Wildebeest: SGD SVM on Migrating Threads

Application Design for the Lucata Pathfinder-S

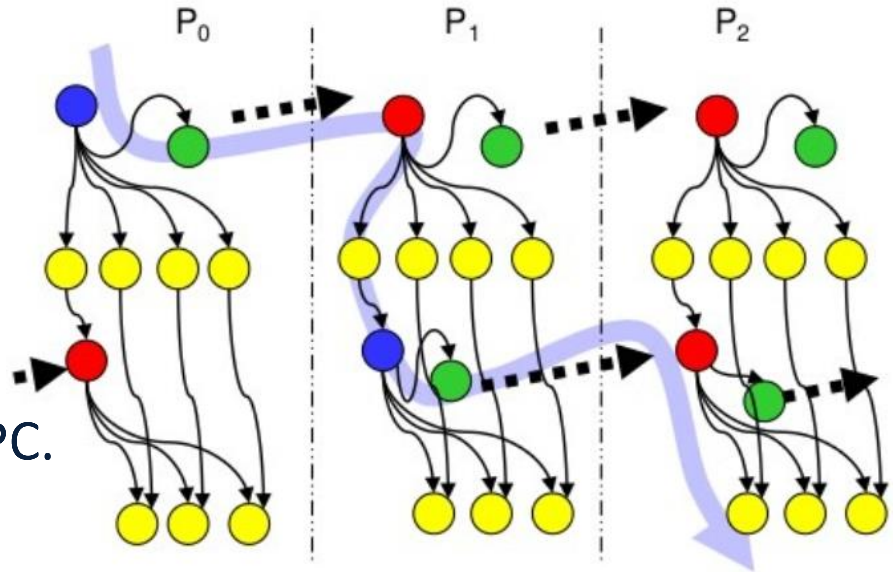
PEARC2021 - Brian A Page, bpage1@nd.edu

Overview

1. Some conventional architecture limitations
2. Introduction into SVM and SGD
3. Hogwild and the origins of Wildebeest
4. Wildebeest
    1. Design
    2. Execution
    3. Scaling
5. Comparison to Conventional

Communication vs Computation

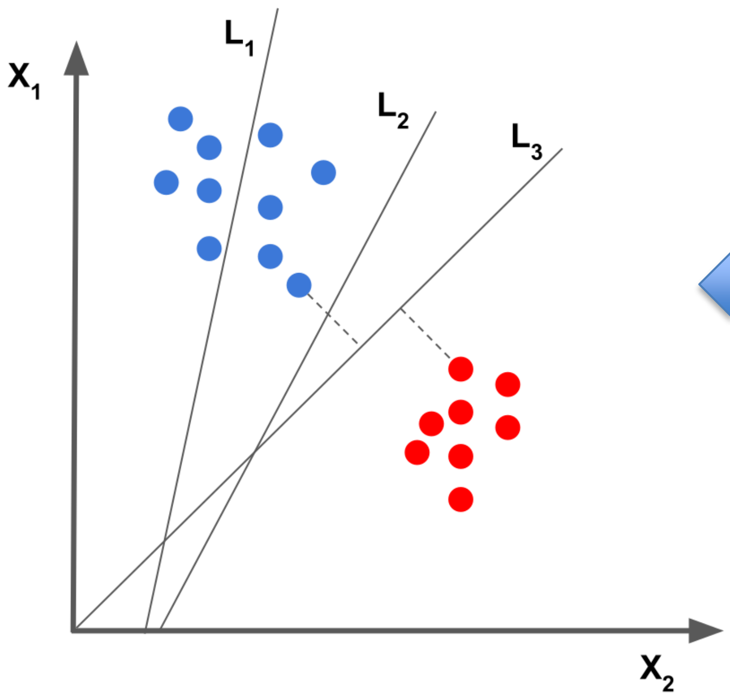
- Distributed systems require
- inter-process communication
- Communication can be synchronous or asynchronous
- Message Passing Interface (MPI) most common communication method in HPC.
- **MPI has significant overhead and degrades overall performance at scale**



Conventional Limitations

1. Cache coherency traffic severely degrade performance
2. Distributed systems require explicit communication
3. Process and Thread parallelism comes adds overhead
4. Can be very difficult to develop highly parallel applications
5. Many more . .

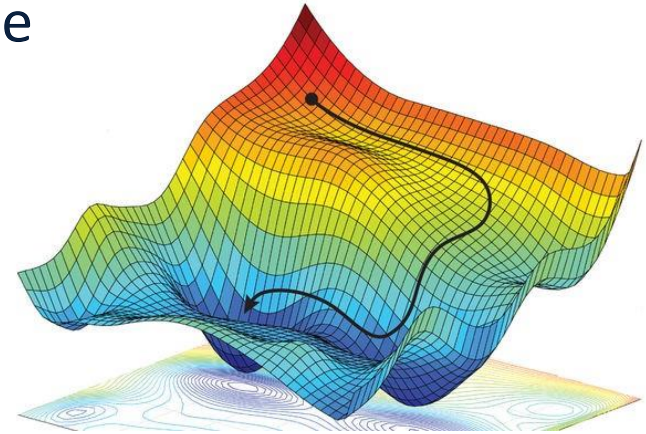
Support Vector Machine (SVM)



- SVMs are supervised learning models primarily used data classification and regression analysis
- "Training" on samples adjusts the hyperplane.
- Iterate through training data to find optimum result (hyperplane)
- Learned hyperplane used to predict classification of future data

Stochastic Gradient Descent (SGD)

- Iterative method which optimizes some objective function (e.g. hinge loss)



- Estimates gradient using randomly selected subset of data

- Improved performance at expense of convergence rate

Algorithm 1

1: **loop**
2: Sample e uniformly at random from E
3: Read current state x_e and evaluate $G_e(x)$
4: **for** $v \in e$ **do** $x_v \leftarrow x_v - \gamma b_v^T G_e(x)$
5: **end loop**

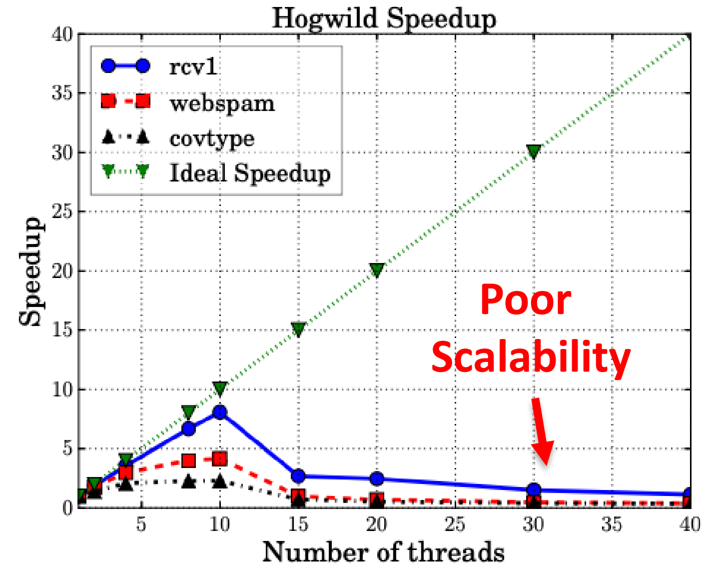
Hogwild Algorithms

Hogwild!:

Lock-Free parallel SGD for Hyper-sparse training data sets

Possible via atomic operations on shared result vector

False sharing and coherency traffic limited thread counts and therefore maximum achievable speedup

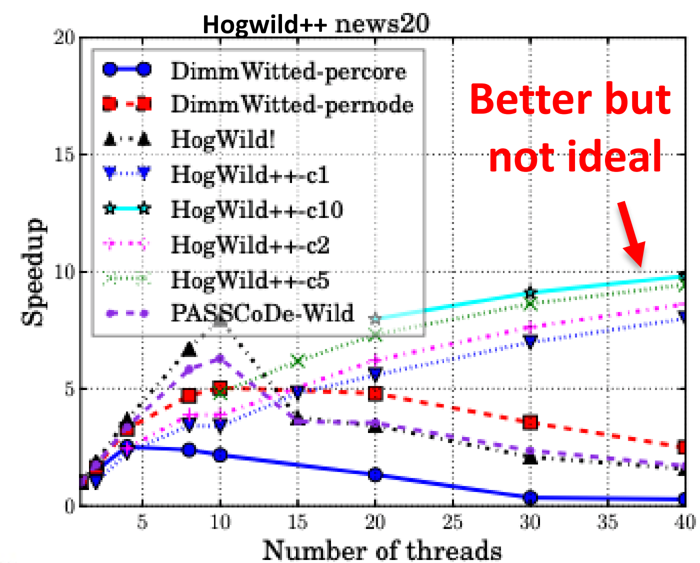


Hogwild++:

Computation divided into "clusters"

Reduced sharing decreases invalidation traffic

Depends on Circular propagation of local results to neighbors via token-passing



Wildebeest

https://github.com/bripage/wildebeest.git

Wildebeest on Pathfinder

Design Considerations

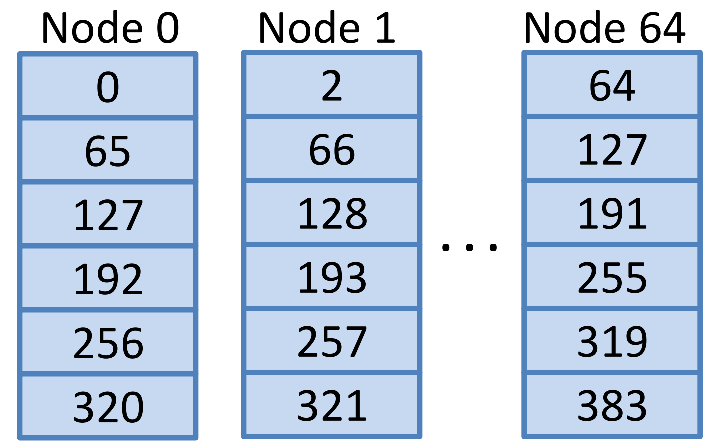
- Want to run in a shared environment
- Reduce unnecessary memory accesses
  - Remember its cacheless!
- Allow threads to migrate freely throughout the system
- But eliminate unnecessary migrations
  - They aren’t free!
- Get better scaling then conventional

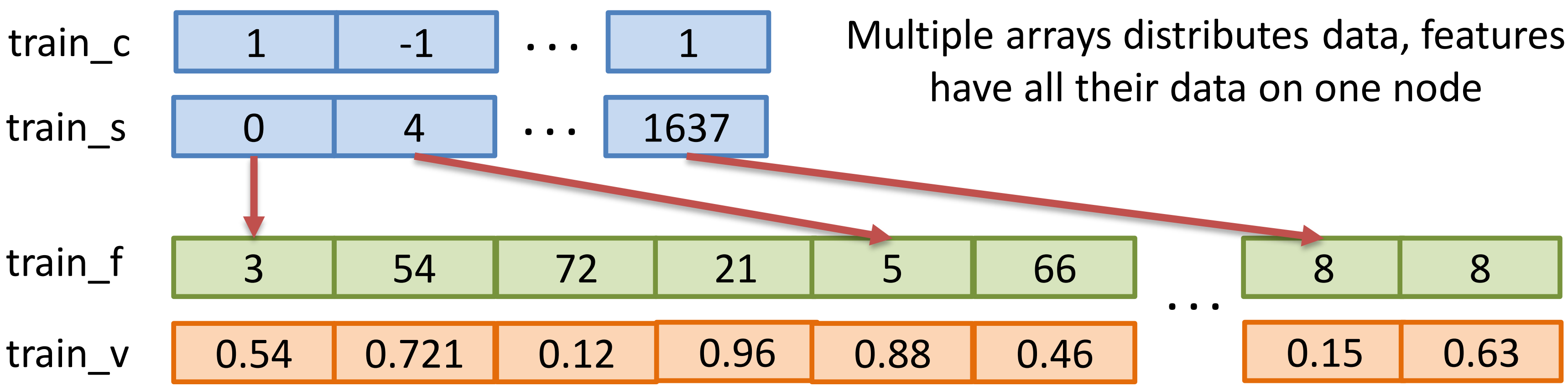
Communication vs Computation

- **Thread migration = communications**
- Can leverage communication to improve performance
- Saturate node/core queues for constant execution
- Migrations overlapped with useful computation
- Efficiency: Migrations done directly in hardware!

Memory Allocation

- **mw_malloc1dlong** stripes array across all nodes
- Utilizes entire PGAS address space
- Distributes memory accesses across more hardware
- Adjacent elements are on different nodes

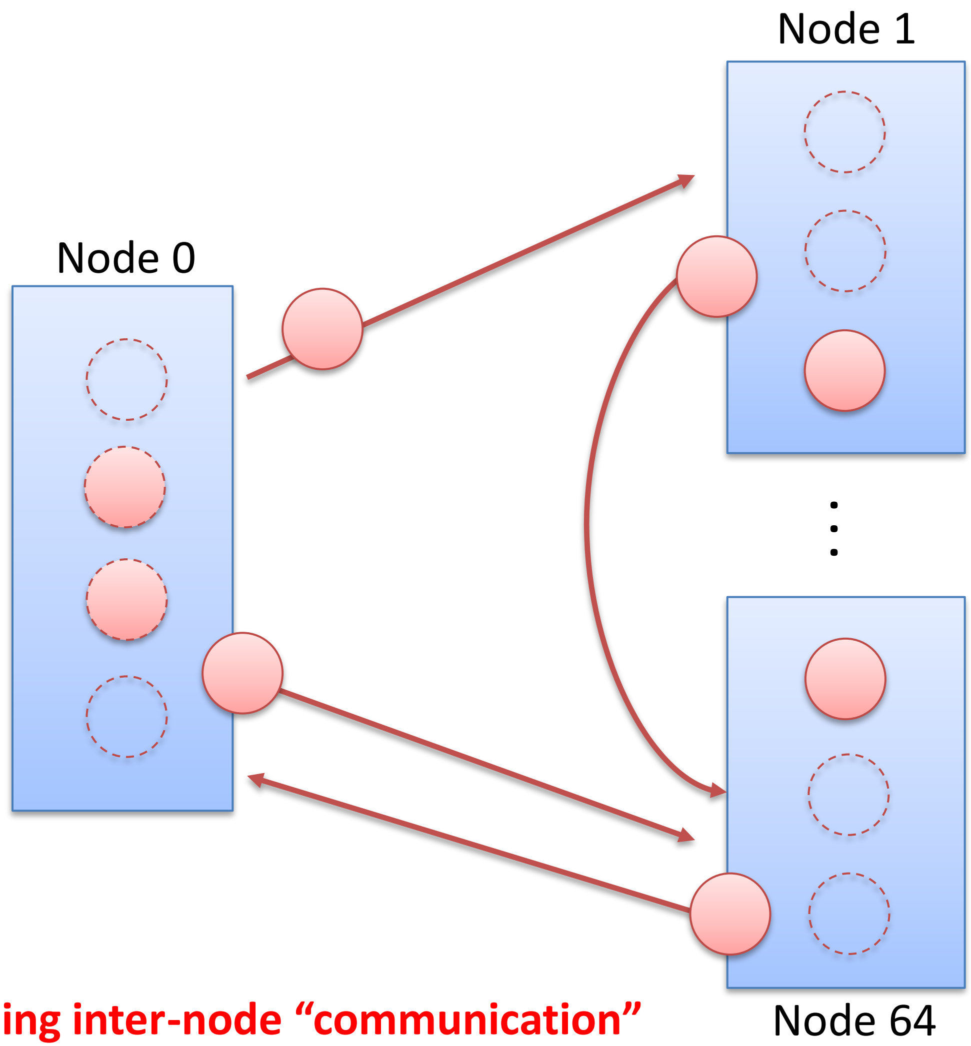




Multiple arrays distributes data, features have all their data on one node

Thread Spawn

- **T** trainer threads spawned
- Thread t_i spawned on node n_(N%T)
- Threads migrate while training as they access remote memory addresses.
- Distributes workload throughout system



Allows computation to continue while performing inter-node "communication"

Thread Spawn: Epoch

Scalar update each epoch (decreases update impact)

```
long scalar = initial_step_size;
Long gamma = initial_step_decay;

for (long e = 0; e <= epochs; e++){
    if (e > 1){
        scalar *= gamma;
    }
    for (long t = 0; t < trainers; t++){
        cilk_migrate_hint(&train_s[t]);
        cilk_spawn train(t, scalar);
    }
    cilk_sync;
}
```

Trainers spawned throughout system

Epoch ends when all samples have been evaluated

Training: Overview

Training loop has 2 distinct phases:

1) Determine gradient
2) Decide model update type

dist is distance from current model

scalar is updated each epoch, decreases as less adjustments is needed

```
void train(long i, long scalar){
    long dist;
    while(s < s_count){
        dist = getDistance(s);
        if (dist < 1){
            full_update(s, scalar);
        } else {
            boost_update(s, scalar);
        }
        s += trainer_count;
    }
}
```

Model wrong
large retraining

Model correct
small "boost"

Training: Gradient

First pass through the system:

To compute sample's gradient need to read all sample's features.

All memory operations for a feature are on same node

Thread migrates only once per iteration

Model's class = dist * known class

```
long getDistance(long s){
    long f, dist = 0, di, ltmp;
    for (j = train_s[s];
            j < train_s[s+1]; j++){
        f = train_f[j];
        ltmp = train_v[j] * model[f];
        dist += ltmp;
    }
    dist *= train_c[s];
    return dist;
}
```

All on same node

Training: Full Update

- Updates are second pass through system.
- Current model *incorrectly* classifies sample.
- Update model using samples features

The sample's features are scaled and their weight in determining model accuracy used to adjust model value.

```
void full_update(long s, long scalar){
        long f, di, mtmp, ltmp;
        di = scalar * train_c[s];
        for (long j = 0;
            j < train_s[s]; j++){
                f = train_f[j];
                ltmp = di * deg[f];
                mtmp = model[f] + ltmp;
                ltmp = scalar * deg[f];
                mtmp = mtmp * (1 - ltmp);
                model[f] = mtmp;
        }
}
```

Reuse temp variables to decrease load/stores from main memory

Training: Boost Update

Current model *incorrectly* classifies sample.

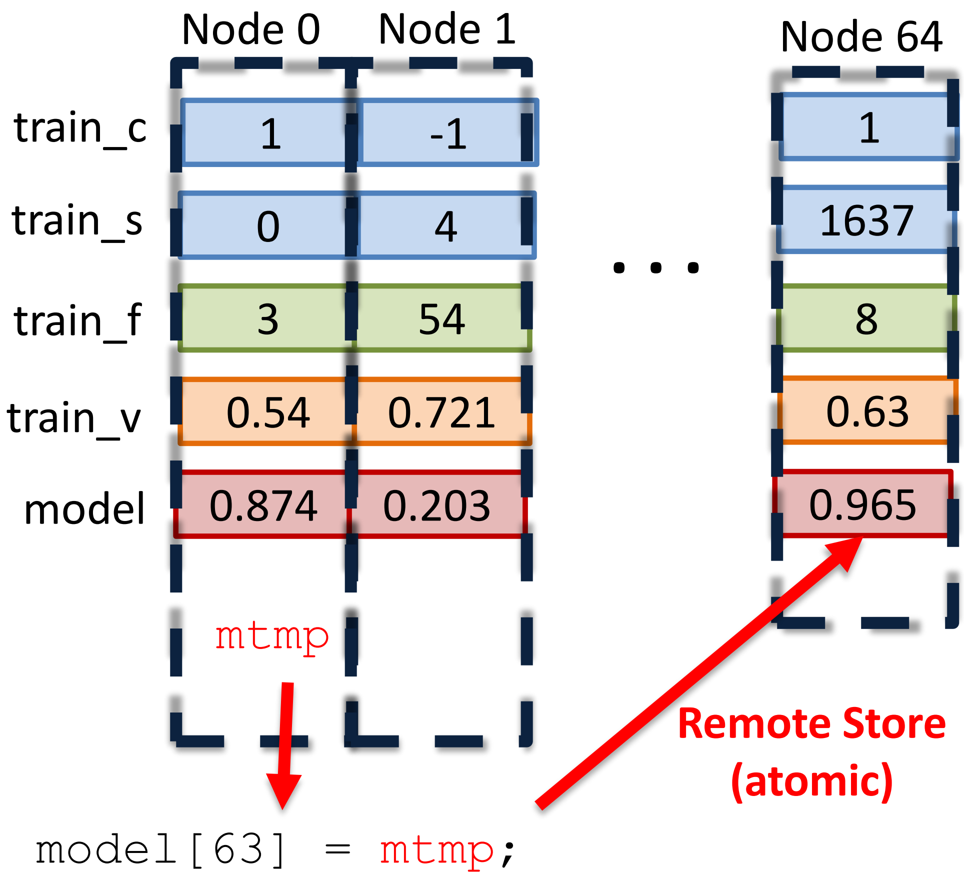
Boosting "nudges" model in the right direction

Requires fewer operations then full_update()
Becomes more common with higher epoch counts

```
void boost_update(long s, long scalar){
    long f, di, mtmp, ltmp;
    for (long j = 0;
      j < train_s[s]; j++){
        f = train_f[j];
        mtmp = model[f];
        ltmp = scalar * deg[f];
        mtmp = mtmp * (1 - ltmp);
        model[f] = mtmp;
    }
}
```

Reuse temp variables to decrease load/stores from main memory

Workload Imbalance Avoidance



- Possible Node/Core "hot" spots from over-migration
- Model is stripped
- Model is common write target for EVERY training loop iteration
- <u>Remote Stores!</u>

Running on Pathfinder

Data Set Preparation

- Originally LIBSVM format
- Dataset utilities included in repository

training

```
shuffle -i a8a_train.dat > a8a_train.shuf
libsvm2csv -i a8a_train.shuf -s 22696 > a8a_train.csv
csv2bin -i a8a_train.csv -o a8a_train.bin
```

testing

```
libsvm2csv -i a8a_test.dat -s 9865 > a8a_test.csv
csv2bin -i a8a_test.csv -o a8a_test.bin
```

- Shuffle training samples (optional for training)
- Convert LIBSVM to CSV
- Convert CSV to Binary (int64_t)

Execution: Single Node

Loads program instructions to node card

Migrating thread Code *.mwx

```
n0:~/wildebeest/migthreads$ emu_handler_and_loader 0 16 -- wildebeest.mwx --train-data ../test_data/a8a_train.bin
--train-samples 22696 --train-points 314815 --test-data ../test_data/a8a_test.bin --test-samples 9865 --test-points
136777 -f 123 -e 10 --initial-step-size 0.35 --initial-step-decay 0.9 --trainers 512  points = 547108
Epoch 1: 84.835276
Epoch 2: 84.723770
Epoch 3: 84.470349
Epoch 4: 84.470349
Epoch 5: 84.916371
Epoch 6: 84.561581
Epoch 7: 84.967055
Epoch 8: 84.865686
Epoch 9: 84.875823
Epoch 10: 84.277749
```

Training/Testing Data parameters

Post epoch model accuracies

Execution: Multi-node

Loads program instructions to all node cards

```
n0:~/wildebeest/migthreads$ emu_multinode_exec 0 -- wildebeest.mwx --train-data ../test_data/a8a_train.bin --train
-samples 22696 --train-points 314815 --test-data ../test_data/a8a_test.bin --test-samples 9865 --test-points 136777
-f 123 -e 10 --initial-step-size 0.35 --initial-step-decay 0.9 --trainers 2048
[STATUS]: Checking nodes to ensure ok to run...
[STATUS]: Copying wildebeest.mwx to nodes.
[STATUS]: Launching emu_loader and emu_seq_handler_background on n1 to n7
[STATUS]: Gathering launch logs from non-local nodes.
[STATUS]: Running emu_handler_and_loader
Epoch 1: 84.885960
Epoch 2: 84.764318
Epoch 3: 84.774455
Epoch 4: 85.007602
Epoch 5: 84.906234
Epoch 6: 84.896097
Epoch 7: 84.531170
Epoch 8: 84.896097
Epoch 9: 84.916371
Epoch 10: 84.774455
[STATUS]: Run complete; gathering logs.
[STATUS]: Copying concatenated logs with PID=11139 into /home/bpage/wildebeest/migthreads.
[STATUS]: Checking mn_exec_usr.11139.log for errors...
[STATUS]: Checking mn_exec_sys.11139.log for errors...
[STATUS]: emu_multinode_exec complete.
```

More hardware
More trainers!

Post epoch model accuracies

Run logs gathered from all nodes upon completion (or error)

Comparing to Conventional

Minimal Code Alternations

Pathfinder

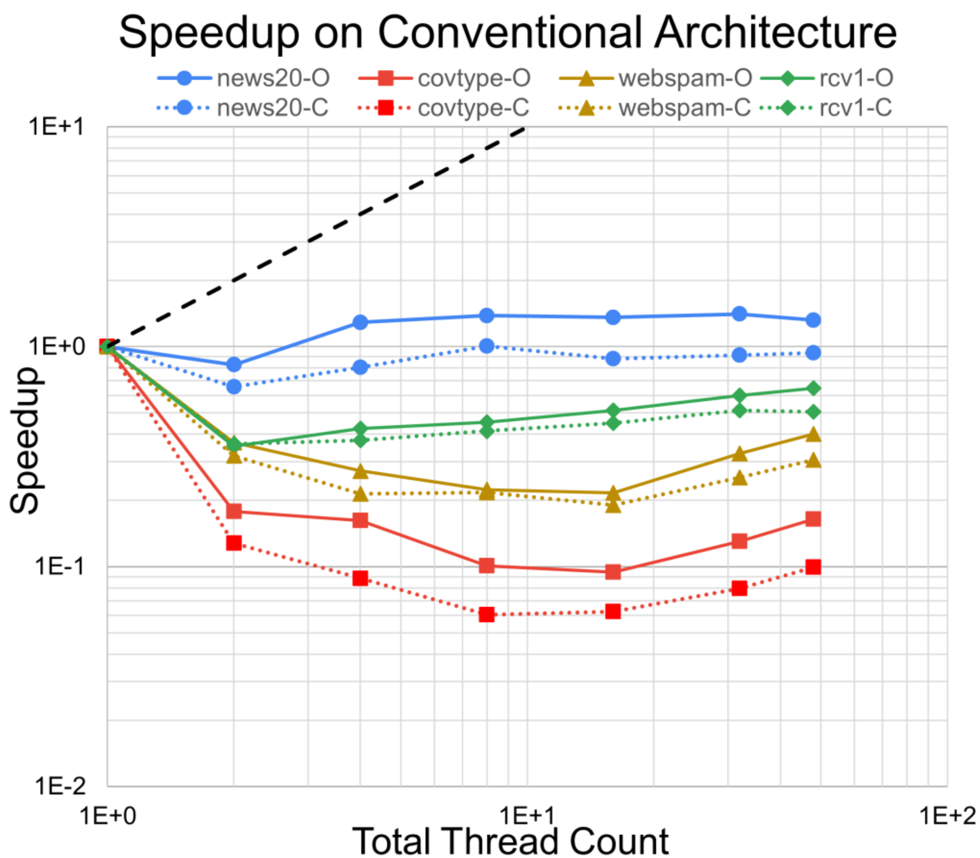
```
for (long e = 0; e <= epochs; e++){
    if (e > 1){
        scalar *= gamma;
    }
    for (long t = 0;
        t < trainers; t++){
        cilk_migrate_hint(&train_s[t]);
        cilk_spawn train(t, scalar);
    }
    cilk_sync;
}
```

VS

Conventional OpenMP

```
for (long e = 0; e <= epochs; e++){
    if (e > 1){
        scalar *= gamma;
    }
   #pragma omp parallel num_threads(trainers) \
    shared(scalar, train_s, train_f, train_c, \
    train_v, deg, model) private(thread_id)
   {
        thread_id = omp_get_thread_num();
        train(thread_id, scalar)
    }
}
```

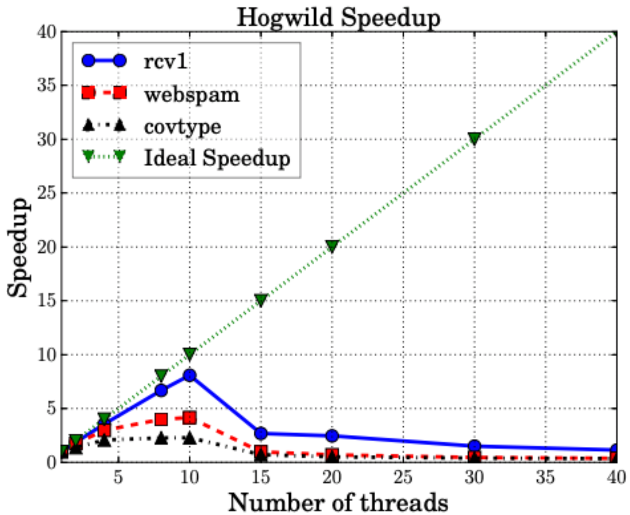
Conventional Scaling

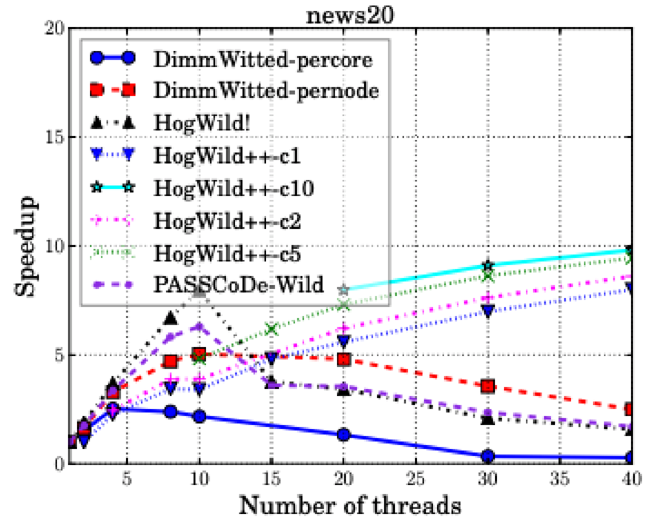


- Shared Memory implementation requires minimal code changes and no algorithmic alterations
- Tested both OpenMP and Cilk Plus conventional versions
- Conventional system achieves poor scalability regardless of thread library

Migrating Thread Scaling

Conventional





Lucata