Efficient Approximations for Cache-Conscious Data Placement

Ali Ahmadi[1] Majid Daliri[2] Amir Goharshady[3] Andreas Pavlogiannis[4]

[1]Sharif University of Technology, Tehran, Iran
[2]University of Tehran, Tehran, Iran
[3]Hong Kong University of Science and Technology, Hong Kong, China
[4]Aarhus University, Aarhus, Denmark

PLDI 2022

Cache-conscious Data Placement (CDP)

- CDP is a classical problem in cache management introduced by Calder et al in ASPLOS 1998

Cache-conscious Data Placement (CDP)

- CDP is a classical problem in cache management introduced by Calder et al in ASPLOS 1998
- Consider a memory system consisting of
  - Main memory: huge but slow
  - Cache: small but fast

Cache-conscious Data Placement (CDP)

- CDP is a classical problem in cache management introduced by Calder et al in ASPLOS 1998
- Consider a memory system consisting of
  - Main memory: huge but slow
  - Cache: small but fast
- When a program wants to access an object o:
  - If o is in the cache, the access is successful

Cache-conscious Data Placement (CDP)

- CDP is a classical problem in cache management introduced by Calder et al in ASPLOS 1998
- Consider a memory system consisting of
  - Main memory: huge but slow
  - Cache: small but fast
- When a program wants to access an object o:
  - If o is in the cache, the access is successful
  - Otherwise, we have a ***cache miss*** and o has to be copied to the cache (possibly by evicting another item)

Cache-conscious Data Placement (CDP)

- CDP is a classical problem in cache management introduced by Calder et al in ASPLOS 1998
- Consider a memory system consisting of
  - Main memory: huge but slow
  - Cache: small but fast
- When a program wants to access an object o:
  - If o is in the cache, the access is successful
  - Otherwise, we have a ***cache miss*** and o has to be copied to the cache (possibly by evicting another item)
- Given a sequence Σ of accesses, our goal is to *minimize cache misses* over Σ

Details of CDP

- We have n objects (data items) $O = \{o_1, o_2, \ldots, o_n\}$ and a sequence $\Sigma \in O^N$ of accesses to these items

Details of CDP

- We have n objects (data items) $O = \{o_1, o_2, \ldots, o_n\}$ and a sequence $\Sigma \in O^N$ of accesses to these items
- The cache consists of k *lines* and is initially empty

Details of CDP

- We have n objects (data items) $O = \{o_1, o_2, \ldots, o_n\}$ and a sequence $\Sigma \in O^N$ of accesses to these items
- The cache consists of k *lines* and is initially empty
- Each line can hold up to t data items at a time
  - $t = 1$ in this talk, but our algorithm works for any t

Details of CDP

- We have n objects (data items) $O = \{o_1, o_2, \ldots, o_n\}$ and a sequence $\Sigma \in O^N$ of accesses to these items
- The cache consists of k *lines* and is initially empty
- Each line can hold up to t data items at a time
  - $t = 1$ in this talk, but our algorithm works for any t
- Each item o_i has a dedicated line $f(o_i)$ in the cache and is always copied to this line. We call f a *placement map*

Details of CDP

- We have n objects (data items) $O = \{o_1, o_2, \ldots, o_n\}$ and a sequence $\Sigma \in O^N$ of accesses to these items
- The cache consists of k *lines* and is initially empty
- Each line can hold up to t data items at a time
  - $t = 1$ in this talk, but our algorithm works for any t
- Each item o_i has a dedicated line $f(o_i)$ in the cache and is always copied to this line. We call f a *placement map*
- The goal is to find an optimal placement map f^* that minimizes the number of cache misses

Details of CDP

- We have n objects (data items) $O = \{o_1, o_2, \ldots, o_n\}$ and a sequence $\Sigma \in O^N$ of accesses to these items
- The cache consists of k *lines* and is initially empty
- Each line can hold up to t data items at a time
  - $t = 1$ in this talk, but our algorithm works for any t
- Each item o_i has a dedicated line $f(o_i)$ in the cache and is always copied to this line. We call f a *placement map*
- The goal is to find an optimal placement map f^* that minimizes the number of cache misses
- We analyze the runtime based on n and N and assume that t and k are small constants

Hardness of CDP

Theorem (Petrank and Rawitz, POPL 2002)

For a cache with more than two lines, CDP is not only NP-hard, but also hard to approximate within any non-trivial factor $O(N^{1-\epsilon})$ *unless P=NP*

Hardness of CDP

Theorem (Petrank and Rawitz, POPL 2002)

For a cache with more than two lines, CDP is not only NP-hard, but also hard to approximate within any non-trivial factor $O(N^{1-\epsilon})$ *unless P=NP*

- This is a very strong hardness result and it holds for the simple cache structure in CDP
- All following works are heuristics with no guarantees

Our Contribution

- The first positive theoretical results for CDP
  - based on sparsity of ***access graphs*** in real-world instances

Our Contribution

- The first positive theoretical results for CDP
  - based on sparsity of ***access graphs*** in real-world instances
- A *linear-time* $(1+\epsilon)$-approximation of the optimal number of cache misses assuming access graph of a specific degree d_ϵ is sparse
  - Sparser instances lead to better approximations

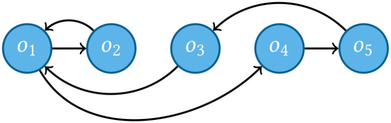
Our Contribution

- The first positive theoretical results for CDP
  - based on sparsity of ***access graphs*** in real-world instances
- A *linear-time* $(1+\epsilon)$-approximation of the optimal number of cache misses assuming access graph of a specific degree d_ϵ is sparse
  - Sparser instances lead to better approximations
- Stronger hardness results
  - Both approximation and parameterization (sparsity) are needed at the same time
  - Not covered in the talk (Please see the paper)

Our Contribution

- The first positive theoretical results for CDP
  - based on sparsity of ***access graphs*** in real-world instances
- A *linear-time* $(1+\epsilon)$-approximation of the optimal number of cache misses assuming access graph of a specific degree d_ϵ is sparse
  - Sparser instances lead to better approximations
- Stronger hardness results
  - Both approximation and parameterization (sparsity) are needed at the same time
  - Not covered in the talk (Please see the paper)
- Experimental Results
  - Not practical
    - Only applicable to small caches with a handful of lines
    - Due to exponential dependence on k
  - On these small caches, our approach beats the heuristics in 84-88% of the cases
  - Not covered in the talk (Please see the paper)

Access Graphs



$$\Sigma = \langle o_1, o_2, o_1, o_4, o_5, o_3, o_3, o_1, o_2 \rangle$$

Higher-order Access Hypergraphs

- **Basic Idea:** If we are accessing o_i and the previous access to o_i was far in the past, then it is very likely that the current access is a miss.

Higher-order Access Hypergraphs

- **Basic Idea:** If we are accessing o_i and the previous access to o_i was far in the past, then it is very likely that the current access is a miss.
- **Formalization:** If this is the first access to o_i or we have seen at least d distinct data items since the last access to o_i, then we assume a cache miss.

Higher-order Access Hypergraphs

- **Basic Idea:** If we are accessing o_i and the previous access to o_i was far in the past, then it is very likely that the current access is a miss.
- **Formalization:** If this is the first access to o_i or we have seen at least d distinct data items since the last access to o_i, then we assume a cache miss.

$$\Sigma = \langle o_1, o_2, o_1, o_4, o_5, o_3, o_3, o_1, o_2 \rangle$$

$$d = 3$$

Higher-order Access Hypergraphs

- **Basic Idea:** If we are accessing o_i and the previous access to o_i was far in the past, then it is very likely that the current access is a miss.
- **Formalization:** If this is the first access to o_i or we have seen at least d distinct data items since the last access to o_i, then we assume a cache miss.

$$\Sigma = \langle o_1, o_2, o_1, o_4, o_5, o_3, o_3, o_1, o_2 \rangle$$

$$d = 3$$

$$\begin{aligned} e_1 &= \langle o_1 \rangle & e_2 &= \langle o_1, o_2 \rangle & e_3 &= \langle o_1, o_2, o_1 \rangle \\ e_4 &= \langle o_2, o_1, o_4 \rangle & e_5 &= \langle o_1, o_4, o_5 \rangle & e_6 &= \langle o_4, o_5, o_3 \rangle \\ e_7 &= \langle o_3, o_3 \rangle & e_8 &= \langle o_5, o_3, o_3, o_1 \rangle & e_9 &= \langle o_3, o_1, o_2 \rangle \end{aligned}$$

Higher-order Access Hypergraphs

- **Basic Idea:** If we are accessing o_i and the previous access to o_i was far in the past, then it is very likely that the current access is a miss.
- **Formalization:** If this is the first access to o_i or we have seen at least d distinct data items since the last access to o_i, then we assume a cache miss.

$$\Sigma = \langle o_1, o_2, o_1, o_4, o_5, o_3, o_3, o_1, o_2 \rangle$$

$$d = 3$$

$$\begin{array}{rrr} \cancel{e_1 = \langle o_1 \rangle} & \cancel{e_2 = \langle o_1, o_2 \rangle} & e_3 = \langle o_1, o_2, o_1 \rangle \\ \cancel{e_4 = \langle o_2, o_1, o_4 \rangle} & \cancel{e_5 = \langle o_1, o_4, o_5 \rangle} & \cancel{e_6 = \langle o_4, o_5, o_3 \rangle} \\ e_7 = \langle o_3, o_3 \rangle & \cancel{e_8 = \langle o_5, o_3, o_3, o_1 \rangle} & \cancel{e_9 = \langle o_3, o_1, o_2 \rangle} \end{array}$$

Approximation Theorem

Theorem

For any $\epsilon > 0$, by applying the approach above using the sparsified access hypergraph $\tilde{G}_{d_\epsilon}$ of order $d_\epsilon := \lceil t \cdot k + \frac{t \cdot k}{\epsilon} \rceil$, we obtain a $(1+\epsilon)-$approximation of the optimal number of cache misses in a direct-mapped cache, i.e. $\mathit{Misses}_k(\hat{f}, \Sigma) \leq (1+\epsilon) \cdot \mathit{Misses}_k(f^*, \Sigma)$.

Graph Coloring

- Every edge in our hypergraph starts and ends with the same vertex

$$\begin{array}{rrr}
\cancel{e_1 = \langle o_1 \rangle} & \cancel{e_2 = \langle o_1, o_2 \rangle} & e_3 = \langle o_1, o_2, o_1 \rangle \\
\cancel{e_4 = \langle o_2, o_1, o_4 \rangle} & \cancel{e_5 = \langle o_1, o_4, o_5 \rangle} & \cancel{e_6 = \langle o_4, o_5, o_3 \rangle} \\
e_7 = \langle o_3, o_3 \rangle & \cancel{e_8 = \langle o_5, o_3, o_3, o_1 \rangle} & \cancel{e_9 = \langle o_3, o_1, o_2 \rangle}
\end{array}$$

Graph Coloring

- Every edge in our hypergraph starts and ends with the same vertex

$$\begin{array}{rrr} \cancel{e_1 = \langle o_1 \rangle} & \cancel{e_2 = \langle o_1, o_2 \rangle} & e_3 = \langle o_1, o_2, o_1 \rangle \\ \cancel{e_4 = \langle o_2, o_1, o_4 \rangle} & \cancel{e_5 = \langle o_1, o_4, o_5 \rangle} & \cancel{e_6 = \langle o_4, o_5, o_3 \rangle} \\ e_7 = \langle o_3, o_3 \rangle & \cancel{e_8 = \langle o_5, o_3, o_3, o_1 \rangle} & \cancel{e_9 = \langle o_3, o_1, o_2 \rangle} \end{array}$$

- A placement map is basically a coloring of our hypergraph with k colors

Graph Coloring

- Every edge in our hypergraph starts and ends with the same vertex

$$
\begin{array}{rrr}
\cancel{e_1 = \langle o_1 \rangle} & \cancel{e_2 = \langle o_1, o_2 \rangle} & e_3 = \langle o_1, o_2, o_1 \rangle \\
\cancel{e_4 = \langle o_2, o_1, o_4 \rangle} & \cancel{e_5 = \langle o_1, o_4, o_5 \rangle} & \cancel{e_6 = \langle o_4, o_5, o_3 \rangle} \\
e_7 = \langle o_3, o_3 \rangle & \cancel{e_8 = \langle o_5, o_3, o_3, o_1 \rangle} & \cancel{e_9 = \langle o_3, o_1, o_2 \rangle}
\end{array}
$$

- A placement map is basically a coloring of our hypergraph with k colors
- An edge corresponds to a cache miss if the color assigned to its endpoints is reused in its middle, as well.

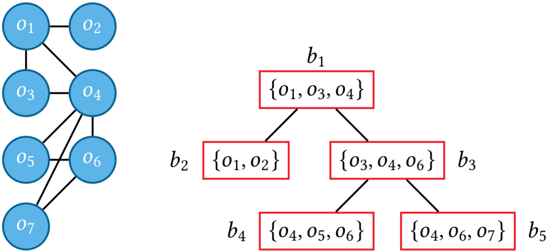
Graph Coloring

- Every edge in our hypergraph starts and ends with the same vertex

$$
\begin{aligned}
&\cancel{e_1 = \langle o_1 \rangle} && \cancel{e_2 = \langle o_1, o_2 \rangle} && e_3 = \langle o_1, o_2, o_1 \rangle \\
&\cancel{e_4 = \langle o_2, o_1, o_4 \rangle} && \cancel{e_5 = \langle o_1, o_4, o_5 \rangle} && \cancel{e_6 = \langle o_4, o_5, o_3 \rangle} \\
&e_7 = \langle o_3, o_3 \rangle && \cancel{e_8 = \langle o_5, o_3, o_3, o_1 \rangle} && \cancel{e_9 = \langle o_3, o_1, o_2 \rangle}
\end{aligned}
$$

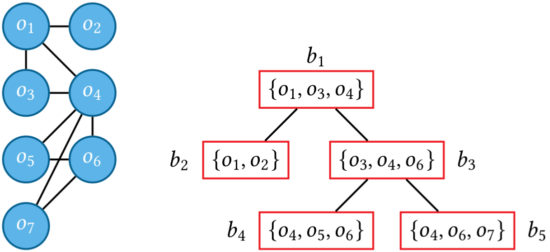
- A placement map is basically a coloring of our hypergraph with k colors
- An edge corresponds to a cache miss if the color assigned to its endpoints is reused in its middle, as well.
- NP-hard Problem: Find a coloring that minimizes missed edges.

Treewidth-based Dynamic Programming



- We assume our sparsified access graphs have small treewidth
  - In reality, they do [Chatterjee et al, POPL 2019]

Treewidth-based Dynamic Programming



- We assume our sparsified access graphs have small treewidth
  - In reality, they do [Chatterjee et al, POPL 2019]
- Do a linear-time bottom-up dynamic programming as if you are coloring a tree

$$\texttt{dp}[b_i, \text{partial coloring } c] =$$

$$\text{minimum number of missed edges in the subtree of } b_i$$

$$\text{if we color the vertices in } b_i \text{ according to } c$$

Conclusion

- CDP is really hard, even when the sequence of accesses is given apriori
- It is not as hard as previously thought since real-world instances are sparse
- The sparsity (tree-likeness) can be exploited to obtain $(1+\epsilon)$-approximations for any $\epsilon > 0$
- CDP requires both approximation and parameterization (by the treewidth of access graphs) to become tractable
- We provided the first positive theoretical result for CDP but there is still a long way until it becomes practical
- The current algorithm can be used for limit studies and comparison of heuristics