A Cache-conscious Profitability Model for Empirical Tuning of Loop Fusion

Apan Qasem Ken Kennedy
Rice University
Houston, TX

Outline

- Motivation
- Related Work
- Profitability Model
  - Using hierarchical classification of reuse
  - Accounting for conflict misses
  - Enforcing resource constraints
  - Tuning fusion parameters
- Preliminary Experiments
- Conclusions and Future Work

Motivation

- Making the right fusion choices is a non-trivial task
  - Optimal fusion known to be NP-complete
  - Profitability depends on the underlying architecture
    - Conflict misses
    - Resource Constraints
  - Exploiting inter-loop nest locality is not enough

outer loop reuse in a()

```
L1: do j = 1, N
      do i = 1, M
        b(i,j) = a(i,j)+a(i,j-1)+a(i,j-2)
      enddo
    enddo
```

loop-crossing reuse in b()

```
L2: do j = 1, N
      do i = 1, M
        c(i,j) = b(i,j) + d(i,j)
      enddo
    enddo
```

lost reuse in a()

```
L12: do j = 1, N
       do i = 1, M
         b(i,j) = a(i,j)+a(i,j-1)+a(i,j-2)
         c(i,j) = b(i,j) + d(i,j)
       enddo
     enddo
```

saved loads for b()

Fused loop nest from weather modeling application

```
   do j = -2, D__ujUpper00, 2
     do i = -2, N
       do k = -2, N
         ab(k, i, j) = (f60 * (a(k, i, j - 1) + a(k, i, j)) + f61 *
* (a(k, i, j - 2) + a(k, i, j + 1)) + f62 * (a(k, i, j - 3) + a(k,
*i, j + 2))) * thirddtbydy * uyb(k, i, j)
         ab(k, i, j + 1) = (f60 * (a(k, i, j) + a(k, i, j + 1)) + f
*61 * (a(k, i, j - 1) + a(k, i, j + 2)) + f62 * (a(k, i, j - 2) + a
*(k, i, j + 3))) * thirddtbydy * uyb(k, i, j + 1)
         al(k, i, j) = (f60 * (a(k, i - 1, j) + a(k, i, j)) + f61 *
* (a(k, i - 2, j) + a(k, i + 1, j)) + f62 * (a(k, i - 3, j) + a(k,
*i + 2, j))) * thirddtbydx * uxl(k, i, j)
         al(k, i, j + 1) = (f60 * (a(k, i - 1, j + 1) + a(k, i, j +
* 1)) + f61 * (a(k, i - 2, j + 1) + a(k, i + 1, j + 1)) + f62 * (a(
*k, i - 3, j + 1) + a(k, i + 2, j + 1))) * thirddtbydx * uxl(k, i,
*j + 1)
         af(k, i, j) = (f60 * (a(k - 1, i, j) + a(k, i, j)) + f61 *
* (a(k - 2, i, j) + a(k + 1, i, j)) + f62 * (a(k - 3, i, j) + a(k +
* 2, i, j))) * thirddtbydz * uzf(k, i, j)
         af(k, i, j + 1) = (f60 * (a(k - 1, i, j + 1) + a(k, i, j +
* 1)) + f61 * (a(k - 2, i, j + 1) + a(k + 1, i, j + 1)) + f62 * (a(
*k - 3, i, j + 1) + a(k + 2, i, j + 1))) * thirddtbydz * uzf(k, i,
*j + 1)
         athird(k, i, j) = a(k, i, j) + (al(k, i, j) - al(k, i - 1,
* j)) + (ab(k, i, j) - ab(k, i, j - 1)) + (af(k, i, j) - af(k - 1,
*i, j))
         athird(k, i, j + 1) = a(k, i, j + 1) + (al(k, i, j + 1) -
*al(k, i - 1, j + 1)) + (ab(k, i, j + 1) - ab(k, i, j)) + (af(k, i,
* j + 1) - af(k - 1, i, j + 1))
         abthird(k, i, j) = (f60 * (athird(k, i, j - 1) + athird(k,
* i, j)) + f61 * (athird(k, i, j - 2) + athird(k, i, j - 4)) + f62
** (athird(k, i, j - 3) + athird(k, i, j - 5))) * halfdtbydx * uybt
*hird(k, i, j)
         abthird(k, i, j + 1) = (f60 * (athird(k, i, j) + athird(k,
* i, j + 1)) + f61 * (athird(k, i, j - 1) + athird(k, i, j - 3)) +
*f62 * (athird(k, i, j - 2) + athird(k, i, j - 4))) * halfdtbydx *
*uybthird(k, i, j + 1)
         althird(k, i, j) = (f60 * (athird(k, i - 1, j) + athird(k,
* i, j)) + f61 * (athird(k, i - 2, j) + athird(k, i - 4, j)) + f62
** (athird(k, i - 3, j) + athird(k, i - 5, j))) * halfdtbydx * uxlt
*hird(k, i, j)
         althird(k, i, j + 1) = (f60 * (athird(k, i - 1, j + 1) + a
*third(k, i, j + 1)) + f61 * (athird(k, i - 2, j + 1) + athird(k, i
* - 4, j + 1)) + f62 * (athird(k, i - 3, j + 1) + athird(k, i - 5,
*j + 1))) * halfdtbydx * uxlthird(k, i, j + 1)
         afthird(k, i, j) = (f60 * (athird(k - 1, i, j) + athird(k,
* i, j)) + f61 * (athird(k - 2, i, j) + athird(k - 4, i, j)) + f62
** (athird(k - 3, i, j) + athird(k - 5, i, j))) * halfdtbydx * uzft
*hird(k, i, j)
         afthird(k, i, j + 1) = (f60 * (athird(k - 1, i, j + 1) + a
*third(k, i, j + 1)) + f61 * (athird(k - 2, i, j + 1) + athird(k -
*4, i, j + 1)) + f62 * (athird(k - 3, i, j + 1) + athird(k - 5, i,
*j + 1))) * halfdtbydx * uzfthird(k, i, j + 1)
         ahalf(k, i, j) = a(k, i, j) + (althird(k, i, j) - althird(
*k, i - 1, j)) + (abthird(k, i, j) - abthird(k, i, j - 1)) + (afthi
*rd(k, i, j) - afthird(k - 1, i, j))
         ahalf(k, i, j + 1) = a(k, i, j + 1) + (althird(k, i, j + 1
*) - althird(k, i - 1, j + 1)) + (abthird(k, i, j + 1) - abthird(k,
* i, j)) + (afthird(k, i, j + 1) - afthird(k - 1, i, j + 1))
       enddo
     enddo
   enddo
```

Related Work

- Heuristic algorithms to find good fusion solutions
  - Gao et. al. [92], Kennedy [00], Lim and Lam [01],
- Approaches that aim to reduce bandwidth
  - Ding and Kennedy [01], Song et. al. [01]
- Main distinction from previous work
  - Use of architecture specific information
  - Empirical tuning of fusion parameters

Outline

- Motivation
- Related Work
- **Profitability Model**
  - Using hierarchical classification of reuse
  - Accounting for conflict misses
  - Enforcing resource constraints
  - Tuning fusion parameters
- Preliminary Experiments
- Conclusions and Future Work

Hierarchical Reuse

- Use the concept of ***reuse level*** as a way to quantify reuse at each level of the memory hierarchy
- Associate with each reference a value that expresses the level at which the reuse is exploited

Reuse Level = smallest k such that

Reuse Distance $\leq$ Capacity(L_k)

Hierarchical Reuse

- Obtain benefit from reuse of **r** only if

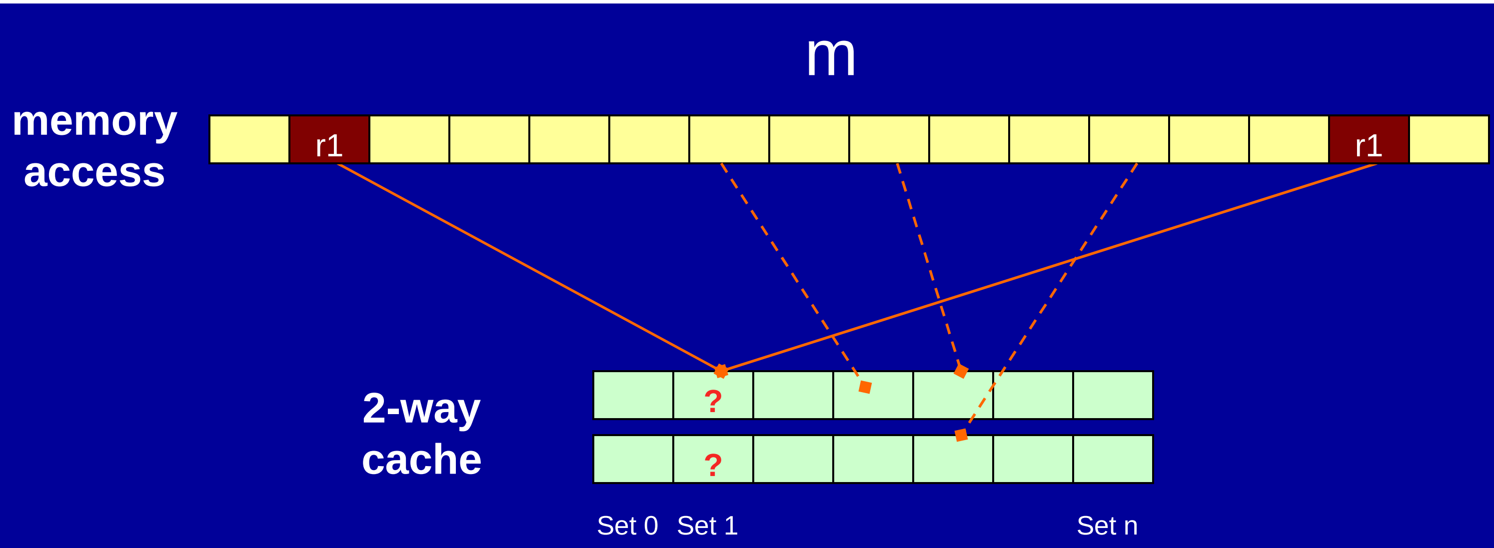
$$\textit{Reuse Level(r)}_{pre} > \textit{Reuse Level(r)}_{post}$$

- Perform this check for every reused reference
- Account for miss access cost for each level of memory

Conflict Miss Model

- Use a probabilistic model to predict when a conflict miss might occur
  - Derived from Hill & Smith model for associativity [HS:IEEE89]
- Ask the question:

  *If **m** distinct cache lines are accessed between references to the same cache line **r** what is the probability that **n** of them are going to land in the line occupied by **r**?*



Set P to be ≤ T

$$P = 1 - \sum_{i=0}^{a-1} \binom{m}{i} \left[\frac{1}{s}\right]^{i} \left[\frac{s-1}{s}\right]^{m-i}$$

Effective Cache Capacity

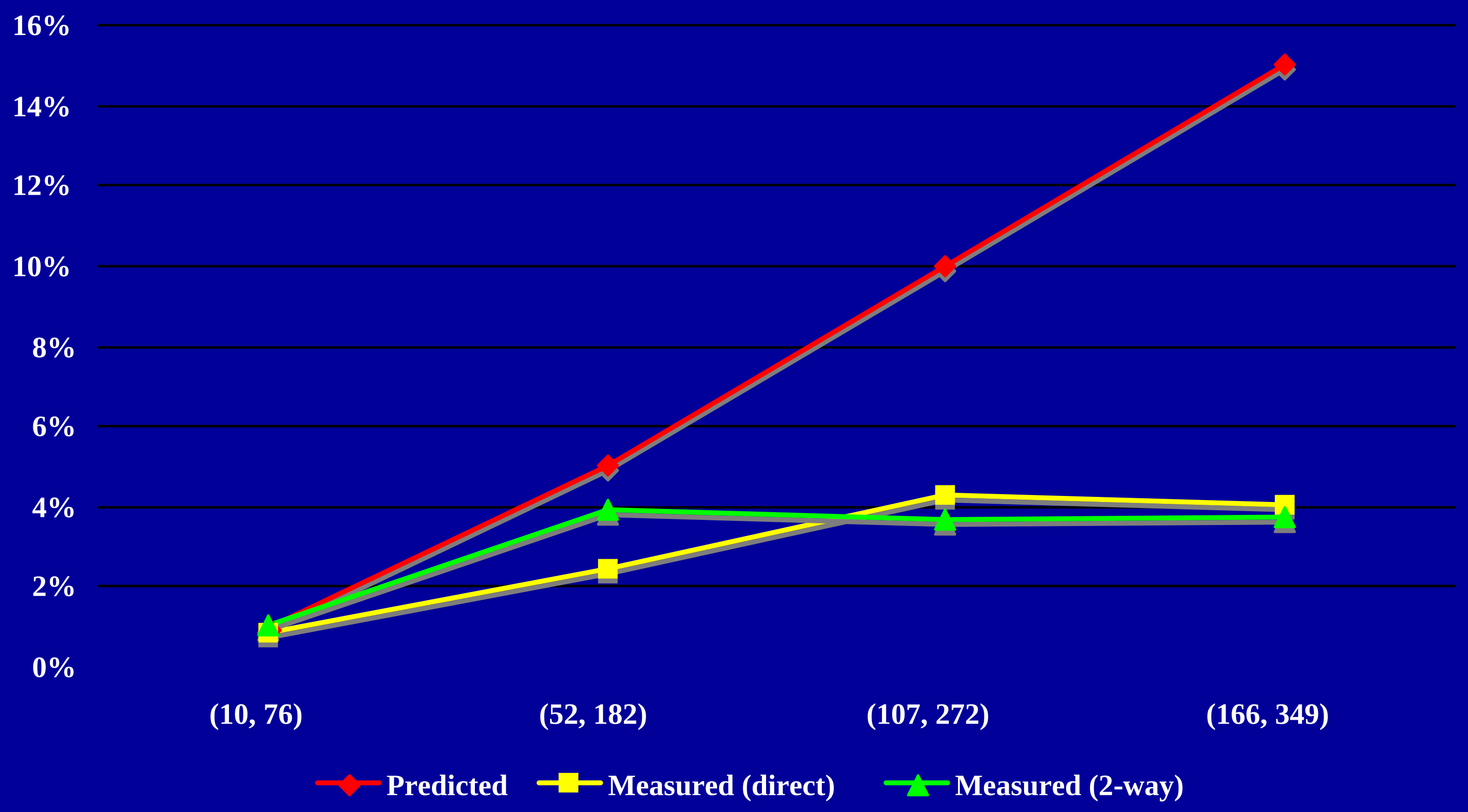
$$m \le E(a,s,T)$$

Effective Cache Capacity

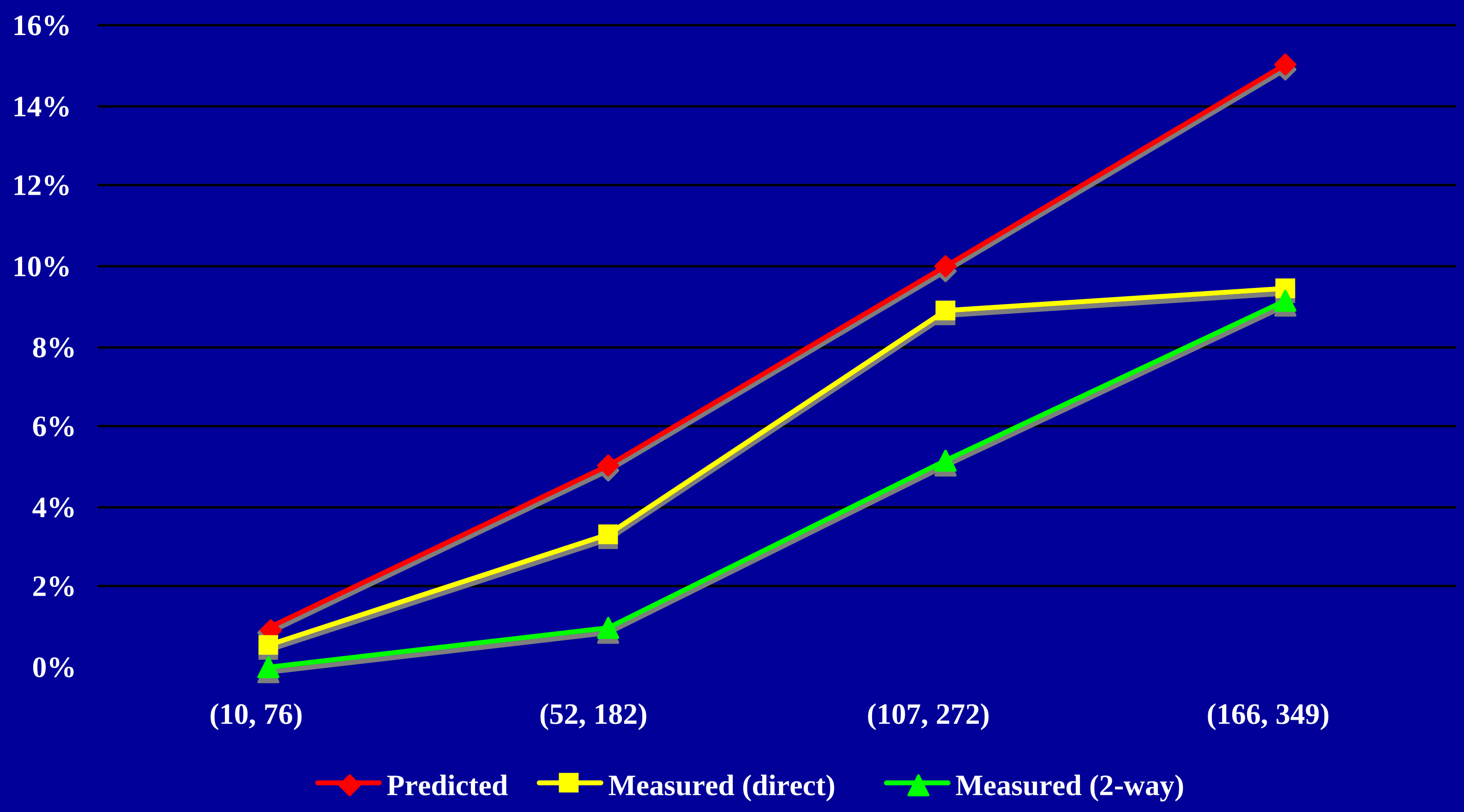
- Effective cache capacity is the maximum reuse distance for which we can expect a reused value to still be in cache
- We adjust the definition of reuse level based on the definition of effective cache capacity

Reuse Level = smallest k such that
Reuse Distance $\leq$ ***ECC(*** L_k ***)***

Evaluation of Conflict Miss Model: `erlebacher`



Evaluation of Conflict Miss Model: `arraysweep`



Resource Constraints

– Need to constrain resource demands of fused loop

Register Pressure(L_{fused}) < Register Set Size
Instructions(L_{fused}) < I-Cache Capacity

– Easy to incorporate into a constrained weighted fusion algorithm

Parameterizing the Model

- Parameters amenable to tuning
  - Effective Cache Capacity
  - Register Set Size
  - I-Cache Capacity

Parameterizing the Model

– Use a ***tolerance factor*** to determine how much of a resource we can use at each tuning step

Effective Registers = T x Register Set Size
$[0 < T \leq 1]$

Effective Cache Capacity = E(a, s, T)
$[0.01 \leq T \leq 0.20]$

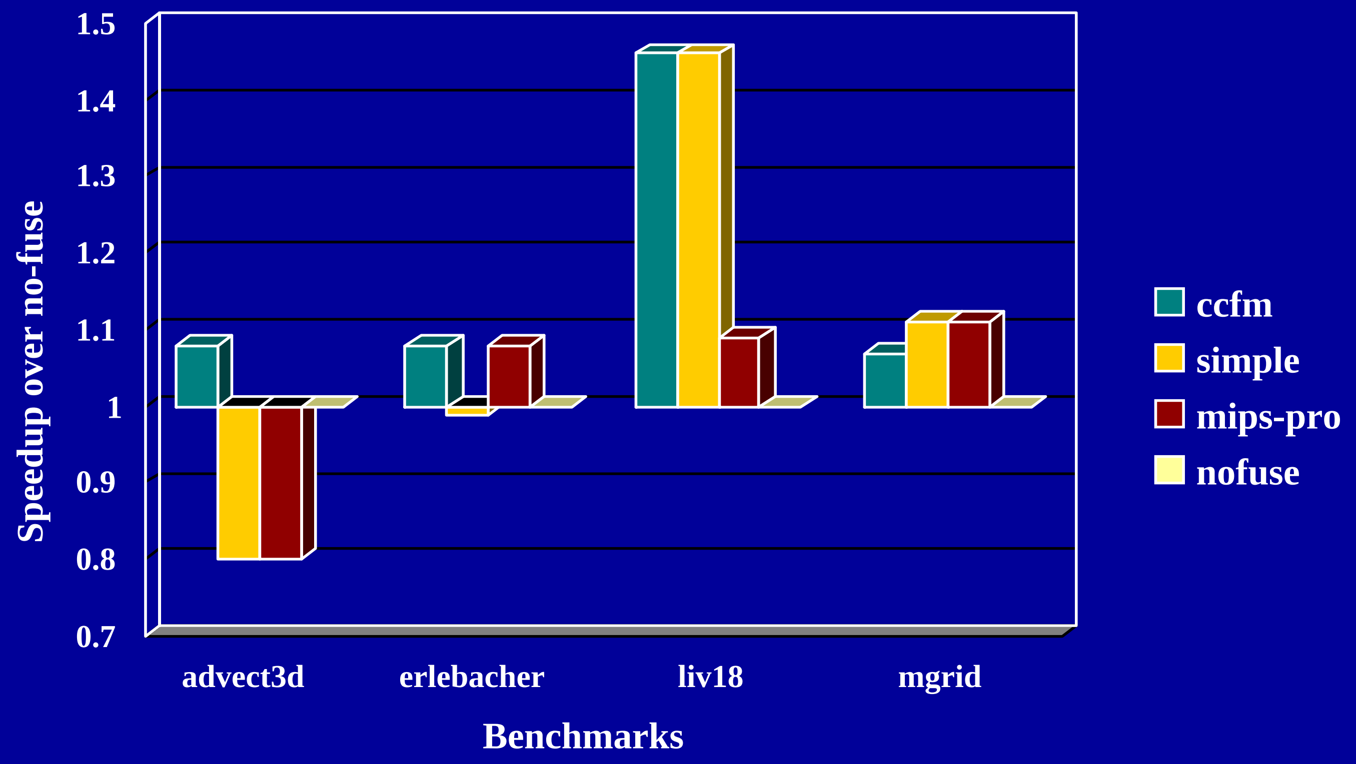
Tuning Fusion Parameters

- Start off conservatively with a low tolerance value and increase tolerance at each step
- Each tuning parameter constitutes a single search dimension
- Search is ***sequential*** and ***orthogonal***
  - stop when performance starts to worsen
  - use reference values for other dimension when searching a particular dimension

Experimental Setup

- Four different strategies
  - ccfm, simple, mips-pro, no-fuse
- Four benchmarks
  - advect3d, erlebacher, livermore18, mgrid
- Platform
  - SGI R12K
  - 2-level cache hierarchy
  - Primary L1 I-Cache, Unified L2

Performance Improvement Summary

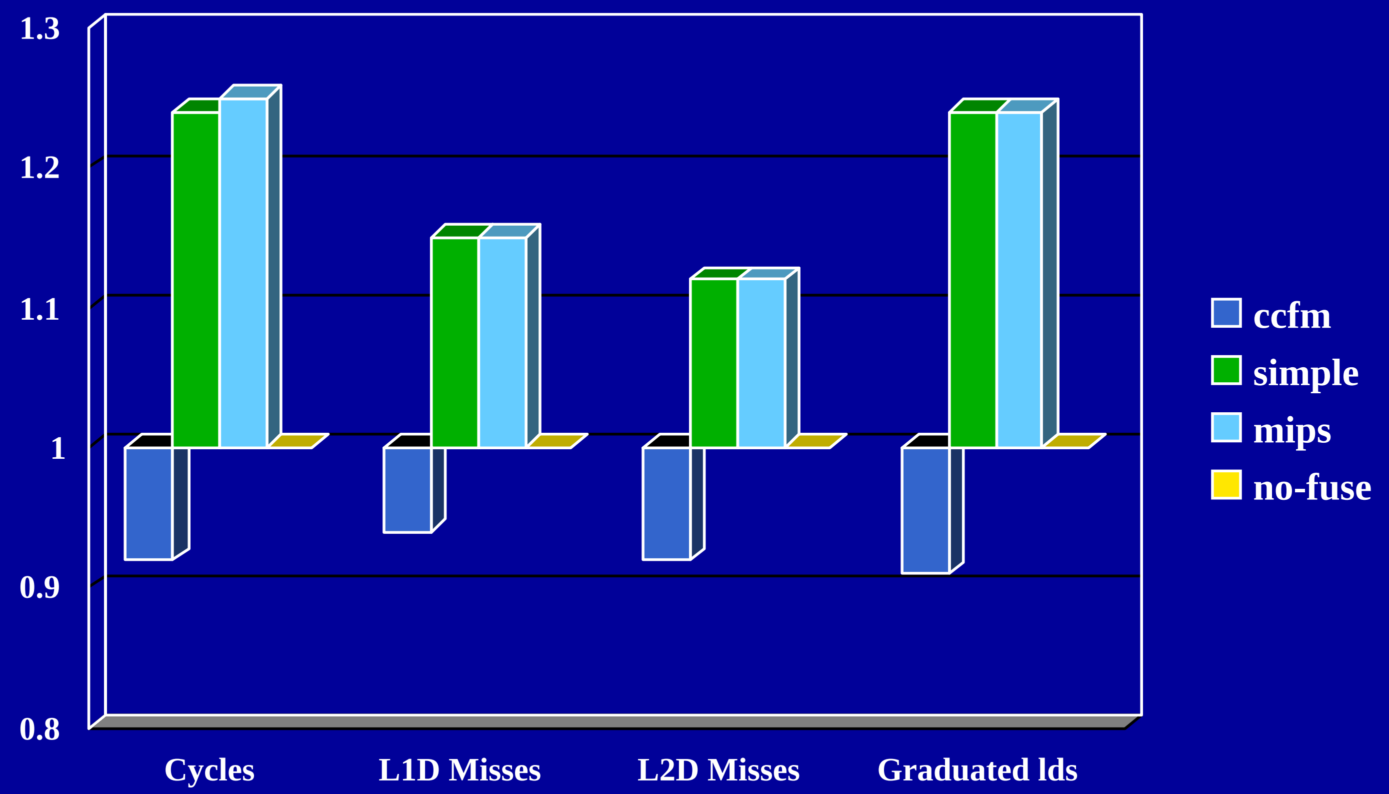


Conclusions

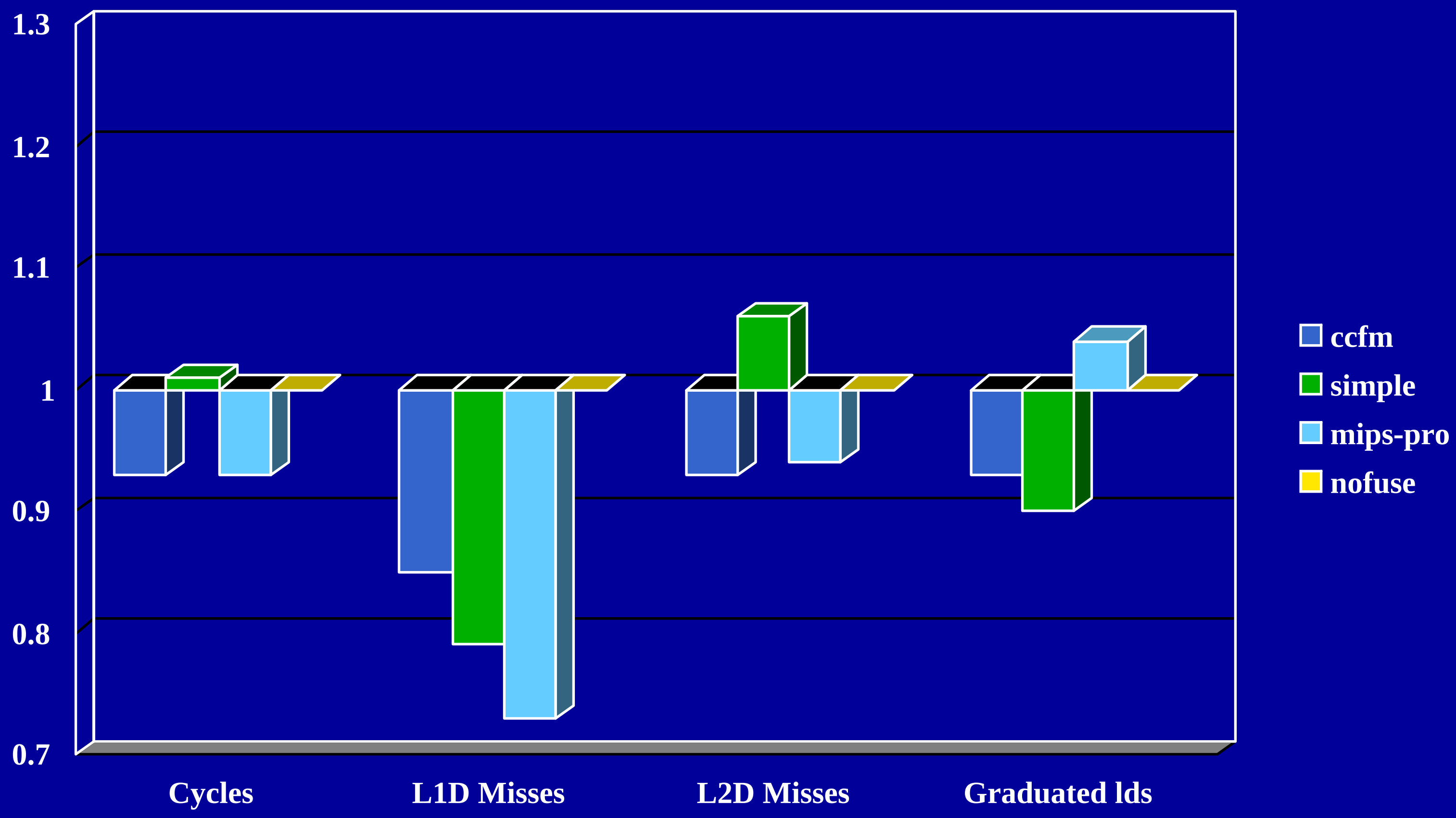
- Detailed cache effect analysis combined with empirical search can lead to better fusion choices
- Overall memory performance can be further improved by considering fusion and tiling interactions

Extra Slides Begin Here

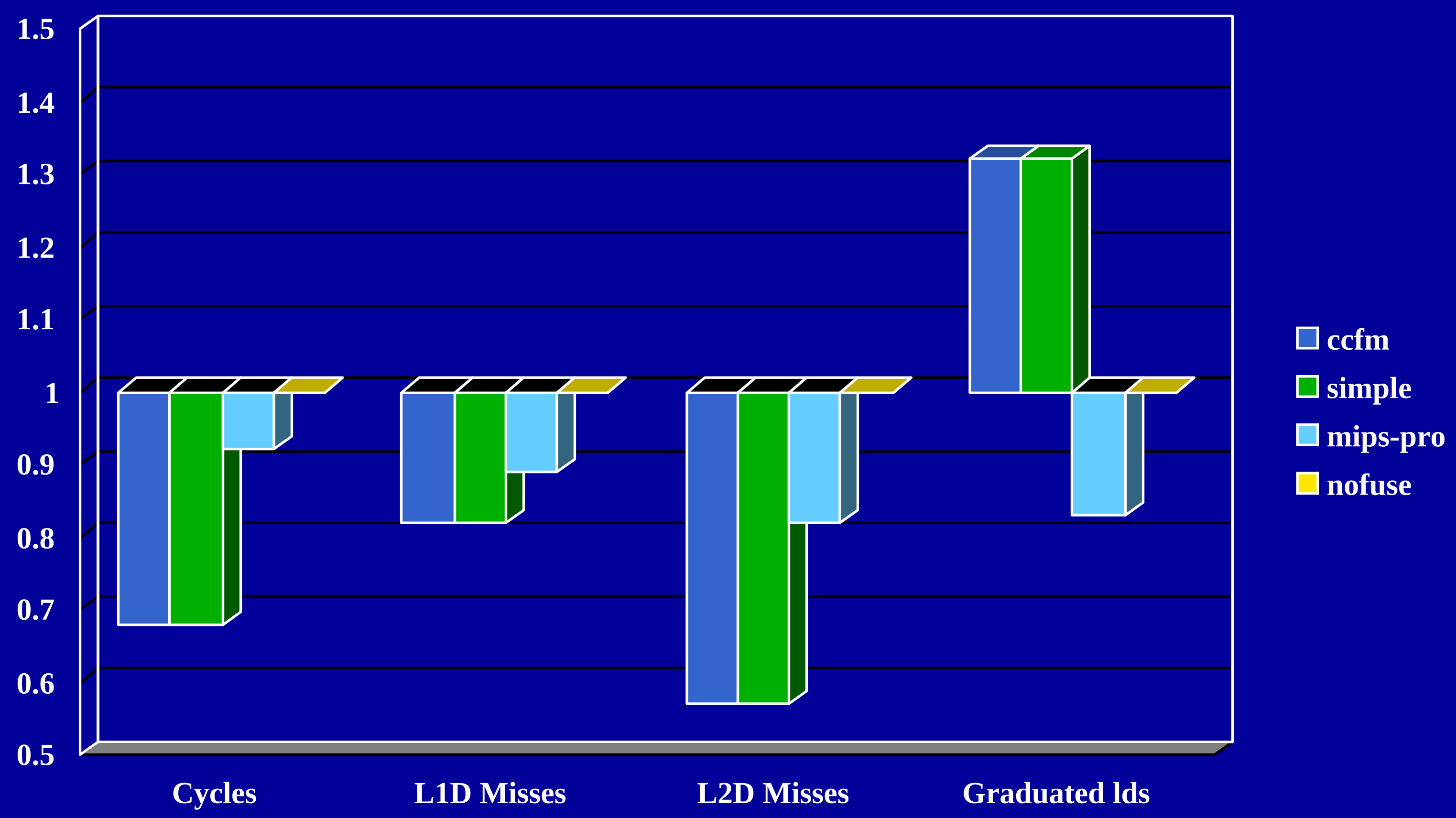
Memory Performance Comparison: `advect3d`



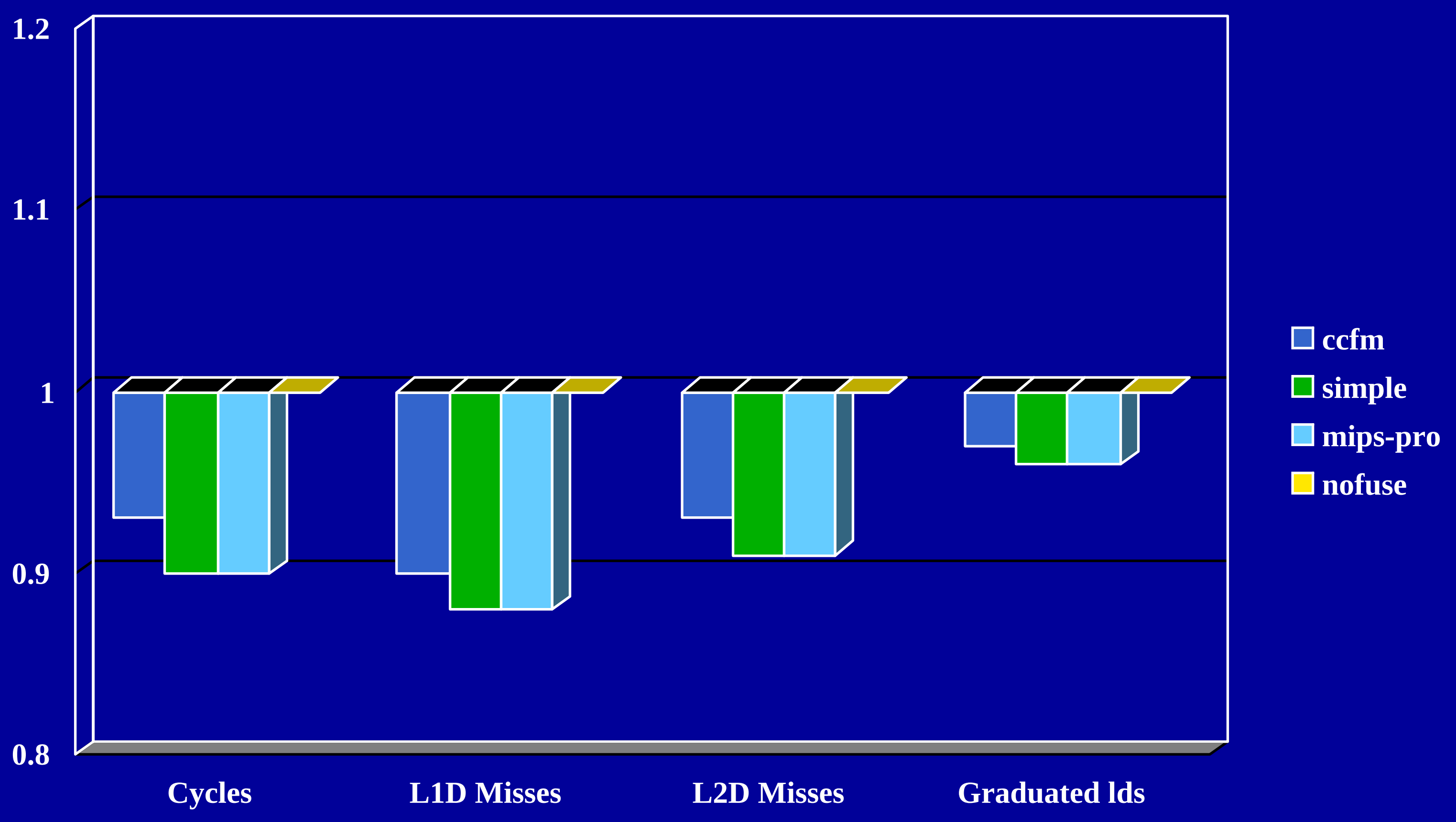
Memory Performance Comparison: `erlebacher`



Memory Performance Comparison: `livermore18`



Memory Performance Comparison: `mgrid`



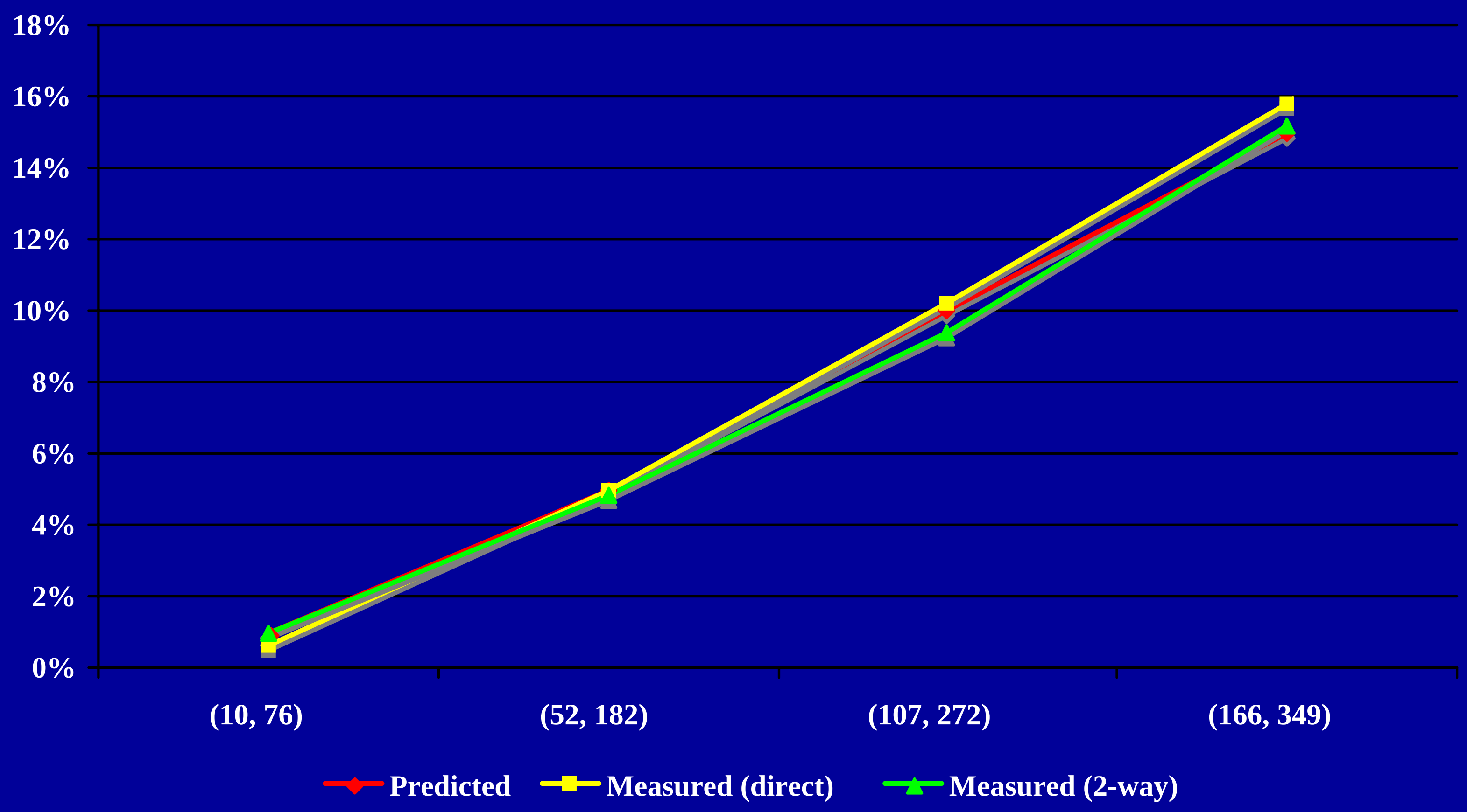
Experimental Results on `advect3d`

Fusion Strategy	Cycle Count	L1D Misses	L2D Misses	Graduated Loads	Speedup
ccfm	8.41E+04	4.48E+04	5.13E+04	3.66E+05	1.17
simple	1.23E+05	3.78E+04	5.08E+04	4.26E+05	0.80
mips-pro	9.88E+04	3.76E+04	9.19E+04	3.06E+05	1.00
nofuse	9.88E+04	3.76E+04	9.19E+04	3.06E+05	1.00

Experimental Results on `erlebacher`

Fusion Strategy	Cycle Count	L1D Misses	L2D Misses	Graduated Loads	Speedup
ccfm	5.23E+09	2.00E+08	2.72E+07	4.02E+08	1.08
simple	5.68E+09	1.85E+08	3.09E+07	3.90E+08	0.99
mips-pro	5.23E+09	1.70E+08	2.74E+07	4.52E+08	1.08
nofuse	5.65E+09	2.34E+08	2.92E+07	4.34E+08	1.00

Evaluation of Conflict Miss Model : `randaccess`



Putting It All Together

- Use hierarchical reuse analysis and conflict miss model to assign ***weights*** between fusible loops
- Use weights to drive a ***resource constraint-based*** fusion algorithm
- Empirically tune for ***effective cache capacity*** and other parameters