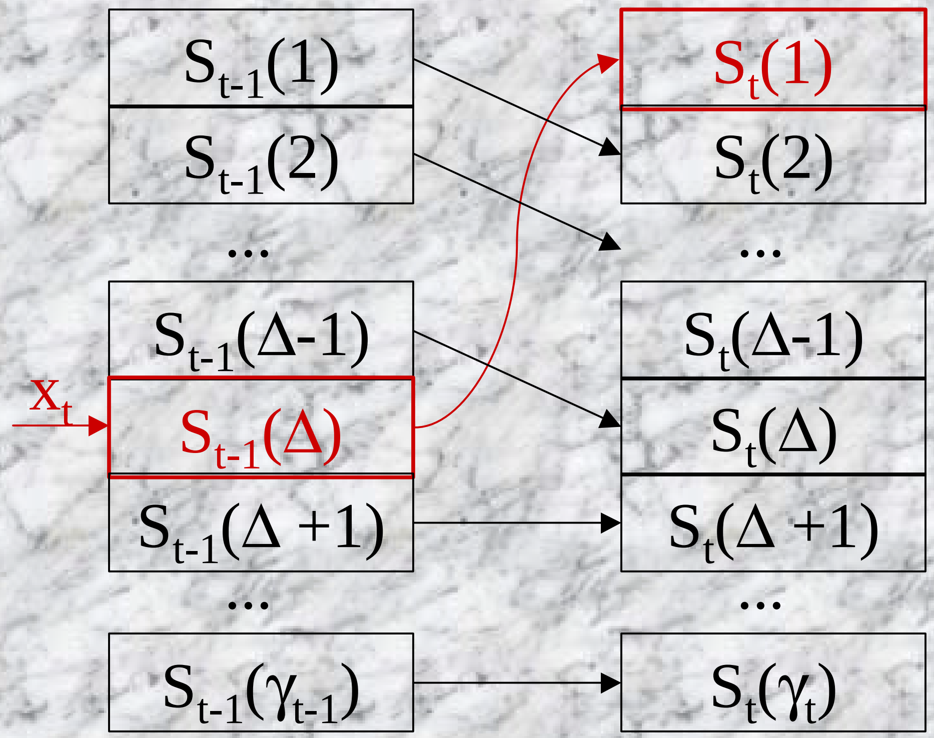
Cache Misses Prediction Using Stack Distances

Calin Cascaval and David A. Padua
Department of computer science
University of Illinois at Urbana-Champaign
cascaval,padua@cs.uiuc.edu

Outline

- Stack Distances as a Metric for Locality
- Run-time Instrumentation
- Compile-time Analysis
- Future Work

Stack Algorithms

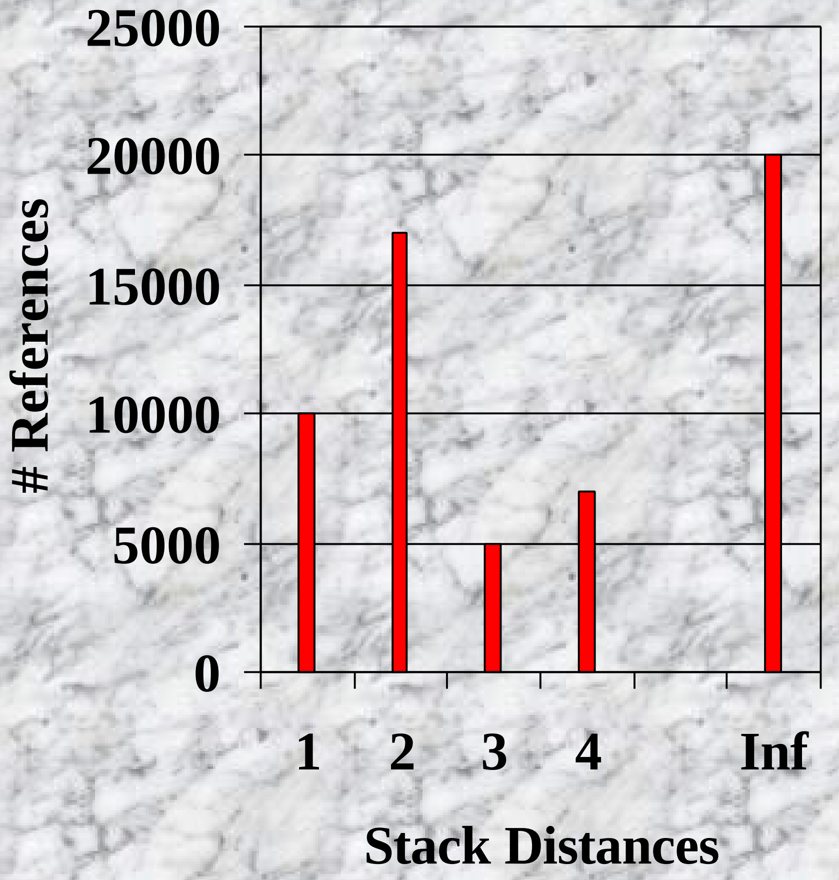


Stack Distances As Cache Misses

- compute the number of cache hits and misses as follows:

$$\text{hits}(C) = \sum_{\Delta=1}^{C} s(\Delta)$$

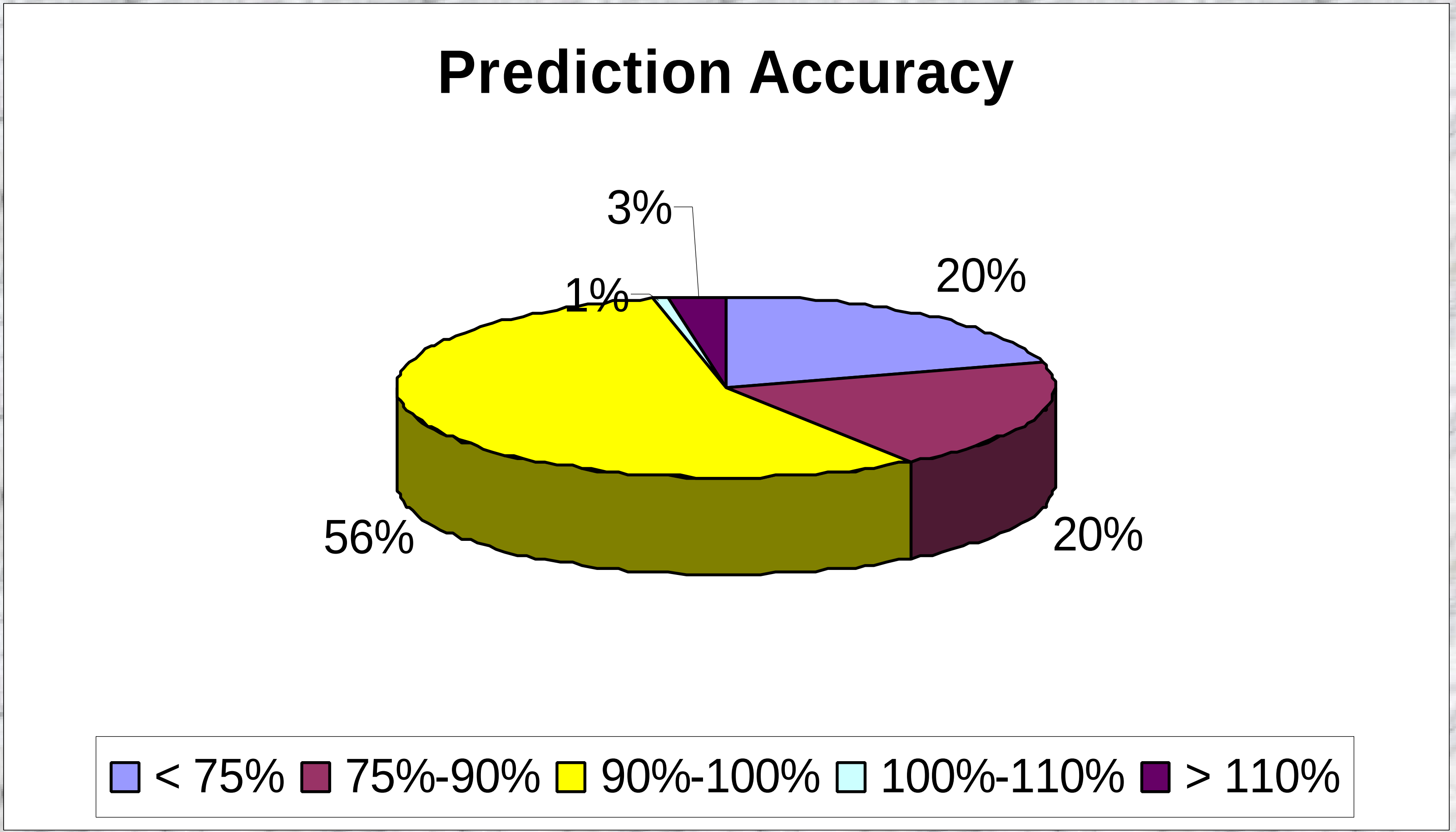
$$\text{misses}(C) = \sum_{\Delta=C+1}^{Inf} s(\Delta)$$



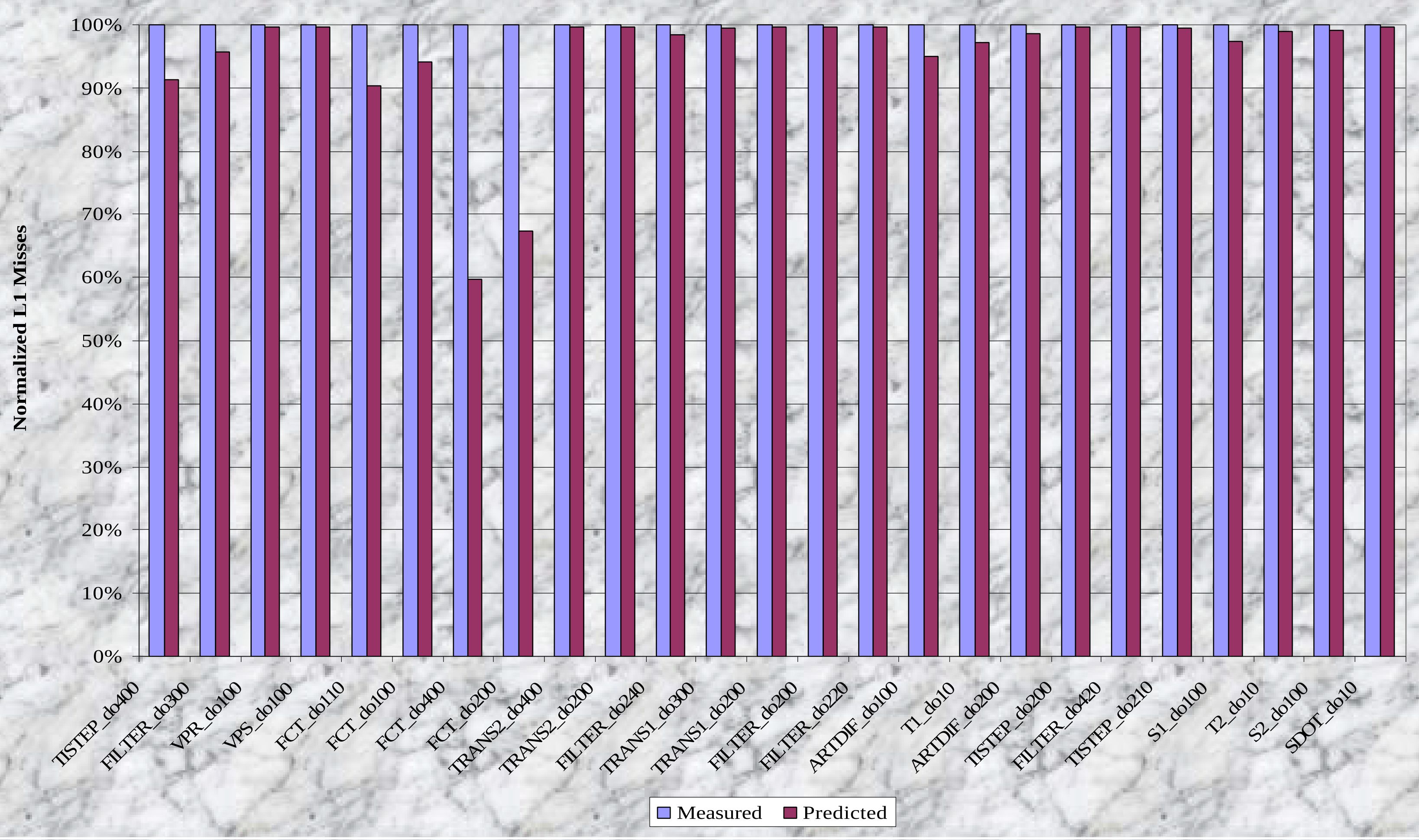
Metric Validation

- Stack algorithm implemented as a library
- Polaris instrumented codes from the SPEC95 and Perfect Club benchmarks
- measured actual number of cache misses using the hardware counters on the R10K processors (for both L1 and L2 caches)

Experiments Serial

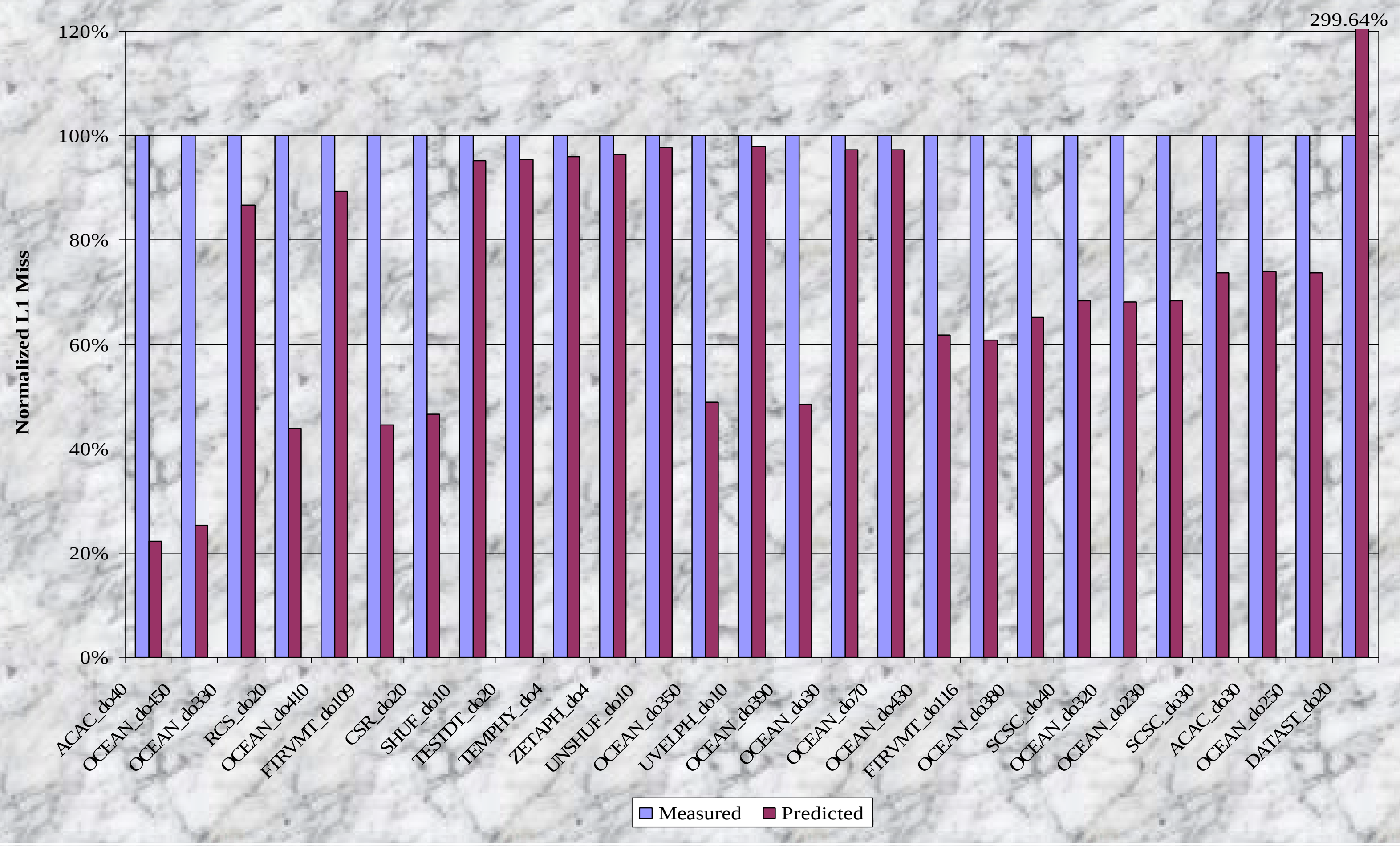


HYDRO2D



Normalized L1 Misses
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
TISTEP_do400
FILTER_do300
VPR_do100
VPS_do100
FCT_do110
FCT_do100
FCT_do400
FCT_do200
TRANS2_do400
TRANS2_do200
FILTER_do240
TRANS1_do300
TRANS1_do200
FILTER_do200
FILTER_do220
ARTDIF_do100
T1_do10
ARTDIF_do200
TISTEP_do200
FILTER_do420
TISTEP_do210
S1_do100
T2_do10
S2_do100
SDOT_do10
Measured
Predicted

Ocean



299.64%
120%
100%
80%
60%
40%
20%
0%
Normalized L1 Miss
ACAC_do40
OCEAN_do450
OCEAN_do330
RCS_do20
OCEAN_do410
FIRVMT_do109
CSR_do20
SHUF_do10
TESTDT_do20
TEMPHY_do4
ZETAPH_do4
UNSHUF_do10
OCEAN_do350
UVELPH_do10
OCEAN_do390
OCEAN_do30
OCEAN_do70
OCEAN_do430
FIRVMT_do116
OCEAN_do380
SCSC_do40
OCEAN_do320
OCEAN_do230
SCSC_do30
ACAC_do30
OCEAN_do250
DATAST_do20
Measured
Predicted

Solution Advantages

- Accurate in most cases.
- One pass through the trace estimates misses for any cache size
- Architecture independent (except for the cache line size)
- Same model applicable to independent loops as well as entire programs
- Easily applicable to parallel programs

Run-time Solution Advantages

- Works in all cases

Run-time Solution Disadvantages

- It is run-time, therefore consumes CPU time
- Cannot easily identify references with bad locality
- Needs separate runs for different cache sizes/processors
- Assumes a fully associative cache (to reduce overhead)

Compile-time Solution

- Integrated within the Polaris parallelizing compiler
- Algorithm based on data dependence distance vectors to compute the stack distances for loops
- Computes symbolic expressions (based on loop bounds) for stack distances and number of references

Compile-time Solution (cont.)

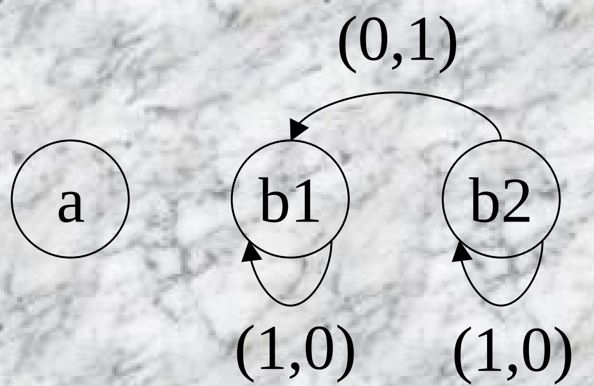
- Preserves the advantages given by the run-time solution, adding
  - misbehaving reference identification
  - array size knowledge may improve accuracy
  - flexibility and locality information readily available in the compiler
- May need a run-time pass for unknown loop bounds or data depending on the input

Example

```
do j = 1, n
  do i = 1, n
    a(i,j) = b(i,1) + b(i+1,1)
```

Distance	References
$\lvert \delta R_i^1 + d - s \rvert$	N-1
Inf	3N-(N-1)

Distance	References
$\lvert \delta R_i^1 + d - s \rvert$	N(N-1)
$\lvert \delta R_j^1 + d - s \rvert$	(N-1)(N+1)
Inf	N^2+N+1



Future work

- Integrating the compile-time solution with the run-time instrumentation
- Solve the false sharing integration within the compiler approach
- Address some of the limitations: cache associativity, multi-word cache lines