SEMANTIC LOCALITY AND ITS USAGE IN MEMORY PREFETCHING

Leeor Peled¹, Uri Weiser¹, Shie Mannor¹, Yoav Etsion¹,²
Electrical Engineering¹ and Computer Science² Depts.
Technion
The premise

- Memory access streams often exhibit some regularity
  - Can be used for prefetching, smart caching, address prediction, task scheduling, etc. But how do we identify them?

- Current memory systems (caches, prefetchers) are most efficient with tempo-spatial locality

- However, many algorithms require linked data structures or contextual linking (e.g. linked data structures)
The cache-friendliness dilemma

- In many cases, performance drives programmers to layout data such that it benefits from spatial cache layout
  - Graphs as arrays or adj. matrices
  - Sparse matrix coding
- The result is non-intuitive, complex code
- Some structures are less amenable to cache-friendly restructuring

Can we capture semantic structure of data regardless of programming model and data structure layout?
Semantic vs. Spatial locality

- Grid structured memories are tuned for adjacent accesses
  - Consequently, we aim to map data to adjacent locations

- We are used to thinking of locality as a property of space
  i.e. Spatial Locality

- But data recurrence (locality) is inherent in program semantics and is also common in irregular data structures

- Spatial locality is a manifestation of Semantic Locality
Identifying program semantics

The following are *semantically* identical

```
for(i=0; i++; i<N)
    if(array[i].key==key)
        return array[i].val
```

```
while(node != NULL)
    if(node->key == key)
        return node->val;
    node = node->next;
```

But can we automatically identify semantics?
Approximate semantics with context

- Use machine context at the time of access to learn about the context of the access
  - Location of reference (IP)
  - Memory region (heap, stack)
  - Branch history
  - Performance monitors
  - …
Extract context from the code

- Programming languages (especially objective ones) have the power to tell us much about how we access memory
  - Pointers to strongly-typed data structures
  - Object sizes and internal fields organization
  - Location of reference (IP)
  - Different forms of access (e.g. in C: [], *, ->)
  - Recognizable pointer arithmetic

- Dynamic run-time observation can fill in the rest
Example: Context through compiler hints

- Some accesses are more important than others…

```c
while (node != NULL) {
    if (node->key == key) {
        return node->val;
        node = node->next;
        hint_correleation(node, node->next);
    }
}
```
Context-based access predictor

- Collecting all the context information for each access provides a massive quantity of data.
  - Need some technique to store it and find hidden patterns.

- **Ultimate goal:** recognize recurring links between any given context-state to future access, and learn how to produce context-based prediction.
  - Bind together block, stride and irregular data semantics

- **First cut:** focus on irregular data semantics
Training a context-based prefetcher: A sketch
Schematic flow

Access flow
\{Ctxt, addr\}
A, u
B, v
C, w
D, x
E, y
F, z
...

Last refs
Hkey_{n-11}
Hkey_{n-10}
....
Hkey_{current}

Context check

hash

Add JPs
Reset score

Predict

Collect

Reward

f_{Reward}(dist)

Hkey_n

PrefetchQ

W
predicting Hkey

Addr1
Score1
Addr2
Score2
Addr3
Score3

JP_THRESH
JP_DEPTH

lookup

hit

\text{predicting}
Predictor results (SW model – LearningPtr lib)

Predictor hit rates (prefetchQ distance)

% of lines (accumulated)

Distance between prediction and access

- MCF 300M skip
- SSCA_LDS(4,6)
- Clique(128)
- Clique(256)
- BST(32)
- BST(128)
- LL(Fwd=0)
- LL(Fwd=8)
Predictor results (SW model: LearningPtr lib)
Context prefetcher performance comparison
Layout-agnostic implementation
Conclusions (so far)

• Program semantics hold information about data correlation and association and can help performance

• …but automatically extracting program semantics is a big and open problem

• Machine state and compiler cues can help approximate data semantics

• Program/machine context, coupled with simple reinforcement learning, can help data prefetching

• We show that layout-agnostic data structures are feasible, but more research is needed before we can call it a success
QUESTIONS?

…and thank you…
Linked list lookahead potential

- Simple List traversal, Augmented using SW prefaches:
- Prefetch distance benefit limited according to locality method

- When prefetching mem-resident list through SW, hitting max potential at ~16 elements
  - This is a hard limit, reflecting shift from being latency-bound to BW-bound
PREV TEXT SLIDES
Research Track:
Semantic locality and context-based prefetching

[with Leeor Peled and Uri Weiser]
Machine Learning

• Investigating approaches similar to Contextual Multi-Armed Bandits.
  • The potential set of “actions” is as big as the effective memory space
    • Classic reinforced learning cannot be applied
  • Huge context space - hard to exploit similarities
  • Rewards are delays according to how far we attempt to predict

• Big question: how to implement in hardware…
Address prediction – basic algorithm

• For each memory access n:
  • Record current program state $X_n$
    • (dst_type, src_obj, access_type, ptr_offset, IP, IP-hist, addr-hist, delta, index, etc…)
  • For each m in the predictable depth range (e.g. [20-40]):
    • Push $\text{Addr}_n$ into the reference hash, with key = hash($X_{n-m}$)
    • Search entry at hash($X_{n-m}$) for the addresses with leading score, and issue them into the predictionQ
  • Match $\text{Addr}_n$ with predictionQ.
    • On hit, update the state that created the prediction in the reference hash according to the reward function
Additional Issues

• Hash reduction -
  • Prevent overfitting and slow learning due to huge state-space
  • 2-level hashing by using partial attributes, open more on overflow
  • In the future - dynamically select best attributes

• Drop off predictorQ -
  • Update if no match by that time

• Various options for back2back double address updating
  • Do we want to predict an address that was already predicted by another context?