SEMANTIC LOCALITY & CONTEXT BASED PREFETCHING

Leeor Peled¹, Shie Mannor¹, Uri Weiser¹, Yoav Etsion¹,²

¹Electrical Engineering and ²Computer Science
Technion – Israel Institute of Technology

42nd International Symposium on Computer Architecture (ISCA)
June 2015, Portland Oregon - http://dx.doi.org/10.1145/2749469.2749473
Agenda

• Origins of memory locality
• Semantic domain
• Machine learning approach
• Prefetcher scheme
  • Dynamic feature selection
  • Reward mechanism
• Evaluation
  • Layout agnostic programming
• Conclusions
Origins of Memory Locality

• Access patterns often exhibit forms of locality
  • Spatial: consecutive accesses are physically adjacent

• Potential sources:
  • Abstract algorithm level (data elements)
  • Software implementation (array cell, list node)
  • HW implementation (memory / cache line)

• Spatial locality is mostly an artifact of HW structure
  • Programmers often adapt to physical memory constraints

Can we identify locality at the algorithm level, regardless of implementation details?
Origins of Memory Locality - Example

- Insertions into a sorted linked list

What the CPU sees:

Access distribution over linear address space

What the programmer intended:

Access distribution over semantic space (index values)
Semantic Locality

• Higher abstraction level: Locality between program objects

• Accesses are semantically local if they are related through a sequence of actions
  • Dictated by program **semantics** (e.g.: \( \text{cur} = \text{cur} \rightarrow \text{next} \))

• Correlates actions that are **consequential**, *not necessarily consequent*
  • Not just temporal adjacency
Semantic Locality: Binary Tree

(a) Linked graph

```c
if (val == node->val) return node;
else if (val < node->val)
    node = node->left;
else
    node = node->right;
```

(b) Array

```c
if (val == array[index])
    return index;
else if (val < array[index])
    index = index * 2;
else
    index = index * 2 + 1;
```
Learning Semantic Locality

• How can we dynamically identify and use semantic locality?

• Collect program attributes:
  • **Compiler hints**
    • Type of variable, struct offsets
  • **Machine state**
    • Program counter, reference type, register data
  • **Machine history**
    • Access history, branch history

• Identify statistical correlations using **machine learning**
Machine Learning Approach

• Reinforcement learning model based on “Contextual Bandits” [Langford & Zhang, NIPS 2007]
  • **Associate** the state of the program and machine with future addresses
  • **Predict** “most likely” address per current state
  • Provide **Feedback** through usefulness metric

• The model balances *exploration* (learning by taking unknown actions), and *exploitation* (making use of existing knowledge)
Machine Learning Approach - Definitions

• Notation
  • **State**: vector of program and machine attributes
  • **Action**: predict and prefetch a given address
  • **Feedback**: reward prefetch operations based on their accuracy and timeliness

**Goal** - state / action pairs that are semantically related will accumulate a higher reward over time
Prefetcher Schematic Flow

- **History Queue**: Contains entries like Hash(A), Hash(B), and Hash(D) ordered by depth.
- **Context State Table (CST)**: Includes elements like Delta1 Score1, Delta2 Score2, Delta3 Score3.
- **Reducer**: Processes context attributes, partial attributes, and hash values.
- **Prefetch Queue**: Contains elements like w predicting CST index, z.
- **Feedback**:
  - Dynamic pref. BW control
  - Feedback to mem unit

Access flow: \{Ctx, addr\}
Dynamic Feature Selection

• Useful attributes are selected dynamically
  • Avoid overfitting or underfitting!
• State is rehashed after the filtering
Reward Mechanism

- Feedback is based on prediction timeliness and accuracy
- Bell-shape reward function accommodates varying depth

- Target distance:
  - L1 miss penalty = L2 lat. + L2 miss rate × DRAM latency
  - Prefetch distance = L1 miss penalty × IPC × P(mem.op.)
Evaluation

- Implemented using gem5
  - 4-way OOO
  - L1 cache: 32KB, 4 MSHRs
  - L2 cache: 2MB, 20 MSHRs
  - Prefetcher storage size: 30KB

- Compiled with a modified version of LLVM

- Compared to state of the art prefetchers:
  - Global History Buffer (GHB) G/DC and PC/DC
    [Nesbit & Smith, HPCA’04]
  - Spatial memory streaming (SMS)
    [Somogyi et al, ISCA’06]
Data Layout Agnostic Programming

• We implemented several algorithms with both linked data structures and arrays
• Our contextual prefetcher provides almost the same optimized performance
MPKI Comparison

Level 1 data cache (MPKI > 5)

Level 2 cache (MPKI > 1)
Storage Size Analysis

• CST and Reducer sizes reflect a tradeoff
  • Bigger tables:
    • Allow better learning history and more potential addresses
    • Reduce the chance of forgetting useful but infrequent associations
  • But also:
    • Increase learning time and harm convergence
Conclusions

• We argue that
  • Locality is a property of program semantics

• We show that
  • Program + machine attributes can successfully represent a semantic execution context
  • Machine learning can approximate semantic locality

• Future work: branch prediction, value prediction
Thank You
BACKUP FOILS
Semantic Locality: BFS traversal

(a) Linked graph

while (!Q.empty) {
  node = Q.pop();
  for (v:neighbours(node)) {
    if (!v->visited)
      Q.push(v);
  }
}

(b) Adjacency matrix

while (!Q.empty) {
  index = Q.pop();
  for (i: 1 .. N-1) {
    if (adjMat[index][i]>0 && !visited[i])
      Q.push(i);
  }
}

(c) Compressed sparse row (CSR)

while (!Q.empty) {
  index = Q.pop();
  for (j: RowIndex[i] .. RowIndex[i+1]-1) {
    if (!visited[Cols[j]])
      Q.push(Cols[j]);
  }
}
Prefetcher accuracy (bucketing)