NYU

Parallelizing CoLA

Team 9
11/27/24

Jaideep Singh Chawla	jc12751
Rahul Raman	rr4549
Satyanarayana Chillale	sc9960

CoLA

- CoLA is a framework for scalable linear algebra in machine learning.
  - GPU backend: Pytorch, Jax
  - Algorithms that can exploit matrix structure for efficiency

- Our Objective: Focus on parallelizing the underlying algorithm. (Green)
  - CoLA Kernels

Motivation: Fused Kernels

```
n = 16384
A = torch.randn((n, n), device='cuda')
B = torch.randn((n, n), device='cuda')
C = torch.randn((n, n), device='cuda')

D = A * B + C
```

Pytorch

```
Name
aten::mul
cudaStreamIsCapturing
cudaMalloc
cudaLaunchKernel
aten::add
void at::native::vectorized_elementwise_kernel<4, at...
[memory]
cudaDeviceSynchronize
void at::native::vectorized_elementwise_kernel<4, at...
Self CPU time total: 33.791ms
Self CUDA time total: 25.542ms
```

Pytorch Compile

Can't generate all possible optimized kernels

```
Name
TorchDynamo Cache Lookup
Torch-Compiled Region
triton_poi_fused_add_mul_0
cuLaunchKernel
cudaDeviceSynchronize
triton_
Self CPU time total: 32.215ms
Self CUDA time total: 16.497ms
```

CoLA kernel

```
int idx = blockIdx.x * blockDim.x + threadIdx.x;
if (idx < numel)
    result[idx] = a[idx] * b[idx] + c[idx];
```

```
Name
cola_kernels::fuse_kernel_1
aten::empty
cudaStreamIsCapturing
cudaMalloc
cudaLaunchKernel
cola_kernels::fuse_kernel_1(int, float const*, float...
[memory]
cudaDeviceSynchronize
Self CPU time total: 34.517ms
Self CUDA time total: 15.929ms
```

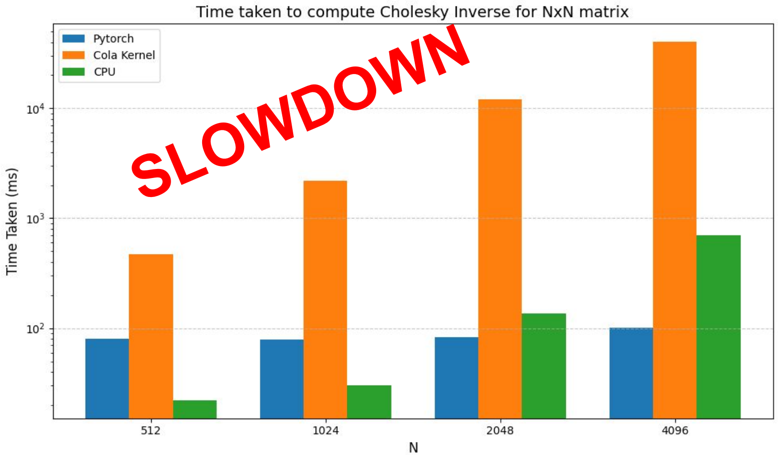
Algorithms:

- Cholesky Inverse
- Arnoldi
- Hutchinson
- SVD

Experiment:

- pip install
- Cuda-3
- cudaSetDevice(1)

Cholesky Inversion



Kernel Name	CPU Time	CUDA Time	CUDA Memory
Memcpy DtoH (Device → Pinned)	0 ms	2.5 ms	0 MB
`aten::linalg_cholesky`	66.074 ms	791.001 μs	16.00 MB
`aten::linalg_cholesky_ex`	59.751 ms	0 ms	16.00 MB
`aten::cholesky_inverse`	28.812 ms	8.444 ms	32.00 MB
Total (Cholesky)	154.537 ms	11.2 ms	128 MB

Table 3: Performance analysis of Pytorch decomposition kernel.

Kernel Name	CPU Time	CUDA Time	CUDA Memory
`decompose_cholesky(float*, int)`	0 ms	859.062 ms	16.00 MB
`cudaLaunchCooperativeKernel`	0 ms	905.684 μs	0 MB

Table 4: Performance analysis of our Cholesky decomposition kernel.

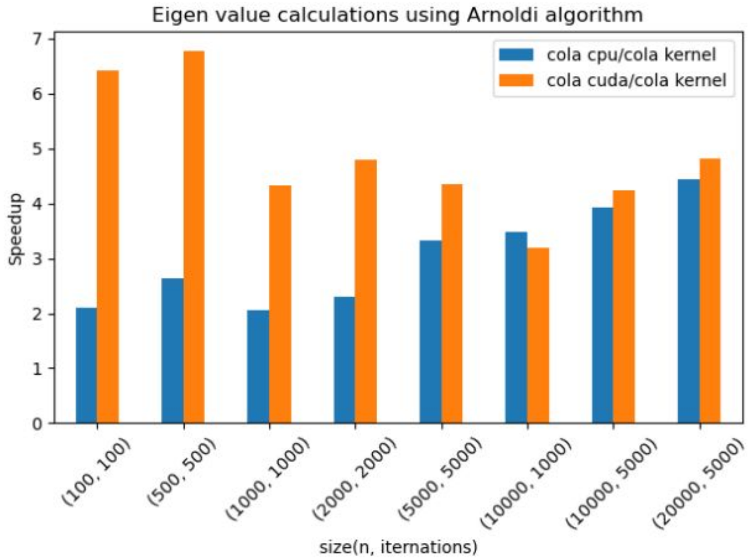
- **Iterative approach and interdependency of elements** are not GPU friendly:
  - Cooperative_groups
  - atomicAdd

Key Insight:

- Choose GPU-friendly algorithm
- Reduce / remove block-level synchronization.

NYU

Arnoldi Eigen calculation



Implementation/Matrix size	100k	400k
Pytorch	67	80
ColA GPU	20	20
ColA Cuda	60	70

Table 2: SM utilization (%)

Key Insight:

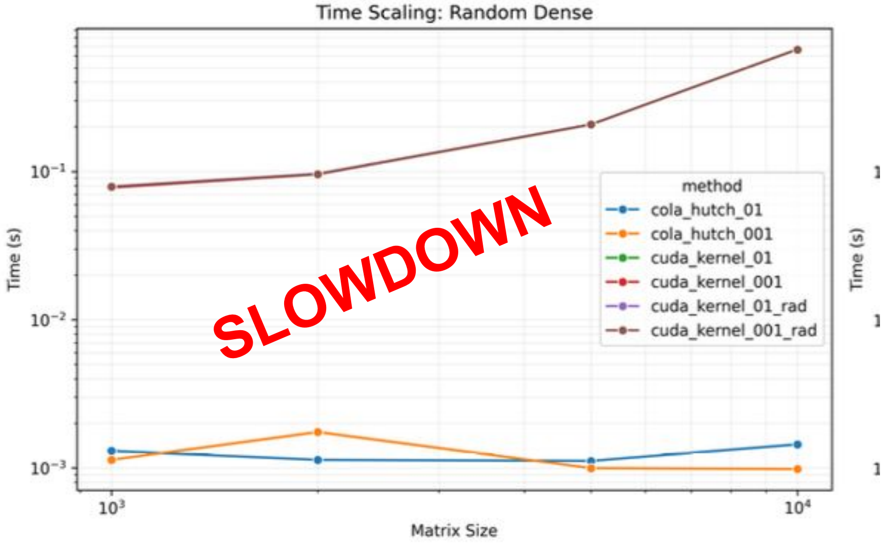
- Data coalescing
- Privatization
- No transfer of data from GPU to CPU

NYU

Hutchinson Method for Diagonal Estimation

Key Features:

- Batch processing
- Shared Memory Optimizations
- Parallel Reduction for checking convergence
- cudaMemcpyAsync, cudaMemsetAsync
- Custom stream for computation, only synchronizes every 10 iterations (when we want to check for convergence)



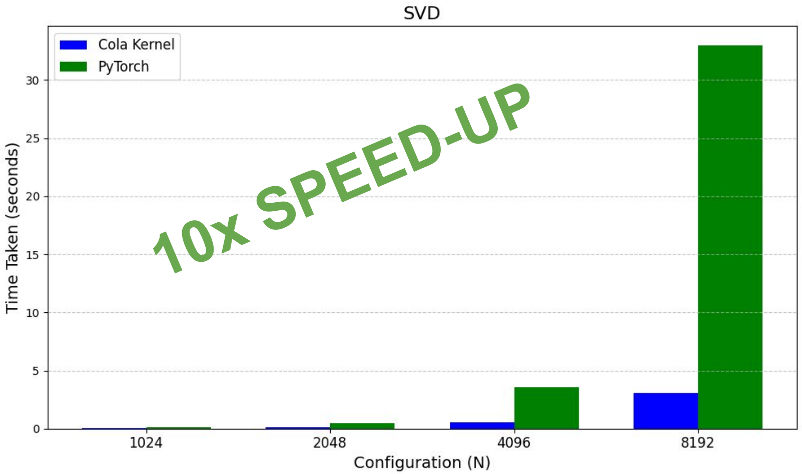
Why is it slow?

NYU

Team 9

SVD

- Used **cuBLAS** and **cuSOLVERDn**



Kernel Name	Total CUDA Time	Avg. CUDA Time	No. of Kernel calls
svd_column_rotate_batch	18.146 s	1.366 ms	13286
svd_row_rotate_batch	13.981 s	2.105m s	6643

Table 5: Profiler of Pytorch SVD

Kernel Name	Total CUDA Time	Avg. CUDA Time	No. of calls
cuds_symv_alg6_stage1_upper	1.541 s	188.064 μs	8192

Table 6: Profiler of cuBLAS SVD

Key Insight:

- Breakdown to **submatrices**.
- **Iterative algorithm:** launch kernel for every iteration.

NYU

Conclusion

1. We **eliminated PyTorch's overheads** by writing custom GPU code, optimizing memory usage, parallelism, and grid configurations, **hoping** for speedup and efficiency.

2. Deciding which operations to **merge** while maintaining **separate kernel calls for each iteration** resulted in a notable speedup.

3. Better memory bandwidth utilization, using shared memory, and ensuring coalesced memory access patterns are only **scratching the surface of CUDA optimizations**, Need to look out for potential bottlenecks due to memory management and branch divergence.

Q & A

NYU