Outline of Today’s Lecture

Today’s Paper:

Learning Memory Access Patterns

Milad Hashemi 1 Kevin Swersky 1 Jamie A. Smith 1 Grant Ayers 1* Heiner Litz 3* Jichuan Chang 1 Christos Kozyrakis 2 Parthasarathy Ranganathan 1

1. Prefetching Overview

2. LSTMs Overview

3. LSTMs for Prefetching

4. Evaluation

5. Lessons Learned
Outline of Today’s Lecture

Today’s Paper:

Learning Memory Access Patterns

Milad Hashemi 1  Kevin Swersky 1  Jamie A. Smith 1  Grant Ayers 2,4  Heiner Litz 3,4  Jichuan Chang 1  Christos Kozyrakis 2  Parthasarathy Ranganathan 1

1. Prefetching Overview

2. LSTMs Overview

3. LSTMs for Prefetching

4. Evaluation

5. Lessons Learned
Cache Hierarchy

Data is allocated in memory. When accessed from memory, it gets cached.

1. Cache Placement Policy
   e.g., direct-mapped

2. Cache Eviction Policy
   e.g., LRU, LFU

Memory Access Trace

- **0x40001ee0**: R 0xbfffe798
- **0x40001efd**: W 0xbfffe7d4
- **0x40001f09**: W 0xbfffe7d8
- **0x40001f20**: W 0xbfffe864
- **0x40001f20**: W 0xbfffe868

Program Counter (PC)

Access Type (Read or Write)

Memory Address of the data (hex number)
Data Prefetching

In addition to caching upon memory accesses, hardware prefetches data from memory into the cache, as well.

**What to Prefetch?**

Learn various patterns.
- Sequential: A, A+1, A+2...
- Strided: A, A+4, A+8...
- Correlated: A, B, C, A, D, B, A
- More complex ones..

**When to Prefetch?**

- On every data access.
  - Overheads?
- On every cache miss.
  - Patterns filtered by cache.
- On prefetching hits.

**Where to put the prefetched data?**

- In the cache.
- In separate buffers.
  - Avoid cache pollution.

It’s a prediction problem! Predict *which* data will be accessed in the future and cache them.
Next Line (1) + Stride Prefetchers (2)

Next Line Prefetcher

Prefetch data one after the other: A, A+1, A+2

Data in Memory

Stride Prefetcher

Matrix allocated in memory.

Column in matrix

Strided patterns are frequent.

- Need a mechanism to detect length of the stride.

Very easy to implement.

- Works only on sequential patterns.
Stream Prefetcher (3)

1. Cache miss A
2. Check buffer “heads”.

LRU replacement of buffers.

Buffers do not “pollute” the cache.
Extra hardware overhead (number and length of buffers).
Correlation Prefetcher (4)

Prefetches data based on history of memory accesses.

Memory Accesses: A B C D C A C D B C A

<table>
<thead>
<tr>
<th>Tag</th>
<th>1st time</th>
<th>2nd time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>C</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>A</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

*History Table*

Records the address that was next to “tag” the past 2 times.

Edges: Probability

Markov Graph

Captures variety of patterns.

Limited size, possible conflicts due to indexing.
Global History Buffer - GHB (5)

Decouples table indexes from the storage of prefetch-related history.

Memory Accesses: A B C D C A C D B C A

- Captures more complete history.
- Multiple table accesses.

Some version of GHB is mostly used in modern architectures!

Miss A - Prefetch C B
Evaluation Metrics

**Prediction**

- Positive / Prefetched
- Negative / Not Prefetched

**Real / Cache**

- Positive / Hit
- Negative / Miss

**Accuracy**

\[
\text{Accuracy} = \frac{\text{True Positives}}{(\text{True} + \text{False}) (\text{Positives} + \text{Negatives})}
\]

**Precision**

\[
\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}
\]

**Recall**

\[
\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}
\]

**Timeliness**

How early data is prefetched, versus when it is actually accessed, if at all.
Prefetching = Forecasting Time Series

Prefetching is a prediction problem = Forecasting future values of data that are ordered in time.

 Timestamp1 0x40001ee0: R 0xbfffe798
 Timestamp2 0x40001efd: W 0xbfffe7d4
 Timestamp3 0x40001f09: W 0xbfffe7d8
 Timestamp4 0x40001f20: W 0xbfffe864
 Timestamp5 0x40001f20: W 0xbfffe868

= Time series of accesses to memory addresses.

Sequential strides
Triangular Traversal
Sparse Tensors
Cycles of Randomness
Outline of Today’s Lecture

Today’s Paper:

Learning Memory Access Patterns

Milad Hashemi 1, Kevin Swersky 1, Jamie A. Smith 1, Grant Ayers 2,3, Heiner Litz 3,4, Jichuan Chang 1, Christos Kozyrakis 2, Parthasarathy Ranganathan 1

1. Prefetching Overview

2. LSTMs Overview

3. LSTMs for Prefetching

4. Evaluation

5. Lessons Learned

ML for Systems

Systems Software

Machine Learning

for Cache Prefetching
Long Short Term Memory (LSTM) networks are a type of Recurrent Neural Networks (RNNs) used for forecasting.

In the context of this lecture’s paper, LSTM is a solid box, no need to understand the internals.

We’ll focus on the inputs and outputs: what exactly it learns, what exactly it predicts.

... but since you’re curious let’s see it’s internal functionality.
ML for Forecasting Time Series

Long Short Term Memory (LSTM) networks are a type of Recurrent Neural Networks (RNNs) used for forecasting.

RNNs use information from many time steps $X_0, X_1 \ldots X_t$ to make a prediction $h_t$

E.g., the clouds are in the .. sky.

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/
ML for Forecasting Time Series

Long Short Term Memory (LSTM) networks are a type of Recurrent Neural Networks (RNNs) used for forecasting.

RNNs struggle to capture long-term dependencies.

E.g., I grew up in France, I speak fluent .. French.

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/
Long Short Term Memory (LSTM) networks are a type of Recurrent Neural Networks (RNNs) used for forecasting.

RNNs

1 internal layer

LSTMs

4 interacting internal layers

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/
Long Short Term Memory (LSTM) networks are a type of Recurrent Neural Networks (RNNs) used for forecasting.

Cell State $C_t$ can change through pointwise operations.

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/
Long Short Term Memory (LSTM) networks are a type of Recurrent Neural Networks (RNNs) used for forecasting.

\[ i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \]  
= “input gate” layer = which values to update.

\[ \tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \]  
= new candidate values to add to the cell state.

What new information are we storing in the cell state?

Source: [https://colah.github.io/posts/2015-08-Understanding-LSTMs/](https://colah.github.io/posts/2015-08-Understanding-LSTMs/)
Long Short Term Memory (LSTM) networks are a type of Recurrent Neural Networks (RNNs) used for forecasting.

Update the old state $C_{t-1}$ with the new one $C_t$.

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/
ML for Forecasting Time Series

Long Short Term Memory (LSTM) networks are a type of Recurrent Neural Networks (RNNs) used for forecasting.

\[
o_t = \sigma \left( W_o \left[ h_{t-1}, x_t \right] + b_o \right) = \text{“output gate” layer.}
\]

\[
h_t = o_t \times \tanh(C_t) = \text{push between -1..1, to output part of the cell state.}
\]

Output \(h_t\) is a filtered version of \(C_t\).

Source: [https://colah.github.io/posts/2015-08-Understanding-LSTMs/](https://colah.github.io/posts/2015-08-Understanding-LSTMs/)
Outline of Today’s Lecture

Today’s Paper:

Learning Memory Access Patterns

Milad Hashemi, Kevin Swersky, Jamie A. Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, Parthasarathy Ranganathan

1. Prefetching Overview
2. LSTMs Overview
3. LSTMs for Prefetching
4. Evaluation
5. Lessons Learned

ML for Systems

Systems Software

Machine Learning

for Cache Prefetching
Learning Memory Access Patterns

The data available for ML training is a *memory access trace*:

```
0x40001ee0: R 0xbfffe798
0x40001efd: W 0xbfffe7d4
0x40001f09: W 0xbfffe7d8
0x40001f20: W 0xbfffe864
0x40001f20: W 0xbfffe868
```

```plaintext
(0x40001f20, 0xbfffe864)  
(PC, Address) at time $t_N$
```

![Graph showing memory access patterns](image)

LSTM-based Deep Neural Network

```
output
0xbfffe868  
Address at time $t_{N+1}$
```

- Size of Trace: O(100M)
- Huge.. and extremely sparse. Only O(10M) unique addresses.
- Don’t learn address numbers!

64-bit binary number

Possible values = $2^{64}$

Normalizing that to [0, 1] leads to information loss.
Prefetching as Classification

Memory footprint is sparse means that a relatively **small**, and **consistent** set of addresses is used.

Learn address deltas, not raw addresses!

The number of uniquely occurring deltas is *often* orders of magnitude **smaller** than uniquely occurring addresses.

- **Input**: 
  - PC $t_N$, Address Delta($t_N$, $t_{N-1}$)

- **Output**: 
  - Address Delta($t_{N+1}$, $t_N$)

1. Go through the memory access trace.
2. Compute address deltas for every $(t_N, t_{N-1})$.
3. Keep the deltas that appear at least 10 times.
4. Create a “vocabulary” of these unique deltas.

Prefetching as Classification = Prediction will be one of these deltas.
Approach 1: Embedding LSTM

1. Input
PC $t_N$, Address Delta$(t_N, t_{N-1})$

2. Concatenated embeddings

3. Output
Vector length = X
$X = \text{number of unique deltas}$
1 such vector per delta.

“one hot encoding”

4. Prefetching Action
Prefetch the top-10 predictions, at each timestep $t_N$.

With Classification, the LSTM predicts probability for each of the X vectors.
Focus on *local* context, e.g., data structures are stored in contiguous memory address and accessed repeatedly.

1. Run k-means to cluster the addresses.
2. Deltas are computed within each cluster.

+ Smaller “vocabulary” of unique deltas.
- Potentially missing the “global” context.
Approach 2: Clustering + LSTM (2)

1. Train an LSTM per cluster of deltas.
2. Add cluster ID as an extra feature.
3. Tie weights.

+ Reduced model size, faster training.

- 1 extra pre-processing step for clustering.
Outline of Today’s Lecture

Today’s Paper:

Learning Memory Access Patterns

Milad Hashemi, Kevin Swersky, Jamie A. Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, Parthasarathy Ranganathan

1. Prefetching Overview
2. LSTMs Overview
3. LSTMs for Prefetching
4. Evaluation
5. Lessons Learned

Systems Software ML for Systems Machine Learning

for Cache Prefetching

LSTMs
Evaluation Metrics (1)

<table>
<thead>
<tr>
<th>Prediction</th>
<th>True Positive</th>
<th>False Positive</th>
<th>False Negative</th>
<th>True Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive / Hit</td>
<td>Positive</td>
<td></td>
<td>Negative</td>
<td></td>
</tr>
<tr>
<td>Negative / Miss</td>
<td>False</td>
<td>Positive</td>
<td>True</td>
<td>Negative</td>
</tr>
</tbody>
</table>

Real / Cache

Correct Prediction, if the *real* delta is one of the 10 predictions.

**Precision**

\[
\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}
\]

**Precision-at-10**

\[
\text{Precision-at-10} = \frac{\# \text{Correct Delta Predictions}}{\# \text{All Real Deltas}}
\]
### Evaluation Metrics (2)

<table>
<thead>
<tr>
<th>True Positive</th>
<th>False Positive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>False Negative</strong></td>
<td><strong>True Negative</strong></td>
</tr>
</tbody>
</table>

- **Recall**
  - Real / Cache
  - Positive / Hit
  - Negative / Miss

\[
\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}
\]

- **Recall-at-10**
  - # Unique Predicted Deltas
  - # All Predicted Deltas

\[
\text{Recall-at-10} = \frac{\# \text{Unique Predicted Deltas}}{\# \text{All Predicted Deltas}}
\]

Records all 10 predicted deltas.
Quantifies the % of the “vocabulary” that could be predicted.
Evaluation Baselines

- **Stream Prefetcher**
- **Global History Buffer (GHB)**
- **Embedding LSTM**
- **Clustering LSTM**
LSTM models achieve high precision, especially for complex patterns (e.g., websearch). No great difference between the embedding and clustering LSTM.

Remember.. They assume precision-at-10.
Evaluation

Stream prefetcher achieves highest recall, due to its dynamic vocabulary (set of deltas). Clustering LSTM better than embedding, because creates better vocabulary (set of deltas).
Sensitivity to Feature Selection

What happens when using only PC or Deltas as input features.

For precision, *only deltas* contributes the most.

For recall, *PC* helps improve it.
Outline of Today’s Lecture

Today’s Paper:

Learning Memory Access Patterns

Milad Hashemi 1  Kevin Swersky 1  Jamie A. Smith 1  Grant Ayers 1,2,4  Heiner Litz 1,3  Jichuan Chang 1  Christos Kozyrakis 2  Parthasarathy Ranganathan 1

1. Prefetching Overview
2. LSTMs Overview
3. LSTMs for Prefetching
4. Evaluation
5. Lessons Learned

ML for Systems
Software
Machine Learning

for Cache Prefetching
LSTMs
Lessons Learned (1)

What to remember when using LSTMs for prefetching.

Don’t Learn the Address, learn Address Deltas instead.

<table>
<thead>
<tr>
<th>Address</th>
<th>Access Type</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x40001ee0</td>
<td>R</td>
<td>0xbfffe798</td>
</tr>
<tr>
<td>0x40001efd</td>
<td>W</td>
<td>0xbfffe7d4</td>
</tr>
<tr>
<td>0x40001f09</td>
<td>W</td>
<td>0xbfffe7d8</td>
</tr>
<tr>
<td>0x40001f20</td>
<td>W</td>
<td>0xbfffe864</td>
</tr>
<tr>
<td>0x40001f20</td>
<td>W</td>
<td>0xbfffe868</td>
</tr>
</tbody>
</table>

Size of Trace: O(100M)

- Huge.. and extremely sparse. O(10M) unique addresses.
- Normalizing that to [0, 1] leads to information loss.

Record the most frequently seen

Address Delta \( (t_{N+1}, t_N) \)

Convert each unique delta to:

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 1 & 0 & 0 & 0
\end{array}
\]

“one hot encoding”

+ “Small” set of deltas.

Classification: predict specific values.
Lessons Learned (2)

What to remember when using LSTMs for prefetching.

Prefetching allows for multiple predictions, thus higher *perceived* model accuracy.

Prefetch the top-10 predictions, at each timestep $t_N$.

**Precision-at-10** = \(rac{\# \text{ Correct Delta Predictions}}{\# \text{ All Real Deltas}}\)

Correct Prediction, if the *real* delta is one of the 10 predictions.

LSTM models achieve much higher precision-at-10, not precision.

... and probably that’s why observe similar performance between the Embedding and Clustering LSTMs.
Lessons Learned (3)

What to remember when using LSTMs for prefetching.

The Clustering LSTM delivers higher recall, but similar precision to the Embedding LSTM.

The Embedding of (PC, Delta) deliver high precision due to the Deltas and high recall due to the PCs.
Report Due March 28 at 18.00

Answer / expand upon these 4 questions:

1. What problem is the paper addressing and why is it important?
2. How do they approach to solve the problem?
3. What are the main evaluation results?
4. What are 2 things you will remember from this paper?

Contact
• Via email: thaleia.doudali@imdea.org

Website
https://thaleia-dimitradoudali.github.io/