Scalable Pattern Recognition

Heterogeneous System Framework & Practice

Tonghua Su
thsu@hit.edu.cn
School of Software
Harbin Institute of Technology

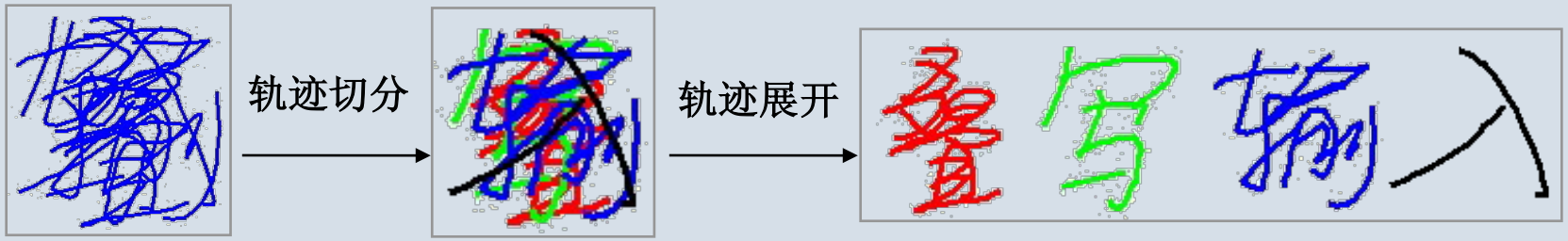
Outline

1. Computing Challenging
2. CPU+GPU Framework
3. Scalable Prototype Learning
4. Conclusion

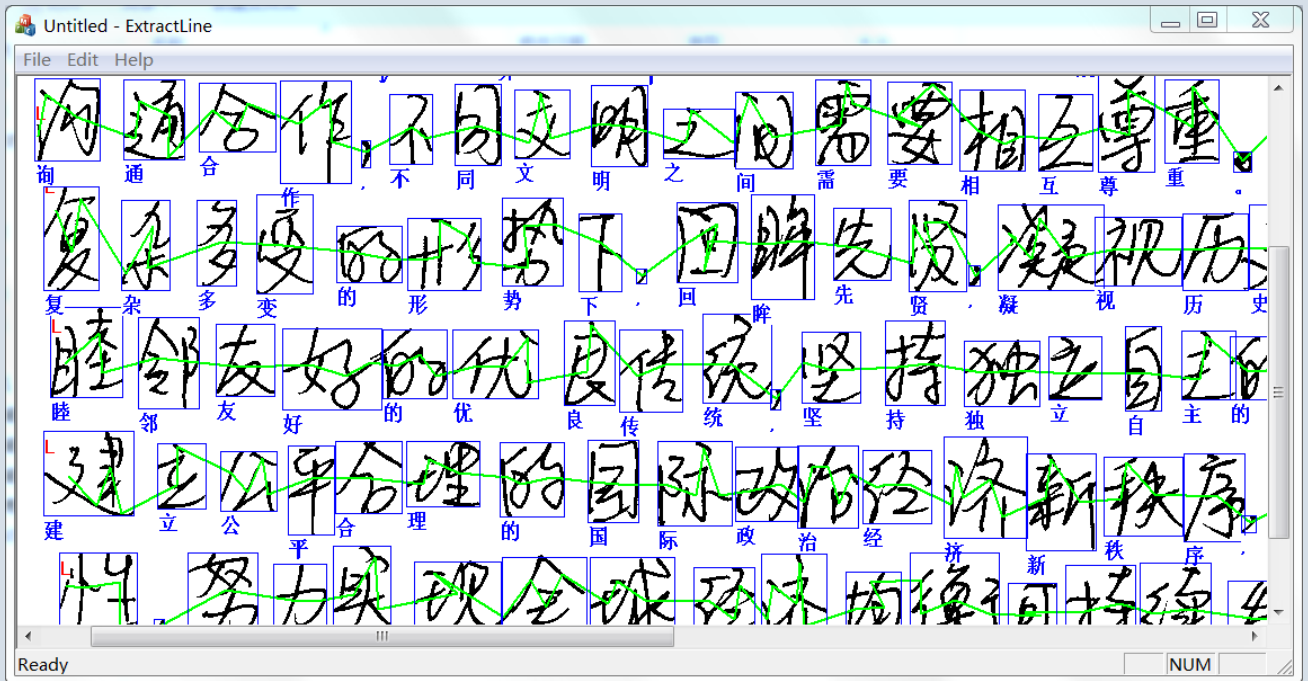
Our Experiences

● Our Work on Pattern Recognition

✓ Chinese Handwriting Input

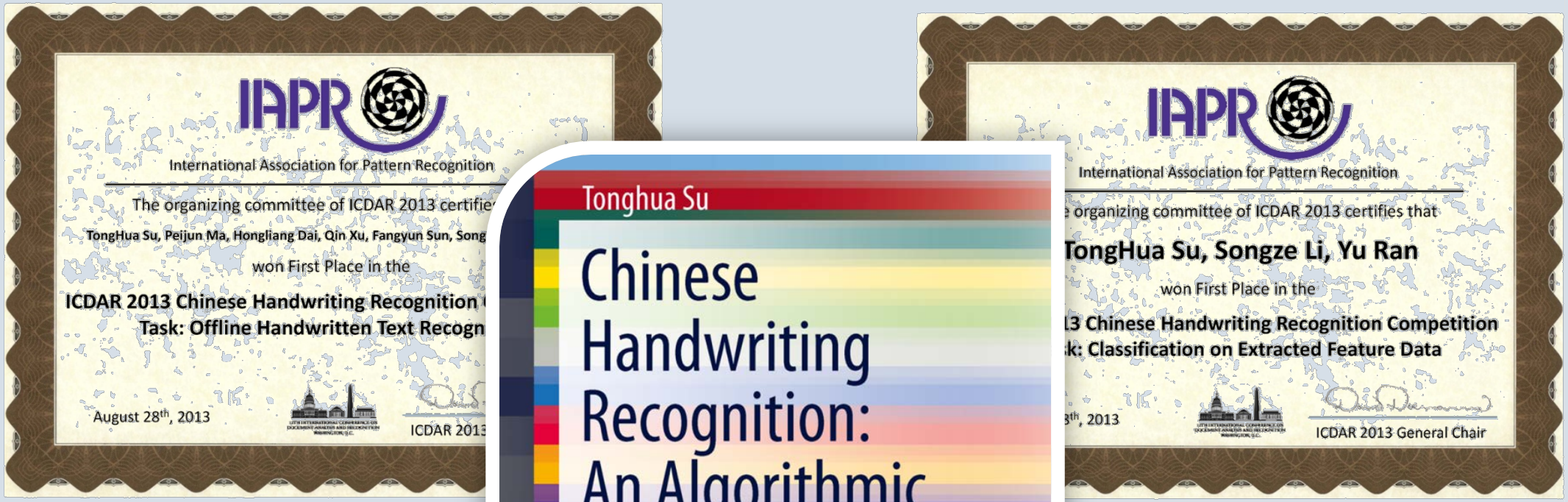


✓ Offline Document Transcription



Our Experiences

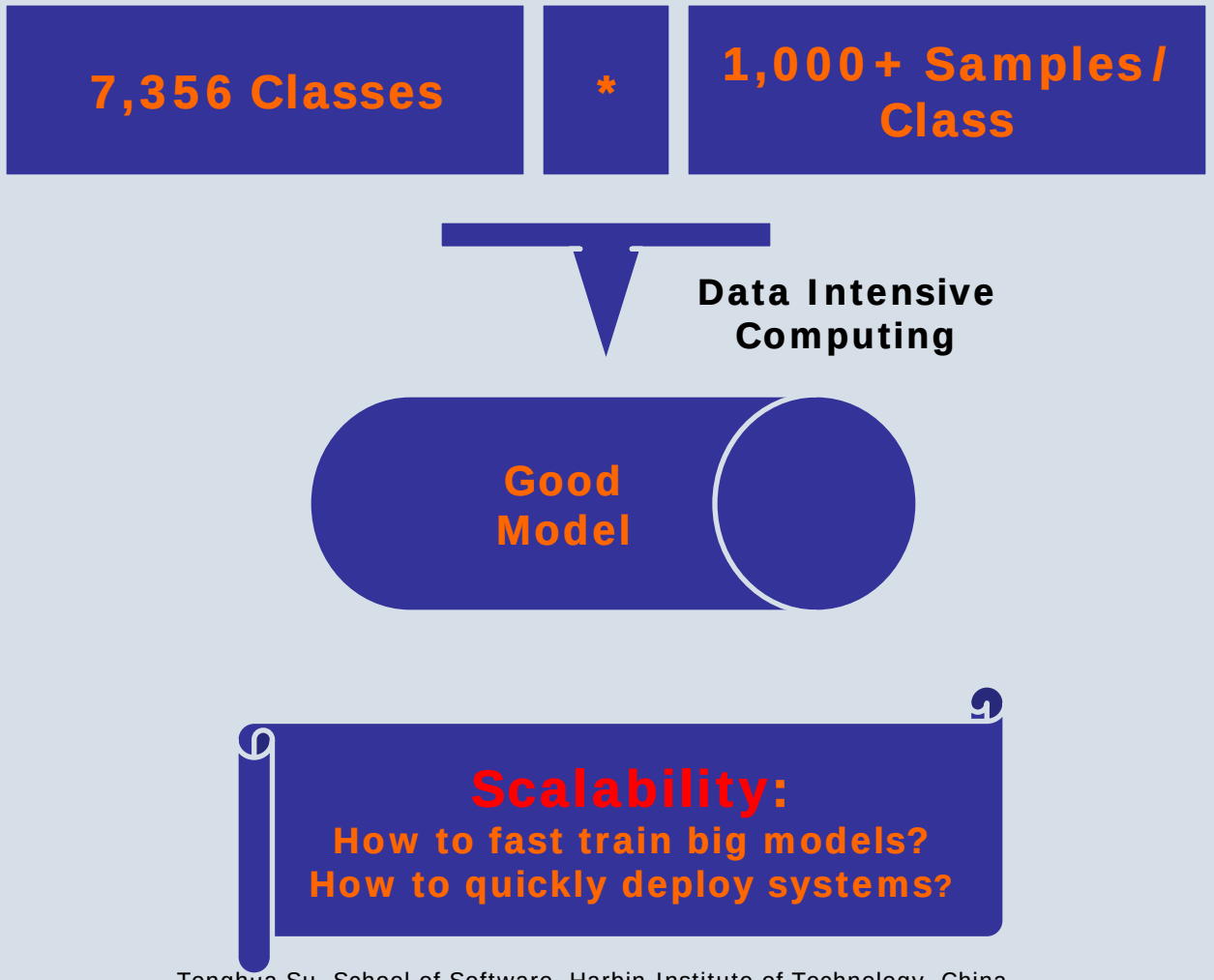
Our Work on Pattern Recognition



Our Experiences

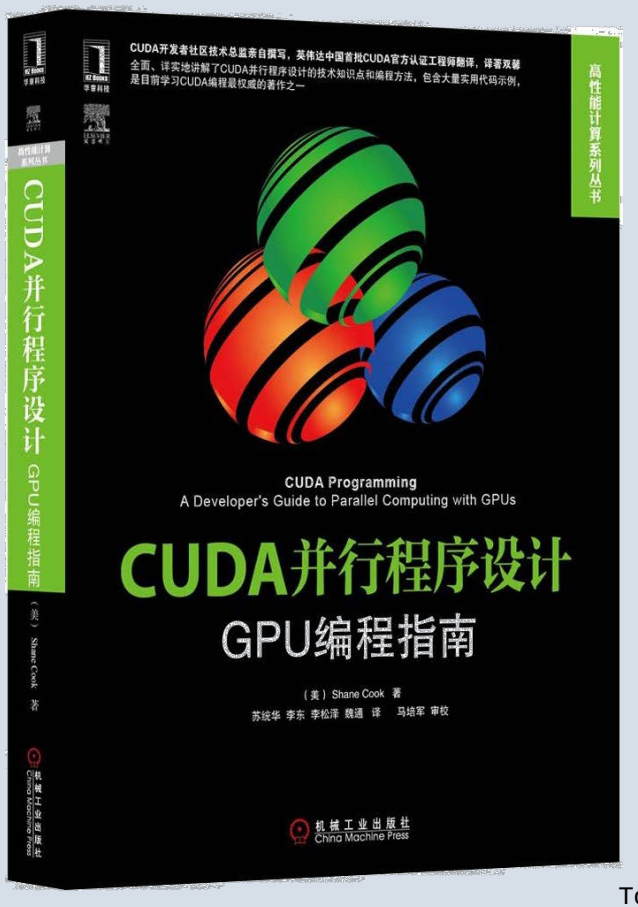
Our Work on Pattern Recognition

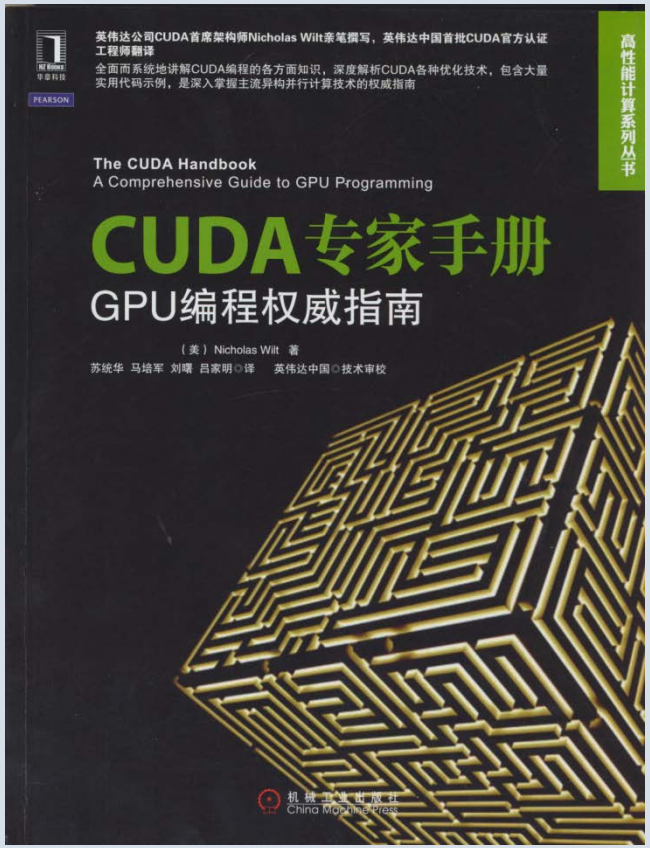
- ✓ We face a challenging problem!



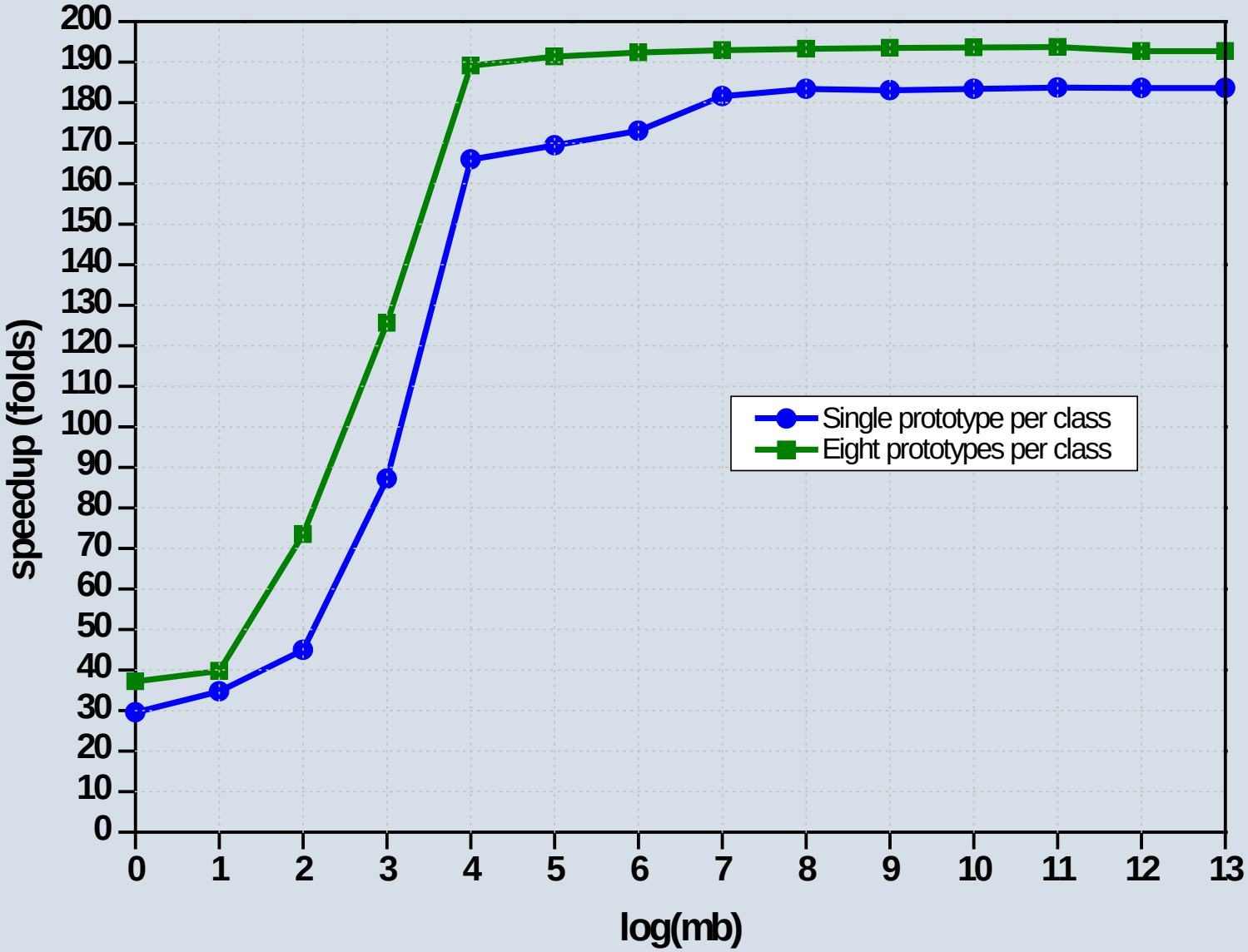
Our Experiences

- **NVIDIA CUDA CTC & CRC (@2014)**
  - ✓ CTC (CUDA Teaching Center)
  - ✓ CRC (CUDA Research Center)





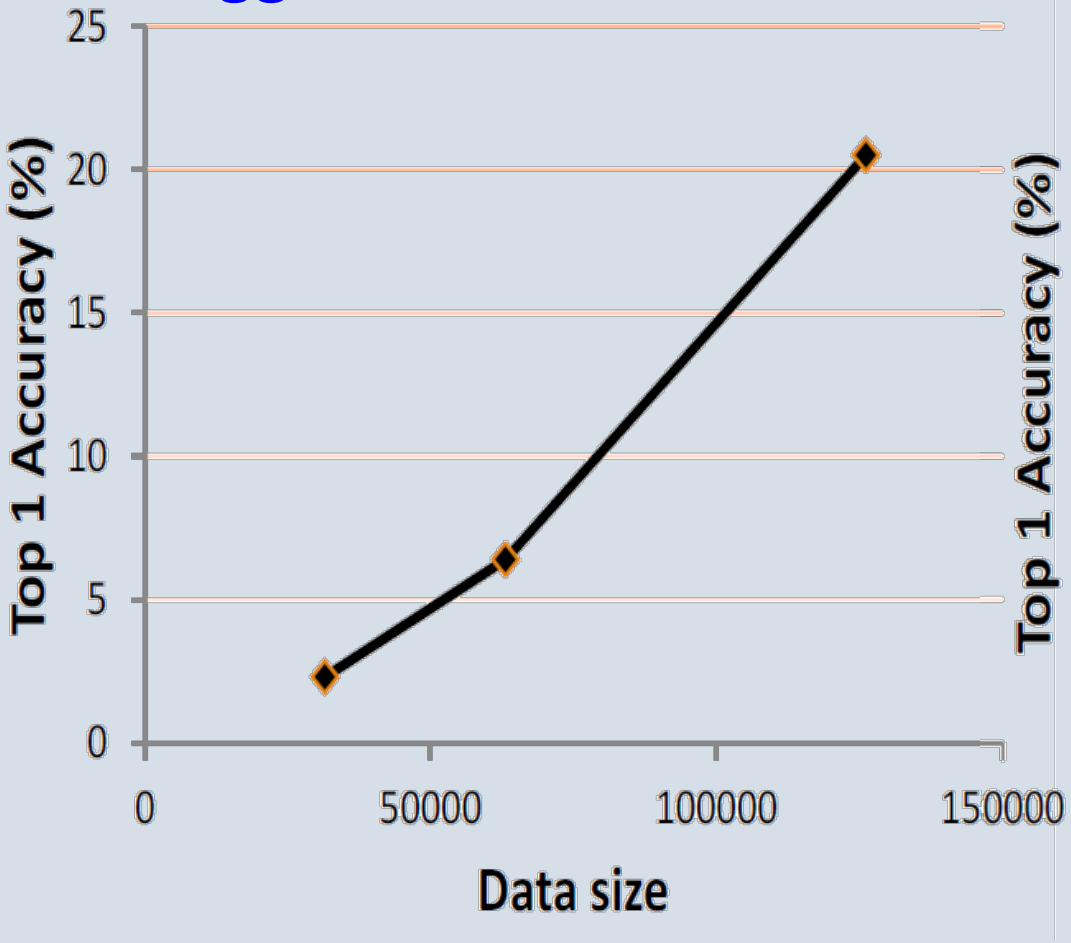
Our Experiences

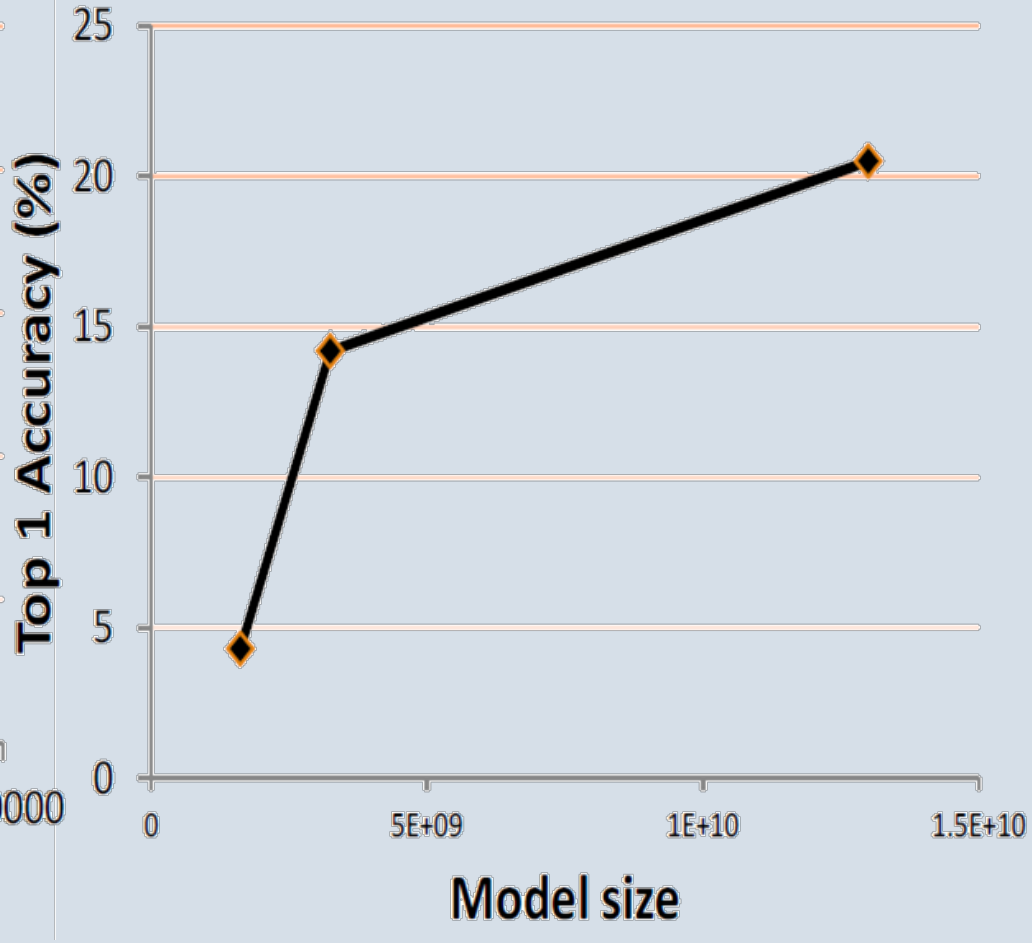


Tonghua Su, Songze Li, et. al. Scalable Prototype Learning using GPUs. in: International Conference on Image Analysis and Recognition, 2014

State of the Art

- More data, more accurate
- Bigger model, more accurate



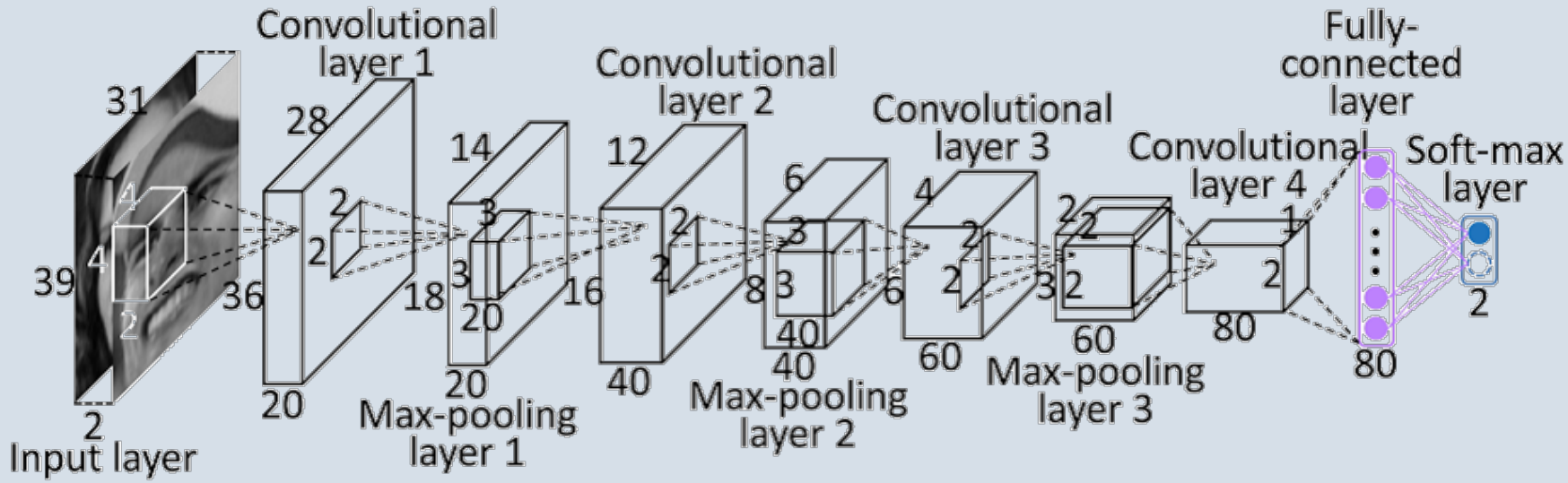


Trishul Chilimbi Yutaka Suzue Johnson Apacible Karthik Kalyanaraman. Project Adam: Building an Efficient and Scalable Deep Learning Training System. in: 11th USENIX Symposium on Operating Systems Design and Implementation, 2014

State of the Art

Face Recognition

- ✓ Big Data + Deep Neual Network
- ✓ e.g. DeepID



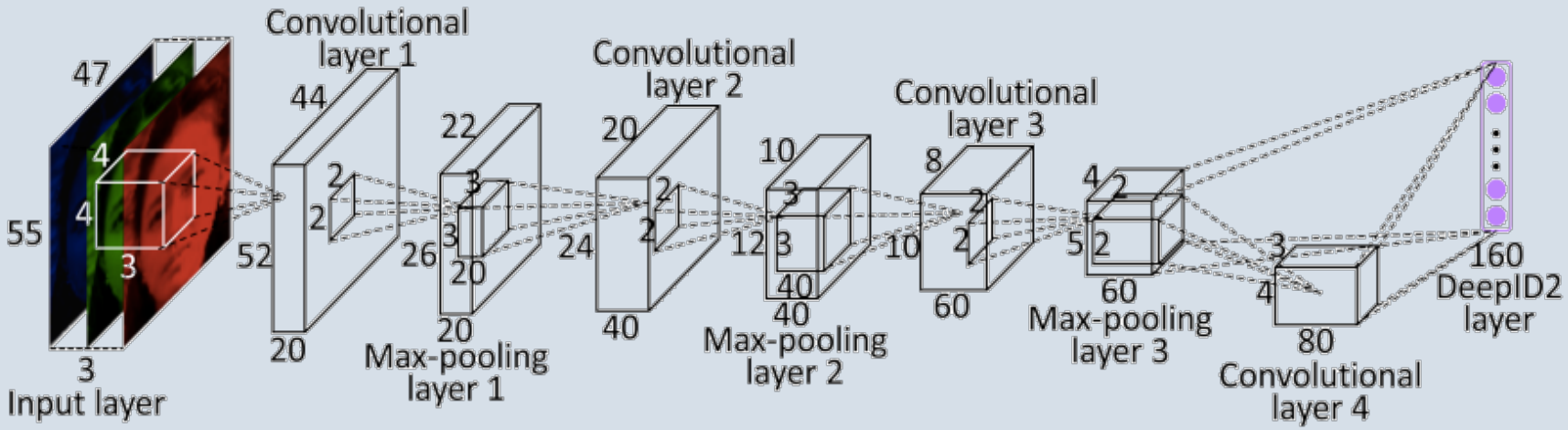
- 39x31 Patch
- max. 2x2x60 conv kernels
- 200 CPU cores, 1 month

Sun Y., Wang X., Tang X. Hybrid deep learning for face verification. ICCV, 2013.

State of the Art

Face Recognition

- ✓ Big Data + Deep Neual Network
- ✓ e.g. DeepID2

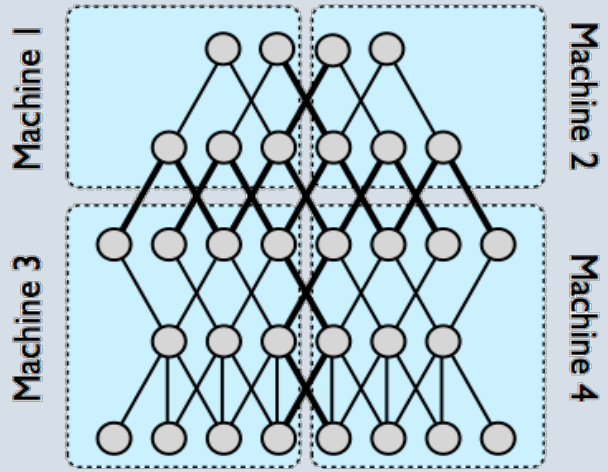


- 55x47 Patch
- 2 Identical CNN to combine a larger network
- max. 2x2x60 conv kernels (with cuBLAS GEMM)
- 10 K20/TITAN GPUs, ~7 days

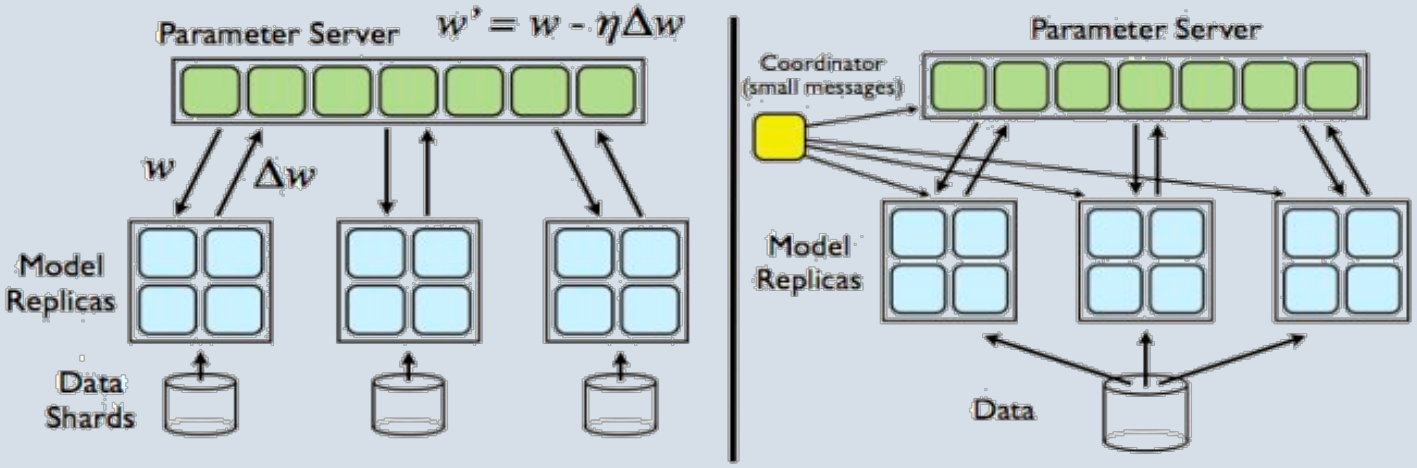
Sun Y., Wang X., Tang X. Deep Learning Face Representation by Joint Identification-Verification, NIPS, 2014.

State of the Art

- **How about CPU Cluster?**
  - ✓ Google's **DistBelief**
    - 1,000 Machines
    - 16,000 CPU cores
    - Model Parallelism
    - Also Data Parallelism



Model Parallelism

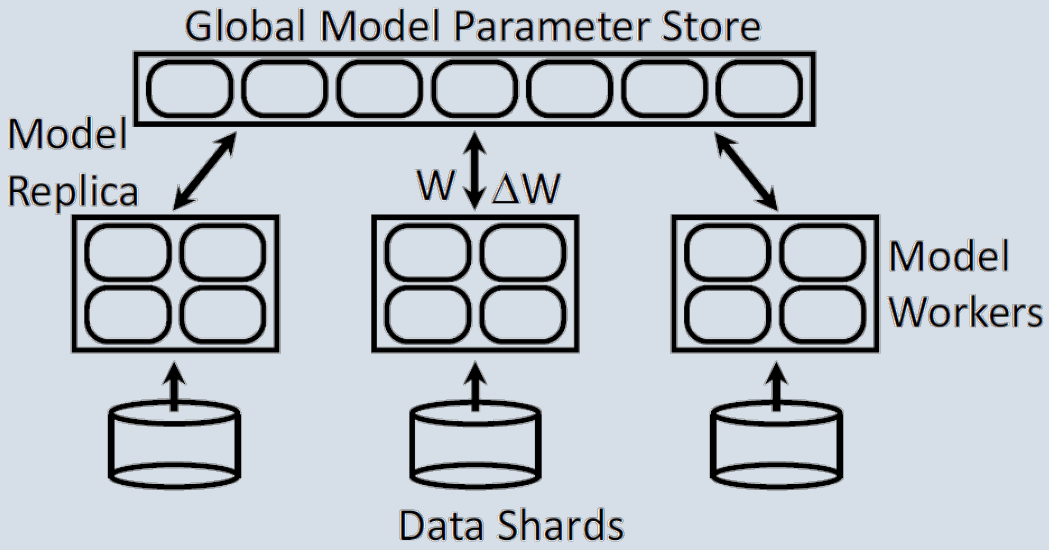


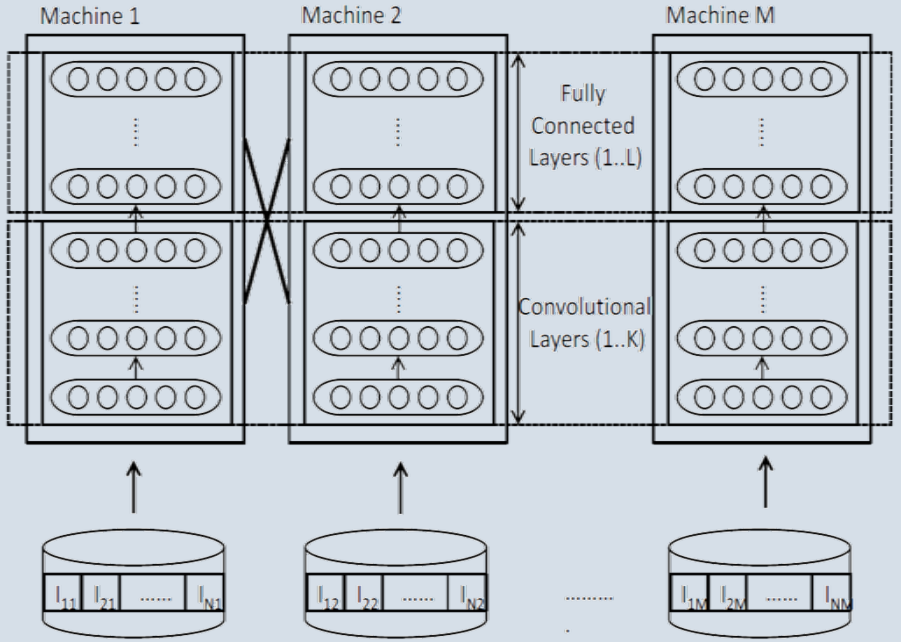
Learning Architecture

Dean Jeffrey, Corrado Greg, Monga Rajat, et al.,. *Large Scale Distributed Deep Networks*. NIPS, 2012.

State of the Art

- **How about CPU Cluster?**
  - ✓ **Microsoft's Adam**
    - **110 machines (HP Proliant Server with 16 cores)**
    - **Special Optimization**
    - **Similar Performance with DistBelief**





Trishul Chilimbi Yutaka Suzue Johnson Apacible Karthik Kalyanaraman. Project Adam: Building an Efficient and Scalable Deep Learning Training System. in: 11th USENIX Symposium on Operating Systems Design and Implementation, 2014

Computing Challenging

- **Brief Summary**
  - ✓ We arrived at a big data era
  - ✓ Big model + big data means big surprise
  - ✓ Scalability is a crucial factor to success
  - ✓ CPU cluster is that we cannot afford

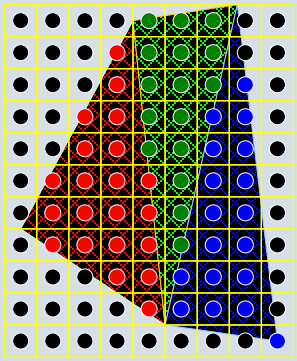
Outline

1. Computing Challenging
2. CPU+GPU Framework
3. Scalable Prototype Learning
4. Conclusion

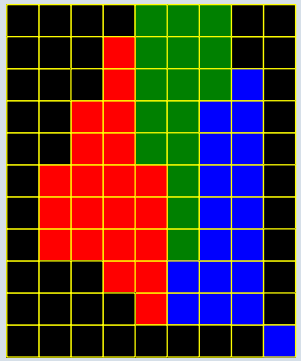
CPU+GPU Framework

‘GPUS LOVE BIG PROBLEMS. THEY’RE DESIGNED TO PROCESS HUGE AMOUNTS OF INFORMATION IN PARALLEL. MIMICKING THE HUMAN BRAIN — WHERE YOU HAVE BILLIONS OF NEURONS ALL FIRING AT THE SAME TIME — IS REALLY JUST ONE BIG PARALLEL SIMULATION.’

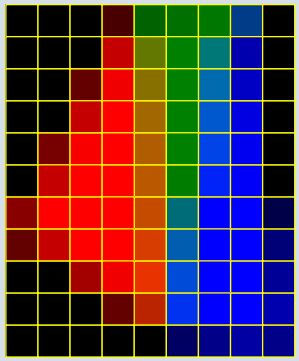
— IAN BUCK



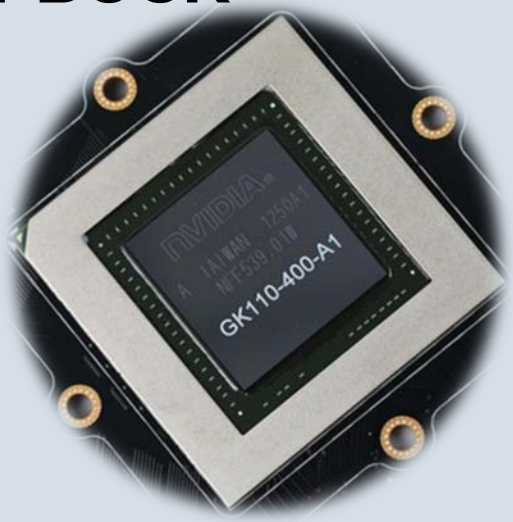
Triangle Geometry



Aliased



Anti-Aliased



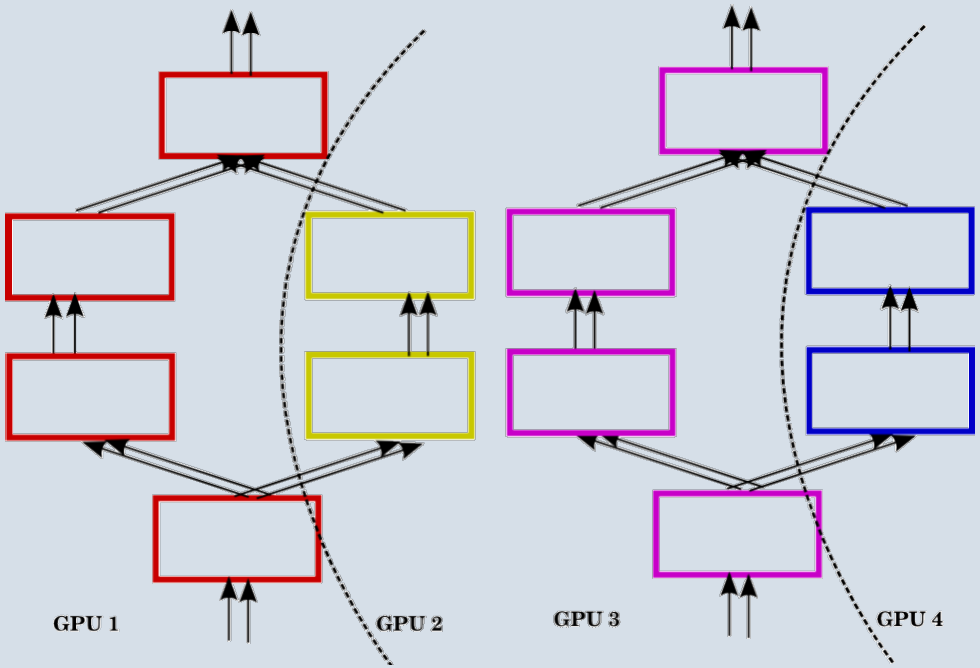
CPU+GPU Framework

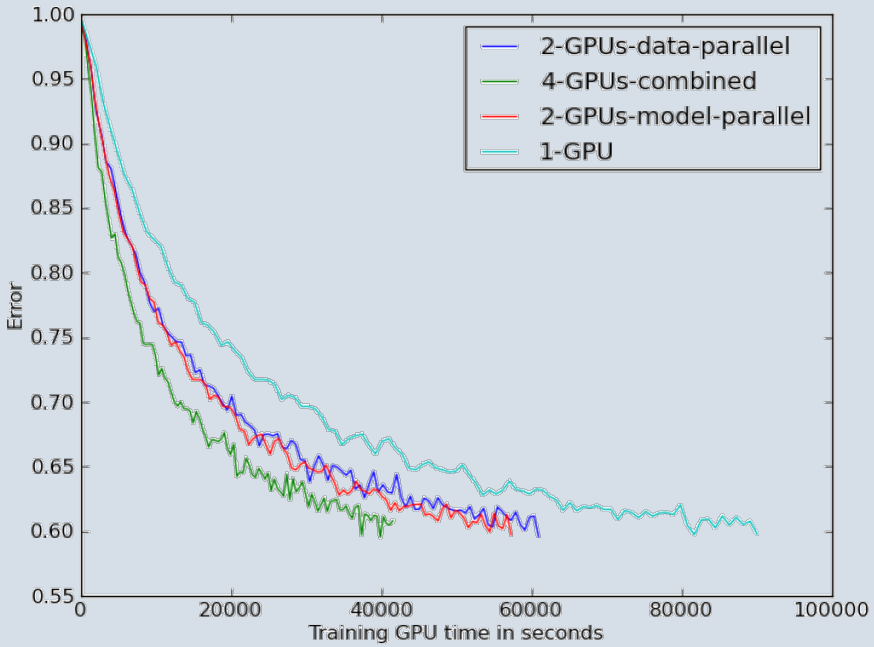
- **Chinese Handwriting Recognition Competition (@ICDAR'13)**
  - ✓ Four Groups Leverage CPU+GPU Solution
  - ✓ Fujitsu: GTX 690
  - ✓ IDSIA: Quad GTX 580
  - ✓ Uwarwick: GTX 680
  - ✓ HIT: GTX 680

Yin Fei, Wang Qiu-Feng, Zhang Xu-Yao, et al. ICDAR 2013 Chinese Handwriting Recognition Competition. ICDAR, 2013.

CPU+GPU Framework

- **Facebook's Framework**
  - ✓ Single GPU Server w/ 4 TITANs
  - ✓ Synchronous mini-batch SGD



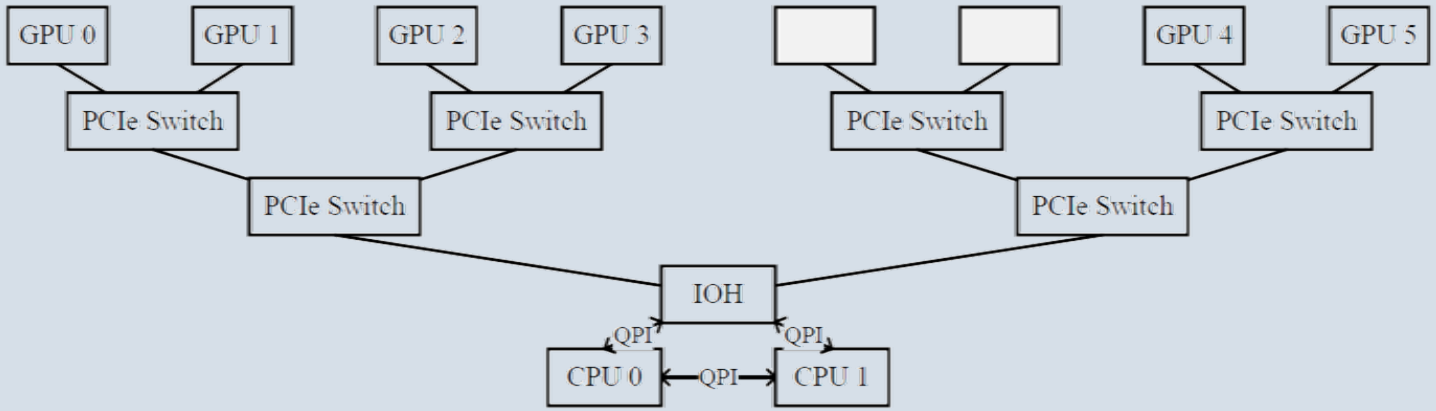


Omry Yadan Keith Adams Yaniv Taigman Marc'Aurelio Ranzato. Multi-GPU Training of ConvNets. CoPR, 2014.

CPU+GPU Framework

Tecent's Mariana

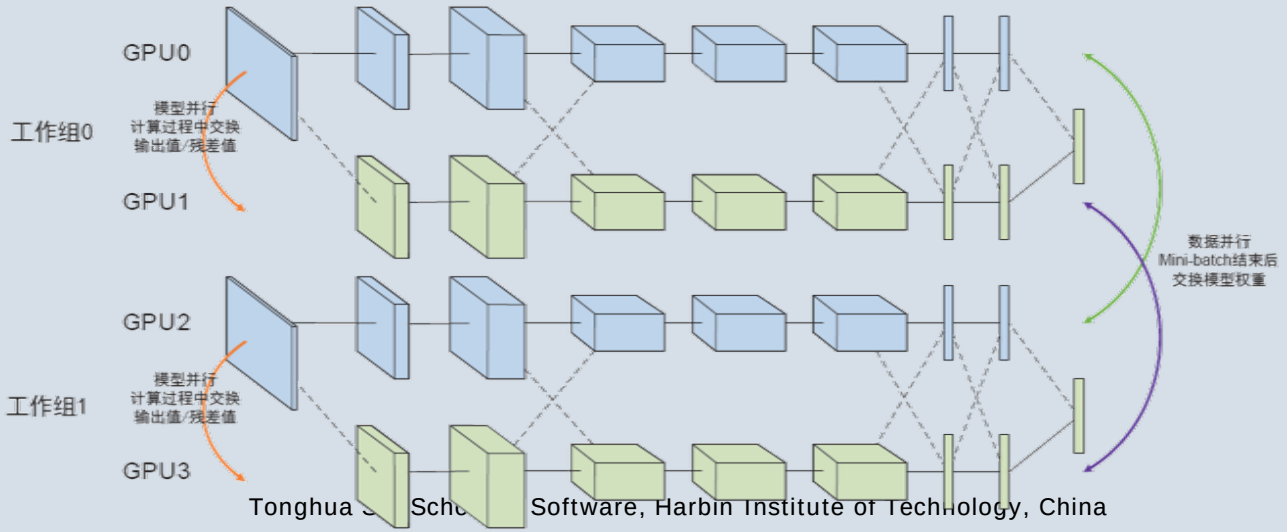
- ✓ **Multi-GPU Data Parallism**
  - **DNN-based ASR training for Webchat**
  - **6 GPUs (Tesla K20) in a server**
  - **4.6X speedup than single GPU**



Yongqiang Zou, Xing Jin, Yi Li, Zhimao Guo, Eryu Wang, Bin Xiao. Mariana: Tencent Deep Learning Platform and its Applications, VLDB, 2014.

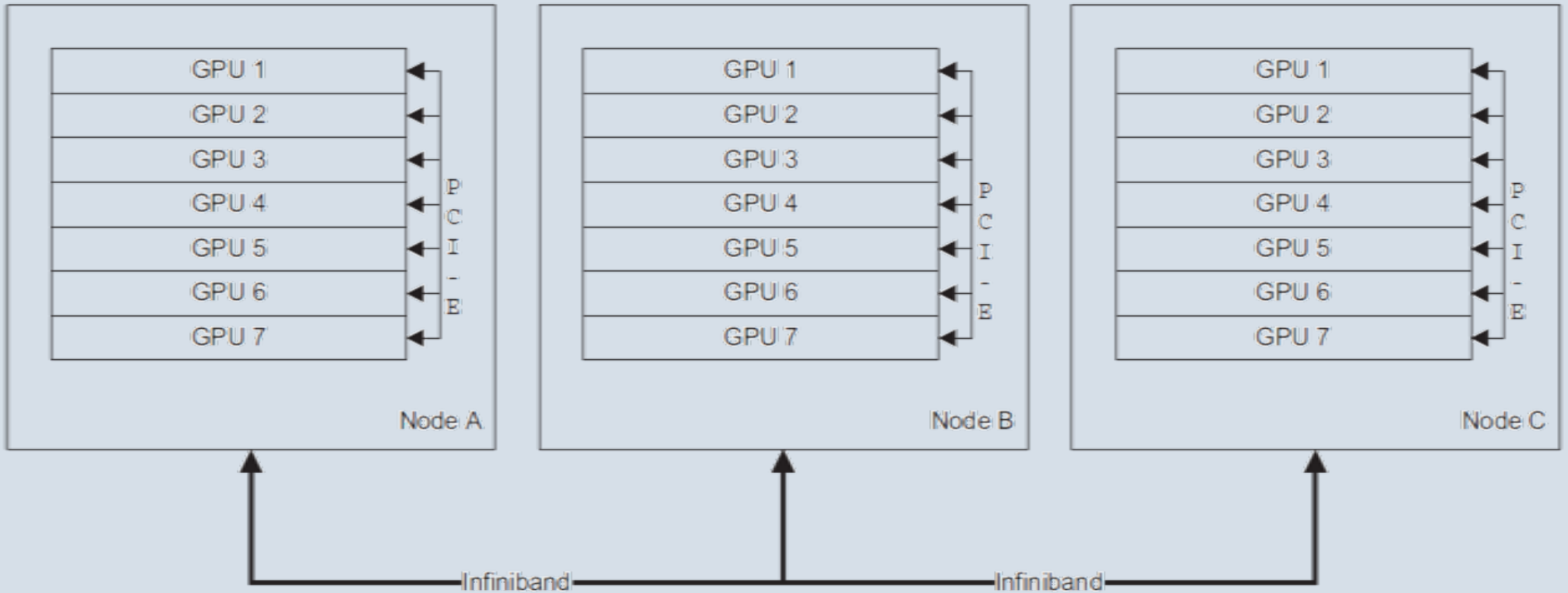
CPU+GPU Framework

- **Tecent's Mariana**
  - ✓ **Multi-GPU Data Parallism**
    - **DNN-based ASR training for Webchat**
    - **6 GPUs in a server**
    - **4.6X speedup than single GPU**
  - ✓ **Multi-GPU Data Parallism & Model Parallism**
    - **CNN-based training for Image Recognition, Ads pCTR**
    - **4 GPUs in a server**
    - **2.52X speedup than single GPU**



CPU+GPU Framework

- **Standord's COTS HPC system**
  - ✓ **16 GPU Servers**
  - ✓ **4 GTX 680/GPU Server**
  - ✓ **Infiniband Interconnected**



CPU+GPU Framework

- **Standord's COTS HPC system**
  - ✓ 16 GPU Servers
  - ✓ 4 GTX 680/GPU Server
  - ✓ Infiniband Interconnected
  - ✓ MPI communication + CUDA platform

int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)

CPU+GPU Framework

Process 0

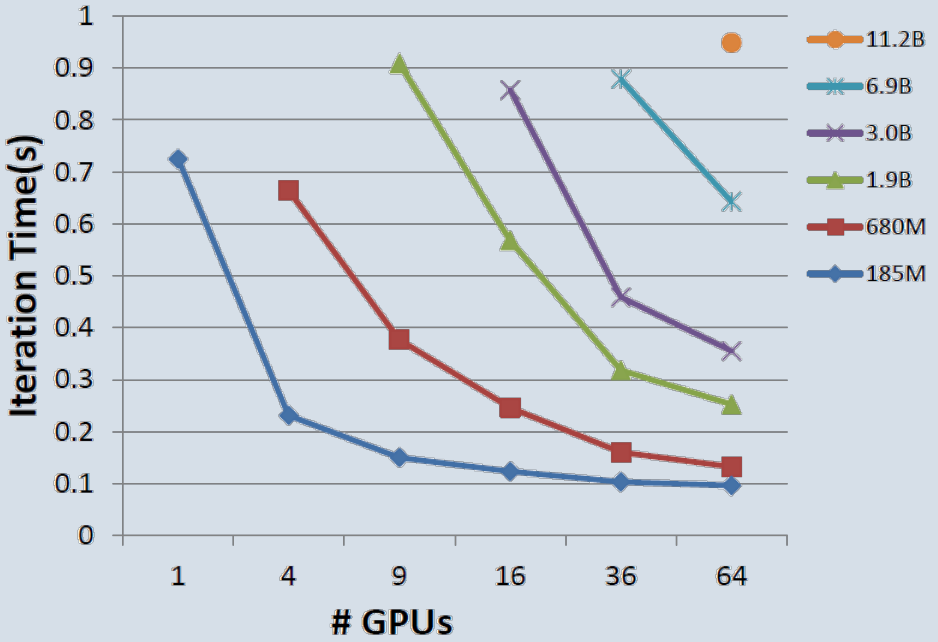
```
1. #define TAG 999
2. float a[10];
3. int dest=1;
4. MPI_Send(a, 10, MPI_FLOAT,
   dest, TAG,
   MPI_COMM_WORLD);
```

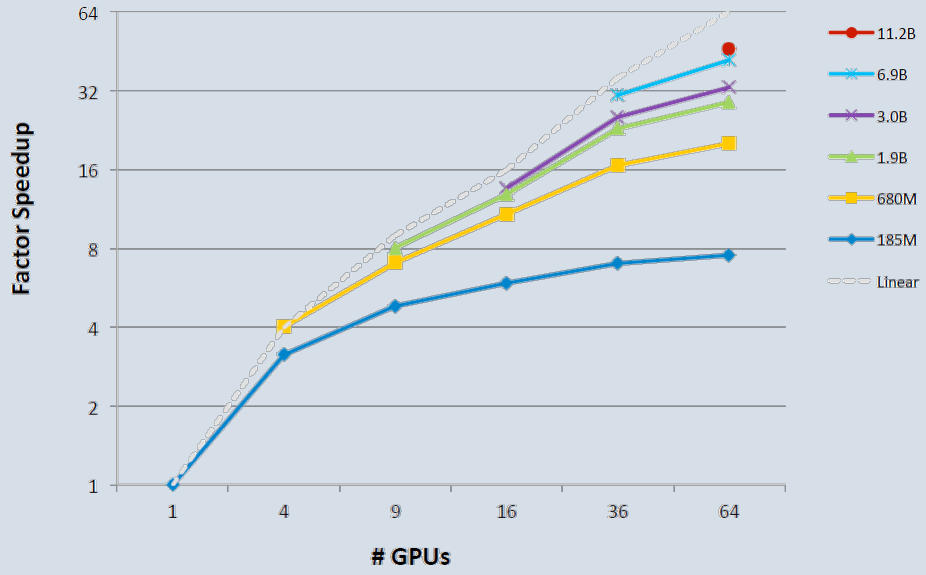
Process 1

```
1. #define TAG 999
2. MPI_Status status;
3. int count;
4. float b[20];
5. int sender=0;
6. MPI_Recv(b, 20, MPI_FLOAT,
   sender, TAG,
   MPI_COMM_WORLD, &status);
7. MPI_Get_count(&status,
   MPI_FLOAT, &count);
```

CPU+GPU Framework

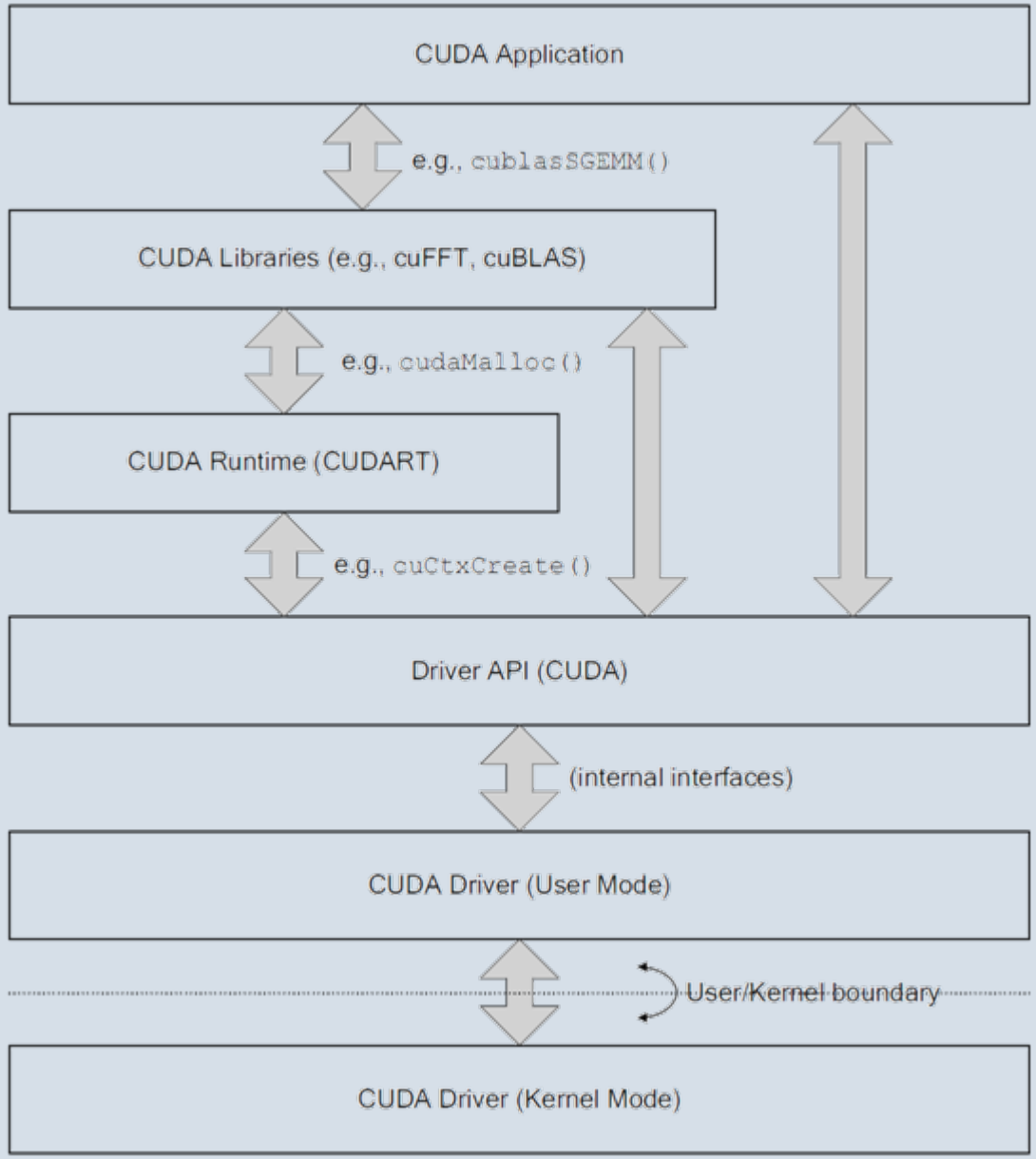
- **Standord's COTS HPC system**
  - ✓ 16 GPU Servers
  - ✓ 4 GTX 680/GPU Server
  - ✓ Infiniband Interconnected
  - ✓ MPI Communication + CUDA Platform
  - ✓ Performance Comparable to DistBelief with just 3 Servers!





CPU+GPU Framework

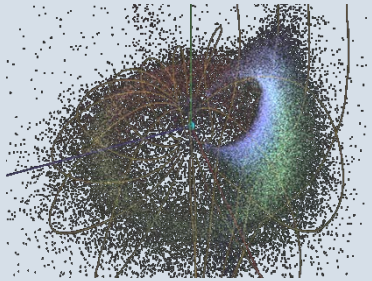
- **Programming Model**
  - ✓ **CUDA**
  - ✓ **Open CL**
  - ✓ **OpenACC**
  - ✓ **Jacket**
  - ✓ **…**



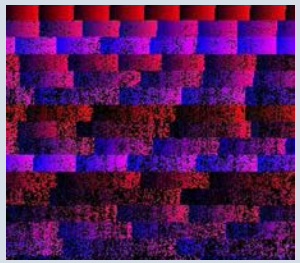
CUDA Worldwide

- 自2007年，全球600多个高校开设课程

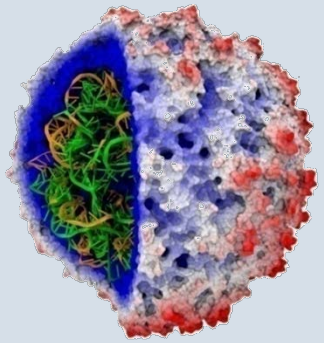
- GPU+CUDA成为最有潜力的加速器之一



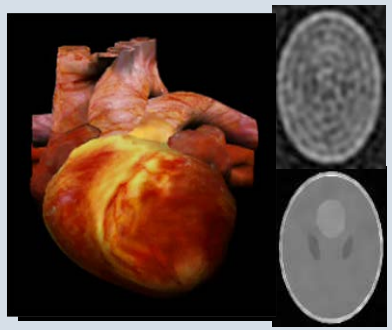
100X(N-body模拟)



35X(基因组匹配)



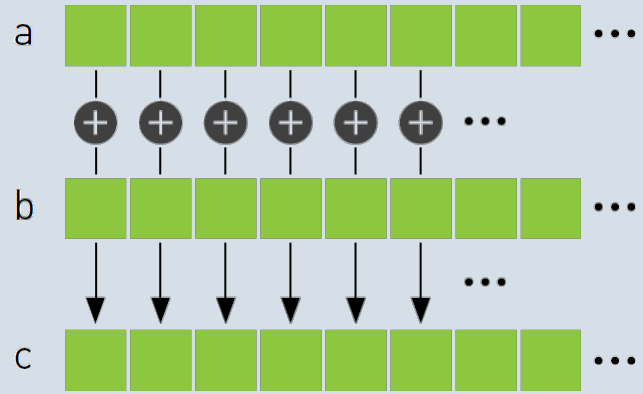
110-240X(分子模拟)



13–457X(MRI断层重构)

Hello CUDA

Vector Sum



✓ **Serial code**

```
for (i=0;i<128;i++)
{
        c[i] = a[i] + b[i];
}
```

✓ **Translate into CUDA threads**

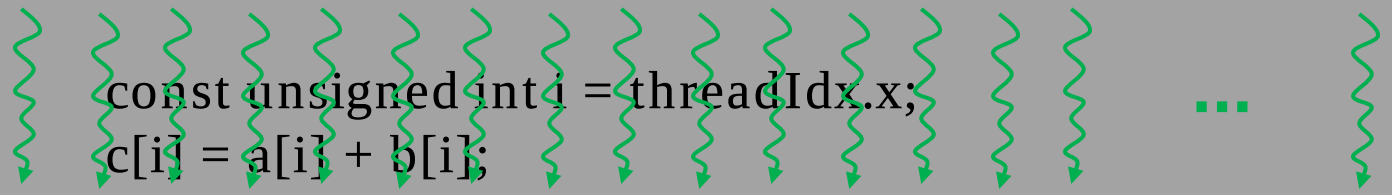
```
__global__ void addKernel(int * const a, const int * const b, const int * const c)
{
        c[i] = a[i] + b[i];
}
```

Hello CUDA

Vector Sum

- ✓ **Identify thread id**

```
__global__ addKernel(int * const a, const int * const b, const int * const c)
{
    const unsigned int i = threadIdx.x;
    c[i] = a[i] + b[i];
}
```



- ✓ **Invoke CUDA kernel**
  - **kernel_function<<<num_blocks, num_threads>>>(param1, param2, .)**
  - **e.g. addKernel<<< 1, 128 >>>(a, b, c);**

Hello CUDA

```
__global__ void addKernel(int *c, const int *a, const int *b)
{
        int i = threadIdx.x;
        c[i] = a[i] + b[i];
}
```

```
void main(){
        ......
        int *dev_a,*dev_b,*dev_c;
        // Allocate GPU buffers for three vectors (two input, one output)    .
        cudaMalloc((void**)&dev_c, 128* sizeof(int));
        cudaMalloc((void**)&dev_a, 128* sizeof(int));
        cudaMalloc((void**)&dev_b, 128* sizeof(int));

        // Copy input vectors from host memory to GPU buffers.
        cudaMemcpy(dev_a, a, 128* sizeof(int), cudaMemcpyHostToDevice);
        cudaMemcpy(dev_b, b, 128* sizeof(int), cudaMemcpyHostToDevice);

        // Launch a kernel on the GPU with one thread for each element.
        addKernel<<<1, 128>>>(dev_c, dev_a, dev_b);

        // Copy output vector from GPU buffer to host memory.
        cudaMemcpy(c, dev_c, 128* sizeof(int), cudaMemcpyDeviceToHost);

        cudaFree(dev_c);
        cudaFree(dev_a);
        cudaFree(dev_b);
}
```

CUDA Tutorial

- **NVIDIA CUDA初级教程视频**
  - ✓ http://www.iqiyi.com/a_19rrhbvoe9.html#vfrm=2-3-0-1

Amdahl's Law

- **Gene Amdahl in 1967:**

$$Speedup = \frac{1}{r_s + \frac{r_p}{N}}$$

where $r_s + r_p = 1$ and r_s represents the ratio of the sequential portion.

- ✓ **Consider: 30% portion of time with 100X speedup through paralleling vs 99% portion with 100X**

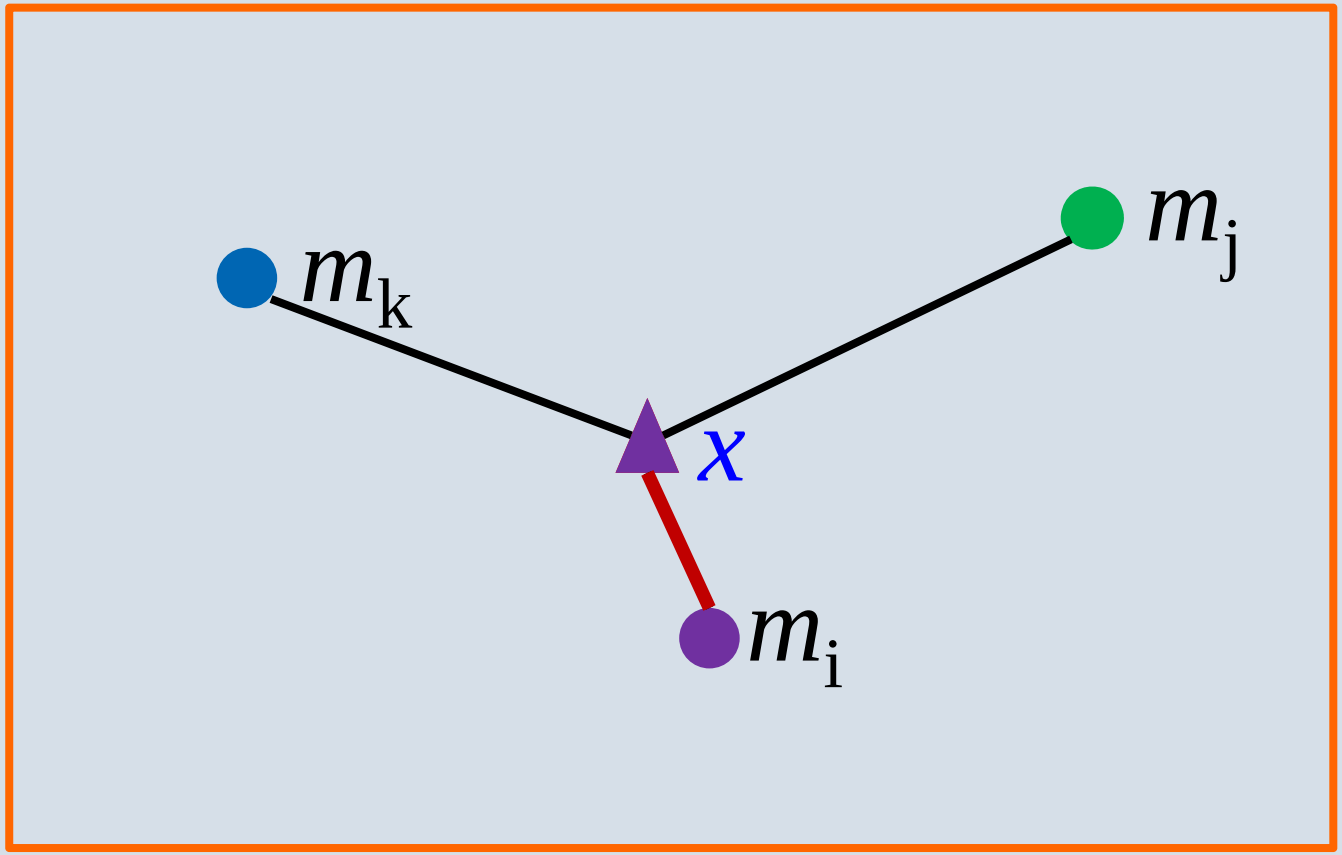
Amdahl, Gene (1967). Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities. AFIPS Conference Proceedings (30): 483–485.

Outline

1. Computing Challenging
2. CPU+GPU Framework
3. Scalable Prototype Learning
4. Conclusion

Prototype Learning Basics

- **Definition**
  - ✓ For a C-class classification task
    - generate a set of prototype vectors $\Theta = \{m_i, i=1, 2,\ldots,C\}$
- **Classification Process**

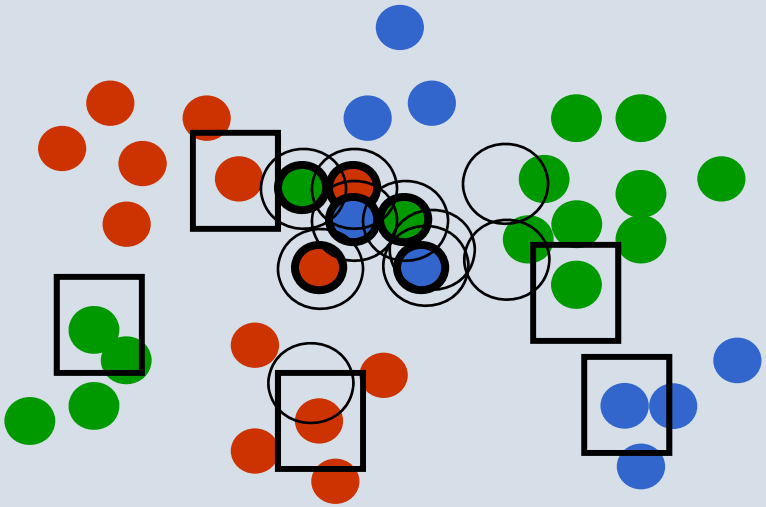


Prototype Learning Basics

Learning Process

- ✓ Training samples: $\{(\mathbf{x}_n, y_n), n=1,2,\ldots,N\}$
- ✓ Objective to minimize:

$$\hat{\Theta} = \arg\min_{\Theta} \frac{1}{N} \sum_{n=1}^{N} \phi(f(\mathbf{x}_n, \Theta)) \quad , \qquad (1)$$



Prototype Learning Basics

Learning Process

✓ GLVQ (Generalized Learning Vector Quantization):

- Define a error measure for x:

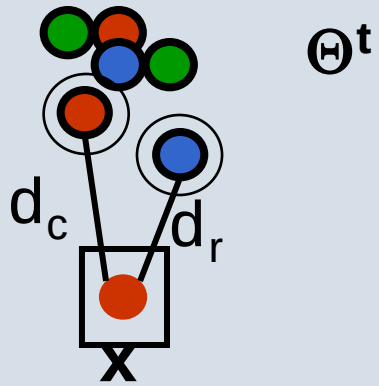
$$f(\mathbf{x}) = \frac{d_c - d_r}{d_c + d_r} \quad , \qquad (2)$$

- Loss function:

$$\phi(f(\mathbf{x})) = \frac{1}{1 + e^{-\xi f(\mathbf{x})}} \quad , \qquad (3)$$

- Learning rule with stochastic gradient desent:

$$\begin{cases} \mathbf{m}_c \leftarrow \mathbf{m}_c + \eta f(\mathbf{x})\left(1 - f(\mathbf{x})\right)\dfrac{d_r}{\left(d_c + d_r\right)^2}\left(\mathbf{x} - \mathbf{m}_c\right) \\ \mathbf{m}_r \leftarrow \mathbf{m}_r - \eta f(\mathbf{x})\left(1 - f(\mathbf{x})\right)\dfrac{d_c}{\left(d_c + d_r\right)^2}\left(\mathbf{x} - \mathbf{m}_r\right) \end{cases} . \qquad (4)$$



A. Sato and K. Yamada. Generalized learning vector quantization. in NIPS, 1996.

Prototype Learning Basics

GLVQ Learning Algorithm (Sequential Version)

Input: training set $\{\mathbf{x}_n, y_n\}_{n=1,\ldots,N}$, initial prototypes $\{\mathbf{m}_i\}_{i=1,\ldots,C}$

Output: $\{\mathbf{m}_i\}_{i=1,\ldots,C}$

```
while not convergent do
    for each {x_n, y_n}
        find out (m_c, d_c) and (m_r, d_r) through compute-then-compare distances
        compute error measure f(x) using Equation (2)
        derive loss function φ(x) using Equation (3)
        update m_c, m_r using Equation (4)
    end for
end while
return {m_i}_{i=1,...,C}
```

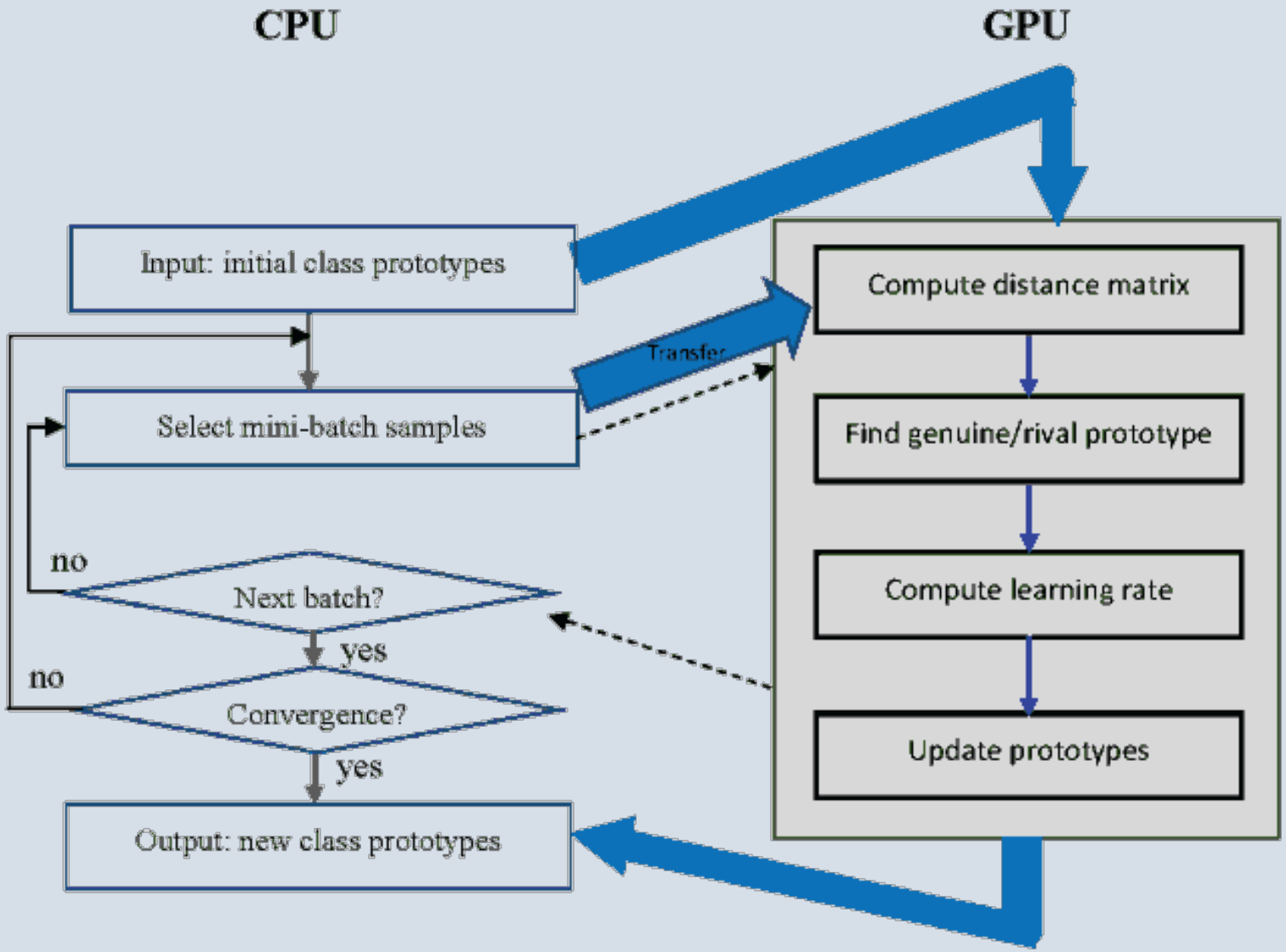
Scalability of Prototype Learning

GLVQ Learning Algorithm (Parallel Framework)

1: **while** not convergent **do**
2: **for each** mini-batch $T_i = \{(\mathbf{x}_{i_1}, y_{i_1}), \ldots, (\mathbf{x}_{i_M}, y_{i_M})\}$
3: compute all distances as a matrix **in parallel**
4: find out genuine/rival pair **in parallel**
5: derive loss function **in parallel**
6: update prototypes **in parallel**
7: **end for**
8: **end while**
9: **return** $\{\mathbf{m}_i\}_{i=1,\ldots,C}$

Scalability of Prototype Learning

- **HSA Computing Model**



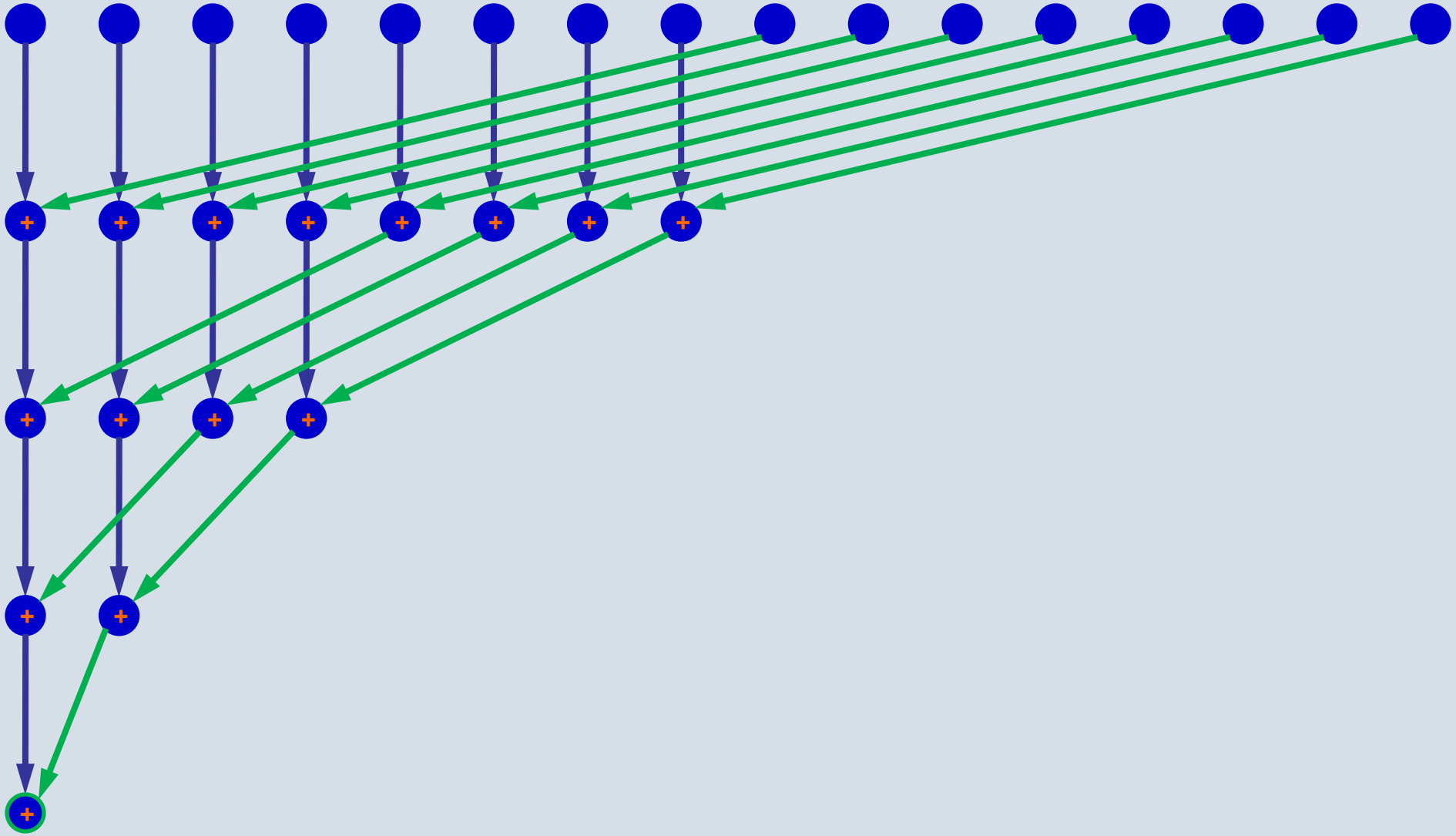
Scalability of Prototype Learning

- **Compute the distance matrix**
  - ✓ A separate CUDA kernel
  - ✓ Two approaches are investigated:
    - Parallel Reduction
    - Tiling Sum
- **Search Minima**
  - ✓ A separate CUDA kernel
  - ✓ Two approaches are investigated:
    - Parallel Reduction
    - Compare-and-Exchange
- **Renew Parameters**
  - ✓ A single CUDA kernel covers the last two steps
  - ✓ Trivially invoke #threads of feature dimension

Scalability of Prototype Learning

- **Compute the distance matrix**
  - ✓ A separate CUDA kernel
  - ✓ Two approaches are investigated:
    - Parallel Reduction
    - Tiling Sum
- **Search Minima**
  - ✓ A separate CUDA kernel
  - ✓ Two approaches are investigated:
    - Parallel Reduction
    - Compare-and-Exchange
- **Renew Parameters**
  - ✓ A single CUDA kernel covers the last two steps
  - ✓ Trivially invoke #threads of feature dimension

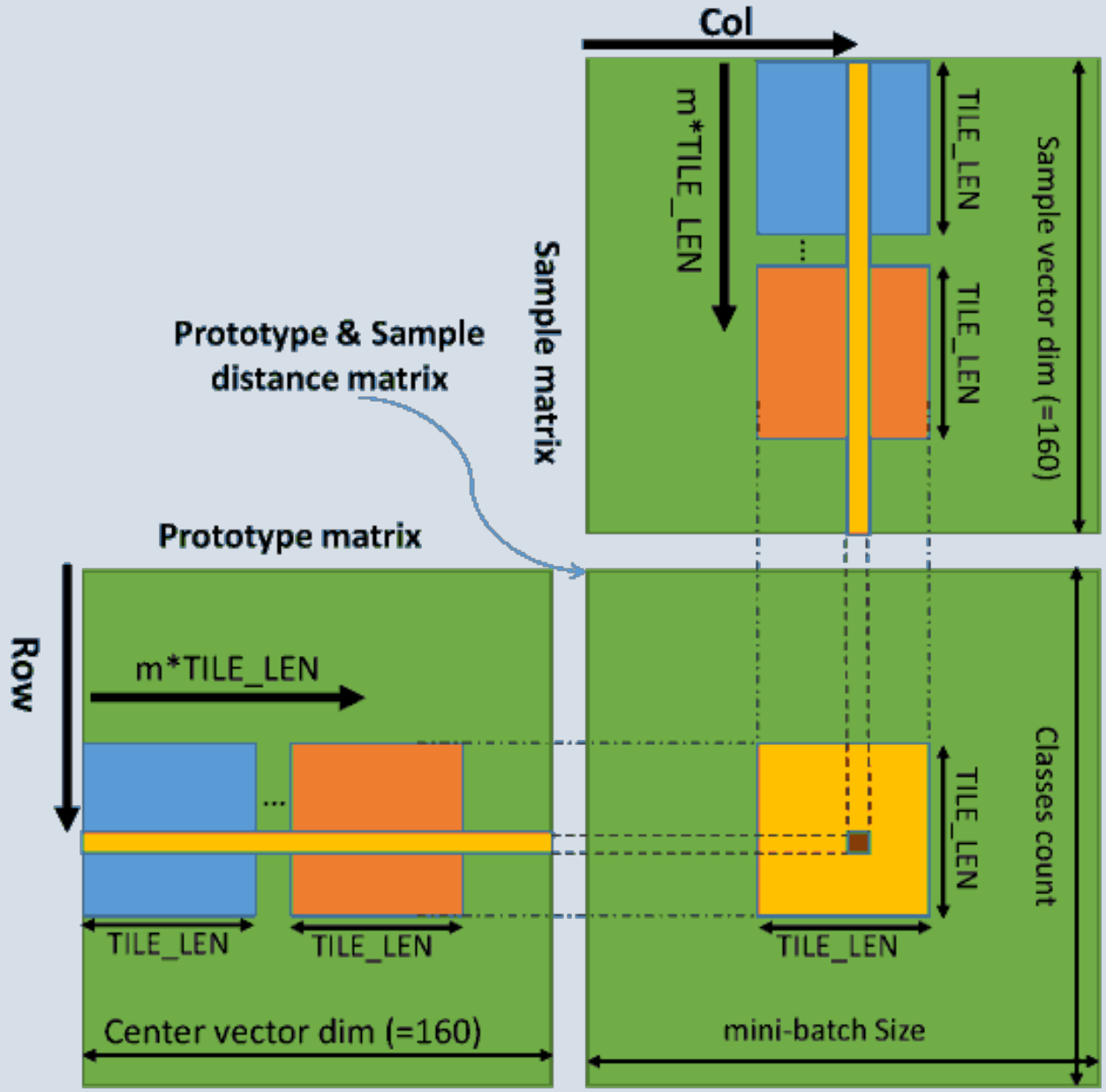
Parallel Reduction



Scalability of Prototype Learning

- **Compute the distance matrix**
  - ✓ A separate CUDA kernel
  - ✓ Two approaches are investigated:
    - Parallel Reduction
    - Tiling Sum
- **Search Minima**
  - ✓ A separate CUDA kernel
  - ✓ Two approaches are investigated:
    - Parallel Reduction
    - Compare-and-Exchange
- **Renew Parameters**
  - ✓ A single CUDA kernel covers the last two steps
  - ✓ Trivially invoke #threads of feature dimension

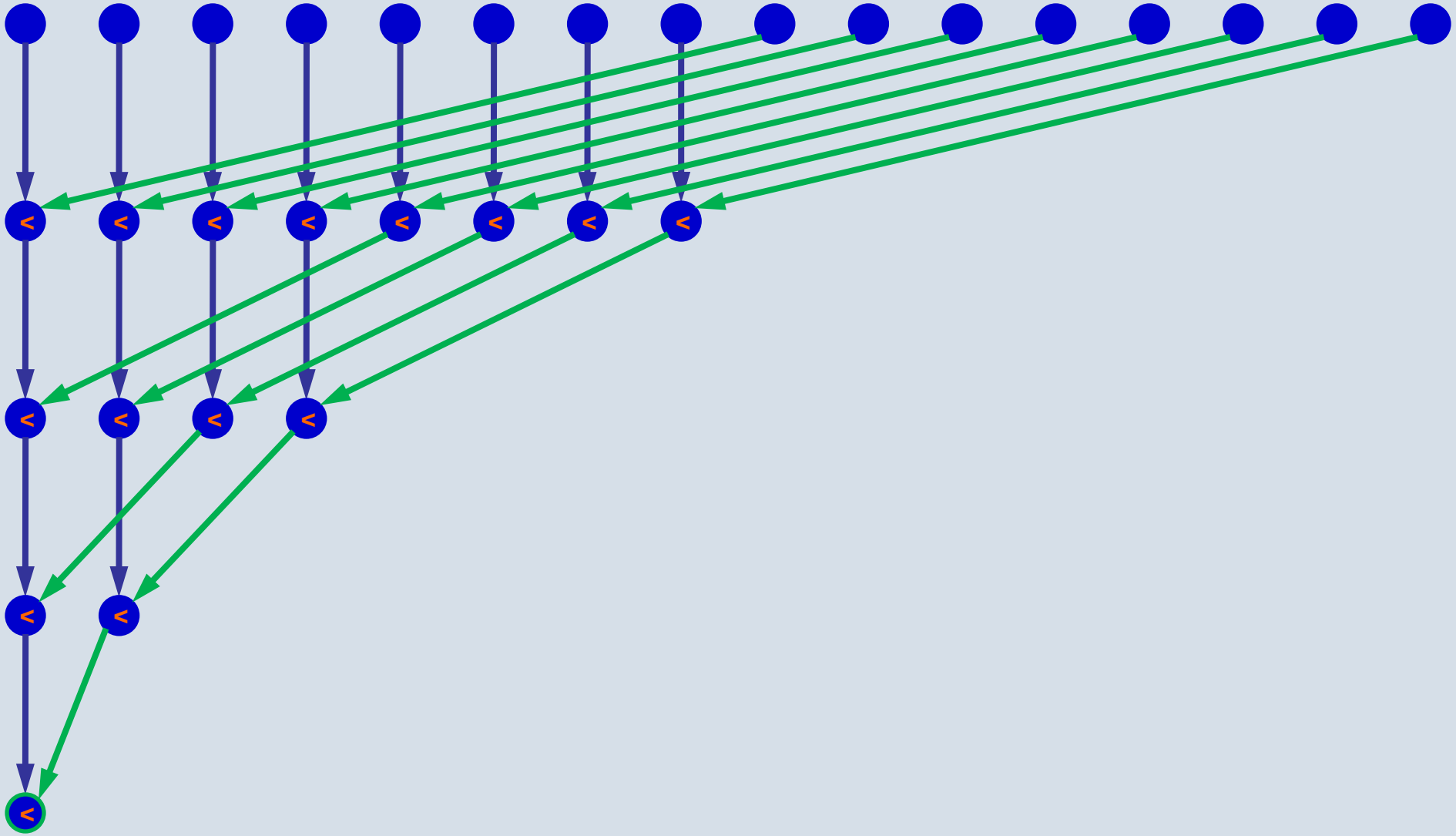
Tiling Sum


Col
m*TILE_LEN
TILE_LEN
TILE_LEN
Sample vector dim (=160)
Sample matrix
Prototype & Sample
distance matrix
Prototype matrix
Row
m*TILE_LEN
...
TILE_LEN
TILE_LEN
TILE_LEN
TILE_LEN
Classes count
Center vector dim (=160)
mini-batch Size

Scalability of Prototype Learning

- **Compute the distance matrix**
  - ✓ A separate CUDA kernel
  - ✓ Two approaches are investigated:
    - Parallel Reduction
    - Tiling Sum
- **Search Minima**
  - ✓ A separate CUDA kernel
  - ✓ Two approaches are investigated:
    - Parallel Reduction (Similar to Reduction Sum)
    - Compare-and-Exchange
- **Renew Parameters**
  - ✓ A single CUDA kernel covers the last two steps
  - ✓ Trivially invoke #threads of feature dimension

Parallel Reduction



Scalability of Prototype Learning

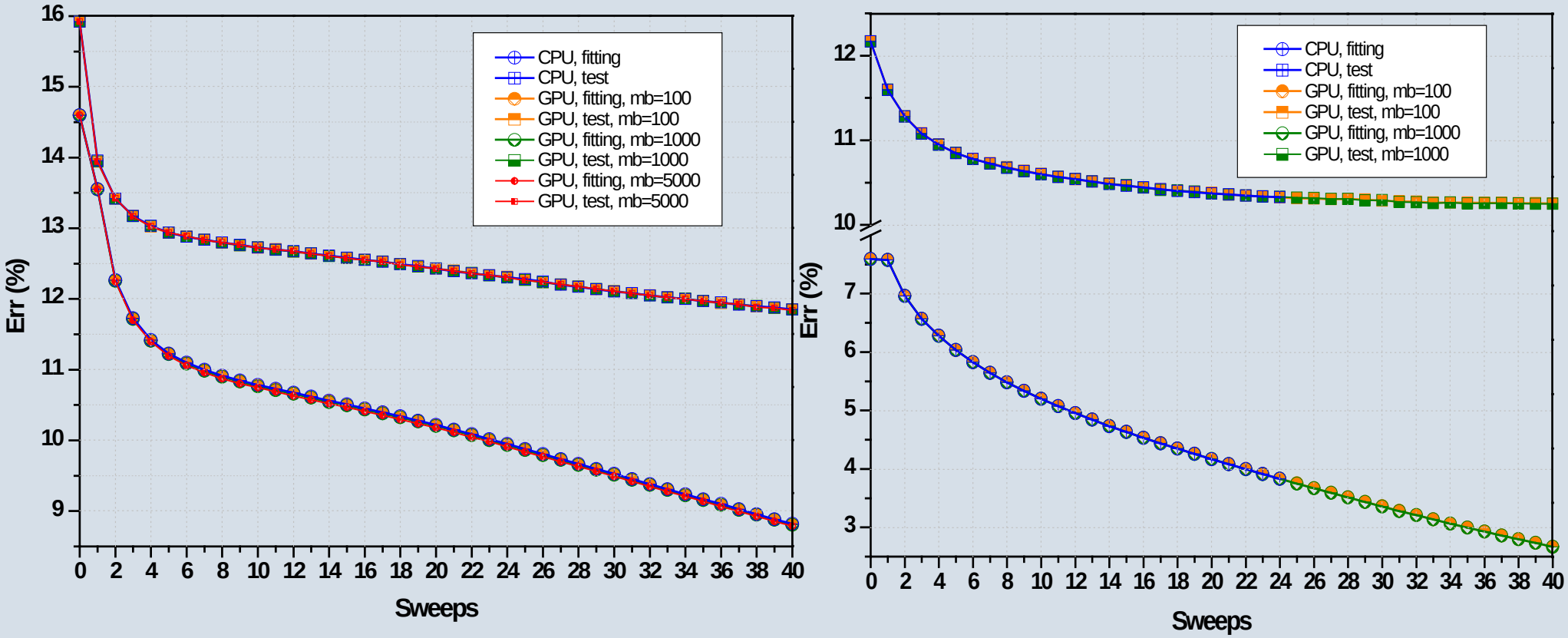
- **Compute the distance matrix**
  - ✓ A separate CUDA kernel
  - ✓ Two approaches are investigated:
    - Parallel Reduction
    - Tiling Sum
- **Search Minima**
  - ✓ A separate CUDA kernel
  - ✓ Two approaches are investigated:
    - Parallel Reduction
    - Compare-and-Exchange
- **Renew Parameters**
  - ✓ A single CUDA kernel covers the last two steps
  - ✓ Trivially invoke #threads of feature dimension

Compare-and-Exchange

- **Key Points:**
  - ✓ **Invoke #threads of mini-batch size**
  - ✓ **Each thread performs a serial searching**

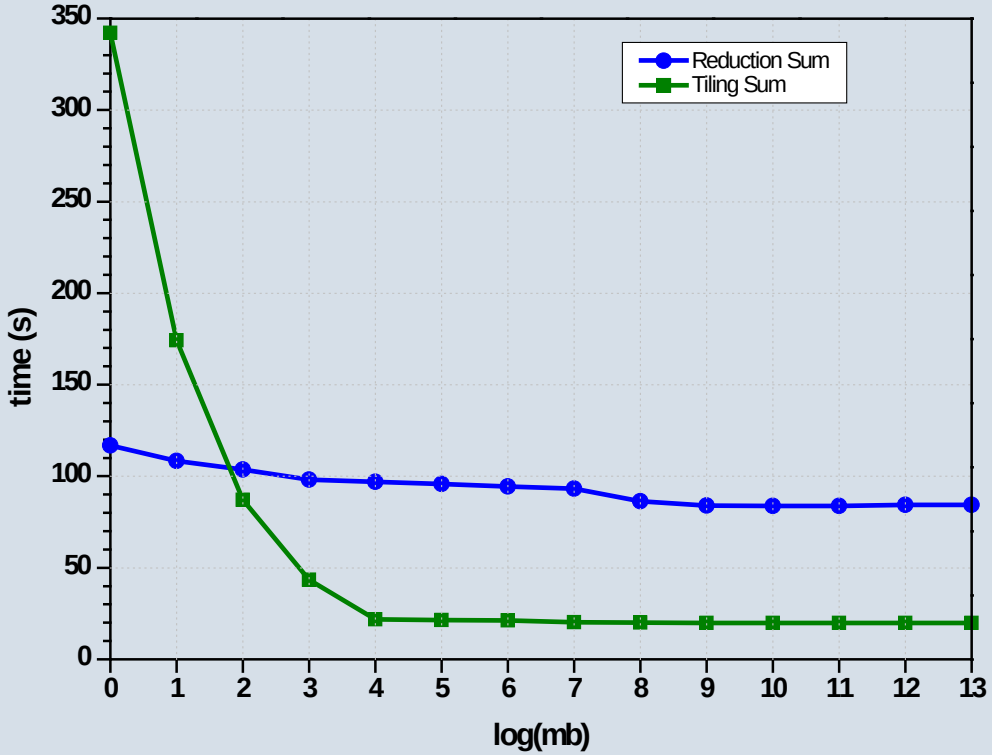
Evaluation on Chinese Character Recognition

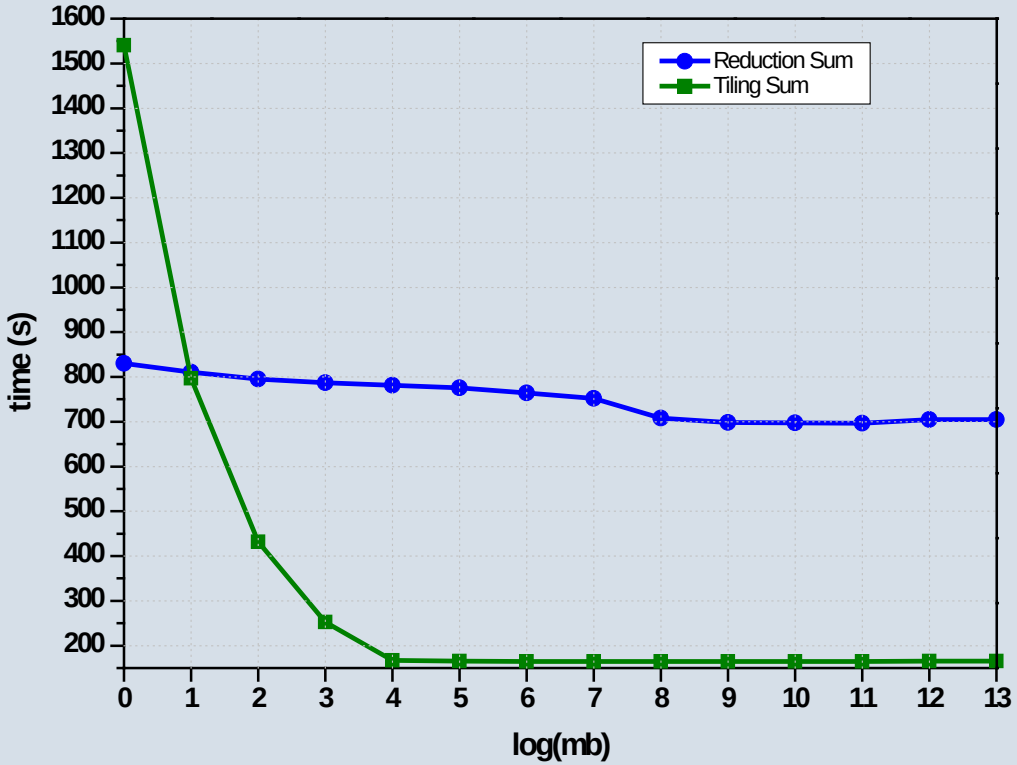
Performance Evaluation



Evaluation on Chinese Character Recognition

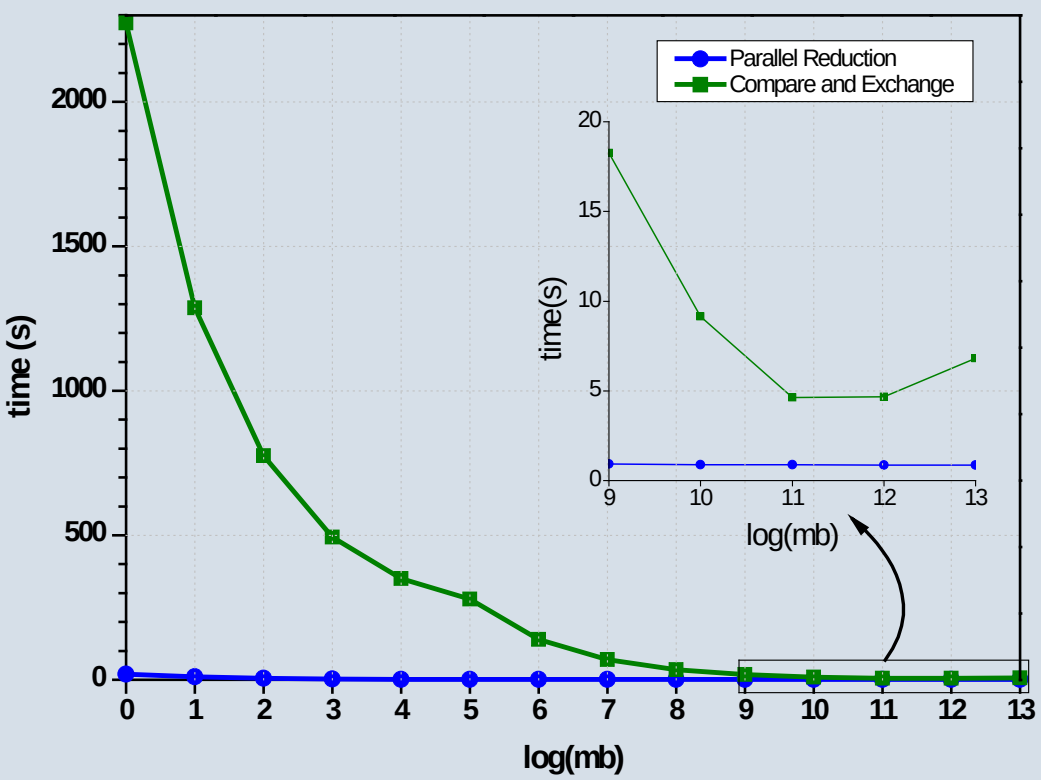
- Distance Computation Comparison

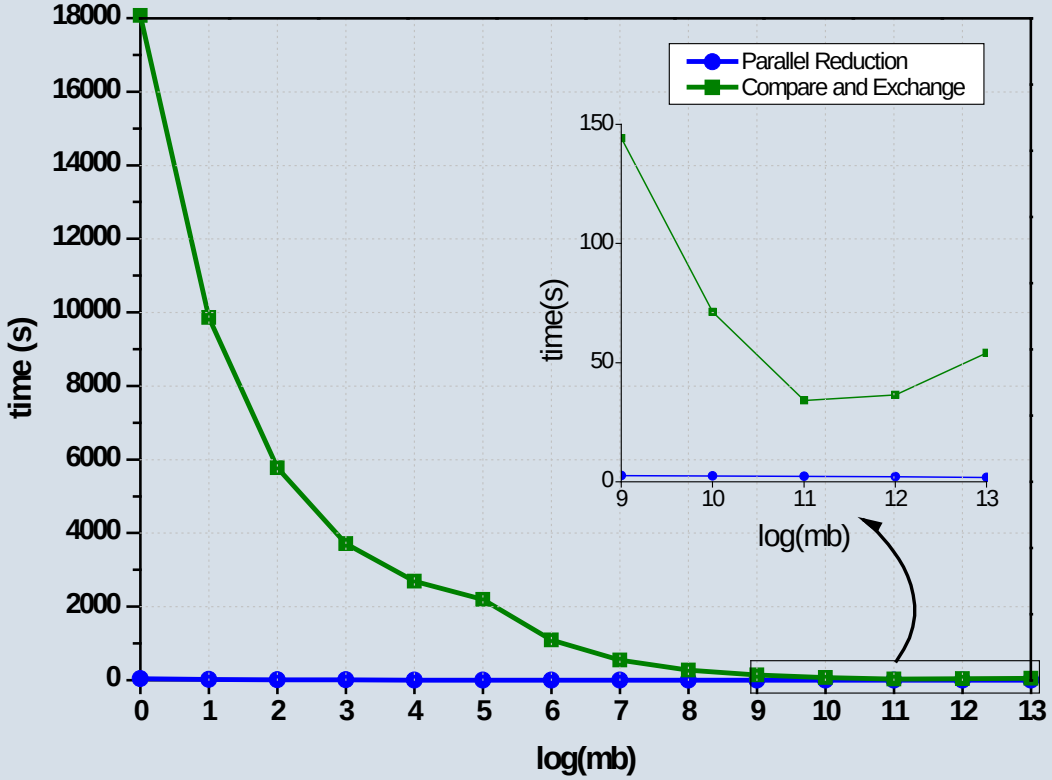




Evaluation on Chinese Character Recognition

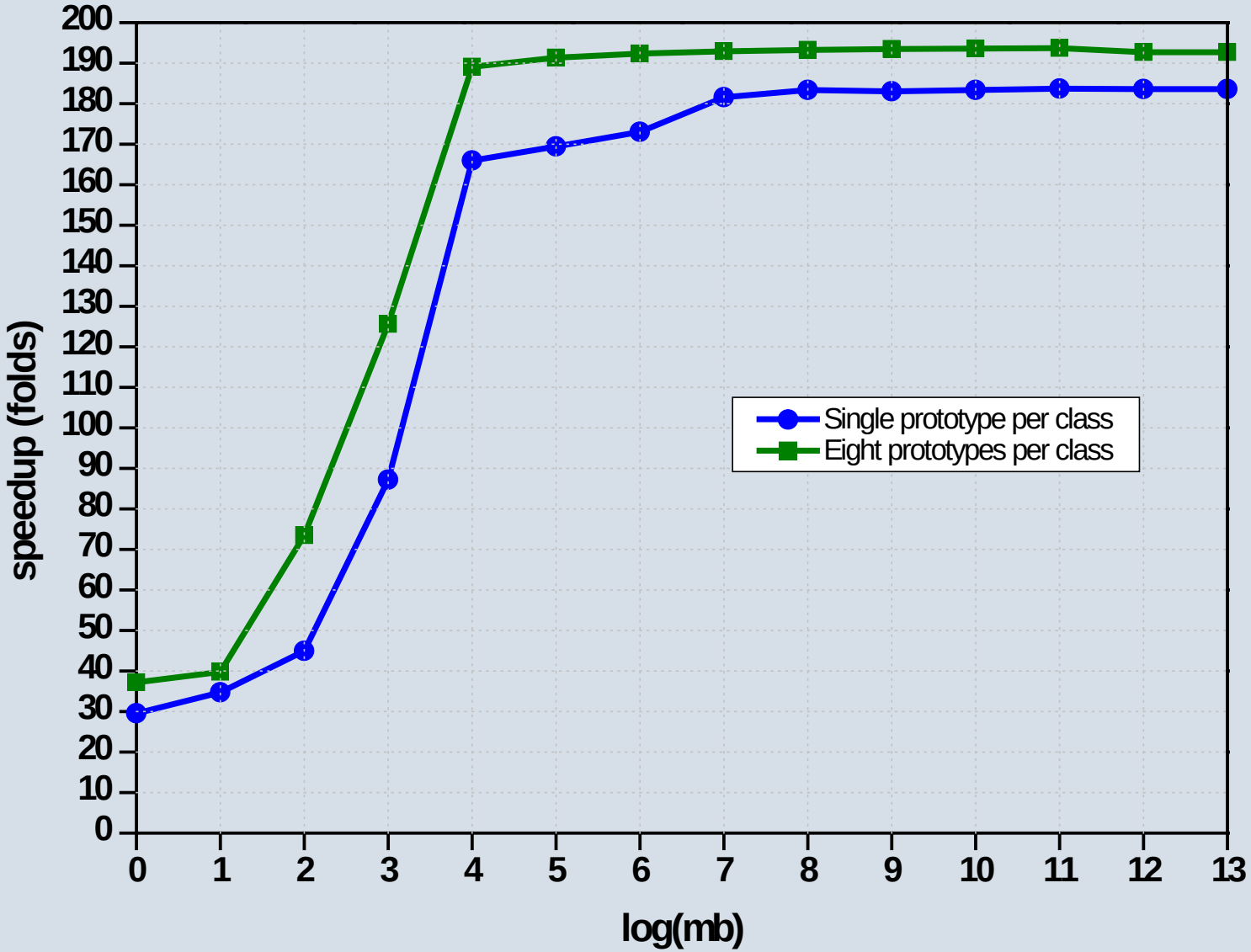
- **Minima Searching Comparison**





Evaluation on Chinese Character Recognition

- Speedup Benchmarks



Outline

1. Computing Challenging
2. CPU+GPU Framework
3. Scalable Prototype Learning
4. Conclusion

Conclusion

- **We Face Computing Challenging**
  - ✓ More data are available
  - ✓ Complex model may yield better performance
  - ✓ Besides Algorithmic Optimization, HSA is more manageable
- **CPU+GPU HSA**
  - ✓ State-of-the-art Practice
  - ✓ CUDA Basics
- **Scalable Prototype Learning**
  - ✓ Parallelize the building blocks
  - ✓ Design HSA
  - ✓ Hotspot Detection and Eliminate the Bottleneck
  - ✓ Programming GPU is so easy
  - ✓ We can port the algorithm with high scalability