High Performance Computing
Homework #5: Phase 1
Due: Wed Oct 20 2021 before 11:59 PM
Email-based help Cutoff: 5:00 PM on Tue, Oct 19 2021

> **!** The runtime data and results in this report are meaningful only if your implementation is functionally correct and produce similar outputs as the reference run.

Name: John Doll

Experimental Platform

The experiments documented in this report were conducted on the following platform:

Component	*Details*
CPU Model	Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz
CPU/Core Speed	2.4Ghz
Operating system used	Linux pitzer-login04.hpc.osc.edu 3.10.0-1160.36.2.el7.x86_64 #1 SMP Thu Jul 8 02:53:40 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux
Interconnect type & speed (if applicable)	Not applicable
Was machine dedicated to task (`yes`/`no`)	Yes (via a `slurm` job)
Name and version of C++ compiler (if used)	gcc version 8.4.0 (GCC)
Name and version of Java compiler (if used)	None
Name and version of other non-standard software tools & components (if used)	

Runtime data for the reference performance

In the table below, record the reference runtime characteristics of the starter code:

Rep	User time (sec)	Elapsed time (sec)	Peak memory (KB)
1	33.27	33.96	3516

2	33.26	33.94	3516
3	33.23	33.89	3516
4	33.22	33.91	3516
5	33.19	33.88	3516

Perf report data for the reference implementation

In the space below, copy-paste the `perf` profile data that you used to identify the aspect/method to reimplement to improve performance:

```
--17.14%--NeuralNet::classify
                  |       |
                  |       |--10.26%--Matrix::dot
                  |       |
                  |       |--3.95%--Matrix::Matrix (inlined)
                  |          |          std::vector<std::vector<double, std::allocator<double> >,
std::allocator<std::vector<double, std::allocator<double> > > >::vector (inlined)
                  |                                                                          |
std::__uninitialized_copy_a<__gnu_cxx::__normal_iterator<std::vector<double,
std::allocator<double> > const*, std::vector<std::vector<double, std::allocator<double> >,
std::allocator<std::vector<double, std::allocator<double> > > >, std::vector<double,
std::allocator<double> >*, std::vector<double, std::allocator<double> > > (inlined)
                  |                                                                          |
std::uninitialized_copy<__gnu_cxx::__normal_iterator<std::vector<double,
std::allocator<double> > const*, std::vector<std::vector<double, std::allocator<double> >,
std::allocator<std::vector<double, std::allocator<double> > > >, std::vector<double,
std::allocator<double> >*> (inlined)
                  |                                                                          |
std::__uninitialized_copy<false>::__uninit_copy<__gnu_cxx::__normal_iterator<std::vector<
double, std::allocator<double> > const*, std::vector<std::vector<double, std::allocator<double>
>, std::allocator<std::vector<double, std::allocator<double> > > >, std::vector<double,
std::allocator<double> >*> (inlined)
                  |       |       std::_Construct<std::vector<double, std::allocator<double> >,
std::vector<double, std::allocator<double> > const&> (inlined)
                  |       |       std::vector<double, std::allocator<double> >::vector (inlined)
                  |       |       |
                  |       |          --3.51%--std::_Vector_base<double, std::allocator<double>
>::_Vector_base (inlined)
                  |       |                  std::_Vector_base<double, std::allocator<double>
>::_M_create_storage (inlined)
                  |       |          |
                  |       |          --3.08%--std::_Vector_base<double, std::allocator<double>
>::_M_allocate (inlined)
                  |       |                  std::allocator_traits<std::allocator<double> >::allocate
(inlined)
```

```
|       |                    __gnu_cxx::new_allocator<double>::allocate (inlined)
|       |                      |
|       |                       --2.78%--operator new
|       |                                 |
|       |                                  --2.64%--__GI___libc_malloc (inlined)
|       |                                            |
|       |                                             --2.20%--_int_malloc
```

Description of performance improvement

Briefly describe the performance improvement you are implementing. Your description should document:

- Why you chose the specific aspect/feature to improve (obviously it should be supported by your `perf` data)
- What is the best-case improvement that you anticipate – for example, if you optimize a feature that takes 25% of runtime, then the best case would be a 25% reduction in runtime.
- Briefly describe what/how you plan to change the implementation

I chose to improve the Neural Net Classify method by not copying the inputs matrix over to a results matrix and instead operating directly on the matrix that we are using writing over on the return anyway. This method takes 17.14% of runtime, so at best the program would experience a 17.14% reduction in runtime. I chose to improve the Neural Net Classify because it has one of the largest operations by percentage with Matrix::dot taking over 10% of runtime.

Runtime statistics from performance improvement

Use the supplied SLURM script to collect runtime statistics for your enhanced implementation.

Rep	User time (sec)	Elapsed time (sec)	Peak memory (KB)
1	30.62	31.60	3516
2	30.51	31.18	3516
3	30.52	31.24	3516
4	30.48	31.21	3516
5	30.51	31.27	3516

Comparative runtime analysis

Compare the runtimes (*i.e.*, before and after your changes) by fill-in the Runtime Comparison Template and copy-paste the full sheet in the space below:

Replicate#	User Time Orig	User Time Adj
1	33.27	30.62
2	33.26	30.51
3	33.23	30.52
4	33.22	30.48
5	33.19	30.51
Average:	33.234	30.528
SD:	0.03209361307	0.05357238094
95% CI Range:	0.03984948392	0.06651889672
Stats:	**33.234 ± 0.04**	**30.528 ± 0.07**
T-Test (H_0: μ1=μ2)	0	

Inferences & Discussions

Now, using the data from the runtime statistics discuss (at least 5-to-6 sentences) the change in runtime characteristics (both time and memory) due to your changes. Compare and contrast key aspects/changes to the implementation. Include any additional inferences as to why one version performs better than the other.

The memory stays at a constant 3516 KB max, but the user time is reduced by 8.1% from the unedited version to the improved version ((33.234-30.528)/33.234). Key changes that were made include removing the ret Matrix in the NeuralNet Classify method. Instead, in the for loop, I made all operations happen to the inputs Matrix that was passed through as a parameter. I made the Classify method return void so then I also removed the return statement at the end. Since I was now editing the inputs Matrix, I could no longer pass it as const or keep the method with a const declaration. This means that I also had to go into the header file and adjust the method accordingly, removing the const parameter, const method declaration, and changing the return type to void. Additionally, in the main.cpp file in the assess method, I went in and changed the Matrix img from being const to being mutable, removed the Matrix res declaration for the net.classify(img), and adjusted the resIdx to transpose img instead of the now non-existent res.