UNIVERSITY of ROCHESTER

Program Assisted Cache Management

Realizing Cache Clairvoyance

Jacob Brock, Xiaoming Gu,
Chen Ding, Bin Bao

Telling the Future

Past is prologue.

- How do we know whether a block belongs in the cache?
- Look at patterns in past performance - make predictions.

Outline

- OPT - An optimal cache replacement policy.
- LRU and MRU: educated guesses.
- Is there a way to do better?
- Pacman Simulations
- Proof of Concept

OPT - An Optimal Cache Replacement Policy

- Replace the page that will not be used for the longest time.
- Pro - It is the **best possible** cache management policy.
- Con - requires knowing future memory requests.

OPT Stack

- Stack distance is smallest OPT cache that will have the block for a hit.
- (1) **Upward Movement:** Move to the top when called
- (2) **Downward Movement:** Vacancies filled by the element above it with the lowest locality.
- (3) **Tiebreaking:** Arbitrary (alphabetical)

Trace	a	b	c	a	d	b	a	d	c	d
OPT Stack 1	a	b	c	a	d	b	a	d	c	d
2		a	a	c	a	a	b	a	d	c
3			b	b	b	d	d	b	a	a
4				c	c	c	c	b	b	
Stack Dist	∞	∞	∞	2	∞	3	2	3	4	2

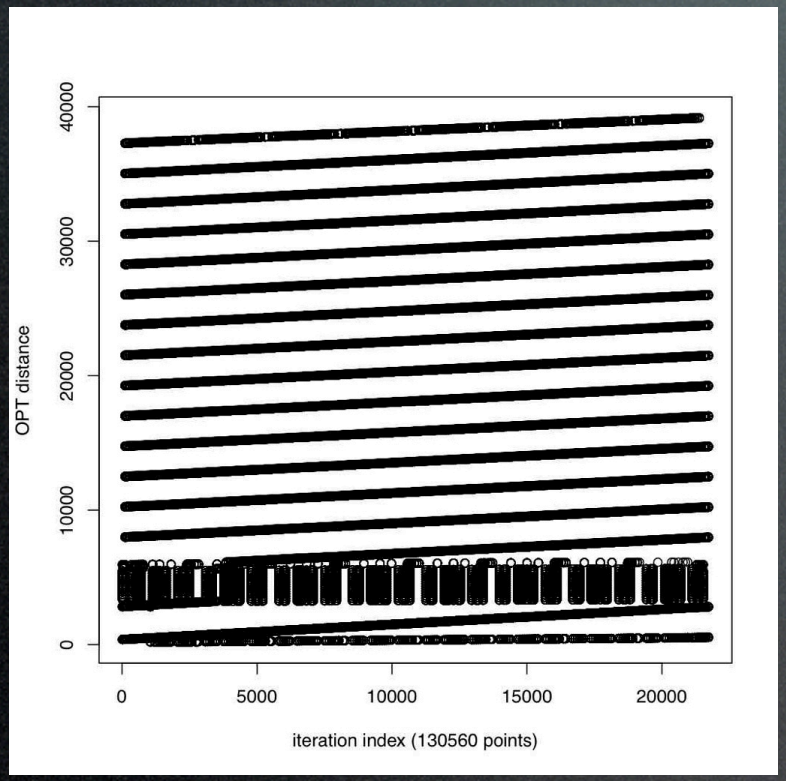
What to Look For

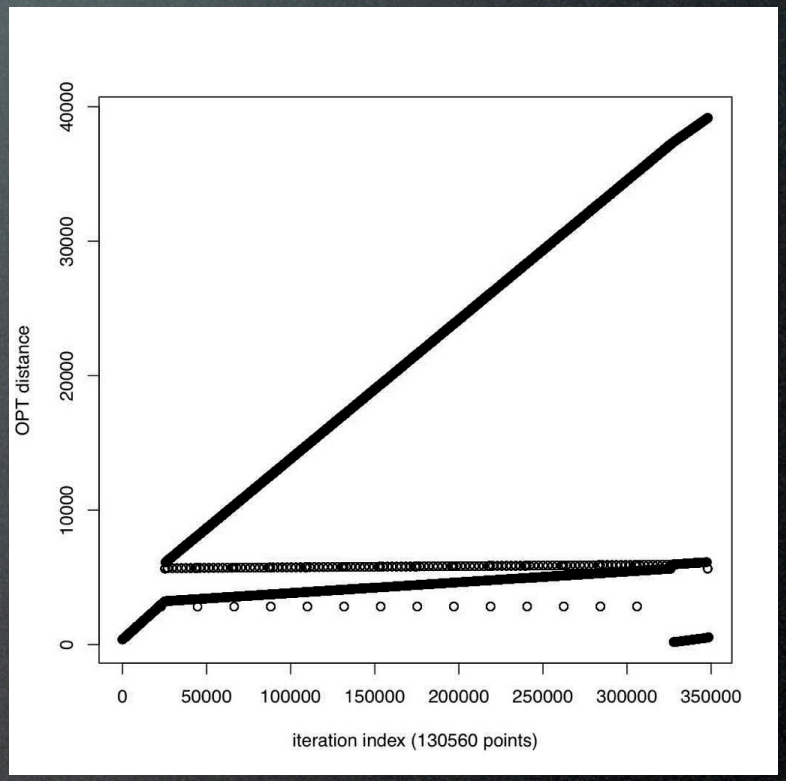
- OPT distance for each reference at each use.
- **Sometimes:** clear linear patterns.

Hint Insertion

- Armed with this, each block can be flagged for LRU or MRU based on the predicted OPTD.
- OPTD > cache size: Flag as MRU
- OPTD < cache size: Flag as LRU

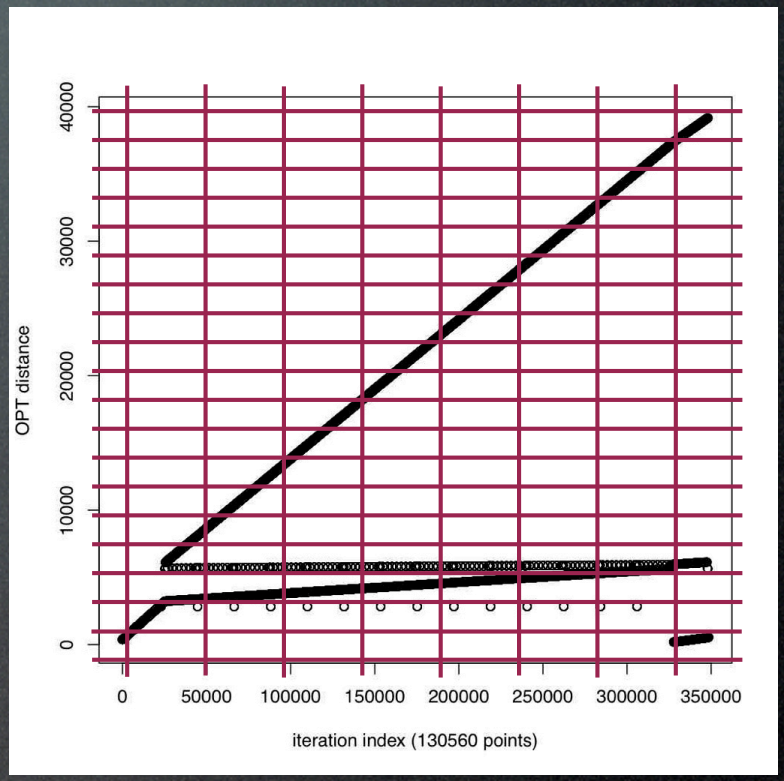
Example Patterns (applu)





Pattern Recognition with "grid regression"

- Run a linear regression on the points in each tile.
- Connect the tiles that contain the same linear patterns.

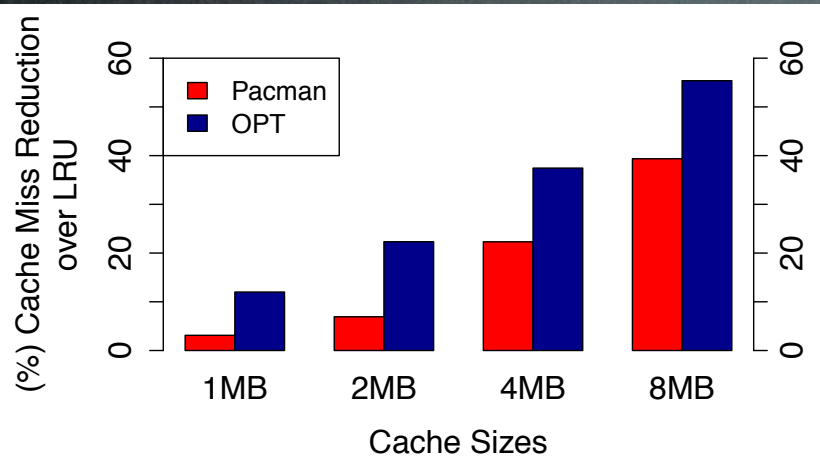


Pattern Prediction

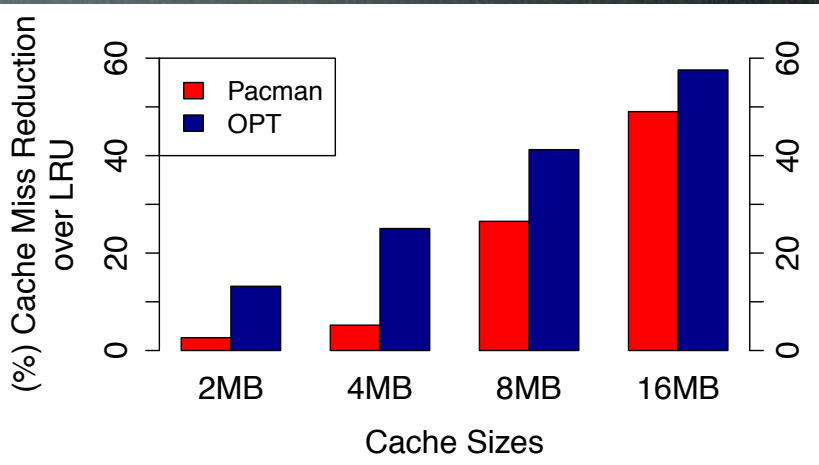
- 3 Runs
  - Baseline (1)
  - Training (2)
  - Real (3)
- Parameters
  - Input size z
  - Slope m
  - Intercept b
- Scaling
  - Proportional with z
  - $\Delta X_i = X_i - X_1$

- $\Delta b_3 = \Delta b_2 \, (\Delta z_3 / \Delta z_2)$
- $m_3 = \text{avg}(m_1, m_2)$

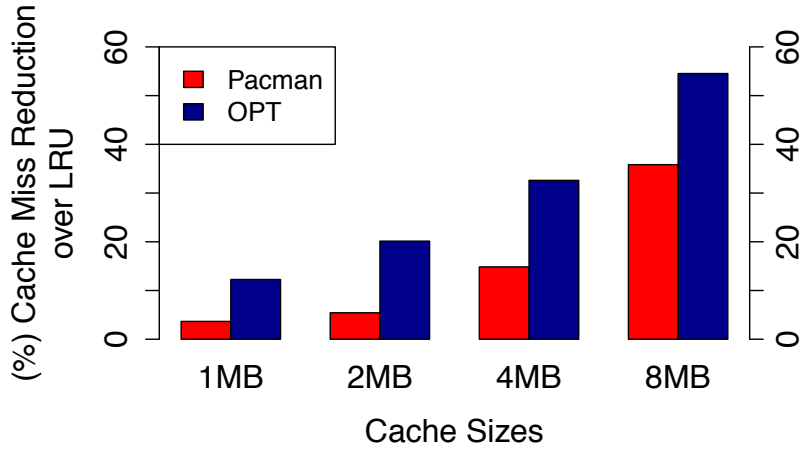
Simulation Results



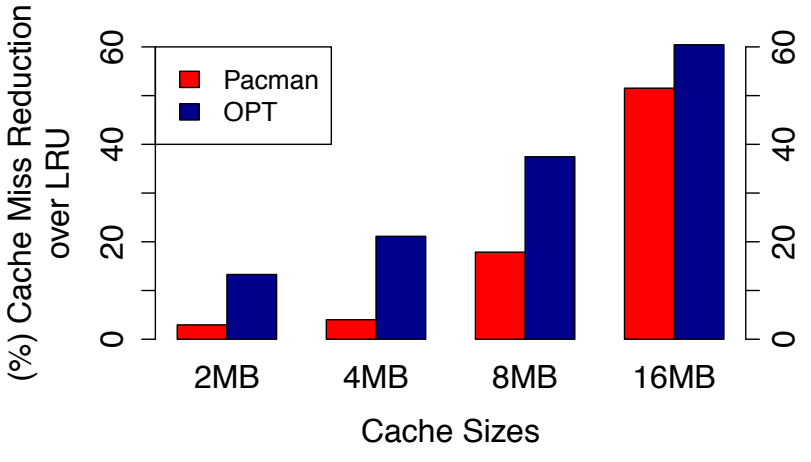
(a) Swim, training on input 384 by 384 and testing on 384 by 384



(b) Swim, training on input 256 by 256 and 384 by 384 and testing on 512 by 512

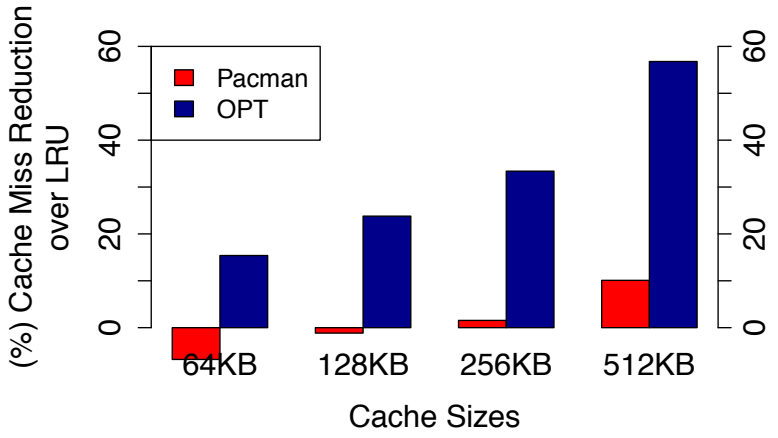


(c) Swim, training on input 384 by 384 and testing on 200 by 737

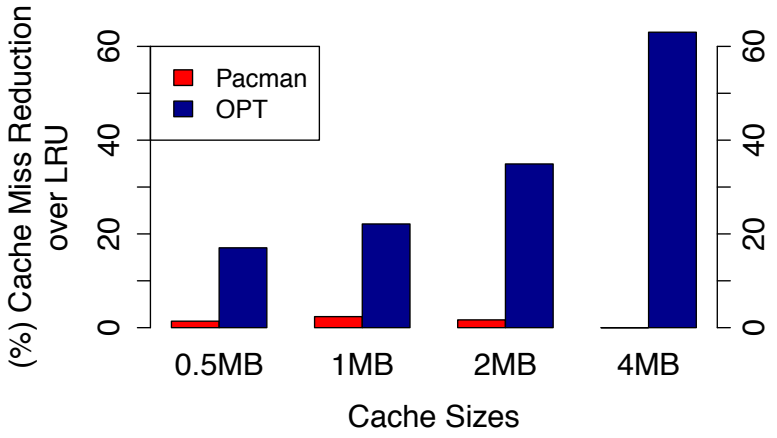


(d) Swim, training on input 256 by 256 and 384 by 384 and testing on 300 by 873

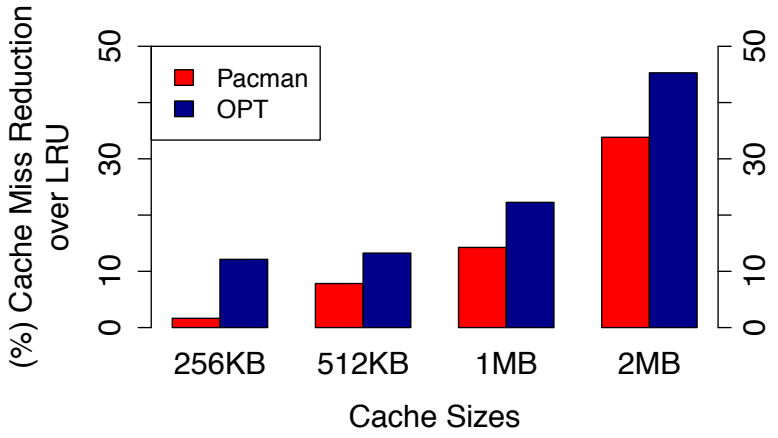
Simulation Results



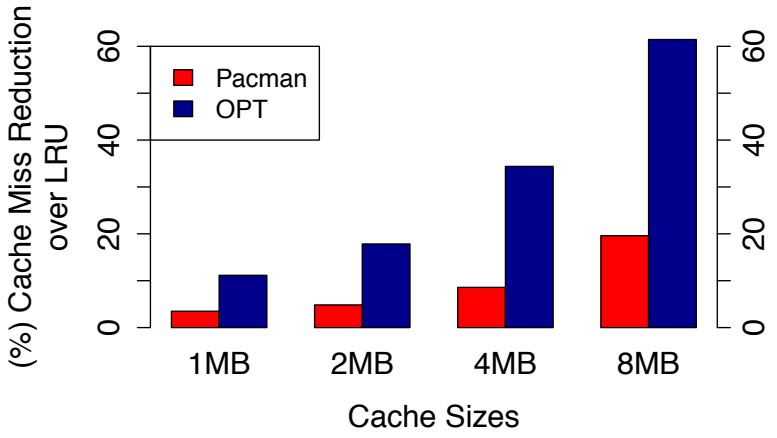
(c) Mgrid, training on *SIZE*=2^5 and testing on *SIZE*=2^5



(d) Mgrid, training on *SIZE*=2^4 and *SIZE*=2^5 and testing on *SIZE*=2^6

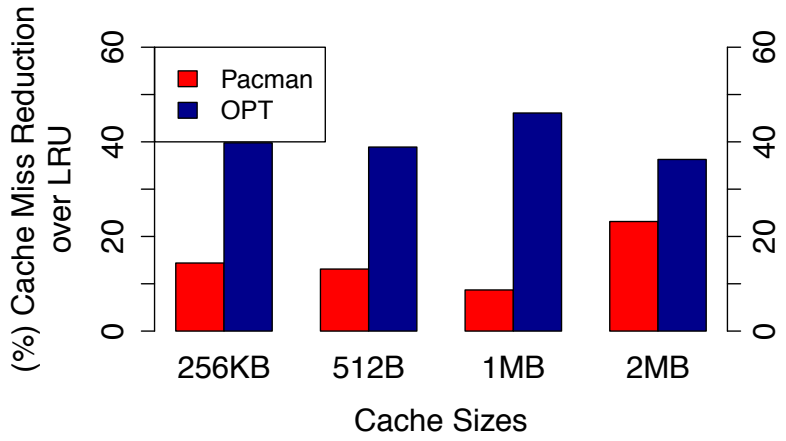


(e) Applu, training on inputs 18 by 18 by 18 and testing on 18 by 18 by 18

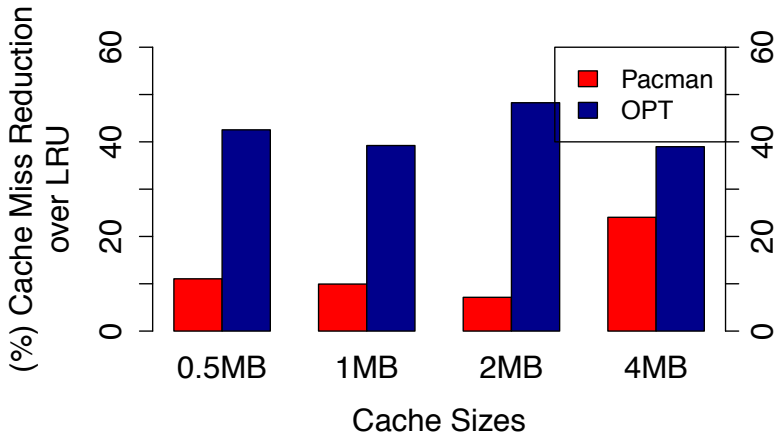


(f) Applu, training on inputs 12 by 12 by 12 and 18 by 18 by 18 and testing on input 24 by 24 by 24

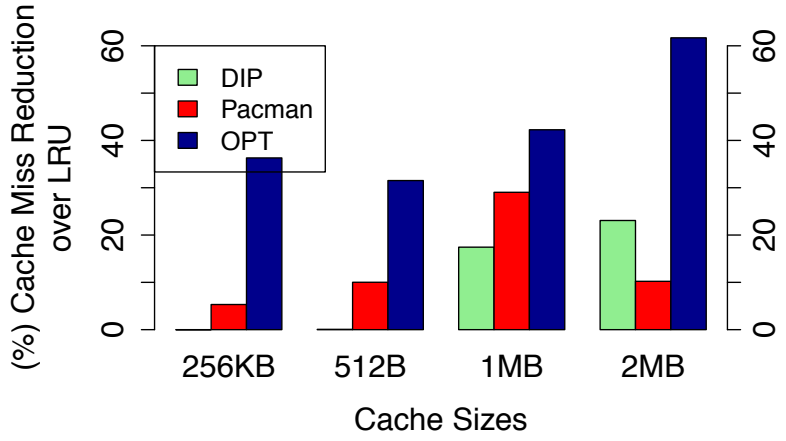
Simulation Results



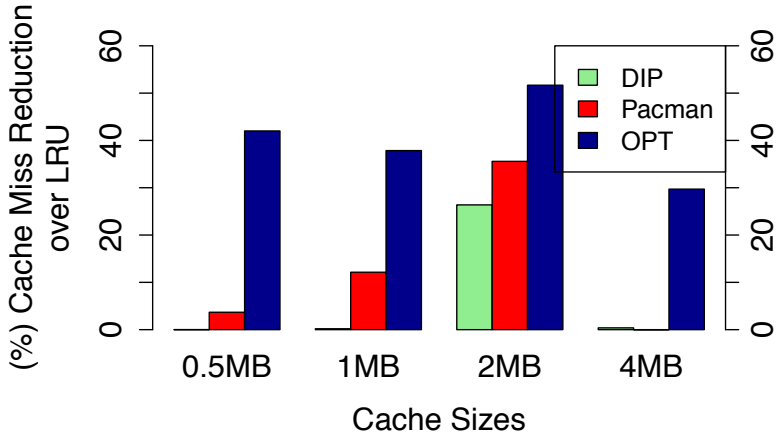
(a) Leslie3d, training on input 31 by 31 by 2 and testing on 31 by 31 by 2



(b) Leslie3d, training on inputs 21 by 21 by 2 and 31 by 31 by 2 and testing on 41 by 41 by 3



(c) Zeusmp, training on input 12 by 12 by 12 and testing on 12 by 12 by 12

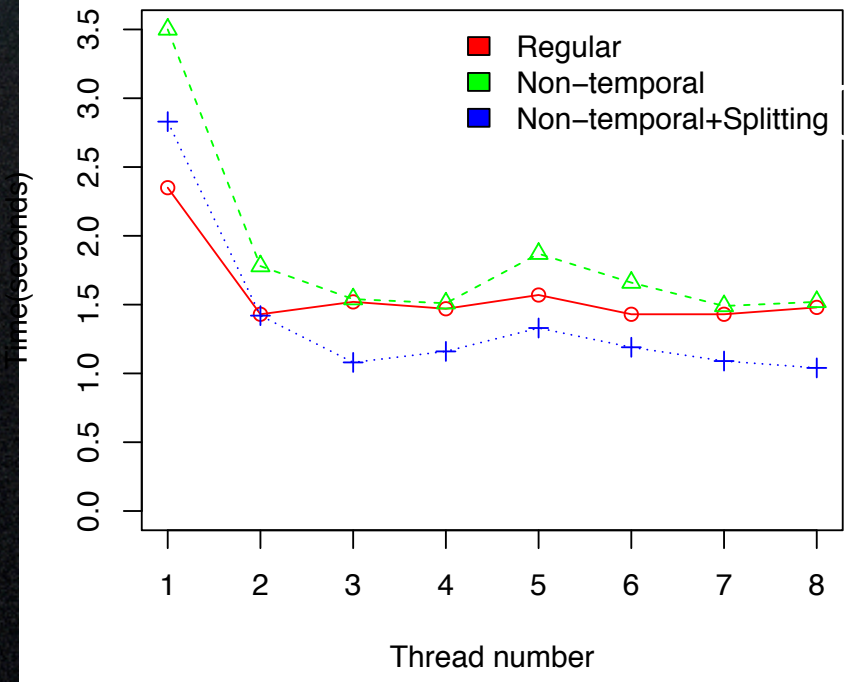


(d) Zeusmp, training on inputs 8 by 8 by 8 and 12 by 12 by 12 and testing on 16 by 16 by 16

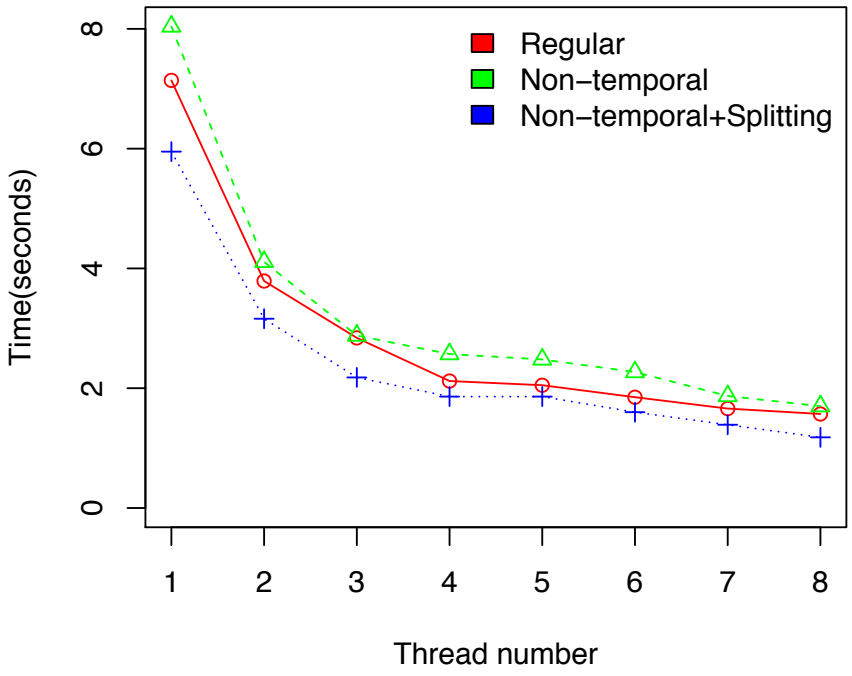
Implementation Results

$$
\begin{aligned}
&for(t = 0; t < MAXITER; t++)\\
&\#pragma\ omp\ parallel\ for\\
&\quad for(i = 0; i < N; i++)\\
&\quad\quad A[i] = foo(A[i]);
\end{aligned}
$$

Figure 8. An openmp example: the inner loop updates the array element by element; the outer loop corresponds to the time step



With Hardware Prefetching



Without Hardware Prefetching

Conclusions

- PACMAN works well with loop-based code.
- Fine grained: caching choice at every instruction.
- PACMAN is a good choice if training runs are an option.