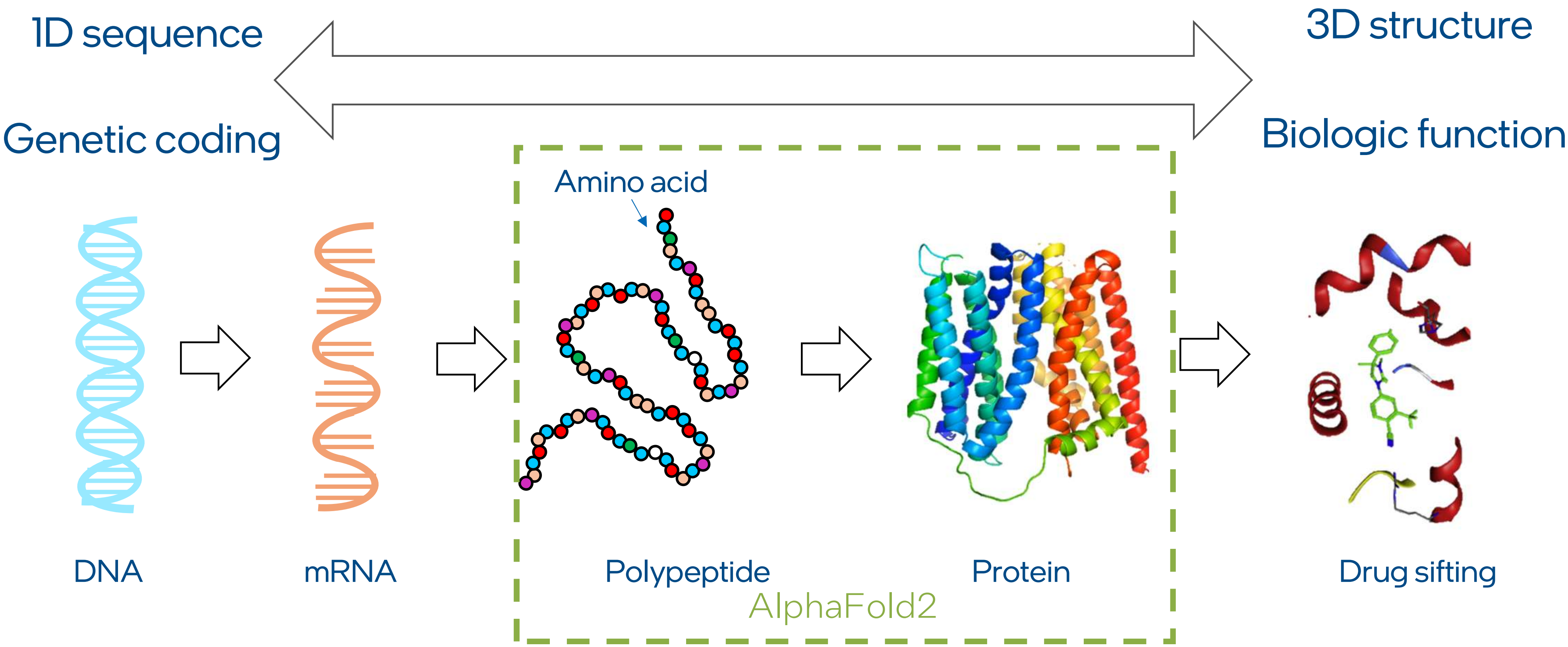
End-to-End Optimization of AlphaFold2 on Intel Architecture

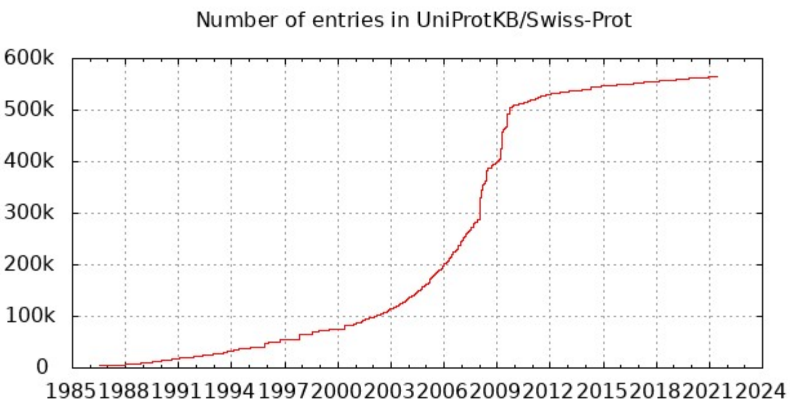
Luke Ge, Intel AI Technical Solution Specialist
Luke.ge@intel.com

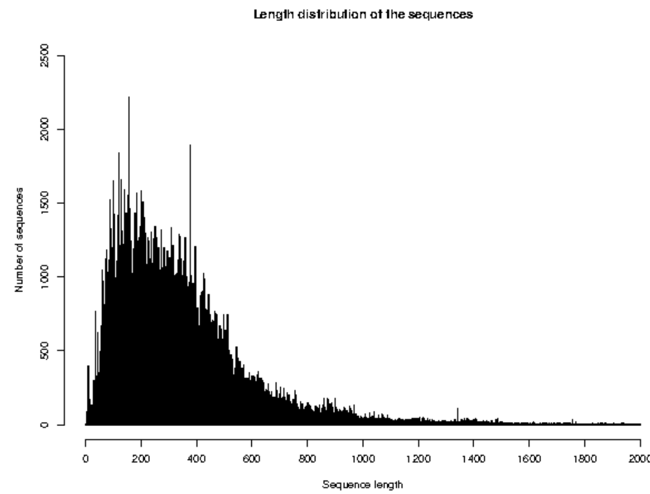
intel.

From DNA to Protein Structure



AI Needed for Processing Huge Sequence Data





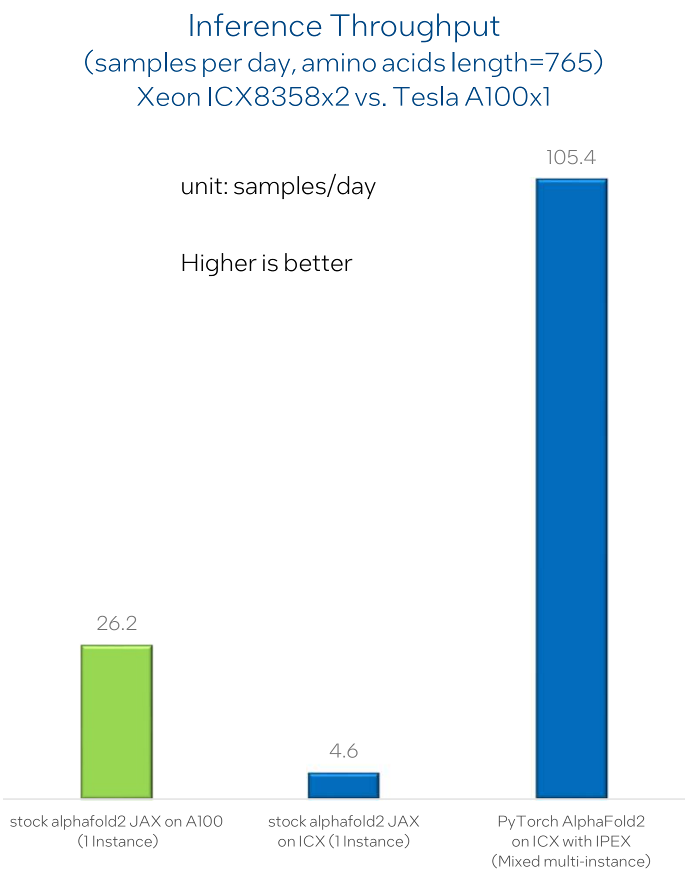
- X-Ray/ Cryo-EM: months ~ years per sequence
- 566,996 sequences in SwissProt wait for analysis
- Traditional methods (CADD) cannot handle huge data reliably
- More powerful protein folding tools are needed

- DeepMind AlphaFold2 is a fast folding algorithm
- Two areas to be improved：
- - GPU Memory limits the sequence length （<1000aa)
- - Original code not optimized for CPU

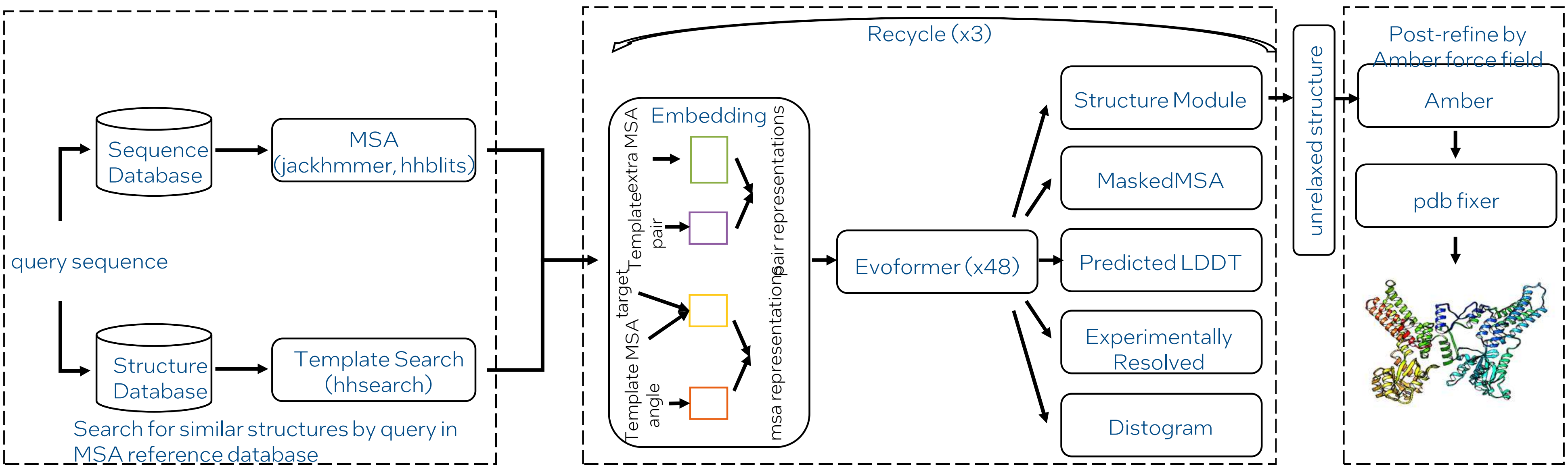
UniProt statistics: https://web.expasy.org/docs/relnotes/relstat.html

AlphaFold2 Inference Performance Claim

- We optimized AlphaFold2 inference pipeline on Xeon ICX server (precision=FP32)
  - **4.01x** throughput on ICX8358 2S vs. single Nvidia A100 GPU card
  - we improve AlphaFold2 performance by **23.11x**: vs stock AlphaFold2
    - 4.56x pipeline throughput by multi-instance with PMEM
    - 5.05x pipeline throughput by model and op optimizations



AlphaFold2 Pipeline Overview



Preprocessing

Get structure fragments templates from similar known protein sequences as references

DL Model Inference

Transformers to predict the graph representations of 3D structures

Postprocessing

Refine the 3D structure parameters

Compute time % on Xeon SP (stock AlphaFold2)

15.3% | 84.1% | 0.6%

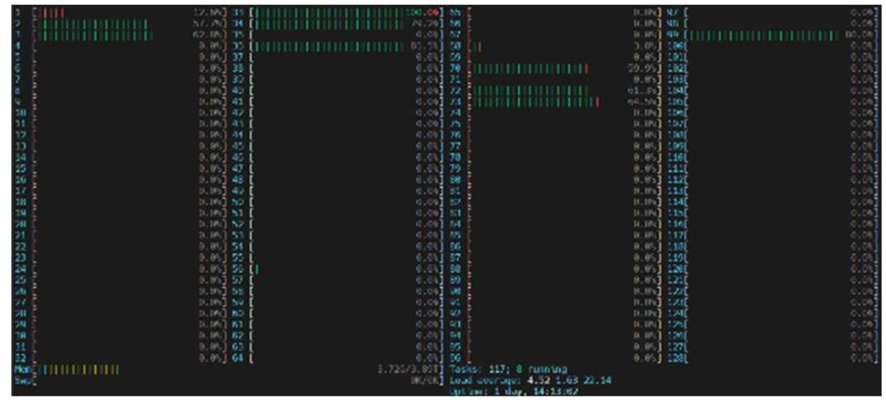
AVX-512 | oneAPI AI Analytics | AVX-512

High Throughput Optimization of AlphaFold2 Pre-processing

AVX512

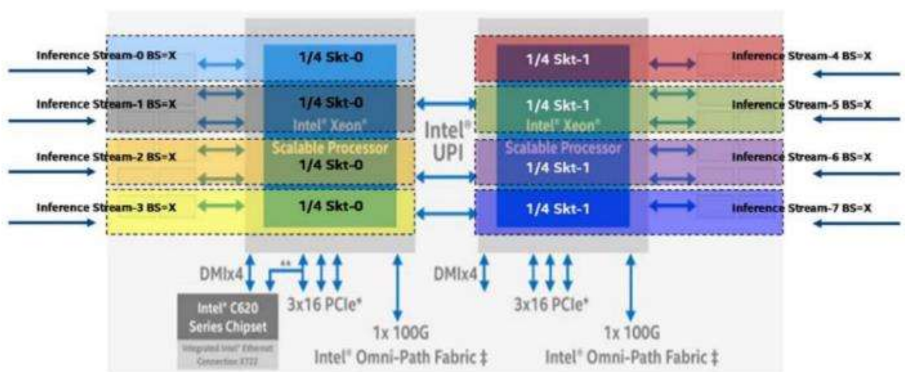
- Use AVX-512 to increase throughput (icc options)
- -O3 -no-prec-div -march=icelake-server

Before

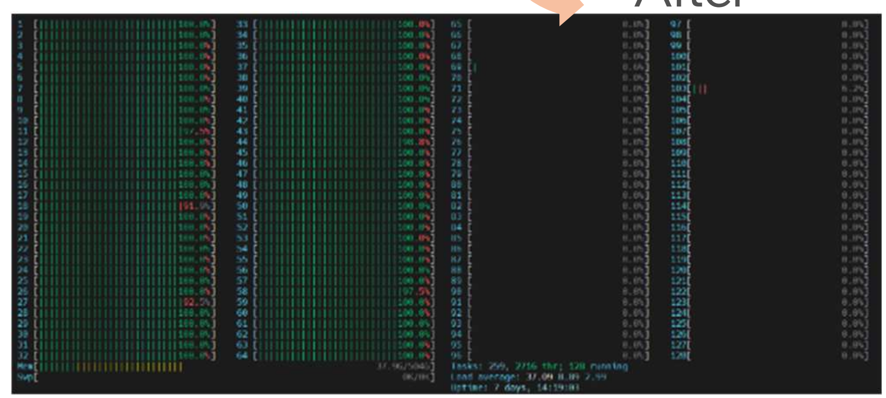


multi-stream

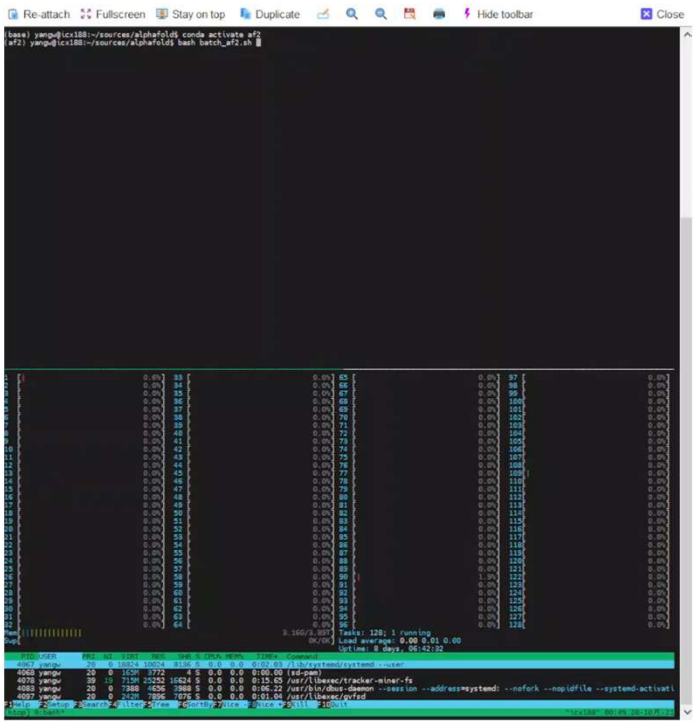
- numactl –C $core_ids -m $socket_id $command
- mpirun –np $nranks –map-by ppr:$core_per_instance:socket:pe=$total_number_core_per_socket $command



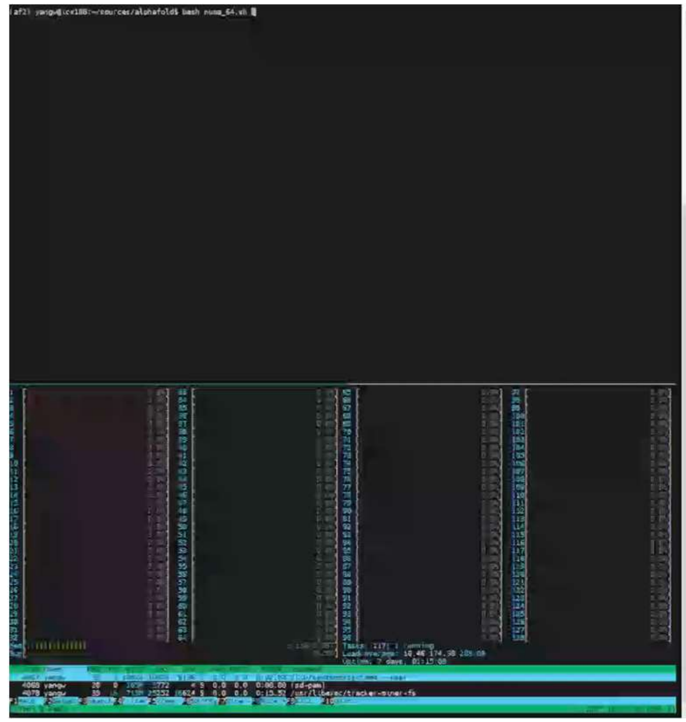
After



Optimized Pre-processing



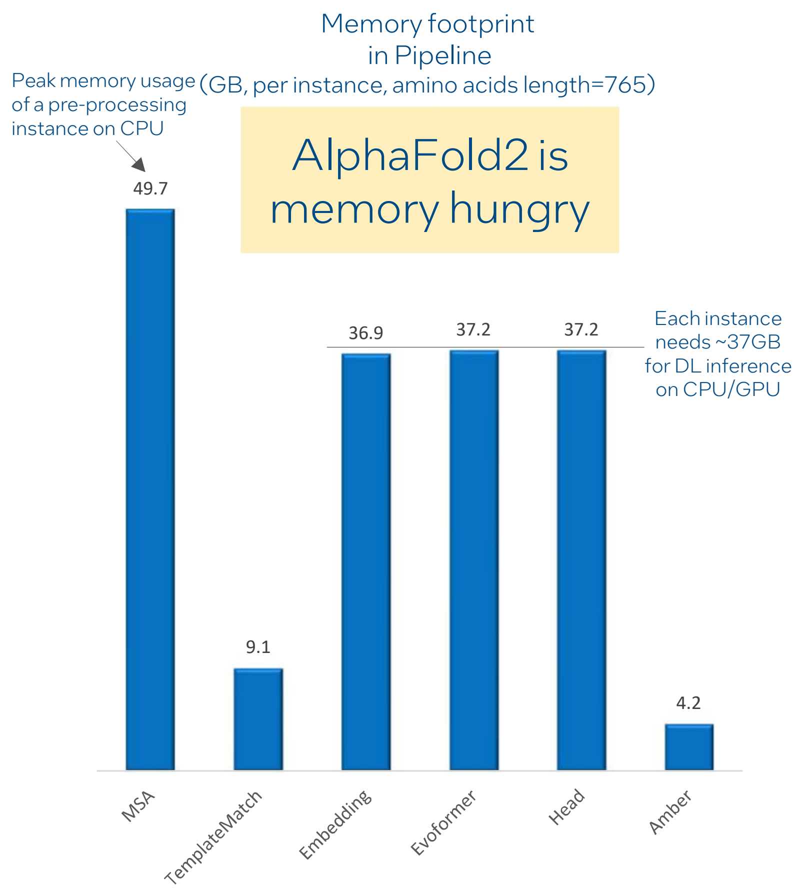
Before: 1 instances * 4 seqs

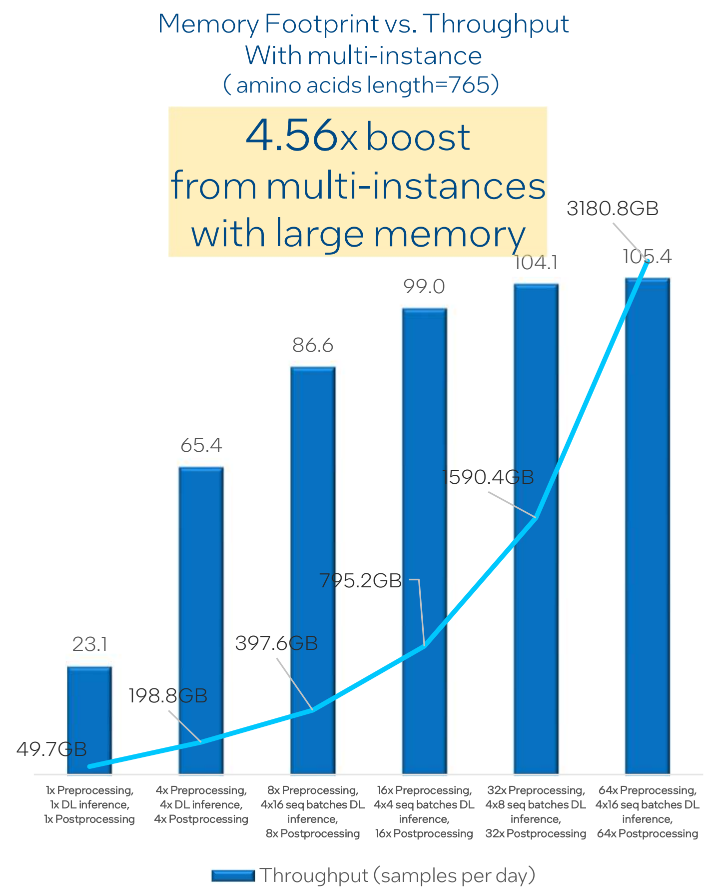


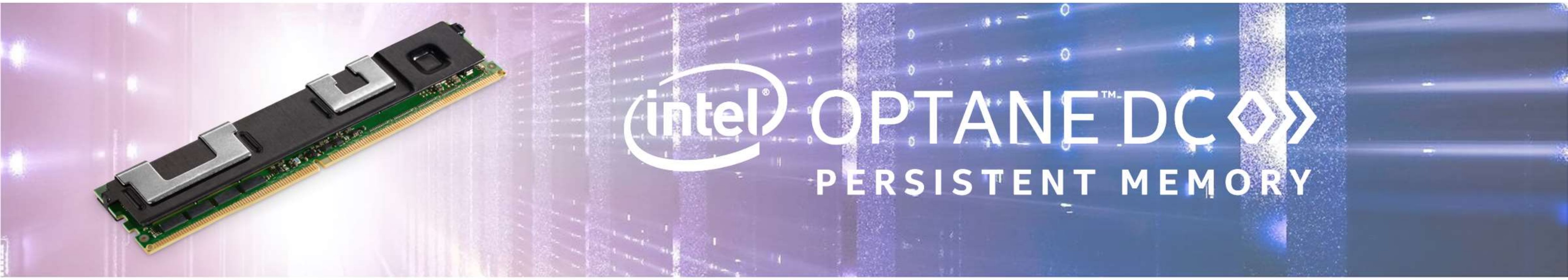
After: 64 instances * 1 seq

Test data: mmcif_3geh.fa , seq length = 765
Compute nodes: Intel® Xeon 8358x2, 512GB DDR4+4TB PMEM, 25GbE NIC
Ubuntu 20.04 server, Python 3.9.7, hmmer 3.3.2

Multi-instance Challenge: Memory







BREAKTHROUGH MEMORY INNOVATION

- AFFORDABLE ALTERNATIVE TO DRAM
- IMPROVE TCO
- ON-MODULE ENCRYPTION

INFRASTRUCTURE CONSOLIDATION

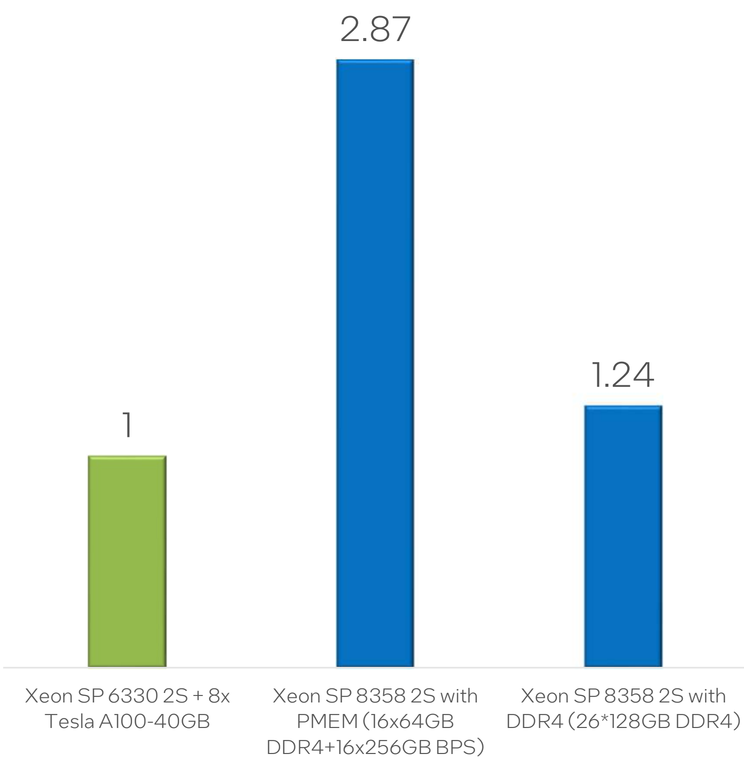
- INCREASE MEMORY SIZE (128/256/512GB)
- CONSOLIDATE WORKLOADS
- SCALE UP TO SCALE OUT

PERFORMANCE AND PERSISTENCE

- BREAK IO BOTTLENECKS
- FASTER RECOVERY
- HIGH SPEED STORAGE

AlphaFold2 TCO: Xeon vs. Nvidia Tesla A100

Normalized Performance per Dollar (Amino acids length=765)



TCO Analysis Configuration

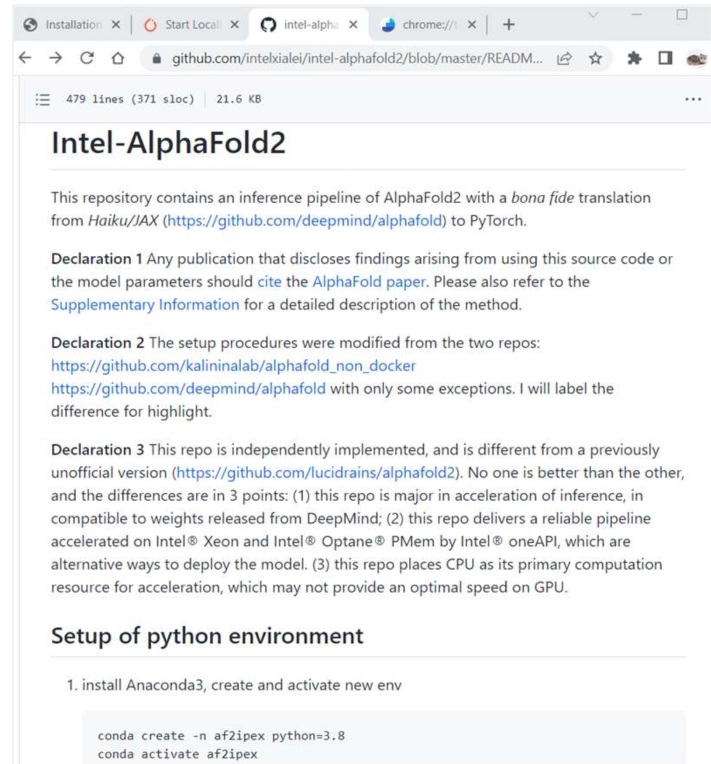
	Nvidia Telsa A100 with Xeon SP	Xeon SP with DDR4+PMEM	Xeon SP with DDR4
Model	Inspur NF5468M6-P	Inspur NF5280M6	Inspur NF5280M6
Processor	3rd Gen Intel® Xeon® Scalable processors 6330 x2	3rd Gen Intel® Xeon® Scalable processors 8358 x2	3rd Gen Intel® Xeon® Scalable processors 8358 x2
GPU	Nvidia A100 PCIe Gen4 40 GB 250W 900-21001-0000000 x 8	-	-
Memory	32GB DDR4 3200MHz RDIMM x16	32GB DDR4 3200MHz RDIMM x16 and 256GB 256GB Optane Intel NMB1XXD256GPSU4 DCPMM Barlow Pass x16	128GB DDR4 3200MHz RDIMM x32
I/O Expansion	Raid Cntrlr - Trinity Dunes RAID Adapter Intel RSP3TD160F x1	Raid Cntrlr - Trinity Dunes RAID Adapter Intel RSP3TD160F x1	Raid Cntrlr - Trinity Dunes RAID Adapter Intel RSP3TD160F x1
Storage	Solidigm Youngsville Refresh SSDSC2KB038T801 S4510 Series x1	Solidigm Youngsville Refresh SSDSC2KB038T801 S4510 Series x1	Solidigm Youngsville Refresh SSDSC2KB038T801 S4510 Series x1
Network	SND I350-AM2 RJ45 Dual Port PCI-E4X_1KM x1	SND I350-AM2 RJ45 Dual Port PCI-E4X_1KM x1	SND I350-AM2 RJ45 Dual Port PCI-E4X_1KM x1
PSU	2+2 redundant, 4x 3000W 80Plus Platinum hot-swap PSU	(1+1) 1200W Platinum	(1+1) 1200W Platinum
Reference Price (USD)	113,150	19,832	48,650

AlphaFold2 DL Model Inference Optimization

Migrate AlphaFold2 to PyTorch

- Stock AlphaFold2 models are based on JAX
  - leverage google's XLA compile
  - not optimized on CPU yet
- Migrate to PyTorch
  - models (embedding, Evoformer, heads) manually rewritten by PyTorch
  - model correctness validated: <1.63% error in output
  - leverage PyTorch's JIT optimization

Model inference latency Before/After migration
(seconds, single instance , amino acids length=765)



Intel-AlphaFold2

This repository contains an inference pipeline of AlphaFold2 with a *bona fide* translation from *Haiku/JAX* (https://github.com/deepmind/alphafold) to PyTorch.

Declaration 1 Any publication that discloses findings arising from using this source code or the model parameters should cite the AlphaFold paper. Please also refer to the Supplementary Information for a detailed description of the method.

Declaration 2 The setup procedures were modified from the two repos:
https://github.com/kalininalab/alphafold_non_docker
https://github.com/deepmind/alphafold with only some exceptions. I will label the difference for highlight.

Declaration 3 This repo is independently implemented, and is different from a previously unofficial version (https://github.com/lucidrains/alphafold2). No one is better than the other, and the differences are in 3 points: (1) this repo is major in acceleration of inference, in compatible to weights released from DeepMind; (2) this repo delivers a reliable pipeline accelerated on Intel® Xeon and Intel® Optane® PMem by Intel® oneAPI, which are alternative ways to deploy the model. (3) this repo places CPU as its primary computation resource for acceleration, which may not provide an optimal speed on GPU.

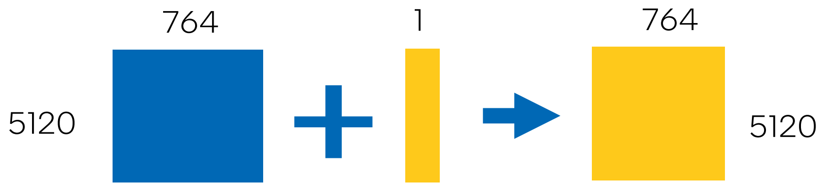
Setup of python environment

1. install Anaconda3, create and activate new env

```
conda create -n af2ipex python=3.8
conda activate af2ipex
```

- Intel internal private repo at this moment
- Plan to be open-source in Intel Model Zoo

Memory Bottleneck in Attention Module (ExtraMsaStack)




```
self.attention(q_data,m_data,bias,nonbatched_bias)
```

5120x764x64
(930MB)

5120x1x1x64
(1.25MB)

broadcast

5120x1x764x64
(930MB)

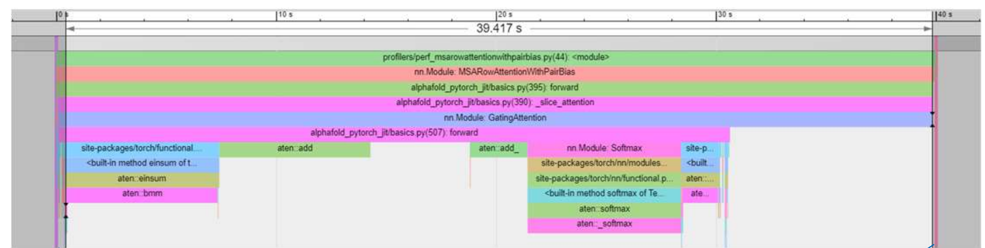
320x1x1x64
(80KB)

broadcast

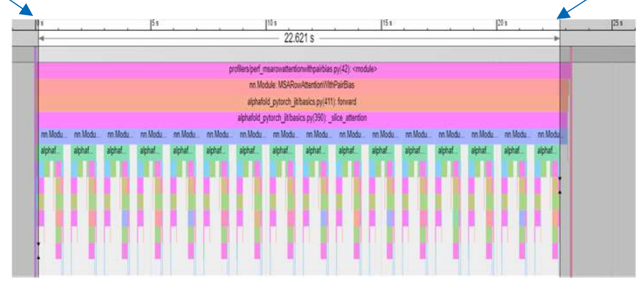
320x1x764x64
(59.69MB)

```
def _slice_attention(self,q_data,m_data,bias,nonbatched_bias=torch.Tensor()):
  ### avoiding huge memory cost
  ### threhold is ajustable
  threhold = 1000
  unit = 320 # unit is ajustable
  if q_data.size()[0] > threhold:
    res = torch.ones_like(q_data)
    for i in range(q_data.size()[0] // unit):
      q_sub_data = q_data[unit*i:unit*(i+1)]
      m_sub_data = m_data[unit*i:unit*(i+1)]
      bias_sub = bias[0:unit]
      res[unit*i:unit*(i+1)] = self.attention(q_sub_data,m_sub_data,bias_sub,nonbatched_bias)
      #print("slice_attention_wrapper finish exec total {} cycles".format(q_data.size()[0] // unit))
    return res
  else:
    return self.attention(q_data,m_data,bias,nonbatched_bias)
```

Original



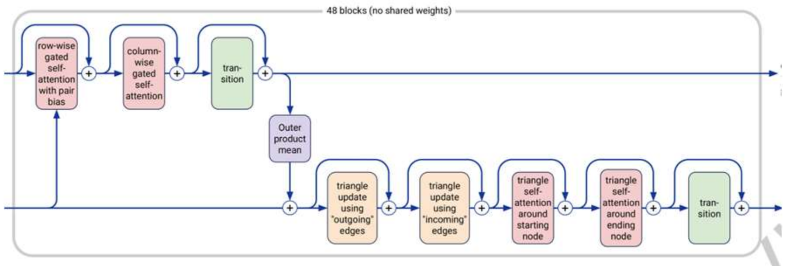
Optimized



1.74x boost in attention unit
21.51% boost in Embedding model

Operation Fusion in Evoformer Module

An Evoformer Block



Name	Self CPU %	CPU total %
aten::einsum	0.07%	42.23%
aten::bmm	26.73%	37.51%
quantized::linear_dynamic	20.15%	20.16%
aten::clone	0.08%	16.45%
aten::copy_	16.35%	16.35%
aten::add	12.62%	12.62%
aten::softmax	0.00%	8.76%
aten::_softmax	8.76%	8.76%

Large Tenors
(~140MB per tensor)

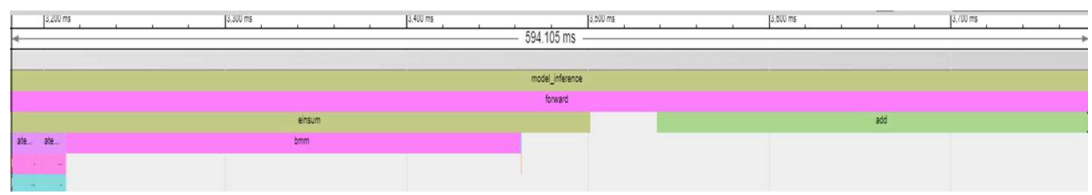
```
= torch.einsum('bqhc,bkhc->bhqk', q, k) + bias
```

Batch Matrix Multiplication(BMM)

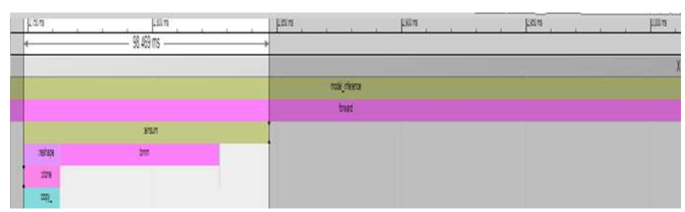
Add

- Fuse einsum and add by using oneAPI
- Fusion is available with oneDNN 2.6

Original



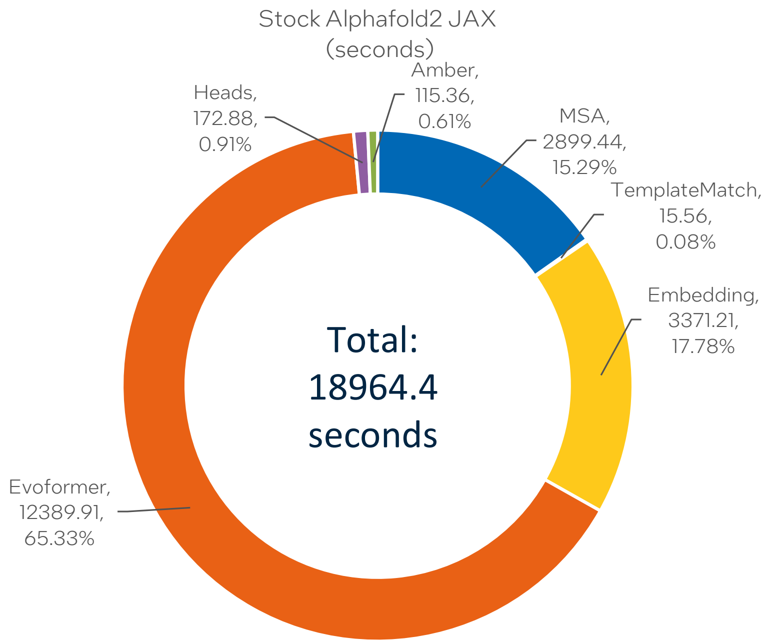
Optimized



6.03x boost in Einsum+Add Ops unit test
10.02% boost in Evoformer model
5.0% boost in AlphaFold2 pipeline

AlphaFold2 Pipeline Inference Latency (Single Instance)

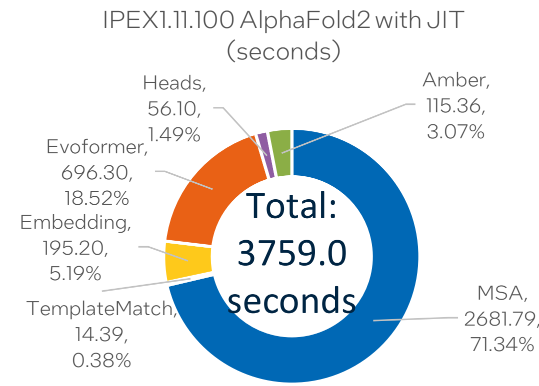
Before Optimization



Preprocessing: 15.37%
DL Inference: 84.02%
Postprocessing: 0.61%

5.05x boost

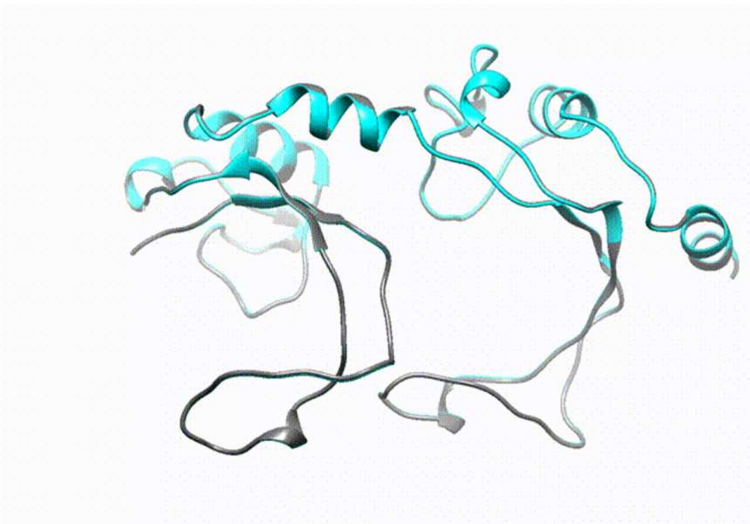
After Optimization



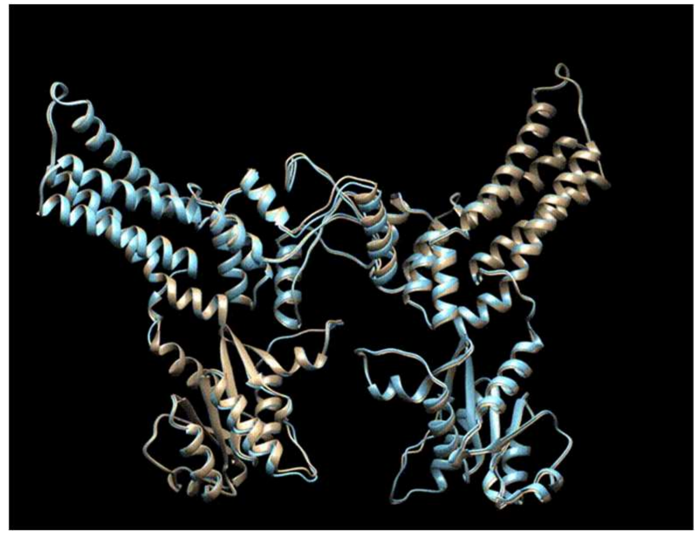
Preprocessing: 71.72%
DL Inference: 25.21%
Postprocessing: 3.07%

Sample length = 765 amino acids
Pre-processing = MSA + TemplateMatch
DL inference = Embedding + Evoformer + Heads
Post-processing = Amber

Accuracy Comparison: Optimized vs. Original



- Input: 206aa
- 2S-ICX8358



- Input: 765aa
- 2S-ICX8358

- MatchMaker
- Gold: original
- Blue: optimized

Summary

- Intel optimized AlphaFold2 inference on Xeon platform
  - Memory: larger memory space (TBs), Optane Persistent Memory
  - Compute: AVX-512 (advanced vector extension), AMX (advanced matrix extension)
  - Communication: easy scaling on cluster
  - oneAPI: free AI software stack
- CPU is a better platform for AlphaFold2 inference
  - Intel Xeon CPU is 4.01x faster than Nvidia A100, 2.87x better Perf/$
  - CPU processes longer sequences than GPU
- The optimized code will be publicly available

AlphaFold2 Inference : Recommended System Configuration

Key hardware configuration

Processor	2x 3rd Gen Intel® Xeon® Scalable processors 8358 or higher bin
DDR4 Memory	16x 32GB DDR4 3200MHz RDIMM
PMEM Memory	16x 256GB PMEM (BPS)
Storage	Intel® Optane SSD 3.8TB+

Key Software configuration

Operation System	CentOS 7.x/Ubuntu 20.04 or later version
AI Framework	PyTorch 1.11.0 Intel PyTorch Extension 1.11.0
Library and compiler	Intel® oneAPI Base Toolkit 2022.1.2 or later version
Python	Intel® distribution for Python integrated in Intel® oneAPI Base Toolkit

Luke.ge@intel.com

intel®