INTEL® EXPERIENCE DAY

GRADIENT BOOSTING ACCELERATION FOR INTEL ARCHITECTURE

Egor Smirnov, Software Engineering Manager

29 October 2019

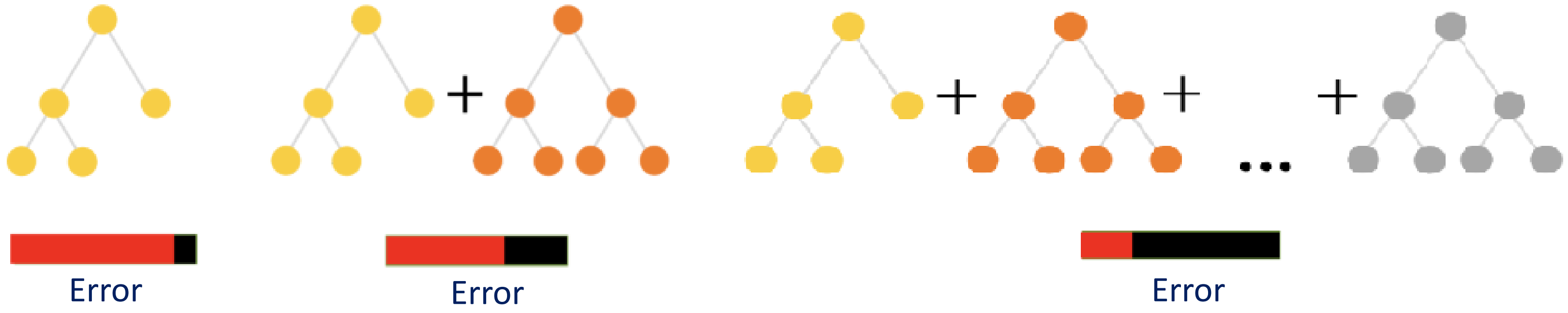
Agenda

1. Gradient Boosting – overview
2. XGBoost acceleration results
3. Intel® DAAL
4. Performance gain sources

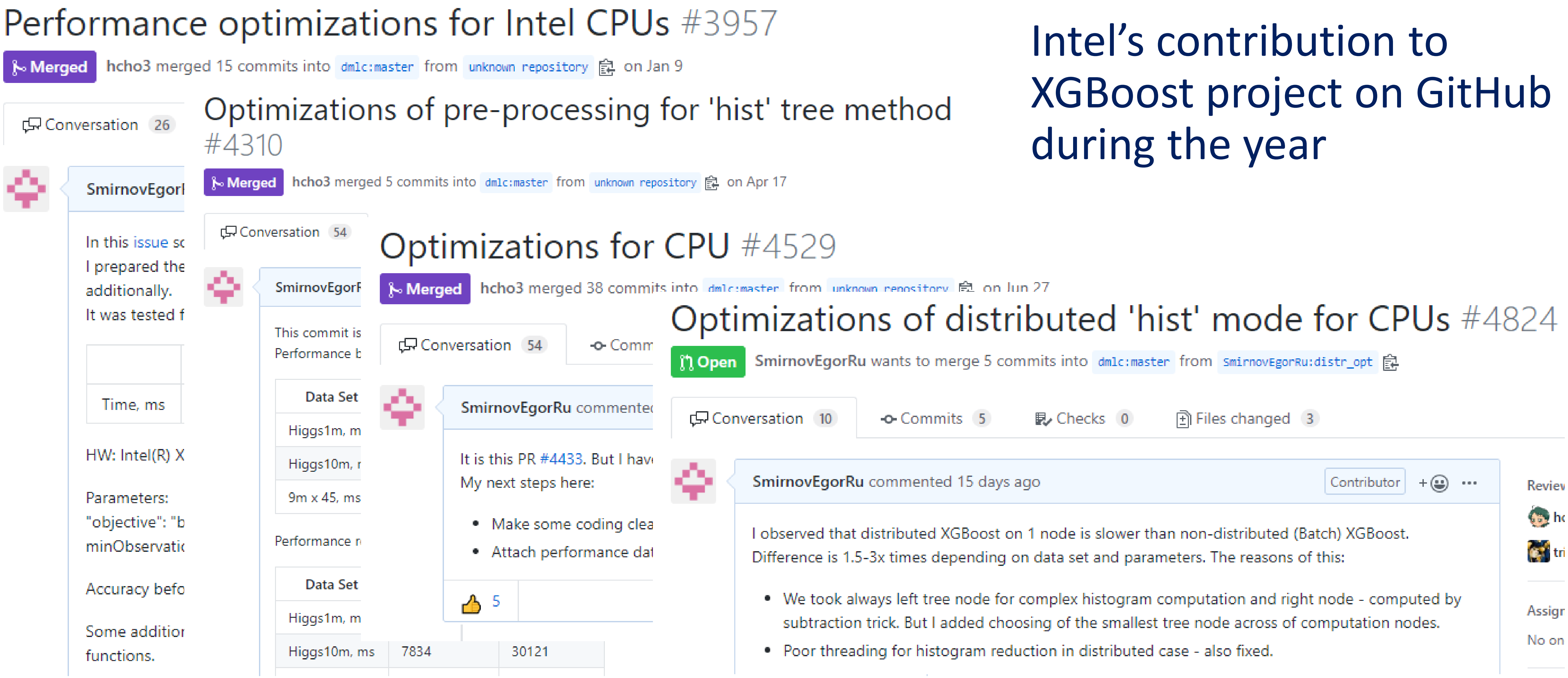
Gradient Boosting - overview

Gradient Boosting:

- Boosting algorithm (Decision Trees - base learners)
- Solve many types of ML problems (classification, regression, learning to rank)
- Highly-accurate, widely used by Data Scientists
- Compute intensive workload
- Known implementations: XGBoost, LightGBM, CatBoost, Intel® DAAL, …

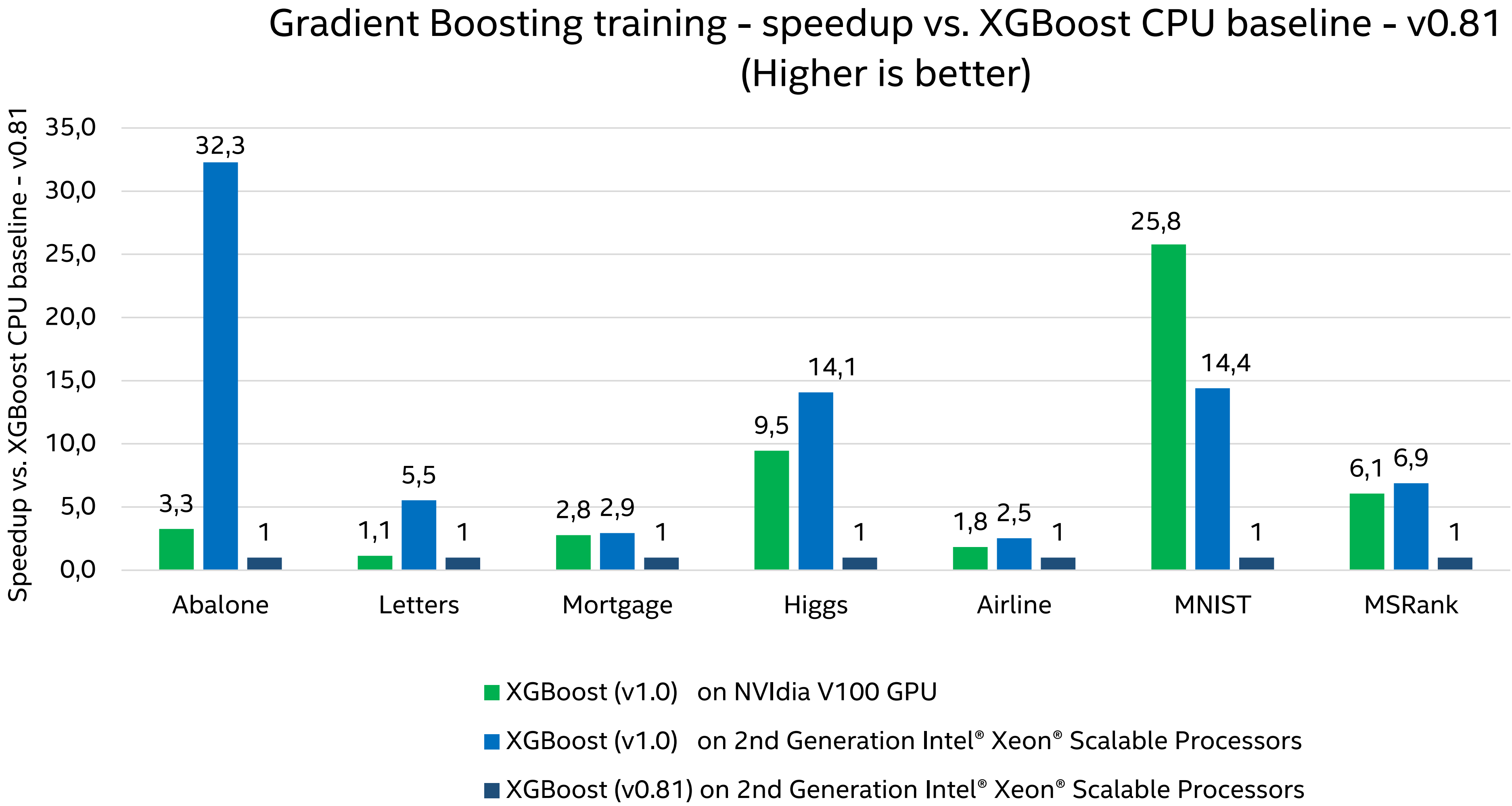


DMLC XGBoost acceleration



Intel's contribution to XGBoost project on GitHub during the year

Results with Intel® Xeon® Cascade Lake



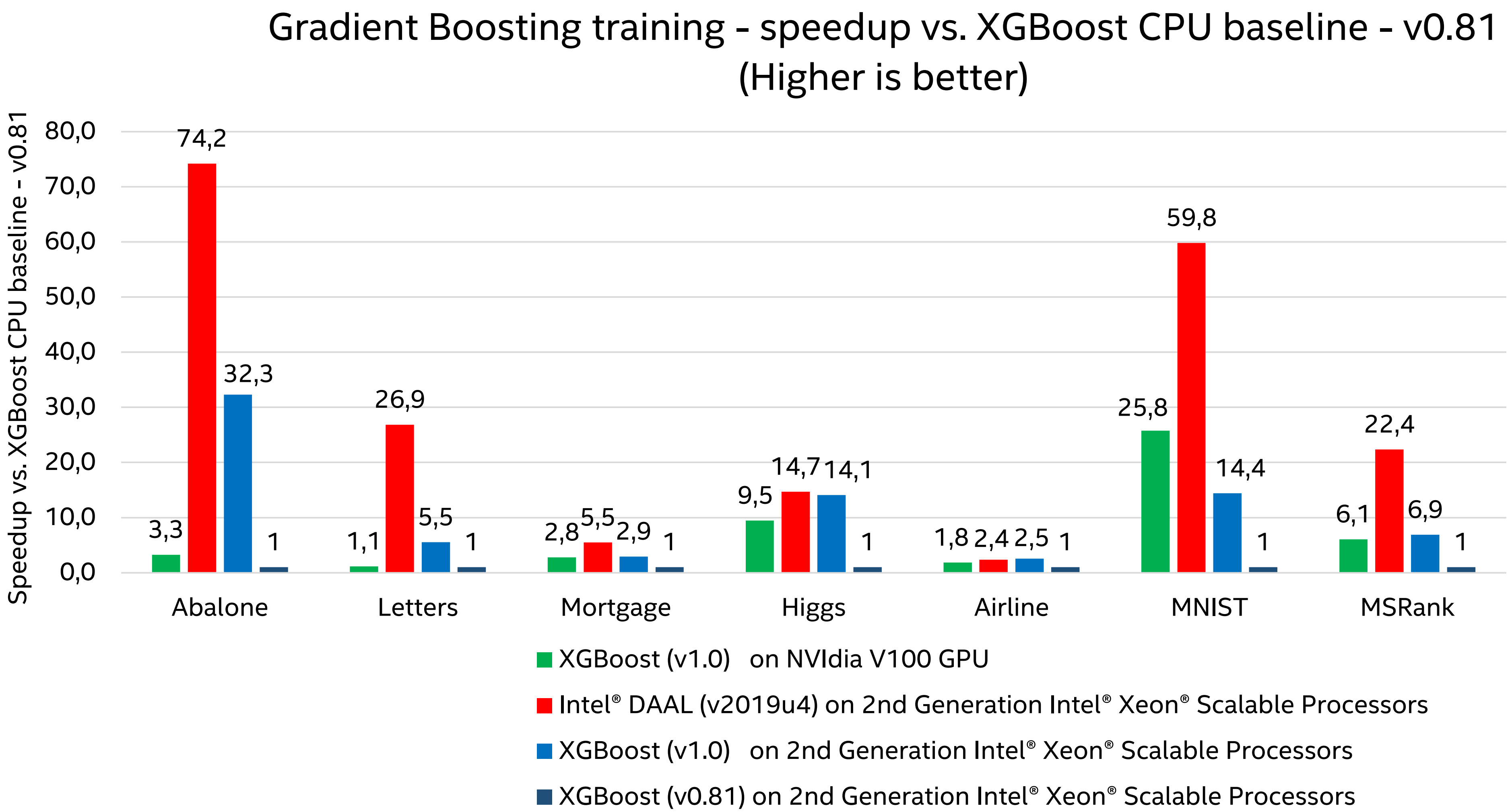
11.2X FASTER[1]

XGBOOST V1.0 VS V0.81 PROCESSORS IN AVERAGE ON 2ND GENERATION INTEL® XEON® SCALABLE

2.9X FASTER[1]

2ND GENERATION INTEL® XEON® SCALABLE PROCESSORS AGAINST NVIDIA V100 IN AVERAGE

Results with Intel® Xeon® Cascade Lake



29.4X FASTER[1]

INTEL® DAAL VS V0.81 PROCESSORS IN AVERAGE ON 2ND GENERATION INTEL® XEON® SCALABLE

8.1X FASTER[1]

2ND GENERATION INTEL® XEON® SCALABLE PROCESSORS AGAINST NVIDIA V100 IN AVERAGE

Distributed performance on Apache Spark*

Version	Run time (seconds)
DMLC XGBoost (0.81)	2089.6
DMLC XGBoost (master)	547.8

3.8x faster[2]

Processors: 4 x Intel(R) Xeon(R) Platinum 8180 @ 2.50GHz

Data set: Mortgage

Gradient Boosting Acceleration – gain sources

Pseudocode for XGBoost (0.81) implementation

```
def ComputeHist(node):
  hist = []
  for i in samples:
    for f in features:
      bin = bin_matrix[i][f]
      hist[bin].g += g[i]
      hist[bin].h += h[i]
  return hist

def BuildLvl:
  for node in nodes:
    ComputeHist(node)

  for node in nodes:
    for f in features:
      FindBestSplit(node, f)

  for node in nodes:
    SamplePartition(node)
```

Pseudocode for Intel® DAAL implementation

```
def ComputeHist(node):
  hist = []
  for i in samples:
    prefetch(bin_matrix[i + 10])
    for f in features:
      bin = bin_matrix[i][f]
      bin_value = load(hist[2*bin])
      bin_value = add(bin_value, gh[i])
      store(hist[2*bin], bin_value)
  return hist


def BuildLvl:
  parallel_for node in nodes:
    ComputeHist(node)

  parallel_for node in nodes:
    for f in features:
      FindBestSplit(node, f)

  parallel_for node in nodes:
    SamplePartition(node)
```

- Memory prefetching to mitigate irregular memory access
- Usage uint8 instead of uint32
- SIMD instructions instead of scalar code
- Nested parallelism
- Advanced parallelism, reducing seq loops
- Usage of AVX-512, vcompress instruction (from Skylake)

Training stage

Legend: Moved from Intel® DAAL to XGBoost (v1.0) | Already available in Intel® DAAL, potential optimizations for XGBoost

Histogram building – code sample

XGBoost (master):

```
for (size_t i = istart; i < iend; ++i) {
  const size_t icol_start = rid[i] * n_features;
  const size_t icol_start_prefetch = rid[i+prefetch_offset] * n_features;
  const size_t idx_gh = 2*rid[i];

  PREFETCH_READ_T0(pgh + 2*rid[i + prefetch_offset]);

  for (size_t j = icol_start_prefetch; j < icol_start_prefetch + n_features;
      j += prefetch_step) {
    PREFETCH_READ_T0(index + j);
  }

  grad_stat.sum_grad += pgh[idx_gh];
  grad_stat.sum_hess += pgh[idx_gh+1];

  for (size_t j = icol_start; j < icol_start + n_features; ++j) {
    const uint32_t idx_bin = 2*index[j];
    data_local_hist[idx_bin] += pgh[idx_gh];
    data_local_hist[idx_bin+1] += pgh[idx_gh+1];
  }
}
```

XGBoost (0.81):

```
for (bst_omp_uint i = 0; i < nrows - rest; i += kUnroll) {
  const bst_omp_uint tid = omp_get_thread_num();
  const size_t off = tid * nbins_;
  size_t rid[kUnroll];
  size_t ibegin[kUnroll];
  size_t iend[kUnroll];
  GradientPair stat[kUnroll];
  for (int k = 0; k < kUnroll; ++k) {
    rid[k] = row_indices.begin[i + k];
  }
  for (int k = 0; k < kUnroll; ++k) {
    ibegin[k] = gmat.row_ptr[rid[k]];
    iend[k] = gmat.row_ptr[rid[k] + 1];
  }
  for (int k = 0; k < kUnroll; ++k) {
    stat[k] = gpair[rid[k]];
  }
  for (int k = 0; k < kUnroll; ++k) {
    for (size_t j = ibegin[k]; j < iend[k]; ++j) {
      const uint32_t bin = gmat.index[j];
      data_[off + bin].Add(stat[k]);
    }
  }
}
```

Naive implementation:

```
for(int i = 0; i < n; i++) {
    for(int j = 0; j < p; j++) {
        int bin = bins[row_idx[i]*p + j];
        hist[bin].g += g[i];
        hist[bin].h += h[i];
    }
}
```

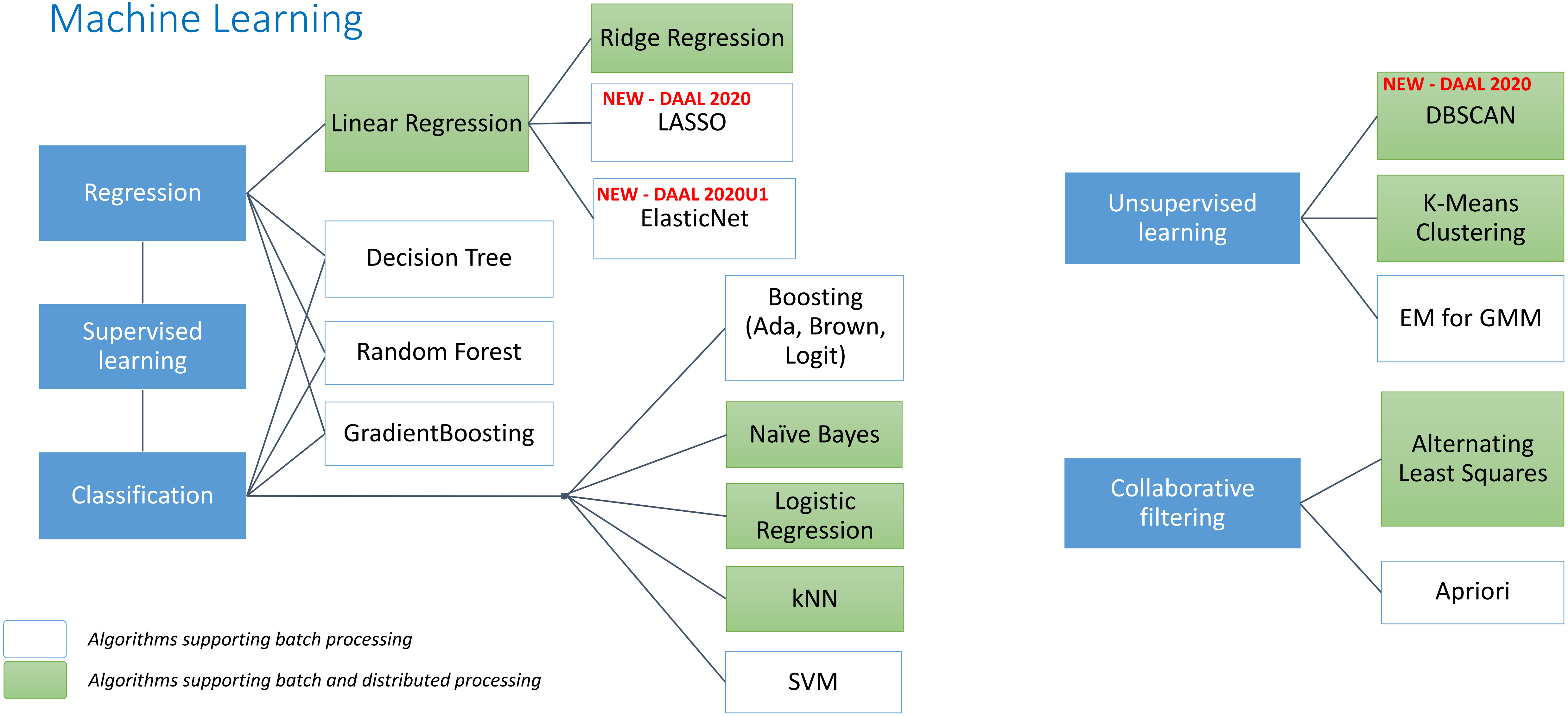
VTune profiling:

	Clockticks	Instructions Retired	CPI Rate	Retiring	Back-End Bound: Memory Bound	Back-End Bound: Core Bound
Before (0.81):	250,382,000,000	185,235,600,000	1.352	36.1%	44.9%	11.3%
	70,985,200,000	18,686,800,000	3.799	11.9%	72.7%	8.1%
After (master):	9,156,400,000	11,451,000,000	0.800	55.9%	28.9%	10.9%

BACKUP

Intel® DAAL Algorithms

Machine Learning



Intel® DAAL Algorithms

Data Transformation and Analysis

- Basic statistics for datasets
  - Low order moments
  - Quantiles
  - Order statistics
- Correlation and dependence
  - Cosine distance
  - Correlation distance
  - Variance-Covariance matrix
- Matrix factorizations
  - SVD
  - QR
  - Cholesky
  - tSVD
- Dimensionality reduction
  - PCA
- Association rule mining (Apriori)
- Optimization solvers (SGD, AdaGrad, lBFGS)
- Outlier detection
  - Univariate
  - Multivariate
- Math functions (exp, log,...)

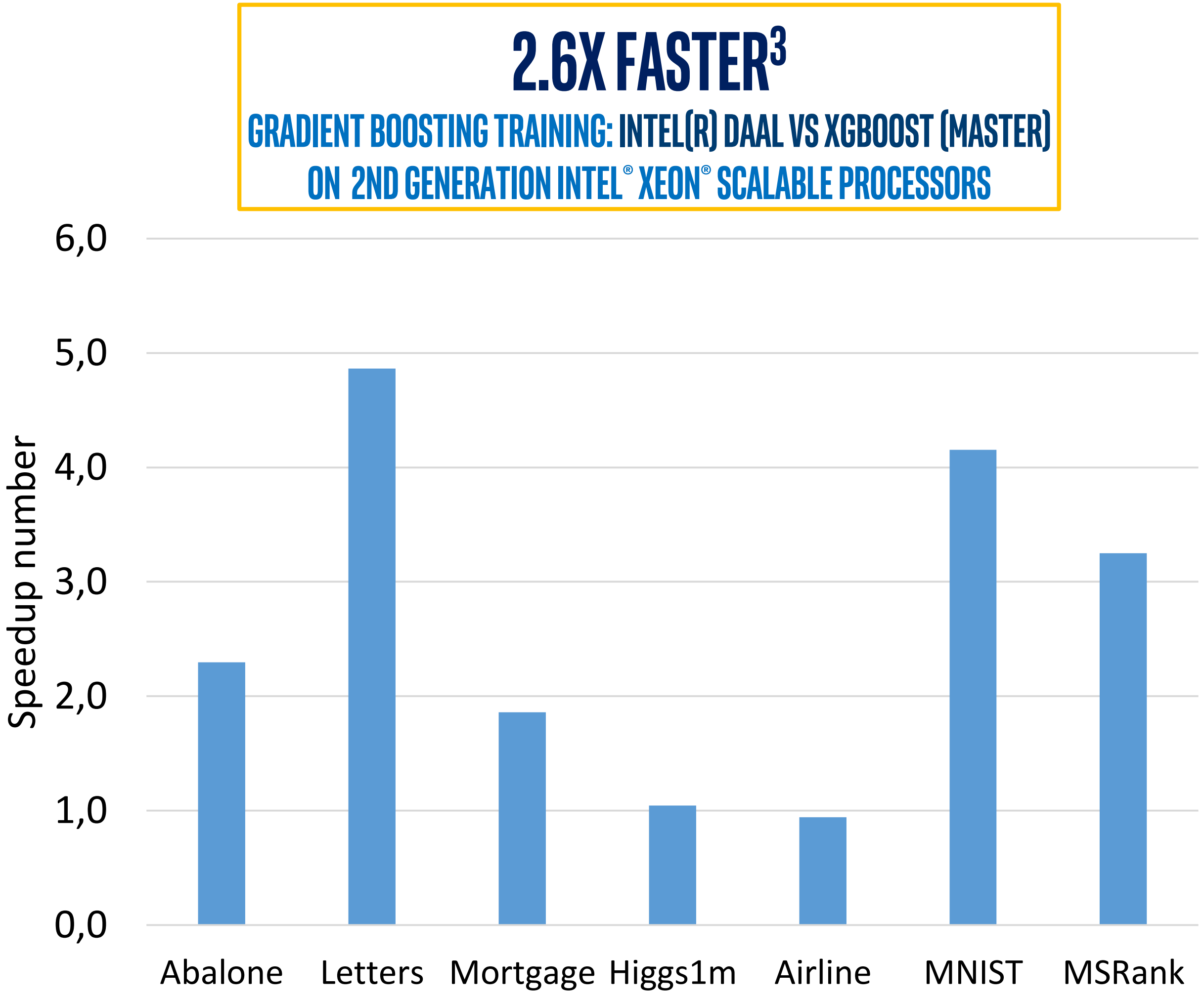
Algorithms supporting batch processing

Algorithms supporting batch, online and/or distributed processing

Get maximum performance with Intel® DAAL

Wide range of Data Analytics and Classical ML algorithms:

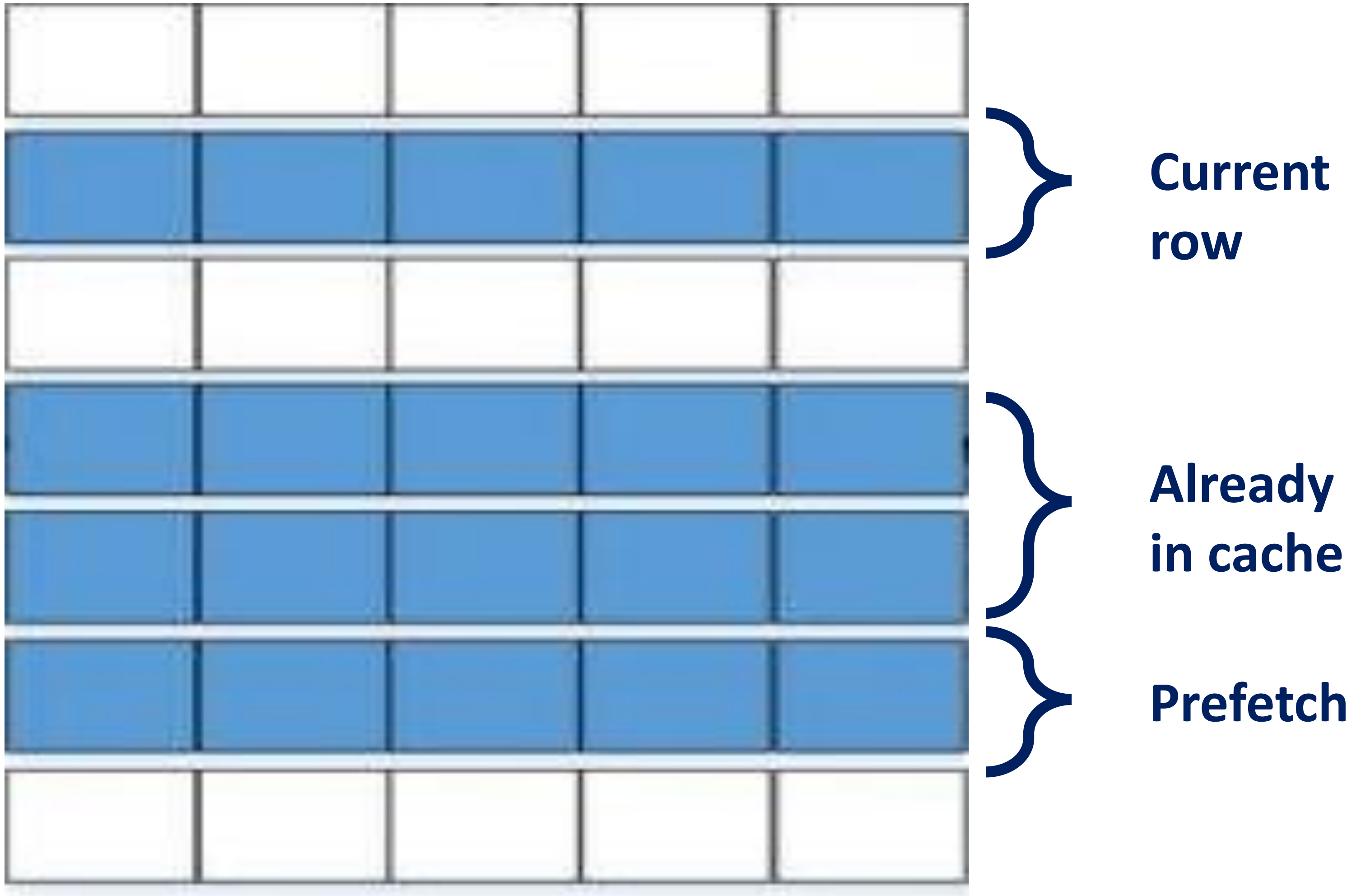
- C++, Python and Java APIs
- Batch, streaming and multi-node
- Can be used with many distributed systems (Spark*, MPI*)
- Open Sourced
- Underneath of Intel® Optimized version of Scikit-learn
- Windows*, Linux*, FBSD*, OS X*



Advanced Memory Prefetching

- Main hotspot – histogram building.
- Performance issue: irregular memory access

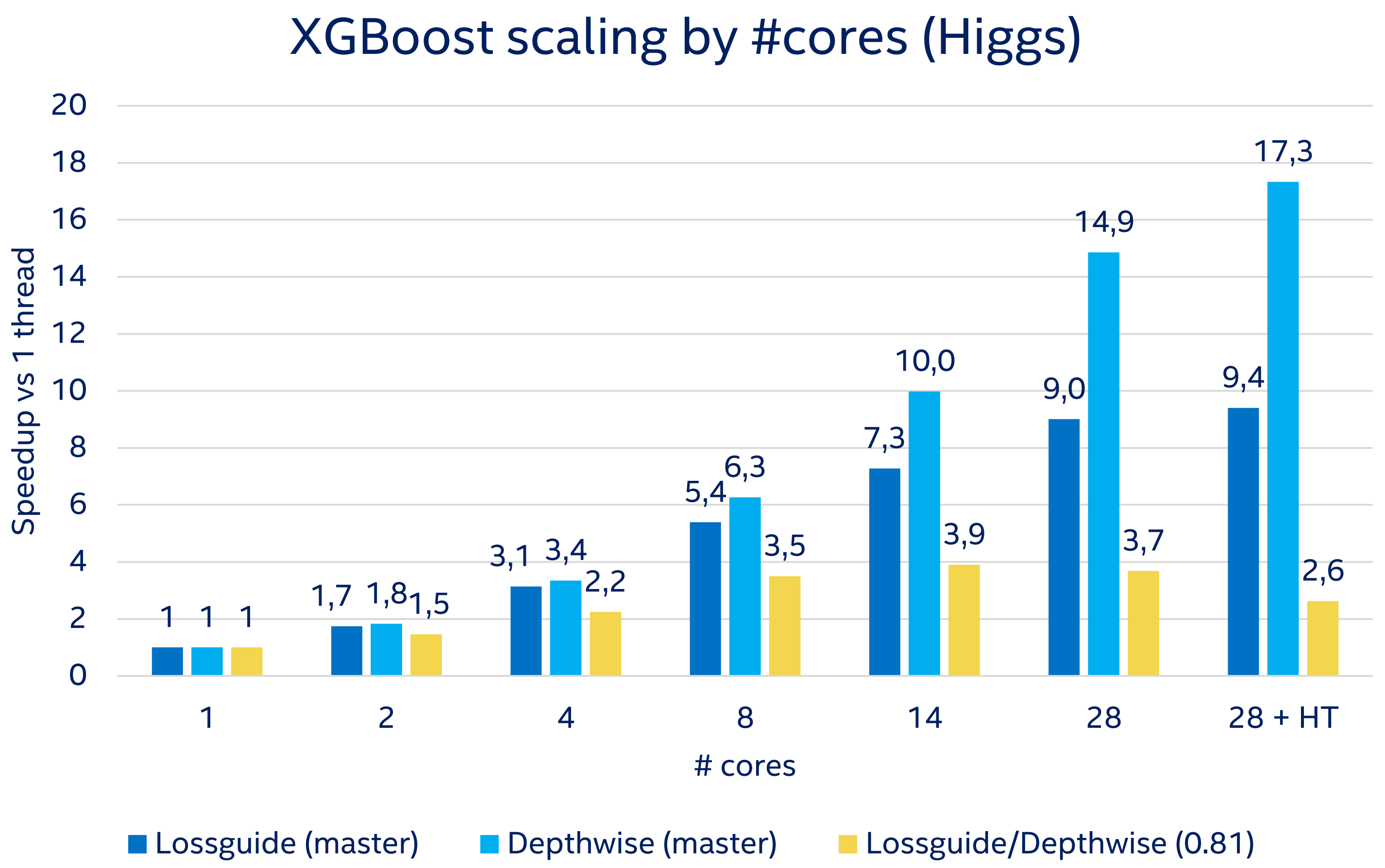
```
for(int i = 0; i < n; i++) {
   prefetch_row(bins[row_idx[i+10]*p);
   for int j = 0; j < p; j++) {
      int bin = bins[row_idx[i]*p + j];
      hist[bin].g += g[i];
      hist[bin].h += h[i];
   }
}
```



Multi-core acceleration

Technics used for optimizations:

- Nested parallelism by nodes
- Removing threading overheads
- Smart blocking



1,3. Hardware config for Single node

	Config1 (CPU)	Config2 (GPU)
Test by	Intel	Intel
Test date	08/22/2019	08/22/2019
AWS Instance	c5.metal	p3.2xlarge
Platform	Amazon EC2	Amazon EC2
# Nodes	1	1
# Sockets	2	1
CPU	2nd Generation Intel® Xeon® Scalable Processors (8275CL @ 3.00GHz)	Xeon E5-2686 v4 @ 2.30GHz
Cores/socket, Threads/socket	24/48	4/8
ucode	0x5000017	0xb000037
HT	On	On
Turbo	On	On
BIOS version (including microcode verison: cat /proc/cpuinfo | grep microcode –m1)	1.0, vendor: Amazon EC2	4.2, Amazon EC2
System DDR Mem Config: slots / cap / run-speed	12 slots / 16GB / 2933 MHz	4 / 13312 MB / Unknown RAM
System DCPMM Config: slots / cap / run-speed	-	
Total Memory/Node (DDR+DCPMM)	193 GB	61 GB
NIC	Amazon.com, Inc. Elastic Network Adapter (ENA)	Amazon.com, Inc. Elastic Network Adapter (ENA)
PCH	Intel C620	Unknown
Other HW (Accelerator)	-	Tesla V100-SXM2-16GB,
OS	Ubuntu 18.04.2 LTS	Ubuntu 18.04.2 LTS
Kernel	4.15.0-1045-aws	4.15.0-1045-aws
Mitigation variants (1,2,3,3a,4, L1TF) https://github.com/speed47/spectre-meltdown-checker	Mitigated	Mitigated

1,3. Software Config for Single node

Configuration	XGB 1.0 CPU	XGB 0.81 CPU	Intel DAAL 2019U4	XGB 1.0 GPU
Workload & version	Gradient Boosting training	Gradient Boosting training	Gradient Boosting training	Gradient Boosting training
HW configuration	Config1	Config1	Config1	Config2
Compiler	GCC 7.4	Unknown, downloaded from pip	Unknown, downloaded from conda	GCC 7.4, nvcc 9.1
Tested Libraries (incl version)	XGBoost master (1.0) (ef9af33a000f09dbc5c6b09aee133e38a6d2e1ff)	XGBoost 0.81	DAAL 2019u4	XGBoost master (1.0) (ef9af33a000f09dbc5c6b09aee133e38a6d2e1ff)
Other python libs used in benchmarks	Numpy 1.16.4, Pandas 0.25, Scikit-lean 0.21.2	Numpy 1.16.4, Pandas 0.25, Scikit-lean 0.21.2	Numpy 1.16.4, Pandas 0.25, Scikit-lean 0.21.2	Numpy 1.16.4, Pandas 0.25, Scikit-lean 0.21.2
Python	3.6	3.6	3.6	3.6
Dataset	Higgs, letter, abalone, MNIST, Airline, MSRank, Mortgage	Higgs, letter, abalone, MNIST, Airline, MSRank, Mortgage	Higgs, letter, abalone, MNIST, Airline, MSRank, Mortgage	Higgs, letter, abalone, MNIST, Airline, MSRank, Mortgage
CUDA				Driver Version: 410.104, CUDA Version: 10.0

2. Spark* acceleration configs

Test by	Intel
Test date	09/06/2019
SUT Setup	
Platform	S2600WF0
# Nodes	4
# Sockets	2 per node
CPU	Intel(R) Xeon(R) Platinum 8180 CPU @ 2.50GHz
Cores/socket, Threads/socket	28/56
Microcode	0x200005e
HT	On
Turbo	On
BIOS version	SE5C620.86B.0X.02.0117.040420182310
BKC version – E.g. ww47	WW362019
AEP FW version – E.g. 5336	-
System DDR Mem Config: slots / cap / speed	10 slots / 32GB / 2666
System DCPMM Config: slots / cap /speed	-
Total Memory/Node (DDR, DCPMM)	384, 0
Storage - boot	1x 480GB Intel SSDSC2BB48
Storage - application drives	8x 1TB Intel SSDSC2KB96
NIC	2x Intel X722
PCH	Intel C620
Other HW (Accelerator)	-

Data sets

Data set	Description	Link
Abalone	Predicting the age of abalone from physical measurements	https://archive.ics.uci.edu/ml/datasets/abalone
Letters	Letter Recognition	https://archive.ics.uci.edu/ml/datasets/letter+recognition
Mortgage	Data set promoted by NVidia. It is 2000Q1 part only	https://rapidsai.github.io/demos/datasets/mortgage-data
Higgs	Recognize signals which produce Higgs bosons (physics). first 1M rows	https://archive.ics.uci.edu/ml/datasets/HIGGS
Airline	Predict delay in scheduling. After One-hot-encoding, first 1m rows	http://stat-computing.org/dataexpo/2009/the-data.html
MNIST	Recognize handwritten digits	https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
MSLR	Microsoft Learning to Rank Datasets	https://www.microsoft.com/en-us/research/project/mslr/

DISCLOSURES

Intel Technology and Manufacturing Day 2017 occurs during Intel's "Quiet Period," before Intel announces its 2017 first quarter financial and operating results. Therefore, presenters will not be addressing first quarter information during this year's program.

Statements in this presentation that refer to forecasts, future plans and expectations are forward-looking statements that involve a number of risks and uncertainties. Words such as "anticipates," "expects," "intends," "goals," "plans," "believes," "seeks," "estimates," "continues," "may," "will," "would," "should," "could," and variations of such words and similar expressions are intended to identify such forward-looking statements. Statements that refer to or are based on projections, uncertain events or assumptions also identify forward-looking statements. Such statements are based on management's expectations as of March 28, 2017, and involve many risks and uncertainties that could cause actual results to differ materially from those expressed or implied in these forward-looking statements. Important factors that could cause actual results to differ materially from the company's expectations are set forth in Intel's earnings release dated January 26, 2017, which is included as an exhibit to Intel's Form 8-K furnished to the SEC on such date. Additional information regarding these and other factors that could affect Intel's results is included in Intel's SEC filings, including the company's most recent reports on Forms 10-K, 10-Q and 8-K reports may be obtained by visiting our Investor Relations website at www.intc.com or the SEC's website at www.sec.gov.

intel®