PetaVision Neural Simulation Toolbox on Intel KNLs

Boram Yoon[1,†], Pete Schultz[1,2], and Garrett T. Kenyon[1,2]

[1]Los Alamos National Laboratory, USA, [2]New Mexico Consortium, USA , [†]boram@lanl.gov

Abstract

We implemented a large-scale neuromorphic algorithm, corresponding to a Sparse Prediction Machine (SPM), on the Los Alamos Trinity supercomputer. Specifically, we used PetaVision, an open source high-performance neural simulation toolbox, to implement a 4-layer SPM applied to ImageNet video that efficiently utilized up to 8192 KNL nodes. The goal of an SPM is to predict future states of a system from a sequence of previous states, or in the case of video, to predict a subsequent frame from previous frames. In our training, the SPM was able to predict the 8th frame from the preceding 7 frames, including successful separation of foreground and background motion.

PetaVision

- Open source neural simulation machine learning toolbox
  - github.com/PetaVision
- Written in C/C++
- Supports GPU acceleration (cuDNN)
- Parallelization through MPI + OpenMP/Cuda
- Supports sparse coding unsupervised learning

Trinity

- Supercomputer at Los Alamos National Laboratory in United States
- Architecture: Cray XC40
- Interconnect: Cray Aries interconnect
- Compute Nodes:

Intel Xeon E5-2698v3 Haswell	Intel Xeon Phi Knights Landing
9436 nodes	> 9500 nodes
2.3GHz, dual socket 16 cores/socket	1.4 GHz, 1 socket, 68 cores
128GB DDR4	96GB DDR4 + 16GB HBM

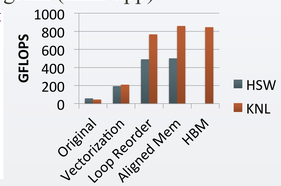
Single-node Optimization

- Bottle-neck routine is extracted as a stand-alone program (mini-app)

```
int accumulate(float *a, float *w, float *gSyn, int nk, float dt_factor, int sf) {
  for (int k = 0; k < nk; k+=sf) {
    *gSyn += dt_factor * a[k] * w[k];}
}

#pragma omp parallel for schedule(static)
for (int idx = 0; idx < numNeurons; idx++) {
  float *activityStartBuf = data1 + (idx/shift_period)*shift;
  float *gSyn             = data2 + idx;
  float *weightStartBuf   = data3 + (idx%shift)*syp*yPatchSize;
  for (int ky = 0; ky < yPatchSize; ky++){
    float *a = activityStartBuf + ky*sy;
    float *w = weightStartBuf   + ky*syp;
    accumulate(a, w, gSyn, numPerStride, dt_factor, sf);
}}
```

```
for (int ky = 0; ky < yPatchSize; ky++){
#pragma omp parallel for schedule(static)
  for (int feature = 0; feature < nfp; feature++) {
    for (int idx = feature; idx < numNeurons; idx += nfp) {
      float *activityStartBuf = data1 + (idx/shift_period)*shift;
      float *gSyn             = data2 + idx;
      float *weightStartBuf   = data3 + (idx%shift)*syp*yPatchSize;
      float *a = activityStartBuf + ky*sy;
      float *w = weightStartBuf   + ky*syp;
      float dv = 0.0;
#pragma vector aligned
      for (int k = 0; k < numPerStride; k++) {
        dv += a[k] * w[k];
      }
      *gSyn += dt_factor * dv;
}}}
```

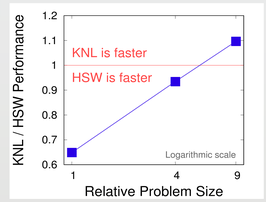


- **Vectorization** of the most inner loop, **avoid multi-versioning**
- **Loop reordering / tiling** for better cache utilization

Only on full program, not on min-app:

- **Collapsing loops** for better threading efficiency
- Optimizing **OpenMP scheduling**

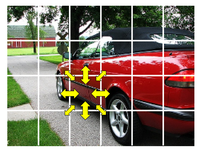
- Haswell vs KNL
  - Full calculation performance (not only the hotspot kernel mini-app)
  - Single node: 2 MPI ranks on HSW, 1 MPI rank on KNL (quad-cache)
  - KNL is faster when problem size is large
  - Performance varies with ML model and run parameters



Multi-node Optimization

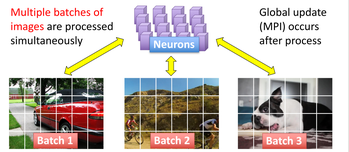
- Two level parallelization:

Parallelization 1: Gridding



- Image is gridded for parallelization
- Each MPI rank works on a cell
- Most MPI communications are between immediate neighbors in 2D grid

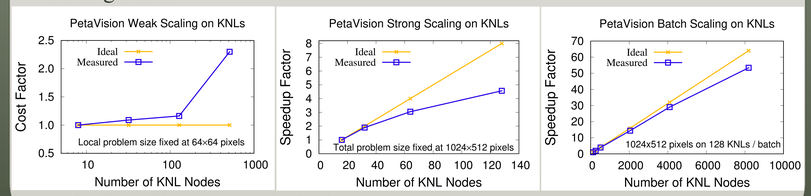
Parallelization 2: Batching



- Optimizations
  - Non-blocking MPI communication
  - Overlapping MPI comm. with calculations
  - N-to-1 Serial I/O → N-to-M Parallel I/O
- Tunings
  - quad_cache is slightly (1-5%) faster than quad_flat
  - 1 MPI rank / KNL is optimal (2 for HSW)
  - N_threads/core = 2 (large problems) or 1 (others)

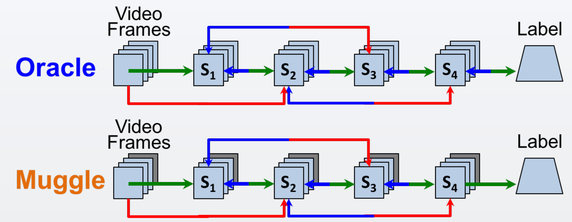
- Direct comparison between "before" and "after" multi-node optimization is NOT available because the ML models we use cannot be run on the "before" version of PetaVision

- Scaling curves



- Cost Factor: Wall-time compared to 8-nodes run
- Speedup Factor: Inverse wall-time compared to the smallest (16 nodes on strong, 128 nodes on batch scaling) run

Sparse Prediction Machine (SPM)



- A pair of deep sparse generative autoencoders(DSGAs): Muggle & Oracle
- Only Oracle can see "future" video frame and Label; share connections
- Muggle tries to generate sparse representations that match Oracle's

Results

- Goal
  - Predict future states (8th frame) of a system from a sequence of previous states (7 frames)
- Training
  - ImageNet video frames
  - 256×128 pixels & 512×256 pixels images
  - Training carried out on 3200 nodes of Trinity cluster
- Results
  - The SPM was able to predict the 8th frame from the preceding 7 frames, including successful separation of foreground and background motion

Original

Prediction

Conclusion

- **PetaVision is optimized for Trinity supercomputer**
  - Optimizations applied for single-node and multi-node
  - Reasonable scaling up to 8192 KNL nodes
- **Sparse prediction machine (SPM) is trained**
  - ImageNet Video frames used for predicting 8th frame from previous 7 frames
  - Results encouraging

Acknowledgement

This work is supported by Los Alamos National Laboratory LDRD program.
Computations were carried out using Trinity cluster at Los Alamos National Laboratory.