IMPLEMENTATION OF HIERARCHICAL TEMPORAL MEMORY ON A MANY-CORE ARCHITECTURE

Sheenam Jayaswal & Ziyang Liu
AGENDA

• INTRODUCTION
• HTM PRINCIPLES AND CORE FUNCTIONS
• LEARNING AND PREDICTION IN HTM
• SPATIAL POOLER
• TEMPORAL POOLER
• HTM IMPLEMENTATION OF SPATIAL AND TEMPORAL POOLER
• MAPPING METHODS USED FOR PARALLELIZATION
• RESULTS
• CONCLUSIONS AND FUTURE WORK
INTRODUCTION

• HTM is a bio-inspired, significantly complex machine learning model developed by Dileep George and Jeff Hawkins of Numenta.

• HTM has promising applications in pattern recognition and inference.

• Parallel implementation of HTM on the proposed many-core platform has been done in C.

• Block-based, row-based and column-based parallelization schemes were used for the parallelization of HTM.

• Speedup, efficiency and scalability have been measured while performing pattern recognition tasks.
HTM

• HTM is an ANN which aims to capture the structural and functional properties of the neocortex.
• Significantly better than classic ANN models owing to its ability to recognize and infer patterns.
• Better than most pattern recognition algorithms because of its ability to learn spatial and temporal sequences.
• Applications include artificial intelligence, machine learning, pattern recognition, navigation and data mining.
• Very different from traditional computers due to its hierarchical memory organization and dependence on time.
HTM PRINCIPLES

• HIERARCHY
  • Regions are arranged hierarchically in an HTM network.
  • Convergence as one ascends the hierarchy.
  • Divergence as one descends the hierarchy due to feedback.

• REGIONS
  • Modelled after the neocortex which consists of interconnected regions.
  • Not all regions receive direct inputs.
  • Regions contain a set of interconnected cells arranged in columns.

• SPARSE DISTRIBUTED REPRESENTATIONS
  • Sparse- Only a set of neurons are active at any given time due to the property of inhibition.
  • Distributed - Multiple neurons are required to represent some information.

• TIME
  • Visual, auditory and sensory inference requires the presence of time-changing inputs.

Fig: 4 level hierarchy with 4 regions
HTM CORE FUNCTIONS

• LEARNING
  • Uses the principle of ‘on-line’ learning for its region.
  • Each HTM region looks for spatial patterns then learns temporal patterns.
  • The complexity of spatial patterns learned by a region depends on how much memory is allocated to this region

• INFERENCEx
  • Matching of inputs to previously learnt spatial and temporal patterns.
  • No two inputs are exactly the same. Thus, only a portion of the pattern is matched with stored sequences

• PREDICTION
  • Based on what has occurred in the past as well as present inputs.
  • Variable order memory: HTM learns to use as much prior context as needed.
LEARNING AND PREDICTION IN HTM

- Spatial Pooler works on the shared proximal dendrite segment at the level of columns to learn connections between input bits and columns.
- Temporal Pooler works on distal dendrite segments at the level of cells to learn feed-forward connections between cells in the same region.
BACKGROUND

- Cells receive input via proximal dendrite segments.
- Each column of cells has a single proximal dendrite segment.
- Each proximal dendrite has a set of associated potential synapses.
- Lateral input from nearby cells is received via distal dendrites.
- Potential synapses have a permanence value between 0 to 1. They are active if the permanence is above the threshold.
- Column is active when the number of valid synapses is above a certain threshold.
SPATIAL POOLER

Phase 1
- Learns about connections between synapses and inputs.
- Determines how many synapses are connected to valid inputs.

Phase 2
- Multiplies the number of active synapses by the boosting factor.
- Columns with stronger activation inhibit columns with weaker activation in the neighborhood.

Phase 3
- Permanence values of all potential synapses are updated.
- Permanence values of synapses connected to active inputs to be increased.
Phase 1
• Cells activated by feed forward input become active.

Phase 2
• Cells activated by lateral input enter predictive state.

Phase 3
• Synapses are updated for learning.
HARDWARE FOR IMPLEMENTATION

- Adepteva Epiphany is used for the hardware implementation.
- Scalable many-core architecture that uses a shared-memory model.
- Interconnections done via eMesh network which are more power efficient than traditional crossbars.
- Each node consists of an eCore RISC CPU, multicore-optimized Direct Memory Access (DMA) engine, multi-bank local memory, event timer and network interface for all nodes.
In the given implementation, 1 region contains 16x16 columns which contain 4 cells each since the size of the training image is 16x16 pixels.

Single HTM level has been implemented in the project.

Each cell has set 7 distal dendrite segments, each of which has 20 synapses.

Lot of ‘for’ loops exist in the code to enable easier parallelization.
SPATIAL POOLER IMPLEMENTATION

- Phase 1 - Compute Overlap
- Phase 2 - Compute Winning columns
- Phase 3 - Update permanence
- Columns activated via feed-forward input. Active columns marked in blue
TEMPORAL POOLER IMPLEMENTATION

- Cells within active columns activated by feed-forward input or lateral input.
- Active cells represented in blue.
- Cells activated by feed-forward input become active.
- Cells activated by lateral input enter predictive state.
TRAINING SETS

- Used to test the efficacy of the algorithm.
- Small and large training sets used.
- Large training set has 416 training examples generated from a camera sweep across the 26 alphabets.
- The algorithm is supposed to predict the next alphabet after being given one alphabet as input.

![Small Training Set](image1)

![Example of Large Training Set](image2)
### SEQUENTIAL IMPLEMENTATION

<table>
<thead>
<tr>
<th>TRAINING STEP</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Step 1</td>
<td>Previous state of the cell will be assigned current value and the current state will be reset.</td>
</tr>
<tr>
<td>Training Step 2</td>
<td>Set input value</td>
</tr>
<tr>
<td>Training Step 3</td>
<td>Initialize columns</td>
</tr>
<tr>
<td>Training Step 4</td>
<td>Phase 1 of spatial pooler implementation</td>
</tr>
<tr>
<td>Training Step 5</td>
<td>Phase 2 of spatial pooler implementation</td>
</tr>
<tr>
<td>Training Step 6</td>
<td>Phase 1 of temporal pooler implementation</td>
</tr>
<tr>
<td>Training Step 7</td>
<td>Phase 2 of temporal pooler implementation</td>
</tr>
<tr>
<td>Training Step 8</td>
<td>Phase 3 of temporal pooler implementation</td>
</tr>
</tbody>
</table>
MAPPING METHODS USED FOR PARALLELIZATION

- Column-Based Mapping method
- Row-Based Mapping method
- Block-Based Mapping method
COMMUNICATION AND SYNCHRONIZATION

- All cores can read data simultaneously from shared memory but only one can write at a given time.
RESULTS ANALYSIS

- Result and Analysis of the Experiment with the Small Training Set
  - Row-based wins in this experiment.
  - Input pattern leads to imbalanced load.

<table>
<thead>
<tr>
<th>Mapping Method</th>
<th>Max clock cycles</th>
<th>Speedup</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential implementation</td>
<td>141,349,367</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>Block-Based</td>
<td>19,319,505</td>
<td>7.3164</td>
<td>0.4573</td>
</tr>
<tr>
<td>Column-Based</td>
<td>25,949,537</td>
<td>5.4471</td>
<td>0.3404</td>
</tr>
<tr>
<td>Row-Based</td>
<td>10,175,309</td>
<td>13.8914</td>
<td>0.8682</td>
</tr>
</tbody>
</table>
**RESULTS ANALYSIS**

- Result and Analysis of the Experiment with the Full Training Set
  - Three mapping methods have similar performances, but Row-based is better.
  - In large size of training set, the variability is not obvious anymore.
  - The phase 2 of temporal pooling becomes the dominant part of execution time.

<table>
<thead>
<tr>
<th>Block-Based</th>
<th>Row-based</th>
<th>Column-Based</th>
</tr>
</thead>
<tbody>
<tr>
<td>The number of cores</td>
<td>Total Execution time (Minutes)</td>
<td>Total Speedup</td>
</tr>
<tr>
<td>1 core</td>
<td>448.0657</td>
<td>1.0000</td>
</tr>
<tr>
<td>2 cores</td>
<td>228.0833</td>
<td>1.9993</td>
</tr>
<tr>
<td>4 cores</td>
<td>118.0667</td>
<td>3.7851</td>
</tr>
<tr>
<td>8 cores</td>
<td>62.9583</td>
<td>7.1169</td>
</tr>
<tr>
<td>16 cores</td>
<td>32.5833</td>
<td>13.6770</td>
</tr>
<tr>
<td>1 core</td>
<td>448.0657</td>
<td>1.0000</td>
</tr>
<tr>
<td>2 cores</td>
<td>228.0917</td>
<td>1.9644</td>
</tr>
<tr>
<td>4 cores</td>
<td>118.0667</td>
<td>3.8356</td>
</tr>
<tr>
<td>8 cores</td>
<td>62.9583</td>
<td>7.2219</td>
</tr>
<tr>
<td>16 cores</td>
<td>32.5833</td>
<td>13.6770</td>
</tr>
<tr>
<td>1 core</td>
<td>448.0657</td>
<td>1.0000</td>
</tr>
<tr>
<td>2 cores</td>
<td>228.0917</td>
<td>1.9644</td>
</tr>
<tr>
<td>4 cores</td>
<td>118.0667</td>
<td>3.8356</td>
</tr>
<tr>
<td>8 cores</td>
<td>62.9583</td>
<td>7.2219</td>
</tr>
<tr>
<td>16 cores</td>
<td>32.5833</td>
<td>13.6770</td>
</tr>
</tbody>
</table>
CONCLUSIONS AND FUTURE WORK

• The HTM algorithm was implemented sequentially as well as in parallel with 16 available cores.
• For the small training set, row-based mapping was found to be most effective.
• For the larger training set, all mapping methods were nearly identical.
• Future work can include construction of a multi-layered HTM network.
• Testing complex RGB images can help verify correctness.
• Parallelization at the level of dendrites or synapses could be evaluated.
• Implementation of HTM on FPGAs and GPUs could be done.
QUESTIONS?