Accelerating the XGBoost algorithm using GPU computing

Rory Mitchell and Eibe Frank
Department of Computer Science, University of Waikato, Hamilton, New Zealand

Outline

- Introduction
- BACKGROUND AND RELATED WORK
  - Tree boosting algorithms
  - Graphics processing units
  - Parallel primitives
  - Scan and reduce on multiple sequences
- PARALLEL TREE CONSTRUCTION
- EVALUATION
- CONCLUSION

BACKGROUND AND RELATED WORK

- Input：

$$(\vec{x}_0, y_0), (\vec{x}_1, y_1) \cdots (\vec{x}_n, y_n)$$

- Output：

$$F(\vec{x}) = y$$

BACKGROUND AND RELATED WORK

- Decision Tree

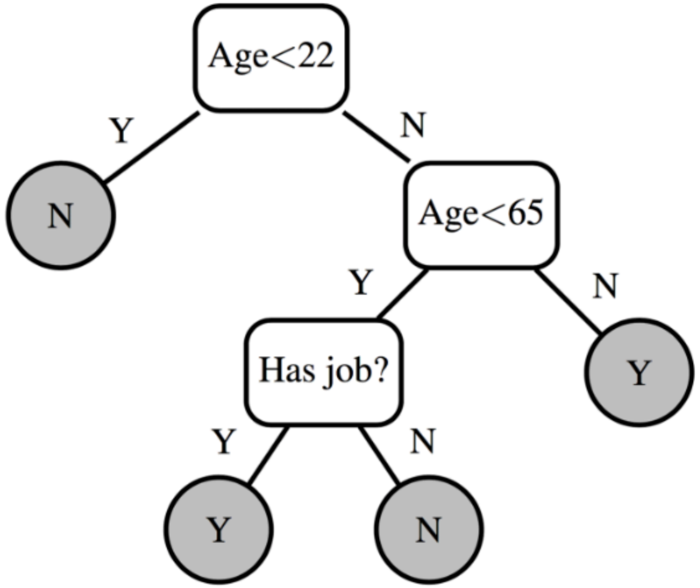


Figure 1 **Example decision tree.**

Table 1 **Example training instances.**

Instance	Age	Has job	Owns house
0	12	N	N
1	32	Y	Y
2	25	Y	Y
3	48	N	N
4	67	N	Y
5	18	Y	N

$$H(T) = -\sum_{y \in Y} P(y) \log_b P(y)$$

$$\mathrm{IG}(T, T_{\text{left}}, T_{\text{right}}) = H_T - (n_{\text{left}}/n_{\text{total}}) * H(T_{\text{left}}) - (n_{\text{right}}/n_{\text{total}}) * H(T_{\text{right}})$$

BACKGROUND AND RELATED WORK

- Gradient boosting

$$F_{m+1}(x) = F_m(x) + f(x) = y$$

$$f(x) = y - F_m(x)$$

$$L(y, F(x)) = \frac{1}{2}(y - F(x))^2 \qquad J = \sum_i L(y_i, F(x_i))$$

$$\frac{dJ}{dF(x_i)} = \frac{d\sum_i L(y_i, F(x_i))}{dF(x_i)} = \frac{dL(y_i, F(x_i))}{dF(x_i)} = F_m(x_i) - y_i$$

$$f(x) = y - F_m(x) = -\frac{dL(y, F(x))}{dF(x)}$$

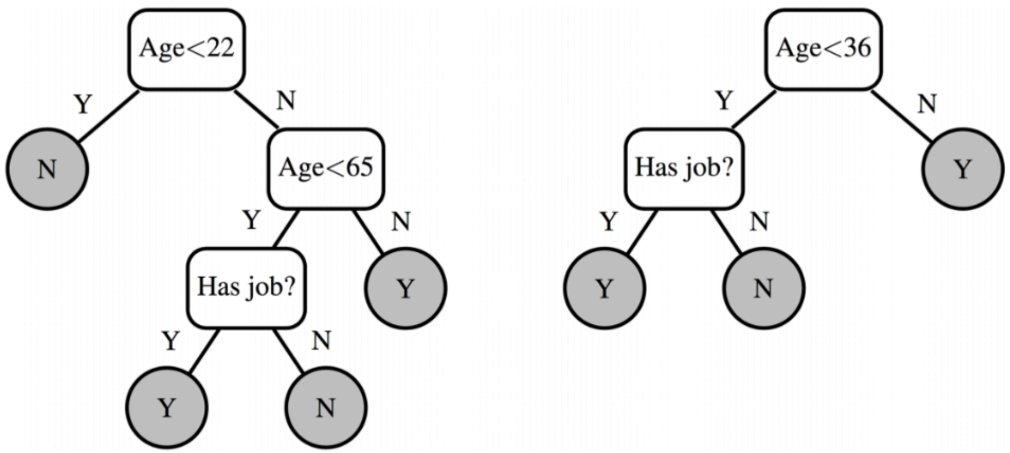


Figure 2 Decision tree ensemble.

BACKGROUND AND RELATED WORK

- XGBoost

$$\text{Obj} = \sum_i L(y_i, \hat{y}_i) + \sum_k \Omega(f_k) \qquad \Omega(f_k) = \gamma T + \frac{1}{2}\lambda w^2$$

$$\text{Obj}^m = \sum_i L(y_i, \hat{y}_i^{(m-1)} + f_k(x_i)) + \sum_k \Omega(f_k)$$

$$f(x + \triangle x) \simeq f(x) + f'(x) \triangle x + \frac{1}{2} f''(x) \triangle x^2$$

$$\text{Obj}^m \simeq \sum_i [L(y_i, \hat{y}_i^{(m-1)}) + g_i f_k(x) + \frac{1}{2} h_i f_k(x)^2] + \sum_k \Omega(f_k) + \text{constant}$$

$$g_i = \frac{dL(y_i, \hat{y}_i^{(m-1)})}{d\hat{y}_i^{(m-1)}} \qquad h_i = \frac{d^2 L(y_i, \hat{y}_i^{(m-1)})}{d(\hat{y}_i^{(m-1)})^2}$$

BACKGROUND AND RELATED WORK

- XGBoost

$$\mathrm{Obj}^m = \sum_i [g_i f_k(x) + \frac{1}{2} h_i f_k(x)^2] + \sum_k \Omega(f_k)$$

$$\mathrm{Obj}^m = \sum_{j=1}^{T} \left[\left(\sum_{i \in I_j} g_i \right) w_{q(x)} + \frac{1}{2} \left(\sum_{i \in I_j} h_i \right) w_{q(x)}^2 \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w^2$$

$$G_j = \sum_{i \in I_j} g_i \quad H_j = \sum_{i \in I_j} h_i$$

$$\mathrm{Obj}^m = \sum_{j=1}^{T} \left[G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma T$$

BACKGROUND AND RELATED WORK

- XGBoost

$$w_j = -\frac{G_j}{H_j + \lambda} \qquad \mathrm{Obj}^m = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T$$

$$\mathrm{Obj}_{\mathrm{leaf}} = -\frac{1}{2}\frac{G_j^2}{H_j + \lambda} + \gamma \qquad \mathrm{Obj}_{\mathrm{split}} = -\frac{1}{2}\left(\frac{G_{j\mathrm{L}}^2}{H_{j\mathrm{L}} + \lambda} + \frac{G_{j\mathrm{R}}^2}{H_{j\mathrm{R}} + \lambda}\right) + 2\gamma$$

$$\mathrm{Gain} = \mathrm{Obj}_{\mathrm{leaf}} - \mathrm{Obj}_{\mathrm{split}} = \frac{1}{2}\left[\frac{G_{\mathrm{L}}^2}{H_{\mathrm{L}} + \lambda} + \frac{G_{\mathrm{R}}^2}{H_{\mathrm{R}} + \lambda} - \frac{(G_{\mathrm{L}} + G_{\mathrm{R}})^2}{H_{\mathrm{L}} + H_{\mathrm{R}} + \lambda}\right] - \gamma$$

BACKGROUND AND RELATED WORK

- GPU

Listing 1 **Example CUDA kernel**

```
__global__ void example(float *d_a, float *d_b,
  float *d_output, int n){

  //Calculate global thread index
  //blockIdx.x – the current thread block number
  //blockDim.x – the thread block size
  //threadIdx.x – the thread index within the current block
  int global_tid = blockIdx.x * blockDim.x + threadIdx.x;

  if(global_tid < n){
    d_output[global_tid] = d_a[global_tid] + d_b[global_tid];
  }
}
```

K20:
Global Memory: 440 clocks
Shared Memory: 48 clocks

Parallel primitives

- Reduction

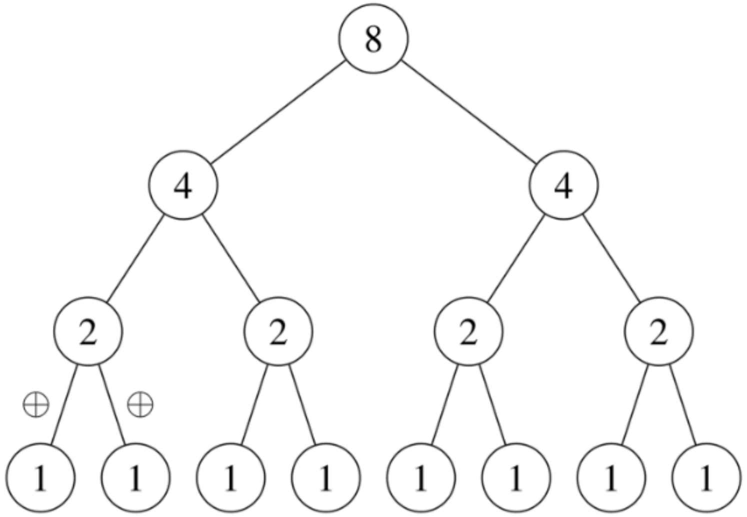
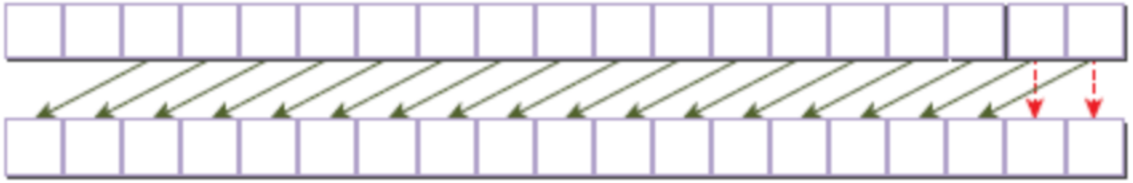


Figure 3 **Sum parallel reduction.**

```
__device__
float warp_reduce(float x) {
  for (int d = 16; d > 0; d /= 2)
    x += __shfl_down(x, d);
  return x;
}
```



_shfl_down (val,2): Shift the value to the left two lanes.

Parallel primitives

- Parallel prefix sum

Algorithm 1 Simple scan

```
for d=1 to log₂n do
    for k=0 to n−1 in parallel do
        if k ≥ 2^(d−1) then
            x[k] := x[k − 2^(d−1)] + x[k]
        end
    end
end
```

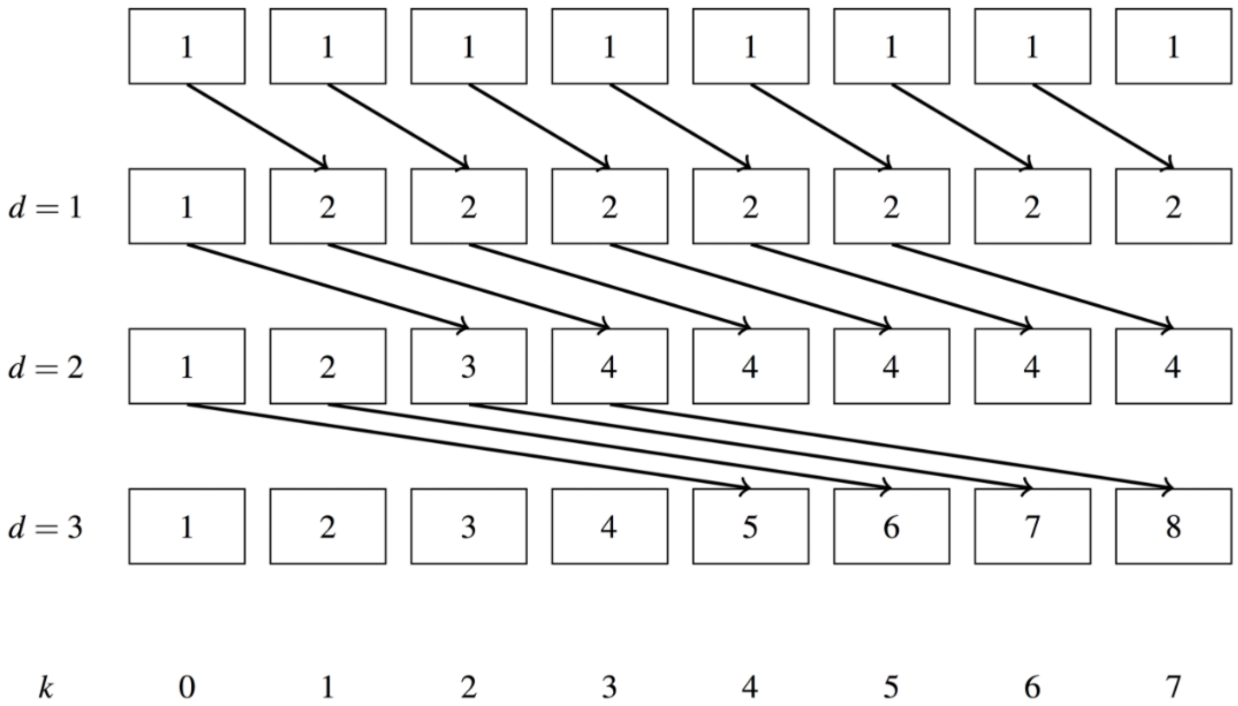


Figure 4 Simple parallel scan example.

Parallel primitives

- Parallel prefix sum

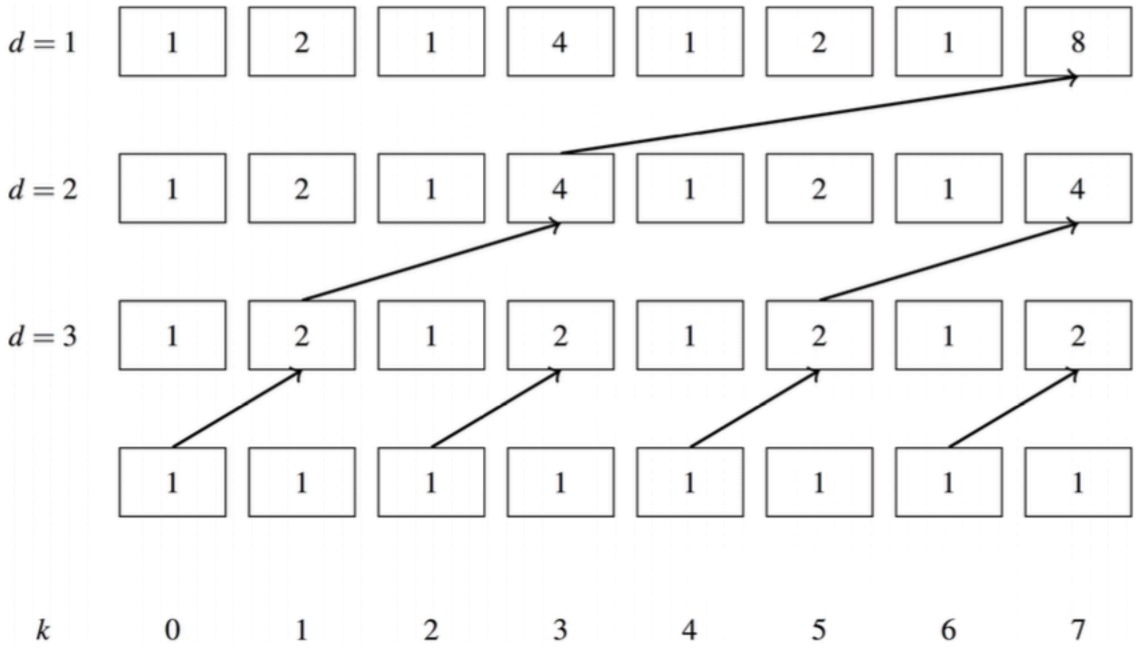


Figure 5 **Blelloch scan upsweep example.**

Algorithm 2 **Blelloch scan—upsweep**

```
offset = 1
for d= log2 n to 1 do
    for k=0 to n–1 in parallel do
        if k < 2^(d–1) then
            ai = offset × (2 × k + 1) – 1
            bi = offset × (2 × k + 2) – 1
            x[bi] = x[bi] + x[ai]
        end
    end
    offset = offset * 2
end
```

Parallel primitives

- Parallel prefix sum

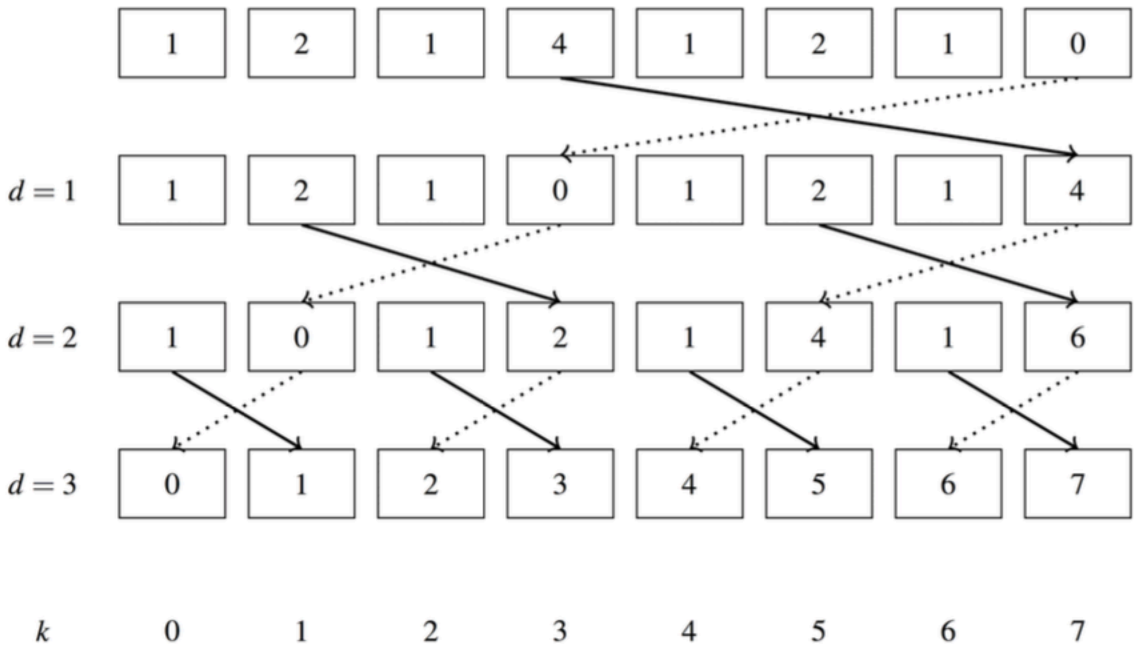


Figure 6 **Blelloch scan downsweep example.**

Algorithm 3 **Blelloch scan—downsweep**

```
offset = 2^(log2 n − 1)
x[n − 1] := 0
for d=1 to log2n do
    for k=0 to n−1 in parallel do
        if k < 2^(d−1) then
            ai = offset × (2 × k + 1) − 1
            bi = offset × (2 × k + 2) − 1
            t = x[ai]
            x[ai] = x[bi]
            x[bi] = x[bi] + t
        end
    end
    offset = offset/2
end
```

Parallel primitives

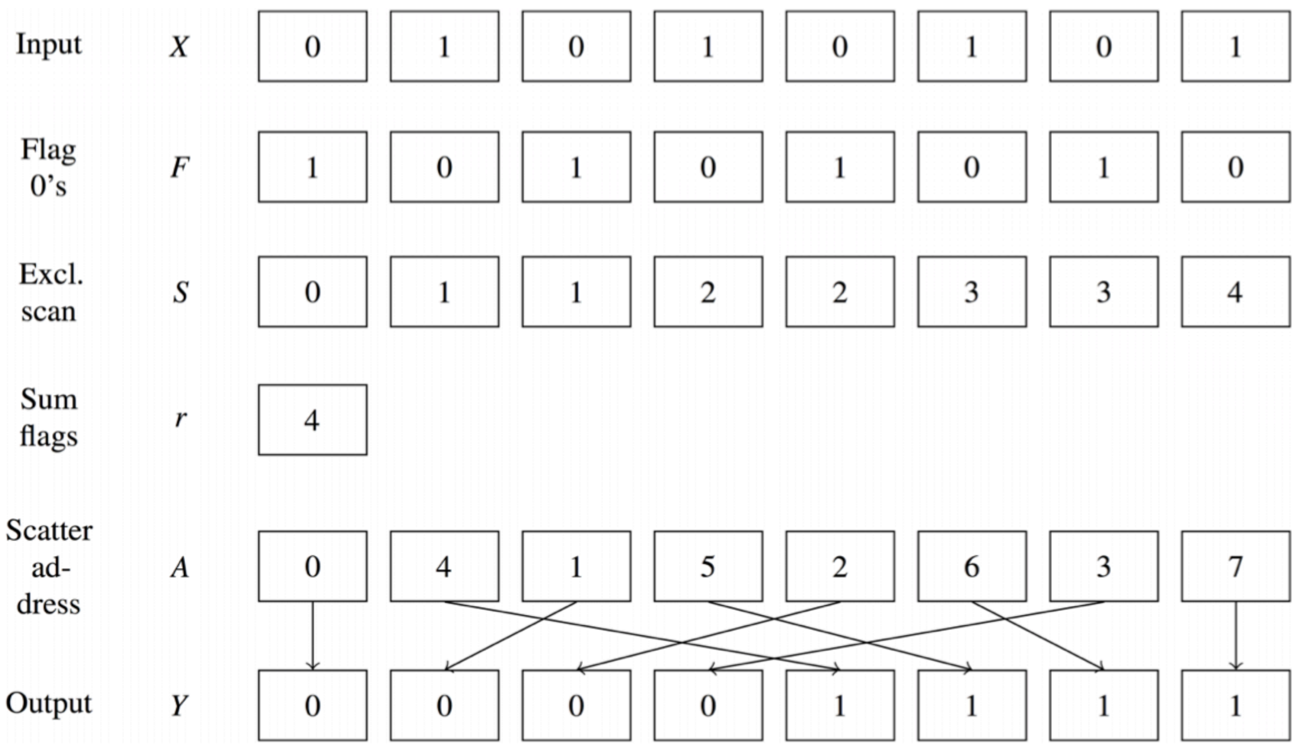


Figure 7 **Radix sort example.**

Algorithm 4 **Radix sort pass**

```
Input :X
Output :Y
for i = 0 to n − 1 in parallel do
    F[i] := bit_flip(X[i])
end
S := exclusive_scan(F)
r := S[n − 1] + F[n − 1]
for i = 0 to n − 1 in parallel do
    if X[i] = 0 then
        A[i] := S[i]
    else if X[i] = 1 then
        A[i] := i − S[i] + r
end
for i = 0 to n − 1 in parallel do
    Y[A[i]] := X[i]
end
```

Parallel primitives

- Scan and reduce on multiple sequences

Input labels	1	0	2	2	1	0	0	2	1	0	1
Input values	5	3	6	0	1	9	9	3	7	1	4
Multireduce: sum of all values for each label					22	17	9				
Multican: prefix sums for same label	0	0	0	6	5	3	12	6	6	21	13

PARALLEL TREE CONSTRUCTION

- Phase 1: find splits

Table 9 **Device memory layout: feature values.**

	f0			f1	f2			
Node id	0	0	0	0	0	0	0	0
Instance id	0	2	3	3	2	0	1	3
Feature value	0.1	0.5	0.9	5.2	3.1	3.6	3.9	4.7

Table 10 **Device memory layout: gradient pairs.**

Instance id	0	1	2	3
Gradient pair	p_0	p_1	p_2	p_3

PARALLEL TREE CONSTRUCTION

- Phase 1: find splits

Table 11 **A single thread block evaluating splits.**

	Thread block 0				⇒			
	↓	↓	↓	↓				
	f0							
Instance id	0	2	3	1	7	5	6	4
Feature value	0.1	0.2	0.3	0.5	0.5	0.7	0.8	0.8
Gradient pair	p_0	p_2	p_3	p_1	p_7	p_5	p_6	p_4

PARALLEL TREE CONSTRUCTION

- Phase 1: find splits

Table 12 Interleaved node buckets.

	f0			f1	f2			
Node id	2	1	2	2	1	2	1	2
Instance id	0	2	3	3	2	0	1	3
Feature value	0.1	0.5	0.9	5.2	3.1	3.6	3.9	4.7

Table 13 Sorted node buckets.

	f0			f1	f2			
Node id	1	2		2	1		2	
Instance id	0	2	3	3	2	1	0	3
Feature value	0.5	0.1	0.9	5.2	3.1	3.9	3.6	4.7

PARALLEL TREE CONSTRUCTION

- Phase 2: update node positions

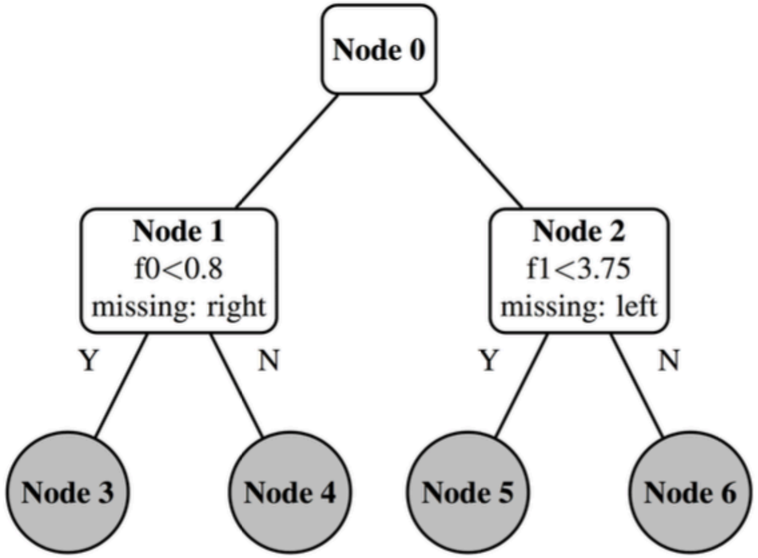


Figure 8 **Decision tree: four new leaves.**

Table 16 **Updated missing direction.**

Instance id	0	1	2	3
Node id	4	5	5	4

Table 17 **Node ID map: update based on feature value.**

Instance id	0	1	2	3
Node id	3	5	6	4

	f0			f1			
Node id	1	2	2	1	1	2	2
Instance id	0	2	1	3	1	1	2
Feature value	0.75	0.5	0.9	2.7	4.1	3.6	3.9

PARALLEL TREE CONSTRUCTION

- Phase 3: sort node buckets

Table 18 **Per feature value array: updated.**

	f0			f1			
Node id	3	6	5	4	3	5	6
Instance id	0	2	1	3	0	1	2
Feature value	0.75	0.5	0.9	2.7	4.1	3.6	3.9

EVALUATION

- Hardware

Table 19 **Hardware configurations.**

Configuration	CPU	GHz	Cores	CPU arch.
#1	Intel i5-4590	3.30	4	Haswell
#2	Intel i7-6700K	4.00	4	Skylake
#3	2× Intel Xeon E5-2695 v2	2.40	24	Ivy Bridge

Configuration	GPU	GPU memory (GB)	GPU arch.
#1	GTX970	4	Maxwell
#2	Titan X	12	Pascal
#3	–	–	–

EVALUATION

Table 20 Datasets.

Dataset	Training instances	Test instances	Features
YLTR[a]	473,134	165,660	700
Higgs[b]	10,500,000	500,000	28
Bosch[c]	1,065,373	118,374	968

Table 21 Parameters.

Dataset	Objective	eval_metric	max_depth	Eta	Boosting iterations
YLTR	rank:ndcg	ndcg@10	6	0.1	500
Higgs	binary:logistic	auc	12	0.1	500
Bosch	binary:logistic	auc	6	0.1	500

EVALUATION

- Accuracy

Table 22 Accuracy benchmarks.

Dataset	Subset	Metric	CPU accuracy	GPU accuracy
YLTR	0.75	ndcg@10	0.7784	0.7768
Higgs	0.25	auc	0.8426	0.8426
Bosch	0.35	auc	0.6833	0.6905

Table 23 Accuracy benchmarks—sorting version only.

Dataset	Subset	Metric	GPU accuracy (sorting version only)
YLTR	0.75	ndcg@10	0.7776
Higgs	0.25	auc	0.8428
Bosch	0.35	auc	0.6849

EVALUATION

- Speed

Table 24 Configuration #1 speed benchmarks.

Dataset	Subset	CPU time (s)	GPU time (s)	Speedup
YLTR	0.75	1,577	376	4.19
Higgs	0.25	7,961	1,201	6.62
Bosch	0.35	1,019	249	4.09

Table 25 Configuration #2 speed benchmarks.

Dataset	Subset	CPU time (s)	GPU time (s)	Speedup
YLTR	1.0	877	277	3.16
Higgs	1.0	14,504	3,052	4.75
Bosch	1.0	3,294	591	5.57

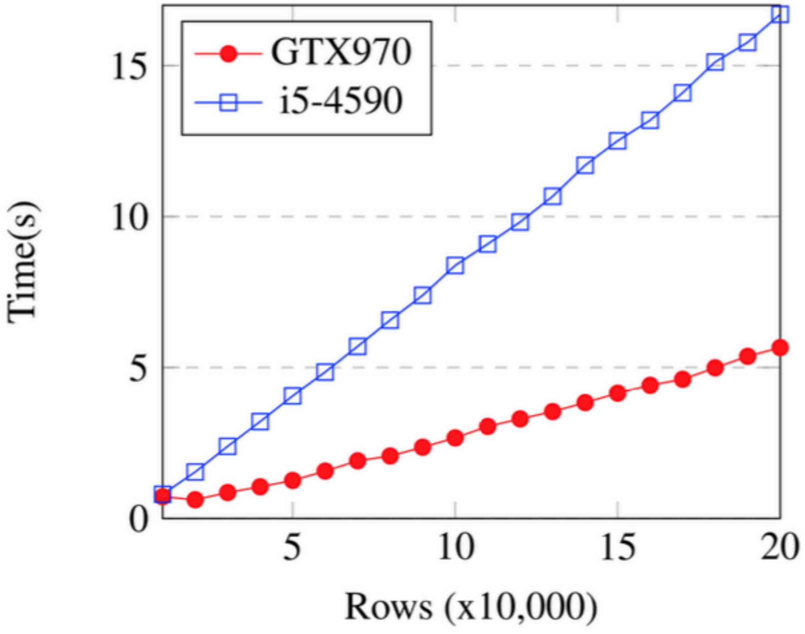


Figure 9 Bosch: time vs problem size.

EVALUATION

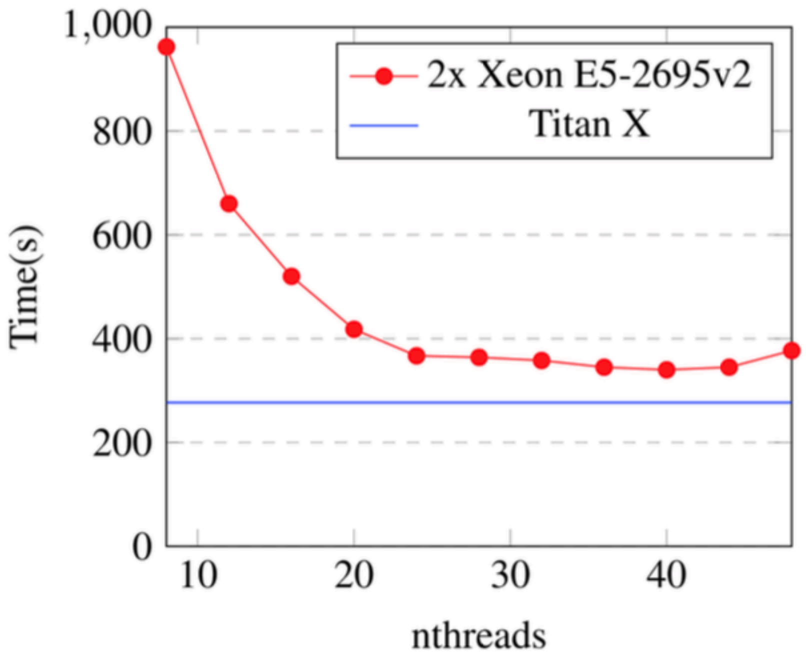


Figure 10 **Yahoo LTR: *n*-threads vs time.**

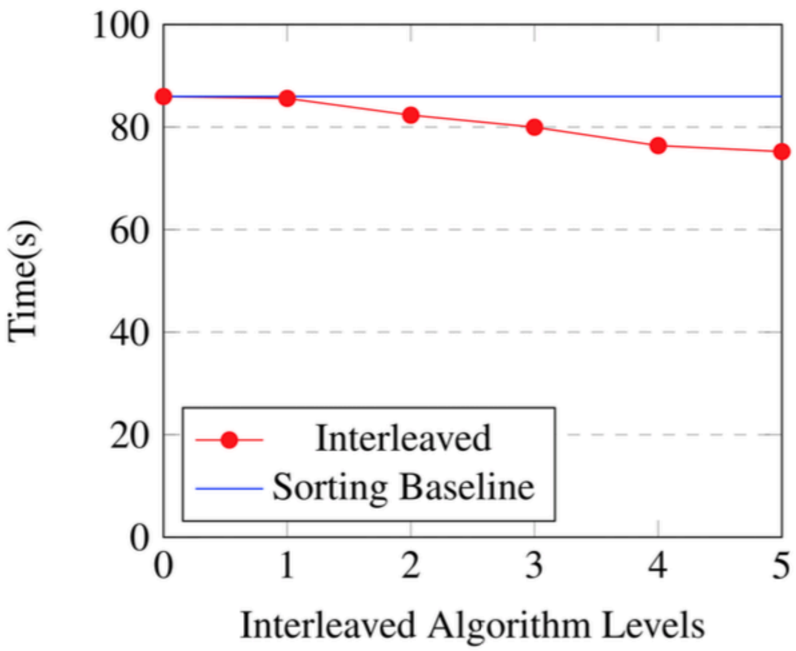


Figure 11 **Bosch: interleaved algorithm threshold.**

EVALUATION

Table 27 **Memory: GPU algorithm.**

Dataset	Device memory (GB)
YLTR	4.03
Higgs	11.32
Bosch	8.28

Table 28 **Memory: CPU algorithm.**

Dataset	Host memory (GB)
YLTR	1.80
Higgs	6.55
Bosch	3.28

CONCLUSION

- The algorithm is built on top of efficient parallel primitives and switches between two modes of operation depending on tree depth.
- All nodes in a level concurrently.
- Sparsity aware.
- Problem：
  - The entire input matrix must fit in device memory and device memory consumption is approximately twice that of the host memory used by the CPU algorithm.
  - The number of streaming multiprocessors is limited.
  - Shared memory capacity.

Thank you！
Q&A