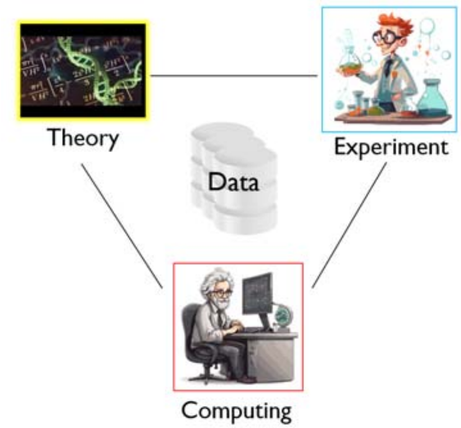
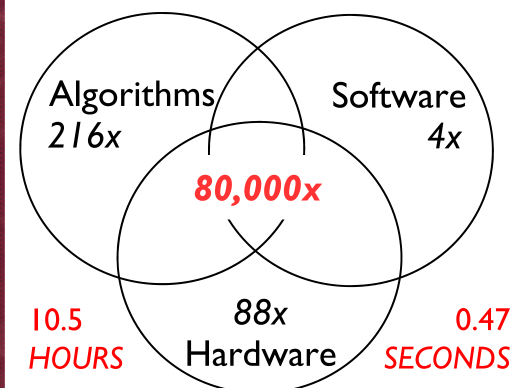
Synergistic Co-design: A Multi-Faceted Perspective

Wu Feng
wfeng@vt.edu

QCD at the Femtoscale in the Era of Big Data
Institute of Nuclear Theory





http://www.youtube.com/watch?v=zPBFenYg2Zk

SyNeRGy
http://synergy.cs.vt.edu

VIRGINIA TECH.

A Little Bit About Me …

- Education
  - Ph.D., Computer Science,
    U. Illinois at Urbana-Champaign, 1996
- Professional
  - Current Appointments
    - Professor and Elizabeth & James Turner Fellow; Departments of Computer Science, Electrical & Computer Engg., and Health Sciences; Virginia Tech
    - Director, SyNeRG Laboratory (http://synergy.cs.vt.edu/) → SEEC Center
    - Site Director, Center for Space, High-performance, and Resilient Computing (SHREC)
  - Previous Appointments & Professional Stints
    - *Academia:* Ohio State U. ('00-'03), Purdue U. ('98-'00), U. of Illinois at Urbana-Champaign ('96-'98)
    - *Government:* Los Alamos Nat'l Lab ('98-'06), NASA Ames Research Ctr ('93)
    - *Industry:* IBM T.J. Watson Rsch ('90), Vosaic ('97), Orion Multisystems ('04-'05), EnergyWare ('08-'10)

Summary: Re-visiting the Third Pillar of Science

1. The third pillar of science is simply COMPUTING, encompassing simulating physical reality ***and*** computing on the data.

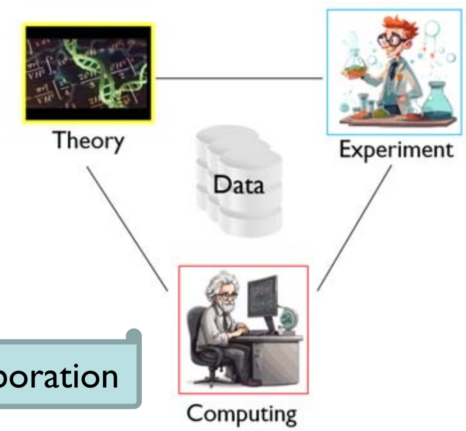
   ... for the technical layperson

2. Synergistic co-design of algorithms, software, and hardware can massively accelerate discovery.

   ... for scientific collaboration

3. Don't fool yourself and, in turn, fool the masses.

   ... for computer scientists



12 Ways to Fool the Masses

(David H. Bailey, NASA & LBL, 1991)

1. Quote only 32-bit performance results, ***not*** 64-bit results. → 2x speedup
2. Present performance figures for an inner kernel and then represent these figures as the performance of the entire application.

her
mber of
his fact
to a full
stems.

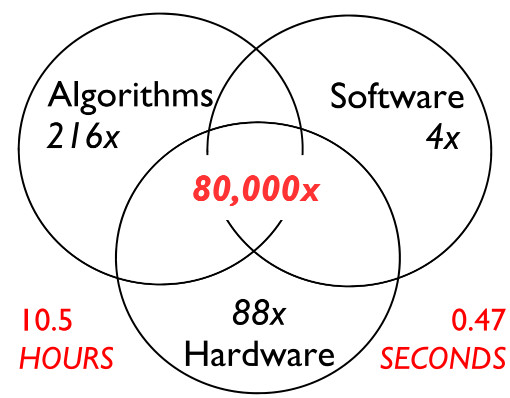
7. When direct run-time comparisons are required, compare with an old code on an obsolete system.
8. If Mflop/s rates must be quoted, base the operation count on the parallel [version], ***not*** on the best sequential [version].
9. Quote performance in terms of processor utilization, parallel speedups, or Mflop/s per dollar.
10. Mutilate the algorithm used in the parallel implementation to match the architecture.
11. Measure parallel run-times on a dedicated system but measure conventional run times in a busy environment.
12. If all else fails, show pretty pictures and videos, and don't talk about performance.

Debunking the 100X GPU vs. CPU Myth:
An Evaluation of Throughput Computing on CPU and GPU

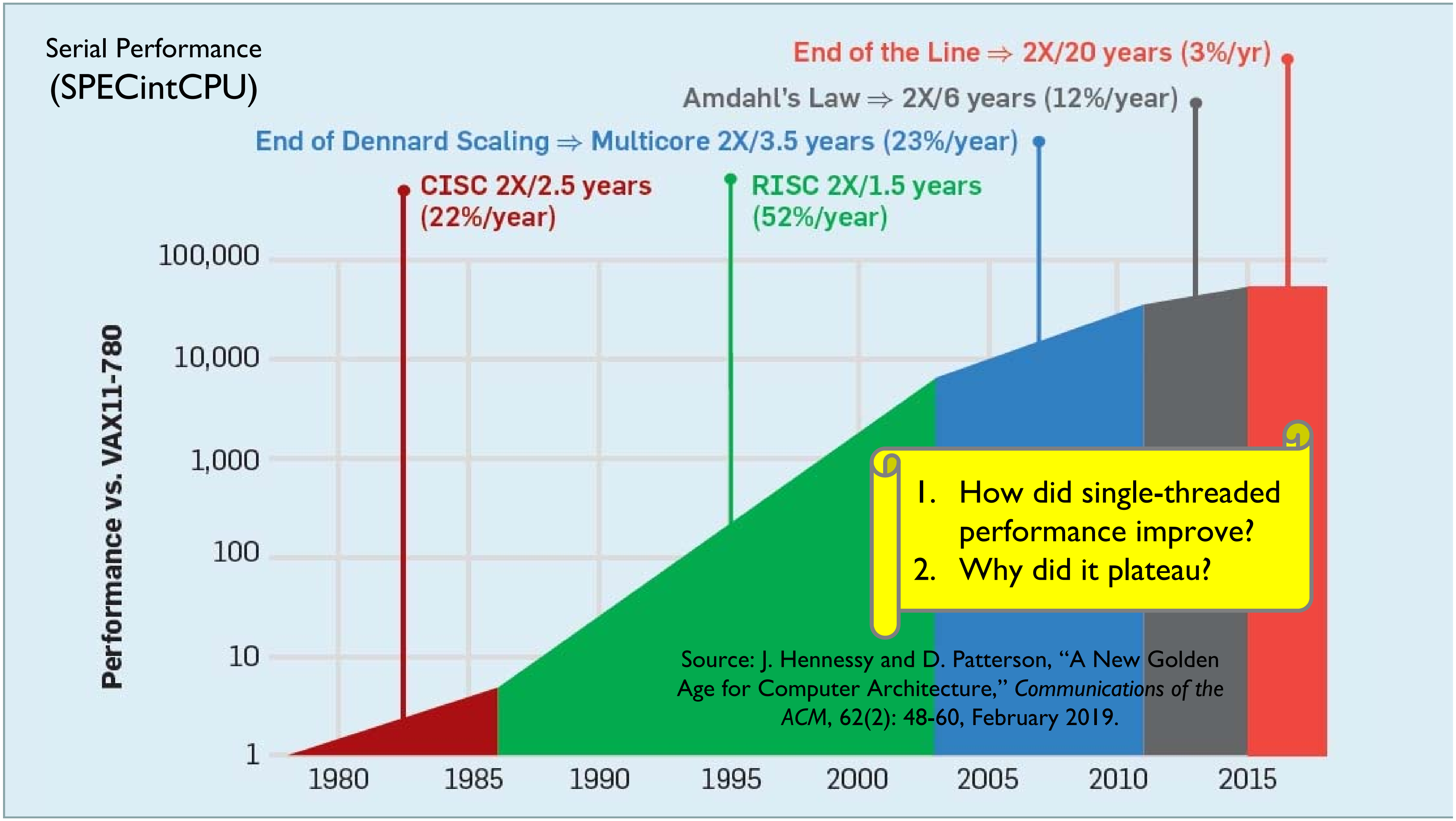
Victor W Lee[†], Changkyu Kim[†], Jatin Chhugani[†], Michael Deisher[†], Daehyun Kim[†], Anthony D. Nguyen[†], Nadathur Satish[†], Mikhail Smelyanskiy[†], Srinivas Chennupaty[*], Per Hammarlund[*], Ronak Singhal[*] and Pradeep Dubey[†]

victor.w.lee@intel.com

[†]Throughput Computing Lab, Intel Corporation
[*]Intel Architecture Group, Intel Corporation

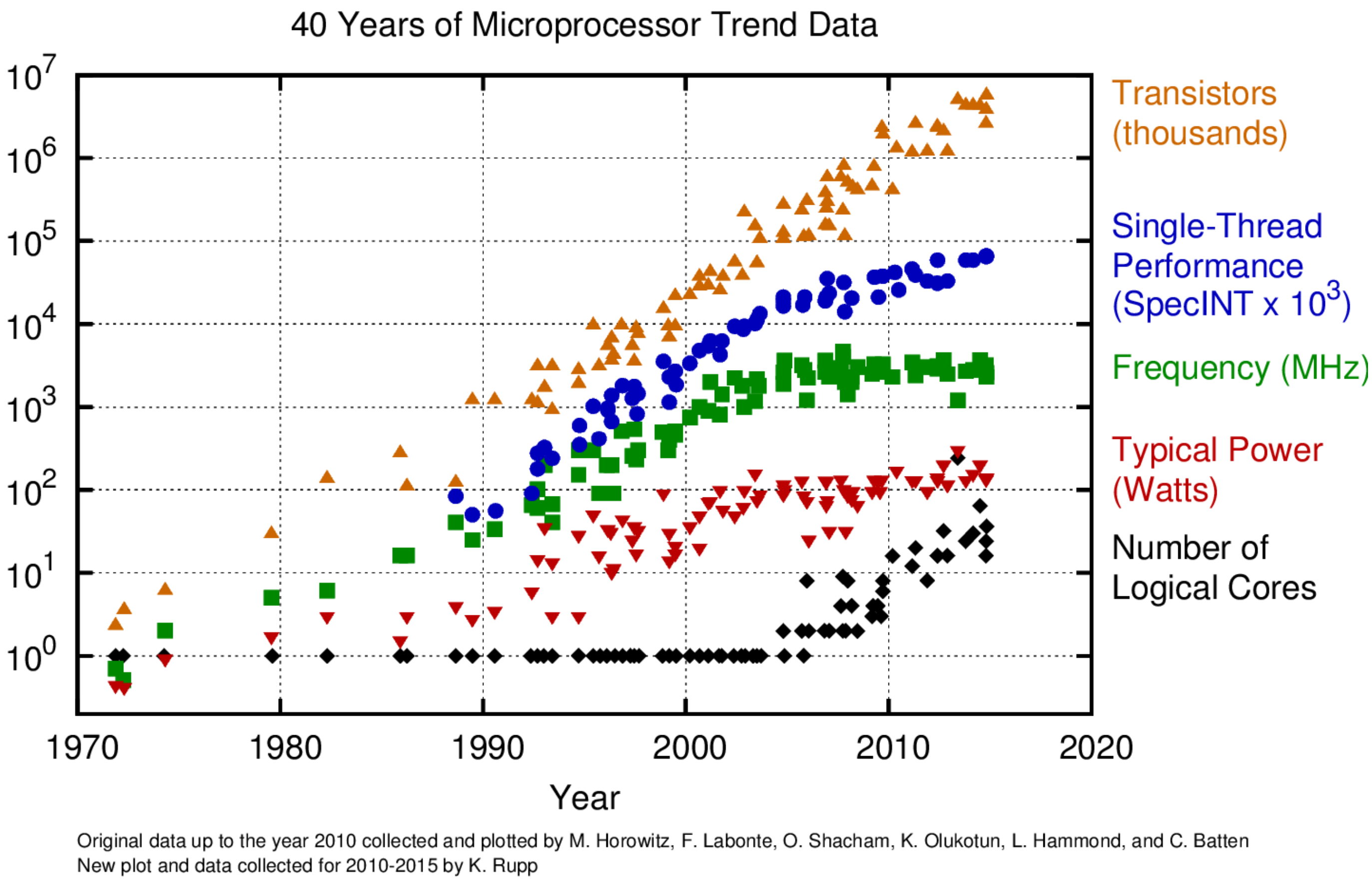


http://www.youtube.com/watch?v=zPBFenYg2Zk



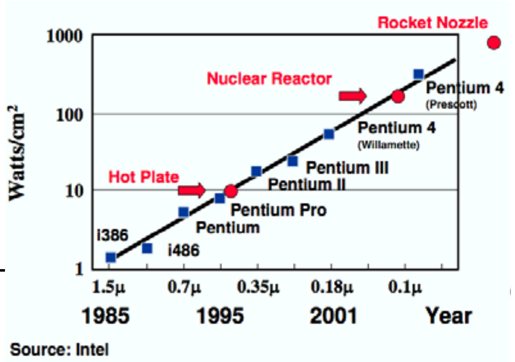
1. How did single-threaded performance improve?
2. Why did it plateau?

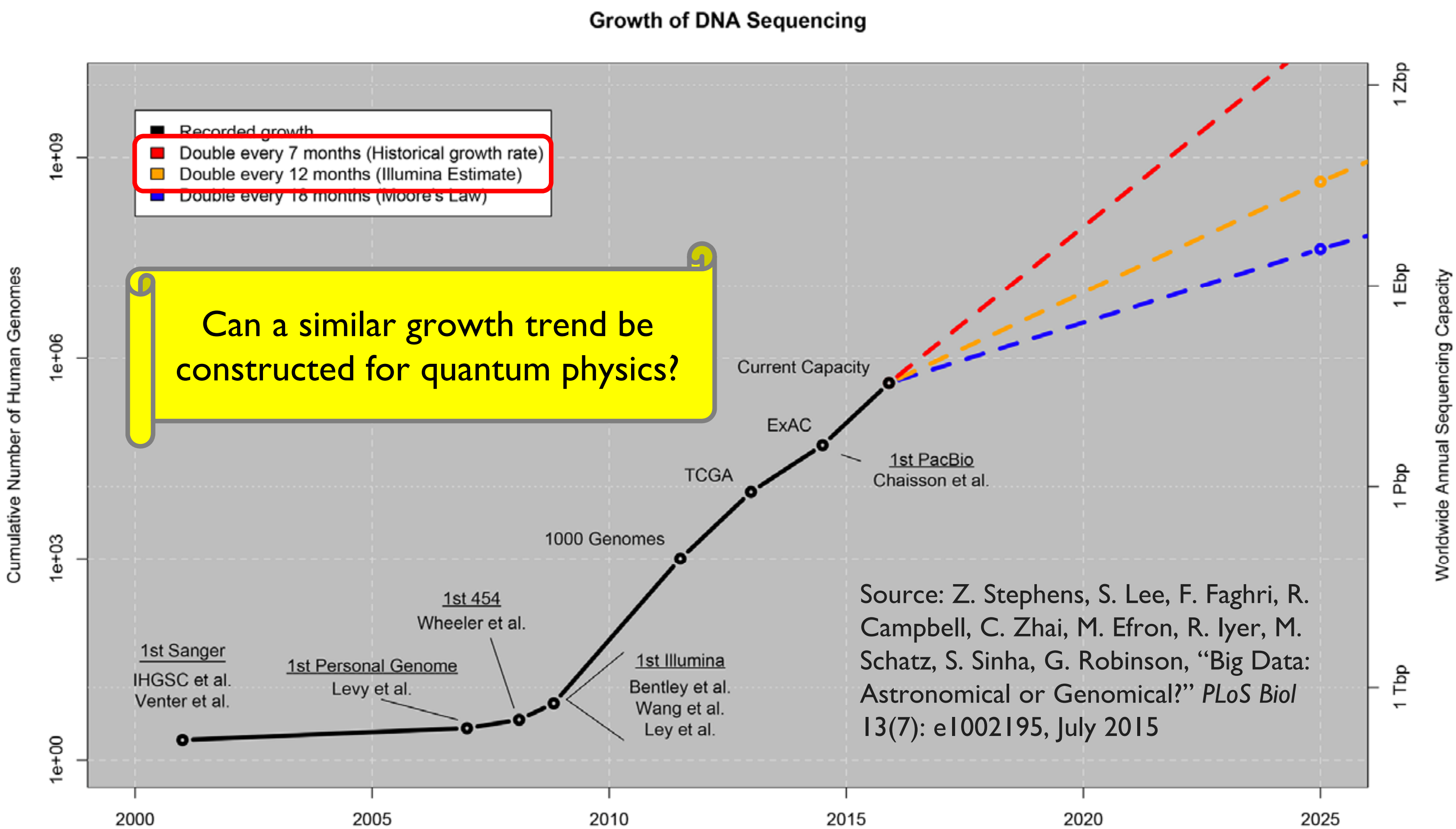
Source: J. Hennessy and D. Patterson, "A New Golden Age for Computer Architecture," *Communications of the ACM*, 62(2): 48-60, February 2019.



https://www.karlrupp.net/wp-content/uploads/2015/06/40-years-processor-trend.png

Intel's Paxville: too slow, too hot, too dumb, 2005.



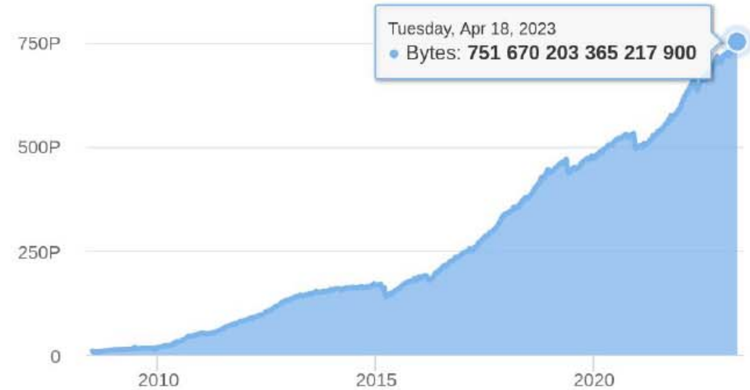


Growth of DNA Sequencing
Recorded growth
Double every 7 months (Historical growth rate)
Double every 12 months (Illumina Estimate)
Double every 18 months (Moore's Law)
Can a similar growth trend be constructed for quantum physics?
Current Capacity
ExAC
TCGA
1st PacBio
Chaisson et al.
1000 Genomes
1st 454
Wheeler et al.
1st Sanger
IHGSC et al.
Venter et al.
1st Personal Genome
Levy et al.
1st Illumina
Bentley et al.
Wang et al.
Ley et al.
Source: Z. Stephens, S. Lee, F. Faghri, R. Campbell, C. Zhai, M. Efron, R. Iyer, M. Schatz, S. Sinha, G. Robinson, "Big Data: Astronomical or Genomical?" PLoS Biol 13(7): e1002195, July 2015
Cumulative Number of Human Genomes
1e+00
1e+03
1e+06
1e+09
2000
2005
2010
2015
2020
2025
1 Tbp
1 Pbp
1 Ebp
1 Zbp
Worldwide Annual Sequencing Capacity

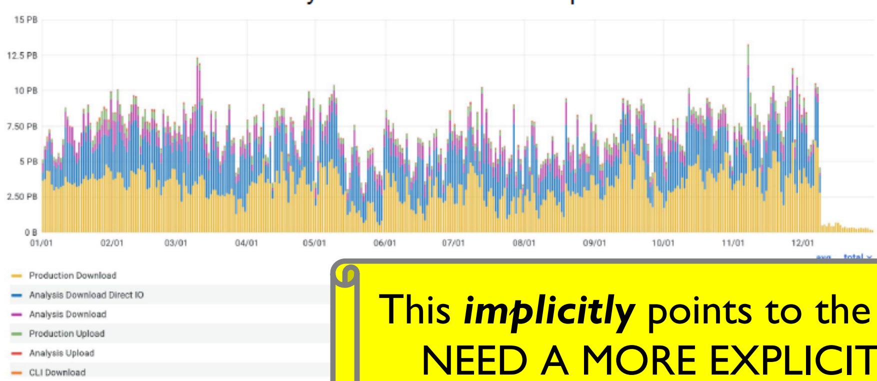
ATLAS Distributed Computing today

- A few numbers showing the scale of ATLAS data
  - 1B+ files, 750+ PB of data, 400+ Hz interaction
  - 120 data centres, 5 HPCs, 3 clouds, 1000+ users
  - 1.2 Exabytes/year transferred
  - 2.7 Exabytes/year uploaded & downloaded
- Expect an increase of at least one order of magnitude for the HL-LHC

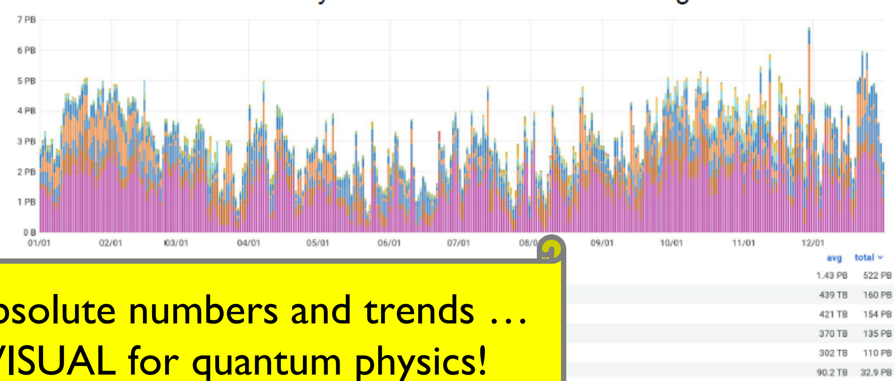
ATLAS data registered in Rucio



5+ PB/day data access for computation



2+ PB/day data transfers between storage



This ***implicitly*** points to the absolute numbers and trends …
NEED A MORE EXPLICIT VISUAL for quantum physics!

D. South. ATLAS Distributed Computing Evolution. CHEP 23, Norfolk, VA, USA. May 2023

Challenge

- The rate of growth in **big data** is ***far outstripping*** the rate at which computing can (brute-force) **compute** on the data.

Challenge

- The rate of growth in **big data** is ***far outstripping*** the rate at which computing can (brute-force) **compute** on the data.

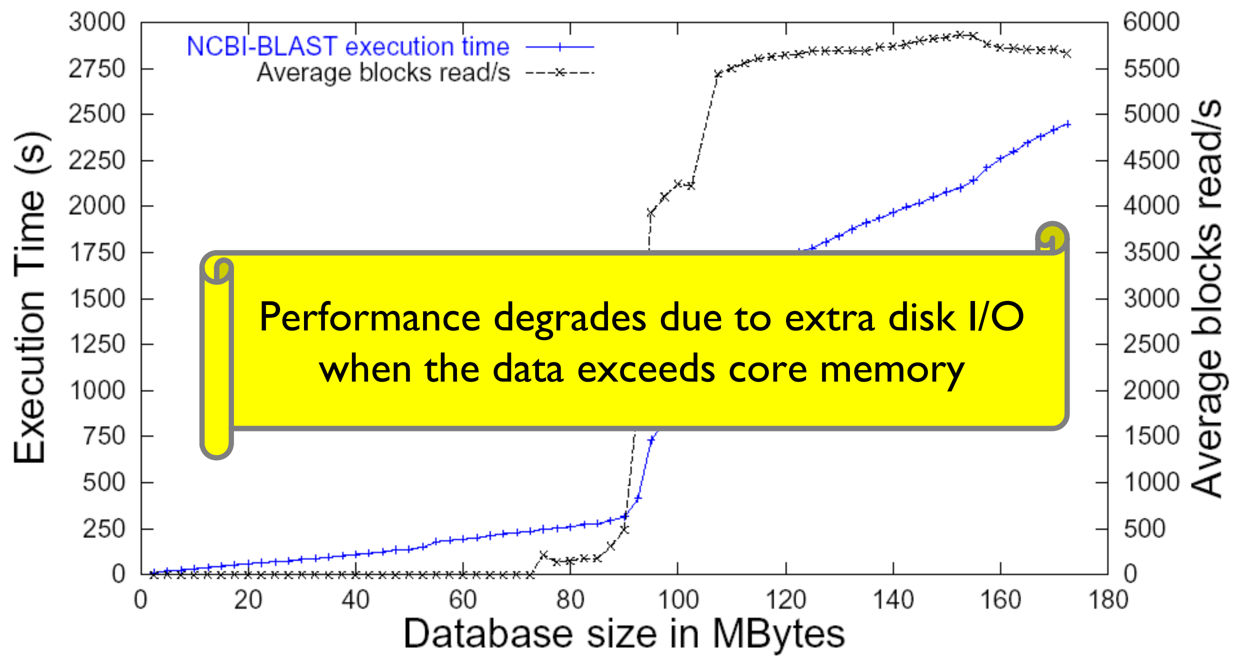
Approach

- Synergistic co-design of architecture, software, and in particular, algorithms to more ***efficiently*** and ***intelligently*** compute on the data.

Importance of Trend Graphs: Compute, Data, Compute/Data

- Impacts how programs should be written, e.g., 2004: BLAST → mpiBLAST

Standard BLAST pairwise sequence alignment (128MB RAM)



Computational Science (→ OpenDwarfs → Berkeley Dwarfs)

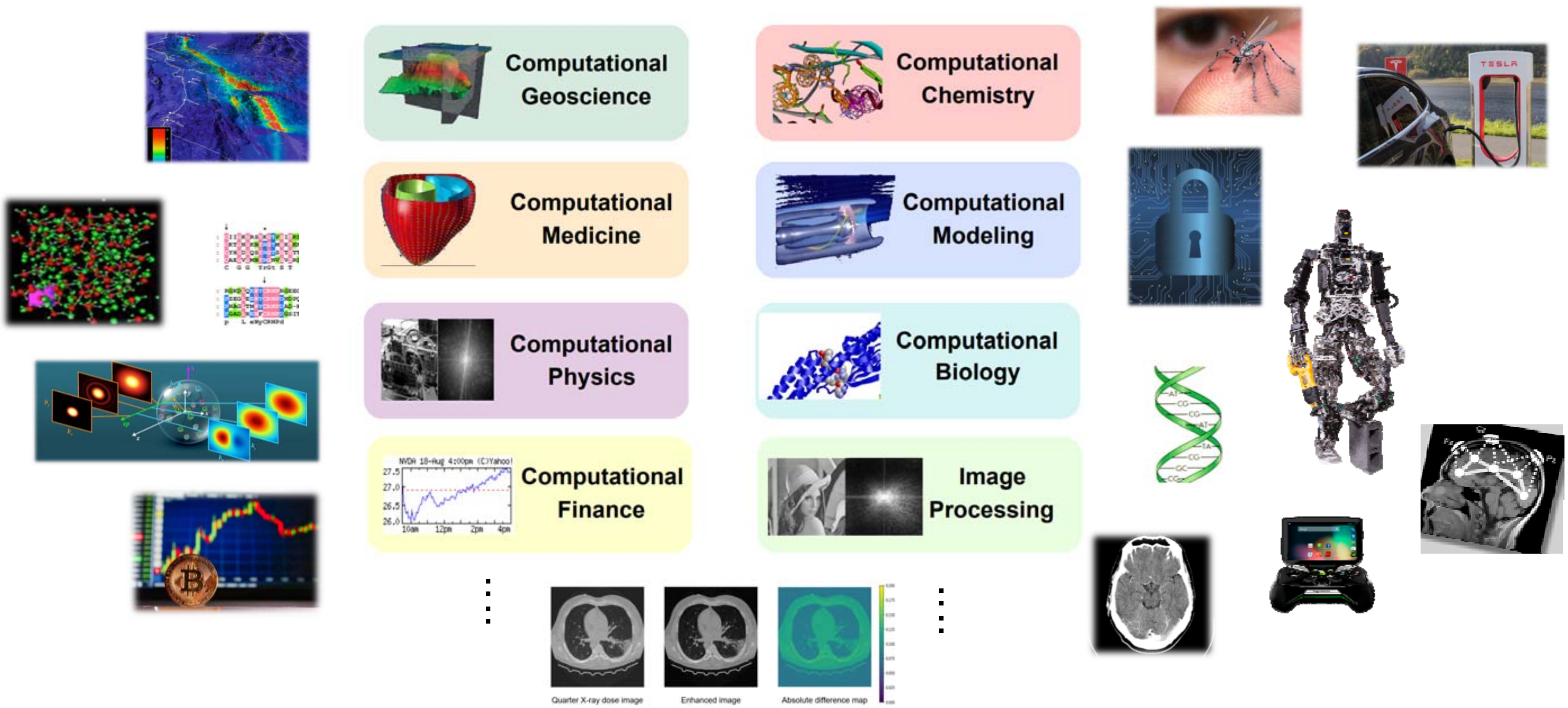
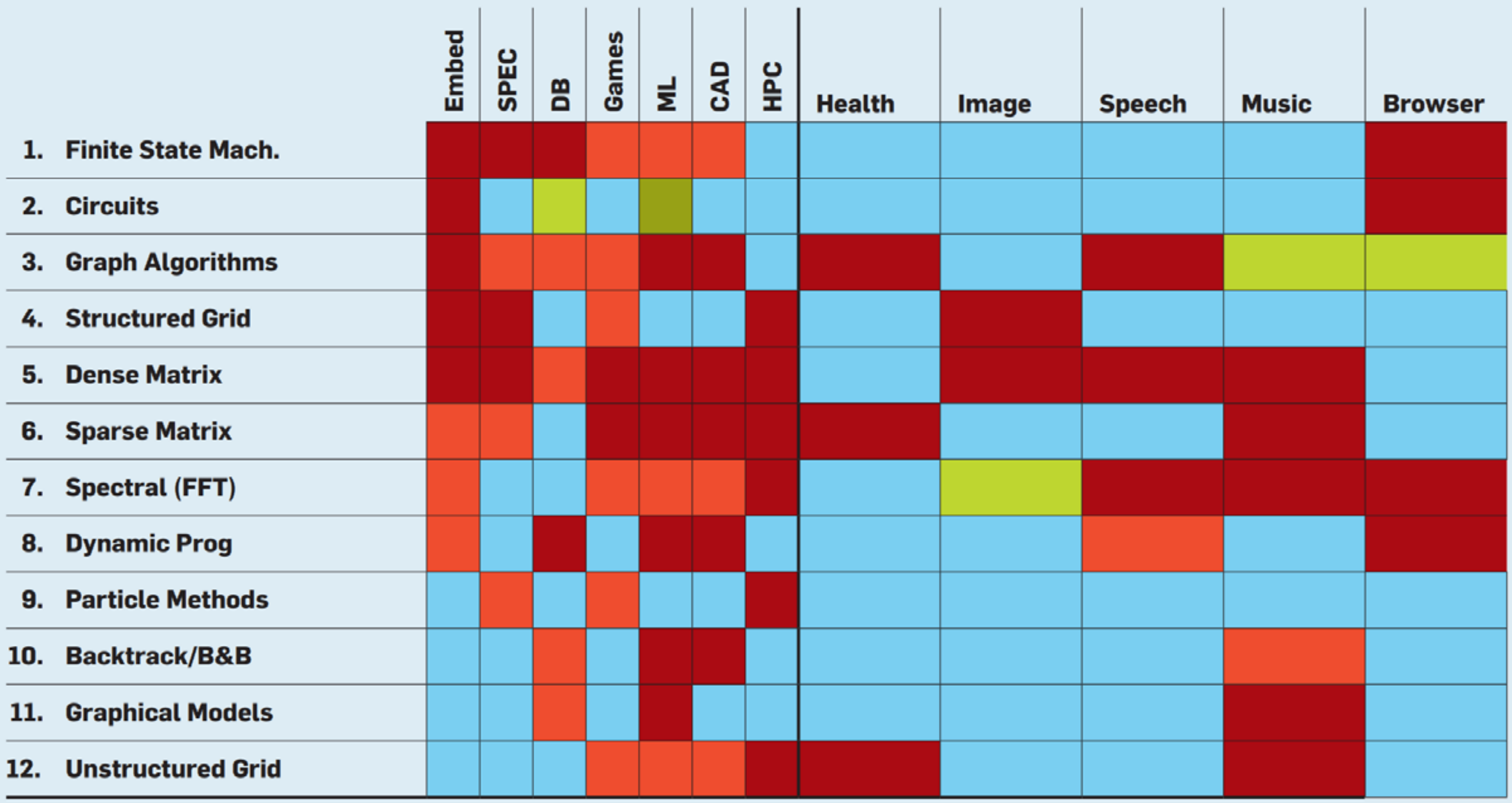
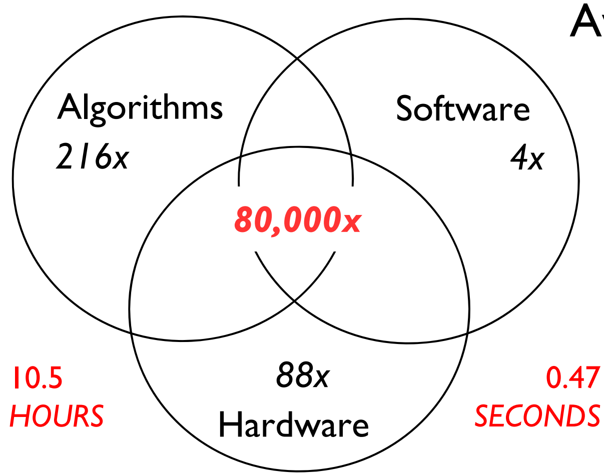


Figure 3. The color of a cell (for 12 computational patterns in seven general application areas and five Par Lab applications) indicates the presence of that computational pattern in that application; red/high; orange/moderate; green/low; blue/rare.



Molecular Dynamics → Cosmology → ???

- Primary computational dwarf? *N*-body method → particle method
- A computational dwarf (or pattern) describes a program's machinery, flow of resources, and outputs.



http://www.youtube.com/watch?v=zPBFenYg2Zk

Awesome for the domain scientist!

– Can run "what-if" simulations for rational drug design on a GPU server in his office, but

80,000x relative to *serial.* What if "level playing field?"

– 80,000 / 216 = 371 (algo. refactor)
– 371 / 16-core CPU = 23 (1 → 16 cores)
– 23 / 2 = 11 (DP → SP)
– 11 / "2" = 5 (calc → lookup)

12 Ways to Fool the Masses
(David H. Bailey, NASA & LBL, 1991)

…ance results, *not*
…up
…es for an inner

7. When direct run-time comparisons are required, compare with an old code on an obsolete system.
8. If Mflop/s rates must be quoted, base the operation count on the parallel [version],

Debunking the 100X GPU vs. CPU Myth:
An Evaluation of Throughput Computing on CPU and GPU

Race to Sequence the Human Genome

- Theory & Experiment (Collins@NIH)
  - Goal: Complete in 15 years
    1990 – 2005
  - Cost: $3,000M (1990-2000/2003)

- Computing (Venter@Celera)
  - Goal: Complete in 3 years & cheaper
    1998 - 2001
  - Cost: $300M (1998-2000/2003)

mpiBLAST

"String matching" → dynamic programming dwarf

Pairwise Sequence Alignment (Smith-Waterman Algorithm)

- Performs local sequence alignment by identifying similar regions between two strings of nucleic acid sequences or protein sequences.

Algorithm [edit]

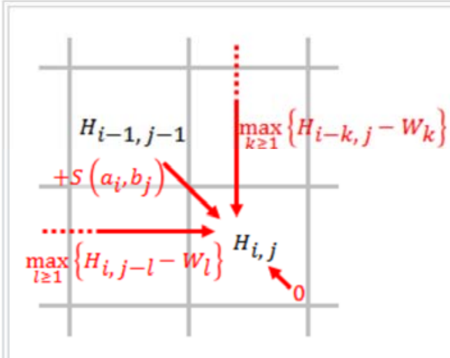
Let $A = a_1 a_2 \ldots a_n$ and $B = b_1 b_2 \ldots b_m$ be the sequences to be aligned, where n and m are the lengths of A and B respectively.

1. Determine the substitution matrix and the gap penalty scheme.
   - $s(a, b)$ - Similarity score of the elements that constituted the two sequences
   - W_k - The penalty of a gap that has length k
2. Construct a scoring matrix H and initialize its first row and first column. The size of the scoring matrix is $(n+1) * (m+1)$. The matrix uses 0-based indexing.

$$H_{k0} = H_{0l} = 0 \quad for \quad 0 \le k \le n \quad and \quad 0 \le l \le m$$

3. Fill the scoring matrix using the equation below.

$$H_{ij} = \max \begin{cases} H_{i-1,j-1} + s(a_i, b_j), \\ \max_{k \ge 1}\{H_{i-k,j} - W_k\}, \\ \max_{l \ge 1}\{H_{i,j-l} - W_l\}, \\ 0 \end{cases} \qquad (1 \le i \le n, 1 \le j \le m)$$



Scoring method of the Smith–Waterman algorithm

Fill the scoring matrix

		T	G	T	T	A	C	G	G
	0	0	0	0	0	0	0	0	0
G	0	0	3	1	0	0	0	3	3
G	0	0	3	1	0	0	0	3	6
T	0	3	1	6	4	2	0	1	4
T	0	3	1	4	9	7	5	3	2
G	0	1	6	4	7	6	4	8	6
A	0	0	4	3	5	10	8	6	5
C	0	0	2	1	3	8	13	11	9
T	0	3	1	5	4	6	11	10	8
A	0	1	0	3	2	7	9	8	7

Source: Wikipedia

Wavefront Loops

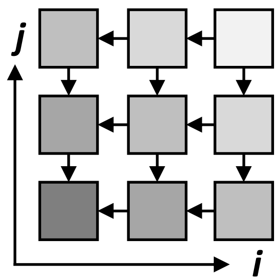
- Update each entry of a grid based on already-updated values from its neighbors
- Used in many scientific applications, e.g., PDE solver, sequence alignment tools, etc.

Example: a wavefront loop (2D matrix)

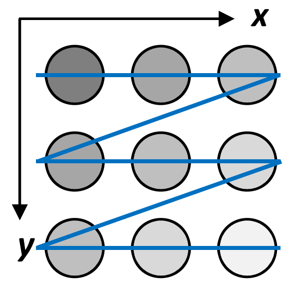
Neither loop can be parallelized.

```
for(int i = 0; i < m; i++)
  for(int j = 0; j < n; j++)
    A[i][j] = A[i][j-1] * 0.5 + A[i-1][j] * 0.5;
```

Data Dependence (Iteration Space)



Memory Access (Memory Space A[y][x])



Wavefront Loops

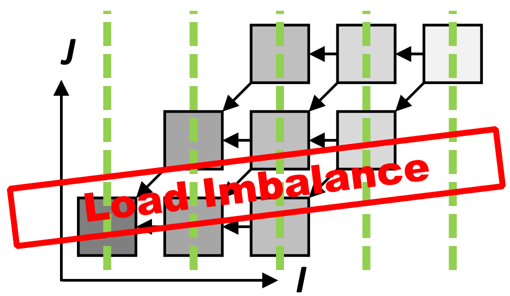
- Update each entry of a grid based on already-updated values from its neighbors
- Used in many scientific applications, e.g., PDE solver, sequence alignment tools, etc.

Example: a wavefront loop (2D matrix) -- Tra

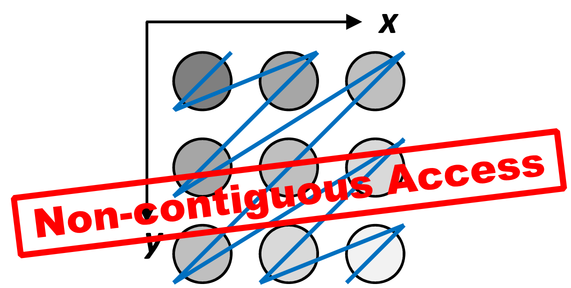
J-loop can be parallelized.

```
for(int I = 0; I < m+n-1; I++)
  for(int J = max(0, I-n+1); J < min(m, I+1); J++)
    A[J][I-J] = A[J][I-J-1] * 0.5 + A[J-1][I-J] * 0.5;
```

Data Dependence (Iteration Space)

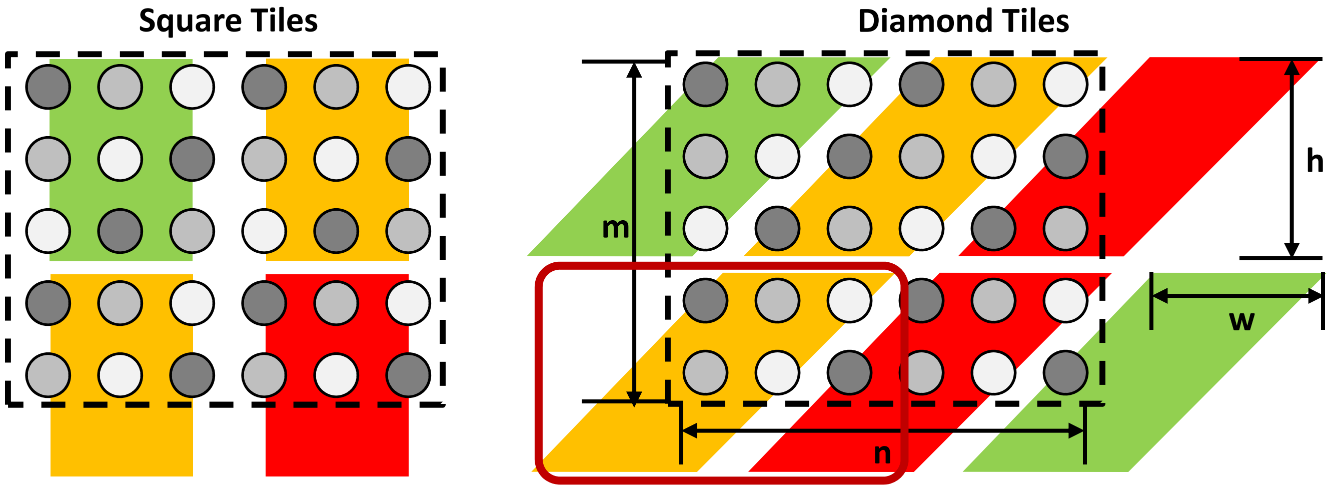


Memory Access (Memory Space A[y][x]**)**



Existing Parallel Solutions

- Tiling-based solutions and their limitations
  - Problem 1: Wasted memory and computing resources



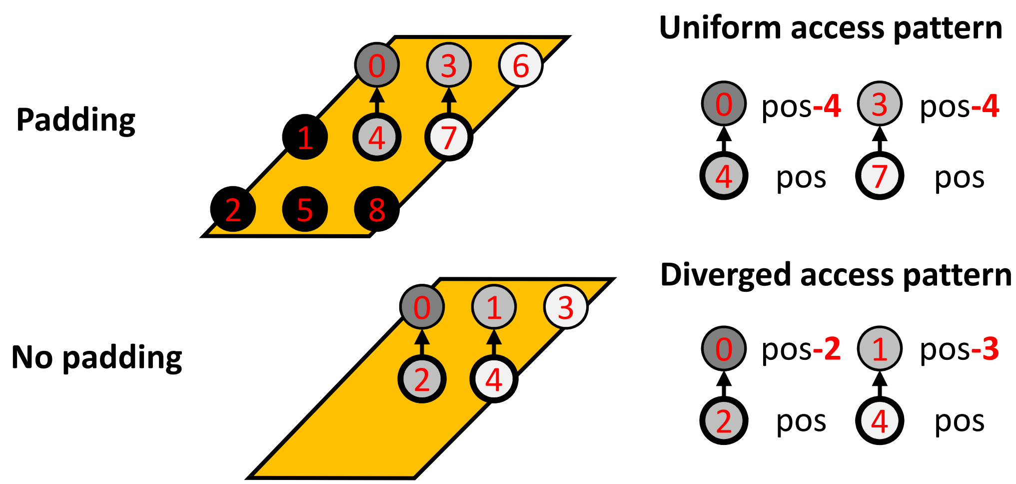
Tiles with same color can be executed in parallel

Non-contiguous memory access still exists

Much memory space will be wasted (The rate of effective memory usage ≈$n/(n+h)$)

Existing Parallel Solutions

- Tiling-based solutions and their limitations
  - Problem 1: Wasted memory and computing resources



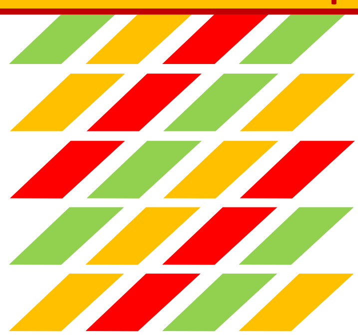
Padding-free strategy may greatly increase the complexity of indexing and lead to more branches in GPU kernels

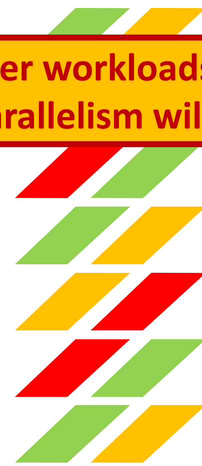
Existing Parallel Solutions

- Tiling-based solutions and their limitations
  - Problem 1: Wasted memory and computing resources
  - Problem 2: Layout transformation overhead
  - Problem 3: Task scheduling

For some workloads, sufficient parallelism can be exposed

For other workloads, insufficient parallelism will be met





Tiles with same color can be executed in parallel

For some workloads, tiling-based solution may lose efficiency because of the small amount of tiles along anti-diagonals

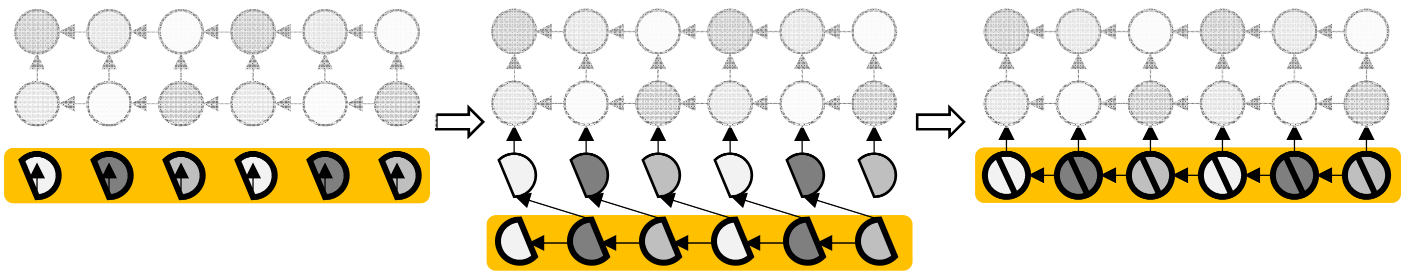
Existing Parallel Solutions

- Compensation-based solutions and their limitations
  - Problem 1: Global synchronizations
  - Problem 2: Limited usage in sequence alignment algorithms

❶ Compute partial results by ignoring horizontal dependency

❷ Compensate the partial results

❸ Combine the results from ❶ and ❷



Multiple expensive global synchronizations are required for processing each row; the compensation-based solution works well for string-matching operations

Our Highly ***Efficient*** Wavefront Parallelism

Memory Access Optimization

② **Locality**
tiling [3,4,5]

① **Direct Parallelism**
loop transf. [1,2]

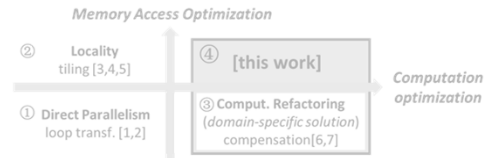
④ **[Our Work]**

③ **Comput. Refactoring**
(*domain-specific solution*)
compensation[6,7]

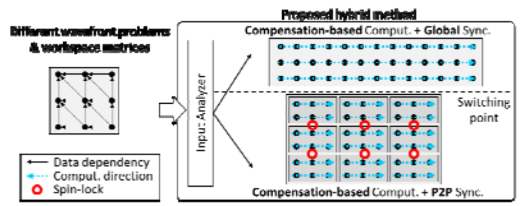
Computation optimization

Outline

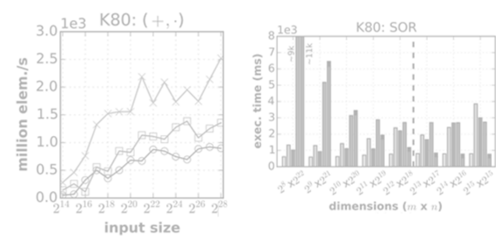
- Introduction
- Motivation



- Our Method
  - Compensation-based Method
  - GPU Implementation
  - Hybrid Parallel Strategy



- Evaluation
  - Weighted-scan Kernel Performance
  - Wavefront Kernel Performance



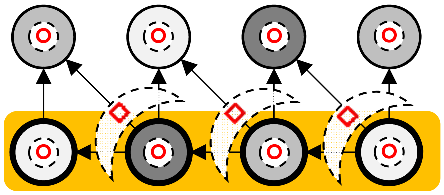
Compensation-based Method

- Wavefront Pattern

$$A_{i,j} = (A_{i,j-1} \circ b_0) \diamond (A_{i-1,j} \circ b_1) \diamond (A_{i-1,j-1} \circ b_2)$$

$\circ$ *generic distribution operator (for adding weights)*

$\diamond$ *generic accumulation operator (for adding neighbors)*



- Compensation-based Method

Step 1: $\tilde{A}_{i,j} = (A_{i-1,j} \circ b_1) \diamond (A_{i-1,j-1} \circ b_2)$

Step 2: $B_{i,j} = \begin{cases} \sum_{u=0}^{j-1} (\tilde{A}_{i,u} \circ \prod_{v=u}^{j-1} b_0) & \text{when } \circ \neq \diamond \\ \sum_{u=0}^{j-1} (\tilde{A}_{i,u} \diamond b_0) & \text{when } \circ = \diamond \end{cases}$

Step 3: $A_{i,j} = \tilde{A}_{i,j} \diamond B_{i,j}$

This is valid when (1) $\circ$ has the distributive property over $\diamond$; (2) $\circ$ is same with $\diamond$. *

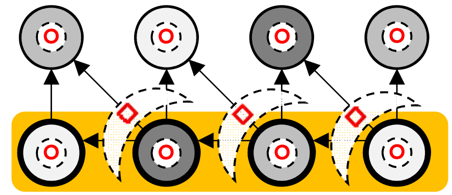
* The mathematical proof is included in our paper.

Compensation-based Method

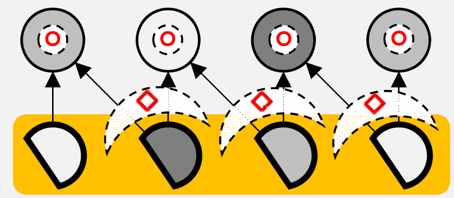
- Wavefront Pattern

$$A_{i,j} = (A_{i,j-1} \circ b_0) \diamond (A_{i-1,j} \circ b_1) \diamond (A_{i-1,j-1} \circ b_2)$$

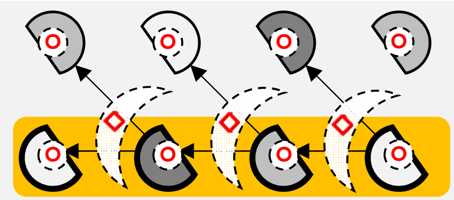
$\circ$ *generic distribution operator (for adding weights)*
$\diamond$ *generic accumulation operator (for adding neighbors)*



-



Step 1: $\tilde{A}_{i,j} = (A_{i-1,j} \circ b_1) \diamond (A_{i-1,j-1} \circ b_2)$



Step 2: $B_{i,j} = \begin{cases} \sum_{u=0}^{j-1} (\tilde{A}_{i,u} \circ \prod_{v=u}^{j-1} b_0) & \text{when } \circ \neq \diamond \\ \sum_{u=0}^{j-1} (\tilde{A}_{i,u} \diamond b_0) & \text{when } \circ = \diamond \end{cases}$

Step 3: $A_{i,j} = \tilde{A}_{i,j} \diamond B_{i,j}$

Compensation-based Method

- Wavefront loops can be expressed as compensation-based parallelism patterns
- SOR (*Successive Over-Relaxation*) Solver:

  ($\diamond$, $\circ$) maps to $(+, \cdot)$

```
A[i][j] = (A[i][j] + A[i][j-1] + A[i-1][j] +
           A[i+1][j] + A[i][j+1]) / 5;
```

- SW (Smith-Waterman):

  ($\diamond$, $\circ$) maps to $(\max, +)$

```
A[i][j] = max(A[i][j-1] - 2, A[i-1][j] - 2,
              A[i-1][j-1] + s(i,j), 0);
```

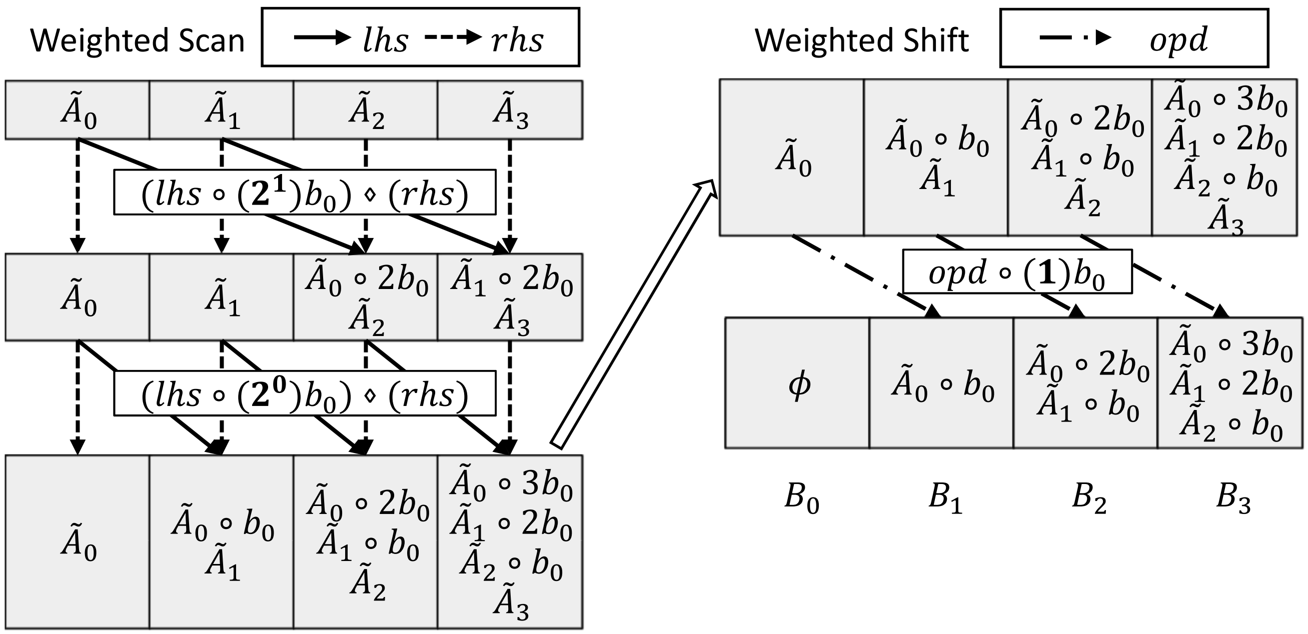
- SAT (Summed-Area Table):

  ($\diamond$, $\circ$) maps to $(+, +)$

```
A[i][j] = p[i][j] + A[i][j-1] + A[i-1][j] - A[i-1][j-1];
```

GPU Implementation

- Step 2 of the compensation-based method is the critical part: "***Weighted Scan***"*



* which also includes a weighted shift operation

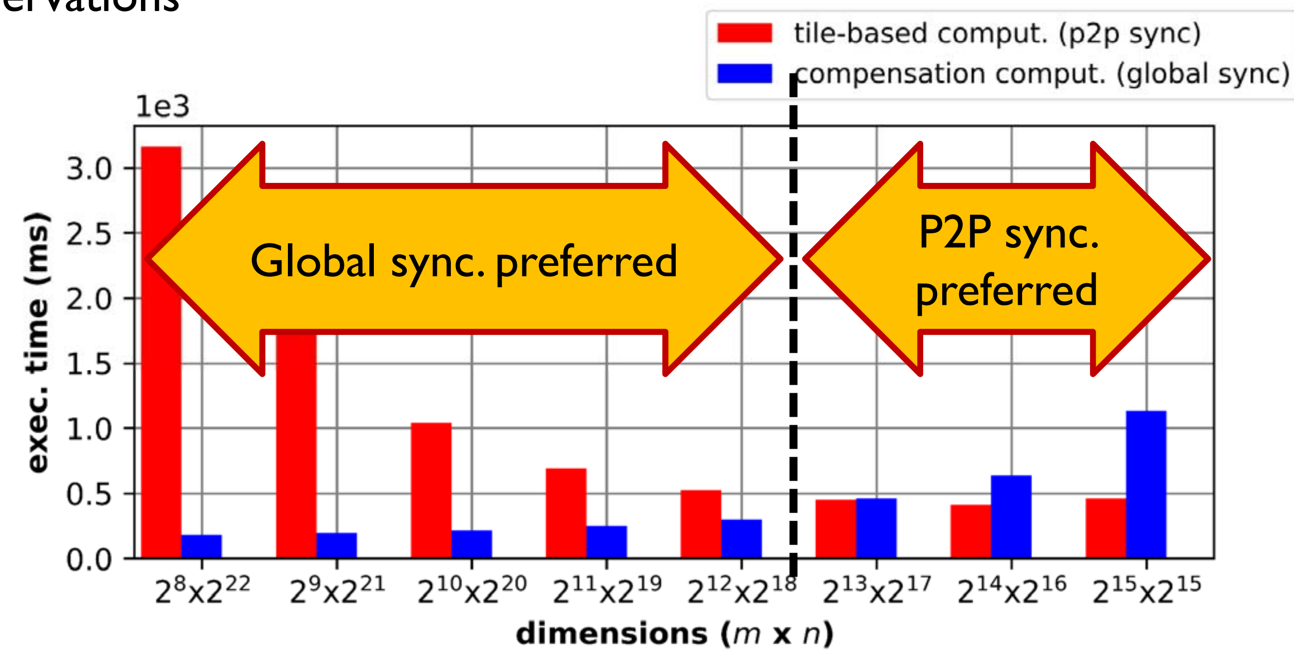
GPU Implementation

- Step 2 of the compensation-based method is the critical part: "***Weighted Scan***"
- Our algorithm handles the changing weights during each stages of the operations
- A hierarchical design is used for GPUs
  - *Register level*: compute how the preceding neighbor affects the current one via <u>*data shuffle instructions*</u>
  - *Shared memory level*: compute how the preceding **"warp"*** of neighbors affect the current one via <u>*shared memory access*</u>
  - *Global memory level*: compute how the preceding **"block"*** of neighbors affect the current one via <u>*global memory access*</u>

* which are thread organization units in NVIDIA GPU terminology

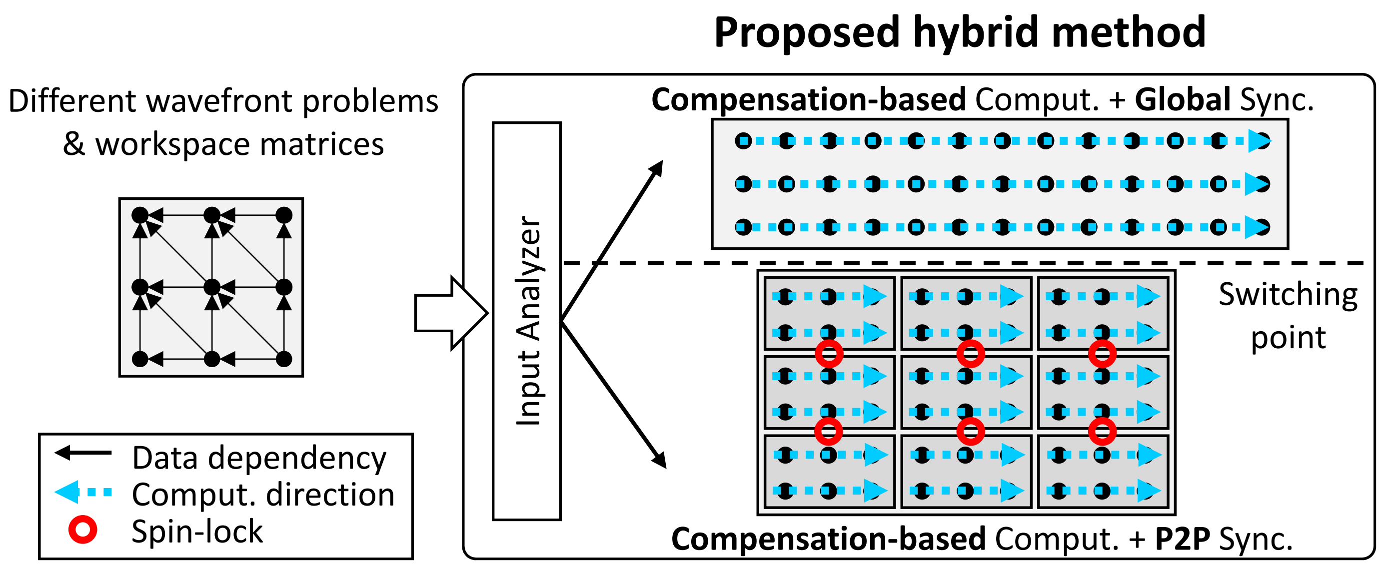
Hybrid Parallel Strategy

- *Is the compensation-based method sufficient for any types of workloads?*
- Observations



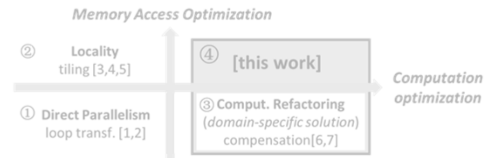
Hybrid Parallel Strategy

- Our hybrid design switches to the appropriate parallel method, based on the input workload
- All the computation follows the compensation-based parallelism pattern

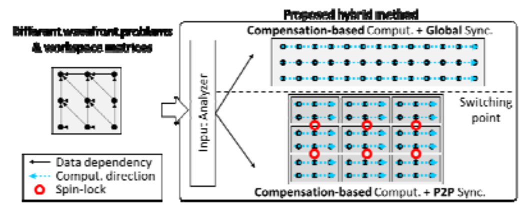


Outline

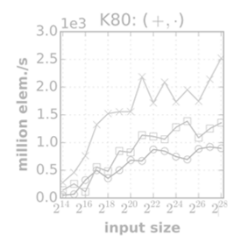
- Introduction
- Motivation

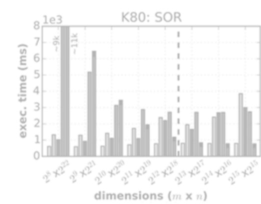


- Our Method
  - Compensation-based Method
  - GPU Implementation
  - Hybrid Parallel Strategy



- Evaluation
  - Weighted-scan Kernel Performance
  - Wavefront Kernel Performance





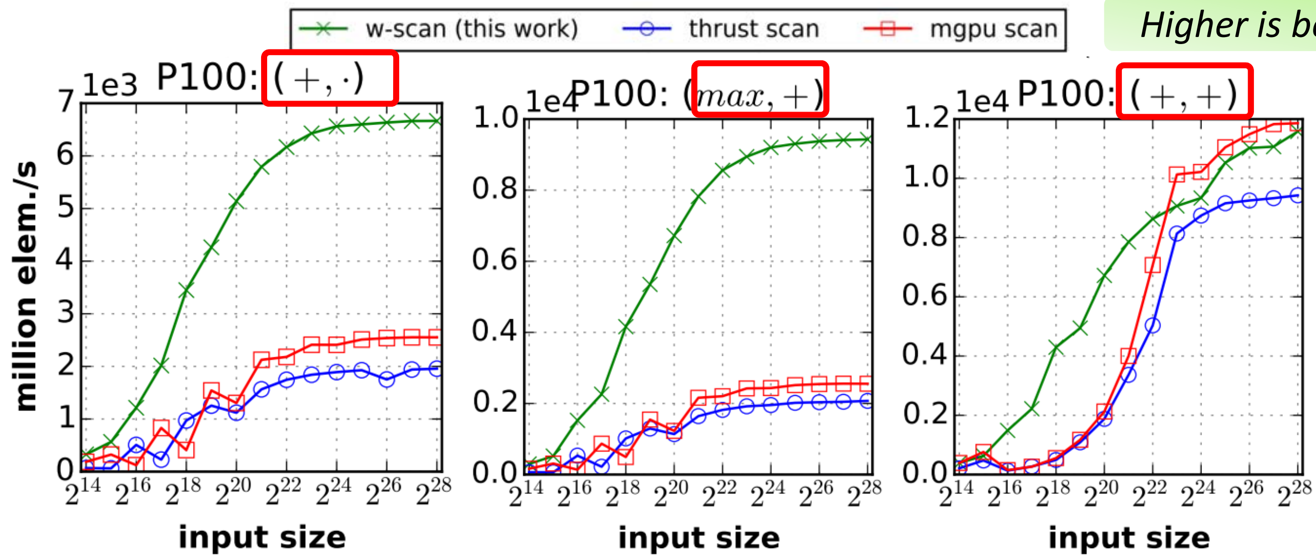
Experimental Platforms

- **nVidia Tesla K80 (Kepler-K80)**, 2496 CUDA cores @ 824 MHz, 240 GB/s bandwidth
- **nVidia Pascal P100 (Pascal-P100)**, 3584 CUDA cores @ 405 MHz, 720 GB/s bandwidth *
- Our **Weighted Scan** vs. other tools
  a. Thrust v.1.8.1 (thrust::exclusive_scan w/ custom comparator)
  b. ModernGPU v.2.0 (mgpu::scan w/ custom comparator)
     - Using 1D array of data to mimic different rows
- Our **Hybrid Wavefront** kernel vs.
  a. Tile-based methods [`15] (incl. square & diamond tiles)
  b. Compensation-based methods [`12, `16, this work]
     - Using 2D array of data to mimic different workloads

* We only show the performance results of P100 GPU here.

Weighted-scan Kernel Performance

- Processing a row of data with variable sizes

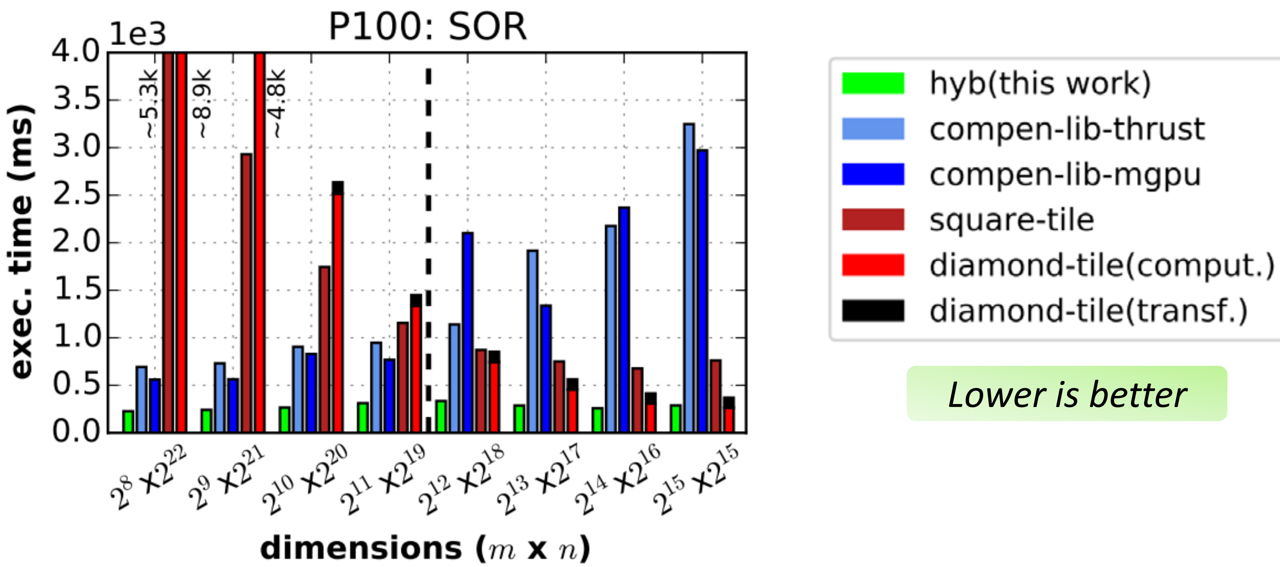


Higher is better

  – For $\circ \neq \diamond$, our method delivers significant performance benefit (*mainly because we can calculate the distance-related weights more* ***efficiently*** *in the kernel*)
  – For $\circ = \diamond$, our method reduces to an ordinary scan kernel

Wavefront Kernel Performance

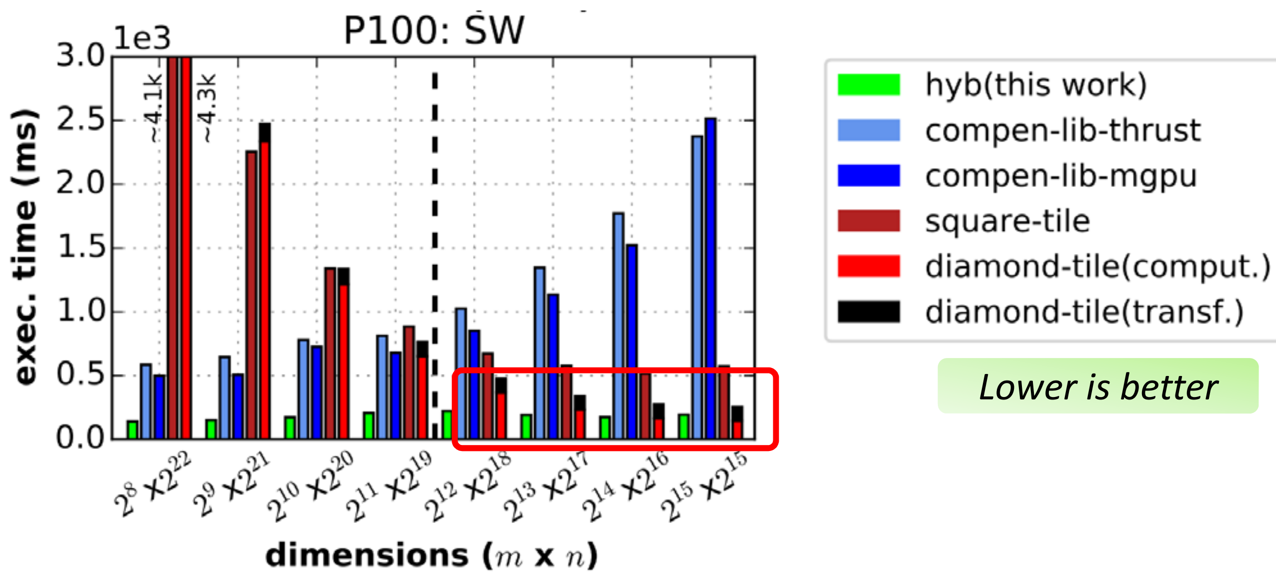
- Using SOR, SW, and SAT as representative wavefront kernels
- Processing 2D matrices of data with variable dimensions



  - Our method always delivers better performance than previous solutions

Wavefront Kernel Performance

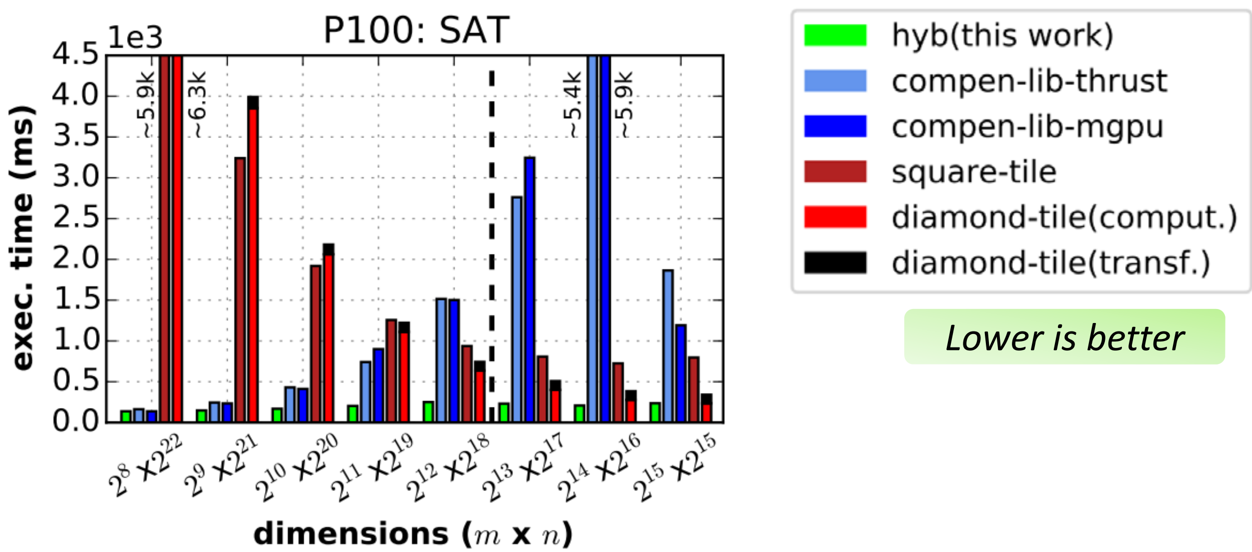
- Using SOR, SW, and SAT as representative wavefront kernels
- Processing 2D matrices of data with variable dimensions



  - The transformation overhead becomes non-negligible for the diamond-tile method

Wavefront Kernel Performance

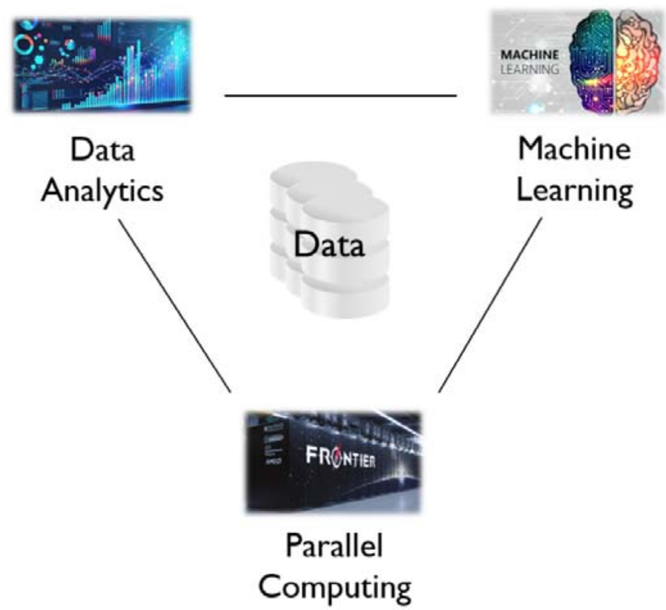
- Using SOR, SW, and SAT as representative wavefront kernels
- Processing 2D matrices of data with variable dimensions



  - Our hybrid method exhibits superior performance *regardless* of the workloads and wavefront types

At the Synergistic Intersection of Parallel Computing, Data Analytics, and Machine Learning

Wu Feng
wfeng@vt.edu



Challenge

- The rate of growth in **big data** is ***far outstripping*** the rate at which computing can (brute-force) **compute** on the data.

Approach

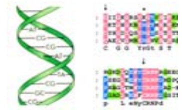
- Synergistic co-design of architecture, software, and in particular, algorithms to more ***efficiently*** and ***intelligently*** compute on the data.

At the Synergistic Intersection of Parallel Computing, Data Analytics, and Machine Learning

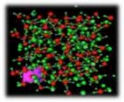
- Systems-Oriented
  - Automated GPU Blocksize Tuning via Iterative Machine Learning (Cui)
  - Scalable I/O for Deep Learning (Pumma)

- Applications-Oriented
  - SparkLeBLAST: High-Productivity DNA Sequence Search (Youssef)
  - Visual Data Analytics (Dash, in collaboration with C. North CS@VT)
  - Data-Oriented Computational Fluid Dynamics (Cui)
  - Understanding Carcinogenesis (Dash, in collaboration with VCOM)
  - Graph Analytics (Wanye, in collaboration with MIT LL)
  - Biomedical Imaging (Goel et al., in collaboration with BEAM@VT)

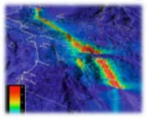
Sequence Alignment



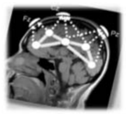
Molecular Dynamics



Earthquake Modeling



Neuro-informatics



CFD for Mini-Drones

Cyber-Security

Automated GPU Blocksize Tuning via Iterative ML

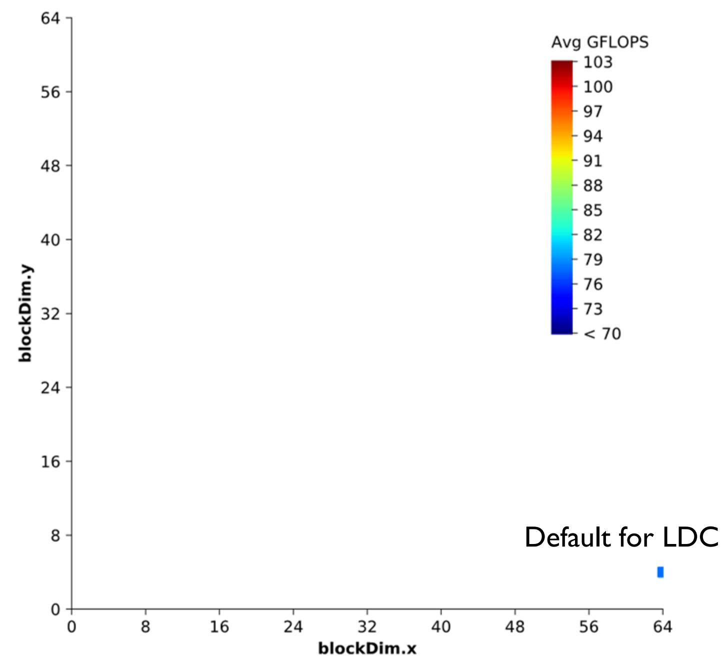
- **Problem**

  Many parameters to tune to achieve best performance
  - ✓ Thread block size
  - ✓ # streams
  - ✓ Register usage
  - ✓ Compiler optimization flags
  - ✓ *… and so on*

  O(millions)* potential software configurations for the *same code
- Our Focus
  - ✓ ***Thread block size***
- Example
  - ✓ Lid-driven cavity (LDC) code with varying GPU thread block size (NVIDIA K20m GPU)

1 <= {blockDim.x , blockDim.y} <= 1024

1 <= blockDim.x ∗ blockDim.y <= 1024

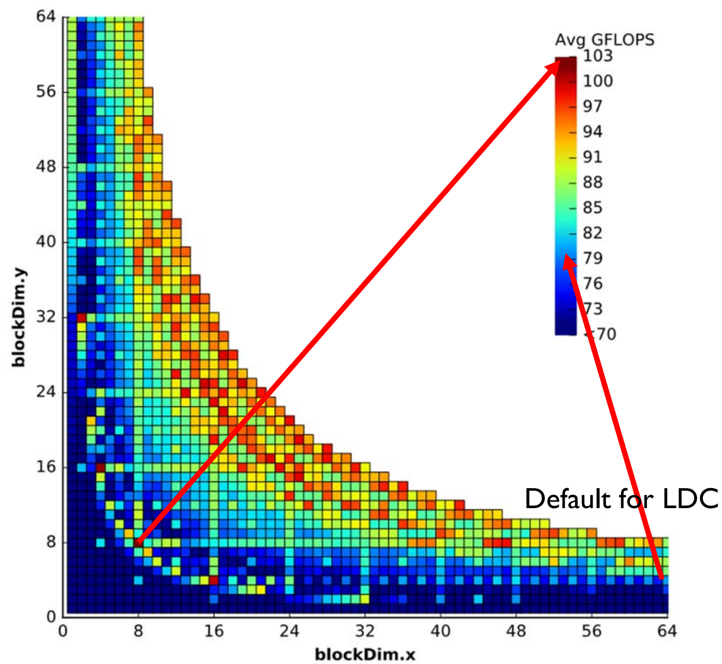


Automated GPU Blocksize Tuning via Iterative ML

- ***Challenge: Huge search space***
  (even when considering only **one** parameter,
  i.e., thread block size)

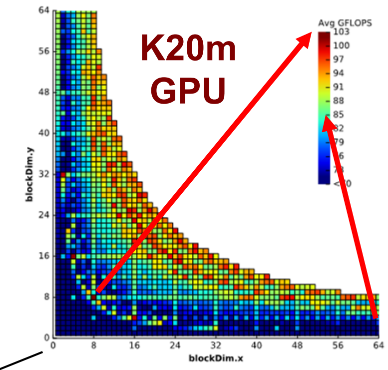
- ***What should I set my thread block size to?***
  - Brute-force search (7262 runs)
    - Takes more than a day to search
  - Reliance on developer experience?
    - Recommended block size
      - 8x8, 8x16, 16x8, 16x16
        (see next slide)

1 <= {blockDim.x , blockDim.y} <= 1024
1 <= blockDim.x * blockDim.y <= 1024

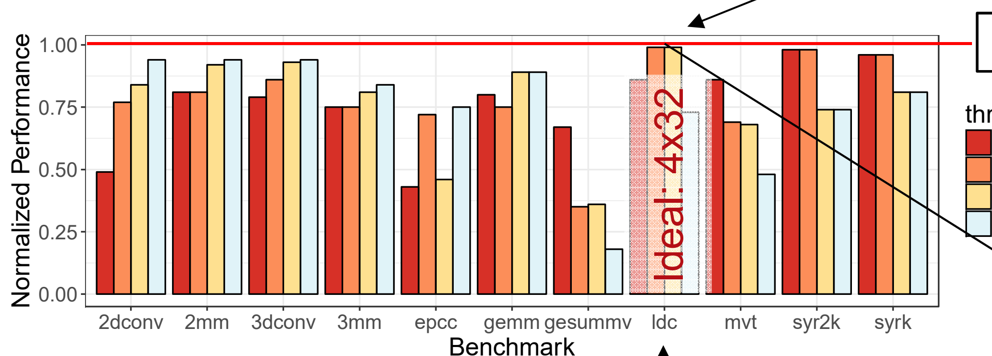


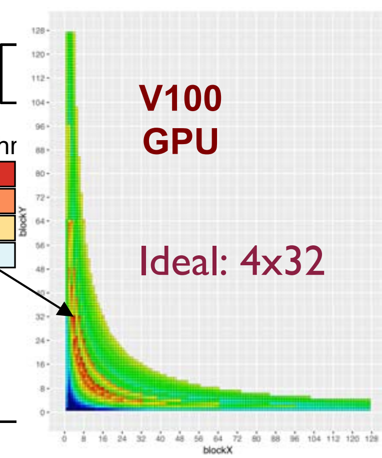
Automated GPU Blocksize Tuning via Iterative ML

- Parameter tuning the GPU
  - Reliance on end-user experience (or intuition)
    - Typically chosen block sizes? 64, 128, and 256
  - Best parameter setup varies …
    - Between applications
    - Between different devices



Ideal: 8x8

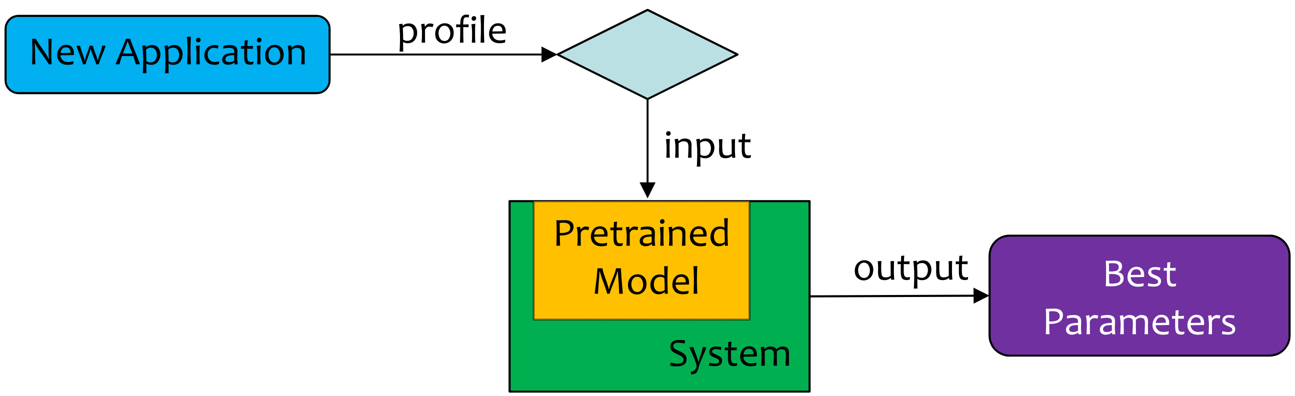




LDC code from upper right
and from previous slide

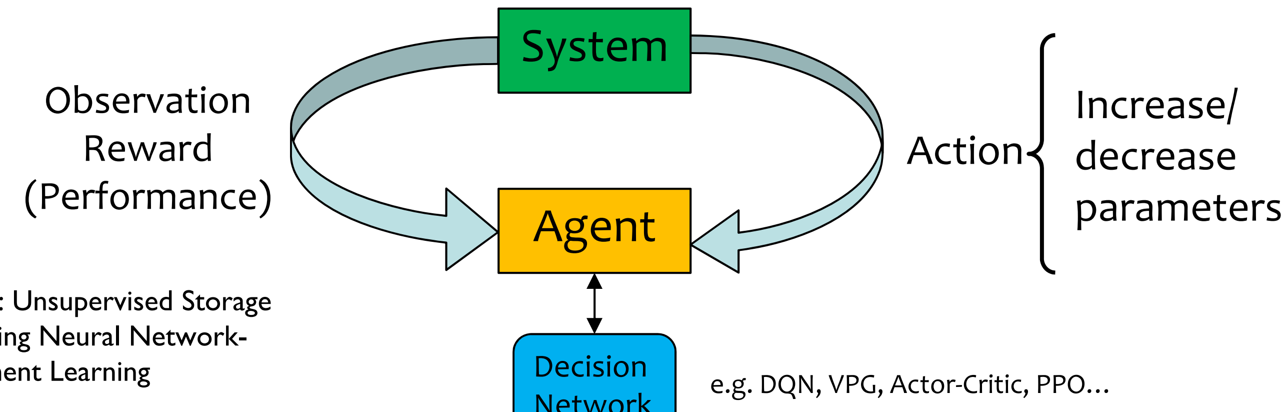
Automated GPU Blocksize Tuning via Iterative ML

- Parameter tuning the GPU
  - Reliance on end-user experience (or intuition)
  - Statistical methods
    - Build a model *a priori* based on a (required) ***large training set***
    - Predict the best parameter(s) based on real-time profiling data and model
    - May perform poorly for ***new*** algorithms on ***new*** devices or systems



Automated GPU Blocksize Tuning via Iterative ML

- Parameter tuning the GPU
  - Reliance on end-user experience (or intuition)
  - Statistical methods
  - Reinforcement learning methods
    - Requires ***no prior knowledge*** of the target system
    - Run continuously to adapt and dynamically update parameters
    - May take the decision system significant time to converge



Example: SC17 : CAPES: Unsupervised Storage Performance Tuning Using Neural Network-Based Deep Reinforcement Learning

Automated GPU Blocksize Tuning via Iterative ML

- Our Challenge: How to deploy

  … new applications

  … new algorithms

  … new systems

  … new hardware accelerators (e.g., GPUs, FPGAs, etc.)

  and tune the multi-dimensional search space of hardware/software/algorithmic parameters to optimize applications

X. Cui and W. Feng, "Iterative Machine Learning (IterML) for Effective Parameter Pruning and Tuning in Accelerators," *16th ACM International Conference on Computing Frontiers*, April-May 2019.

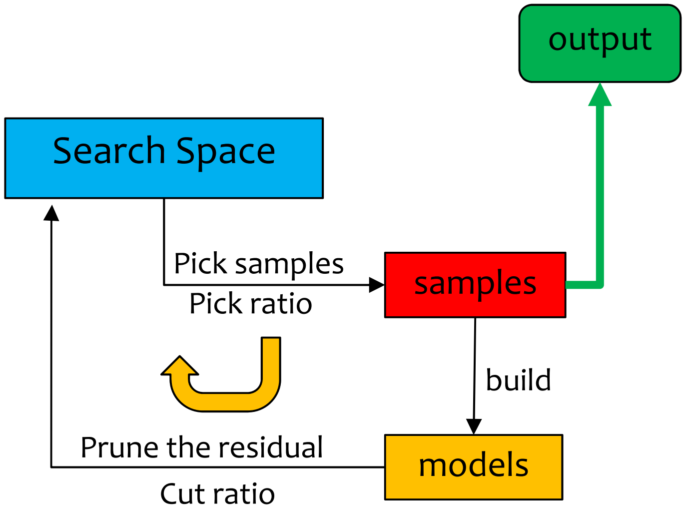
- Our Goal
  - Intelligently tune parameters with *no prior knowledge* and *no pre-trained model*
    - Deliver near-optimal performance with the least amount of effort and domain knowledge.

Automated GPU Blocksize Tuning via Iterative ML

- Iterative Machine Learning (IterML)
  - Uses samples from one iteration to then look for potentially *better samples* in subsequent iterations.

Pick ratio:
- sample ratio in each iteration

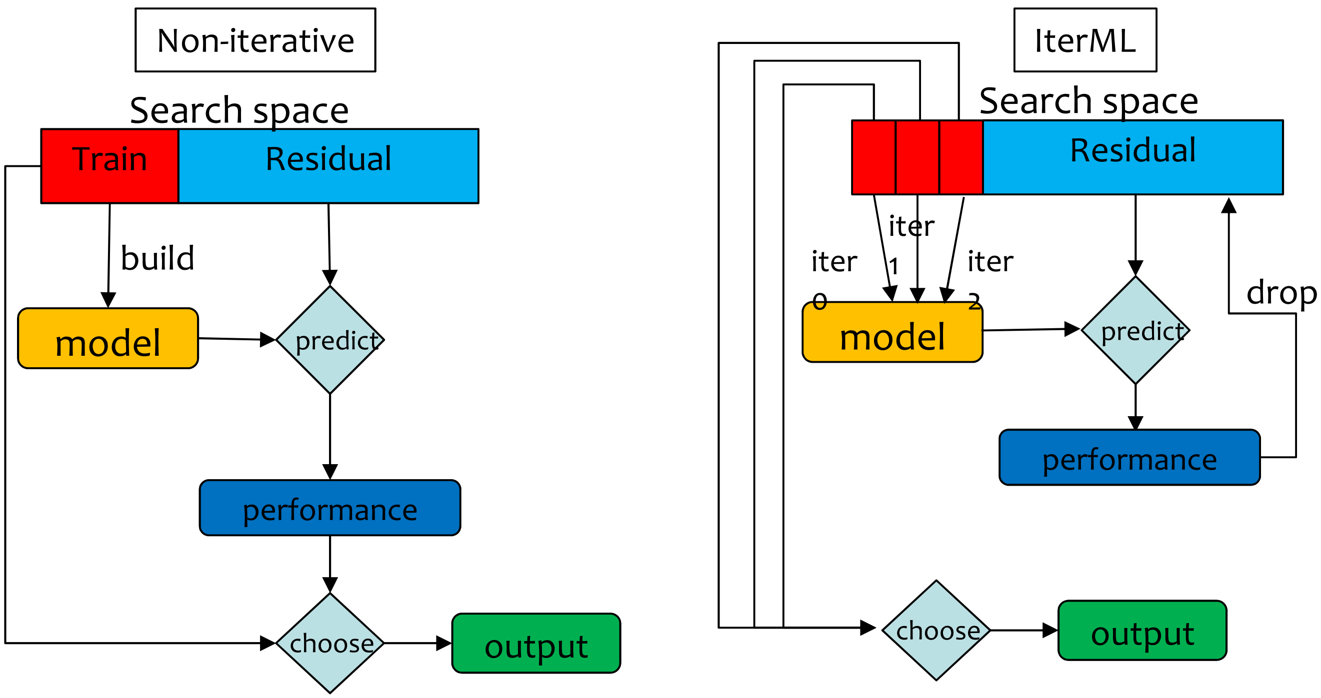
Cut ratio:
- ratio of the space pruned each iteration



X. Cui and W. Feng, "Iterative Machine Learning (IterML) for Effective Parameter Pruning and Tuning in Accelerators," *16th ACM International Conference on Computing Frontiers*, April-May 2019.

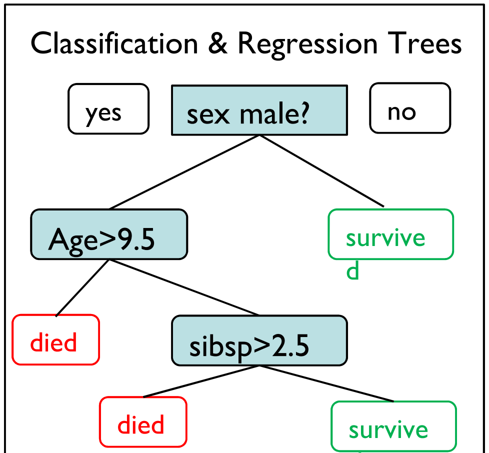
Automated GPU Blocksize Tuning via Iterative ML

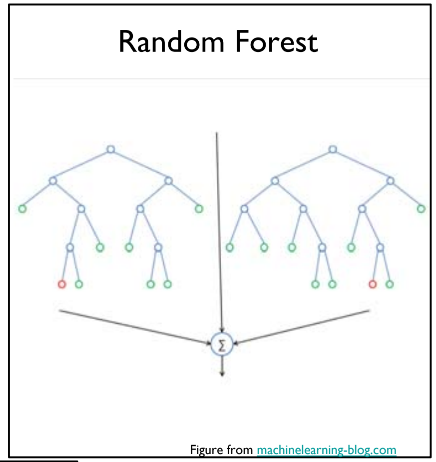
- Non-Iterative vs. Iterative Machine Learning (IterML)
  - With no prior knowledge or pretrained model

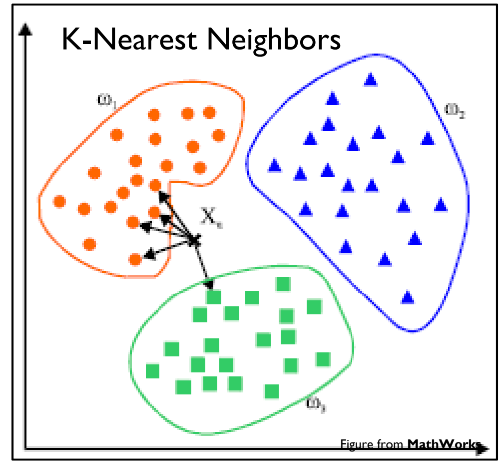


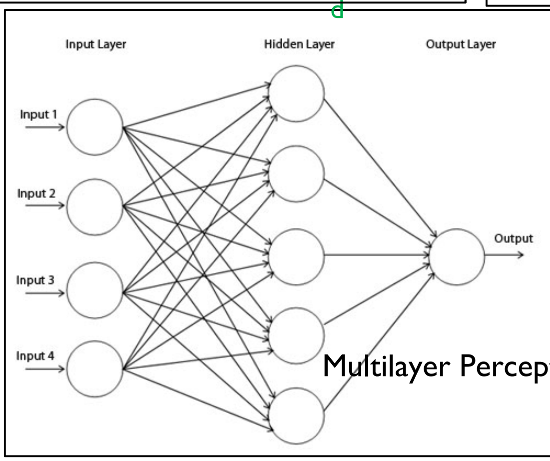
* X. Cui and W. Feng, "Iterative Machine Learning (IterML) for Effective Parameter Pruning and Tuning in Accelerators," *16th ACM Int'l Conf. on Computing Frontiers*, April-May 2019.

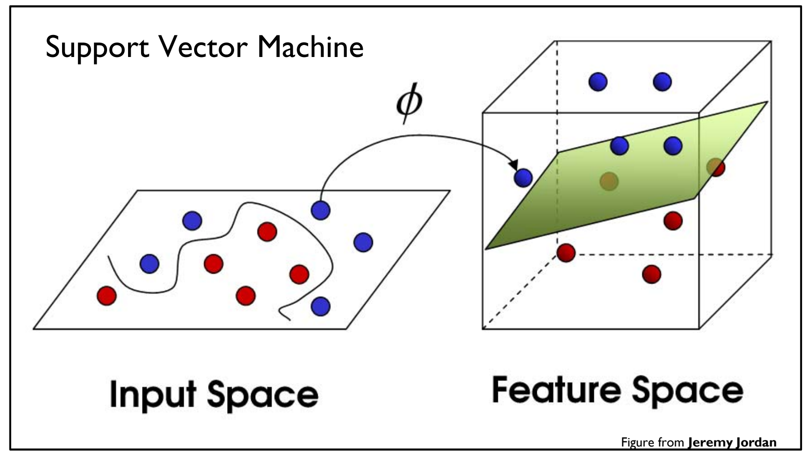
Machine Learning Models





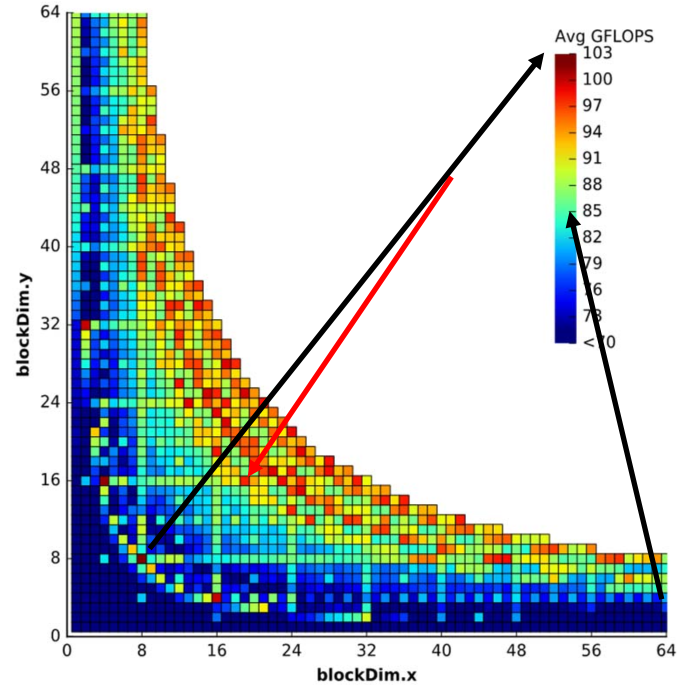






Takeaway:
Automated GPU Blocksize Tuning via Iterative ML

- Iterative machine-learning (IterML) approach
  … to prune the massive parameter search space
  - Performance evaluation of traditional non-iterative ML vs. our IterML
  - Empirical demonstration that IterML with the random forest (RF) model reduces search effort by 40%~80%
  - Random forest (RF) produces better and more stable results than other popular ML models



X. Cui and W. Feng, "Iterative Machine Learning (IterML) for Effective Parameter Pruning and Tuning in Accelerators," *16th ACM International Conference on Computing Frontiers*, April-May 2019.

Challenges: Scalable I/O in Large-Scale Deep Learning

Not all large-scale deep learning is compute-bound…

File I/O-bound

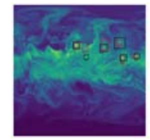
Large per batch data size & DNN with non-dense weight layers

<u>Semi-supervised Bounding Box Prediction</u>[1]

Extreme weather detection

Input size: 768 x 768 x 16

By NERSC, LBNL, DOE, Stanford, Intel



[1] Kurth et al. Deep learning at 15pf: Supervised and semi-supervised classification for scientific data. SC'17.

<u>Image Classification</u>

Lung cancer detection

Input size: 253x235x240

From Kaggle Data Science Bowl 2017

Compute-bound

High-resolution input data

Network I/O-bound

Large number of trainable parameters

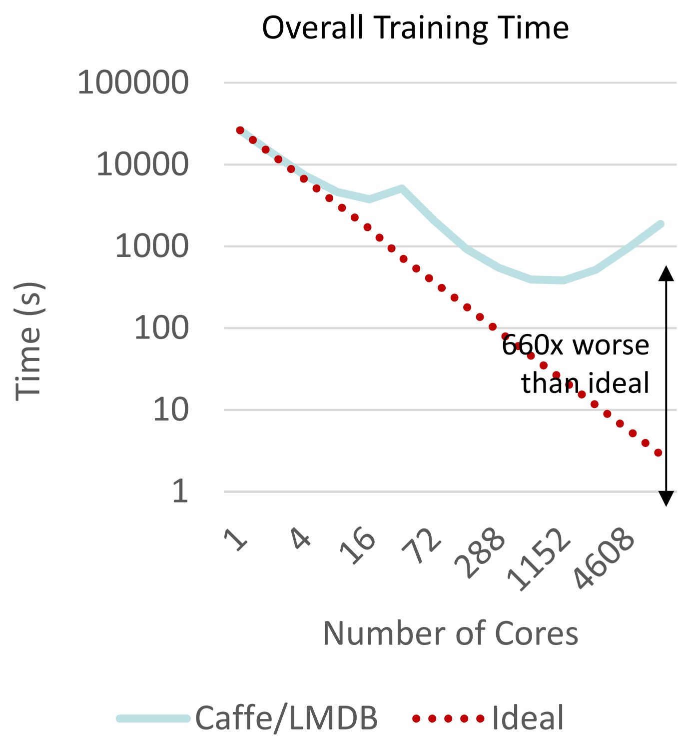
<u>Unsupervised Image Feature Extraction</u>[2]

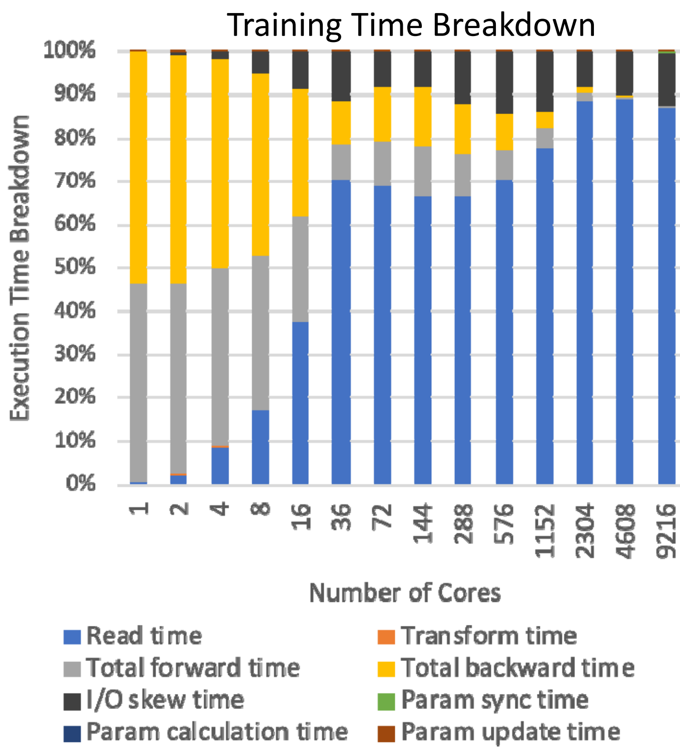
LLNL's DNN with **15 billion** parameters

[2] Ni et al. Large-scale deep learning on the YFCC100M dataset. arXiv preprint arXiv:1502.03409, 2015.

I/O Scaling of Deep Learning

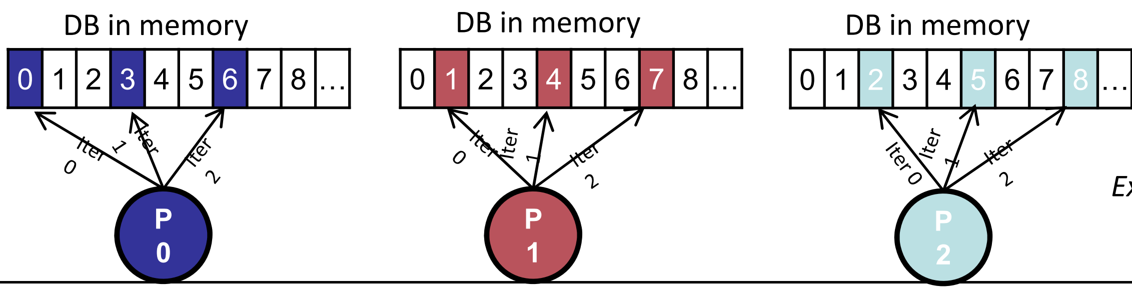
Dataset: CIFAR10-Large (50M images, 10 classes, 190 GB)
DNN: AlexNet (13 layers, 89K parameters)
Batch size: 18,432 **Training iterations:** 512
Framework: Caffe **Testbed/Storage:** LCRC Bebop/GPFS
(Each node: 36 cores Intel Broadwell, 128 GB memory)





Scalable I/O in Deep Learning: Parallel Data Reading

- Lightning Memory-Mapped Database (LMDB)
  - Widely used in deep-learning frameworks, e.g., Caffe (default), Caffe2, TensorFlow, Keras-TensorFlow
  - Uses `mmap` internally (memory-mapped file I/O)
  - Database layout: B+ tree
- No collaboration between readers
  - Each reader opens the LMDB database in its virtual memory space
  - In each iteration, each reader reads B/NP samples of data via LMDB's API in a strided manner (B = batch size, NP = number of processes)



Example of 3 processes performing strided data access

Scalable I/O in Deep Learning

Our Solution: LMDB-IO (Lightning Memory-Mapped Database – I/O)

An optimized I/O subsystem for large-scale deep learning

LMDB-IO optimizations are divided into three classes

S. Pumma, M. Si, W. Feng, and P. Balaji, "Scalable Deep Learning via I/O Analysis and Optimization," *ACM Transactions on Parallel Computing (TOPC)*, 6 (2): 6:1--6:34, July 2019.

Intra-node I/O

Problem

High inter-process contention in multi-reader environment due to **mmap**

Solution

- Localizing mmap (using one reader per node)
- Using MPI-3 shared buffer to share data between processes

Speculative Distributed I/O (Inter-node)

Problem

High I/O skew due to indeterministic DB layout (direct DB access is not allowed)

Solution

- Collaborating between reader processes to reduce I/O skew
- Speculatively reading reading data in parallel

Direct I/O

Problem

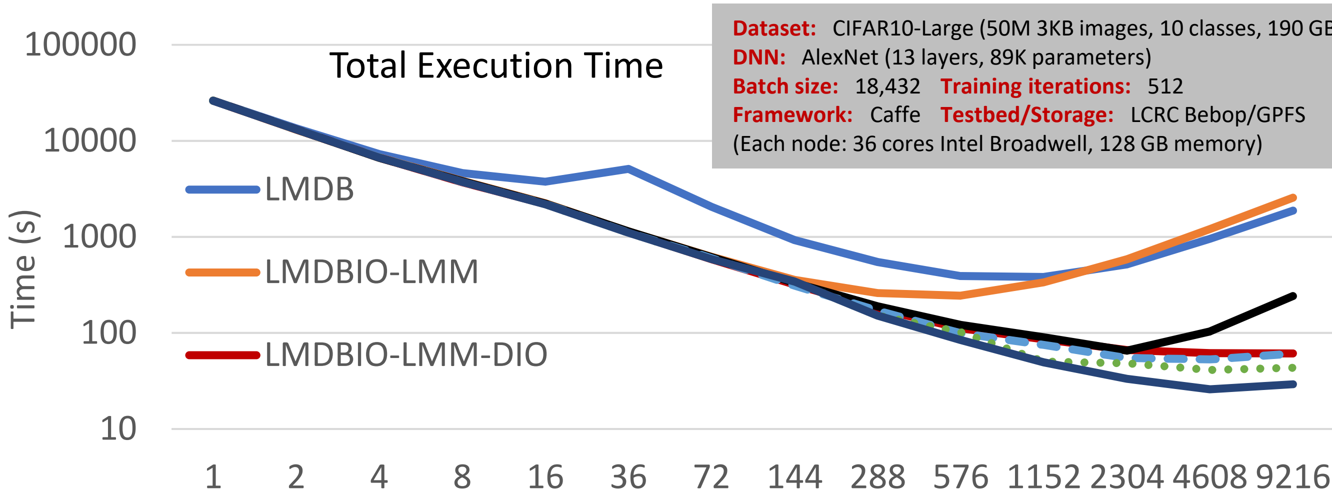
mmap is highly inefficient**.** The user has no control over I/O operations

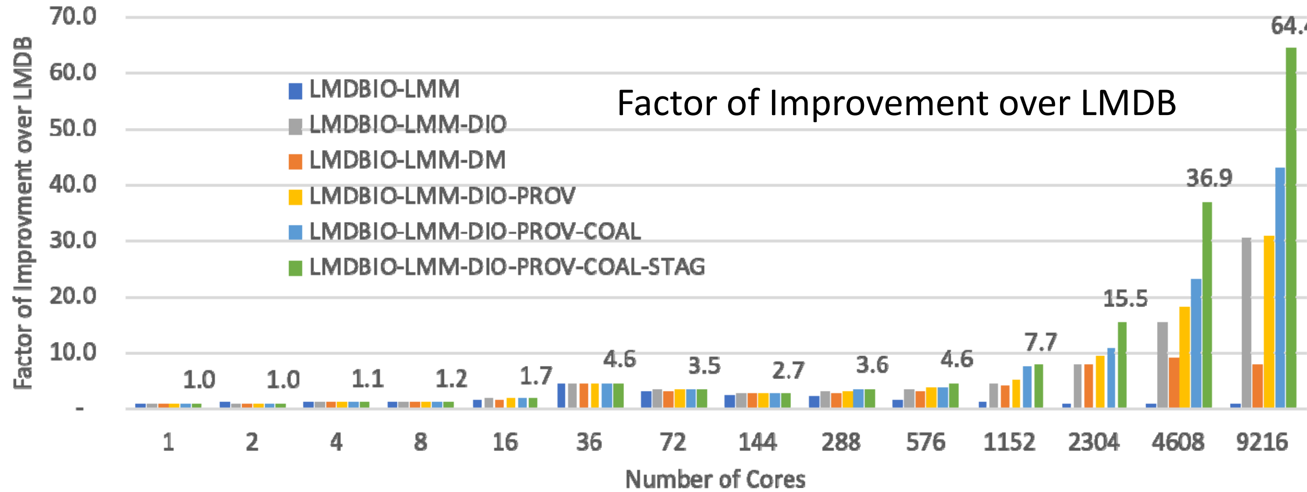
Solution

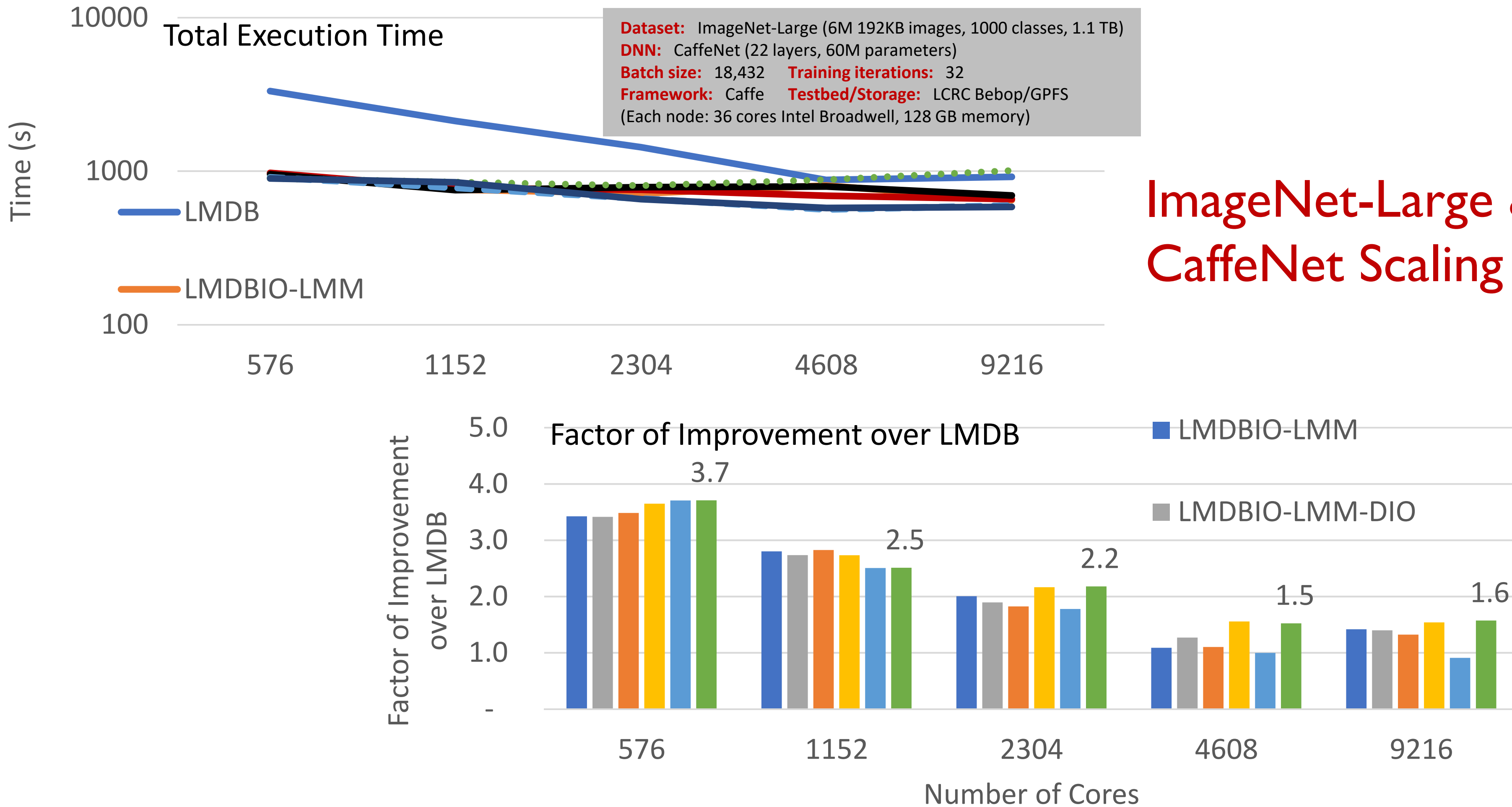
- Replacing mmap with Posix I/O (direct I/O)
- Using several techniques to improve direct I/O performance

CIFAR10-Large & AlexNet Scaling

Dataset: CIFAR10-Large (50M 3KB images, 10 classes, 190 GB)
DNN: AlexNet (13 layers, 89K parameters)
Batch size: 18,432 **Training iterations:** 512
Framework: Caffe **Testbed/Storage:** LCRC Bebop/GPFS
(Each node: 36 cores Intel Broadwell, 128 GB memory)







ImageNet-Large & CaffeNet Scaling

Conclusion

- Synergistic co-design of algorithms, software, and hardware can massively accelerate discovery.

What's Next?

- Case Studies on Synergistic Co-Design of Algorithms, Software, and Hardware
  - Brain Tomography on GPU. Carcinogenesis: Weighted Set Cover vs. Graph Cluster. […]
- HPC Systems
  - IterML: Iterative Machine Learning (AFOSR & DOD)
    - Context: Computational fluid dynamics (CFD) → OpenDwarfs, i.e., fundamental "DNA" building blocks for scientific computing
  - CoreTSAR: Core Task-Size Adapting Runtime System (DOE & NSF)
    - Context: Initially, discrete CPU+GPU systems w/ discrete memory
Now, also "fused" co-located CPU+GPU systems w/ shared memory
See Aurora @ ANL with PVC & El Capitan @ LLNL with MI-300a
  - Scalable Deep Learning (with ANL → Meta & Llama-3)
    - Context: Caffe, Caffe2, and Tensorflow
    - Takeaway: Large-scale multi-node DL does *NOT* scale.

An (Intra-Node) Ecosystem for Heterogeneous Parallel Computing