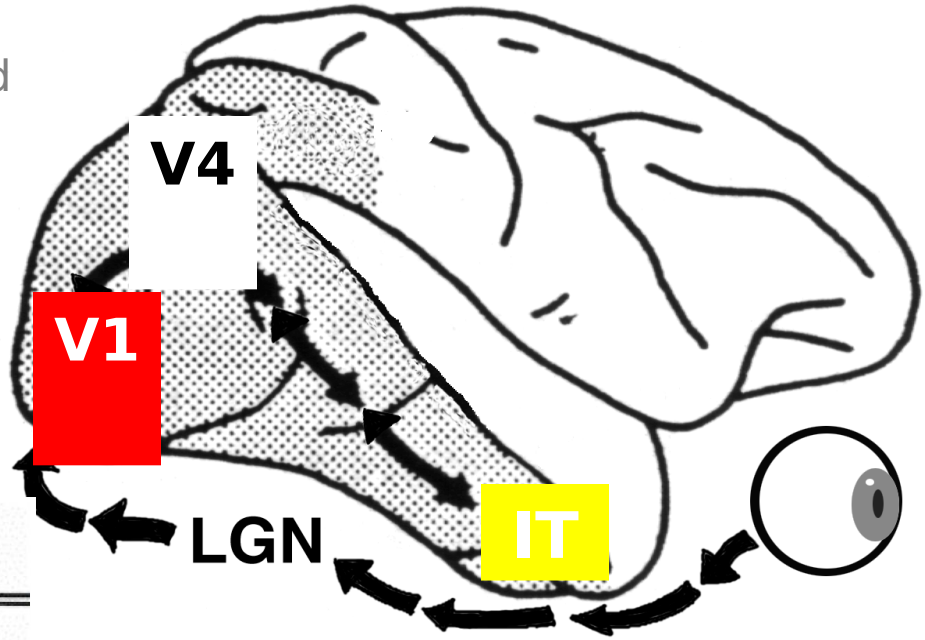
NVIDIA CUDA Implementation of a Hierarchical Object Recognition Algorithm

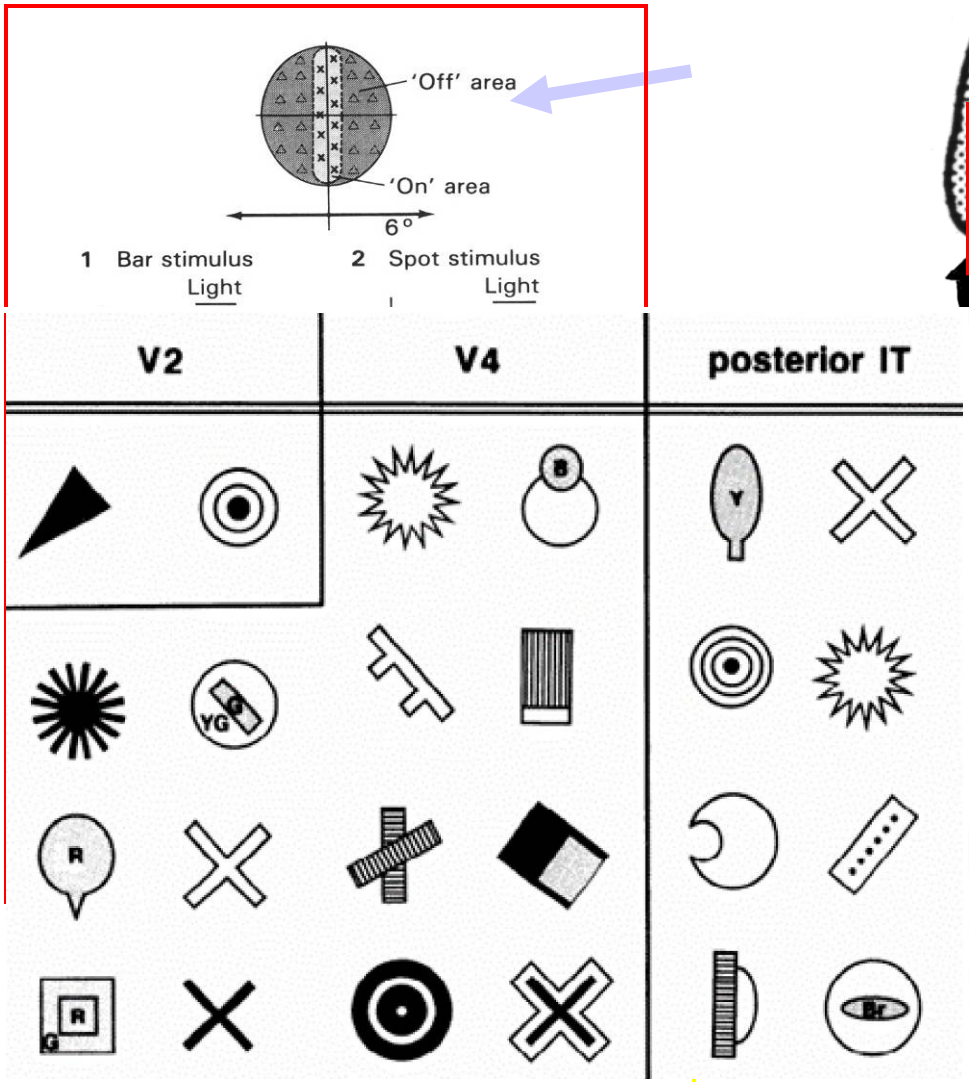
Sharat Chikkerur
CBCL, MIT

Outline

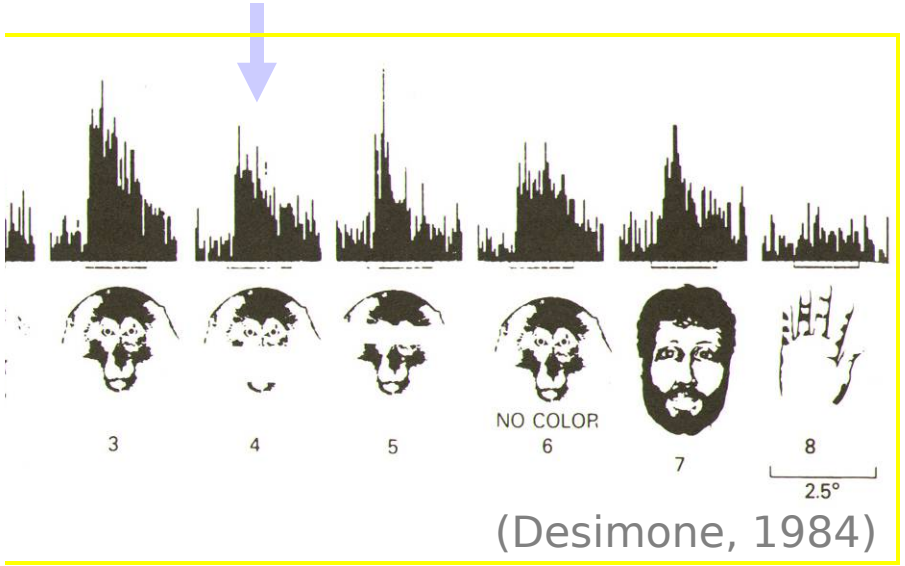
- **Introduction**
  - Motivation
  - Computational model of the ventral stream
- **Multi threaded implementation**
- **CUDA Implementation**
- **Comparison**
- **Conclusion**

modified from (Ungerleider and Haxby, 1994)





(Kobatake and Tanaka, 1994)



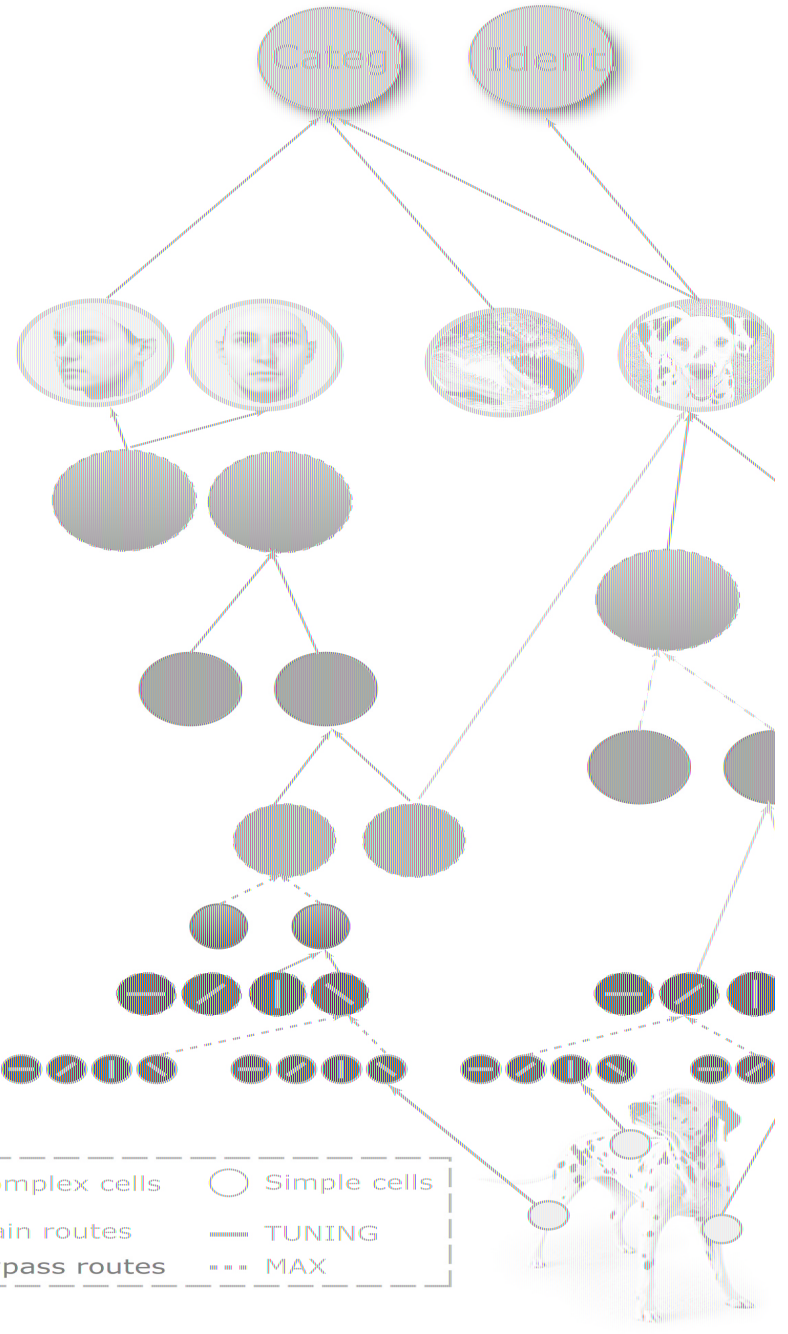
(Desimone, 1984)

Model layers	Corresponding brain area (tentative)	RF sizes	Number units	
classifier	PFC		1.0 10^0	
S4	AIT	>4.4°	1.5 10^2	~ 5,000 subunits
C3	PIT - AIT	>4.4°	2.5 10^3	
C2b	PIT	>4.4°	2.5 10^3	
S3	PIT	1.2°- 3.2°	7.4 10^4	~ 100 subunits
S2b	V4 - PIT	0.9°- 4.4°	1.0 10^7	~ 100 subunits
C2	V4	1.1°- 3.0°	2.8 10^5	
S2	V2 - V4	0.6°- 2.4°	1.0 10^7	~ 10 subunits
C1	V1 - V2	0.4°- 1.6°	1.2 10^4	
S1	V1 - V2	0.2°- 1.1°	1.6 10^6	

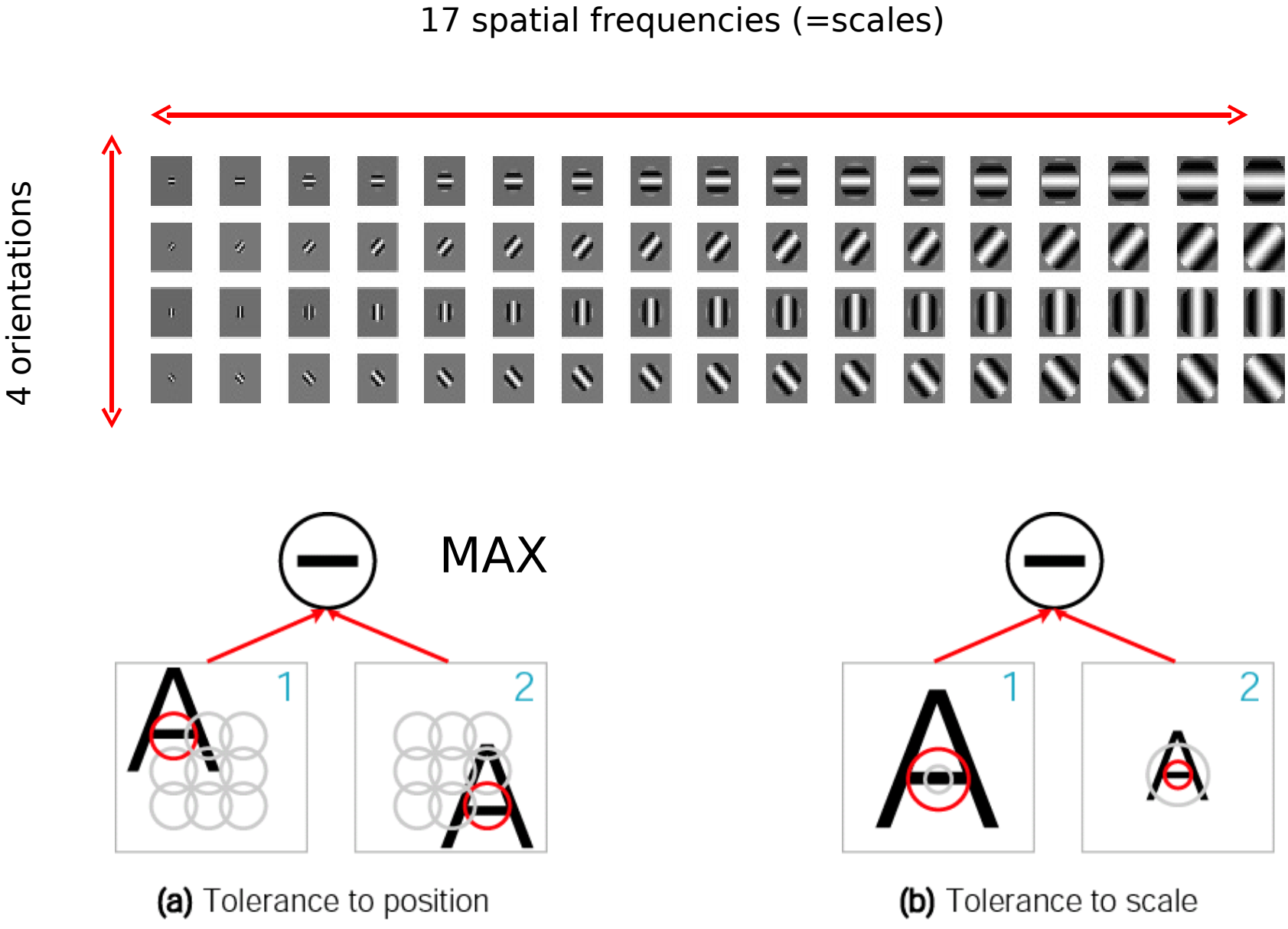
Supervised task-dependent learning

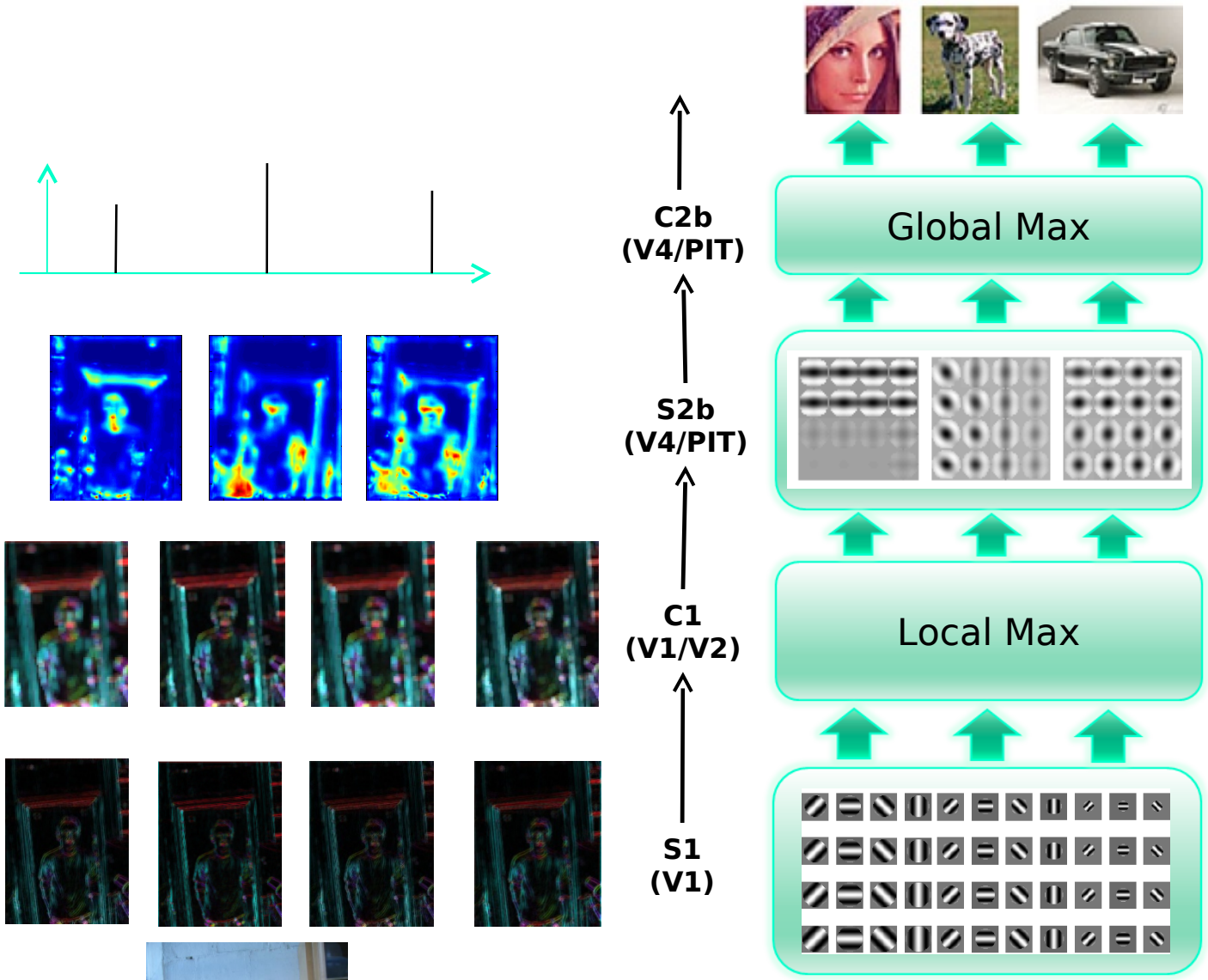
Unsupervised task-independent learning

Increase in complexity (number of subunits), RF size and invariance



Specificity and Invariance




C2b
(V4/PIT)
S2b
(V4/PIT)
C1
(V1/V2)
S1
(V1)
Global Max
Local Max

Baseline performance

Datasets		Benchmark	C_2 features	
			boost	SVM
(CalTech)	Leaves	84.0 [3]	**97.0**	95.9
(CalTech)	Cars	84.8 [4]	99.7	**99.8**
(CalTech)	Faces	96.4 [4]	**98.2**	98.1
(CalTech)	Airplanes	94.0 [4]	**96.7**	94.9
(CalTech)	Motorcycles	95.0 [4]	**98.0**	97.4
(MIT-CBCL)	Faces	90.4 [1]	**95.9**	95.3
(MIT-CBCL)	Cars	75.4 [2]	**95.1**	93.3

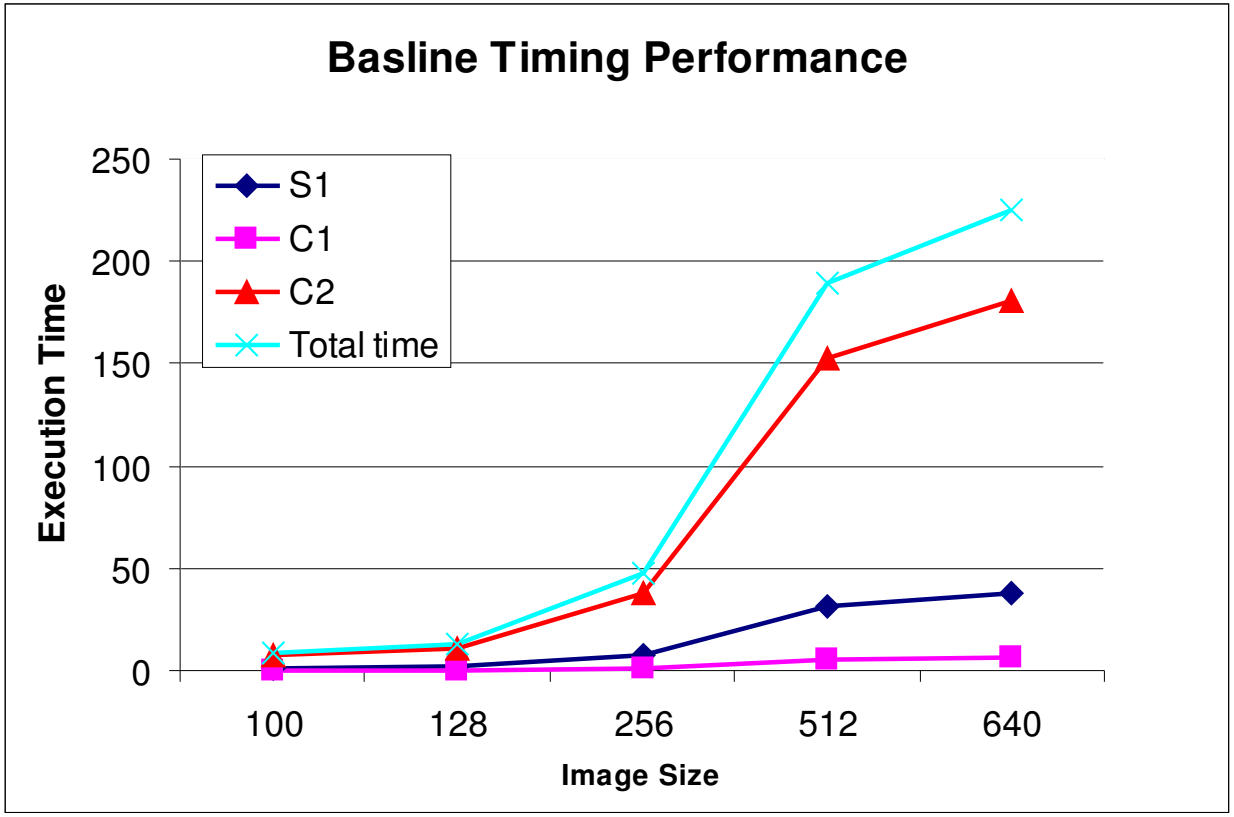
[1] B. Heisele, T. Serre, M. Pontil, T. Vetter, and T. Poggio. Categorization by learning and combining object parts. In *Advances in Neural Information Processing Systems*, volume 14,2002.

[2] B. Leung. Component-based car detection in street scene images. Master's thesis, EECS, MIT, 2004.

[3] M. Weber,W.Welling, and P. Perona. Unsupervised learning of models of recognition. In *Proc. of the European Conference on Computer Vision*, volume 2, pages 1001–108, 2000.

[4] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scaleinvariant learning. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, volume 2,pages 264–271, 2003.

Baseline timing



	S1	C1	C2	Total time
100	1.090	0.320	7.659	9.069
128	1.809	0.459	11.217	13.485
256	7.406	1.422	38.315	47.143
512	31.276	5.648	153.005	189.929
640	37.840	6.580	180.282	224.702

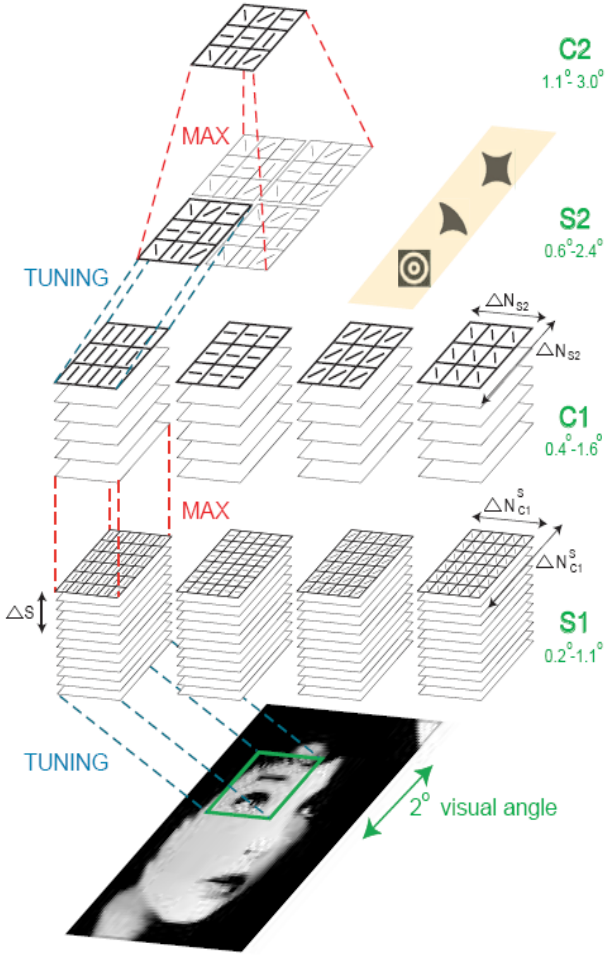
Tests conducted using ‘matlab’ code over a 3GHz machine

Outline

- **Introduction**
  - Motivation
  - Computational model of the ventral stream
- **Multi threaded implementation**
- **CUDA Implementation**
- **Comparison**
- **Conclusion**

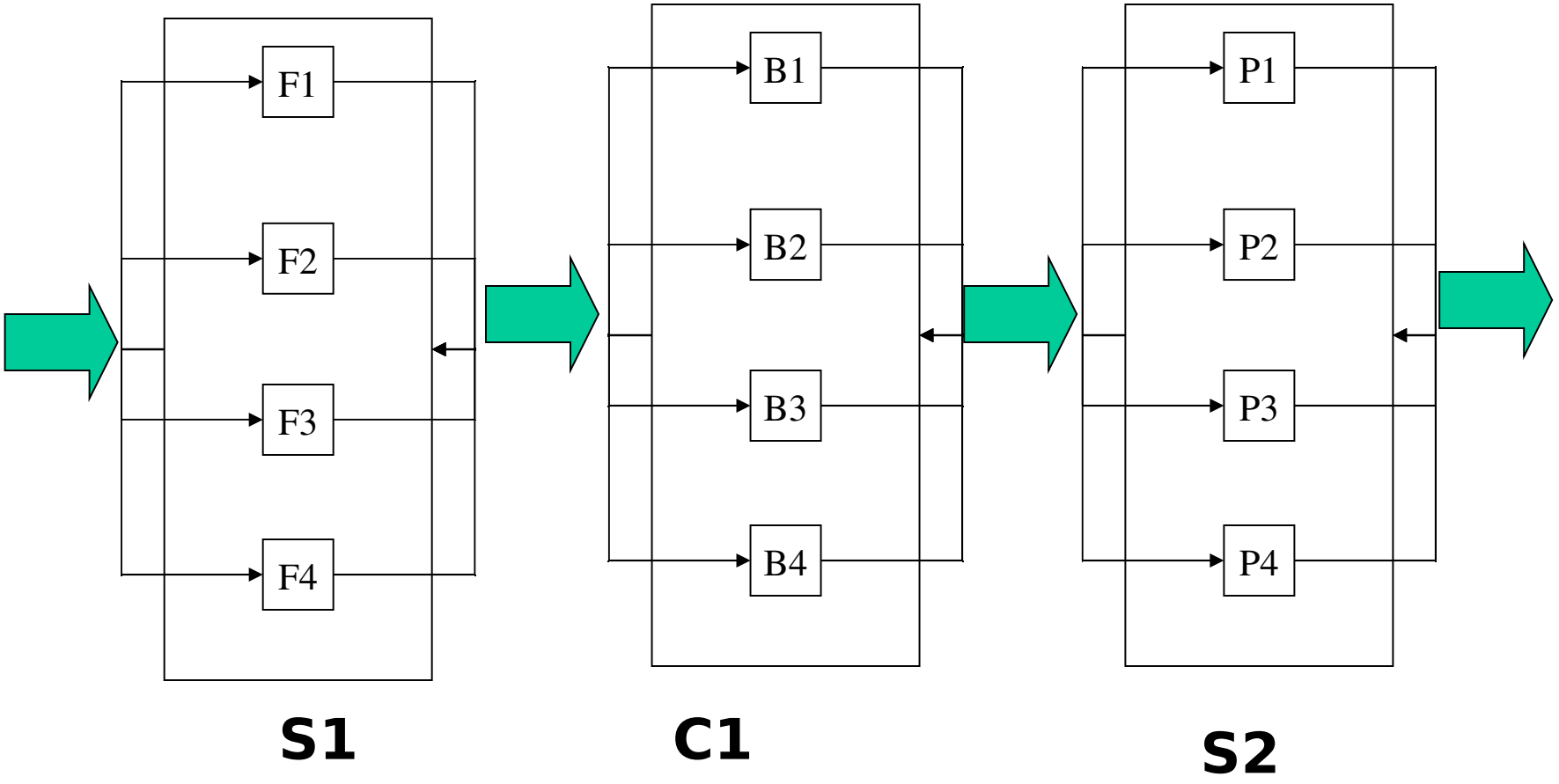
Computational Complexity

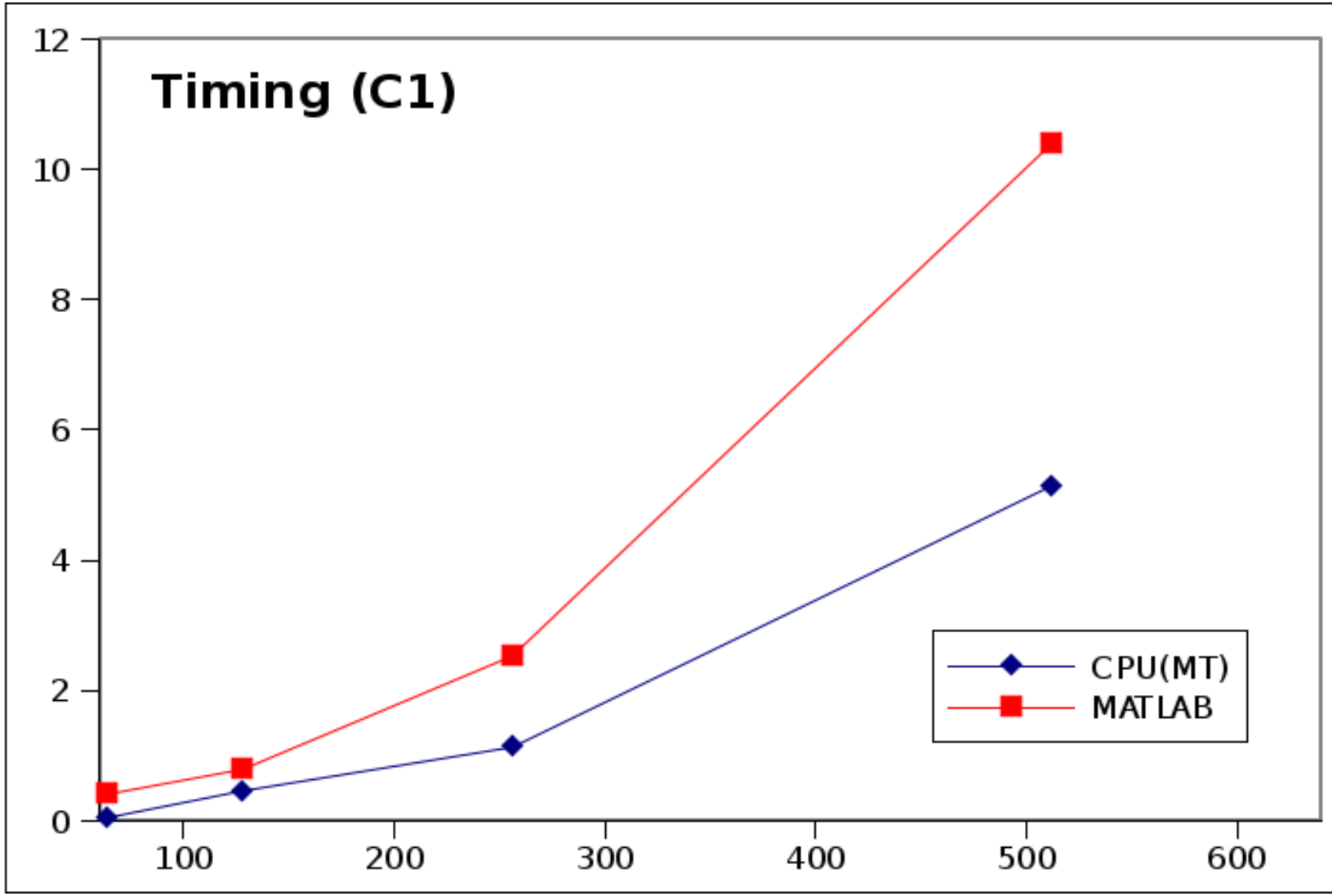
- **S1**
  - Performs normalized cross correlation between a bank of 64 filters (4 directions X 16 scales).
  - Comptutational cost $O(N^2M^2)$
  - NxN – image size, MxM – filter size
- **C1**
  - Performs spatial and across scale max pooling
  - Computational cost $O(N^2M)$
- **S2b**
  - Each S2b unit detects the presence of prototypical C1 patches learnt during training.
  - Computational cost $O(PN^2M^2)$
  - P – number of patches (~2000)
- **C2b**
  - Performs max pooling over all scales and all locations
  - Computational cost $O(N^2MP)$

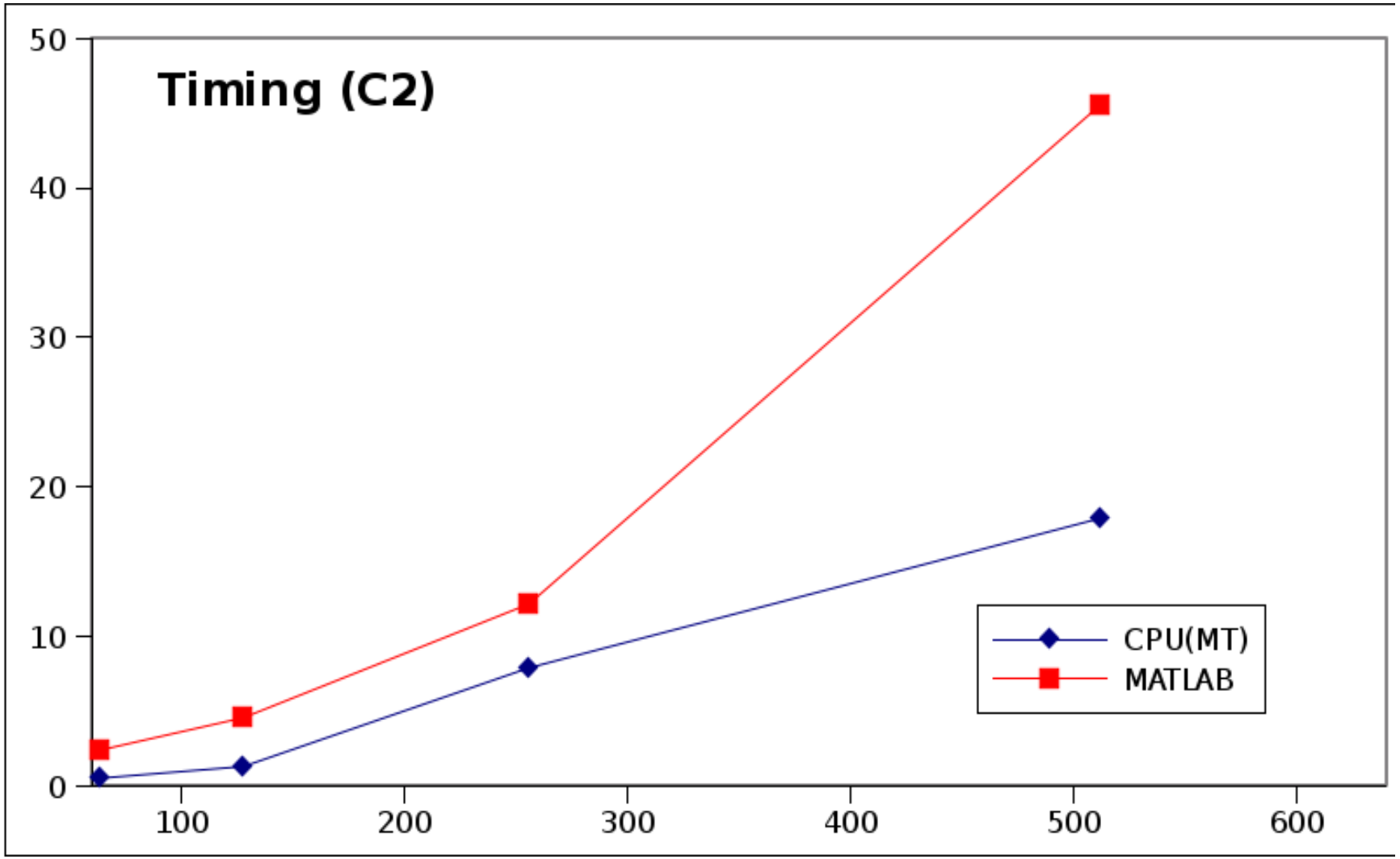


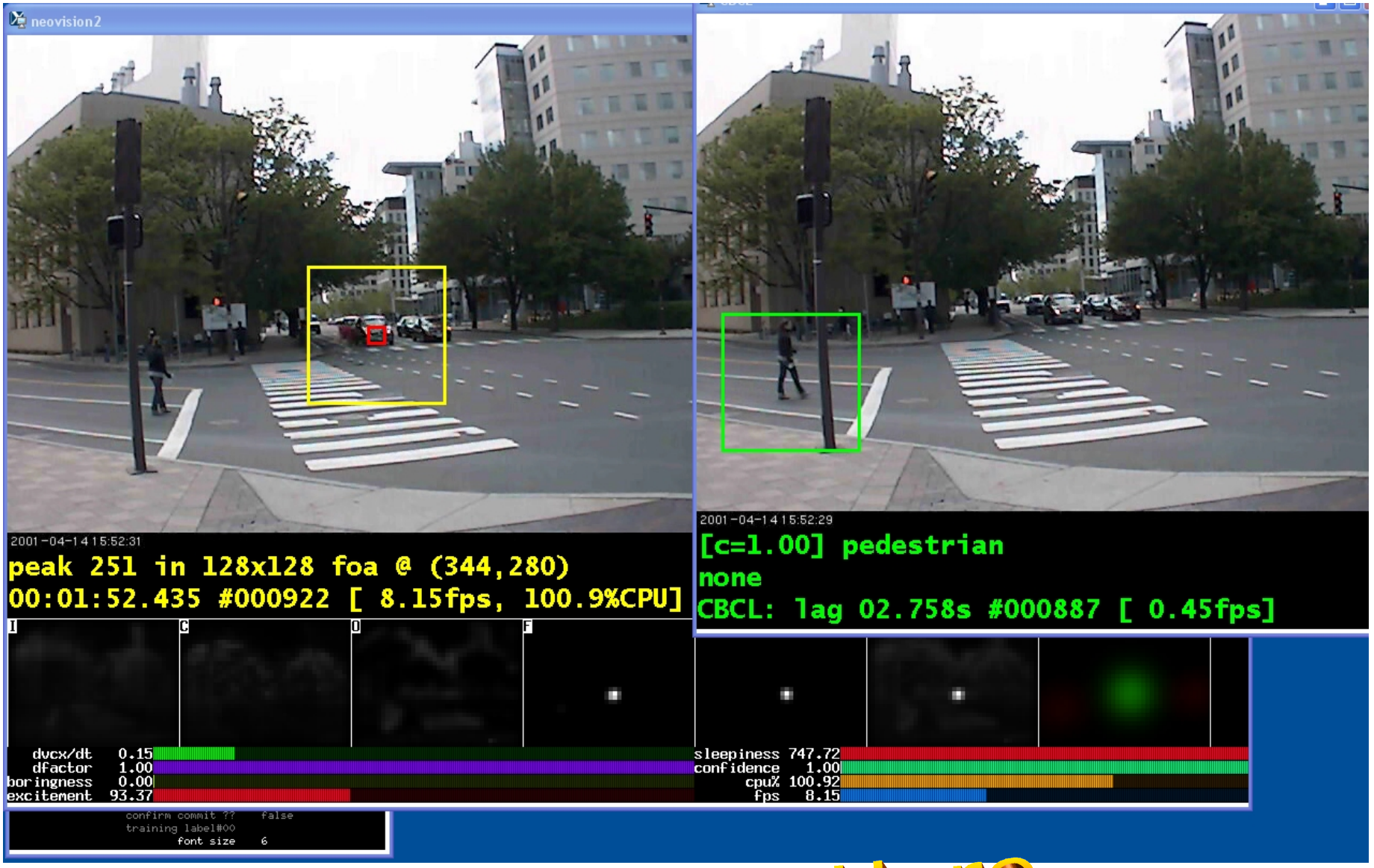
Multi-threaded implementation

- The HMAX(CBCL) algorithm consists of a series of split/merge steps
- The response to each patch(filter) can be computed in parallel




Timing (C1)
12
10
8
6
4
2
0
100
200
300
400
500
600
CPU(MT)
MATLAB


Timing (C2)
50
40
30
20
10
0
100
200
300
400
500
600
CPU(MT)
MATLAB



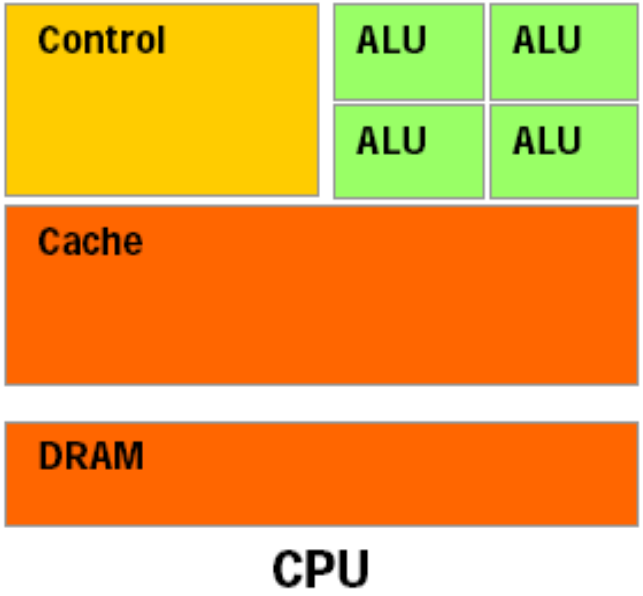
Start Movie Here

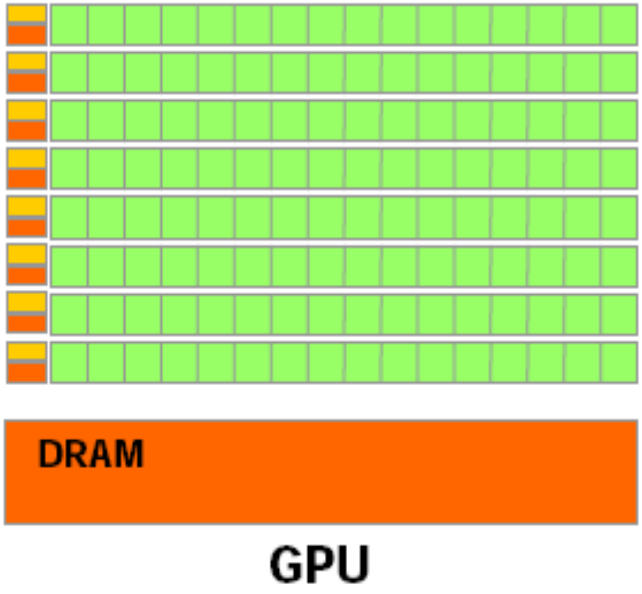
Outline

- **Introduction**
  - Motivation
  - Computational model of the ventral stream
- **Multi threaded implementation**
- **CUDA Implementation**
- **Comparison**
- **Conclusion**

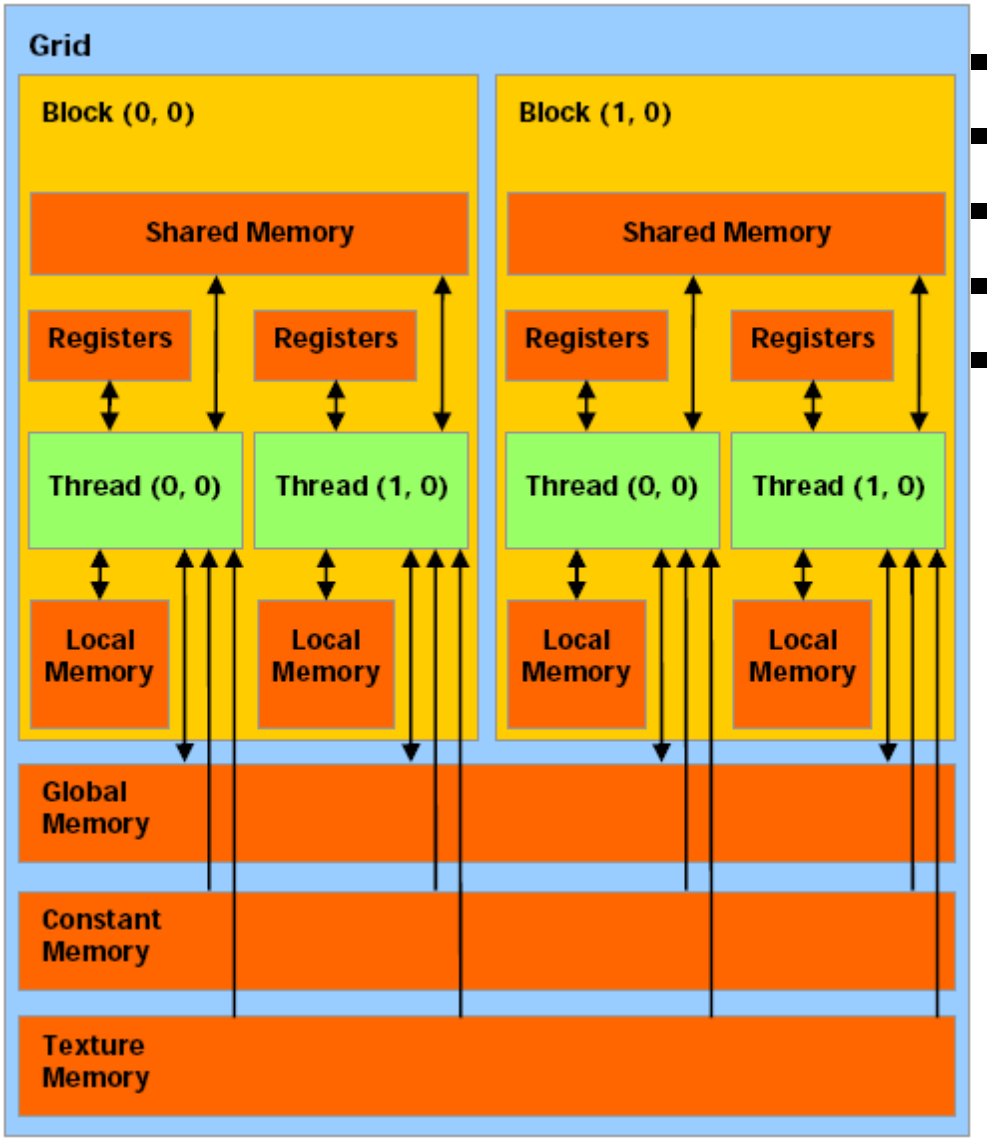
GPU architecture

- CPU:
  - Existing CPUs contain ~10 cores with larger shared memory
  - Memory hierarchy is transparent to the program
- GPU:
  - The GPU consists of ~100 cores with small shared memory
  - Memory hierarchy is exposed and has to be exploited



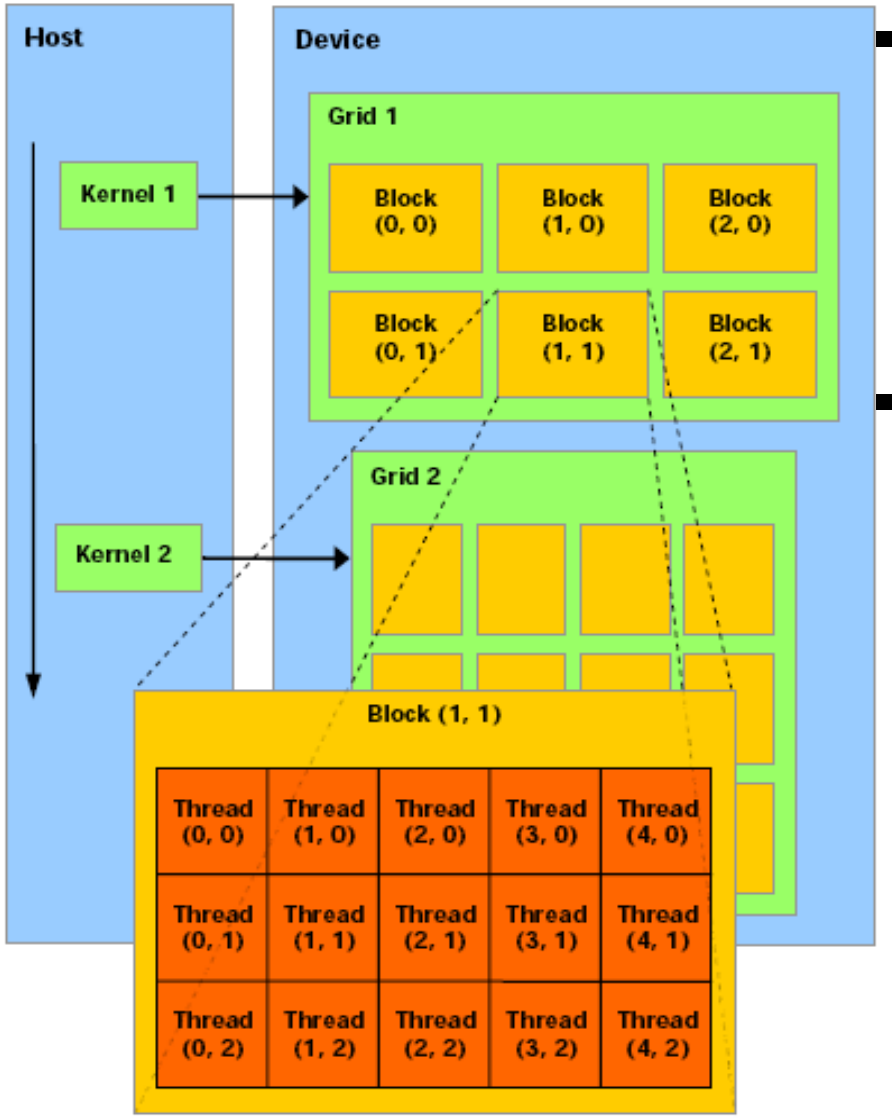


Fine grained memory access



- Registers: Thread R/W
- Shared : Block R/W
- Global : Grid R/W
- Constant : Grid R
- Texture : Grid R

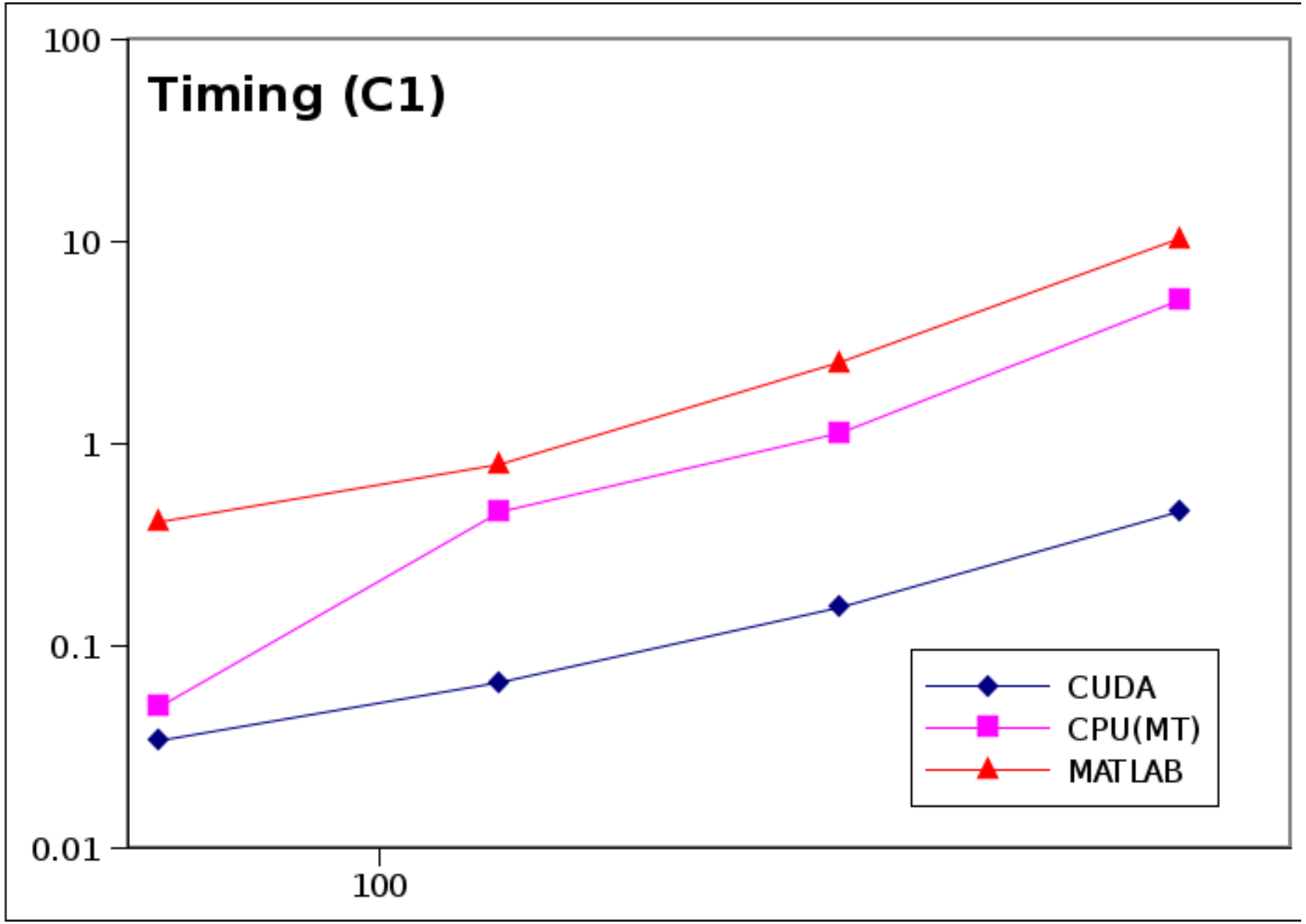
Execution model

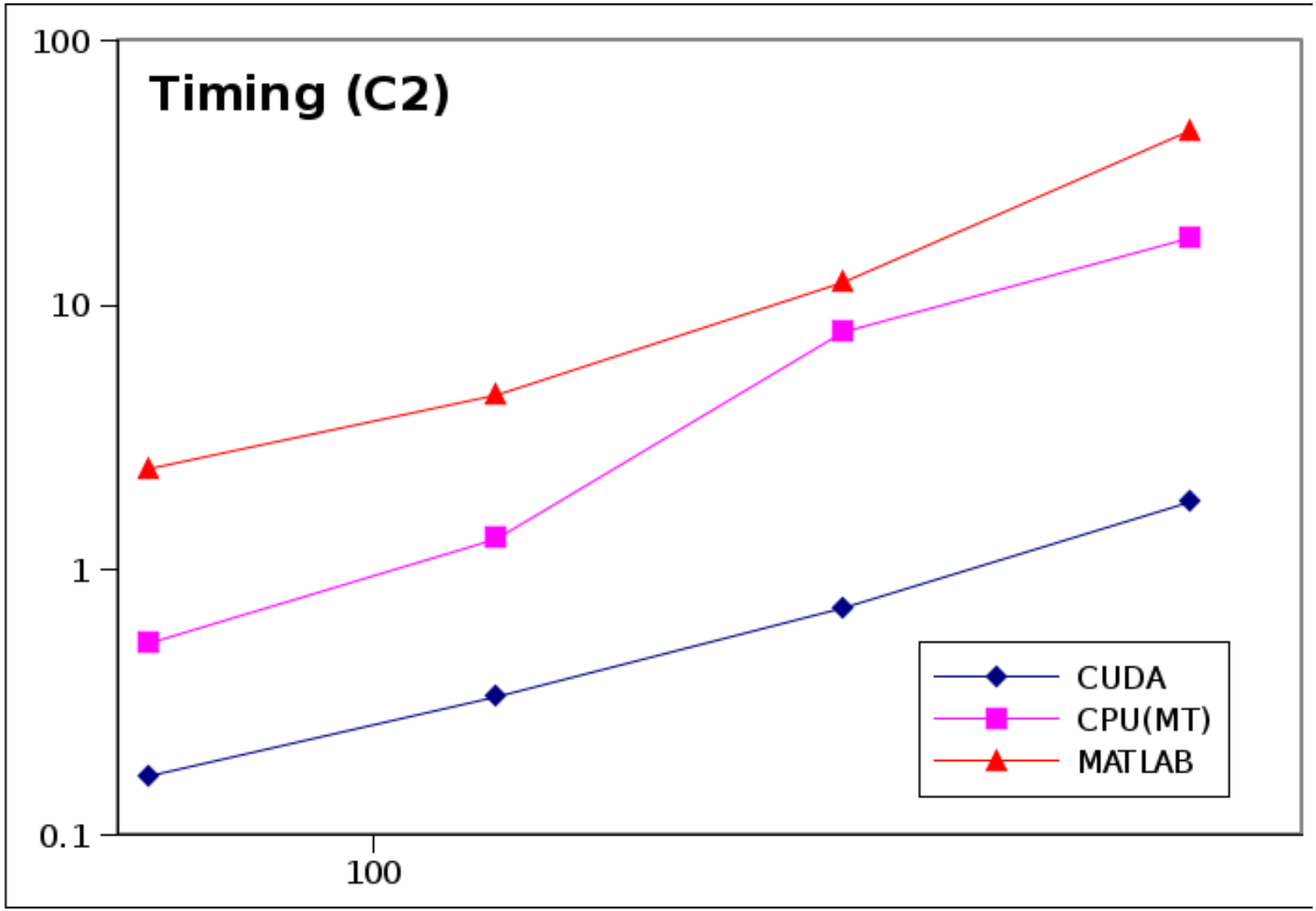


- CPU (MT)
  - Code to be executed has to be put in a function
  - Function executed on a per thread basis
  - ~10 threads
- GPU:
  - Code to be executed on the GPU has to be put in a kernel
  - SIMD type execution
  - Kernel is invoked per thread basis in no specified order
  - BlockIdx, threadIdx provide thread and block identity
  - ~500 threads

Implementation strategy

- **Decomposition**
  - Each filter assigned to a block
  - Each row assigned to a thread
- **Strategy**
  - Transfer input to gpu texture memory
  - Allocate output on gpu global memory
  - Assign each filter to a block to a distinct block
  - Divide the rows among the blocks
  - Each thread writes directly to the output


Timing (C1)
100
10
1
0.1
0.01
100
CUDA
CPU(MT)
MATLAB


Timing (C2)
100
10
1
0.1
100
CUDA
CPU(MT)
MATLAB

Lessons learned

- **New programming strategy**
  - Kernel loading
  - Order of execution not guaranteed
- **Communication overhead is large**
- **Respect memory hierarchy**
- **Read forums when documentation is sparse!**

Thank you
for your attention!

C1	CUDA	CPU (MT)	MATLAB
64	0.03	0.05	0.41
128	0.07	0.46	0.79
256	0.16	1.13	2.53
512	0.46	5.14	10.39
C2	CUDA	CPU (MT)	MATLAB
64	0.17	0.53	2.41
128	0.33	1.31	4.57
256	0.72	7.91	12.18
512	1.81	17.91	45.56