CosmoFlow: Using Deep Learning to Learn the Universe at Scale

Amrita Mathuriya[*], Deborah Bard[†], Peter Mendygral[‡], Lawrence Meadows[*], James Arnemann[§], Lei Shao[¶], Siyu He[**†|], Tuomas Kärnä[*], Diana Moise[‡], Simon J. Pennycook[¶], Kristyn Maschhoff[‡], Jason Sewall[¶], Nalini Kumar[¶], Shirley Ho[**†|], Michael F. Ringenburg[‡], Prabhat[†] and Victor Lee[¶]

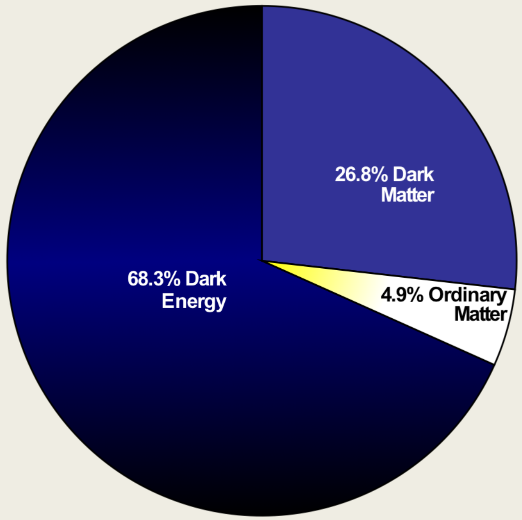
[*]Intel OR, [†]LBNL, [‡]Cray, [§]U.C. Berkeley, [¶]Intel CA, [|]Carnegie Mellon University, [**]Flatiron Institute

presented by Mateusz Bysiek, School of Computing, Tokyo Institute of Technology

in HPC2018 class, 20 December 2018

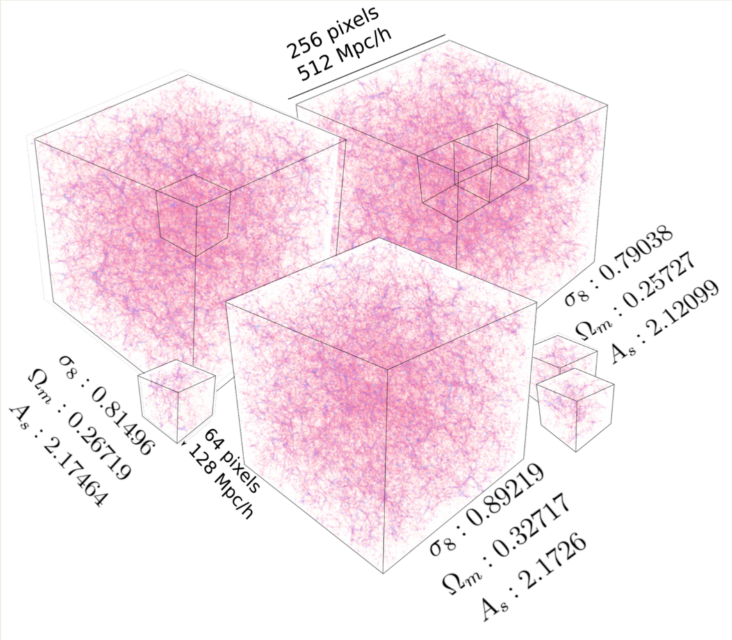
What is the nature of dark energy?

- Unknown force that drives accelerated expansion of the universe
- Impossible to measure directly – only through effects on observable universe
  - *Distribution of matter is an effect of interplay of gravity and dark energy*



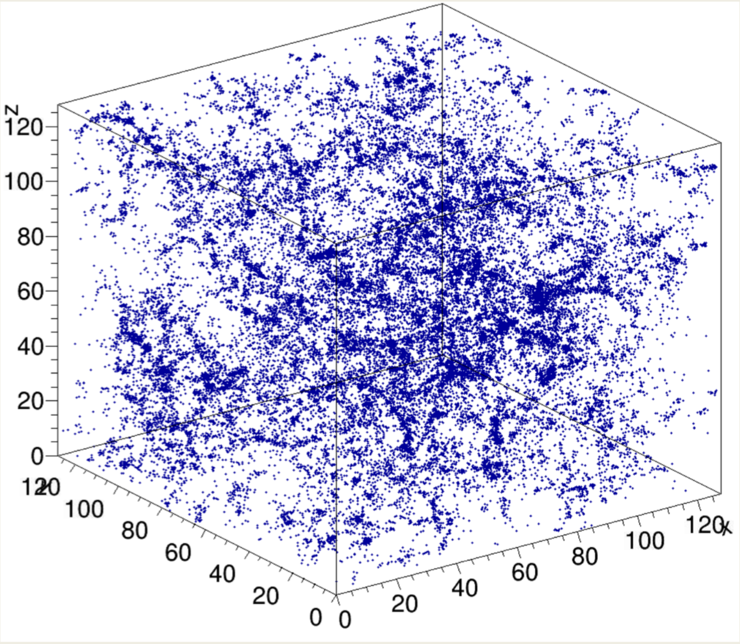
Why use deep learning for cosmology?

- Matter distribution is typically characterised using *few* selected features
- Deep learning networks could capture *all* features
- Simulation-generated matter distributions provide training data



Challenges for deep learning in science

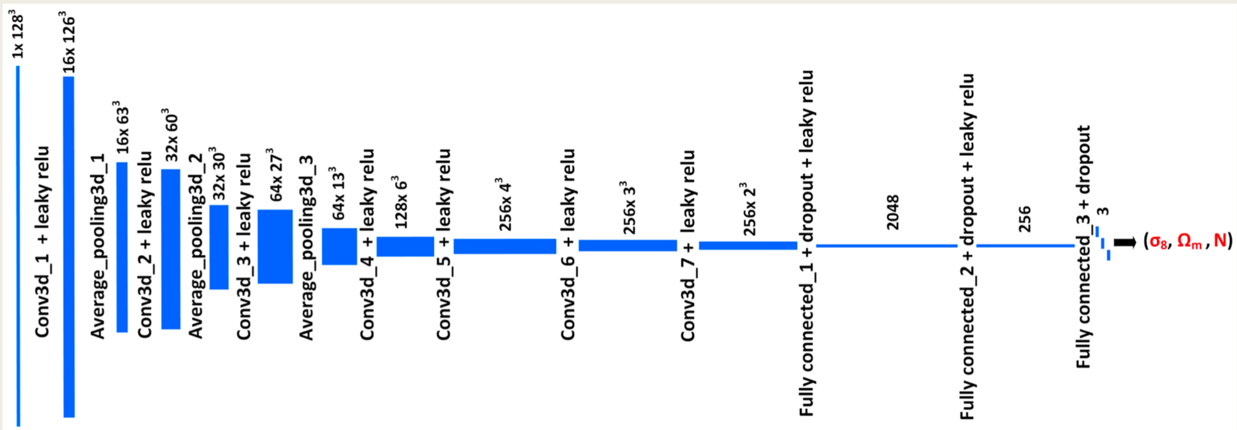
- Scientific data often complex (3+ dimensions and many channels)
- Measured in tera- or petabytes
- Efficient processing at scale essential for relevance of deep learning in science



Modifications in CosmoFlow

- Adapt existing deep learning network to scalable architecture
- Extra convolution layer and average pooling layer (near input) to cope with input size increase

- State of the art work predicted 2 parameters at problem size 64^3 –it did not scale
- Predict 3 cosmological parameters at problem size of 128^3



The predicted cosmological parameters

- Ω_M – Matter density parameter, i.e. proportion of matter in the universe
- σ_8 – The present root-mean-square matter fluctuation averages, i.e. amplitude of mass fluctuations in the universe, over a sphere of radius $8h^{-1}$ Mpc
- n_s – scalar spectral index of the spatial curvature of a comoving slicing of space-time, i.e. how density fluctuations vary with scale

Modifications in CosmoFlow, contd.

- Use batch size of 1 per MPI rank
- Adam Optimizer
- Polynomial learning rate decay schedule

Optimizations: node-level

- Change number of output channels to aid vectorization
- Arrays are blocked by channels in forward propagation
- 3 innermost loops are completely unrolled and vectorized with AVX512
- Similar strategy for backward propagation

Require: $SRC \in \mathbb{R}^{ICB \times ID \times IH \times IW \times 16}$
Require: $DST \in \mathbb{R}^{OCB \times OD \times OH \times OW \times 16}$
Require: $W \in \mathbb{R}^{OCB \times ICB \times KD \times KH \times KW \times 16 \times 16}$
for $ocb = 1 \cdots OCB$ **do** // Output channel block
for $icb = 1 \cdots ICB$ **do** // Input channel block
for $od = 1 \cdots OD$ **do** // Output depth
for $oh = 1 \cdots OH$ **do** // Output height
for $owb = 1 \cdots OWB$ **do** // Output width block
  $d \leftarrow DST[ocb, od, oh, 28owb, 0]$
  for $kd = 1 \cdots KD$ **do** // Kernel depth
  for $kh = 1 \cdots KH$ **do** // Kernel height
  for $kw = 1 \cdots KW$ **do** // Kernel width
    $s \leftarrow SRC[icb, od + kd, oh + kh, 28owb + kw, 0]$
    $w \leftarrow W[ocb, icb, kd, kh, kw, 0, 0]$
    for $ow = 1 \cdots 28$ **do** // Output width
    for $oc = 1 \cdots 16$ **do** // Output channel
    for $ic = 1 \cdots 16$ **do** // Input channel
      $d[16ow + oc] \leftarrow w[16ic + oc]s[16ow + ic]$

Optimizations: scaling

- Burst buffer: CPE ML plugin
- Reduce straggler effect of Synchronous Stochastic Gradient Descent – hide timing imbalances with non-blocking MPI
- No central/unique master node
- batch size of 1 per MPI rank Remove batch-norm layers to aid scaling

Require: N = total number of epochs
Require: n = total number of training samples
Require: k = number of MPI ranks

for epoch $= 1 \cdots N$ **do**
    for step $= 1 \cdots n/k$ **do**
        $g_{step} \leftarrow$compute_gradients(local_batch$_{step}$)
        $G_{step} \leftarrow$mc.gradients(g_{step})
        $loss_{step} \leftarrow$apply_gradients(G_{step})

Cori: Cray XC40 at NERSC

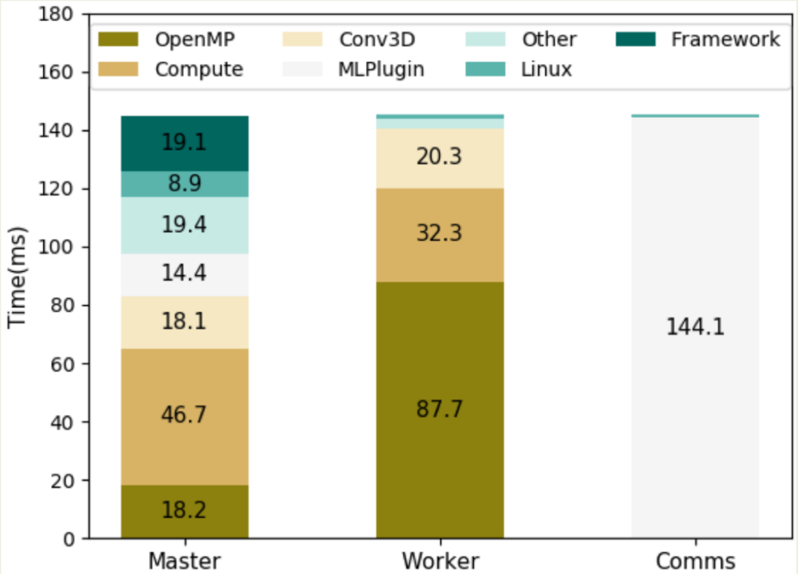
- 2004 nodes with Intel Xeon Phi E5-2698 v3
  9688 nodes with Intel Xeon Phi 7250
  - *68 cores, 16GB memory*
  - *32/32KB instr./data L1 cache*
  - *2D mesh network*
- 96GB of DDR4-2400 DRAM per node
- 288 nodes of Cray DataWarp (i.e. burst buffer)
  - *2x 3.2TB SSD (~1.8PB total)*
  - *1.7TB/sec*
- Sonexion 2000 Lustre: 248 OSTs, 10168 disks, 30 PB

Piz Daint: Cray XC50 at CSCS

- 1431 nodes of 2x Intel Xeon E5-2695 v4
- 5320 nodes of Intel Xeon E5-2690 + NVIDIA P100 PCIe GPU
- Sonexion 3000 Lustre: 40 OSTs, 6.2 PB

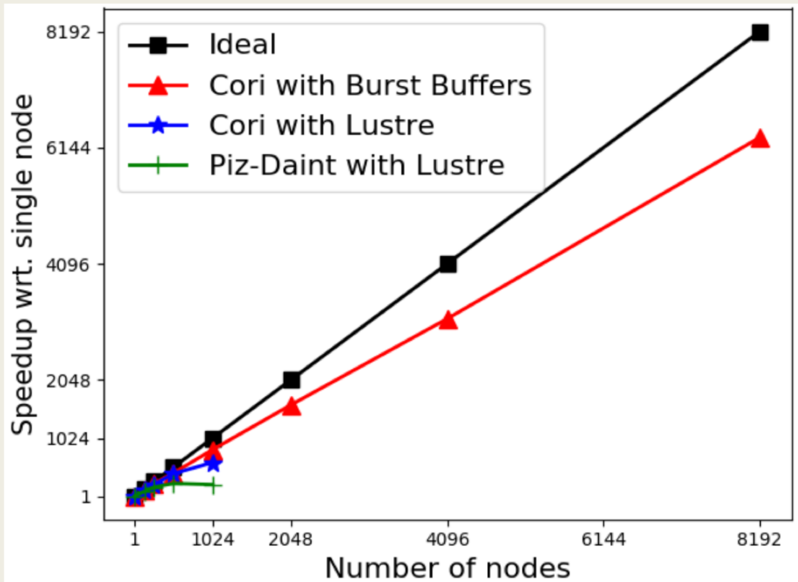
Single node performance results

- Data-parallel training is ideal due to volume of the data
- Times:
  OpenMP spin time and overhead,
  non-convolutional compute time,
  3D convolutions,
  CPE ML Plugin,
  other time,
  Linux kernel time,
  TensorFlow framework time

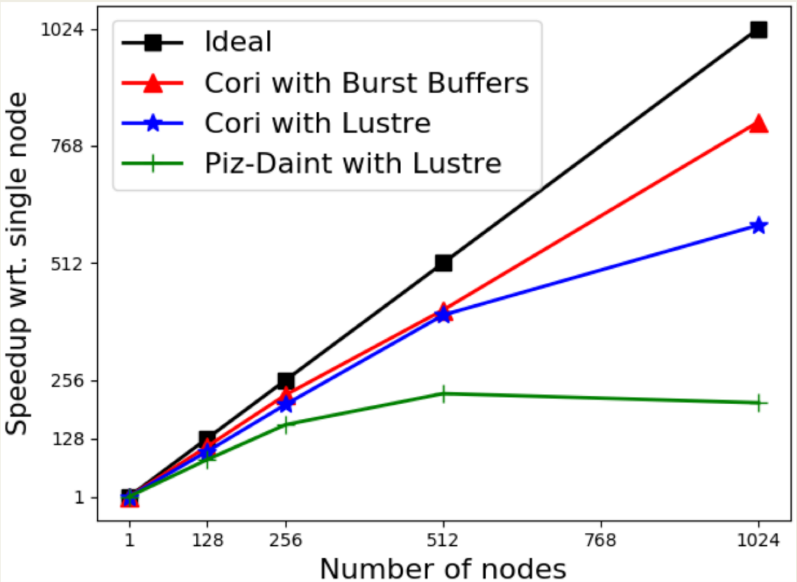


Scalability

- fully-synchronous training on 8192 nodes of Cori
- 77% parallel performance
- sustained 3.5Pflop/s (single precision)

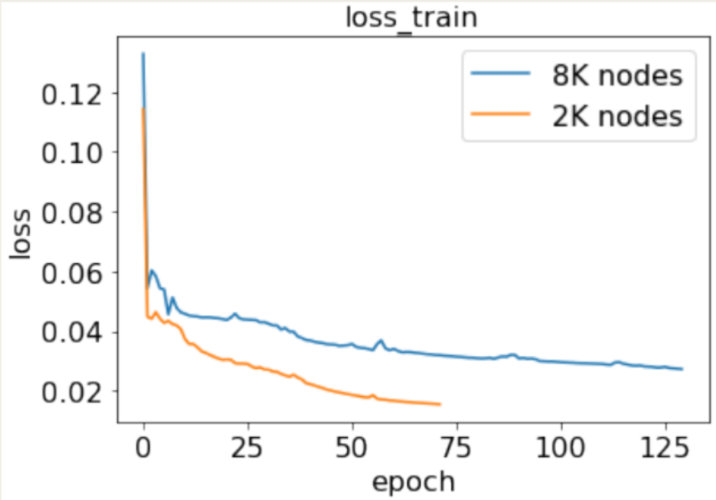


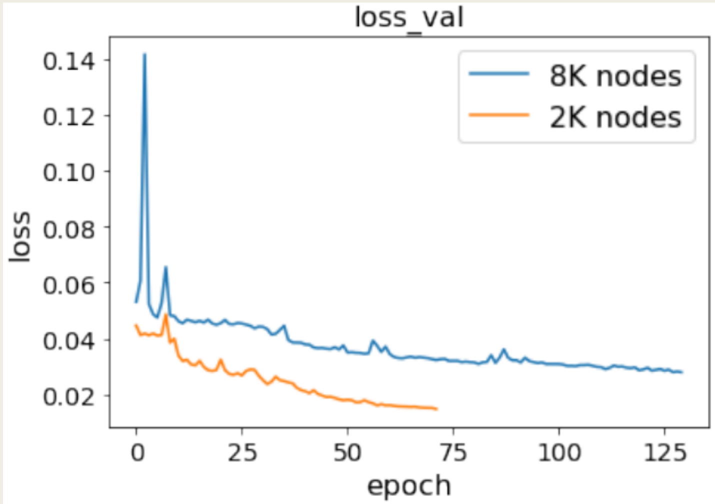
Scalability, contd.



- Lustre-based approaches fail to scale beyond 512~1024 nodes
- Hard to scale beyond anyway without careful optimizer tuning
- IO variability
- CPE ML plugin is MPI-based, as opposed to default inefficient implementation in TensorFlow

Loss function





Physical results