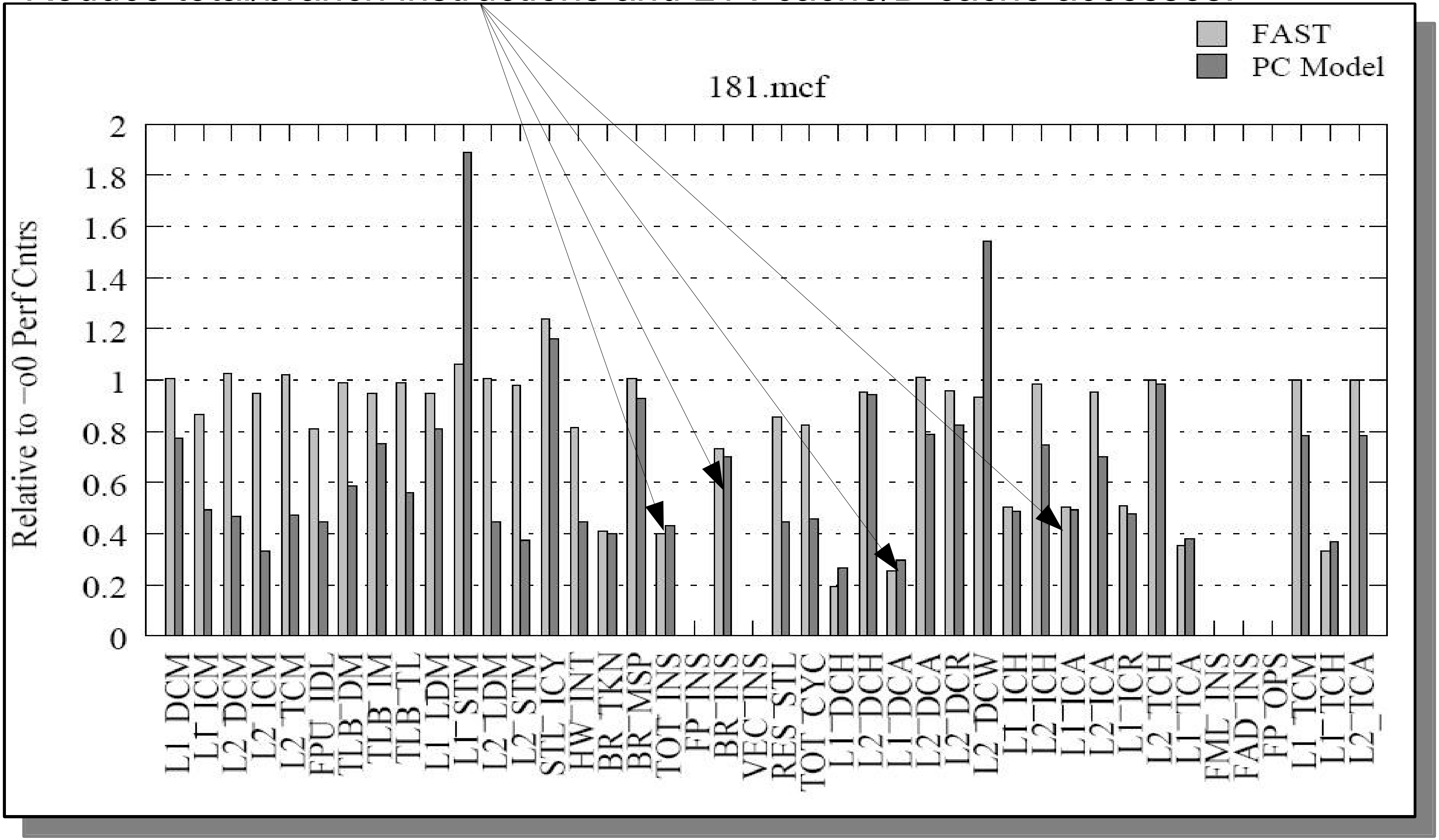
Problem: BUT also Branch Instructions



Characterization of 181.mcf

Use LNO (loop nest optimizations)

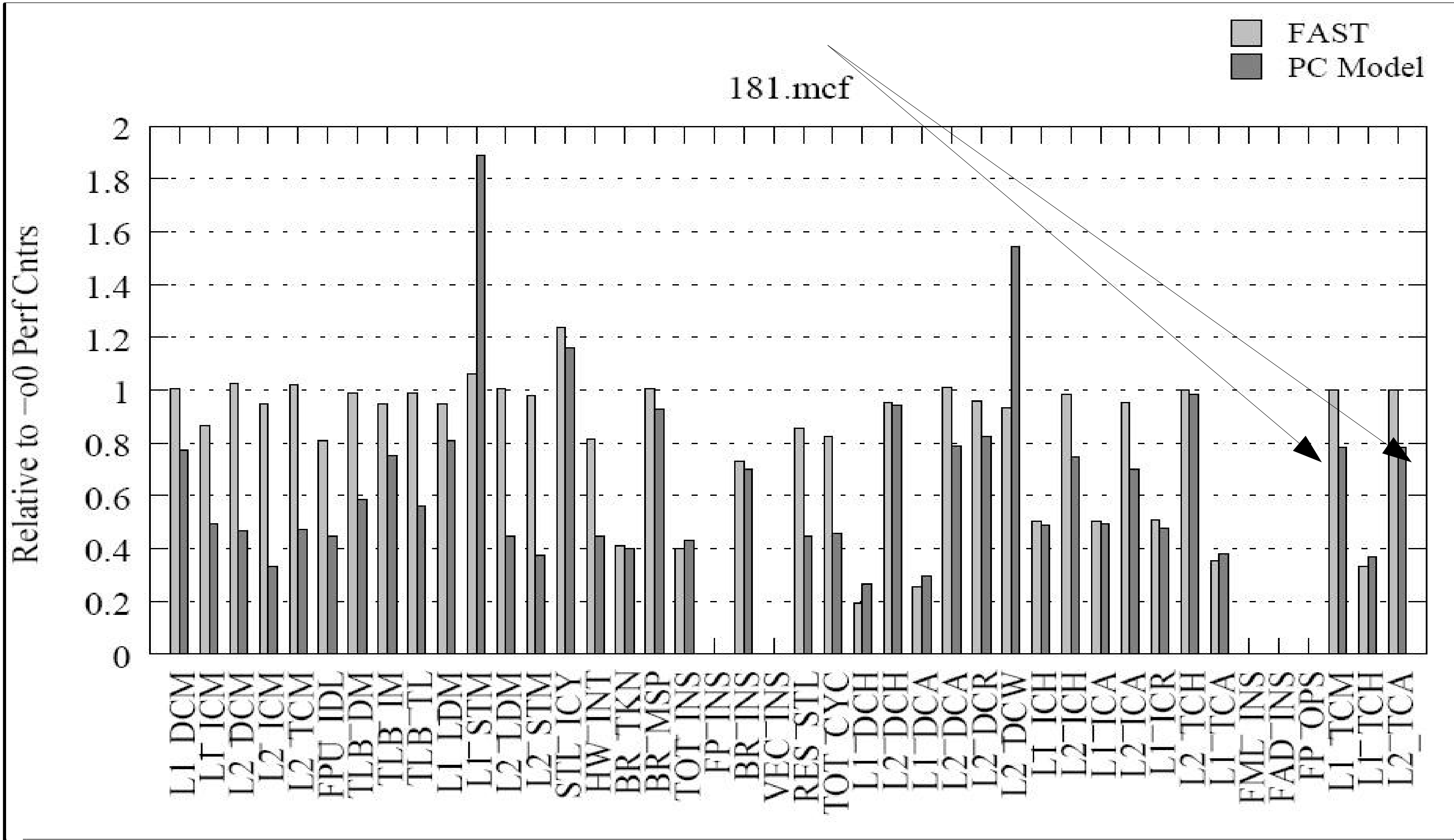
Reduce total/branch instructions and L1 I-cache/D-cache accesses.



Characterization of 181.mcf

Model applies -m32 (32 bit pointers)

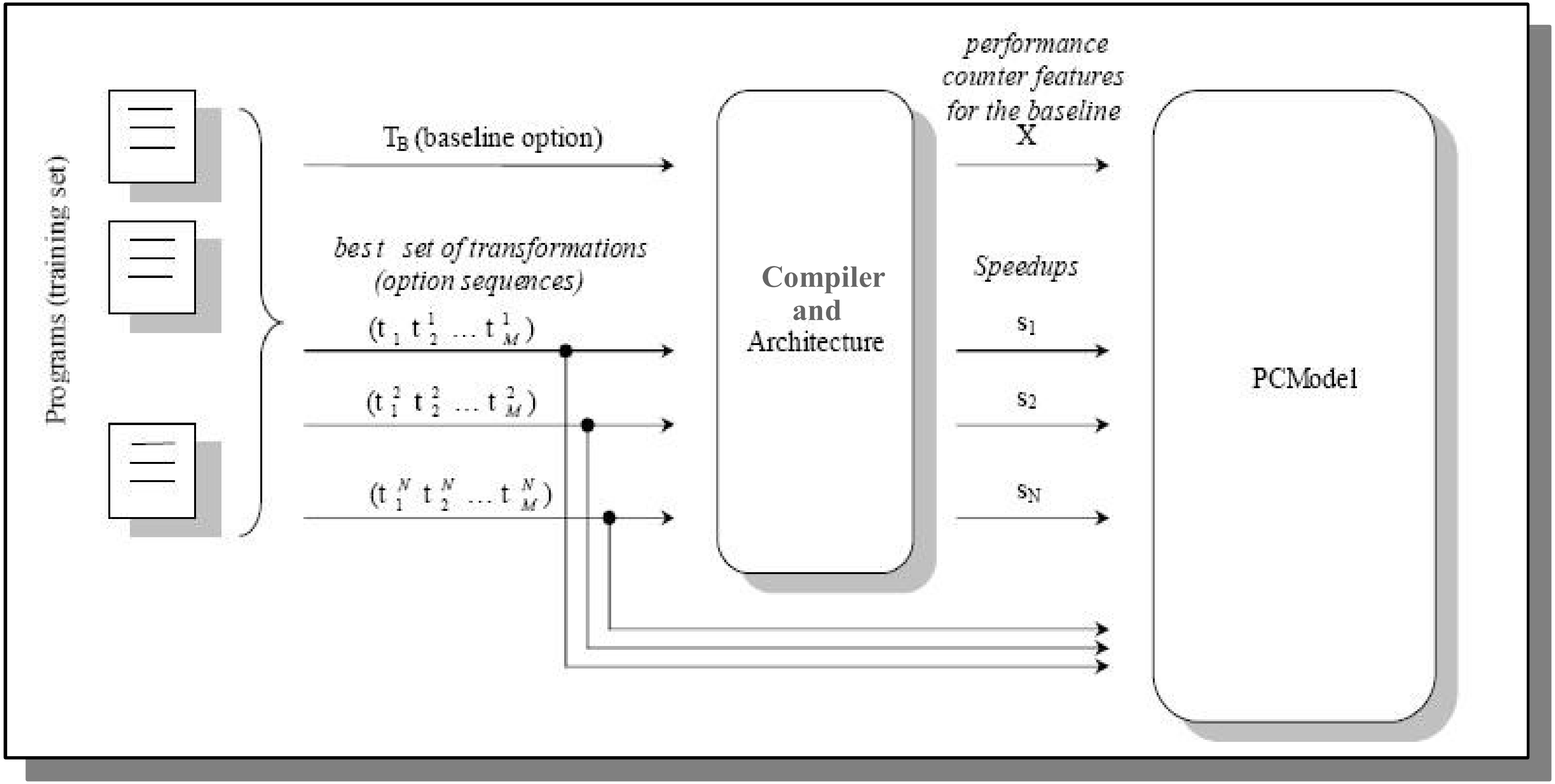
Reduces L1 cache misses which reduces L2 cache accesses.



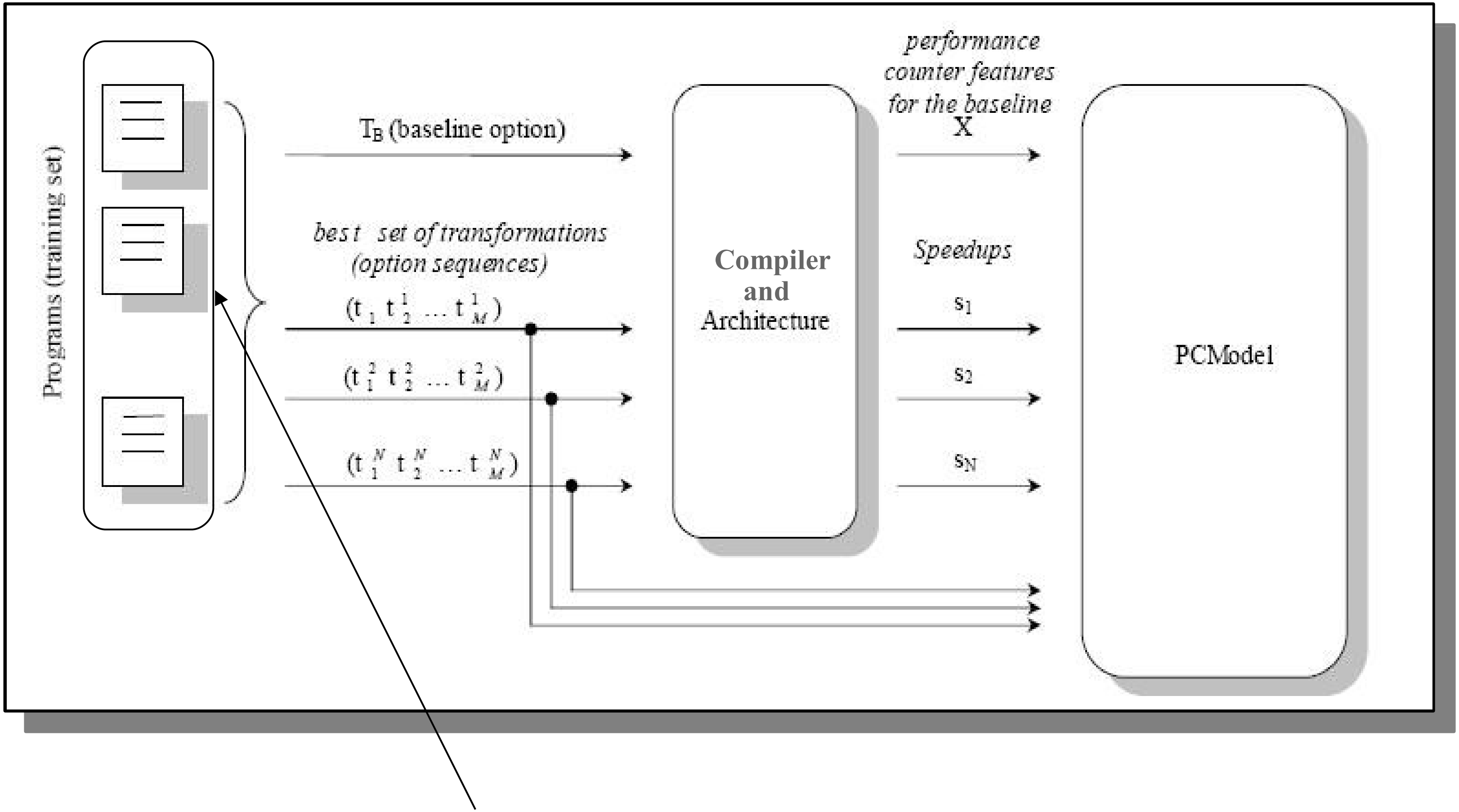
Putting Perf Counters to Use

- Important aspects of programs captured with performance counters
- Automatically construct model (PC Model)
  - Map performance counters to good opts
- Model predicts optimizations to apply
  - Uses performance counter characterization

Training PC Model

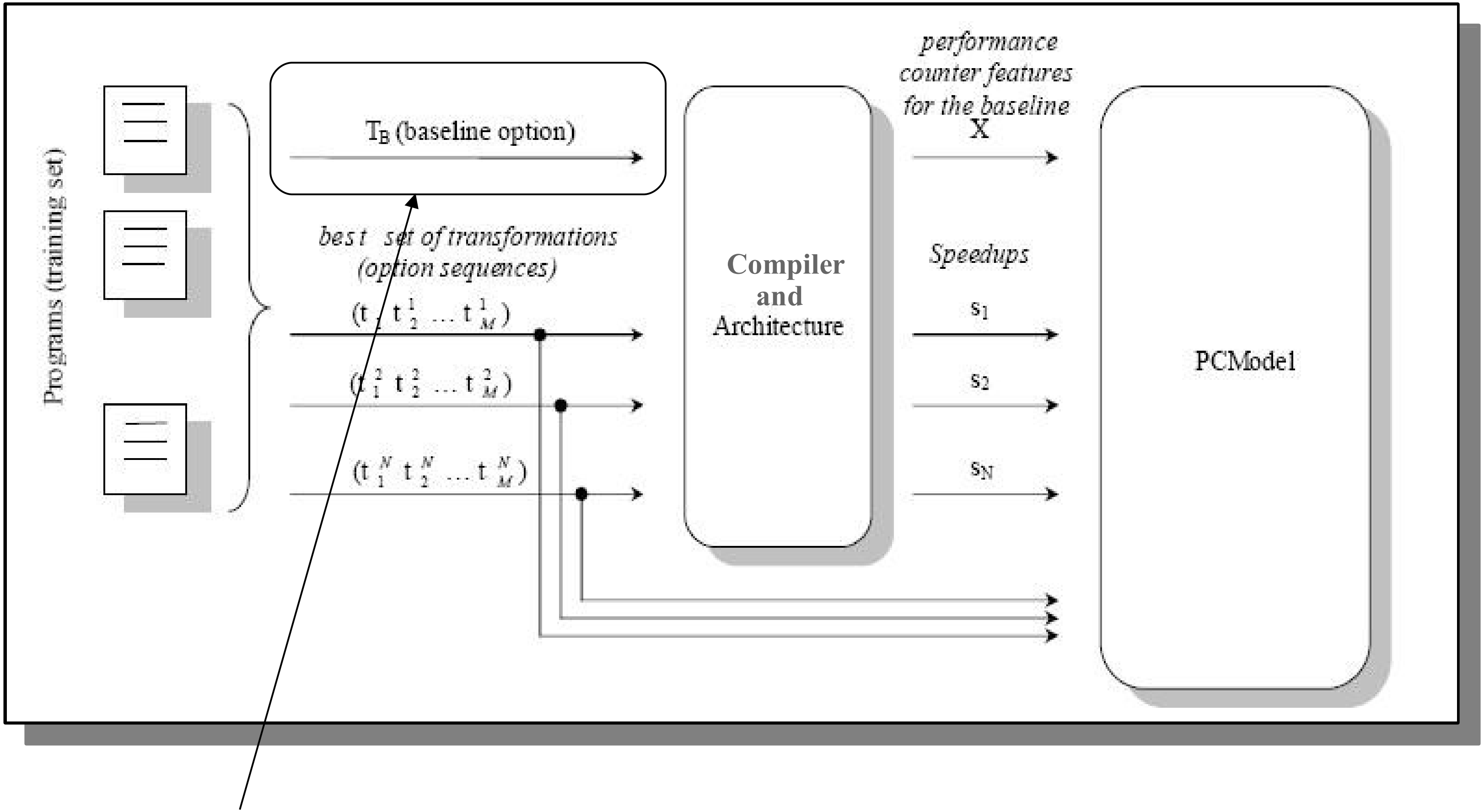


Training PC Model



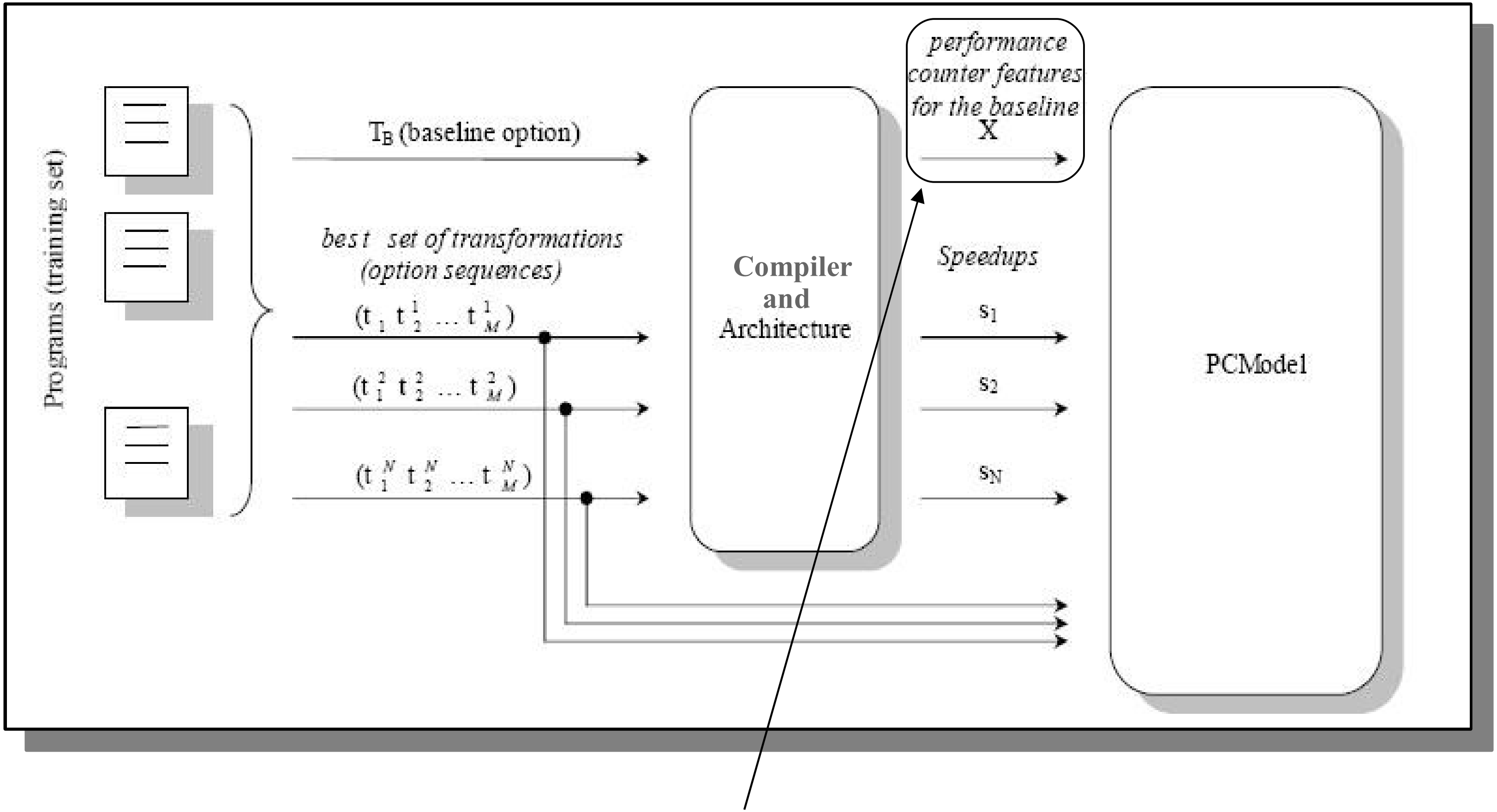
Programs to train model (different from test program).

Training PC Model



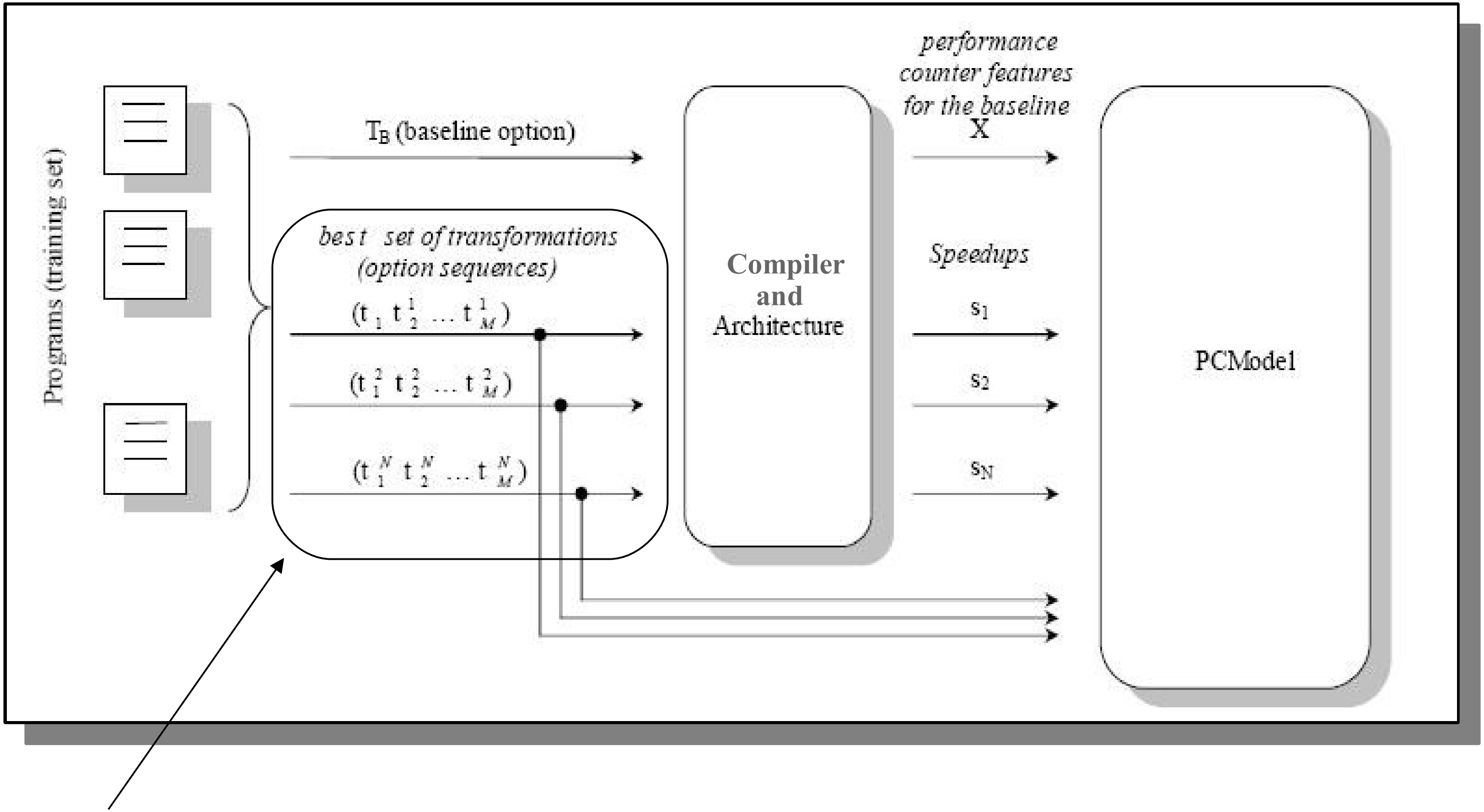
Baseline runs to capture performance counter values.

Training PC Model



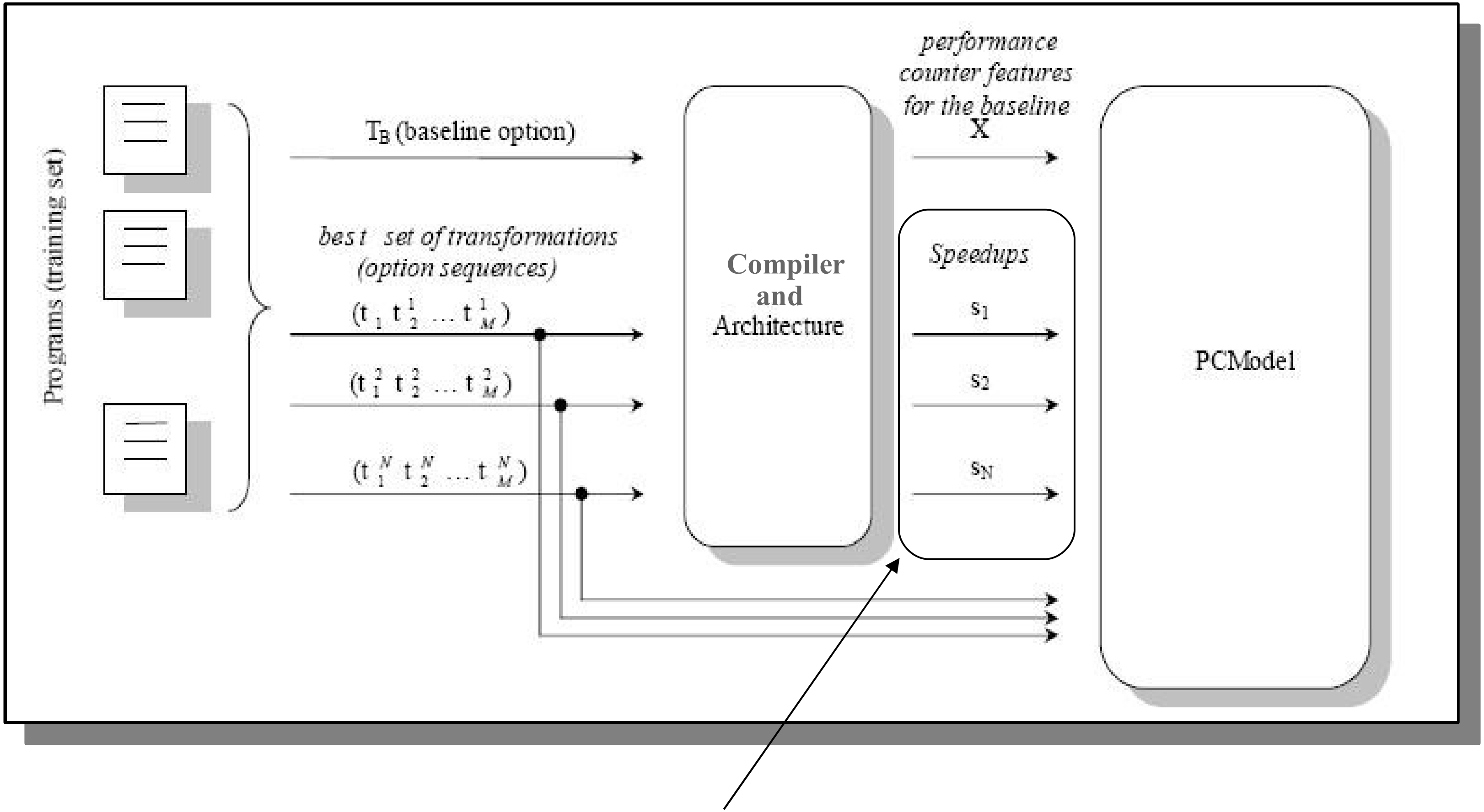
Obtain performance counter values for a benchmark.

Training PC Model



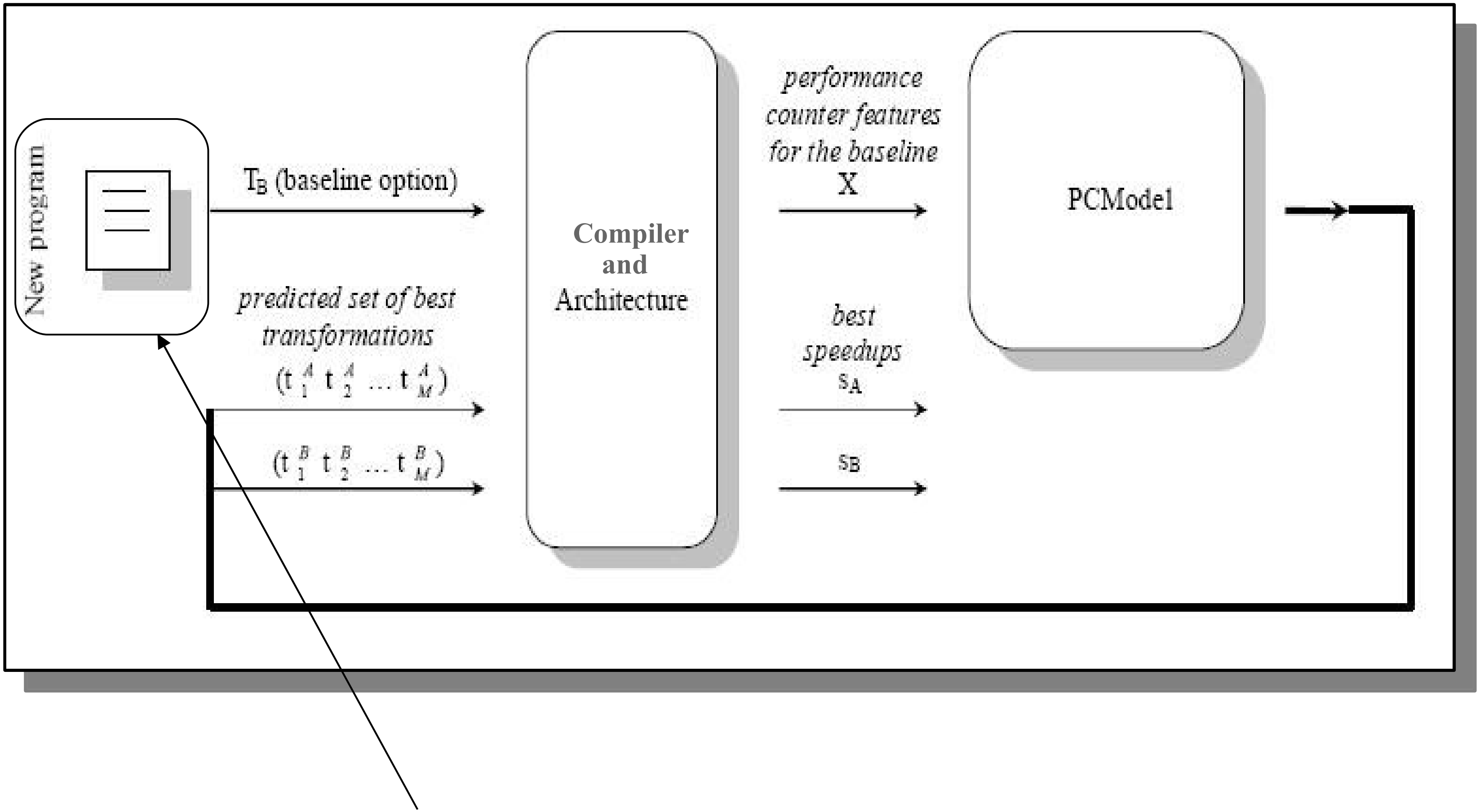
Best optimizations runs to get speedup values.

Training PC Model



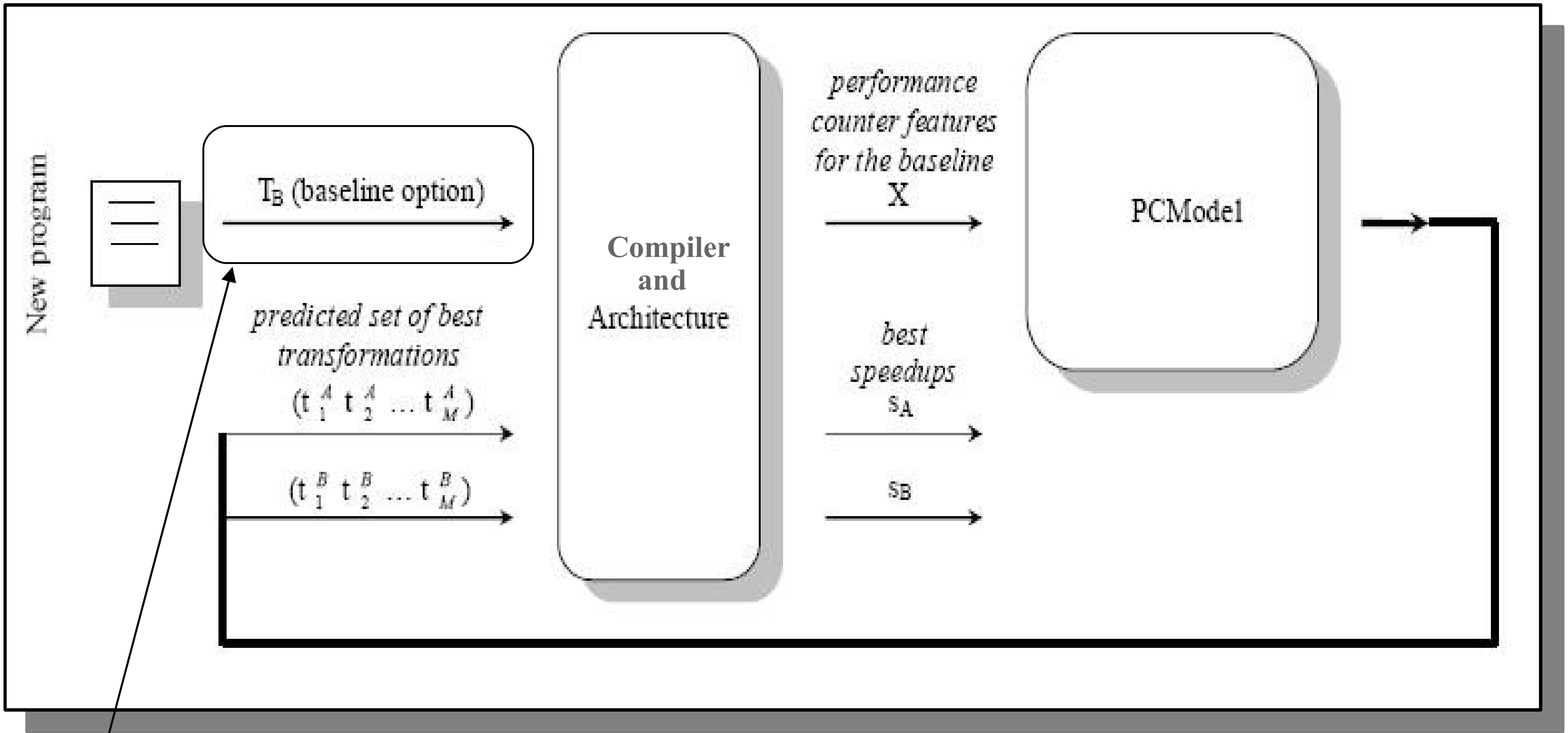
Best optimizations runs to get speedup values.

Using PC Model



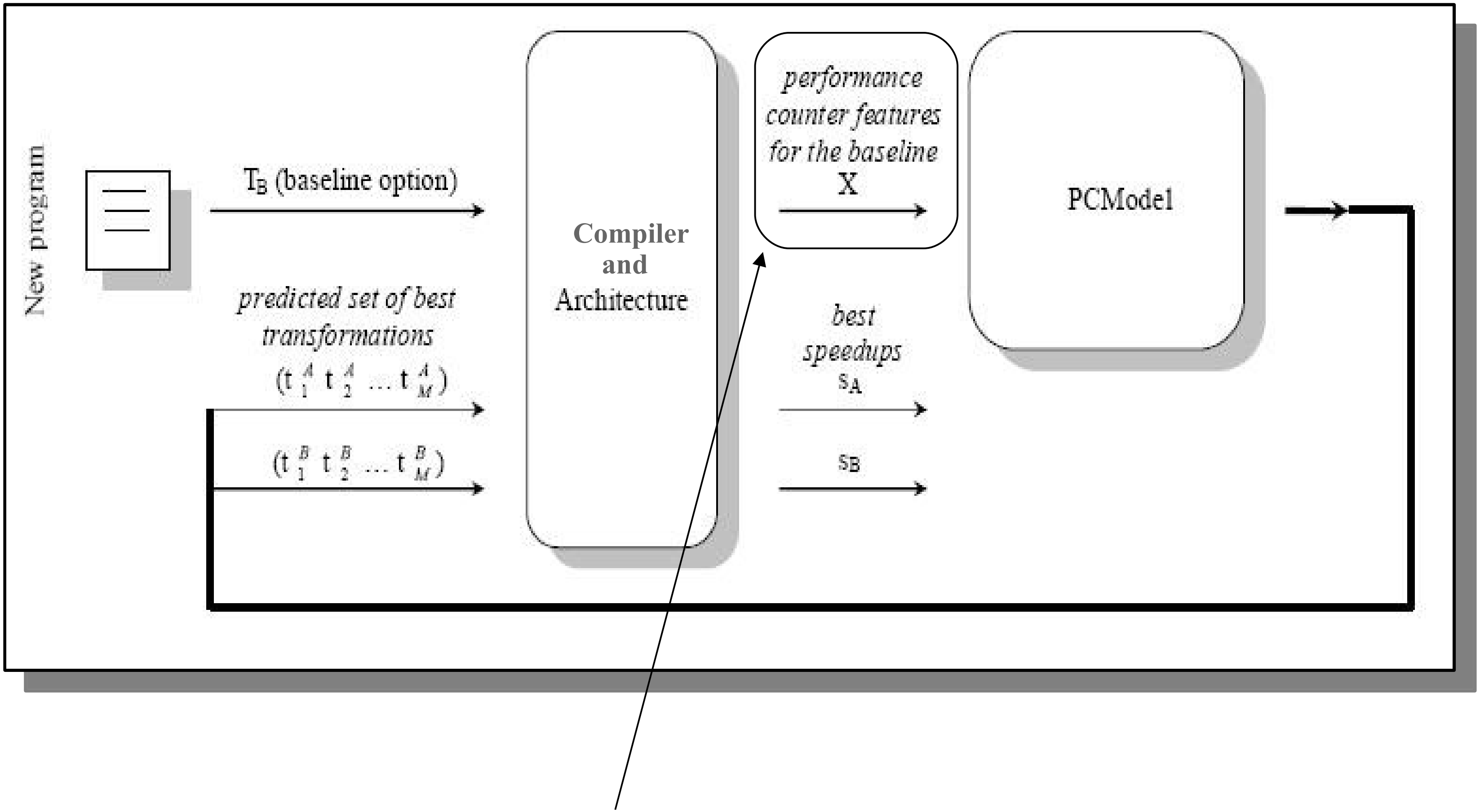
New program interested in obtaining good performance.

Using PC Model



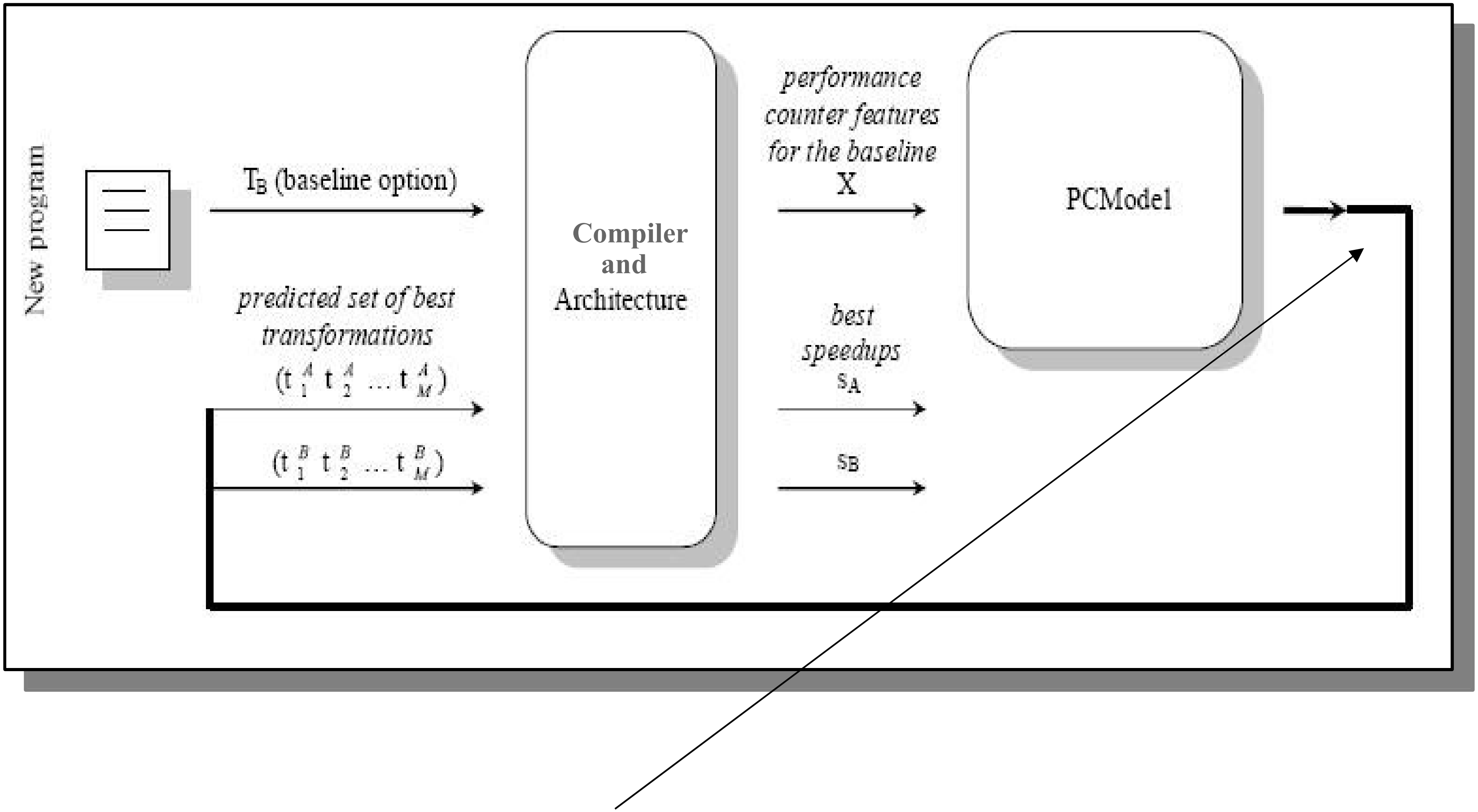
Baseline run to capture performance counter values.

Using PC Model



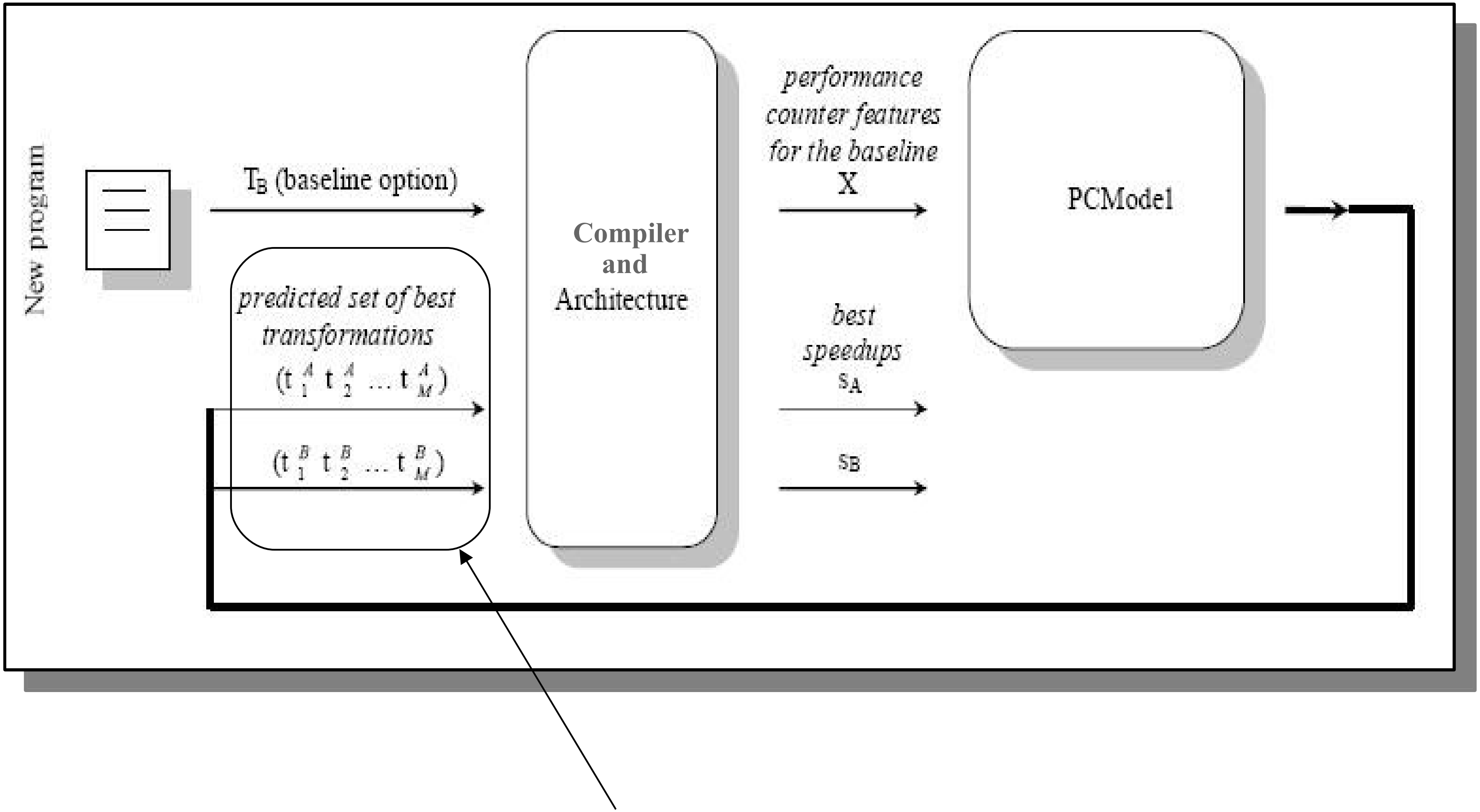
Feed performance counter values to model.

Using PC Model



Model outputs a distribution which we generate sequences from.

Using PC Model



Optimization sequences drawn from distribution.

PC Model

- Trained on data from Random Search
  - 500 evaluations for each benchmark
- Leave-one-out cross validation
  - Training on N-1 benchmarks
  - Test on Nth benchmark
- Logistic Regression

Logistic Regression

- Variation of ordinary regression
- Inputs
  - Continuous, discrete, or a mix
  - 60 performance counters
    - All normalized to cycles executed
- Ouputs
  - Restricted to two values **(0,1)**
  - Probability an optimization is beneficial

Experimental Methodology

- PathScale compiler
  - Compare to highest optimization level
  - 121 compiler flags
- AMD Athlon processor
  - ***Real*** machine; Not simulation
- 57 benchmarks
  - SPEC (INT 95, INT/FP 2000), MiBench, Polyhedral

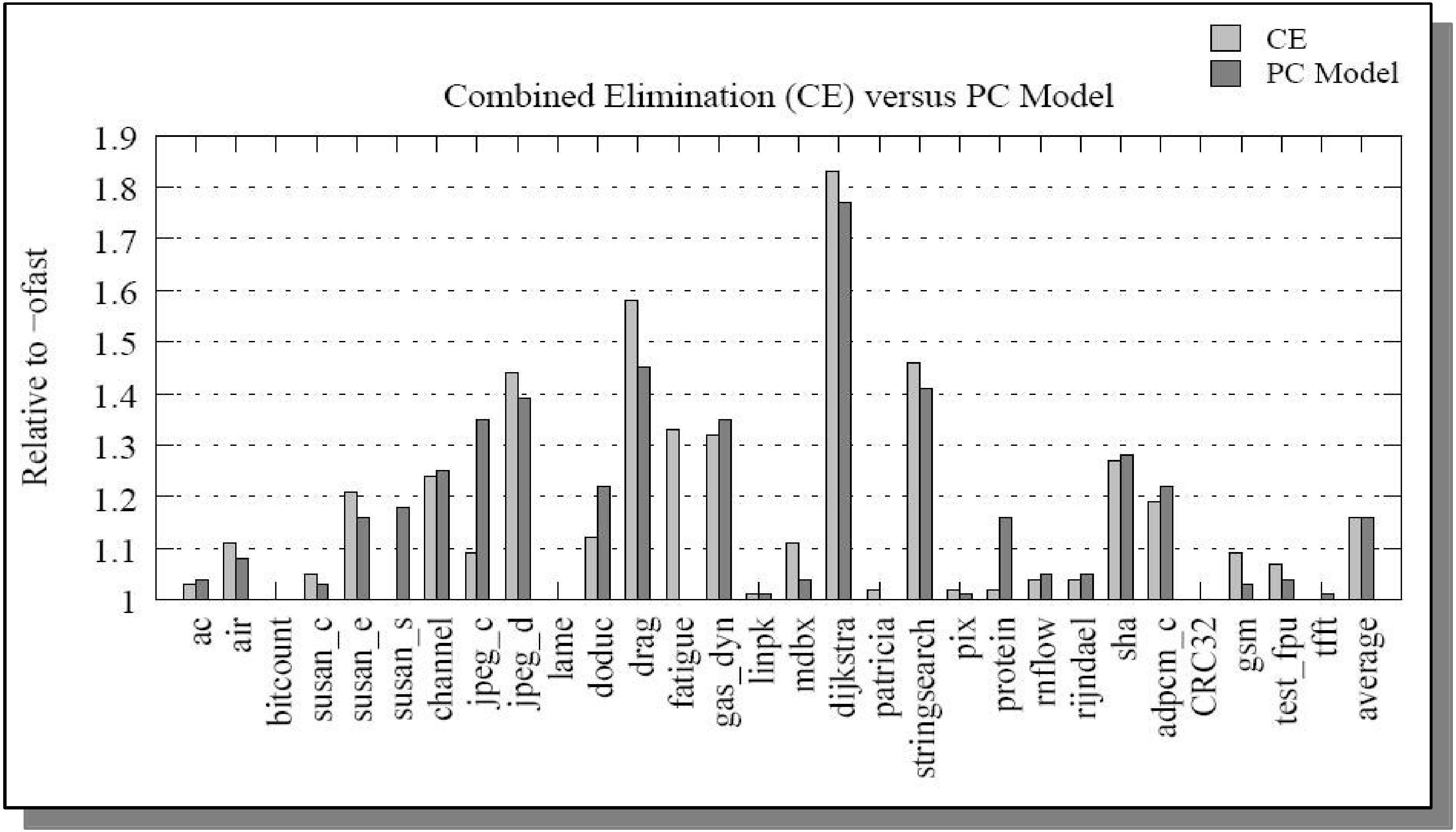
Results

- Combined Elimination and PC Model
- Performance versus Evaluations
- Most Informative Performance Counters

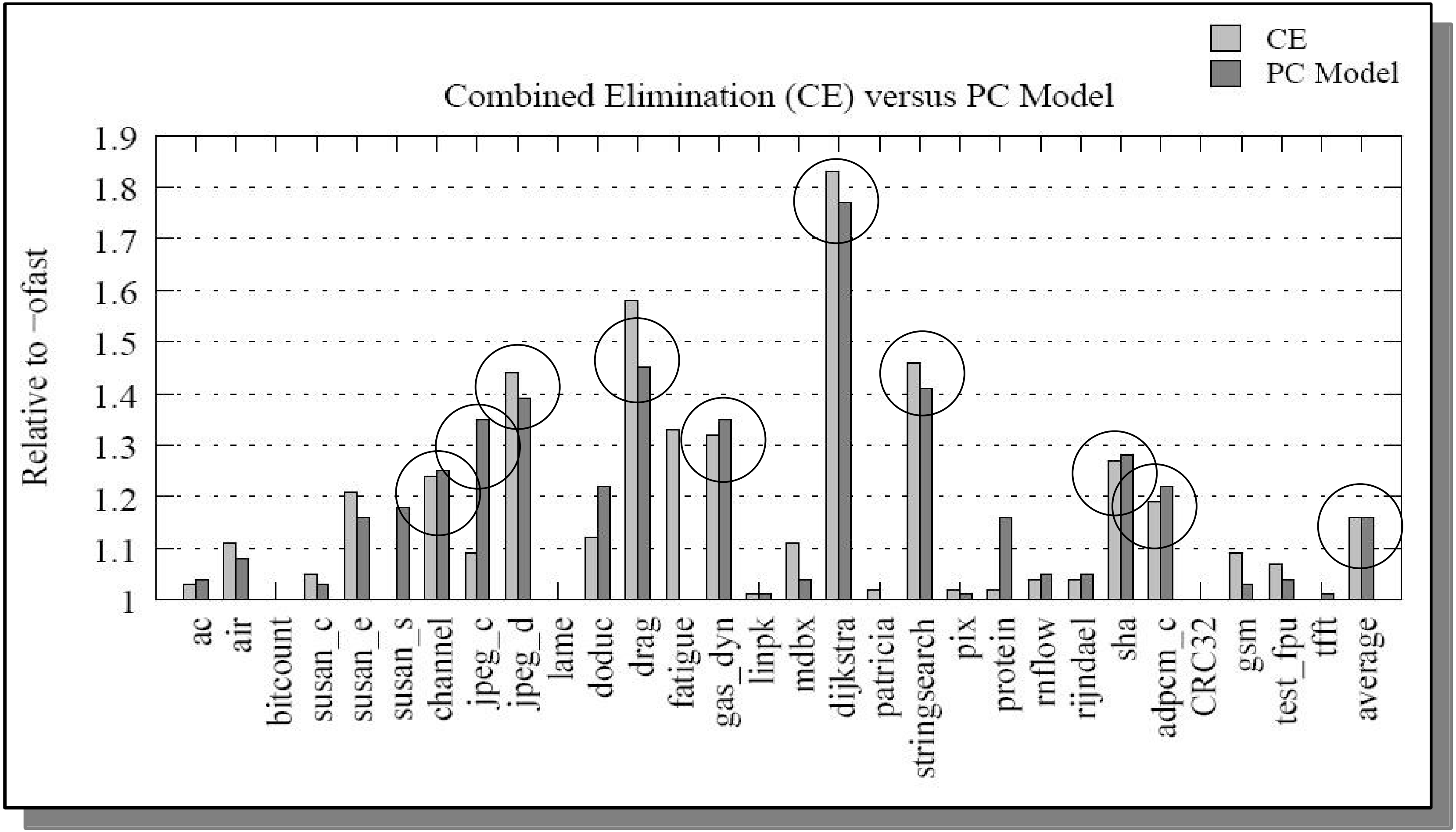
Evaluate Search Strategies

- PC Model
- RAND
  - Randomly select 500 optimization seqs
- Combined Elimination [CGO 2006]
  - Pure search technique
    - Evaluate optimizations one at a time
    - Eliminate negative optimizations in one go
  - Out-performed other pure search techniques

PC Model/CE *(MiBench/Polyhedral)*

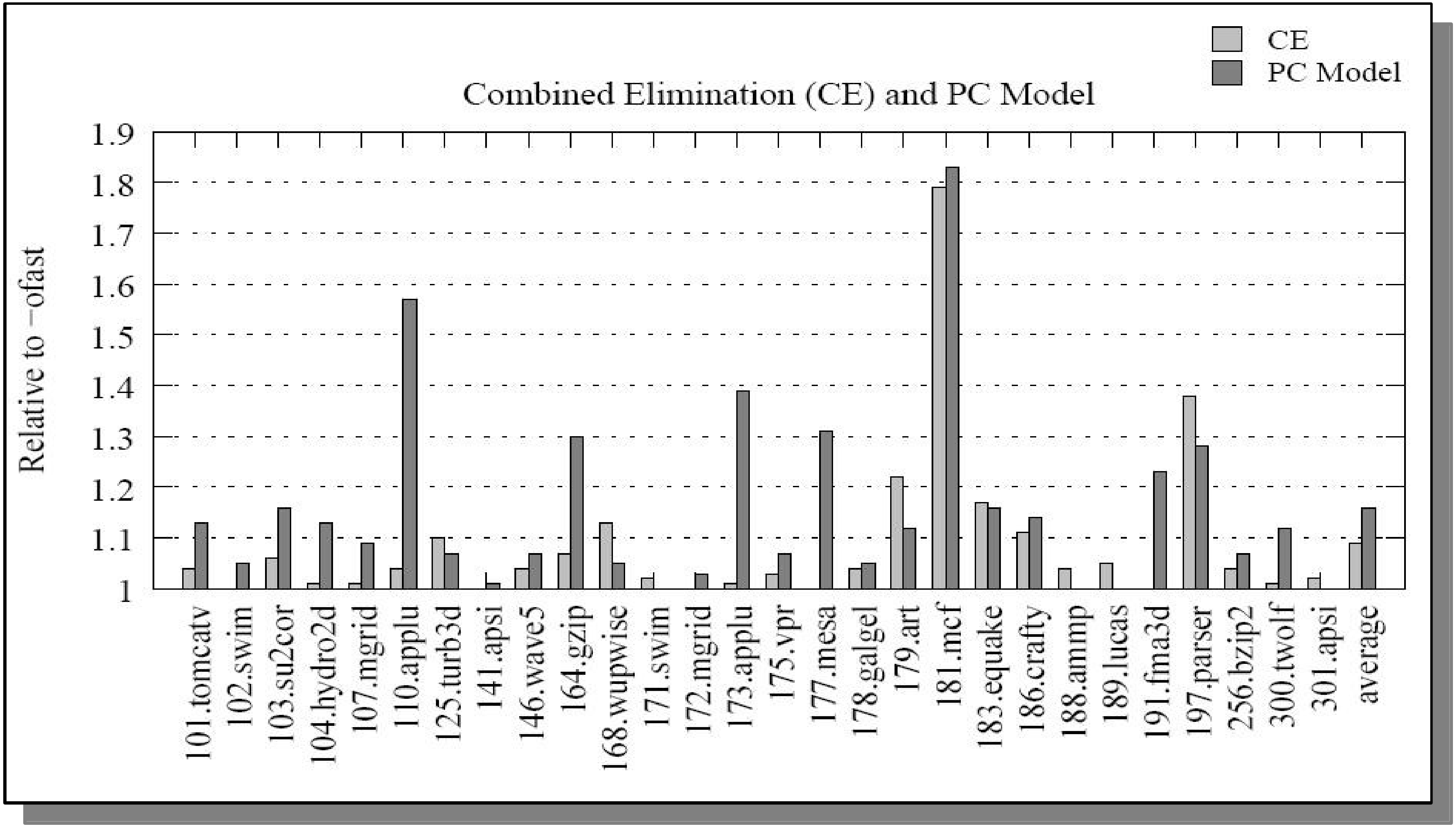


PC Model/CE (MiBench/Polyhedral)

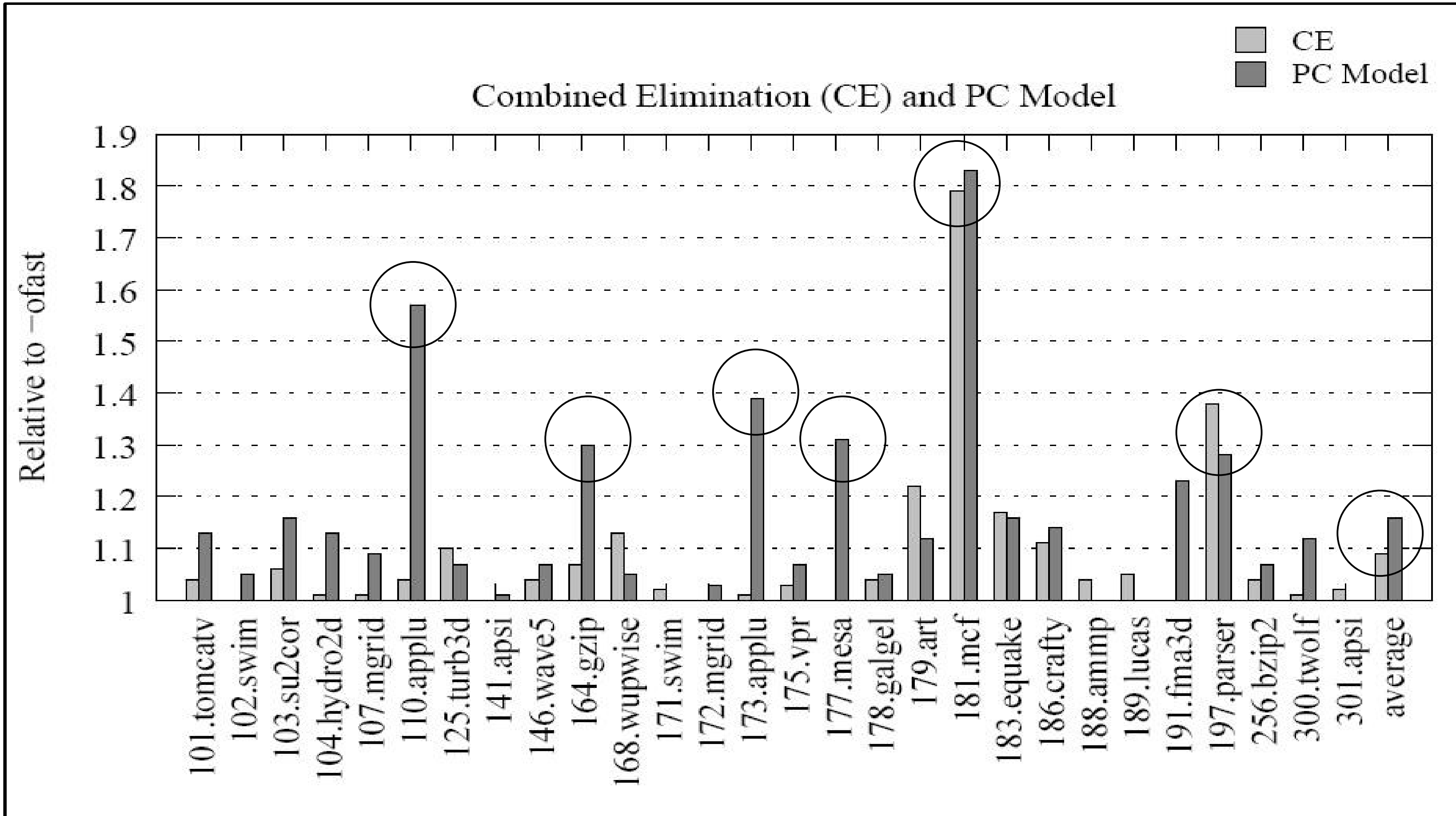


1. 9 benchmarks over 20% improvement and 17% on average!
2. CE uses 607 iterations (240-1550) and PC Model 25 iterations.

PC Model/CE (SPEC INT 95/SPEC 2000)

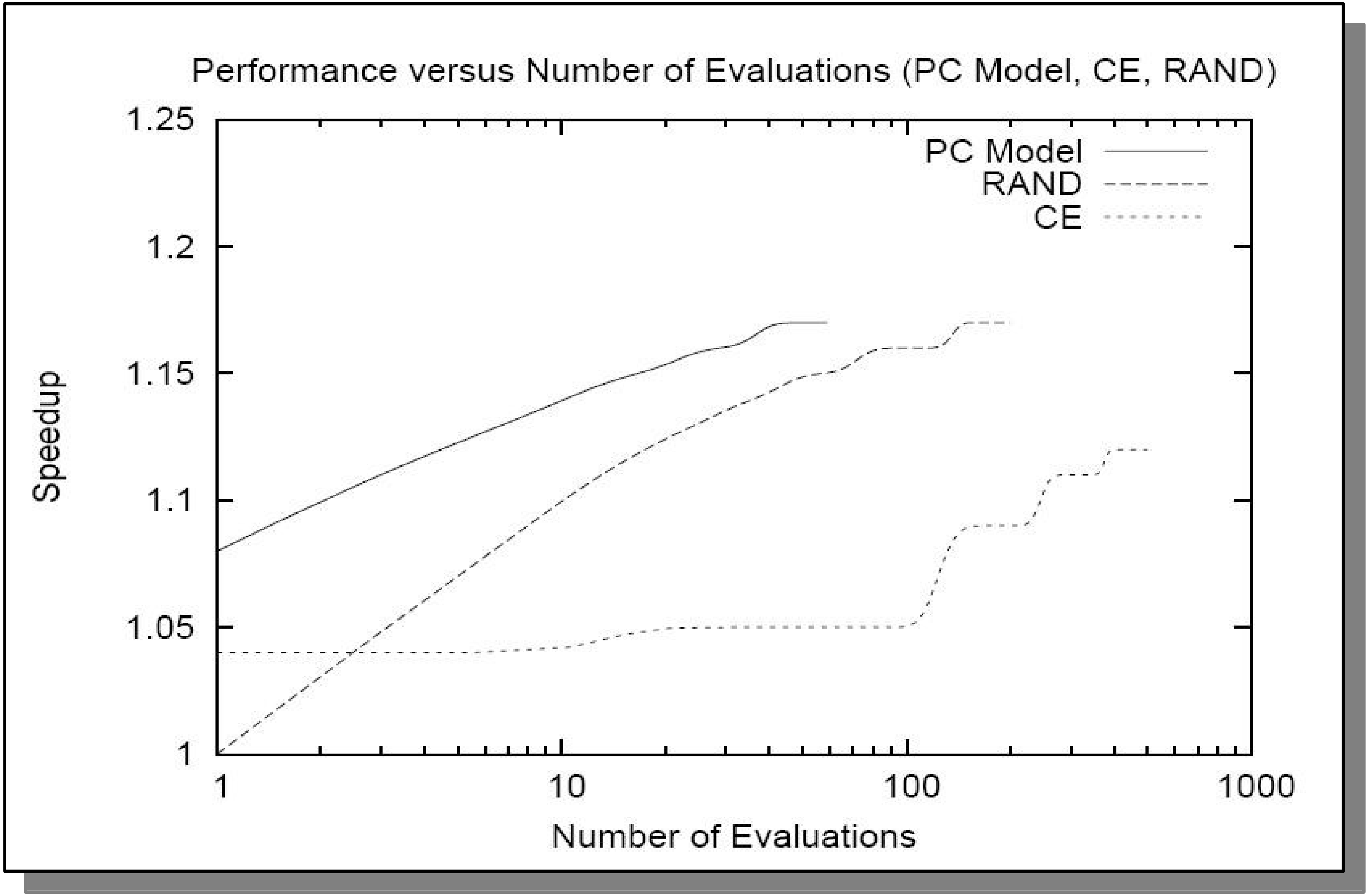


PC Model/CE *(SPEC INT 95/SPEC 2000)*

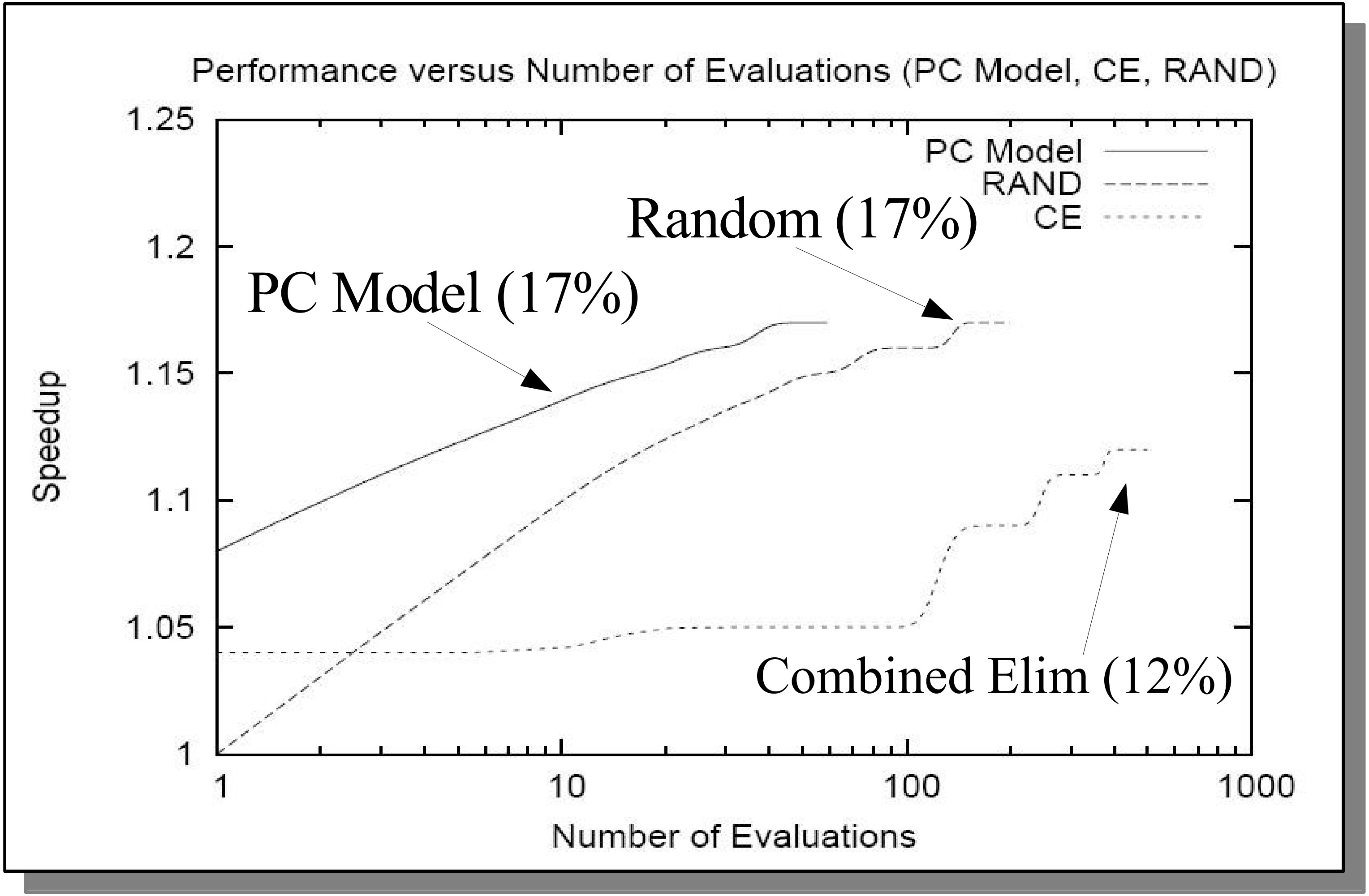


1. Obtain over 25% improvement on 7 benchmarks!
2. On average, CE obtains 9% and PC Model 17% over -ofast!

Performance vs Evaluations



Performance vs Evaluations



Why is CE worse than RAND?

- Combined Elimination
  - Dependent on dimensions of space
  - Easily stuck in local minima
- RAND
  - Probabilistic technique
  - Depends on distribution of good points
  - Not susceptible to local minima

Note: CE would perform better where many opts degrade performance.

Most Informative Features

Most Informative Performance Counters	
1. L1 Cache Accesses	9. L2 Dcache Hits
2. L1 Dcache Hits	10. L2 Cache Accesses
3. TLB Data Misses	11. L1 Dcache Accesses
4. Branch Instructions	12. Hardware Interrupts
5. Resource Stalls	13. L2 Cache Hits
6. Total Cycles	14. L1 Cache Hits
7. L2 Icache Hits	15. Branch Misses
8. Vector Instructions	

Conclusions

- Use performance counters to find good optimization settings
- Out-performs production compiler in few evaluations (+ 3 for counters)
- 2 orders of magnitude faster than best known pure search technique