Systems Science Friday Noon Seminar Series

Systems Science

5-13-2011

Hardware Acceleration of Inference Computing: The Numenta HTM Algorithm

Dan Hammerstrom
Portland State University, dwh@pdx.edu

Hardware Acceleration of Inference Computing – The Numenta HTM Algorithm

Dan Hammerstrom

Electrical And Computer Engineering

- My research program focuses on specialized computer architectures for performing complex inference in embedded applications
  - Massively parallel, based on sparse distributed data representations

- There are a number of different components:
  - New PSU / UCSB NSF sponsored project “CDI Inference at the Nano-scale”
  - The DARPA SyNAPSE program
  - The Intel “non-Boolean” computing project
  - Collaboration with Jeff Hawkins at Numenta on hardware architectures for accelerating Numenta’s HTM

- The goal of this talk is to describe the HTM algorithm and our work on it

Motivation

Computer Architecture

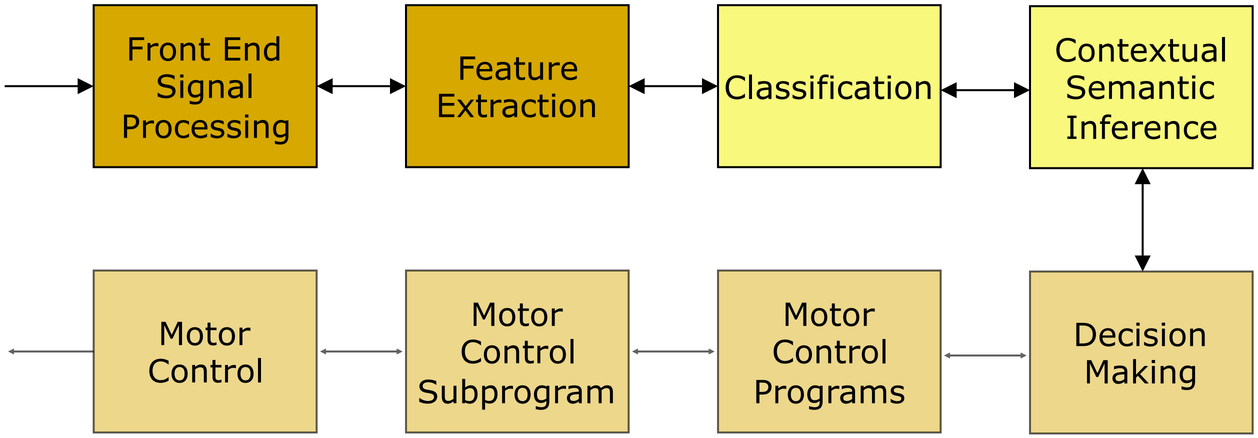
- Computer Architecture is a stagnant field
- “It is a solved problem, Intel solved it ...”
- The current textbook is over 20 years old
- It needs a new mission, radical new computational models, something …
- This work represents one attempt to breath some life into it

- Embedded is the future of computing
  - Computers are becoming more ubiquitous and integrated into ever more everyday applications
- Embedded applications generally involve inferring the state of the environment from sensed data
  - There are massive quantities of noisy, ambiguous data
  - Systems need to make decisions based on such data
- Consequently, inference computing, making sense of these data, will become an ever increasingly important component
  - Intel’s context aware computing
- **I personally believe that some kind of Inference Engine will be the microprocessor of the 21st Century and will enable a wide range of embedded computing applications**

A Prototypical Model of Intelligent Computing

- Although an over-simplification, this diagram characterizes most intelligent computing implementations
- The term Intelligent Signal Processing (ISP) has been used to describe algorithms and techniques that involve the creation, efficient representation, and effective inference over complex models of semantic and syntactic data

The "Front End" - **DSP**　　The "Back End" - **ISP**



Front End Signal Processing

Feature Extraction

Classification

Contextual Semantic Inference

Motor Control

Motor Control Subprogram

Motor Control Programs

Decision Making

- The Front End
  - Well understood, it is the realm of traditional digital signal and image processing
  - Often consists of applying the same computation over large arrays of elements, computation is data parallel, and communication tends to be local

- The Back End
  - No good solutions
  - In the early days of computing, “Artificial Intelligence” focused on the representation and use of contextual and semantic information
  - Knowledge was generally represented by a set of rules
  - However, these systems were “brittle,” exhibiting limited flexibility, generalization, and graceful degradation
  - And they were usually “hand” made and unable to adapt dynamically (i.e., learn) within the context of most real world applications

One Possibility Is Study How Cortex Does This

From *Big Brain* by Gary Lynch and Rick Granger (Palgrave McMillan 2008):

- “… the ‘front end’ circuits of the brain … specialize in their own particular visual and auditory inputs, the rest of the brain converts these to random-access encodings in association areas throughout cortex. … these areas take initial sensory information and construct grammars

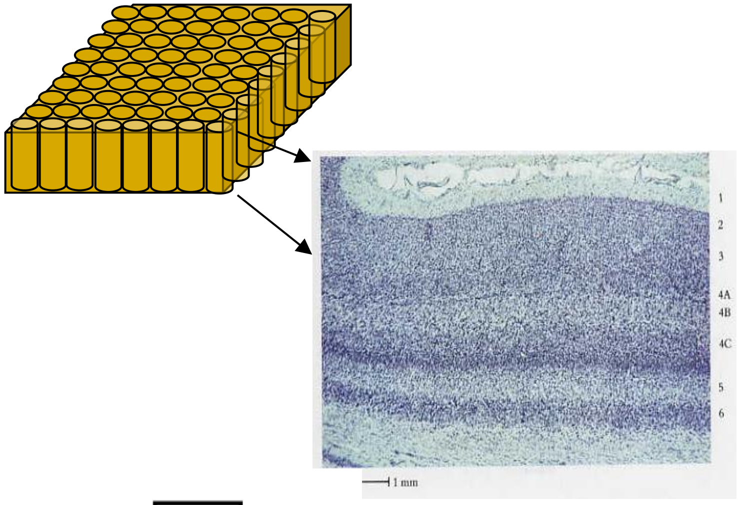
- “These are not grammars of linguistic elements, they are grammatical organizations (nested, hierarchical, sequences of categories) of percepts – visual, auditory, …

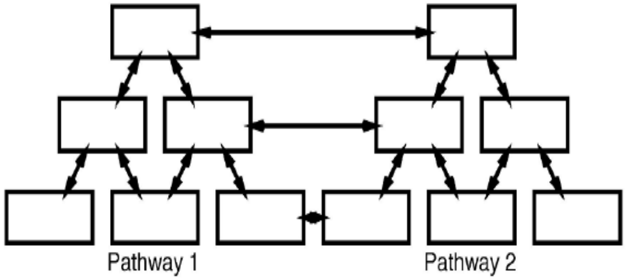
- “Processing proceeds by incrementally assembling these constructs … these grammars generate successively larger ‘proto-grammatical fragments,’ eventually constituting full grammars”

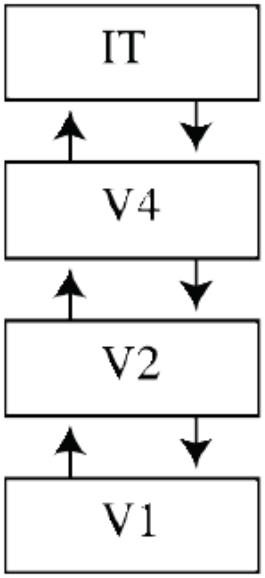
- “They thus are not built in the manner of most hand-made grammars; they are statistically assembled, to come to exhibit rule-like behavior, of the kind expected for linguistic grammars

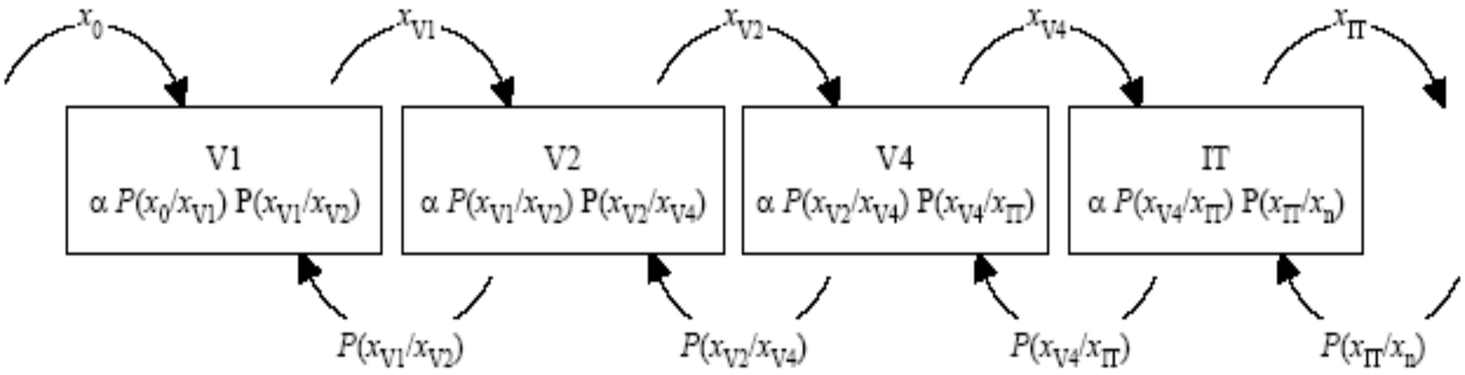
General Characteristics of Cortex

- Scales
- Distributed Representation
  - Sparse Coding
- Modular Structure
- Remarkably uniform anatomical 6-layer structure
- Connectivity leads to a hierarchic functional structure
  - Most likely Bayesian/Probabilistic Inference in the hierarchy








x_0
x_{V1}
x_{V2}
x_{V4}
x_{IT}
V1
$\alpha\, P(x_0/x_{V1})\, P(x_{V1}/x_{V2})$
V2
$\alpha\, P(x_{V1}/x_{V2})\, P(x_{V2}/x_{V4})$
V4
$\alpha\, P(x_{V2}/x_{V4})\, P(x_{V4}/x_{IT})$
IT
$\alpha\, P(x_{V4}/x_{IT})\, P(x_{IT}/x_h)$
$P(x_{V1}/x_{V2})$
$P(x_{V2}/x_{V4})$
$P(x_{V4}/x_{IT})$
$P(x_{IT}/x_h)$

Lee and Mumford’s Visual cortex model

Sparse Distributed Representations

- An very important, if not the most important characteristic we intend to exploit is to use sparse distributed data representations, where each coding unit participates in multiple distinct representations
  - A representation has a more "statistical" aspect to it by virtue of the ensemble of vectors in its representation
  - Fault tolerant, graceful degradation, incremental learning ...
  - Fast, parallel, local processing

- "What is the goal of sensory coding?" David Field, *Unsupervised Learning*, eds G. Hinton, T. Sejnowski, MIT Press 1999, discussed sparse distributed codes
  - In a sparse distributed code, all cells in the code have an equal response probability across all representations
  - But a low response probability for any single representation
  - The dimensionality of the space spanned by the input vectors is not necessarily reduced

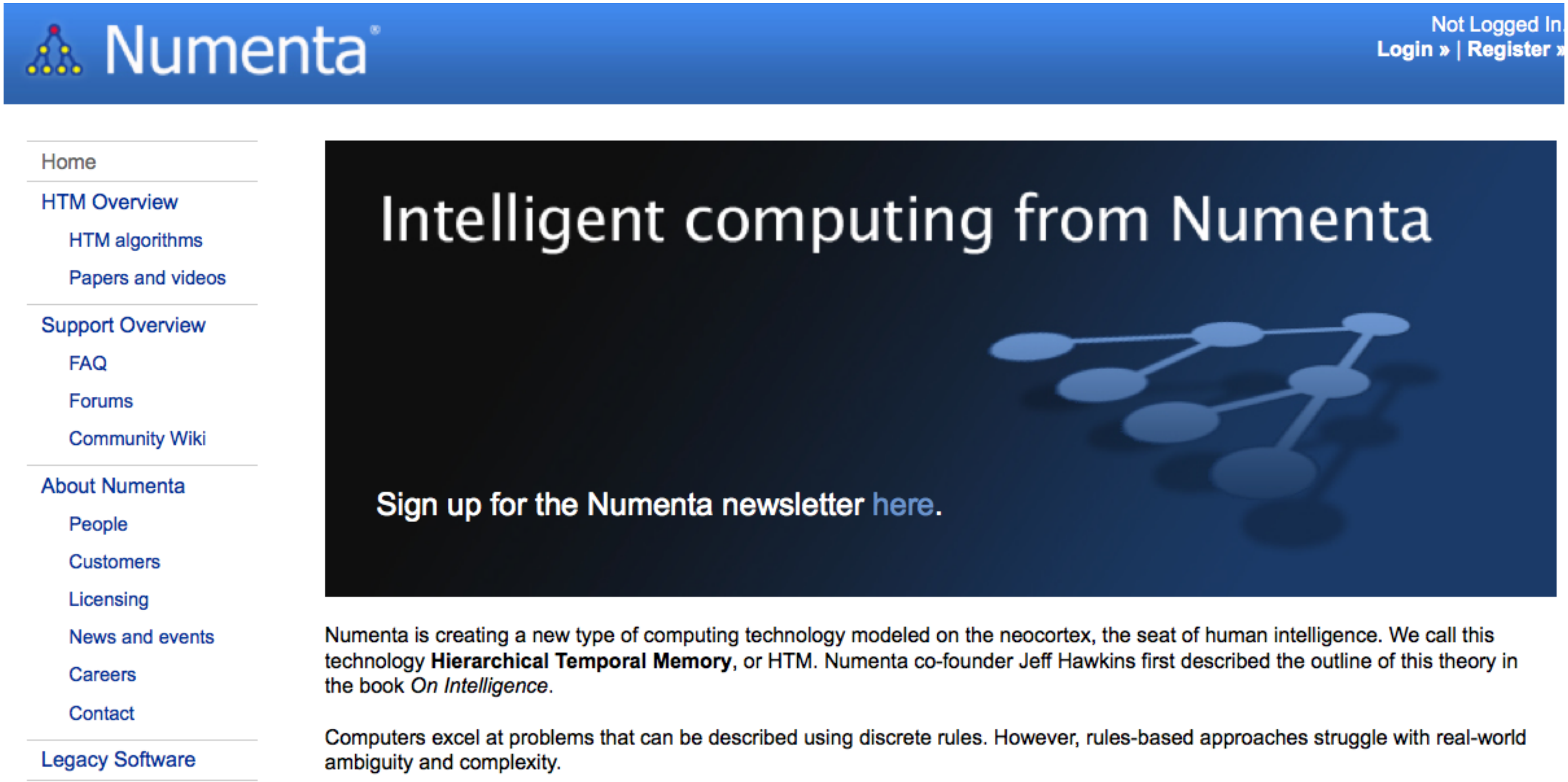
- In fact, "Sparsification" is not the same as reducing dimension:
  - "Compact coding" represents data with a minimum number of units
  - "Sparse distributed coding" represents data with a minimum number of active units
  - "Sparse Coding with an Overcomplete Basis Set: A Strategy Employed by V1?", Olshausen BA, Field DJ (1997). *Vision Research, 37*: 3311-3325

Associative Memories Approximate What We Are Looking For

- Associative Memories approximate Bayesian Inference and are particularly effective when using sparse distributed representations – Perhaps the single most important contribution of biological systems
- They are highly parallel, and have minimal precision requirements
- They do well with both low precision digital or analog implementations, so straight forward implementation is possible in analog CMOS
- Preliminary analysis indicates that reasonably robust implementations are possible at the nanoscale, e.g., memristor, CMOL (Likharev), time-varying components "memcapacitor"
- However, they do have scaling limits
- This is another motivation for a modular structure – where modular, hetero-associative (BAM-like) structures enable scaling

- The latest version of HTM embodies these characteristics …

The HTM Algorithm



Intelligent computing from Numenta

Sign up for the Numenta newsletter here.

Numenta is creating a new type of computing technology modeled on the neocortex, the seat of human intelligence. We call this technology **Hierarchical Temporal Memory**, or HTM. Numenta co-founder Jeff Hawkins first described the outline of this theory in the book *On Intelligence*.

Computers excel at problems that can be described using discrete rules. However, rules-based approaches struggle with real-world ambiguity and complexity.

Jeff Hawkins / Numenta

- Jeff founded the Redwood Neuroscience Institute, http://redwood.berkeley.edu
- From which has emerged a synthesis of a number of existing and new ideas of cortical operation
- The models work well enough that he has now spun out a company, Numenta, Inc., www.numenta.com
- The George / Hawkins model starts with a fairly general Bayesian module, very similar to the AM presented earlier
  - "A Hierarchical Bayesian Model of Invariant Pattern Recognition in the Visual Cortex," D. George and J. Hawkins, *Proceedings of the ICJNN 2005*
- These modules then are combined into a hierarchy to form the Numenta *Hierarchical Temporal Memory* (HTM)

- Their latest work, "Hierarchical Temporal Memory including the HTM Cortical Learning Algorithm," Version 0.1, November 2010, Numenta – a newer version that makes greater use of sparse distributed representations
  - This presentation will focus on the information in this paper, including quotes and figures

- It is an algorithm that embodies the good things we seek for our hardware architectures:
    - Hierarchy
    - Low precision
    - Sparse distributed data representations
    - Associative memory
    - Does inference through space and time
- Very biological in nature – Jeff is constantly referring back to how neurons do things for inspiration

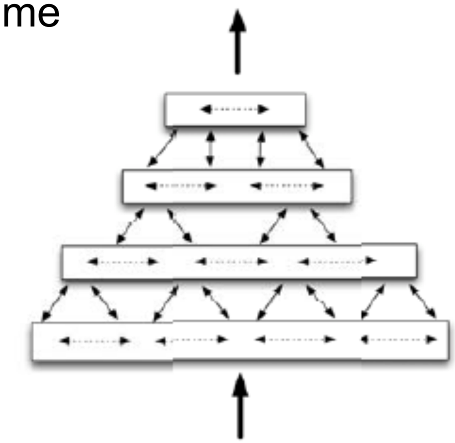


Figure 1.1: Simplified diagram of four HTM regions arranged in a four-level hierarchy, communicating information within levels, between levels, and to/from outside the hierarchy

Description

- Discussion is of one region for one level of processing
- Assume a 2D square input field
- Inputs are binary
- The input feeds a layer of “columns,” assume 1 per input data point
- Each column has some number of cells, let’s assume 4, but it can be from 1 to any reasonable number (hundreds …)
- There are two major phases:
  - Spatial Pooling, where each column looks at a receptive field under it, assume the receptive field is square and is centered under the column
  - Temporal Pooling, each cell looks at the other cells around it (again within in some local neighborhood)

Spatial Pooling (SP)

- The operation is similar to image convolution, but each column's spatial field mask is unique (they start out random, but are modified by learning)
- The weight mask is also binary, but there are some other fields associated with each synapse
  - Connect bit, potential synapse bit, permanence (multi-bit field)

- There are two phases:
  - Each column computes an activation value by convolving the mask and the matching underlying input data field (Large Neighborhood Convolution, LNC)
  - A localized k-WTA is done to ensure that only a small number of nodes are actually on – this leads to a sparse representation

Temporal Pooling (TP)

- The next phase is temporal pooling and involves the individual cells in each of the columns
- Each cell in a column has a set of “dendritic segments” (DS)
- Each DS has a weight mask, as with spatial pooling, the masks are initially random, but change and adapt through learning
- If a column is active after the Spatial Pooling phase and if any cells are active (“predictive”) from the previous data presentation, they remain active – this means that they were predictive of this spatial pooling operation
- If a column is active, but none of the cells are in the predictive state from the previous operation, then they are all activated
- If the column is not active, then any cells that were in the predictive state from the previous presentation are turned off

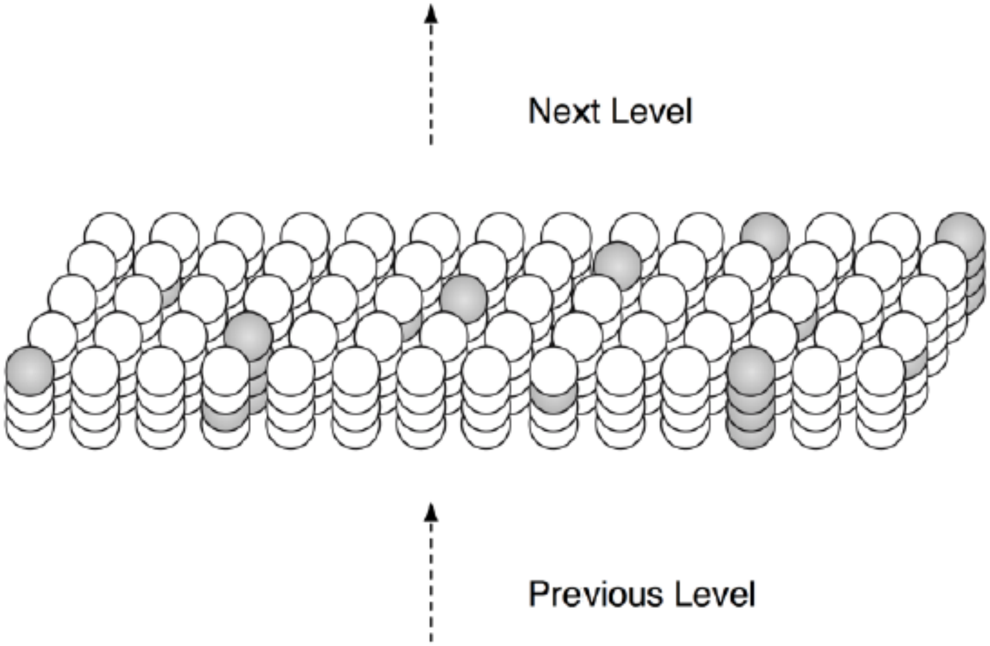


Figure 2.2: By activating a subset of cells in each column, an HTM region can represent the same input in many different contexts. Columns only activate predicted cells. Columns with no predicted cells activate all the cells in the column. The figure shows some columns with one cell active and some columns with all cells active.

.

From The Paper:

- Each column in an HTM region consists of multiple cells
- Each cell in a column can be active or not active
- By selecting different active cells in each active column, we can represent the exact same input differently in different contexts.
- Putting this all together, we make the following hypothesis
  - The neocortex has to do both first order and variable order inference and prediction
  - There are four or five layers of cells in each region of the neocortex
  - The layers differ in several ways but they all have shared columnar response properties and large horizontal connectivity within the layer
- We speculate that each layer of cells in neocortex is performing a variation of the HTM inference and learning rules described in the paper

- Now the DSs, for every cell in every column, perform their LNC operation, convolving their weight mask with the active cells in their local neighborhood that they are connected to
- Though no k-WTA is done, there is a fixed threshold that is applied to each DS, if any DS for a cell is above threshold, the cell is considered active for TP
- A cell with an active DS is put in the “predictive” state (whether in an active column or not)
- When a cell becomes predictive, it forms connections to a subset of the cells nearby that were active in a prior time step
- The cells in a column, as well as the SP value for the column constitute the output of the level and are the input to the next level
- The result is a sparse distributed data representation for both feed-forward (“spatial pooling”) and time sequence (“temporal pooling”) operations

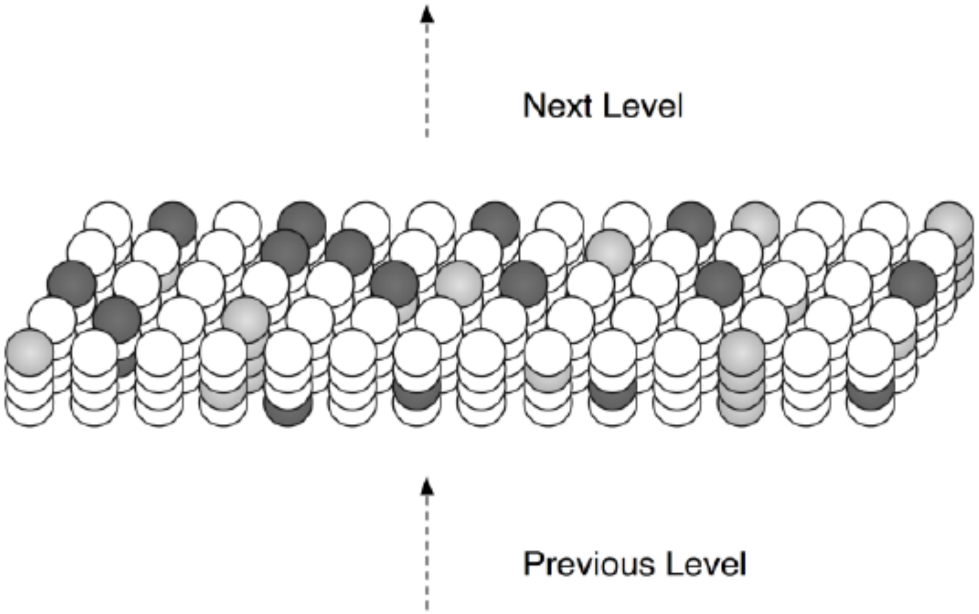


Figure 2.3: At any point in time, some cells in an HTM region will be active due to feed-forward input (shown in light gray). Other cells that receive lateral input from active cells will be in a predictive state (shown in dark gray).

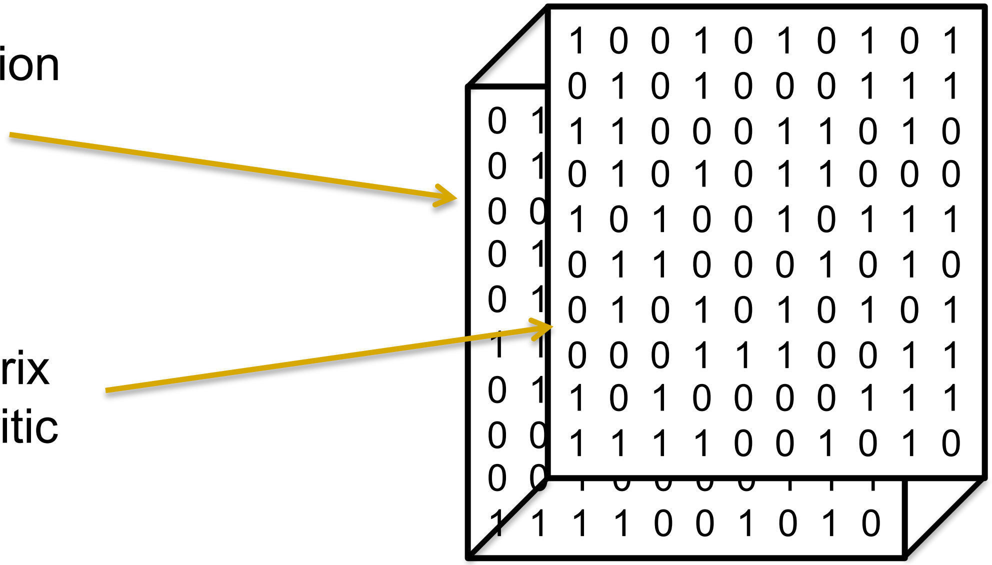
- In summary, when a new input arrives, it leads to a sparse set of active columns
- One or more of the cells in each column become active, these in turn cause other cells to enter a predictive state through learned connections between cells in the region
- The cells activated by connections within the region constitute a prediction of what is likely to happen next
- When the next feed-forward input arrives, it selects another sparse set of active columns
- If a newly active column is unexpected, meaning it was not predicted by any cells, it will activate all the cells in the columns
- If a newly active column has one or more predicted cells, only those cells will become active
- The output of a region is the activity of all cells in the region, including the cells active because of feed-forward input *and the cells active in the predictive state*
- Predictions are not just for the *next time step; predictions in an HTM region can be for several time steps into the future*

- Learning is too complex to go into detail here
- But it is Hebbian, where synapses are made (“connected”) if there is concurrent input and output activity, that is, the synapse is predictive
- This is effected by the use of the “permanence field,” which is incremented on coincident activity, it gets decremented by virtue of permanence normalization in the weight mask
- There is a thresholding of the permanence fields

- Temporal pooler learning adds some complexity that is more detail than I have time to go into right now

Large Neighborhood Convolution – The Compute Intensive Inner Loop

- Convolve neighborhood input region with weight matrix
- Neighborhoods are large (a radius of 20 is typical), very sparse, and binary



Neighborhood region
In input array

Mask / weight matrix
For a single dendritic segment

1	0	0	1	0	1	0	1	0	1
0	1	0	1	0	0	0	1	1	1
1	1	0	0	0	1	1	0	1	0
0	1	0	1	0	1	1	0	0	0
1	0	1	0	0	1	0	1	1	1
0	1	1	0	0	0	1	0	1	0
0	1	0	1	0	1	0	1	0	1
0	0	0	1	1	1	0	0	1	1
1	0	1	0	0	0	0	1	1	1
1	1	1	1	0	0	1	0	1	0

Hardware Acceleration
Short Term

Outline of Multi-core Project – Ryan Price

The first phase:

- Implement a single process version of the full HTM CLA in C++.
- Verify proper functioning of implementation using Numenta provided data.
- Design a pattern recognition task suitable for analysis of the single-process and parallel implementations.
- Benchmark the single process implementation on the pattern recognition task using Intel's VTune.

Outline of Multi-core Project

The second phase consists of:

- Identify key hotspots for parallelization effort.
- Implement a parallel version of the code using multi-threading, Open MP.
- Analyze the parallel version running on multiple cores using VTune.
- Compare the multi-core runs and the benchmark.

The PC used for analyzing the parallel version will have a 6-core processor with 12GB RAM.

Digit Sequences Task

Digit sequences made up from digits from the MNIST Handwritten Character Dataset

10 different sequences of digits 0-9

Each digit is a unique example taken from MNIST dataset

Some of the sequences have shared subsequences

7 0 8 9 5 4 2 6 1 3

6 4 7 2 3 8 9 5 0 1

```
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000011100000000
0000000000111111111100000000
0000000000111111111100000000
0000000000111111111000000000
0000000000111000000000000000
0000000000111000000000000000
0000000000111000000000000000
0000000000111000000000000000
0000000000111111111000000000
0000000000111111111100000000
0000000000111111111110000000
0000000000011100001110000000
0000000000000000000110000000
0000000000000000000110000000
0000000000000000000110000000
0000000000000000011110000000
0000000111111111111110000000
0000000111111111111100000000
0000000111111111110000000000
0000000111111111000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
```

Data Set

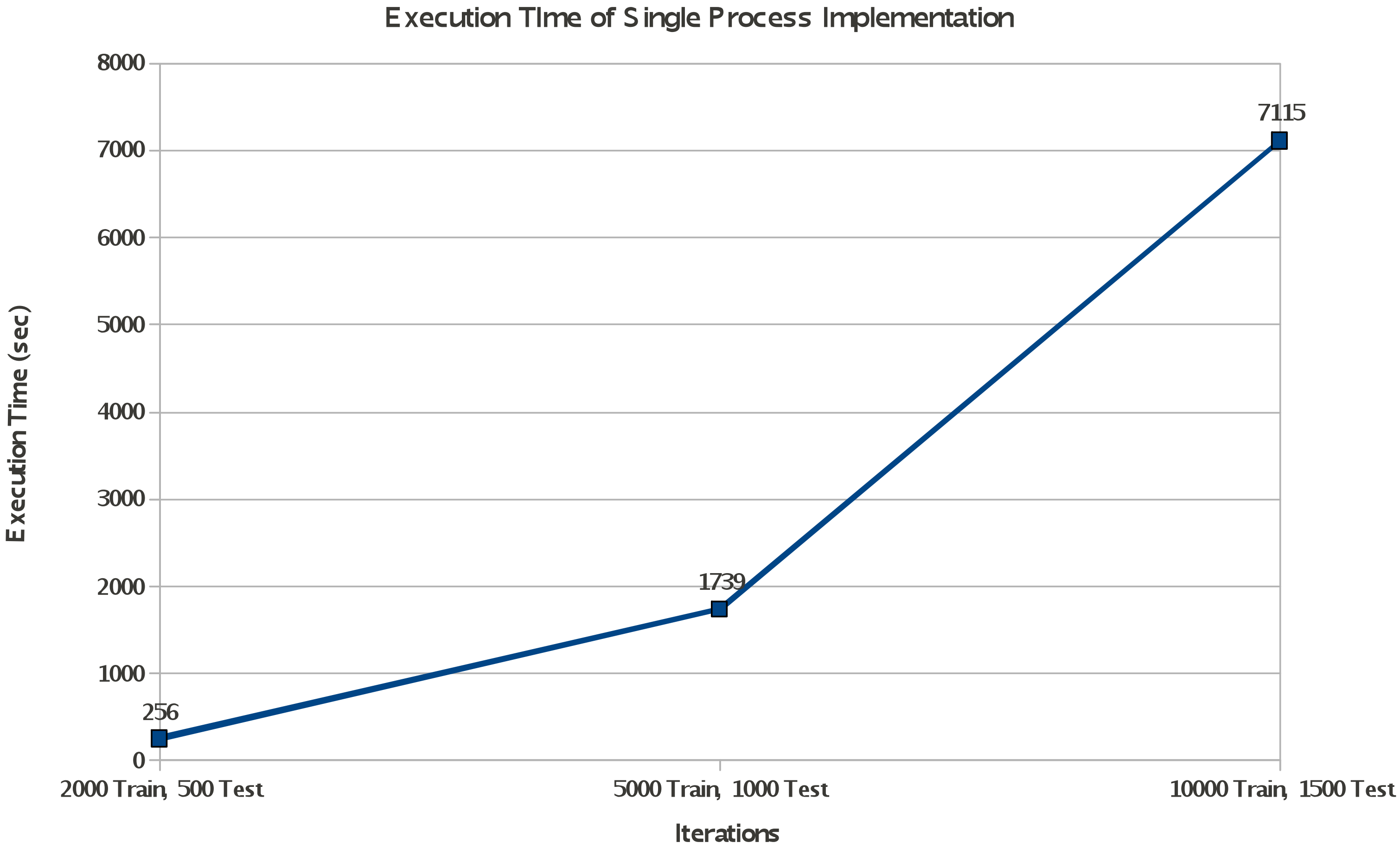
Analysis done using data sets of three sizes:

- 200 training sequences, 50 test sequences
- 500 train, 100 test
- 1000 train, 150 test

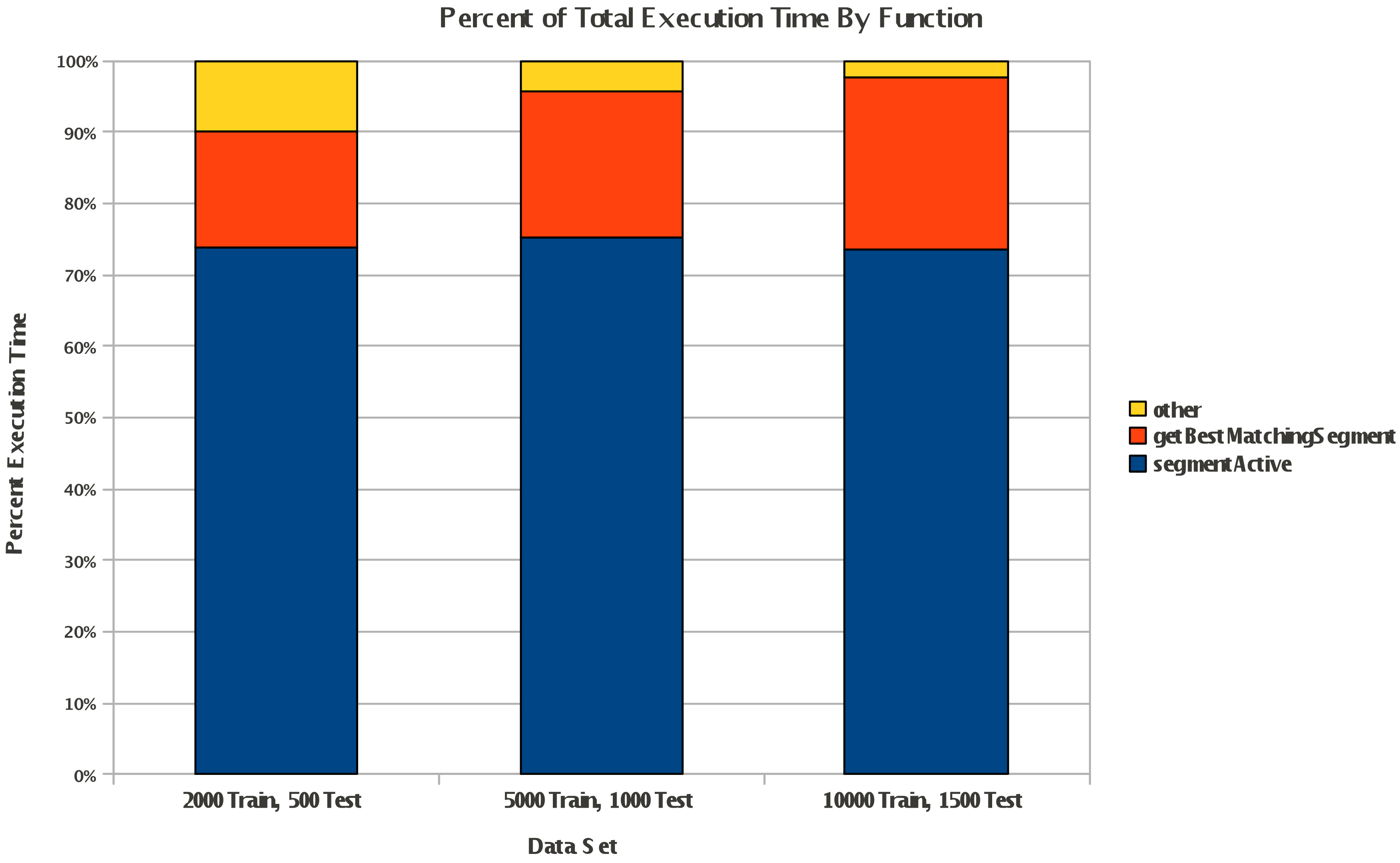
10 digits per sequence...

Total of 2,500, 6,000, and 11,500 iterations per respective data set

Single Process Baseline



Identifying Hotspots



Experience with the network suggests parameter selection may strongly influence execution time – particularly Temporal Pooling parameters

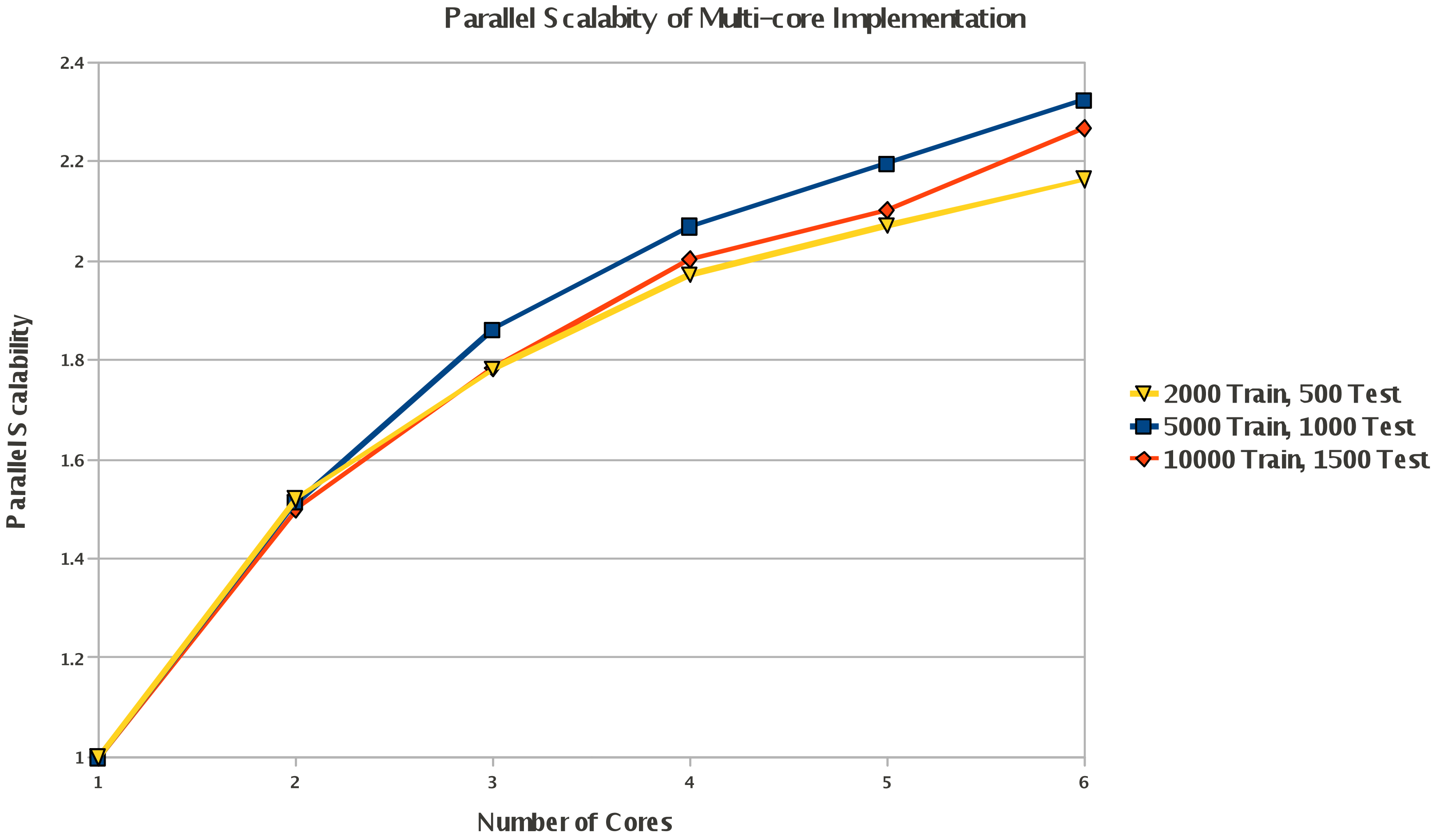
Baseline measurement suggests that the algorithms scale poorly with larger datasets – even with a multi-core implementation

Acceleration is needed

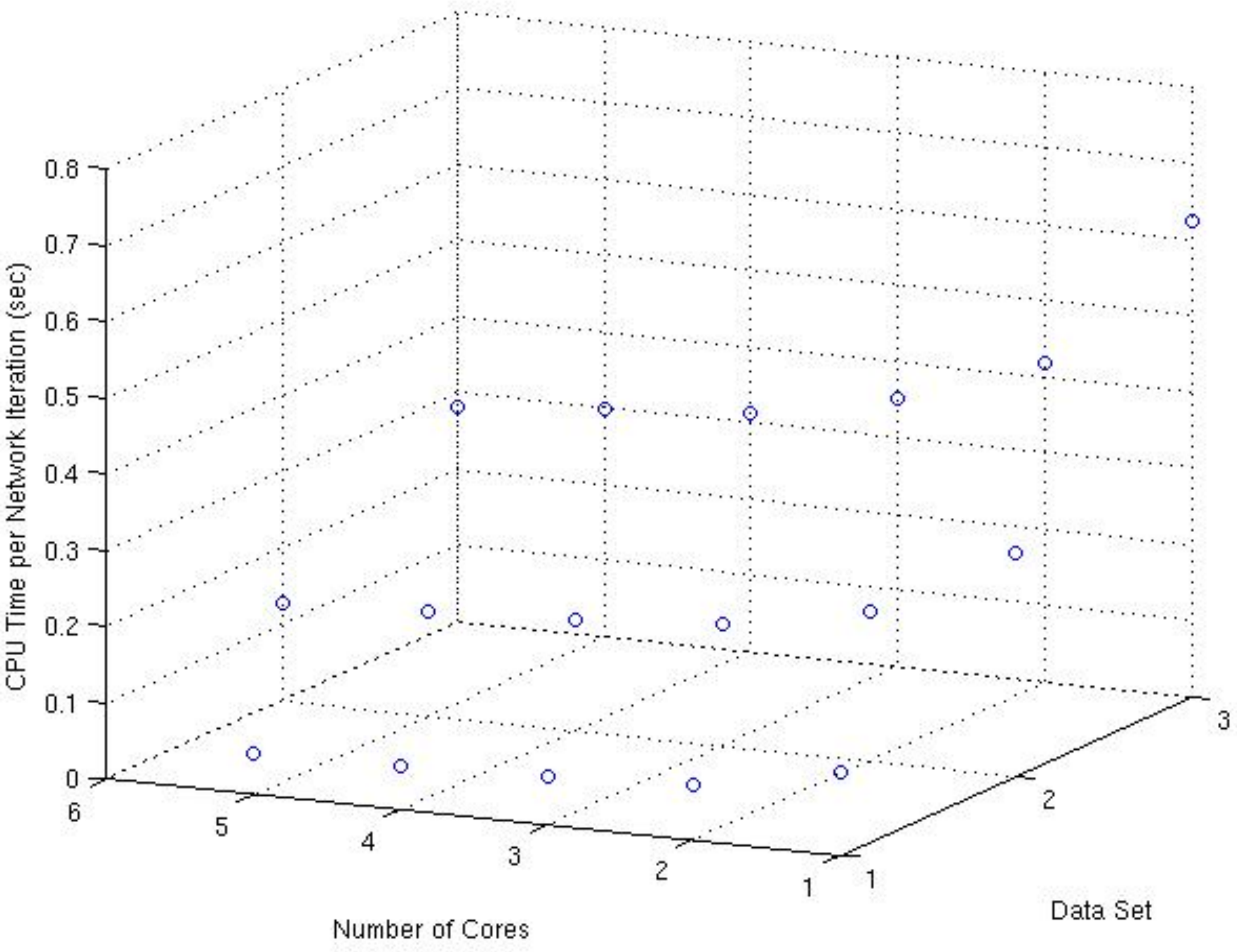
Profiling results show that 2 functions account for ~90%-98% of the total execution time

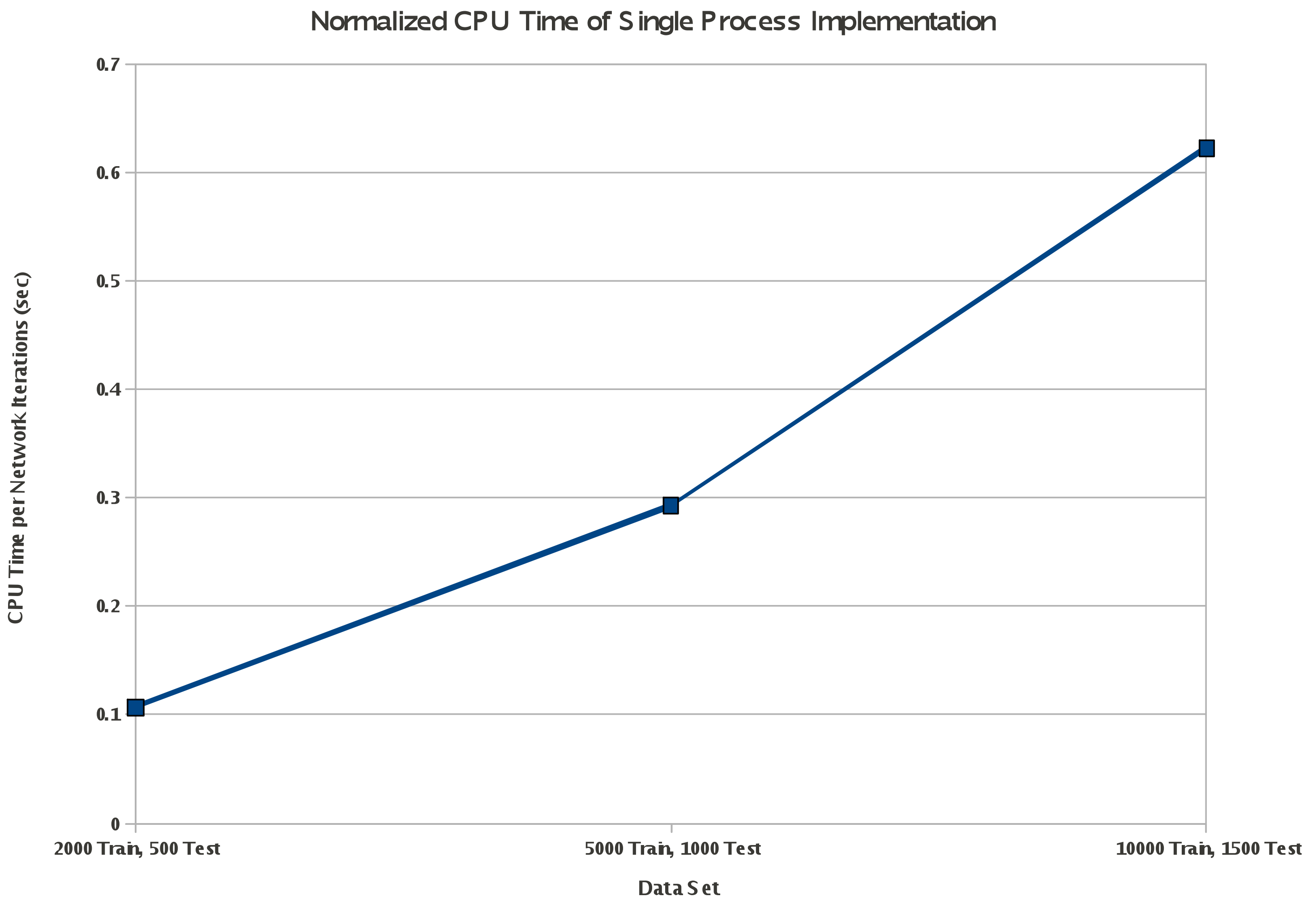
Parallelization effort focuses on these 2 hotspots

Preliminary Results



Note: parallel scalability is the ratio of serial version elapsed time and elapsed time on P processors


CPU Time per Network Iteration (sec)
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
6
5
4
3
2
1
Number of Cores
1
2
3
Data Set

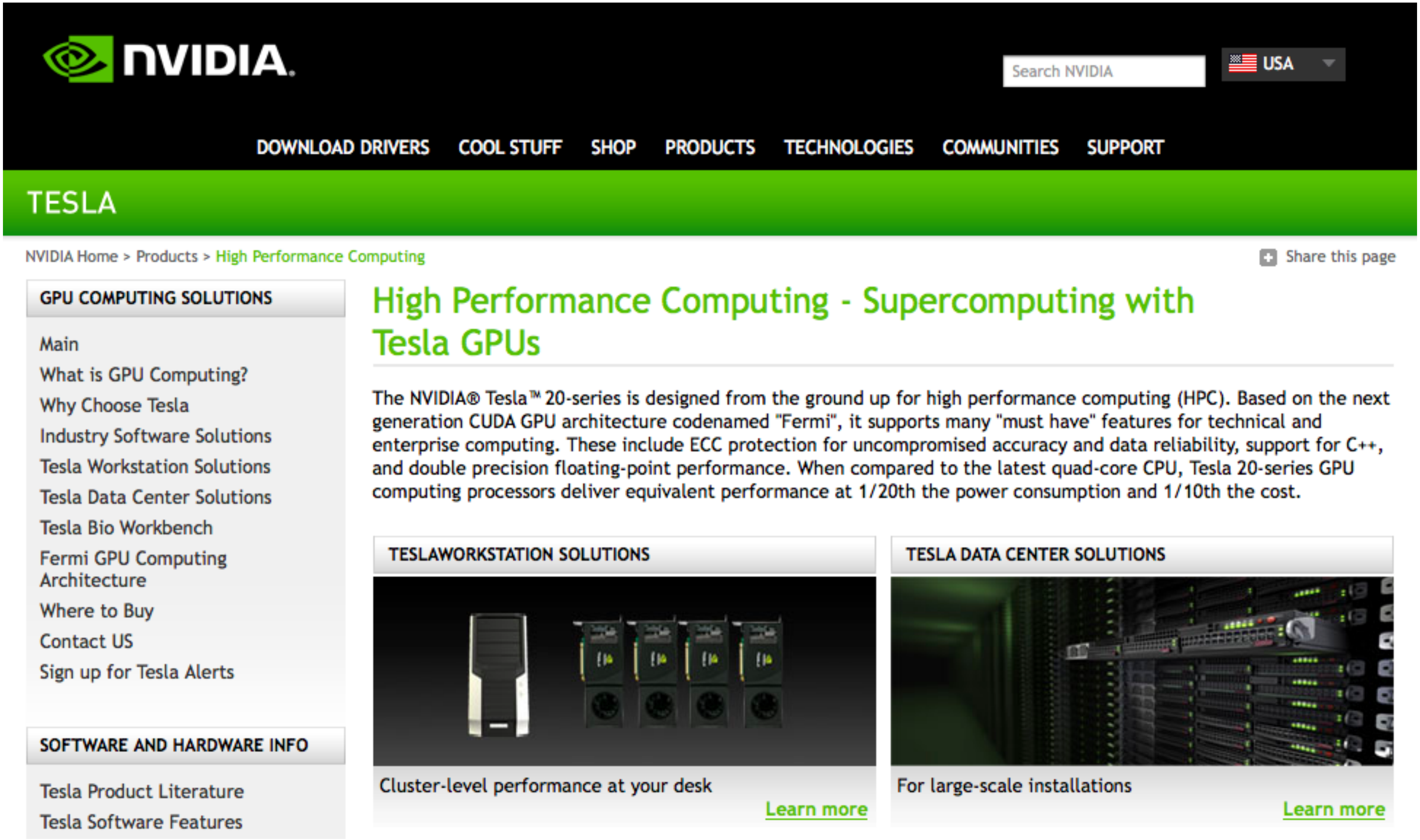

Normalized CPU Time of Single Process Implementation
CPU Time per Network Iterations (sec)
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
2000 Train, 500 Test
5000 Train, 1000 Test
10000 Train, 1500 Test
Data Set

Preliminary Results

Results suggest that modest speed-up is readily achieved on multi-core systems

Multi-core speedup can make HTM more feasible for moderate sized data sets

Significantly larger data sets may remain prohibitive without further acceleration or modifications to the algorithm



NVIDIA

Search NVIDIA

USA

DOWNLOAD DRIVERS COOL STUFF SHOP PRODUCTS TECHNOLOGIES COMMUNITIES SUPPORT

TESLA

NVIDIA Home > Products > High Performance Computing

Share this page

GPU COMPUTING SOLUTIONS

- Main
- What is GPU Computing?
- Why Choose Tesla
- Industry Software Solutions
- Tesla Workstation Solutions
- Tesla Data Center Solutions
- Tesla Bio Workbench
- Fermi GPU Computing Architecture
- Where to Buy
- Contact US
- Sign up for Tesla Alerts

SOFTWARE AND HARDWARE INFO

- Tesla Product Literature
- Tesla Software Features

High Performance Computing - Supercomputing with Tesla GPUs

The NVIDIA® Tesla™ 20-series is designed from the ground up for high performance computing (HPC). Based on the next generation CUDA GPU architecture codenamed "Fermi", it supports many "must have" features for technical and enterprise computing. These include ECC protection for uncompromised accuracy and data reliability, support for C++, and double precision floating-point performance. When compared to the latest quad-core CPU, Tesla 20-series GPU computing processors deliver equivalent performance at 1/20th the power consumption and 1/10th the cost.

TESLAWORKSTATION SOLUTIONS

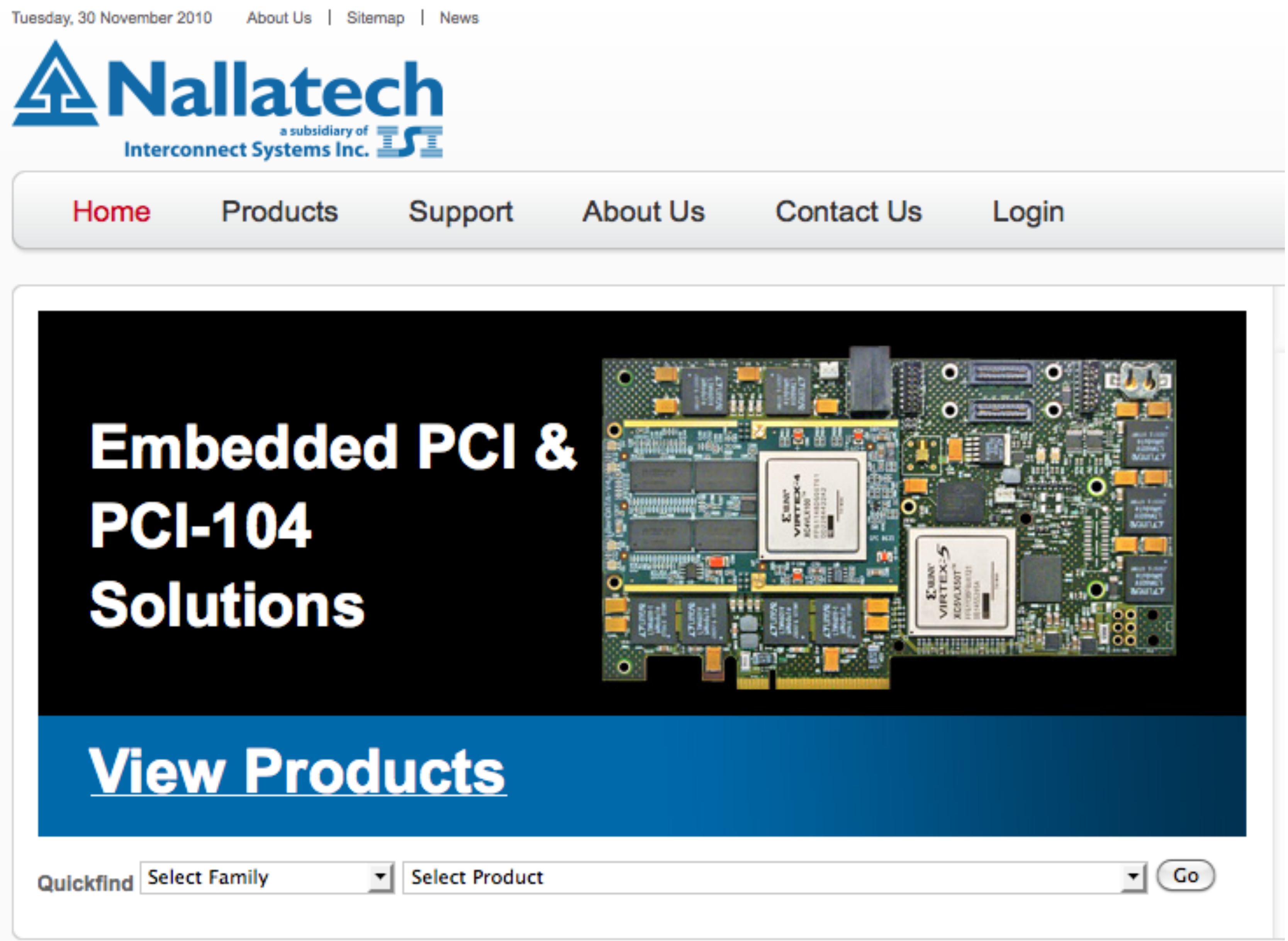
Cluster-level performance at your desk

Learn more

TESLA DATA CENTER SOLUTIONS

For large-scale installations

Learn more


Tuesday, 30 November 2010
About Us | Sitemap | News
Nallatech
a subsidiary of Interconnect Systems Inc. ISI
Home
Products
Support
About Us
Contact Us
Login
Embedded PCI & PCI-104 Solutions
View Products
Quickfind
Select Family
Select Product
Go

GPU and FPGA

- The algorithm is massively parallel with few dependencies, but it is very storage intensive
- The biggest inner loop is the Large Neighborhood Convolution (LNC) used in both spatial and temporal pooling
- Each weight is unique and is only used once, caching and on-chip memory won't help, though caching prefetch may help a little
- GPU:
  - Has the advantage of huge memory bandwidth, 144 GB/sec (with up to 6 GB on-board RAM) for the latest Tesla 2070, 1.3 Tflops/sec
  - Bit level operations mean that most compute hardware sits idle
- FPGA:
  - Fine grained parallel hardware
  - For external I/O the pins are there, but most current board implementations do not leverage the I/O for off chip memory access

- We are now working on both GPU and FPGA implementations of HTM

- We do not have enough data to assess potential speed up

- Implications for computer architecture
  - No caching, large sequential array access
  - Bit level computation, ultra low precision
  - Asynchrony
  - Error tolerance
  - Functional differentiation due to network weight profiles
  - Large, sparse connectivity

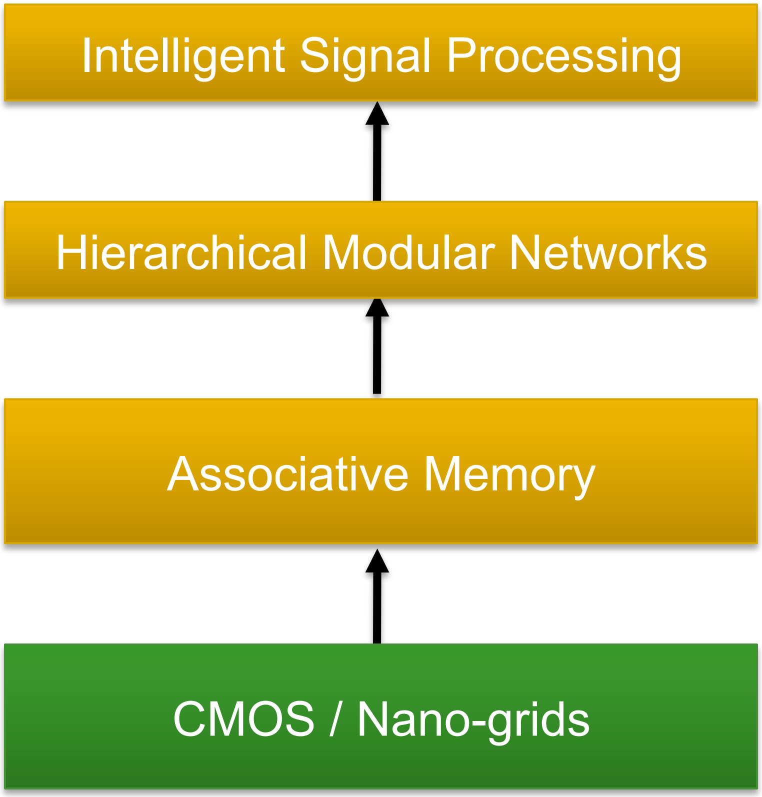
- Our goal is small scale, ultra-low power, portable applications

Hardware Acceleration Long Term

From molecules to ISP …

- One can then use associative memory structures, inspired by cortical circuits, to implement HTM
- Significant research into nanogrid structures (such as CMOL), which have the potential for implementing very dense associative data structures
- Now we are exploring the use of more radical devices to implement associative memory

- NSF CDI "Inference at the Nano-scale"
- Intel Non-Boolean Computing Project

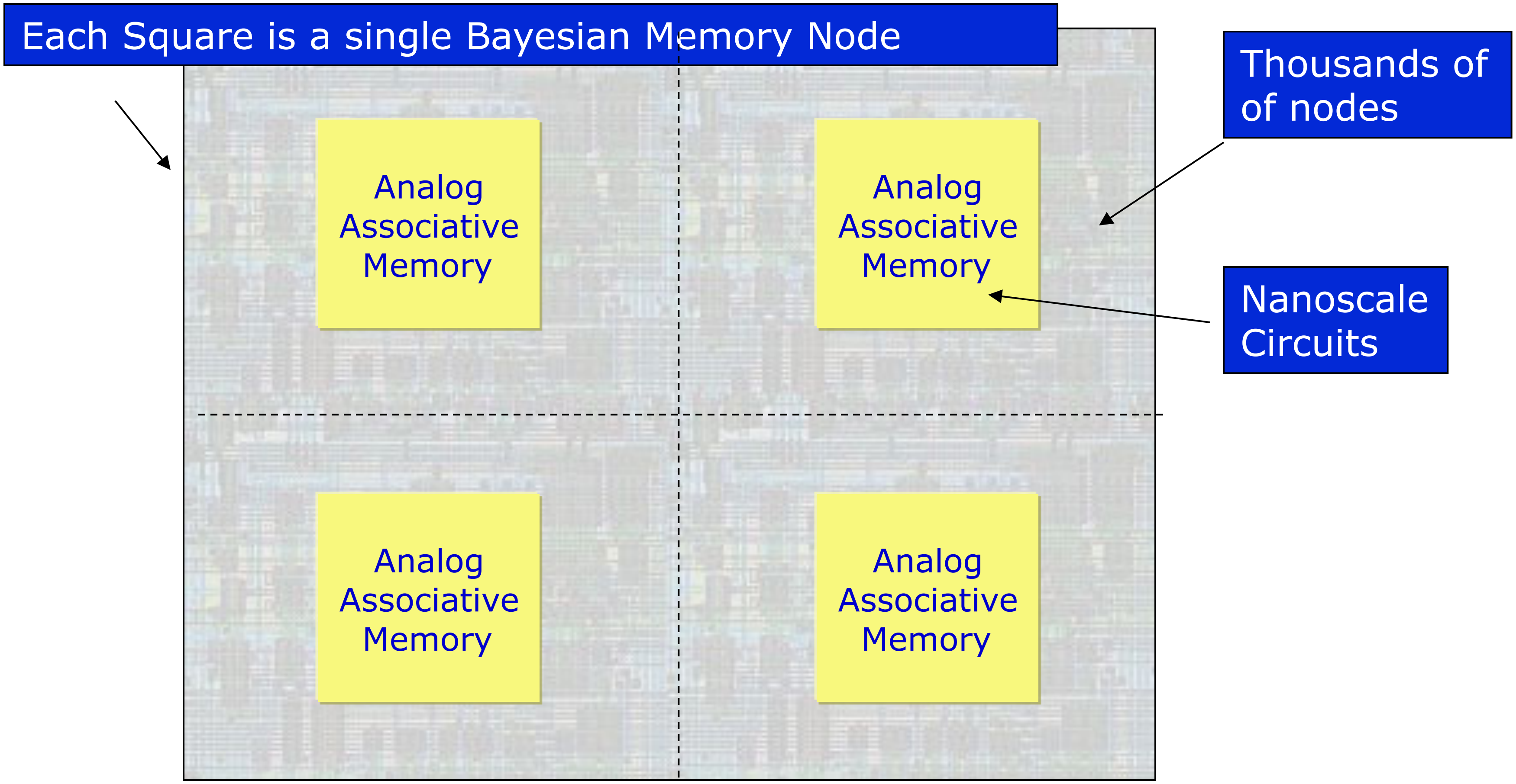

Intelligent Signal Processing

Hierarchical Modular Networks

Associative Memory

CMOS / Nano-grids

Our Goal – A Field Adaptable Bayesian Array (FABA)


Each Square is a single Bayesian Memory Node

Thousands of of nodes

Nanoscale Circuits

Analog Associative Memory
Analog Associative Memory
Analog Associative Memory
Analog Associative Memory

- Digitally multiplexed interconnect provides sparse inter-module connectivity, I/O, signal amplification
- Associative memory implementation in analog CMOS
- Can also be implemented in CMOL with memristor parameters in future versions for very high density computation

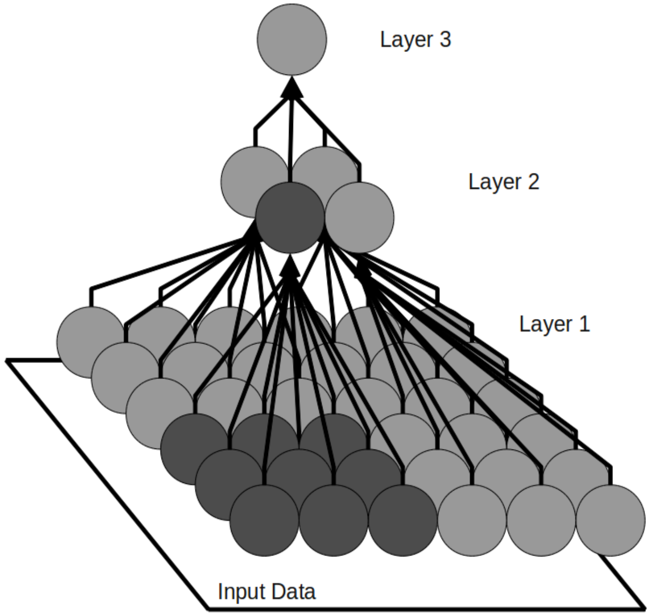
HTM Applications

HTM and Intelligent Signal Processing – Danny Voils

- Using Hierarchical Temporal Memory to do scale invariant object recognition using real world streaming images.
- Real time Images are captured using an open source, WIFI enabled robot.
- Image preprocessing is performed using a biologically inspired intelligent signal processing component.
- Relevant image features are extracted and presented to an HTM network where they are statistically analyzed for spatial and temporal patterns.
- The top layer of the HTM hierarchy combines pattern information to perform object classification.

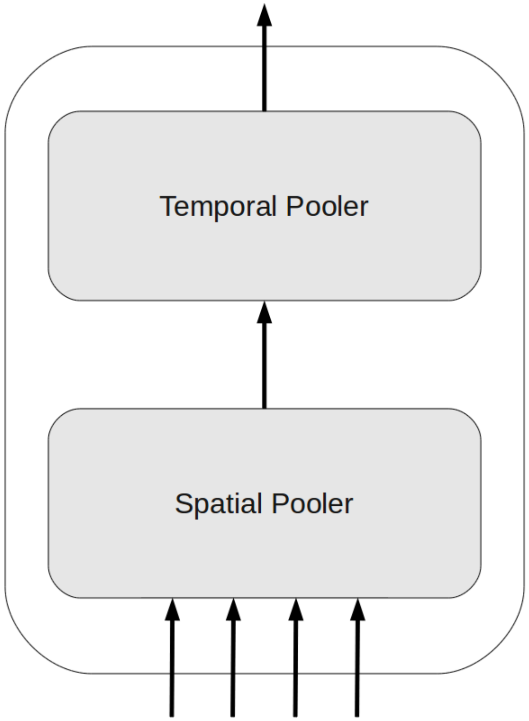
A Hierarchical Processing Network

- Processing nodes are arranged in a hierarchy with parent nodes in higher layers connected to child nodes in lower layers.
- Input data in the form of image features are presented to the bottom layer.
- Child outputs are concatenated together and sent up the the hierarchy.
- The top node performs the final classification step.

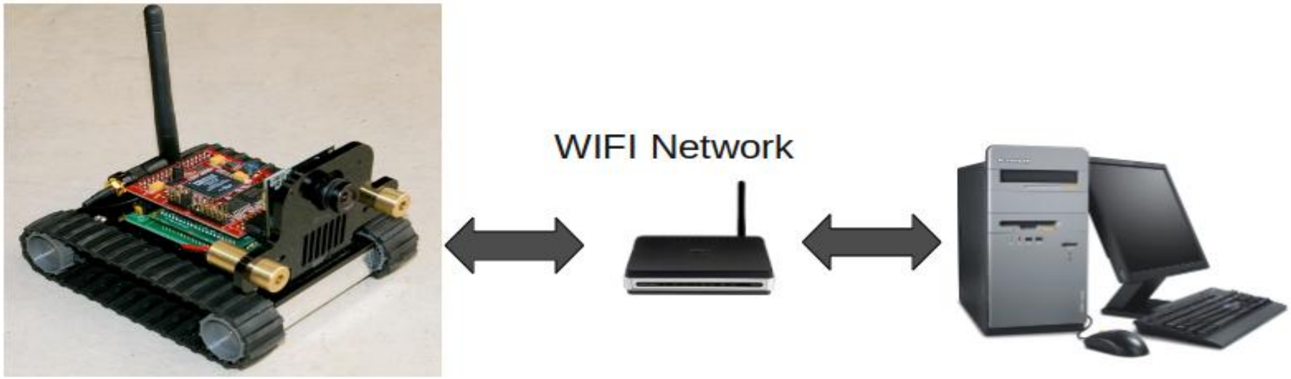


Processing Node Architecture

- Within each processing node, spatial and temporal poolers are used for learning and pattern analysis.
- After training, each input data maps directly to a quantization center using some distance metric - the index of the closest match is sent to the temporal pooler.
- The temporal pooler creates groups of temporally adjacent inputs from the spatial pooler.
- After training, inputs are mapped to a group and the group index is sent to higher layers of the hierarchy.



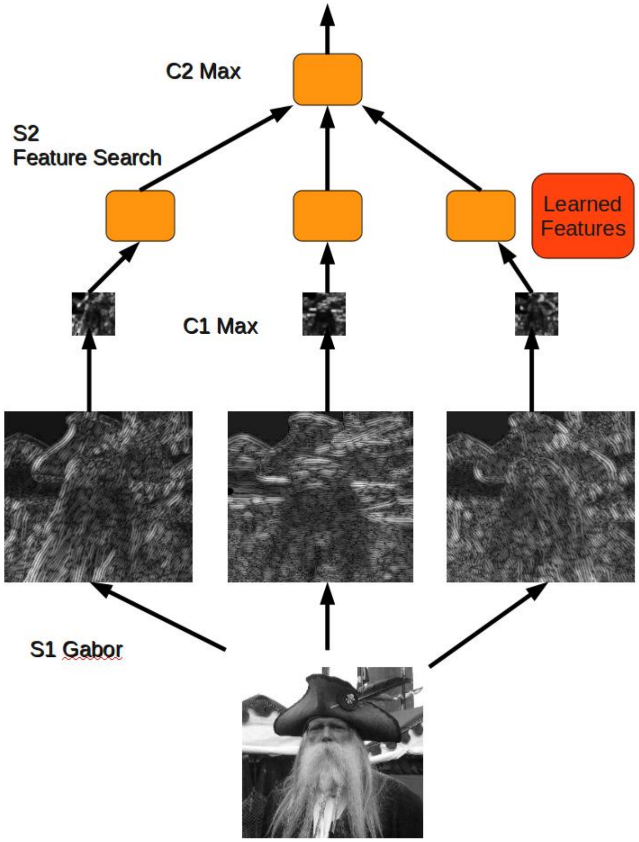
SRV1 Robot



- This robotic vision testbed is used to evaluate biologically inspired computing models.
- It streams video images in real time to a PC for processing.
- Movement commands are sent back to the robot for execution.

Using ISP for Image Feature Extraction

- Rather than expose the HTM network to the input image directly, it is first passed through pre-processing filters.
- These filters are patterned after structures thought to exist in visual cortex.
- The first layer performs localized Gabor wavelet transformations.
- Higher layers perform a localized maximum of inputs from lower layers.
- Features are combined in the highest layers forming an abstract representation of the input image.
- The HTM network can take input from any of these layers, but which one is best?



HTM in power systems analysis and control:

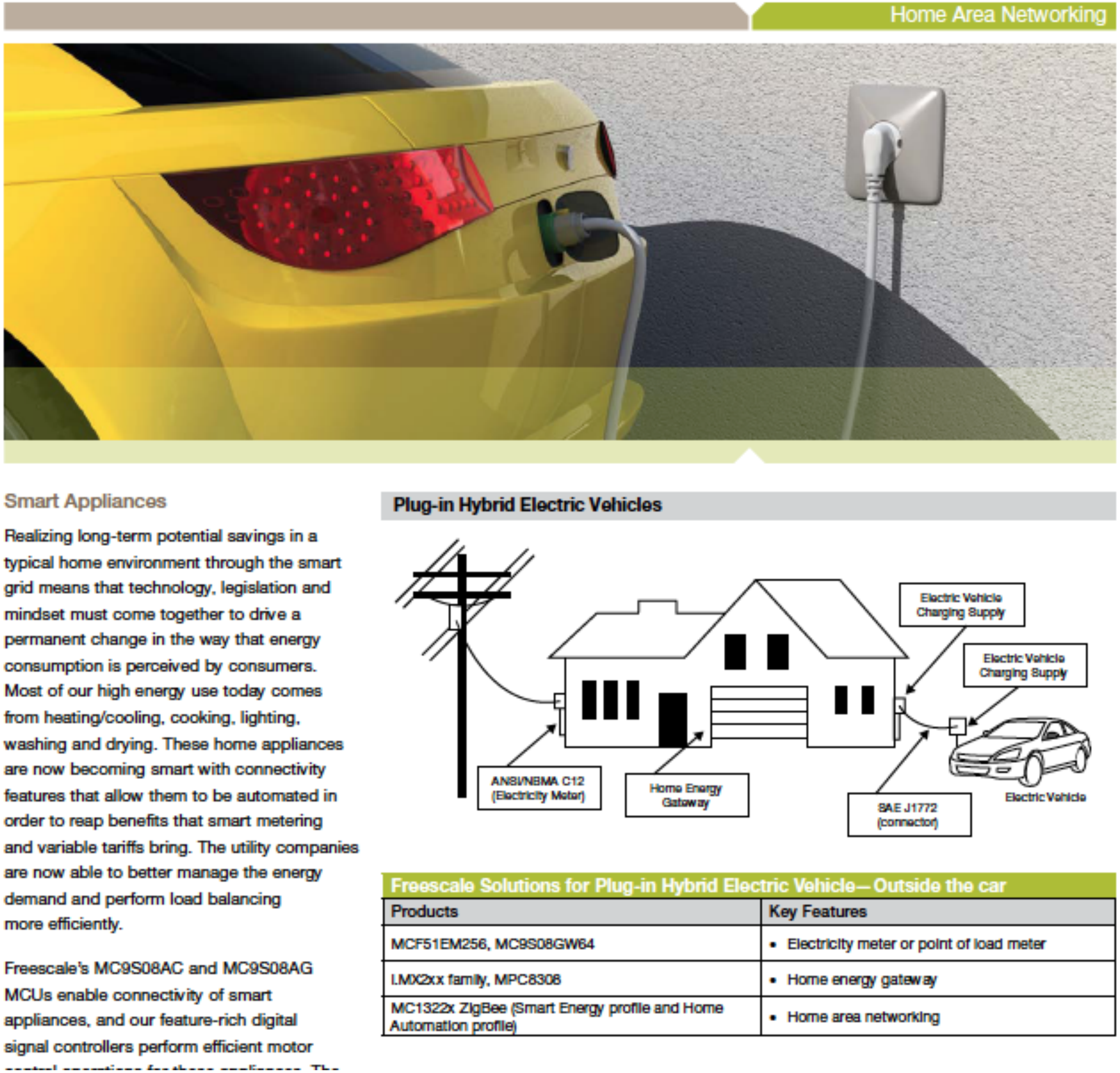
“Self-Learning” Power Analytics System Approved In Factory Acceptance Test for Offshore Oil Platform

- **SAN DIEGO, Calif. – April 20, 2010** – EDSA (www.edsa.com), the leading developer of power analytics solutions for the design, testing, and management of complex electrical power systems, today announced the successful completion of the Factory Acceptance Test for a groundbreaking new electrical power analytics, supervision and diagnostic system, scheduled to be deployed in a 40,700 barrel-per-day oil field in the North Sea. When deployed, it will be the first facility to deploy an autonomous power network using EDSA’s Paladin Live software in conjunction with **Hierarchical Temporal Memory (HTM) from Numenta Inc.**

- “The combination of EDSA’s power analytics and Numenta’s HTM technologies is the first of the coming generation of automated, self-learning systems for managing complex, enterprise-wide infrastructure… something so complex and so dynamic, that humans simply could not do it,” said Kevin Meagher, Chief Technology Officer of EDSA. “These technologies together offer unprecedented benefits for every organization with mission-critical power needs. We are extremely pleased with the synergies that have melded EDSA and Numenta into a strong and innovative partnership.

A Smart Grid Appliance: The Electric Vehicle

- Embedded computing
- Motor control, power electronics
- Battery management / smart power usage
- Hybrid systems
- Other vehicle systems
- It’s not just about communicating data!
- It’s also about analyzing the data and controlling systems
- Safety / Autonomous systems



Smart Appliances

Realizing long-term potential savings in a typical home environment through the smart grid means that technology, legislation and mindset must come together to drive a permanent change in the way that energy consumption is perceived by consumers. Most of our high energy use today comes from heating/cooling, cooking, lighting, washing and drying. These home appliances are now becoming smart with connectivity features that allow them to be automated in order to reap benefits that smart metering and variable tariffs bring. The utility companies are now able to better manage the energy demand and perform load balancing more efficiently.

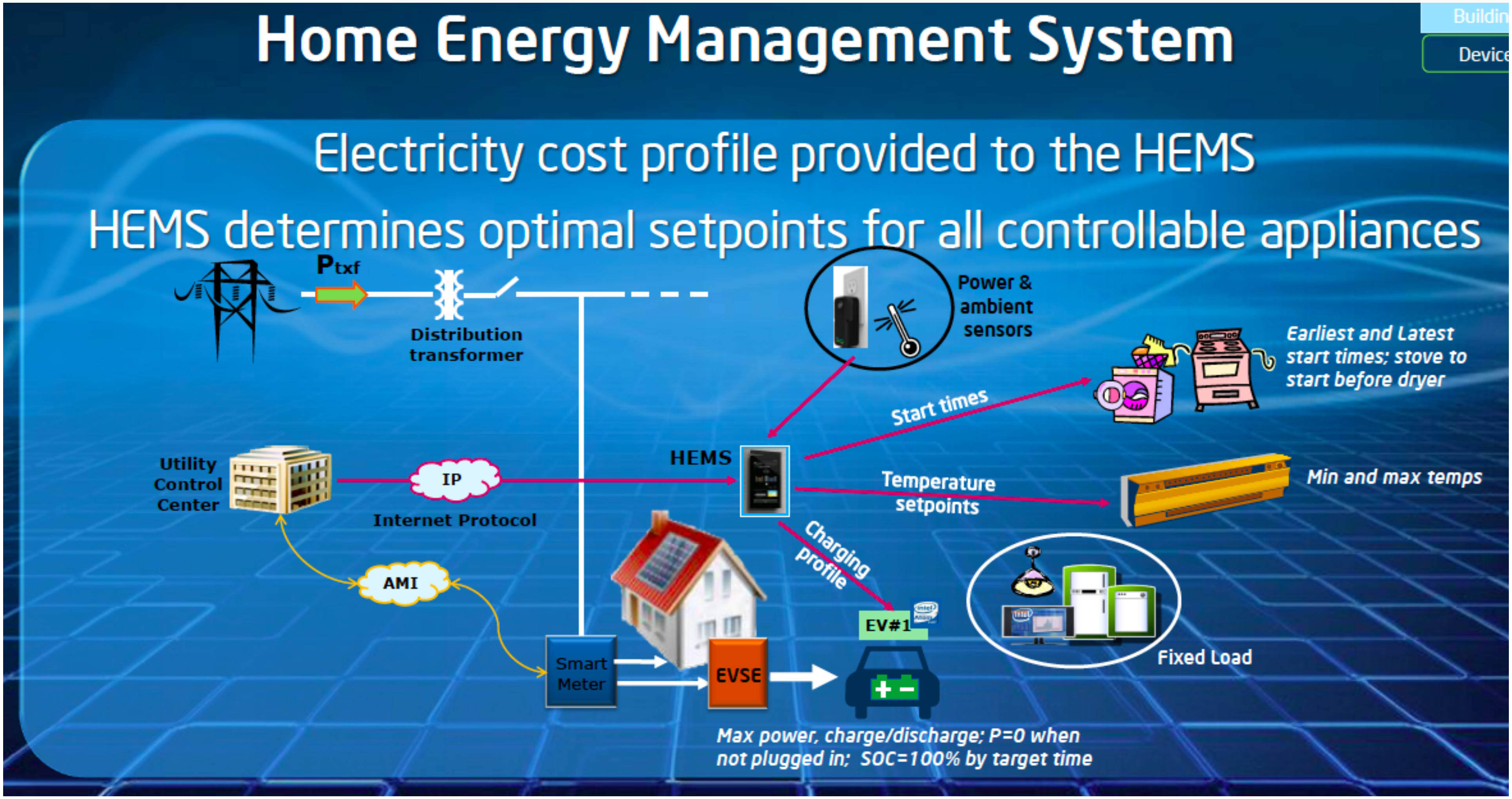
Freescale's MC9S08AC and MC9S08AG MCUs enable connectivity of smart appliances, and our feature-rich digital signal controllers perform efficient motor

Plug-in Hybrid Electric Vehicles

Electric Vehicle Charging Supply
Electric Vehicle Charging Supply
ANSI/NEMA C12 (Electricity Meter)
Home Energy Gateway
SAE J1772 (connector)
Electric Vehicle

Freescale Solutions for Plug-in Hybrid Electric Vehicle—Outside the car

Products	Key Features
MCF51EM256, MC9S08GW64	• Electricity meter or point of load meter
I.MX2xx family, MPC8306	• Home energy gateway
MC1322x ZigBee (Smart Energy profile and Home Automation profile)	• Home area networking



Home Energy Management System

Electricity cost profile provided to the HEMS

HEMS determines optimal setpoints for all controllable appliances

- Investigating the application of HTM to Intel’s Home Energy Management Systems
- Intel lead is Annabelle Pratt, Intel Energy Systems Research Lab