High Performance Computing
Homework #5: Phase 2

Due: Wed Oct 27 2021 before 11:59 PM

Email-based help Cutoff: 5:00 PM on Tue, Oct 26 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. This is the data for your enhanced version from Phase #1:

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

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:

```
--3.47%--Matrix::operator- (inlined)
            |                                                              |                              |
_ZNK6Matrix5applyIZNKS_miERKS_EUlRKT_RKT0_E_EES_S2_S5_ (inlined)
            |                |        |        |
            |                |        |        |--2.21%--Matrix::Matrix (inlined)
            |               |       |        |       std::vector<std::vector<double, std::allocator<double>
>, std::allocator<std::vector<double, std::allocator<double> > > >::vector (inlined)
            |                |        |        |        |
            |                                  |                 |                 |                  --2.05%--
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)
            |                |        |        |                |
            |                            |              |              |                              --1.58%--
std::__uninitialized_copy_a<__gnu_cxx::__normal_iterator<double                        const*,
std::vector<double, std::allocator<double> > >, double*, double> (inlined)
```

```
          |                                              |                     |                     |
std::uninitialized_copy<__gnu_cxx::__normal_iterator<double const*, std::vector<double,
std::allocator<double> > >, double*> (inlined)
          |                                              |                     |                     |
std::__uninitialized_copy<true>::__uninit_copy<__gnu_cxx::__normal_iterator<double
const*, std::vector<double, std::allocator<double> > >, double*> (inlined)
          |          |     |     |              std::copy<__gnu_cxx::__normal_iterator<double
const*, std::vector<double, std::allocator<double> > >, double*> (inlined)
          |               |        |        |                          std::__copy_move_a2<false,
__gnu_cxx::__normal_iterator<double const*, std::vector<double, std::allocator<double> > >,
double*> (inlined)
          |               |       |       |                      std::__copy_move_a<false, double
const*, double*> (inlined)
          |               |        |        |                        std::__copy_move<false, true,
std::random_access_iterator_tag>::__copy_m<double> (inlined)
          |              |       |       |                    __memmove_ssse3_back
          |              |       |       |
          |                                        |                  |                    --0.79%--
_ZZNK6MatrixmiERKS_ENKUlRKT_RKT0_E_clIddEEDaS4_S7_ (inlined)








--1.89%--Matrix::operator* (inlined)
          |              |       |       _ZNK6Matrix5applyIZNKS_mlEdEUlRKT_E_EES_S3_
(inlined)
          |              |       |       |
          |              |       |       --1.26%--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)
            |                |        |                |
            |                                    |                 |                                      --0.94%--
std::__uninitialized_copy_a<__gnu_cxx::__normal_iterator<double                              const*,
std::vector<double, std::allocator<double> > >, double*, double> (inlined)
            |                                                                      |                                  |
std::uninitialized_copy<__gnu_cxx::__normal_iterator<double   const*,   std::vector<double,
std::allocator<double> > >, double*> (inlined)
            |                                                                      |                                  |
std::__uninitialized_copy<true>::__uninit_copy<__gnu_cxx::__normal_iterator<double
const*, std::vector<double, std::allocator<double> > >, double*> (inlined)
            |               |       |                     std::copy<__gnu_cxx::__normal_iterator<double
const*, std::vector<double, std::allocator<double> > >, double*> (inlined)
            |                      |          |                                std::__copy_move_a2<false,
__gnu_cxx::__normal_iterator<double const*, std::vector<double, std::allocator<double> > >,
double*> (inlined)
            |                 |        |                        std::__copy_move_a<false, double const*,
double*> (inlined)
            |                      |          |                              std::__copy_move<false, true,
std::random_access_iterator_tag>::__copy_m<double> (inlined)
            |                |        |                         __memmove_ssse3_back
```

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 noticed the multiply and subtract operators were making many extraneous calls for just operating on what should be one matrix. Unneeded copy matrices were being formed, so my plan is to Implement new and better multiplication and subtraction operators for the matrices that do not create new matrices, and instead operate directly on the matrices that they are being called on. These two are in the NeuralNet::learn method which took 30.71% of my runtime before I made any changes, so at best, there would be a 30.71% decrease in 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	27.20	27.74	3444
2	27.28	27.83	3444
3	27.04	27.51	3531
4	25.69	26.13	3440
5	25.63	26.13	3299

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:

Change tile for cases and the type of data you are entering		
Replicate#	**User time after Pt1**	**User time now**
1	30.62	27.2
2	30.51	27.28
3	30.52	27.04
4	30.48	25.69
5	30.51	25.63
Average:	30.528	26.568
SD:	0.05357238094	0.833648607
95% CI Range:	0.06651889672	1.035111462
Stats:	**30.528 ± 0.07**	**26.568 ± 1.04**
T-Test (H_0: μ1=μ2)	0.0004294022862	

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 user time decreased by about 4 seconds per run on average from part 1 of the homework to completion of part 2 of the homework. From the original code to now, this is about a 7 second decrease, as we used to average roughly 33 seconds. Not only is user (and elapsed) time decreased, but the memory usage also decreases by about 100KB on average per run. So the new implementation is both faster and more efficient for time and memory. This is because we were not creating copy matrices, and we were making less method calls, which each helped to reduce run time and memory usage.