Parallel error-correcting output codes classification in volume visualization: parallelism for AI and AI for parallelism

Oscar Amorós Huguet
Advisors: Sergio Escalera, Anna Puig
UPC-UB-URV
Introduction

● Main Goal: explore AI and parallelism interaction.

● Contributions:
  o New parallel programming tools (AIMparallel and SimpleOpenCL)
  o Semi automatic classification:
    ▪ Framework for 3D scan medical images
    ▪ Parallel implementation of the framework (Parallelism for AI)
  o A new parallel system proposal based on SOMAS (AI for parallelism)
Methodology and nomenclature.

Overheads:

- **TCO**: cycles lost due to data movement
- **WPO**: cycles lost due to load unbalancing
- **TMO**: cycles lost due to thread management
AIMparallel

- Easy to use
- Avoiding the worst implementation
- More useful with complex algorithms
- For fine tuning need to profile

Example

<table>
<thead>
<tr>
<th>OP1</th>
<th>OP2</th>
<th>System GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>aTCO &gt;&gt; aTCO</td>
<td>sTCO</td>
<td>^</td>
</tr>
<tr>
<td>aWPO &lt; aWPO</td>
<td>sWPO</td>
<td>✓</td>
</tr>
<tr>
<td>aTMO = aTMO</td>
<td>sTMO</td>
<td>✓</td>
</tr>
</tbody>
</table>
Classification problem

A semi automatic classification system
C.P.: proposal ECOC

Combines binary classifiers \((h)\) to create a multiclass \((y)\) classifier

![Matrix and HD diagram for ECOC](image)
C.P.: proposal Adaboost

**ALGORITHM 1:** Discrete Adaboost training algorithm.

1. Start with weights \( w_i = 1/k, i = 1, \ldots, k \).
2. Repeat for \( m = 1, 2, \ldots, (M) \):
   - (a) Fit the classifier \( f_m(\rho) \in \{-1, 1\} \) using weights \( w_i \) on the training data.
   - (b) Compute \( \text{err}_m = \mathbb{E}_w[1_{\{l(\rho) \neq f_m(\rho)\}}], c_m = \log((1 - \text{err}_m)/\text{err}_m). \)
   - (c) Set \( w_i \leftarrow w_i \exp[c_m \cdot 1_{\{l(\rho_i) \neq f_m(\rho_i)\}}], i = 1, 2, \ldots, k \), and normalize so that \( \sum_i w_i = 1 \).
3. Output the classifier \( F(\rho) = \text{sign}[\sum_{m=1}^{M} c_m f_m(\rho)] \).

**ALGORITHM 2:** Discrete Adaboost testing algorithm.

1. Given a test sample \( \rho \)
2. \( F(\rho) = 0 \)
3. Repeat for \( m = 1, 2, \ldots, M \):
   - (a) \( F(\rho) = F(\rho) + c_m (P_m \cdot \rho^m < P_m \cdot T_m) \);
4. Output \( \text{sign}(F(\rho)) \)
C.P.: proposal ECOC submatrix
C.P.: adaptive decoding

**Loss-Weighted strategy:** Given a coding matrix $M$,

1) Calculate the performance matrix $H$,

$$H(i, j) = \frac{1}{m_i} \sum_{k=1}^{m_i} \varphi(h^j(\rho^i_k), i, j)$$  \hspace{1cm} (3.3)

based on

$$\varphi(x^j, i, j) = \begin{cases} 1, & \text{if } X^j = y^j_i, \\ 0, & \text{otherwise.} \end{cases}$$ \hspace{1cm} (3.4)

2) Normalize $H$: $\sum_{j=1}^{n} M_W(i, j) = 1, \ \forall i = 1, ..., N$:

$$M_W(i, j) = \frac{H(i, j)}{\sum_{j=1}^{n} H(i, j)}, \quad \forall i \in [1, ..., N], \quad \forall j \in [1, ..., n]$$ \hspace{1cm} (3.5)

3) Given a test data sample $\rho$, decode based on,

$$\delta(\rho, i) = \sum_{j=1}^{n} M_W(i, j)L(y^j_i \cdot f(\rho, j))$$ \hspace{1cm} (3.6)
C.P.: Adaboost Look up table (LUT) representation

ALGORITHM 2: Discrete Adaboost testing algorithm.

1: Given a test sample $\rho$
2: $F(\rho) = 0$
3: Repeat for $m = 1, 2, ... , M$:
   (a) $F(\rho) = F(\rho) + c_m(P_m \cdot \rho_m^m < P_m \cdot T_m)$;
4: Output $\text{sign}(F(\rho))$
Parallelization Implementation

**Algorithm 3:** Critical section serial pseudocode for the testing stage.

- **inputVolume**: Original 3D voxel model with density values \((d)\)
- **outputVolume**: Multiclass labeled voxel model with single value samples
- **voxelFeatures**: pointer to the 8 sample features for the voxel sample being processed
- **\(L\)**: \(\mathcal{L}\) matrix pointer containing the LUTs
- **\(X\)**: code word of binary decision values for a single voxel sample
- **\(M\)**: coding matrix
- **background_value**: density value threshold for the actual voxel sample to be processed

```
for \(z \leftarrow 0\) to \(\text{dim}_z\) do
    for \(y \leftarrow 0\) to \(\text{dim}_y\) do
        for \(x \leftarrow 0\) to \(\text{dim}_x\) do
            \(d \leftarrow \text{input}[z][y][x]\) \hspace{1em} // d refers to the density value in the voxel model
            if \(d > \text{background\_value}\) then
                \(\text{computeGradient}(z, y, x, \text{inputVolume}[z][y][x], \text{voxelFeatures});\) \hspace{1em} // T1
                \(\text{XCodeEstimation}(\text{voxelFeatures}, \mathcal{L}, X);\) \hspace{1em} // T2
                \(\text{FinalLabeling}(X, M, \text{outputVolume}[z][y][x]);\) \hspace{1em} // T3
            end
        end
    end
end
```
P.I.: Parallelization proposals

- Tasks: T1, T2, and T3
- T1 is a 7 point stencil operation
P.I.: Parallelization proposals

Task 2 option 1 and Task 3 option 1
P.I.: Parallelization proposals

Task 2 option 2
P.I.: GPU implementation
P.I.: Simulations and results
P.I.: Simulations and results

Foot classification accuracies

Brain classification accuracies

Torax classification accuracies
## P.I.: Simulations and results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>N</th>
<th>Sel. classes</th>
<th>Z</th>
<th>CPU Core i5</th>
<th>OpenMP Core i5</th>
<th>GCD Core i5</th>
<th>OpenCL GTX 470</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foot</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>0.387</td>
<td>0.111</td>
<td>0.111</td>
<td>0.008</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>0.577</td>
<td>0.165</td>
<td>0.165</td>
<td>0.002</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>0.948</td>
<td>0.271</td>
<td>0.271</td>
<td>0.020</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>1.139</td>
<td>0.325</td>
<td>0.325</td>
<td>0.038</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>6</td>
<td>15</td>
<td>4.986</td>
<td>0.760</td>
<td>0.769</td>
<td>0.062</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>9</td>
<td>36</td>
<td>8.319</td>
<td>1.787</td>
<td>1.777</td>
<td>0.091</td>
</tr>
<tr>
<td>Brain</td>
<td>9</td>
<td>2</td>
<td>15</td>
<td>39.396</td>
<td>11.190</td>
<td>11.177</td>
<td>0.358</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>4</td>
<td>26</td>
<td>68.485</td>
<td>19.475</td>
<td>19.615</td>
<td>0.649</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>6</td>
<td>33</td>
<td>87.558</td>
<td>24.947</td>
<td>24.875</td>
<td>0.848</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>9</td>
<td>36</td>
<td>96.859</td>
<td>27.642</td>
<td>27.557</td>
<td>1.263</td>
</tr>
<tr>
<td>Thorax</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>26.849</td>
<td>7.604</td>
<td>7.600</td>
<td>2.694</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>2</td>
<td>13</td>
<td>321.768</td>
<td>94.589</td>
<td>94.579</td>
<td>2.011</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>4</td>
<td>22</td>
<td>564.577</td>
<td>160.801</td>
<td>160.784</td>
<td>3.532</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>8</td>
<td>28</td>
<td>754.203</td>
<td>220.430</td>
<td>220.388</td>
<td>6.007</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>7</td>
<td>35</td>
<td>923.225</td>
<td>260.751</td>
<td>259.489</td>
<td>5.955</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>9</td>
<td>36</td>
<td>971.915</td>
<td>270.751</td>
<td>269.751</td>
<td>7.763</td>
</tr>
</tbody>
</table>
P.I.: Simulations and results

- With Radeon HD 7970
- Thorax N=9 Sel classes=9 and Z=36
- Execution time = 2,2 seconds
- No code change

<table>
<thead>
<tr>
<th></th>
<th>Geforce GTX 470</th>
<th>Radeon HD 7970</th>
<th>Comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing Elements</td>
<td>448</td>
<td>2048</td>
<td>4,5x</td>
</tr>
<tr>
<td>Execution time</td>
<td>7,763</td>
<td>2,2</td>
<td>3,5x</td>
</tr>
</tbody>
</table>
AI parallel system proposal

OmpSs (BSC)

Scala

OpenMP

OpenCL

Open MPI Documentation
AI.P.S.P.: Environment
AI.P.S.P.: Agents
AI.P.S.P.: Agents
AI.P.S.P.: Desired behaviors

- Hierarchical scheduling
- Data affinity
- Data flow
- Program flow

Resource usefulness evaluation “V”
- Each agent has its own view of “V” for each resource
- $V = PV + Rr$
- $PV$ = amount of matches a resource adds
- $Rr$ = Resource ratio is the amount of computational resources available to the Agent
- We can add network costs
AI.P.S.P.: Experiments and results

- First environment simulator
- We control
  - Number of Blocks
  - Number of Data elements
  - Number of Agents on the Grid
  - Number of Instructions for each Block
  - Number of data requests the Agent will raise to the grid on a single time step
  - Number of instructions the Agent will fetch and execute on each time step
- Next step:
  - Adding the Agent program to generate the exchange Behavior
  - Use agent programming language (2APL for instance)
  - Use Adapteva’s Parallella board as a Node.
Conclusions

● Parallelism for AI:
  o New parallel programming tools (AIMparallel and SimpleOpenCL)
  o Semi automatic classification:
    ▪ Framework for 3D scan medical images
    ▪ Parallel implementation

● AI for parallelism:
  o A parallel computing system
  o Based on an agent strategy for automatic scheduling
Future work

● Parallelism for AI:
  o Apply AIMparallel to other AI methods and architectures
  o Increase accuracy with more features and context
  o Increase performance (new GPU features and reducing aWPO)

● AI for parallelism:
  o Implement a simple working system (Either simulator or on Parallela board)
Future work: Hardware proposal

- Adapteva’s Parallella board as Node