Stanford Graph Learning Workshop 2022

# Accelerating PyG with Intel CPUs

Ke Ding, Principal Engineer

Intel – SATG AIA

Sep 2022

intel.

# Outline

- Intel AI Software Stack
- PyG Optimization for Intel CPUs
  - Message Passing Analysis
  - Kernel Optimization: scatter_add
  - Kernel Optimization: spmm_reduce
  - Node Sampling Optimization
- What's Next

# AI Software Ecosystem and Intel Tools

Engineer Data → Create Machine Learning & Deep Learning Models → Deploy

| Container Repository | MLOps | Developer Sandbox |
| --- | --- | --- |
| oneContainer | Cnvrg.io | DevCloud |

Accelerate End to End Data Science and AI — AI Analytics Toolkit

Connect AI to Big Data — Spark Optimize Analytics Package — Domain Toolkit: NLP, RecSys, TLT, Time Series, PPML — BigDL (Analytics Zoo)

## Data Analytics at Scale

MODIN, NumPy, pandas, SciPy, omni·sci, Numba

## Optimized Frameworks and Middleware

TensorFlow, JAX, PyTorch, PyG, DGL Deep Graph Library, ONNX, PaddlePaddle, mxnet, scikit learn, dmlc XGBoost, DeepSpeed

## Optimize and Deploy Models

| Automate Model Tuning AutoML | Write Once Deploy Anywhere | Automate Low-Precision Optimization |
| --- | --- | --- |
| SigOpt | OpenVINO Toolkit | Neural Compressor (INC) |

oneAPI: oneDAL, oneDNN, oneCCL, oneMKL

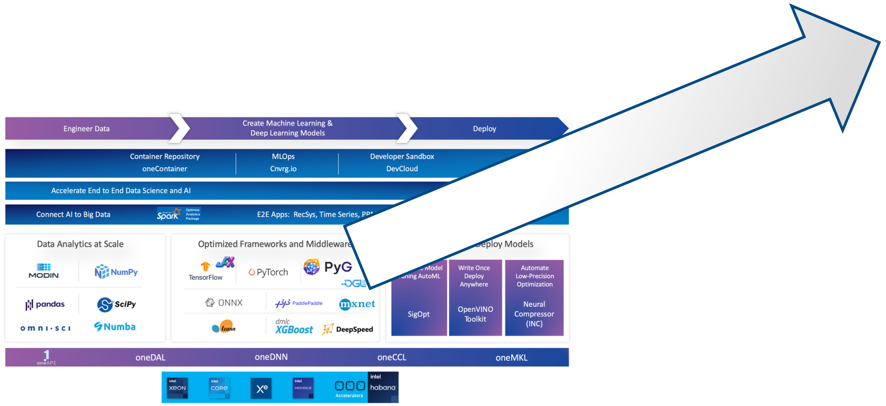
intel XEON, intel CORE, Xe, intel MOVIDIUS, Accelerators, intel habana

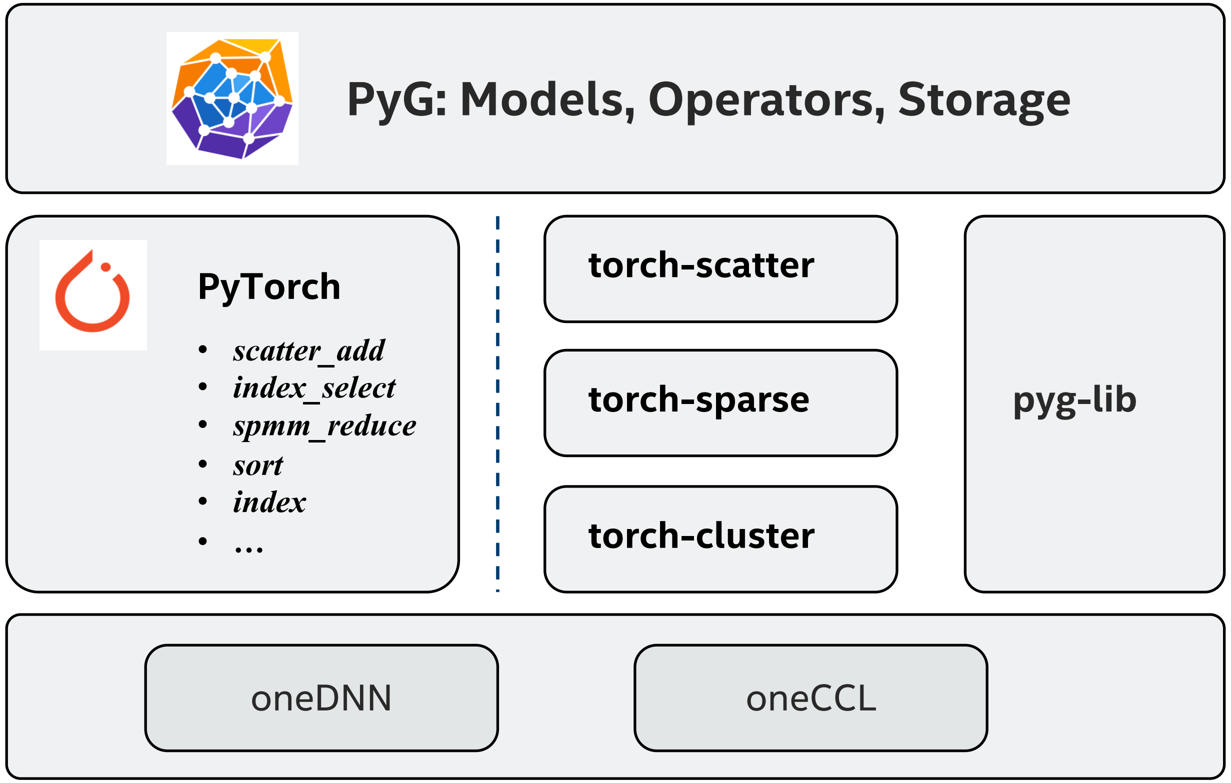


* Other names and brands may be claimed as the property of others

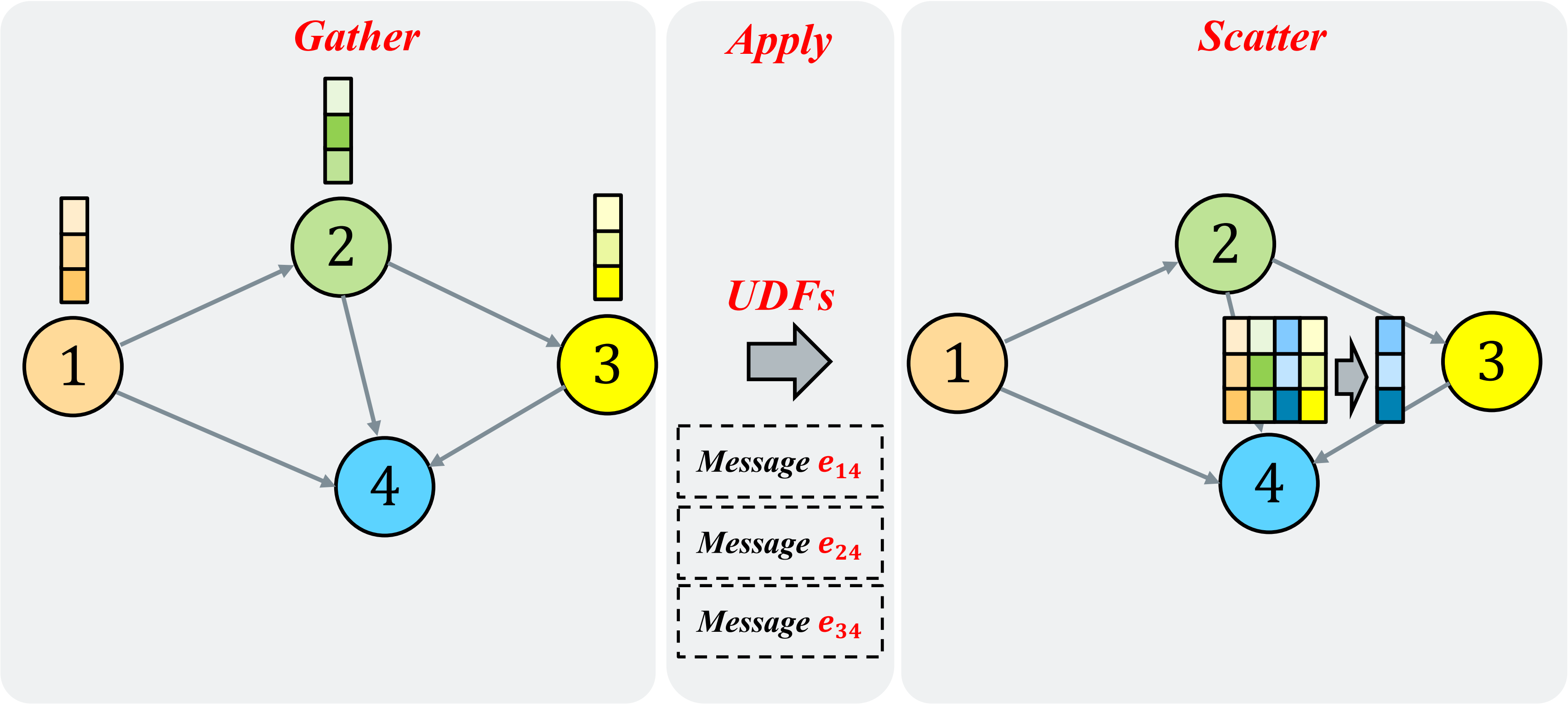
# PyG Overview on Intel Platforms

- Open source upstream first
- Inference and training
- Abstract perf primitives into oneDNN





# Message Passing Paradigm - GAS




- *Graph Conv (GCN)*
- *Graph Attention (GAT)*
- *SAGEConv*
- *GINConv*
- *EdgeConv*
- *PNAConv*
- *RGCN*

# Message Passing Profiling

***Case I:*** EdgeIndex in COO.

| Name | Self CPU % | Self CPU |
|---|---|---|
| aten::scatter_add_ | 49.00% | 37.797s |
| aten::index_select | 19.74% | 15.223s |
| aten::linear | 0.01% | 5.706ms |
| aten::addmm | 6.62% | 5.108s |
| aten::matmul | 0.00% | 2.339ms |
| aten::mm | 7.09% | 5.472s |
| aten::index | 5.89% | 4.544s |
| aten::fill_ | 3.59% | 2.768s |
| aten::zeros | 0.01% | 7.616ms |
| aten::zero_ | 0.00% | 2.728ms |
| aten::true_divide_ | 0.00% | 1.158ms |
| aten::div_ | 2.74% | 2.116s |
| aten::add_ | 1.36% | 1.046s |
| aten::copy_ | 1.30% | 1.005s |

***Profiling of SAGE+Reddit***

***Case II:*** EdgeIndex in CSR.

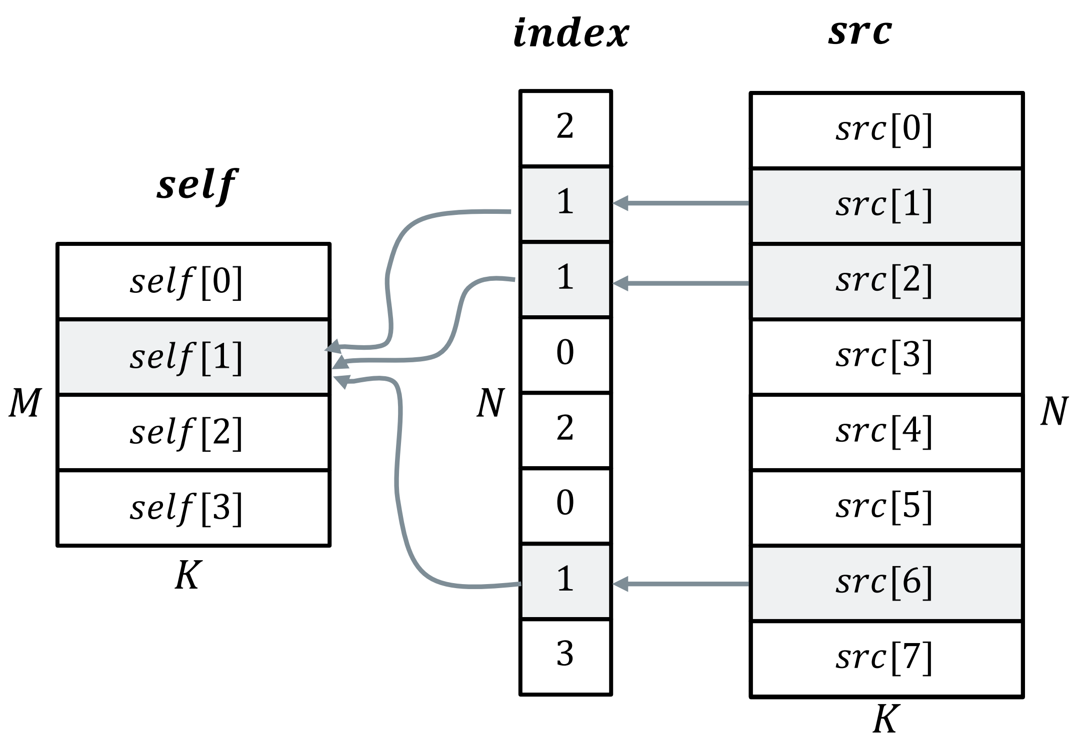
| Name | Self CPU % | Self CPU |
|---|---|---|
| torch_sparse::spmm_sum | 97.09% | 56.086s |
| aten::linear | 0.00% | 85.000us |
| aten::matmul | 0.00% | 57.000us |
| aten::mm | 1.38% | 795.201ms |
| aten::relu | 0.00% | 50.000us |
| aten::clamp_min | 0.76% | 440.384ms |
| aten::add_ | 0.57% | 327.801ms |
| aten::log_softmax | 0.00% | 23.000us |
| aten::_log_softmax | 0.10% | 55.480ms |
| aten::argmax | 0.09% | 53.149ms |
| aten::index | 0.01% | 5.771ms |
| aten::empty | 0.00% | 1.088ms |
| aten::t | 0.00% | 68.000us |
| aten::detach | 0.00% | 65.000us |

***Profiling of GCN+ogbn-products***

* Single batch inference (training hotspot slightly different).

# Kernel Optimization I: scatter_add

- ***scatter_add*** is hotspot when ***EdgeIndex*** stored in COO.
- It adds all values from the tensor ***src*** into ***self*** at the indices specified in the ***index*** tensor.



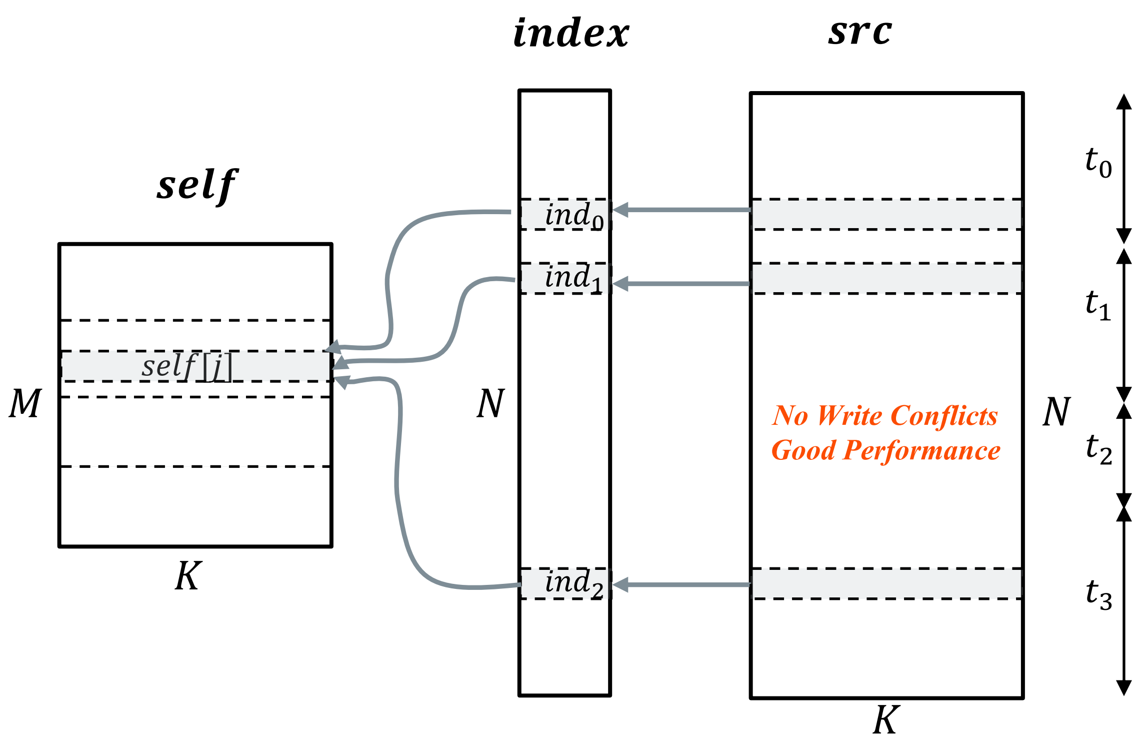

$$self[1] = src[1] + src[2] + src[6]$$

***Scatter_add Analysis:***

- Memory bandwidth bound.
- M refers to ***num_nodes***, N refers to ***num_edges***, K refers to ***num_features***.
- Input shape might be very large: for example, in ***SAGE+Reddit***, M = 135K, N = 447K, K = 256.
- Possible ***write conflicts*** since multiple threads may attempt to write the same address simultaneously.

# Kernel Optimization I: scatter_add

- ***scatter_add*** is hotspot when ***EdgeIndex*** stored in COO.
- It adds all values from the tensor ***src*** into ***self*** at the indices specified in the ***index*** tensor.

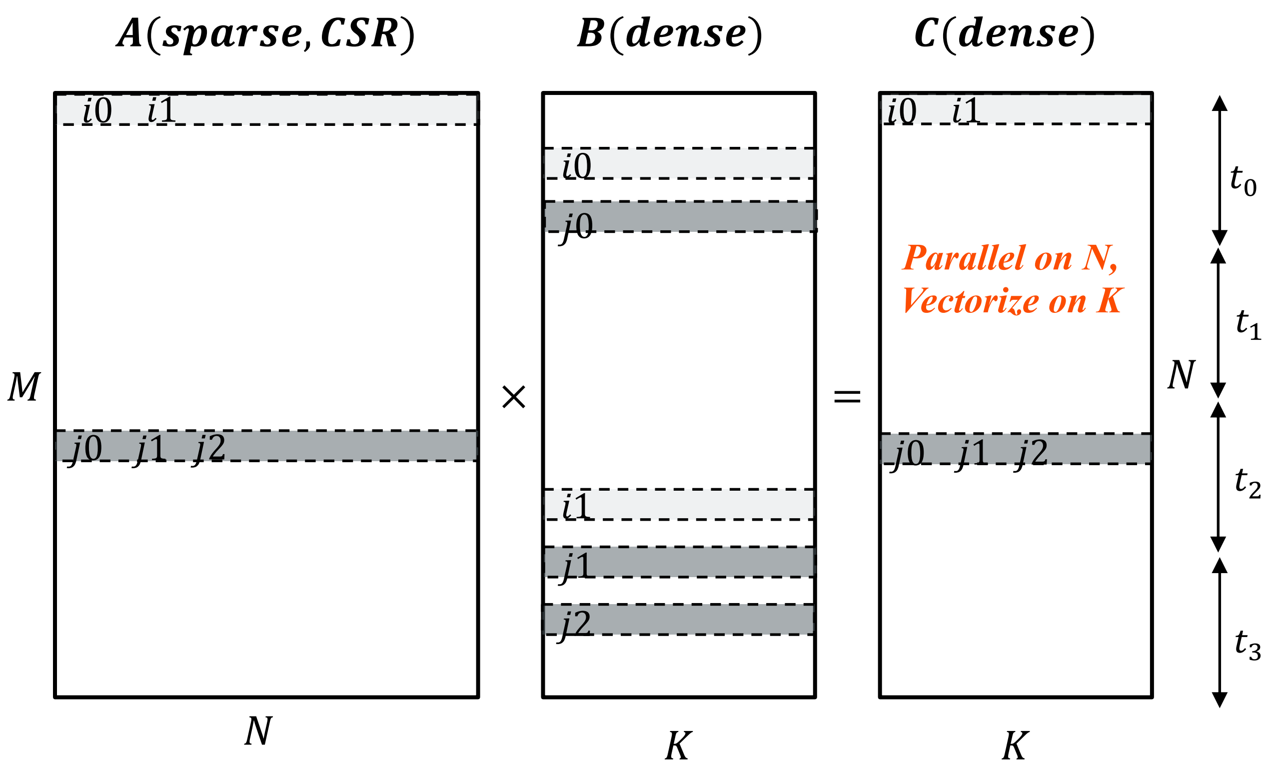



***Scatter_add Optimization:***

- ***Good performance***: parallel on ***outer dimension*** (M or N) and vectorize on ***inner dimension*** (K).
- Solve write conflicts though index sorting via ***radix sort.***
- ***SAGE+Reddit***: single socket inference:
  i. scatter_add time: ***5.9x*** speedup.
  ii. end to end time: ***1.7x*** speedup.
- The algorithm is equivalent to:
  i. convert COO to CSR (sorted indices encoded on CSR format);
  ii. do ***spmm_reduce***.

# Kernel Optimization II: spmm_reduce

- ***spmm_reduce*** is hotspot when ***EdgeIndex*** stored in CSR.
- API definition similar to SpMM, except that more reduction type required “max”, “mean”.

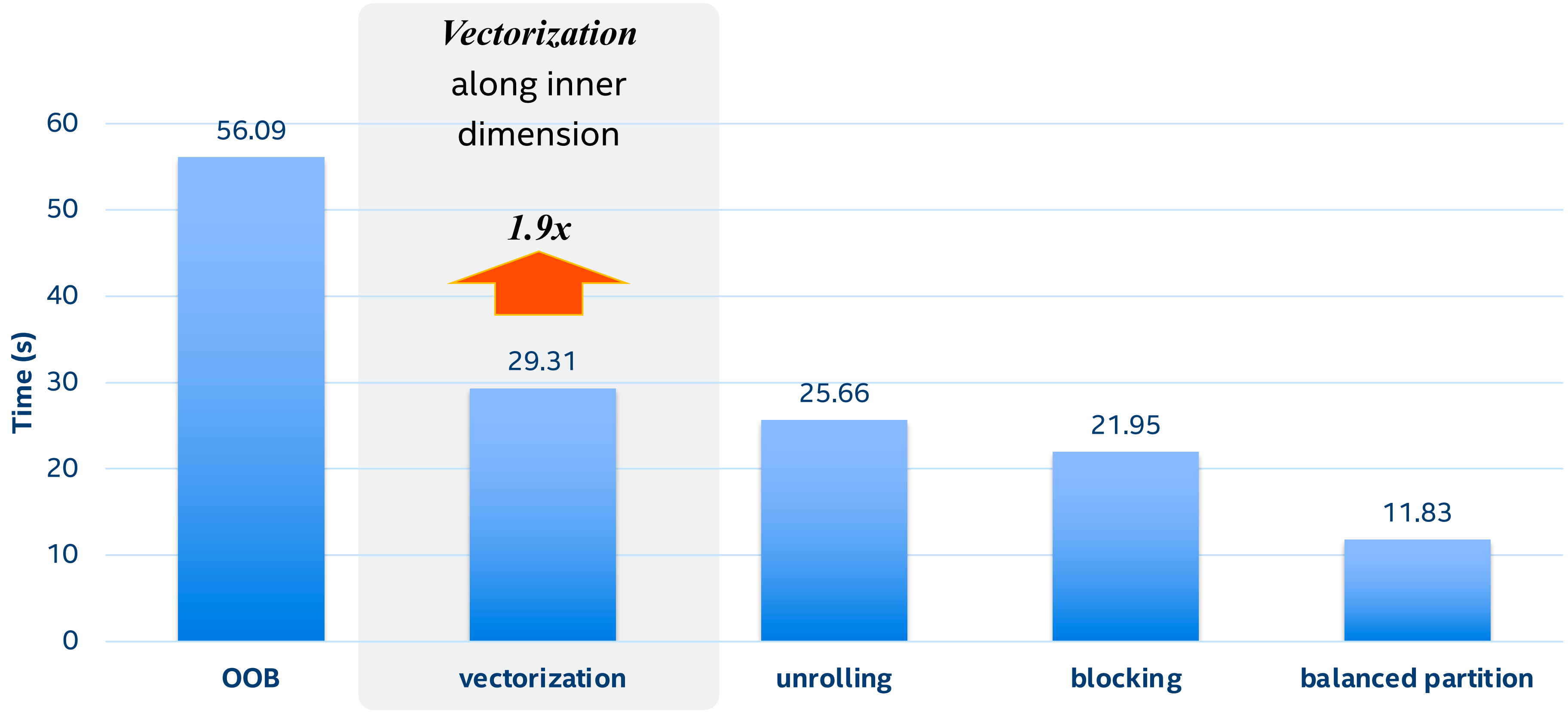



***Spmm Reduce Optimization:***

- Memory bandwidth bound.
- Reduce type: “sum”, “max”, “mean”.
- M and N refers to ***num_nodes***, nnz refers to ***num_edges***, K refers to ***num_features***.
- Input shape might be very large: for example, in ***GCN+ogbn-products:***
  - *i.* ***num_nodes:*** 2.4M
  - *ii.* ***num_edges:*** 126M
  - *iii.* ***num_features:*** 256

# Kernel Optimization II: spmm_reduce

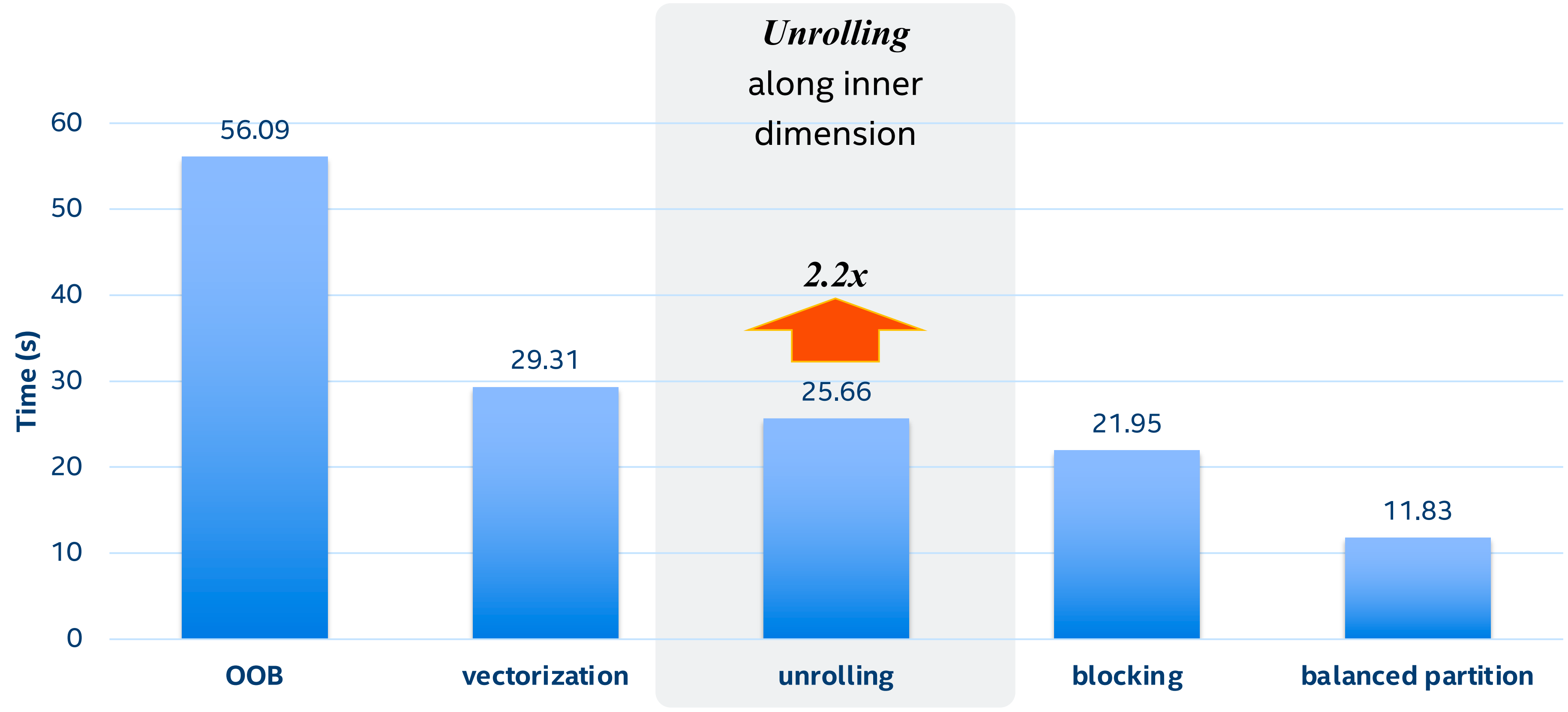
- ***GCN+ogbn-products*** single socket inference got ***4.3x*** speedup (spmm_sum improved by ***4.7x***)




***spmm_sum in GCN+obgn-products***

# Kernel Optimization II: spmm_reduce

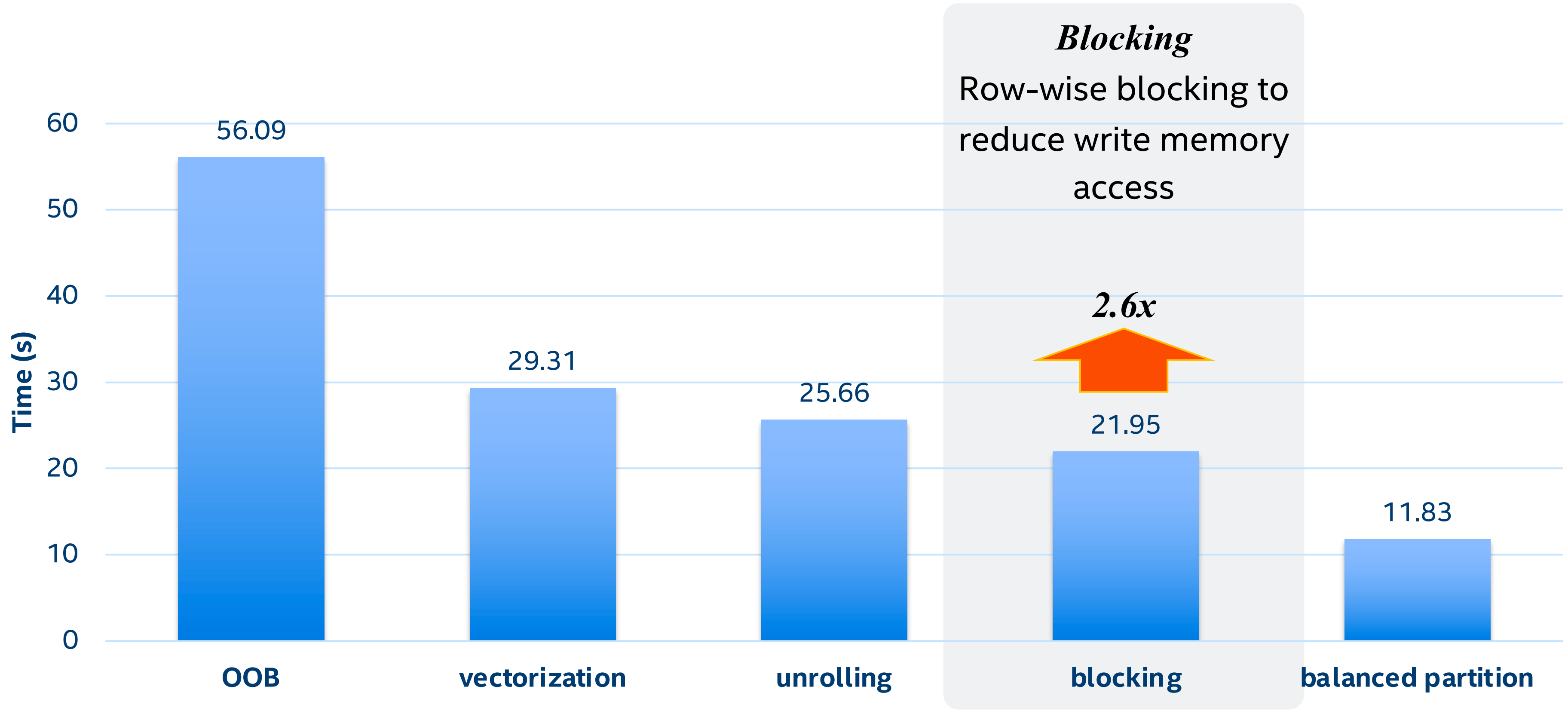
- ***GCN+ogbn-products*** single socket inference got ***4.3x*** speedup (spmm_sum improved by ***4.7x***)




***spmm_sum in GCN+obgn-products***

# Kernel Optimization II: spmm_reduce

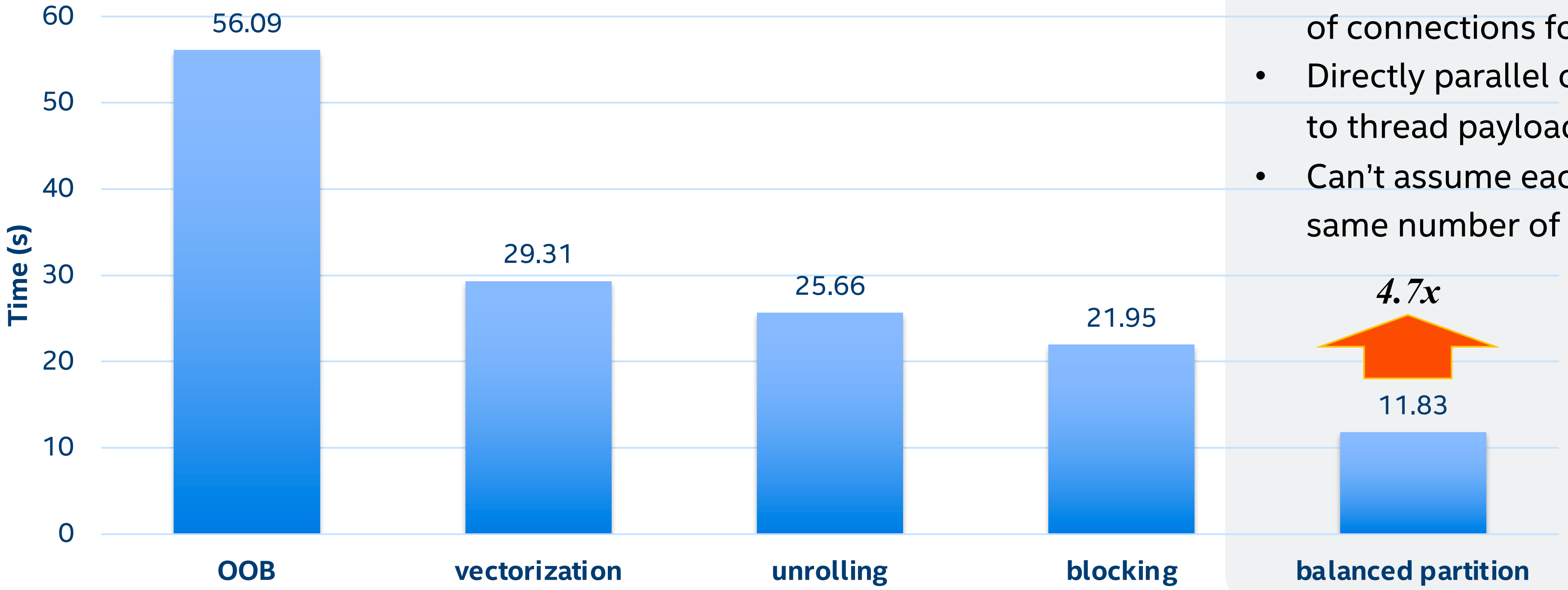
- ***GCN+ogbn-products*** single socket inference got ***4.3x*** speedup (spmm_sum improved by ***4.7x***)




***spmm_sum in GCN+obgn-products***

# Kernel Optimization II: spmm_reduce

- ***GCN+ogbn-products*** single socket inference got ***4.3x*** speedup (spmm_sum improved by ***4.7x***)



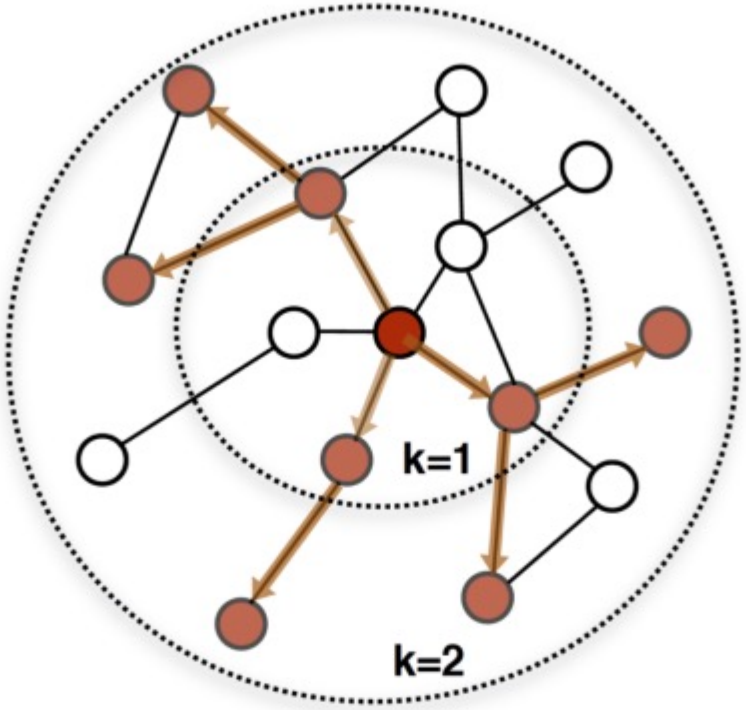

***Balanced Thread Partition***

- Length of each row refers to number of connections for each node.
- Directly parallel on rows would lead to thread payload unbalance.
- Can't assume each node has the same number of connections.

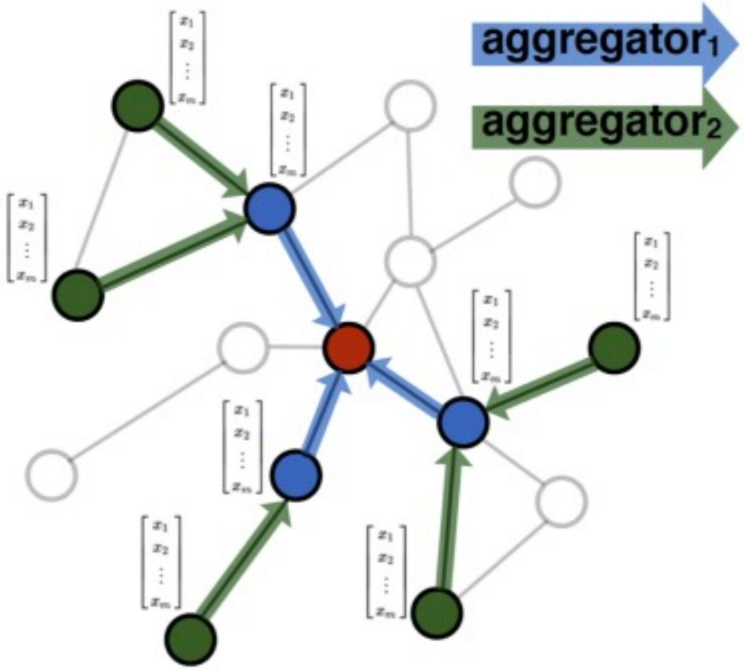
***spmm_sum in GCN+obgn-products***

# Node Sampling

## Larger graphs require node sampling and graph partitioning.




*1. Sampling neighborhood (k=1,2)*




*2. Aggregate features from neighbors*

# Node Sampling

- PyG has implemented commonly used samplers at ***torch_geometric.loader*** such as NeighborLoader, HGTLoader, RandomNodeSampler, etc.

- Concept of Loader is a combination of PyTorch's DataLoader and a specific sampler, which handles data sampling and transformation. On some workloads, loader itself might be ***major performance hotspot***.

| Name | Self CPU % | Self CPU |
| --- | --- | --- |
| _MultiProcessingDataLoaderIter... | 71.27% | 608.842s |
| torch_sparse::spmm_mean | 14.91% | 127.390s |
| aten::addmm | 3.77% | 32.166s |
| aten::copy_ | 3.60% | 30.766s |
| aten::mm | 2.29% | 19.588s |
| aten::native_batch_norm | 0.94% | 7.989s |
| aten::add_ | 0.57% | 4.877s |
| aten::clamp_min | 0.28% | 2.429s |
| aten::empty | 0.23% | 1.935s |
| torch_sparse::ptr2ind | 0.19% | 1.661s |
| aten::to | 0.18% | 1.550s |

***Profiling of GraphSAGE+mag240m***

- In the example of ***GraphSAGE + mag240m***, the DataLoader is major performance hotspot.
- Multi-process data loading is enabled if user set num_workers > 0, and the sampling from each worker will be sequential.
- Multi-process worker + sequential sampler be not optimal.

# Node Sampling

- PyG has implemented commonly used samplers at ***torch_geometric.loader*** such as NeighborLoader, HGTLoader, RandomNodeSampler, etc.
- Concept of Loader is a combination of PyTorch's DataLoader and a specific sampler, which handles data sampling and transformation. On some workloads, loader itself might be ***major performance hotspot***.

| Name | Self CPU % | Self CPU |
|---|---|---|
| _MultiProcessingDataLoaderIter... | 71.27% | 608.842s |
| torch_sparse::spmm_mean | 14.91% | 127.390s |
| aten::addmm | 3.77% | 32.166s |
| aten::copy_ | 3.60% | 30.766s |
| aten::mm | 2.29% | 19.588s |
| aten::native_batch_norm | 0.94% | 7.989s |
| aten::add_ | 0.57% | 4.877s |
| aten::clamp_min | 0.28% | 2.429s |
| aten::empty | 0.23% | 1.935s |
| torch_sparse::ptr2ind | 0.19% | 1.661s |
| aten::to | 0.18% | 1.550s |

*Profiling of GraphSAGE+mag240m*

```
def train_dataloader(self):
    return NeighborSampler(self.adj_t, node_idx=self.train_idx,
                           sizes=self.sizes, return_e_id=False,
                           transform=self.convert_batch,
                           batch_size=self.batch_size, shuffle=True,
                           num_workers=4)
```

```
def convert_batch(self, batch_size, n_id, adjs):
    if self.in_memory:
        x = self.x[n_id].to(torch.float)
    else:
        x = torch.from_numpy(self.x[n_id.numpy()]).to(torch.float)
    y = self.y[n_id[:batch_size]].to(torch.long)
    return Batch(x=x, y=y, adjs_t=[adj_t for adj_t, _, _ in adjs])
```

**48.6%** time spent on this circled line since only 4 cores used!

# Node Sampling

- PyG has implemented commonly used samplers at ***torch_geometric.loader*** such as NeighborLoader, HGTLoader, RandomNodeSampler, etc.
- Concept of Loader is a combination of PyTorch's DataLoader and a specific sampler, which handles data sampling and transformation. On some workloads, loader itself might be ***major performance hotspot***.

| Name | Self CPU % | Self CPU |
|---:|---:|---:|
| _MultiProcessingDataLoaderIter... | 71.27% | 608.842s |
| torch_sparse::spmm_mean | 14.91% | 127.390s |
| aten::addmm | 3.77% | 32.166s |
| aten::copy_ | 3.60% | 30.766s |
| aten::mm | 2.29% | 19.588s |
| aten::native_batch_norm | 0.94% | 7.989s |
| aten::add_ | 0.57% | 4.877s |
| aten::clamp_min | 0.28% | 2.429s |
| aten::empty | 0.23% | 1.935s |
| torch_sparse::ptr2ind | 0.19% | 1.661s |
| aten::to | 0.18% | 1.550s |

***Profiling of GraphSAGE+mag240m***

***Optimize data loader on CPU:***

- Make sure sampling and transformation can be ***properly paralleled***.
- Set CPU affinity if the multi-process data loader is used.
- Fuse multiple operators from ***convert_batch*** in C++ kernel.

# What's Next

- Fully Optimize PyG for both inference and training for Intel platforms
- Large scale distributed GNNs for top use cases
- Unified Graph Platform and workflows supporting query, analytics and GNNs

# Questions?