Rethinking Self-balancing Binary Search Tree over Phase Change Memory with Write Asymmetry

ASP-DAC 2018

Chieh-Fu Chang, Che-Wei Chang, Yuan-Hao Chang, and Ming-Chang Yang

Chieh-Fu Chang
Research Assistant, Institute of Information Science, Academia Sinica, Taiwan
Outline

• Introduction

• Background and Motivation

• Write-Asymmetry-Aware Self-Balance Tree
  – Basic Concepts
  – Analysis of Tree Rotations
  – Depth-First-Alternating Traversal
  – Address Conflict Manager

• Evaluation
  – Experimental Setup
  – Experimental Results

• Conclusion
The Needs of Huge Memory/Storage

• Big data and data mining applications require huge storage
  – Store huge amount of data
  – In-memory computing

• DRAM and NAND Flash are hitting the wall of its transistor scaling*
  – Density limitation
  – High power consumption
    - Leakage power
    - Operation power

Non-volatile Memory

• Phase-change memory and 3D Xpoint
  – High density
  – Low leakage power
  – Non-volatility

• Features of 3D Xpoint
  – 1000X faster than NAND
  – 1000X more endurance of NAND
  – 10X higher density than DRAM
Outline

• Introduction

• Background and Motivation

• Write-Asymmetry-Aware Self-Balance Tree
  – Basic Concepts
  – Analysis of Tree Rotations
  – Depth-First-Alternating Traversal
  – Address Conflict Manager

• Evaluation
  – Experimental Setup
  – Experimental Results

• Conclusion
Phase Change Memory (PCM)

• Physical structure and mechanisms
  – Two phases :
    - The amorphous phase with high resistance
    - The crystalline phase with lower resistance

• Advantages
  – High density
  – Non-volatility
  – Low power consumption
  – Outstanding I/O performance
  – Byte addressability
Issues of Using PCM

• Write Asymmetry
  – Reset
    - High instant power with short time
  – Set
    - Low power with long time

• Write Latency

• Write Endurance

• Suggested Solution?
  – Reducing #write bits

---

Relative Works

• Data-Comparison Write (DCW)
  – Read the old (stored) data.
  – Do comparison with the new data.
  – Skip any bit write if it is not needed.

• Coset Coding
  – Provide a one-to-many mapping for each data word to a (co)set of vectors.
  – Choose the vector with the minimum overhead for each write.
Motivation

• Big/massive data applications demand extremely large main memory space for better performance.
• PCM has low leakage power and high density which make it a promising candidate to replace DRAM.
• Write endurance and latency are critical for using PCM.
• Existing studies improve the write mechanism to handle the given write patterns on PCM.

Why don’t we fundamentally generate more suitable write patterns for PCM
  • By improving address allocation for data structures
Outline

- Introduction
- Background and Motivation
- Write-Asymmetry-Aware Self-Balance Tree
  - Basic Concepts
  - Analysis of Tree Rotations
  - Depth-First-Alternating Traversal
  - Address Conflict Manager
- Evaluation
  - Experimental Setup
  - Experimental Results
- Conclusion
AVL Tree

• The properties of an AVL tree (for n data nodes)
  – The average and worst-case space utilization: $O(n)$
  – The average and worst-case time complexity of tree search, insertion, update and deletion: $O(\log n)$

• The expense for having the above properties
  – **Tree Rotations**
    - Conducting a rotation if the height difference of the left and right sub-trees is more than one level.
  – **Multiple Update Writes**
    - For each rotation, there are multiple update writes of pointers.
    - It could exacerbate the endurance and latency issues on PCM.
An Overview of Our Design

• Tree rotation analysis is conducted to better understand the relation among nodes.

• Our DFAT algorithm is developed to find the node relation path with the consideration of possible tree rotations.
  – The Gray code technique is leveraged to minimize the distance of two given address values in the address sequence that we use to map to our node relation path.
  – An address conflict manager is proposed to resolve possible address conflicts caused by rotations.
Outline

• Introduction

• Background and Motivation

• Write-Asymmetry-Aware Self-Balance Tree
  – Basic Concepts
  – Analysis of Tree Rotations
  – Depth-First-Alternating Traversal
  – Address Conflict Manager

• Evaluation
  – Experimental Setup
  – Experimental Results

• Conclusion
Four Types of AVL Tree Rotations

LL Rotation

LR Rotation

RR Rotation

RL Rotation
Our Idea: Assign nodes B & D with close addresses!
Relation Paths of Tree Nodes
Outline

• Introduction

• Background and Motivation

• Write-Asymmetry-Aware Self-Balance Tree
  – Basic Concepts
  – Analysis of Tree Rotations
  – Depth-First-Alternating Traversal
  – Address Conflict Manager

• Evaluation
  – Experimental Setup
  – Experimental Results

• Conclusion
Depth-First-Alternating Traversal (DFAT)

- A systematic approach for indexing all nodes, where nodes having stronger relations will be assigned closer indexes.
Leveraging Gray Code on DFAT

- **Gray code**: An ordering of the binary numeral system such that two successive values have the shortest distance (differ in only one bit).

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gray Code</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td>1101</td>
<td>1110</td>
</tr>
</tbody>
</table>
A Running Example Of Our Solution

AVL tree

Before RR Rotation

& B \rightarrow & D

After RR Rotation

<table>
<thead>
<tr>
<th>Key value</th>
<th>Binary code address</th>
<th>Gray code address</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>011111111111111101</td>
<td>010000000000000011</td>
</tr>
<tr>
<td>C</td>
<td>01111111111111110</td>
<td>01000000000000001</td>
</tr>
<tr>
<td>B</td>
<td>01111111111111111</td>
<td>01000000000000000</td>
</tr>
<tr>
<td>D</td>
<td>10000000000000000</td>
<td>11000000000000000</td>
</tr>
<tr>
<td>E</td>
<td>10000000000000001</td>
<td>11000000000000001</td>
</tr>
</tbody>
</table>
Outline

• Introduction

• Background and Motivation

• Write-Asymmetry-Aware Self-Balance Tree
  – Basic Concepts
  – Analysis of Tree Rotations
  – Depth-First-Alternating Traversal
  – Address Conflict Manager

• Evaluation
  – Experimental Setup
  – Experimental Results

• Conclusion
Node Address Collisions

• The problem of address collisions
  – The address of a to-be-inserted node might be used because there are rotations.

• Our address conflict manager
  – Build a redundant queue for keeping some addresses.
  – Put free and can-not-be-used (by DFAT) addresses into the redundant queue after a tree rotation.
  – Select one free address from the redundant queue when there is an address collision during an insertion.
Outline

• Introduction
• Background and Motivation
• Write-Asymmetry-Aware Self-Balance Tree
  – Basic Concepts
  – Analysis of Tree Rotations
  – Depth-First-Alternating Traversal
  – Address Conflict Manager
• Evaluation
  – Experimental Setup
  – Experimental Results
• Conclusion
Experimental Setup (1/2)

• The experiment was conducted in a simulator we made to evaluate the proposed solution with different numbers of data items and different sizes of address space.

• The random permutation of the data array is used as the input data, and we sequentially insert the data items into an AVL tree.

• For the input size of N nodes, we reserve memory space for an AVL tree of $L + 2$ levels, where $L = \lceil \log_2(N + 1) \rceil$.
Experimental Setup (2/2)

- We assign an address to each to-be-inserted node by the following solutions
  - *Random*: randomly selects an available address.
  - *Linear*: sequentially selects an available address, and the address value starts at 0.
  - *Gray*: uses the original tree indexes and leverages the Gray code technique to assign an address for each node.
  - *DFAT-Gray*: is our solution which indexes all nodes by the DFAT algorithm and leverages the Gray code technique to assign an address for each node.
Outline

- Introduction
- Background and Motivation
- Write-Asymmetry-Aware Self-Balance Tree
  - Basic Concepts
  - Analysis of Tree Rotations
  - Depth-First-Alternating Traversal
  - Address Conflict Manager
- Evaluation
  - Experimental Setup
  - Experimental Results
- Conclusion
Result of Solutions with Different Data Sizes (1/2)

- *Linear*: is the best solution when the memory space $\leq 2^{10}$.

- *DFAT-Gray*: outperforms the other solutions when the memory space $> 2^{10}$.
Result of Solutions with Different Data Sizes (2/2)

- The percentage of bit flips = \[
\frac{\text{number of bit flips per write}}{\text{number of bits in a pointer}}
\]

- When the number of bits in a pointer increases
  - The bit-flip ratios of \textit{Linear} and \textit{Gray} increase.
  - The bit-flip ratio of \textit{DFAT-Gray} decreases noticeably.

![Graph showing the percentage of bit flips vs. number of bits in a pointer](image)
We consider \(2^{18} - 1 < \text{memory space} < 2^{36} - 1\), but the data size is fixed at \(2^{16}\) for fast simulation.

\textit{Random} significantly increases the number of bit flips per change along with the memory space getting larger.
Result of Simulation for Huge Memory (2/2)

- **Gray** and **Linear**: have a stable performance with the fixed input data size
- **DFAT-Gray**: increases its bit flips ratio slightly when the memory space increases from $2^{18}$ to $2^{22}$, and then gets saturated.

![Chart showing the number of bit flips per change against the number of bits in a pointer.](chart.png)
Outline

• Introduction

• Background and Motivation

• Write-Asymmetry-Aware Self-Balance Tree
  – Basic Concepts
  – Analysis of Tree Rotations
  – Depth-First-Alternating Traversal
  – Address Conflict Manager

• Evaluation
  – Experimental Setup
  – Experimental Results

• Conclusion
Conclusion

• We redesign the memory allocation scheme of a self-balancing binary search tree for PCM.

• *DFAT-Gray* on AVL trees can reduce more than 15% of bit flips when the size of the input data is more than $2^{15} - 1$ nodes.

• Further extending our concepts to other relative data structures is our ongoing studies.
Thanks for Listening!
Q&A
Appendix

- Solutions
  - DFAT
  - Gray

- Collision Issue
  - Redundant Queue

- Experiment Setup
  - Running Environment

- Exceeded Tree Nodes
Solution - DFAT

- Traversal start from the Root
  - Turn left for the step 1.
  - After step 1, choose a direction opposited to the previous way.
Analysis of Time Complexity

if $\alpha \leq \text{the key of the root}$ then
\hspace{1cm} The current direction $D \leftarrow left$; The flag $F \leftarrow -1$;
else
\hspace{1cm} The current direction $D \leftarrow right$; The flag $F \leftarrow 1$;
do
\hspace{2cm} if $\alpha \leq \text{the key of the current node}$ then
\hspace{3cm} if $D = left$ then
\hspace{4cm} $I \leftarrow I + F$;
\hspace{3cm} else
\hspace{4cm} $I \leftarrow I + F(2^{(H-L)})$, where $L$ is the current level;
\hspace{3cm} Change the direction $D$;
\hspace{3cm} Go to the left child of the current node;
\hspace{2cm} else
\hspace{3cm} if $D = left$ then
\hspace{4cm} $I \leftarrow I + F(2^{(H-L)})$, where $L$ is the current level;
\hspace{3cm} else
\hspace{4cm} $I \leftarrow I + F$;
\hspace{3cm} Change the direction $D$;
\hspace{3cm} Go to the right child of the current node;
while the current node is not a leaf node;
return the current position and index $I$;
Solution - Gray

• Simply links tree nodes by the order of original tree index to build the path.

• A solution which leverages Gray code without DFAT algorithm.
Experiment Setup

• Running Environment
  – Visual Studio VC++ 2017 v141
  – Microsoft Windows 10 x64
  – Intel i5-2400 processor
  – 16 GB DRAM
Collision Issue

• The target address already taken by previous inserted node which was switched to other space after tree rotation

• Redundant queue
  – After tree rotations, put all unoccupied node addresses into the redundant queue.
  – Select one node address from the queue when collisions occur
  – In this way, we can avoid any node collisions caused by the way of using greedy searching algorithm
Exceeded Inserted Nodes

- When a depth of a inserted node is deeper than the given depth of tree map (before a tree rotation)
  - This kind of situation would only happen at bottom of the tree, and most of the tree spaces at bottom are barely used
  - Solution
    - Allocate a address after rotation if the node will rotate back into the allocated space.
    - If not, directly allocate a valid node which is close to it’s parent’s node.