Aestimo: A Feedback-Directed Optimization Evaluation Tool

A Compiler Perspective on Input Characterization

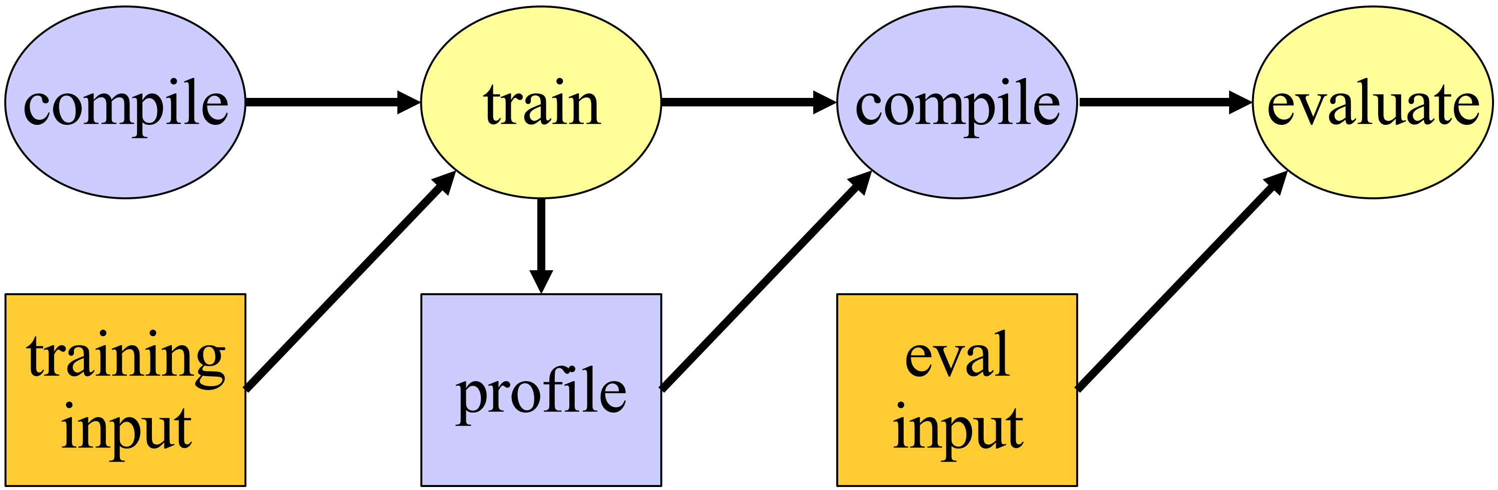
Paul Berube
Compiler Design and Optimization Laboratory
University of Alberta

Outline

- Background
- Brief Overview of *Aestimo*
- Difference Metric
- Alignment Metric

What Is FDO?

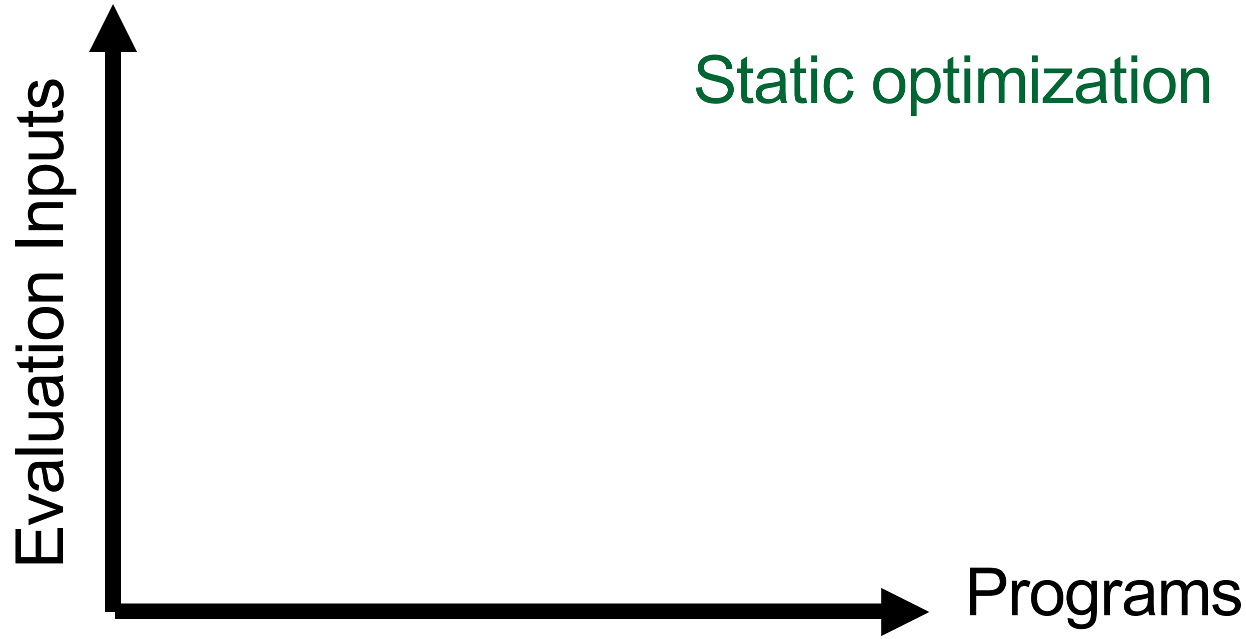
Feedback-Directed Optimization:



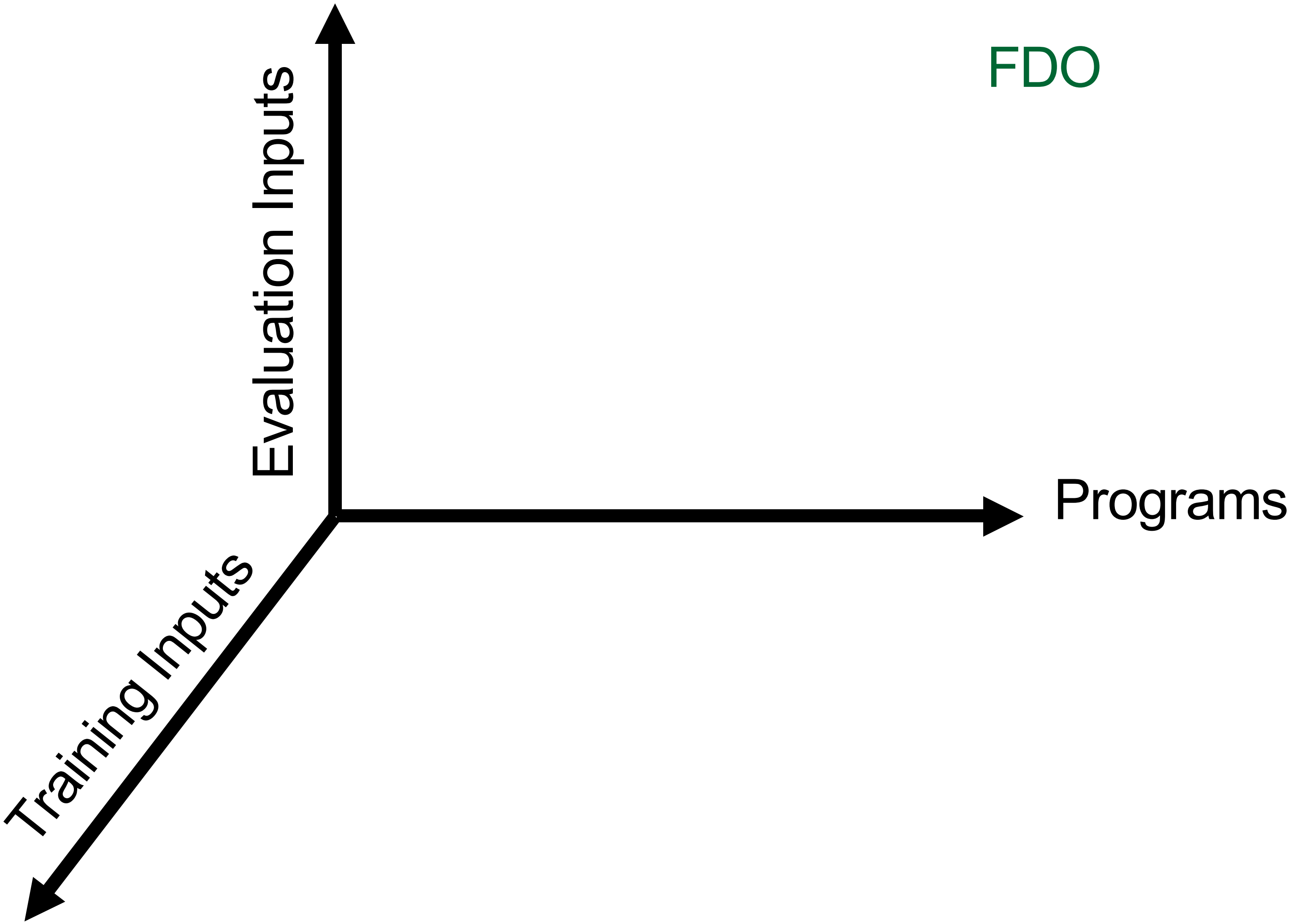
What Is A Profile?

- Frequency counts for program elements that determine program behavior:
  - Block/Edge/Path profiles
  - Branch probabilities
  - Loop iteration counts
  - Function call counts
  - Function invocation counts
  - Value profiles
  - ...and more every year...

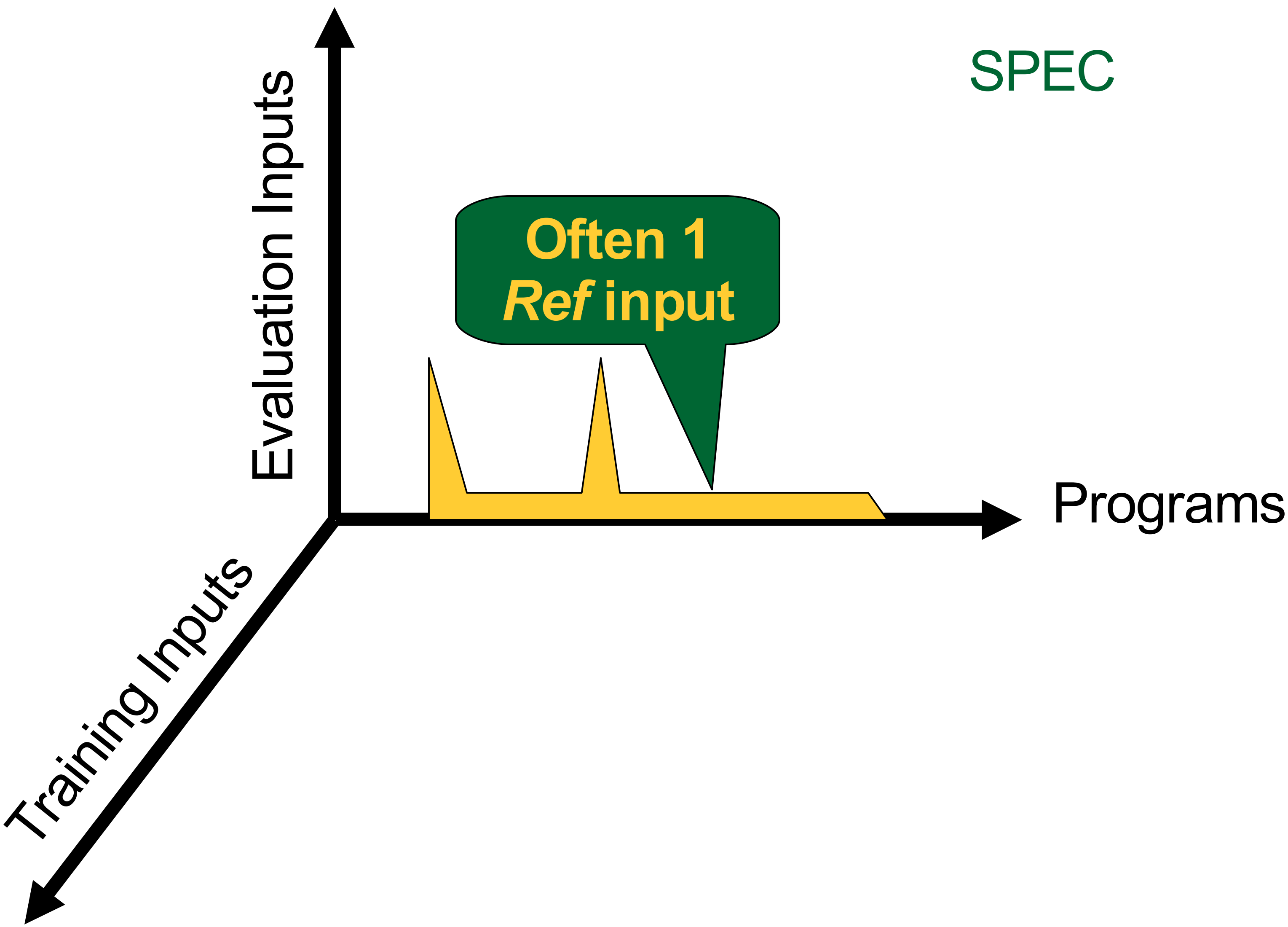
Performance Evaluation Space



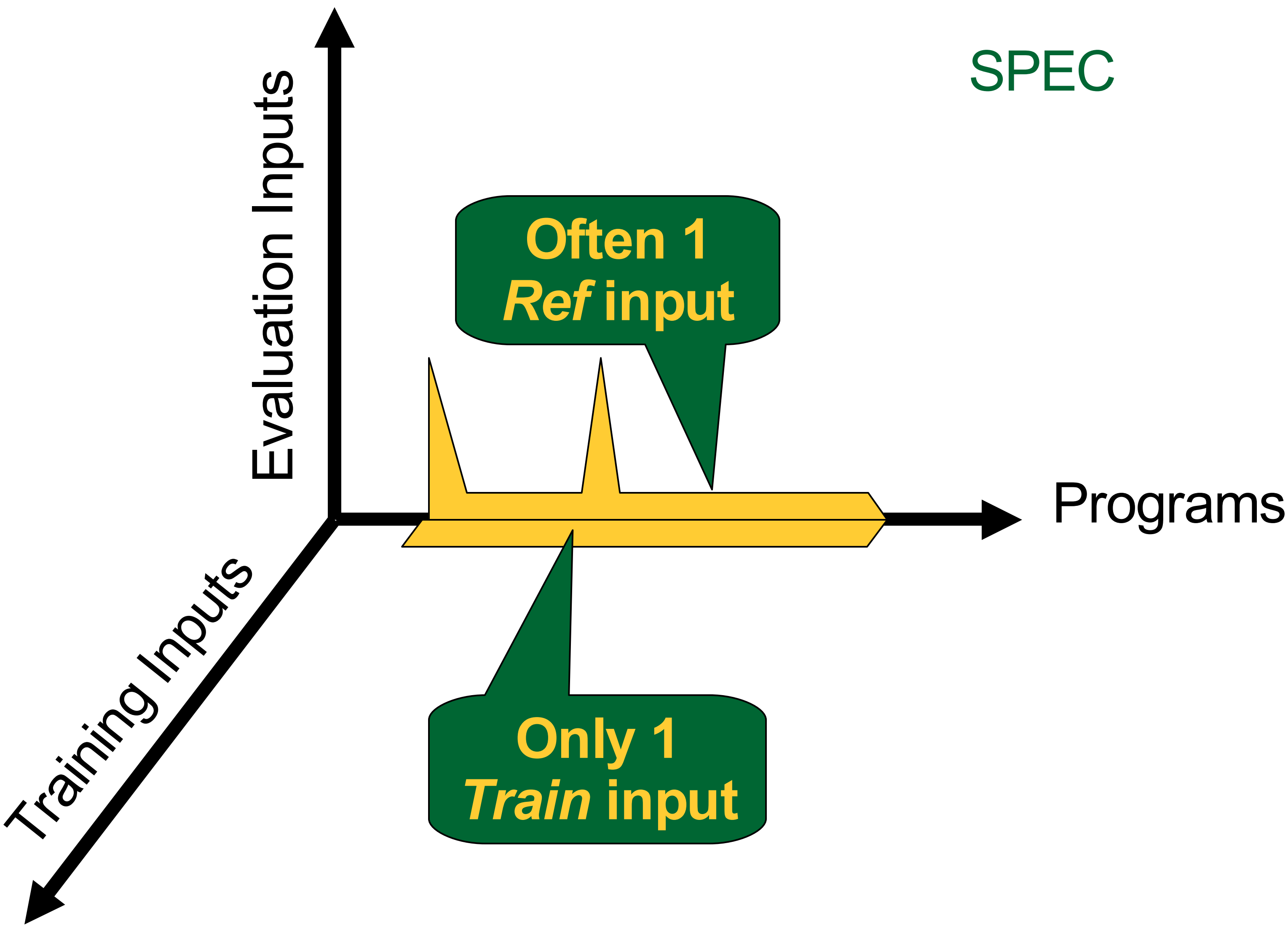
Performance Evaluation Space



Performance Evaluation Space



Performance Evaluation Space



Input Characterization

- Desired to help deal with the {training input X evaluation input} space
- Determine algorithmically and quantitatively the similarity between inputs
- If several inputs are similar, pick a representative
  - Otherwise, we need to consider all of them
- Previous work from a architecture/design-space exploration perspective

Aestimo

- An FDO evaluation tool
- Automates training and evaluating on a large number of inputs
- Isolates individual transformations
  - Fewer experiment variables
  - Results vary by transformation
- Measures:
  - Differences in transformation decisions
  - Performance differences

Aestimo

- An FDO evaluation tool
- Automates training and evaluating on a large number of inputs
- Isolates individual transformations
  - Fewer experiment variables
  - Results vary by transformation
- Measures:
  - **Differences in transformation decisions**
  - Performance differences

Inputs for SPEC CINT2000

- Several additional inputs for SPEC CPU2000 integer benchmarks at:
  - http://www.cs.ualberta.ca/~berube/compiler/fdo/inputs.shtml
- Goal: > 20 real program inputs per benchmark that span the space of typical usage

Input Characterization: Naive

- Are two inputs different?
- “diff” them!
- But this tells us nothing!
  - Inputs might still cause the code to behave identically

Inputs: Human Expert

- "Does the same kind of thing", “They're quite different”
- Can intuition/experience be made explicit, quantitative?
- Nobody can be an expert on every program

Inputs: Computer Architect

- Do the inputs stress the architecture in the same way?
  - IPL, branch predictability, cache behavior, etc...
- Metrics not unique
  - Same ILP for two different functions
  - Similar cache behavior for two different pieces of code that do similar data structure traversals

Inputs: Compiler Designer

- Do the inputs exercises the same code in the same way, have similar memory behavior?
- *I.e.*, how similar are the profiles?
  - Different behavior does not necessitate different transformation decisions
    - Scaled frequencies
    - Same "hot" regions
- How different is different enough?
  - Depends on the compiler heuristics!

Inputs: Compiler Heuristic

- Does the expected code behavior results in the same transformation decisions?
- *I.e.*, same results of cost-benefit analysis
- If all decisions are the same, then we get the same binary even if:
  - Different inputs
  - Different ILP, cache behavior
  - Different frequencies in profiles

The Difference Metric

- Let's *quantitatively* compare two inputs based on transformation decisions!
- Output a log of decisions during FDO compilation
  - Multiple inputs used to create multiple logs
- Treat a log of decisions as a vector
  - Yes/No decisions produce binary vectors
  - Quantitative decisions produce integer vectors

Converting Logs to Vectors

```
void foo () {}

void bar() {
  foo();
}

int main(int argc, char* argv[]) {
  foo();
  bar();
}
```

Converting Logs to Vectors

```
void foo () {}

void bar() {
  foo();
}
```

callsite	log 1	log 2	log 3	log 4
bar.foo	inline	call	inline	call
main.foo	call	call	call	inline
main.bar	call	inline	inline	inline
main.bar.foo		inline	inline	inline

```
int main(int argc, char* argv[]) {
  foo();
  bar();
}
```

Converting Logs to Vectors

```
void foo () {}

void bar() {
  foo();
}

int main(int argc, char* argv[]) {
  foo();
  bar();
}
```

callsite	log 1	log 2	log 3	log 4
bar.foo	inline	call	inline	call
main.foo	call	call	call	inline
main.bar	call	inline	inline	inline
main.bar.foo		inline	inline	inline

callsite	v_1	v_2	v_3	v_4
bar.foo				
main.foo				
main.bar				
main.bar.foo				

Converting Logs to Vectors

```
void foo () {}

void bar() {
  foo();
}

int main(int argc, char* argv[]) {
  foo();
  bar();
}
```

callsite	log 1	log 2	log 3	log 4
bar.foo	inline	call	inline	call
main.foo	call	call	call	inline
main.bar	call	inline	inline	inline
main.bar.foo		inline	inline	inline

callsite	v_1	v_2	v_3	v_4
bar.foo	1		1	
main.foo				1
main.bar		1	1	1
main.bar.foo		1	1	1

Converting Logs to Vectors

```
void foo () {}

void bar() {
   foo();
}

int main(int argc, char* argv[]) {
   foo();
   bar();
}
```

callsite	log 1	log 2	log 3	log 4
bar.foo	inline	call	inline	call
main.foo	call	call	call	inline
main.bar	call	inline	inline	inline
main.bar.foo		inline	inline	inline

callsite	v_1	v_2	v_3	v_4
bar.foo	1	0	1	0
main.foo	0	0	0	1
main.bar	0	1	1	1
main.bar.foo		1	1	1

Converting Logs to Vectors

```
void foo () {}

void bar() {
  foo();
}

int main(int argc, char* argv[]) {
  foo();
  bar();
}
```

callsite	log 1	log 2	log 3	log 4
bar.foo	inline	call	inline	call
main.foo	call	call	call	inline
main.bar	call	inline	inline	inline
main.bar.foo		inline	inline	inline

callsite	v_1	v_2	v_3	v_4
bar.foo	1	0	1	0
main.foo	0	0	0	1
main.bar	0	1	1	1
main.bar.foo	0	1	1	1

Difference: Definition

- $\boldsymbol{d}(\boldsymbol{v}_a,\boldsymbol{v}_b) = |\boldsymbol{v}_a - \boldsymbol{v}_b|^2$
  - L^2-*norm* of vector difference, squared
  - Hamming Distance if vectors are binary
    - Count of bit-wise differences
    - Linear and intuitive

Difference: What it tells us

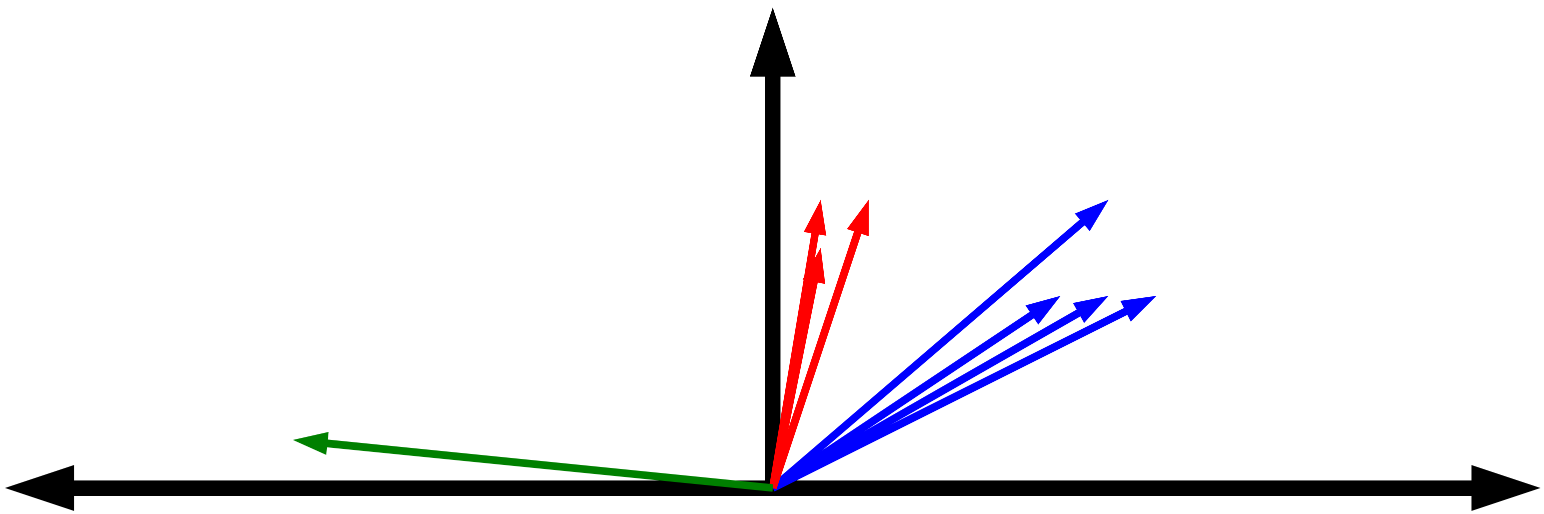
- Does the compiler transform the program the same way?
- Not:
  - Importance of any decision for performance
  - Inputs are equivalent for use as a training input
  - Inputs are equivalent for use for evaluation

The Alignment Metric

- Can we say more if we have many inputs?
- Are there decisions that are made the same way for all inputs?
- Alternately, how “conformist” are the logs?
- Can we quantify “conformity” with a single number?

Alignment: Graphical Analogy

- 8 logs
- Two clusters of similar logs
- One very distinct log
- 32 pairwise difference scores!



Alignment: Definition

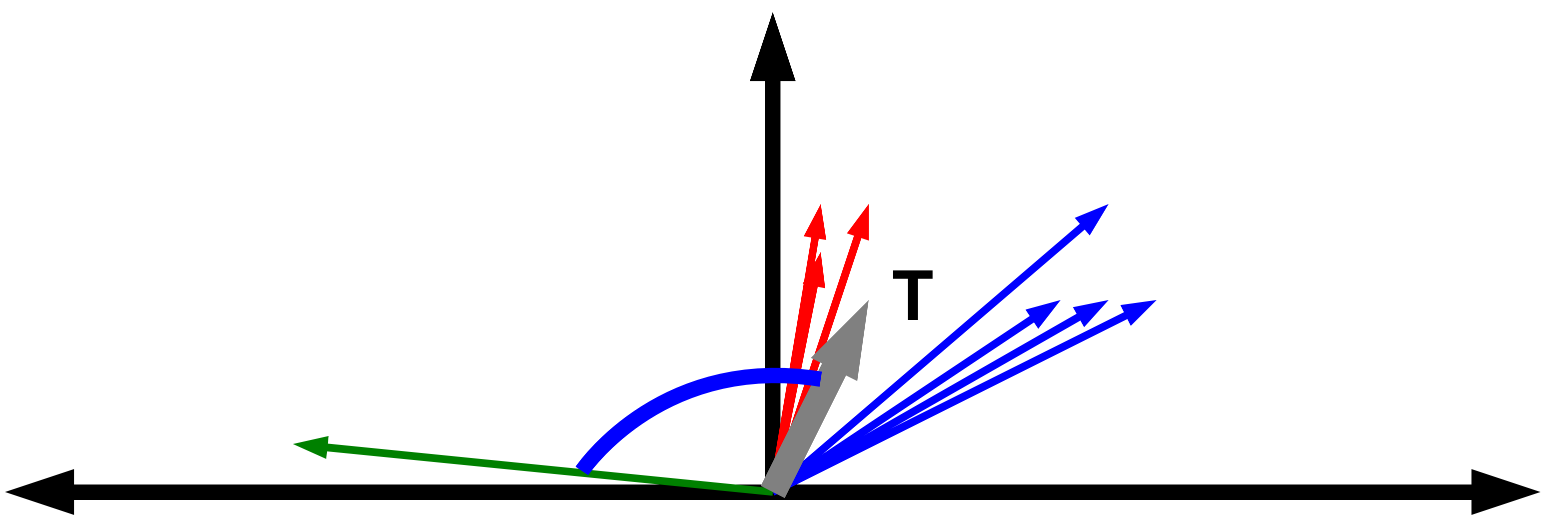
$$\boldsymbol{T} = \sum \boldsymbol{v}_i$$

$$\alpha_i = \frac{\boldsymbol{T} \bullet \boldsymbol{v}_i}{\sum_i \boldsymbol{T}[i]} = \frac{\boldsymbol{T} \bullet \boldsymbol{v}_i}{|\boldsymbol{T}|_1}$$

- ***T*** summarizes/accumulates all the logs
- With binary vectors:
  - dividing by $|\boldsymbol{T}|_1$ normalizes alignment
  - $\boldsymbol{T} \cdot \boldsymbol{v}$ filters ***T*** by choices in ***v***
- Recall: inner product = |a||b|cosθ

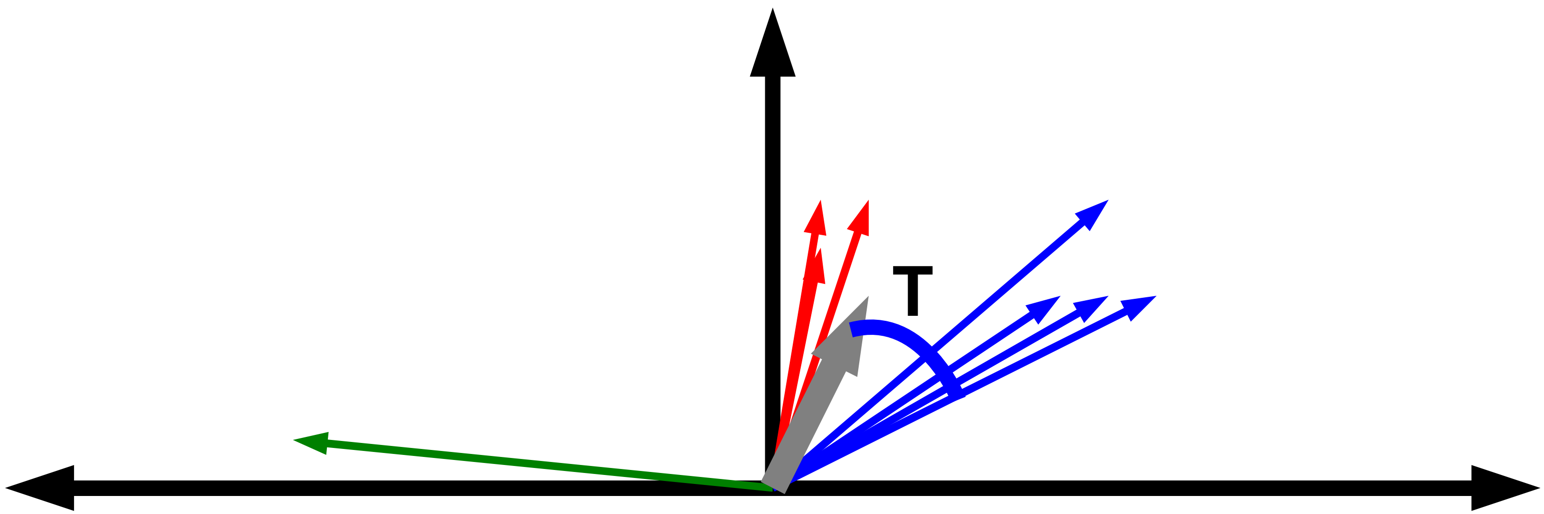
Alignment: What it tells us

- Big angle, very different from all the others



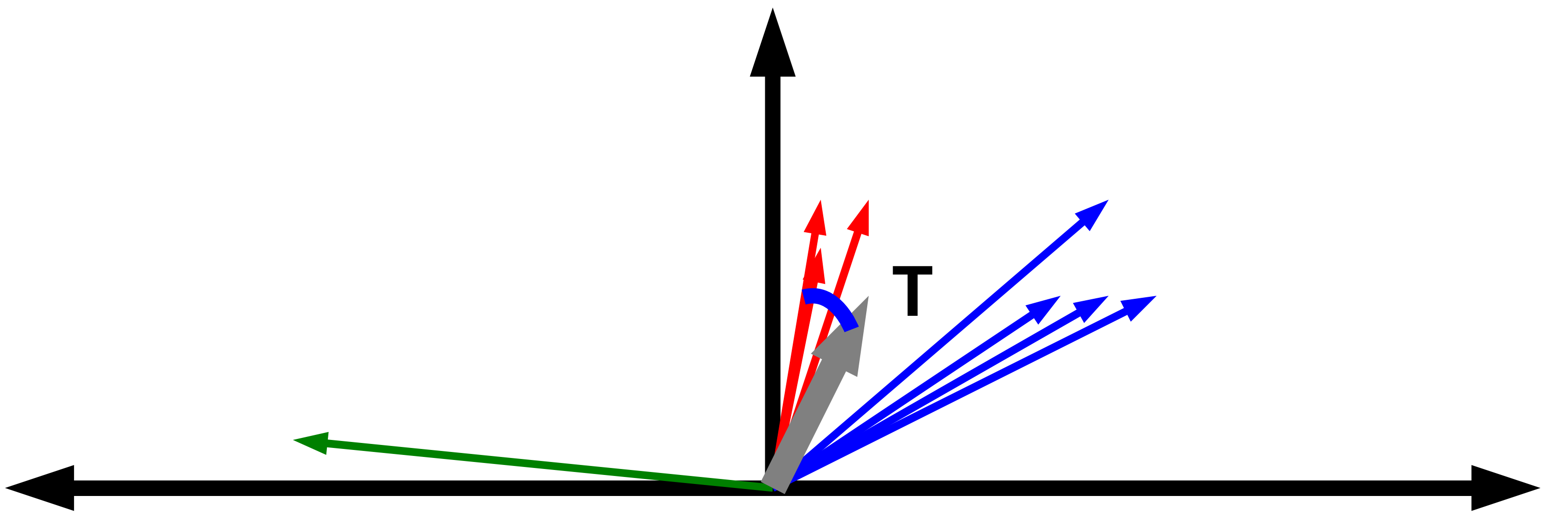
Alignment: What it tells us

- Moderate angle, a bit different from T
- All blue logs have similar alignment



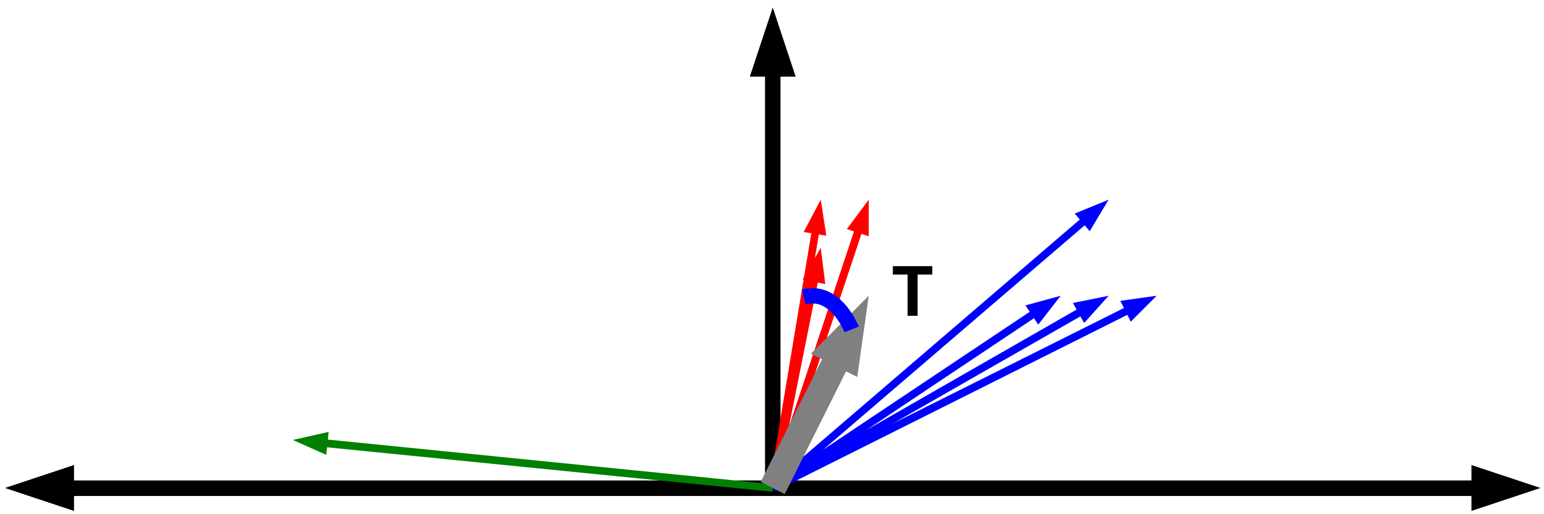
Alignment: What it tells us

- Small angle, fairly similar to T
- All red logs have similar alignment



Alignment: What it tells us

- Does NOT tell us:
  - Green will perform poorly
  - Red will perform well
  - Anything else about relative performance!

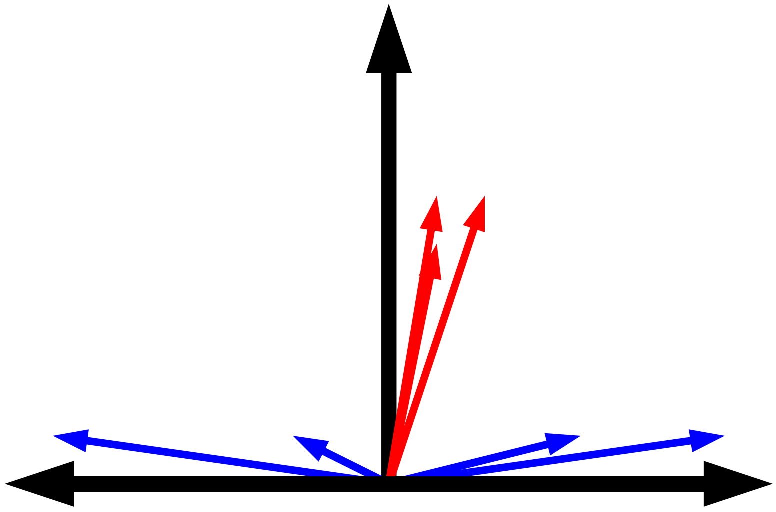


Getting More Information: Cuts

- A real situation:
  - All average difference scores are high
    - 16 logs = 128 pairwise differences
  - Half the alignment scores are low
  - Half the alignment scores are moderate

Cuts

- Separate based on alignment and recalculate:
  - Blue inputs really are different
    - Alignment scores still low
    - Difference scores still high
  - Red inputs are very similar
    - High alignments
    - Low differences



Summary

- Input Characterization is important due to sensitivity of performance evaluation to input selection
- For FDO compiler work, it makes sense to characterize training inputs based on the transformations they induce
- Metrics based on transformation logs can discriminate between inputs

Thanks

Questions?