Parallel error-correcting output codes classification in volume visualization:

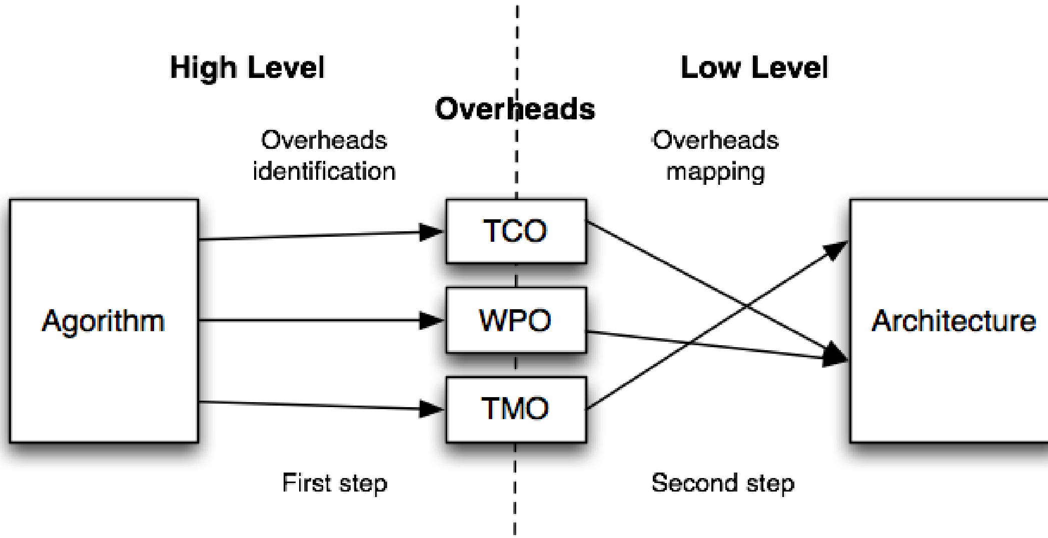
parallelism for AI and AI for parallelism

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

Introduction

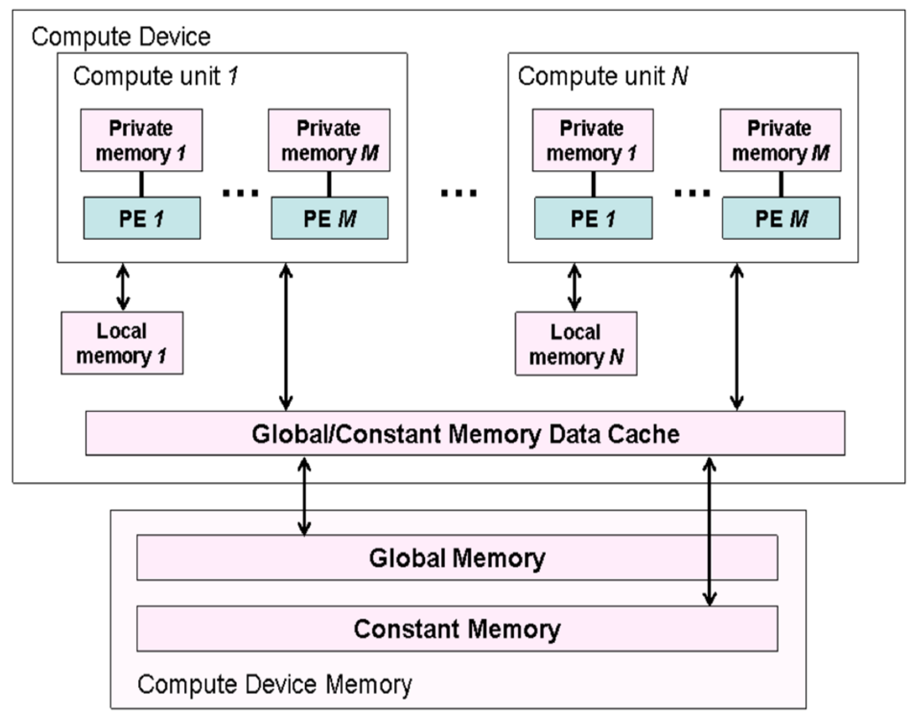
- Main Goal: explore AI and parallelism interaction.
- Contributions:
  - New parallel programming tools (AIMparallel and SimpleOpenCL)
  - Semi automatic classification:
    - Framework for 3D scan medical images
    - Parallel implementation of the framework (Parallelism for AI)
  - A new parallel system proposal based on SOMAS (AI for parallelism)

AIMparallel



Methodology and nomenclature.
Overheads:

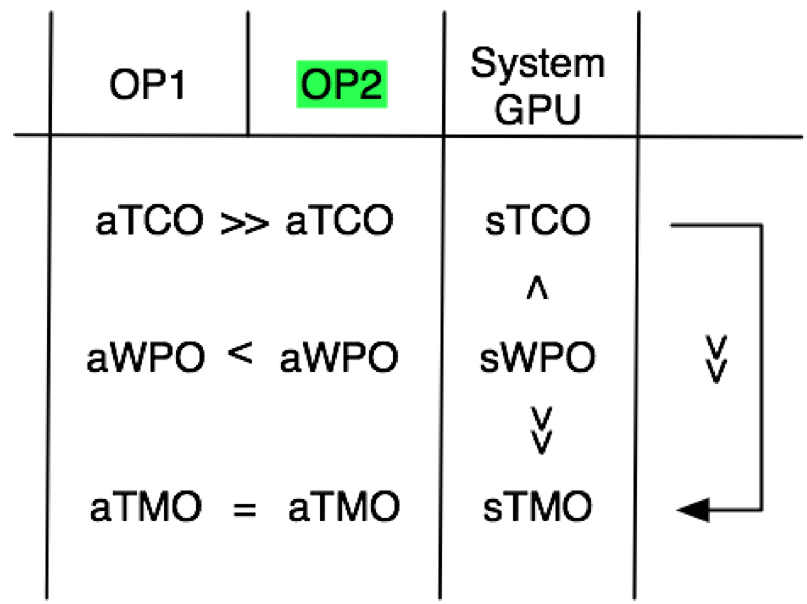
- TCO: cycles lost due to data movement
- WPO: cycles lost due to load unbalancing
- TMO: cycles lost due to thread management



AIMparallel

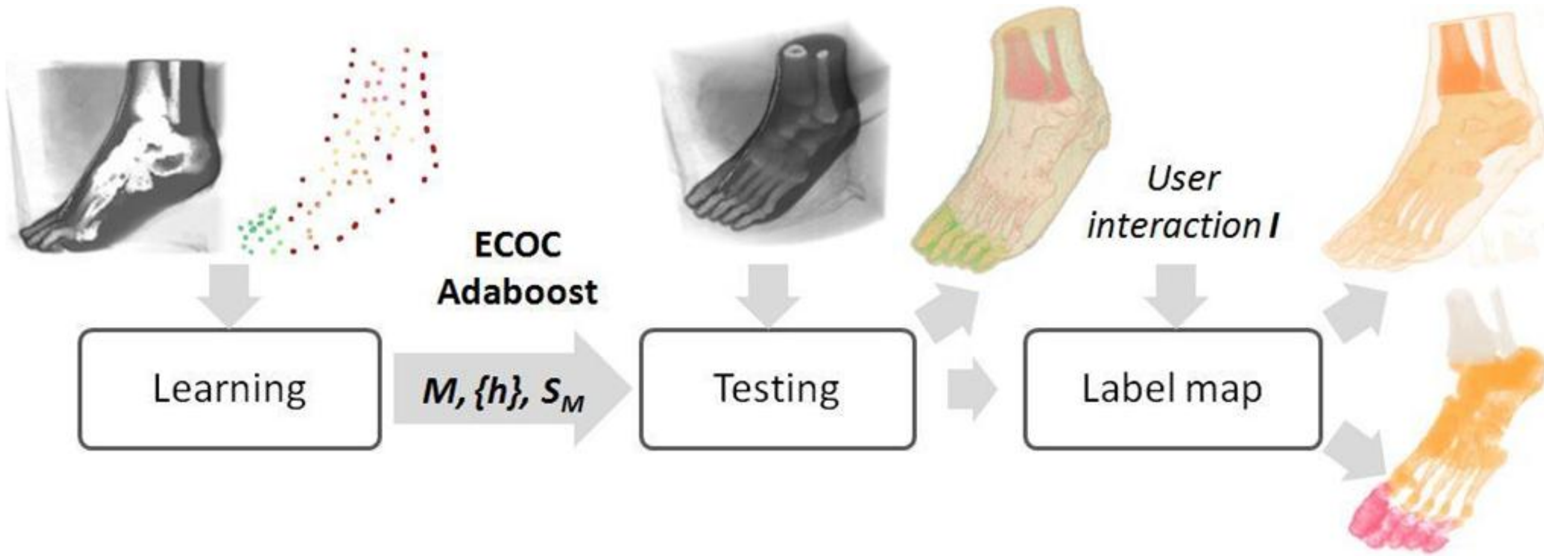
Example

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



Classification problem

A semi automatic classification system



C.P.: proposal ECOC

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

M

	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9	h_{10}	h_{11}	h_{12}	h_{13}	h_{14}	h_{15}	$HD(X,y_i)$
y_1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	→ 0.5
y_2	-1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	→ 4.5
y_3	0	-1	0	0	0	-1	0	0	0	1	1	1	0	0	0	→ 5.5
y_4	0	0	-1	0	0	0	-1	0	0	-1	0	0	1	1	0	→ 5.5
y_5	0	0	0	-1	0	0	0	-1	0	0	-1	0	-1	0	1	→ 5.5
y_6	0	0	0	0	-1	0	0	0	-1	0	0	-1	0	-1	-1	→ 5.5
X	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	

C.P.: proposal Adaboost

ALGORITHM 1: Discrete Adaboost training algorithm.

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

ALGORITHM 2: Discrete Adaboost testing algorithm.

1: Given a test sample ρ
2: $F(\rho) = 0$
3: Repeat for $m = 1, 2, .., \mathcal{M}$:
  (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

M

	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9	h_{10}	h_{11}	h_{12}	h_{13}	h_{14}	h_{15}	$HD(X,y_i)$
y_1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	→ 0.5
y_2	-1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	→ 4.5
y_3	0	-1	0	0	0	-1	0	0	0	1	1	1	0	0	0	→ 5.5
y_4	0	0	-1	0	0	0	-1	0	0	-1	0	0	1	1	0	→ 5.5
y_5	0	0	0	-1	0	0	0	-1	0	0	-1	0	-1	0	1	→ 5.5
y_6	0	0	0	0	-1	0	0	0	-1	0	0	-1	0	-1	-1	→ 5.5
X	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	

S_M

	h_2	h_5	h_6	h_9	h_{10}	h_{11}	h_{12}	h_{14}	h_{15}
y_1	1	1	0	0	0	0	0	0	0
y_2	0	0	1	1	0	0	0	0	0
y_3	-1	0	-1	0	1	1	1	0	0
y_4	0	0	0	0	-1	0	0	1	0
y_5	0	0	0	0	0	-1	0	0	1
y_6	0	-1	0	-1	0	0	-1	-1	-1

C.P.: adaptive decoding

Loss-Weighted strategy: Given a coding matrix M,

1) Calculate the performance matrix H,

$$H(i,j) = \frac{1}{m_i} \sum_{k=1}^{m_i} \varphi(h^j(\rho_k^i), i, j) \tag{3.3}$$

based on $$\varphi(x^j, i, j) = \begin{cases} 1, & \text{if} \quad X^j = y_i^j, \\ 0, & \text{otherwise.} \end{cases} \tag{3.4}$$

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

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

3) Given a test data sample ρ, decode based on,

$$\delta(\rho, i) = \sum_{j=1}^{n} M_W(i,j) L(y_i^j \cdot f(\rho, j)) \tag{3.6}$$

$$M = \begin{bmatrix} 1 & 1 & -1 & 0 \\ 1 & -1 & 0 & 0 \\ -1 & 1 & 1 & -1 \end{bmatrix}$$

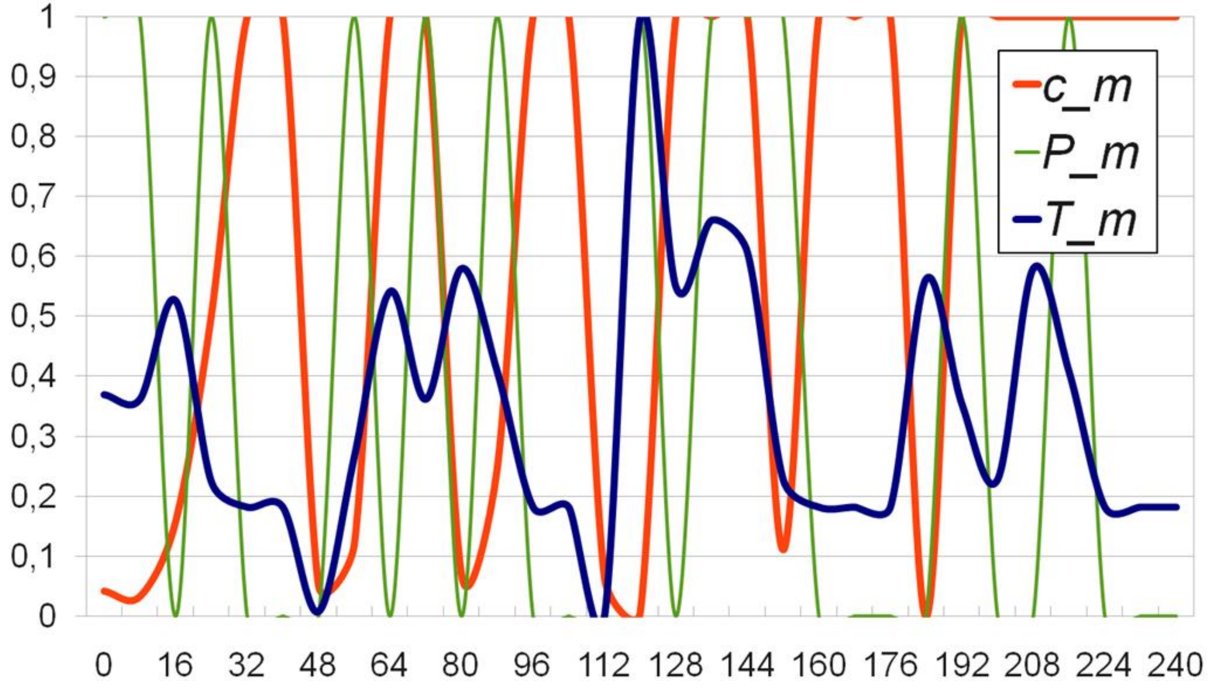
$$H = \begin{bmatrix} 0.955 & 0.955 & 1.000 & 0.000 \\ 0.900 & 0.800 & 0.000 & 0.000 \\ 1.000 & 0.905 & 0.805 & 0.805 \end{bmatrix}$$

$$M_W = \begin{bmatrix} 0.328 & 0.328 & 0.344 & 0.000 \\ 0.529 & 0.471 & 0.000 & 0.000 \\ 0.285 & 0.257 & 0.229 & 0.229 \end{bmatrix}$$

C.P.: Adaboost Look up table (LUT) representation

ALGORITHM 2: Discrete Adaboost testing algorithm.

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



Parallelization Implementation

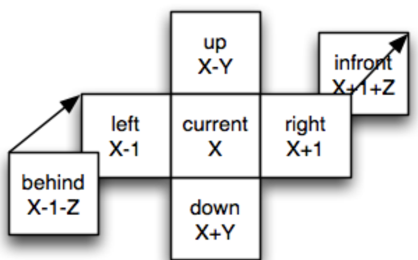
ALGORITHM 3: Critical section serial pseudocode for the testing stage.

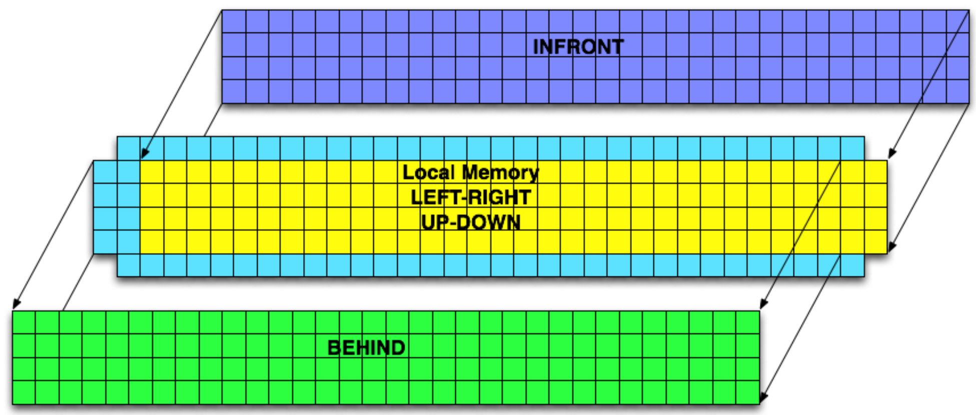
inputVolume	:	Original 3D voxel model with density values (d)
outputVolume	:	Multiclass labeled voxel model with single value samples
voxelFeatures	:	pointer to the 8 sample features for the voxel sample being processed
$\mathcal{L}$	:	$\mathcal{L}$ matrix pointer containing the LUTs
X	:	code word of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

```
for z ← 0 to dim_z do
    for y ← 0 to dim_y do
        for x ← 0 to dim_x do
            d ← input[z][y][x] // d refers to the density value in the voxel model
            if d > background_value then
                computeGradient(z, y, x, inputVolume[z][y][x], voxelFeatures); // T1
                XCodeEstimation(voxelFeatures, L, X); // T2
                FinalLabeling(X, M, outputVolume[z][y][x]); // T3
            end
        end
    end
end
```

P.I.: Parallelization proposals

- Tasks: T1 T2 and T3
- T1 is a 7 point stencil operation

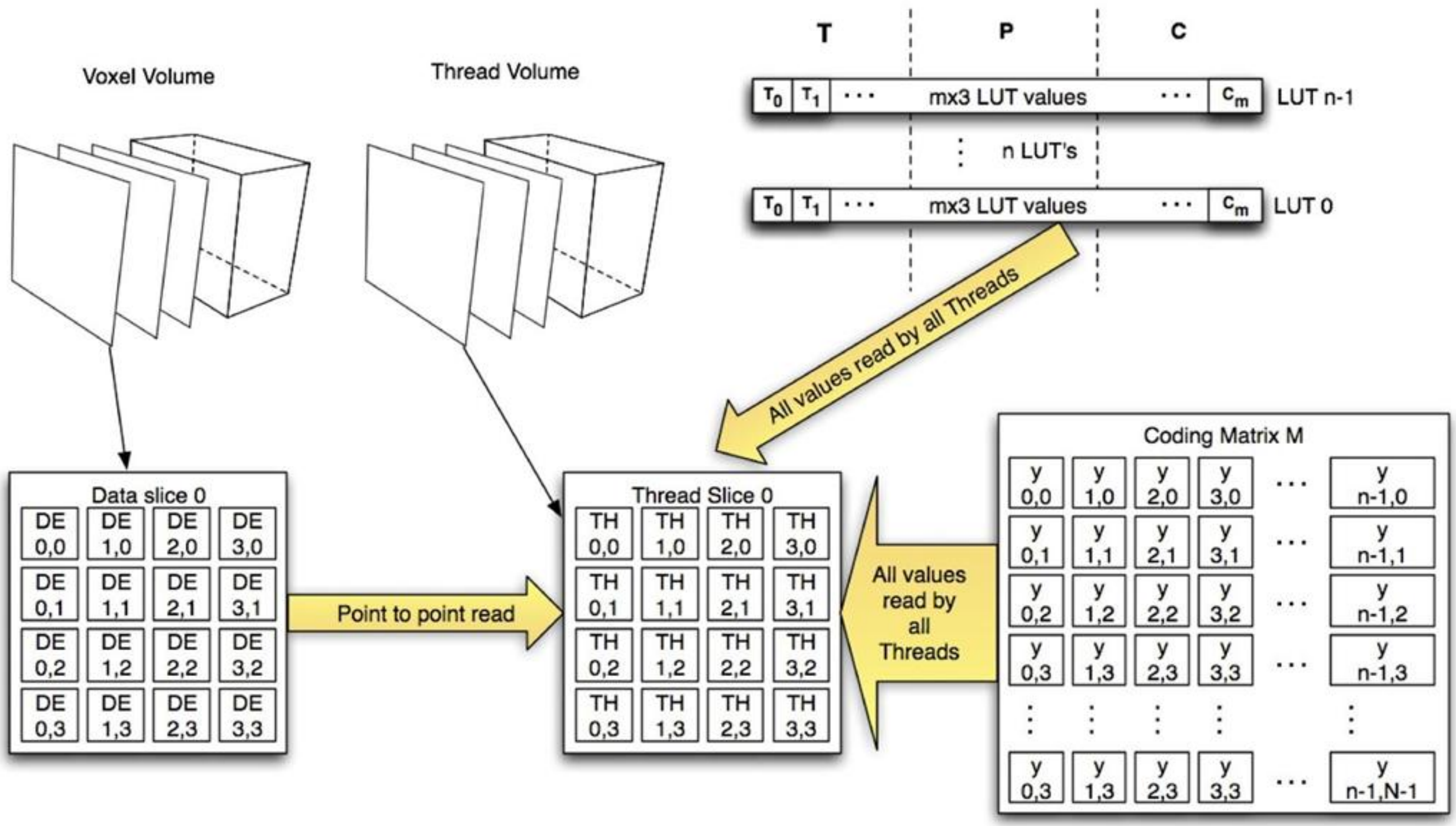




3 position values	Voxel value	4 gradient values

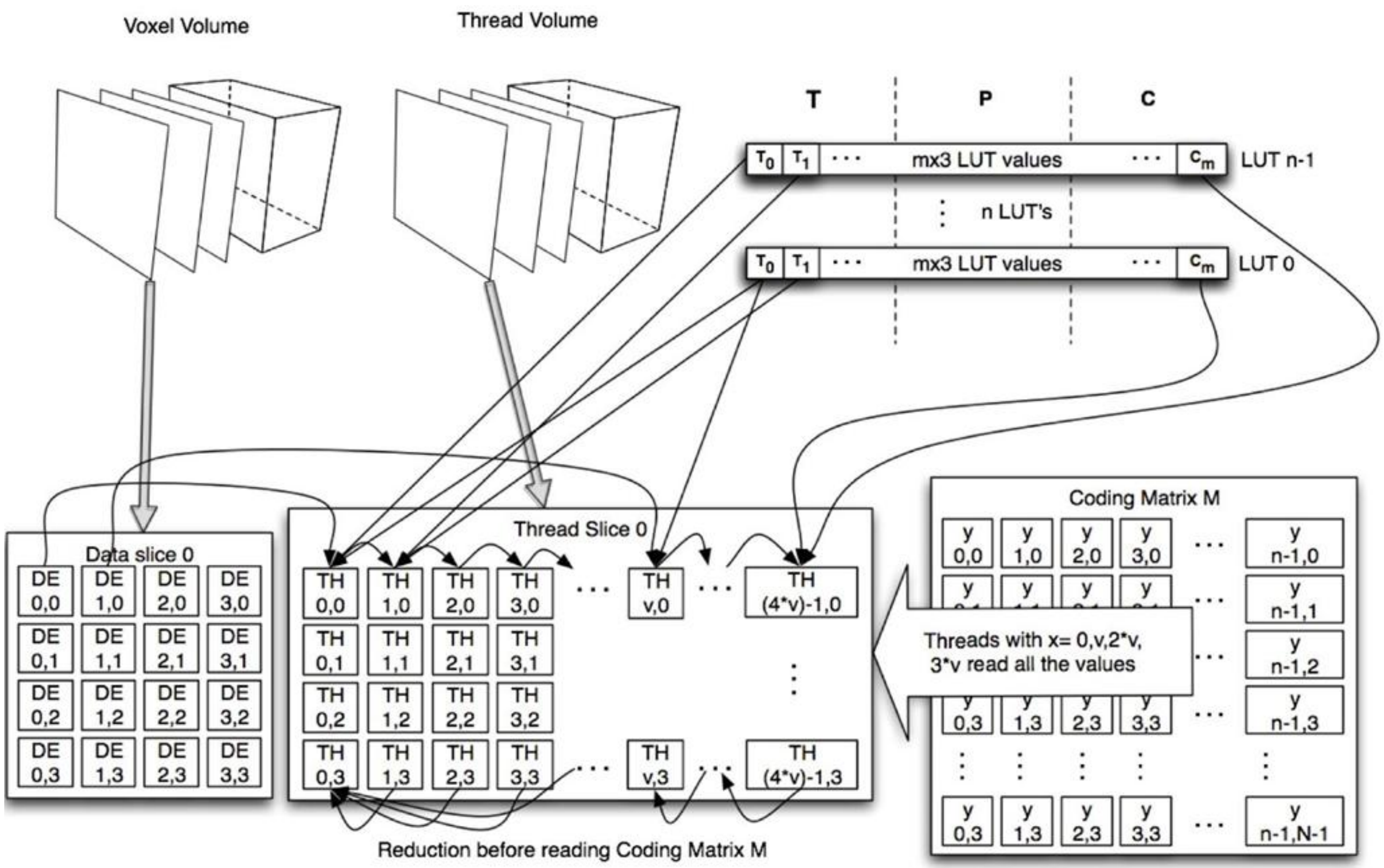
P.I.: Parallelization proposals

Task 2 option 1 and Task 3 option 1

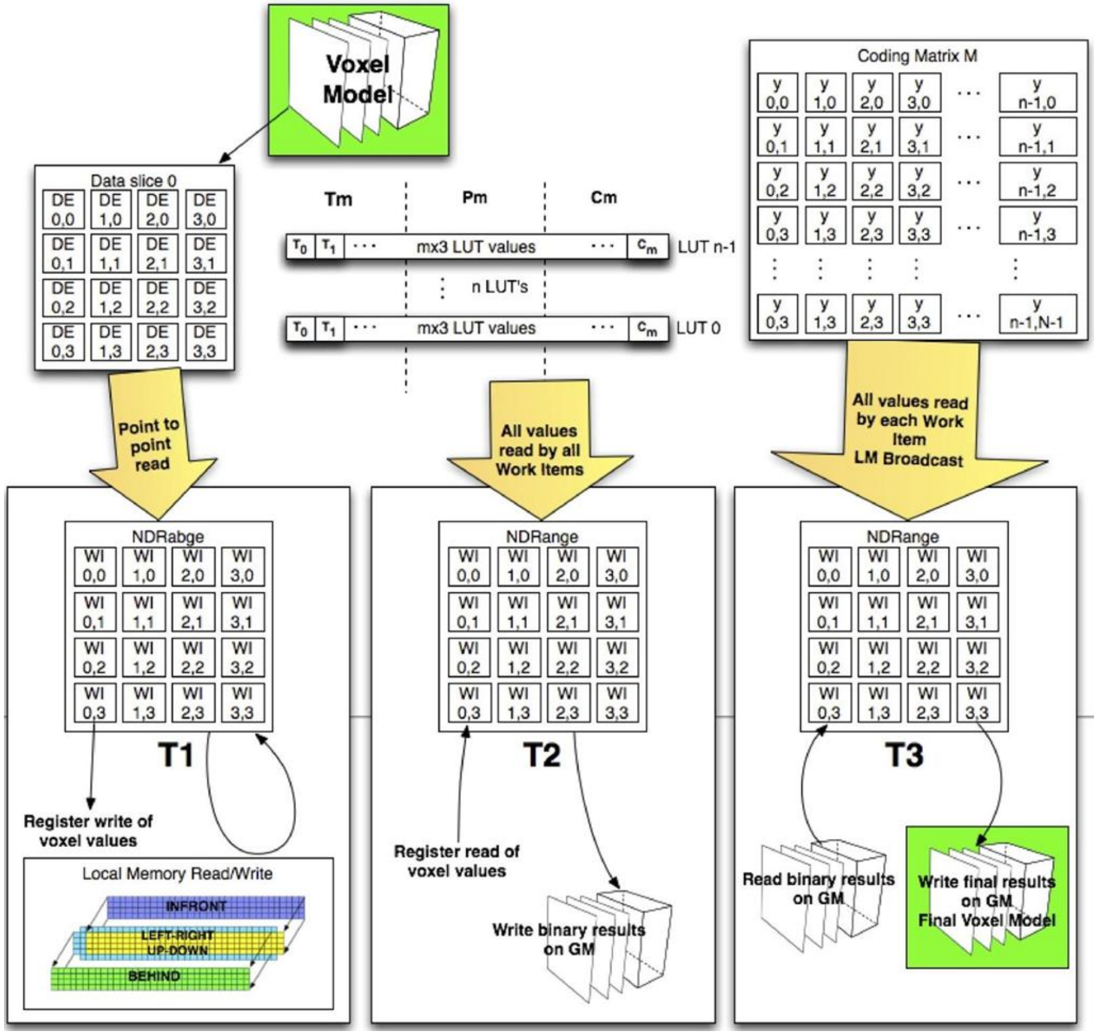


P.I.: Parallelization proposals

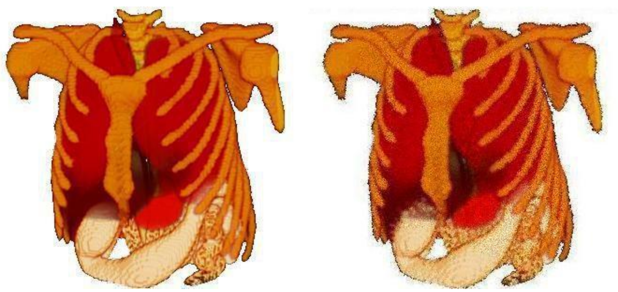
Task 2 option 2

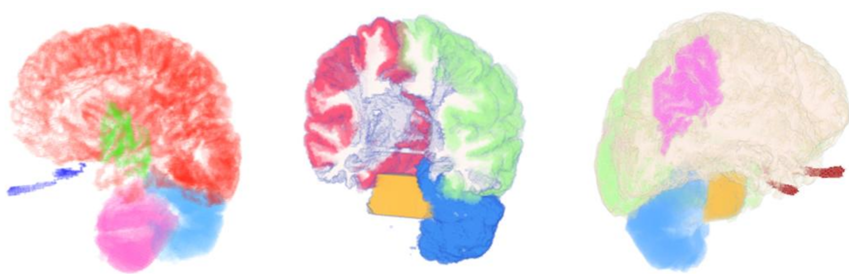


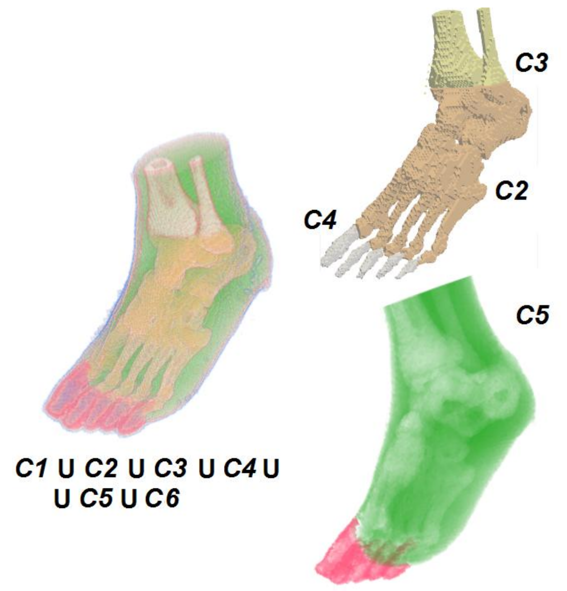
P.I.: GPU implementation



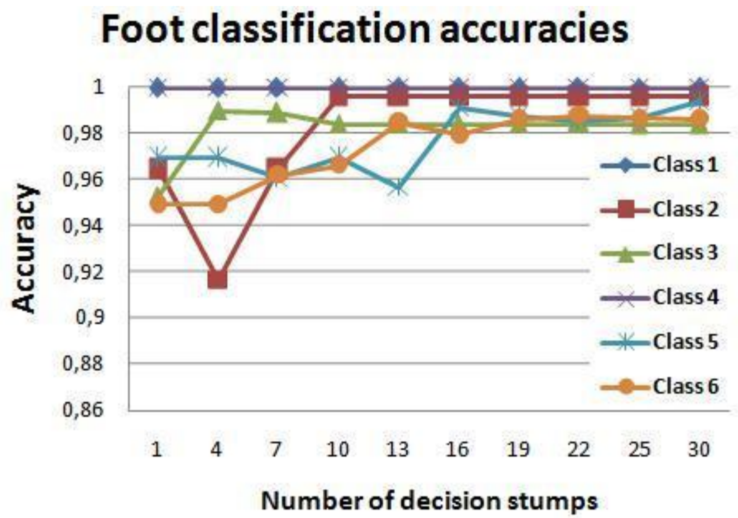
P.I.: Simulations and results

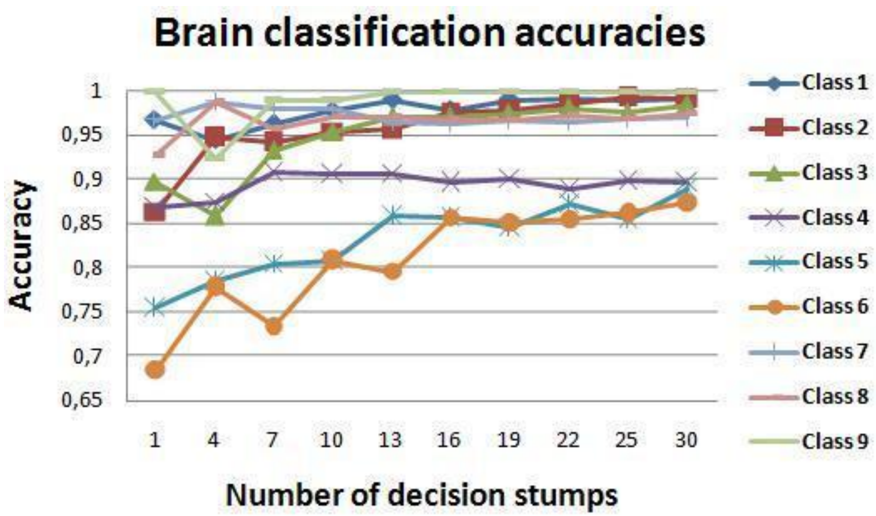


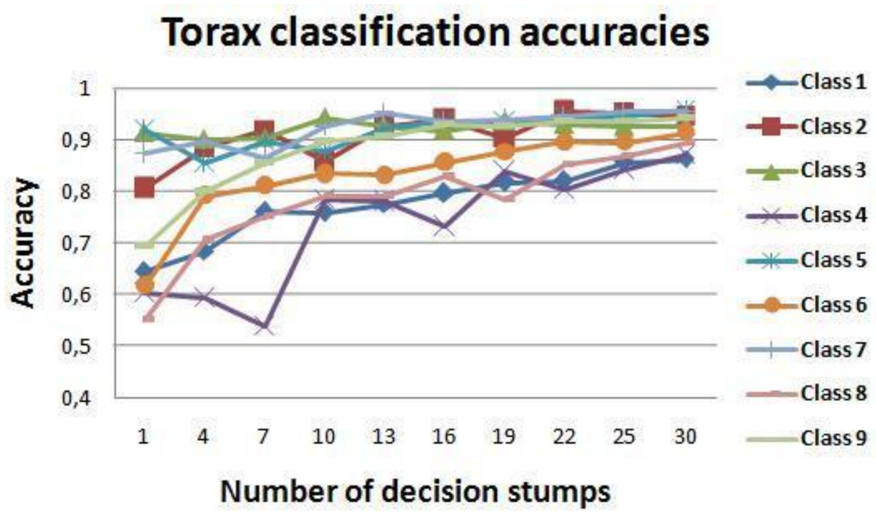




P.I.: Simulations and results







P.I.: Simulations and results

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

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

	Geforce GTX 470	Radeon HD 7970	Comparison
Processing Elements	448	2048	4,5x
Execution time	7,763	2,2	3,5x

AI parallel system proposal

OmpSs (BSC)

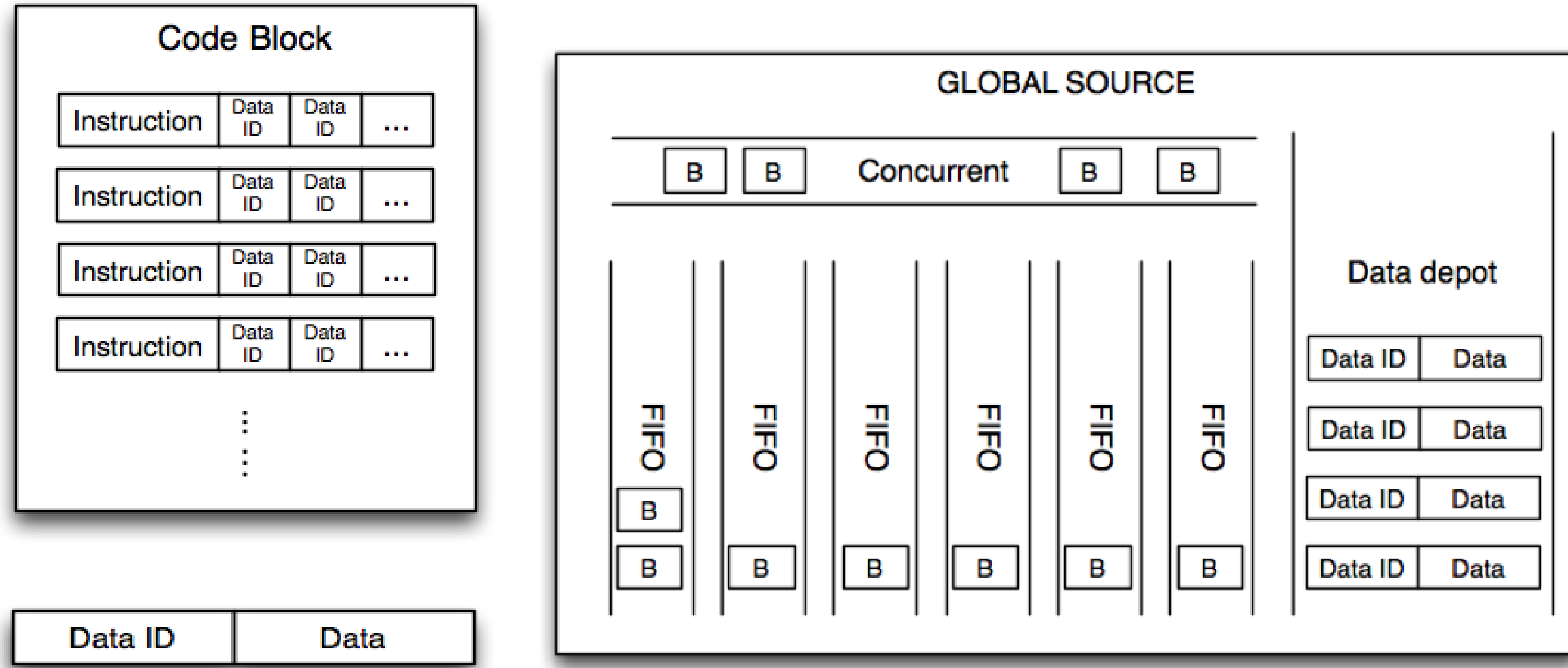
Scala

OpenCL

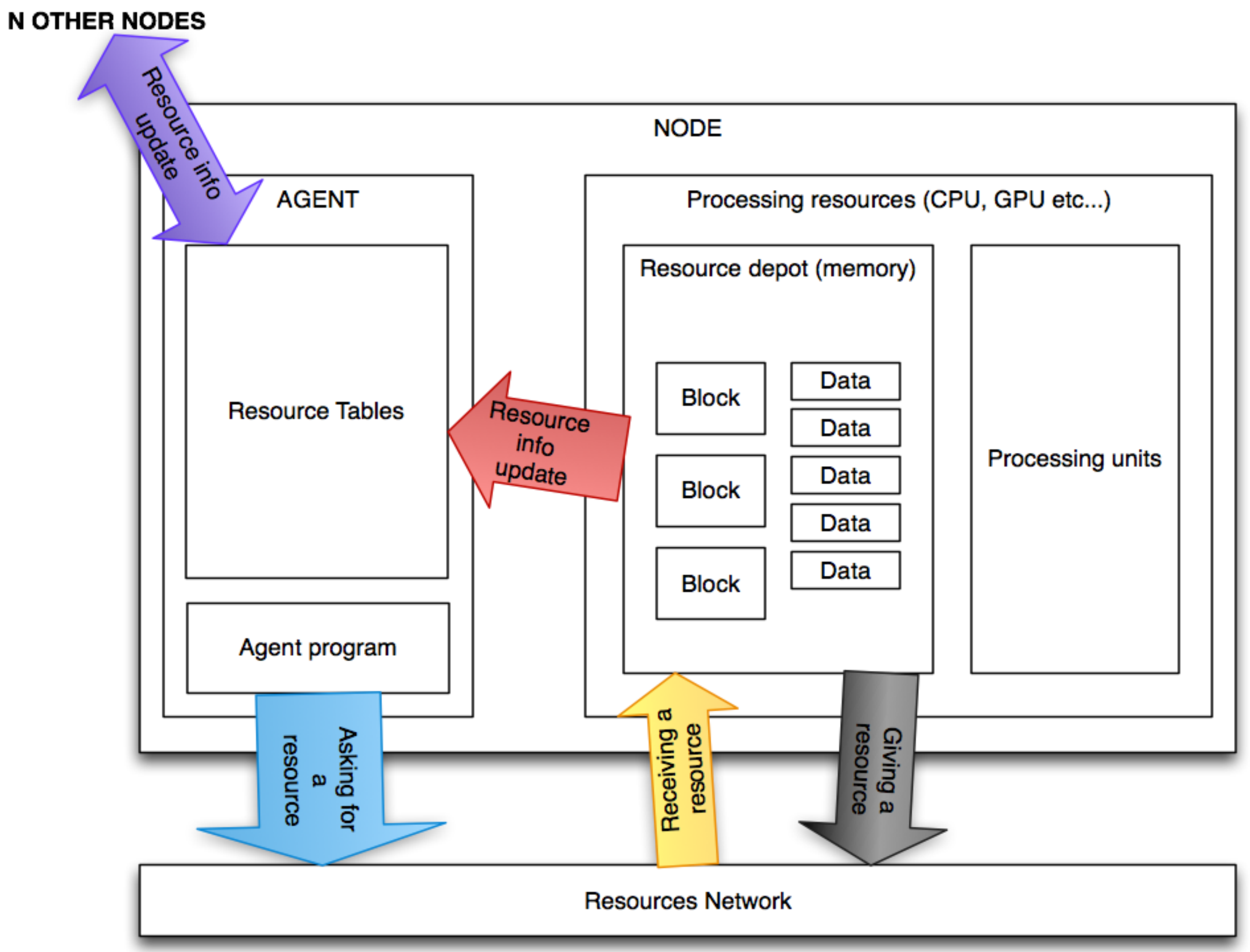
OpenMP®

OPEN MPI
DOCUMENTATION

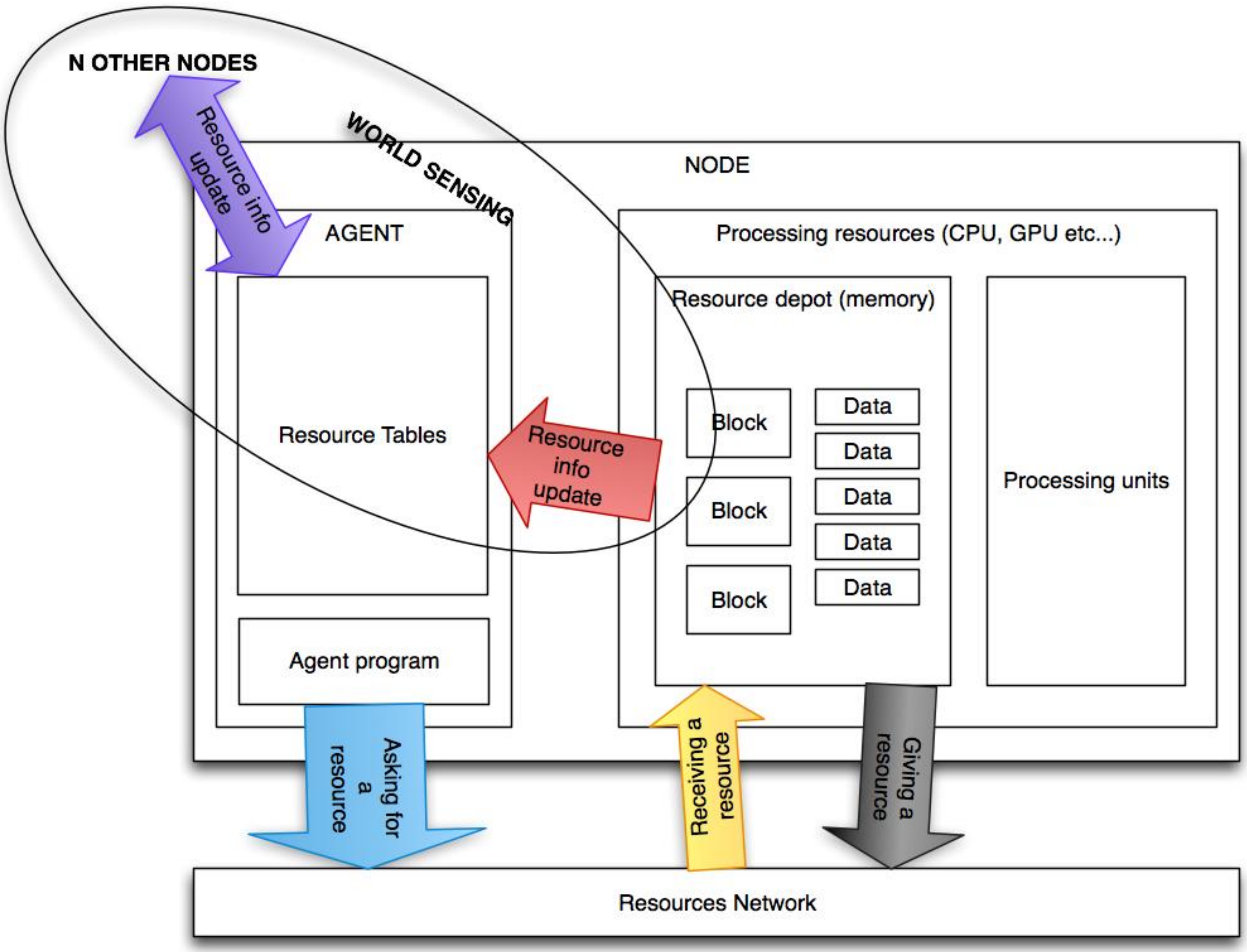
AI.P.S.P.: Environment



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 it’s 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:
  - New parallel programming tools (AIMparallel and SimpleOpenCL)
  - Semi automatic classification:
    - Framework for 3D scan medical images
    - Parallel implementation
- AI for parallelism:
  - A parallel computing system
  - Based on an agent strategy for automatic scheduling

Future work

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

- AI for parallelism:
  - Implement a simple working system (Either simulator or on Parallela board)

Future work: Hardware proposal

- Adapteva's Parallella board as Node