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: **Monu Chaudhary**

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	2.40GHz
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)	
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

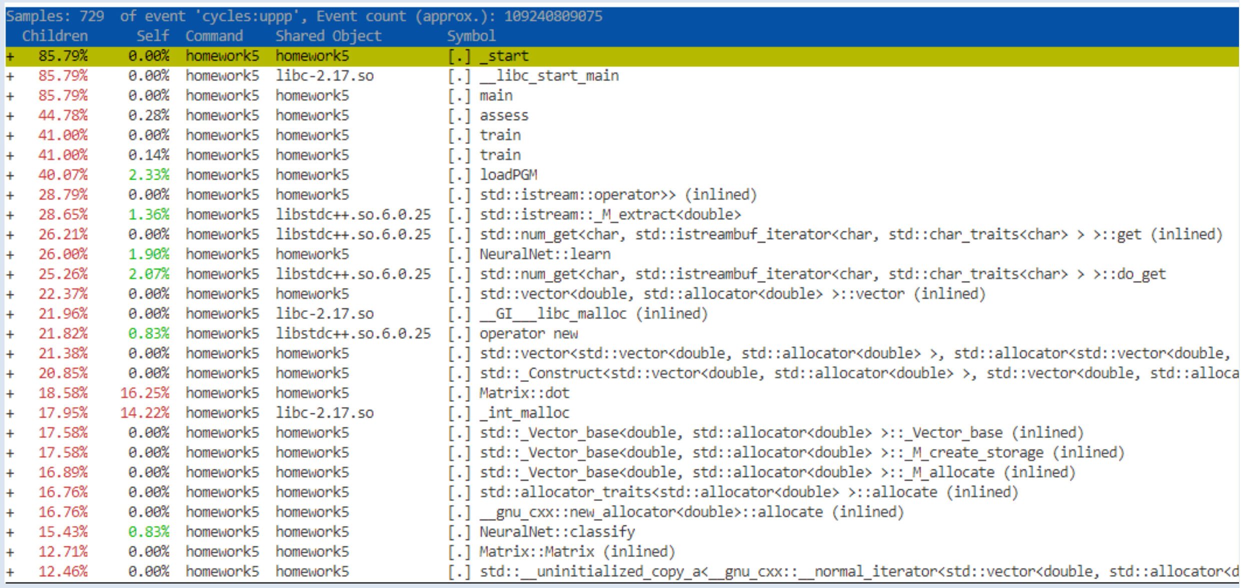
In the table below, record the reference runtime characteristics. This is the data for your enhanced version from Phase #1:

Rep	**User time (sec)**	**Elapsed time (sec)**	**Peak memory (KB)**
1	30.33	31.13	3268
2	30.09	30.91	3268
3	30.23	30.97	3268
4	30.60	31.33	3272

5	30.30	31.02	3268

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:



```
Samples: 729  of event 'cycles:uppp', Event count (approx.): 109240809075
  Children      Self  Command     Shared Object          Symbol
+   85.79%     0.00%  homework5  homework5              [.] _start
+   85.79%     0.00%  homework5  libc-2.17.so           [.] __libc_start_main
+   85.79%     0.00%  homework5  homework5              [.] main
+   44.78%     0.28%  homework5  homework5              [.] assess
+   41.00%     0.00%  homework5  homework5              [.] train
+   41.00%     0.14%  homework5  homework5              [.] train
+   40.07%     2.33%  homework5  homework5              [.] loadPGM
+   28.79%     0.00%  homework5  homework5              [.] std::istream::operator>> (inlined)
+   28.65%     1.36%  homework5  libstdc++.so.6.0.25    [.] std::istream::_M_extract<double>
+   26.21%     0.00%  homework5  libstdc++.so.6.0.25    [.] std::num_get<char, std::istreambuf_iterator<char, std::char_traits<char> > >::get (inlined)
+   26.00%     1.90%  homework5  homework5              [.] NeuralNet::learn
+   25.26%     2.07%  homework5  libstdc++.so.6.0.25    [.] std::num_get<char, std::istreambuf_iterator<char, std::char_traits<char> > >::do_get
+   22.37%     0.00%  homework5  homework5              [.] std::vector<double, std::allocator<double> >::vector (inlined)
+   21.96%     0.00%  homework5  libc-2.17.so           [.] __GI___libc_malloc (inlined)
+   21.82%     0.83%  homework5  libstdc++.so.6.0.25    [.] operator new
+   21.38%     0.00%  homework5  homework5              [.] std::vector<std::vector<double, std::allocator<double> >, std::allocator<std::vector<double,
+   20.85%     0.00%  homework5  homework5              [.] std::_Construct<std::vector<double, std::allocator<double> >, std::vector<double, std::alloca
+   18.58%    16.25%  homework5  homework5              [.] Matrix::dot
+   17.95%    14.22%  homework5  libc-2.17.so           [.] _int_malloc
+   17.58%     0.00%  homework5  homework5              [.] std::_Vector_base<double, std::allocator<double> >::_Vector_base (inlined)
+   17.58%     0.00%  homework5  homework5              [.] std::_Vector_base<double, std::allocator<double> >::_M_create_storage (inlined)
+   16.89%     0.00%  homework5  homework5              [.] std::_Vector_base<double, std::allocator<double> >::_M_allocate (inlined)
+   16.76%     0.00%  homework5  homework5              [.] std::allocator_traits<std::allocator<double> >::allocate (inlined)
+   16.76%     0.00%  homework5  homework5              [.] __gnu_cxx::new_allocator<double>::allocate (inlined)
+   15.43%     0.83%  homework5  homework5              [.] NeuralNet::classify
+   12.71%     0.00%  homework5  homework5              [.] Matrix::Matrix (inlined)
+   12.46%     0.00%  homework5  homework5              [.] std::__uninitialized_copy_a<__gnu_cxx::__normal_iterator<std::vector<double, std::allocator<d
```

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

The `perf` data shows that the highest performance bottleneck are caused by the matrix::dot operation, memory allocation (_int_malloc) with 16.25%, and 14.22 overheads respectively. Hence, for performance improvement, above operations should be optimized.

The best-case improvement that can be anticipated if both matrix::dot and memory allocation operations are optimized to a theoretical optimum performance would be 30.47%. However, in practical, it is not possible to optimize to the theoretical level.

The plans taken into consideration to improve the performance are:

1. Reimplement the Matrix class as a Vector class so that the memory allocation can be optimized.

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	20.01	20.65	3664
2	20.02	20.62	3660
3	20.14	20.82	3660
4	20.01	20.62	3660
5	20.06	20.68	3664

Comparative runtime analysis

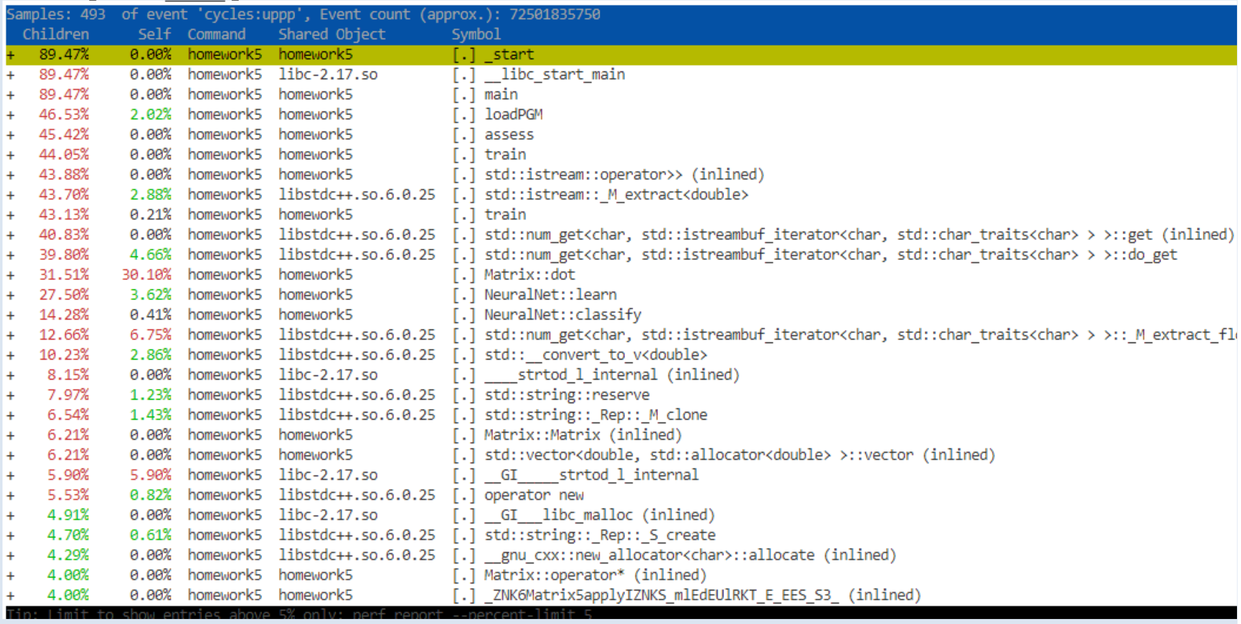
Compare the runtimes (*i.e.*, before and after your changes) by fill-in the Runtime Comparison Template and copy-paste the full sheet in the space below:

Change tile for cases and the type of data you are entering		
Replicate#	**Before Optimization**	**After Optimization**
1	31.13	20.65
2	30.91	20.62
3	30.97	20.82
4	31.33	20.62
5	31.02	20.68
Average:	31.072	20.678
SD:	0.1652876281	0.08318653737
95% CI Range:	0.2052316971	0.1032897286
Stats:	**31.072 ± 0.21**	**20.678 ± 0.1**
T-Test (H_0: μ1=μ2)	0	

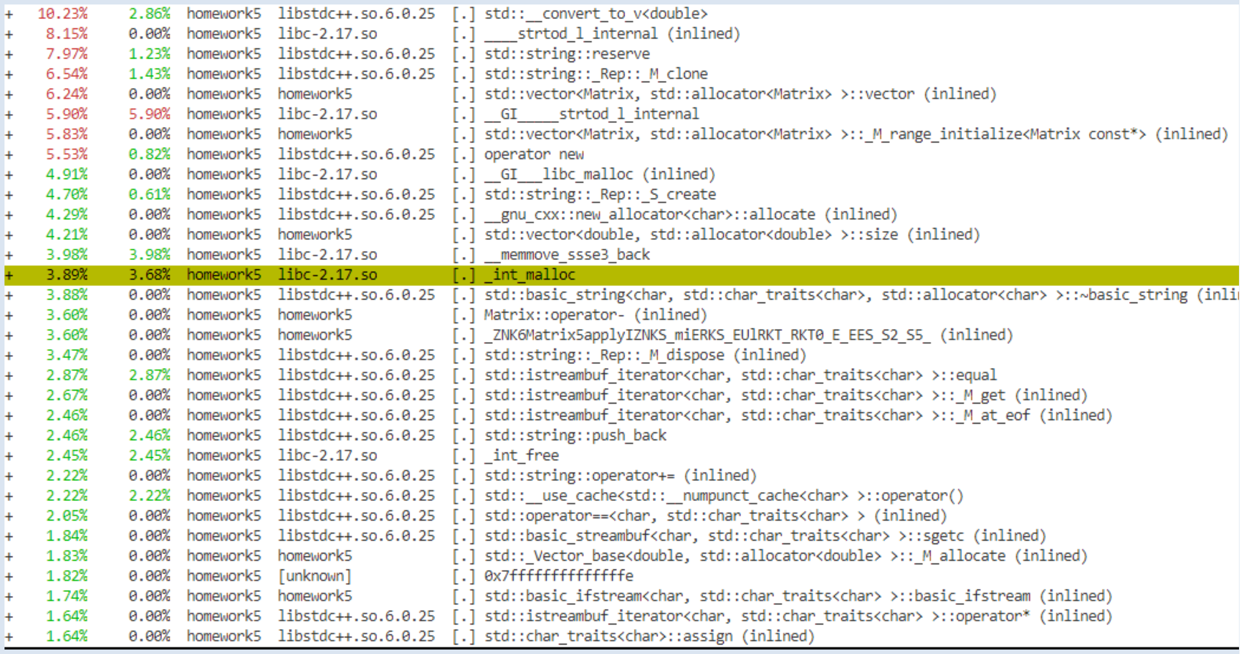
Inferences & Discussions

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.

Perf report after optimization:



```
Samples: 493  of event 'cycles:uppp', Event count (approx.): 72501835750
  Children      Self  Command    Shared Object          Symbol
+   89.47%     0.00%  homework5  homework5              [.] _start
+   89.47%     0.00%  homework5  libc-2.17.so           [.] __libc_start_main
+   89.47%     0.00%  homework5  homework5              [.] main
+   46.53%     2.02%  homework5  homework5              [.] loadPGM
+   45.42%     0.00%  homework5  homework5              [.] assess
+   44.05%     0.00%  homework5  homework5              [.] train
+   43.88%     0.00%  homework5  homework5              [.] std::istream::operator>> (inlined)
+   43.70%     2.88%  homework5  libstdc++.so.6.0.25    [.] std::istream::_M_extract<double>
+   43.13%     0.21%  homework5  homework5              [.] train
+   40.83%     0.00%  homework5  libstdc++.so.6.0.25    [.] std::num_get<char, std::istreambuf_iterator<char, std::char_traits<char> > >::get (inlined)
+   39.80%     4.66%  homework5  libstdc++.so.6.0.25    [.] std::num_get<char, std::istreambuf_iterator<char, std::char_traits<char> > >::do_get
+   31.51%    30.10%  homework5  homework5              [.] Matrix::dot
+   27.50%     3.62%  homework5  homework5              [.] NeuralNet::learn
+   14.28%     0.41%  homework5  homework5              [.] NeuralNet::classify
+   12.66%     6.75%  homework5  libstdc++.so.6.0.25    [.] std::num_get<char, std::istreambuf_iterator<char, std::char_traits<char> > >::_M_extract_fl
+   10.23%     2.86%  homework5  libstdc++.so.6.0.25    [.] std::__convert_to_v<double>
+    8.15%     0.00%  homework5  libc-2.17.so           [.] ____strtod_l_internal (inlined)
+    7.97%     1.23%  homework5  libstdc++.so.6.0.25    [.] std::string::reserve
+    6.54%     1.43%  homework5  libstdc++.so.6.0.25    [.] std::string::_Rep::_M_clone
+    6.21%     0.00%  homework5  homework5              [.] Matrix::Matrix (inlined)
+    6.21%     0.00%  homework5  homework5              [.] std::vector<double, std::allocator<double> >::vector (inlined)
+    5.90%     5.90%  homework5  libc-2.17.so           [.] __GI_____strtod_l_internal
+    5.53%     0.82%  homework5  libstdc++.so.6.0.25    [.] operator new
+    4.91%     0.00%  homework5  libc-2.17.so           [.] __GI___libc_malloc (inlined)
+    4.70%     0.61%  homework5  libstdc++.so.6.0.25    [.] std::string::_Rep::_S_create
+    4.29%     0.00%  homework5  libstdc++.so.6.0.25    [.] __gnu_cxx::new_allocator<char>::allocate (inlined)
+    4.00%     0.00%  homework5  homework5              [.] Matrix::operator* (inlined)
+    4.00%     0.00%  homework5  homework5              [.] _ZNK6Matrix5applyIZNKS_mlEdEUlRKT_E_EES_S3_ (inlined)
Tip: Limit to show entries above 5% only: perf report --percent-limit 5
```



```
+   10.23%     2.86%  homework5  libstdc++.so.6.0.25    [.] std::__convert_to_v<double>
+    8.15%     0.00%  homework5  libc-2.17.so           [.] ____strtod_l_internal (inlined)
+    7.97%     1.23%  homework5  libstdc++.so.6.0.25    [.] std::string::reserve
+    6.54%     1.43%  homework5  libstdc++.so.6.0.25    [.] std::string::_Rep::_M_clone
+    6.24%     0.00%  homework5  homework5              [.] std::vector<Matrix, std::allocator<Matrix> >::vector (inlined)
+    5.90%     5.90%  homework5  libc-2.17.so           [.] __GI_____strtod_l_internal
+    5.83%     0.00%  homework5  homework5              [.] std::vector<Matrix, std::allocator<Matrix> >::_M_range_initialize<Matrix const*> (inlined)
+    5.53%     0.82%  homework5  libstdc++.so.6.0.25    [.] operator new
+    4.91%     0.00%  homework5  libc-2.17.so           [.] __GI___libc_malloc (inlined)
+    4.70%     0.61%  homework5  libstdc++.so.6.0.25    [.] std::string::_Rep::_S_create
+    4.29%     0.00%  homework5  libstdc++.so.6.0.25    [.] __gnu_cxx::new_allocator<char>::allocate (inlined)
+    4.21%     0.00%  homework5  homework5              [.] std::vector<double, std::allocator<double> >::size (inlined)
+    3.98%     3.98%  homework5  libc-2.17.so           [.] __memmove_ssse3_back
+    3.89%     3.68%  homework5  libc-2.17.so           [.] _int_malloc
+    3.88%     0.00%  homework5  libstdc++.so.6.0.25    [.] std::basic_string<char, std::char_traits<char>, std::allocator<char> >::~basic_string (inli
+    3.60%     0.00%  homework5  homework5              [.] Matrix::operator- (inlined)
+    3.60%     0.00%  homework5  homework5              [.] _ZNK6Matrix5applyIZNKS_miERKS_EUlRKT_RKT0_E_EES_S2_S5_ (inlined)
+    3.47%     0.00%  homework5  libstdc++.so.6.0.25    [.] std::string::_Rep::_M_dispose (inlined)
+    2.87%     2.87%  homework5  libstdc++.so.6.0.25    [.] std::istreambuf_iterator<char, std::char_traits<char> >::equal
+    2.67%     0.00%  homework5  libstdc++.so.6.0.25    [.] std::istreambuf_iterator<char, std::char_traits<char> >::_M_get (inlined)
+    2.46%     0.00%  homework5  libstdc++.so.6.0.25    [.] std::istreambuf_iterator<char, std::char_traits<char> >::_M_at_eof (inlined)
+    2.46%     2.46%  homework5  libstdc++.so.6.0.25    [.] std::string::push_back
+    2.45%     2.45%  homework5  libc-2.17.so           [.] _int_free
+    2.22%     0.00%  homework5  libstdc++.so.6.0.25    [.] std::string::operator+= (inlined)
+    2.22%     2.22%  homework5  libstdc++.so.6.0.25    [.] std::__use_cache<std::__numpunct_cache<char> >::operator()
+    2.05%     0.00%  homework5  libstdc++.so.6.0.25    [.] std::operator==<char, std::char_traits<char> > (inlined)
+    1.84%     0.00%  homework5  libstdc++.so.6.0.25    [.] std::basic_streambuf<char, std::char_traits<char> >::sgetc (inlined)
+    1.83%     0.00%  homework5  homework5              [.] std::_Vector_base<double, std::allocator<double> >::_M_allocate (inlined)
+    1.82%     0.00%  homework5  [unknown]              [.] 0x7ffffffffffffffe
+    1.74%     0.00%  homework5  homework5              [.] std::basic_ifstream<char, std::char_traits<char> >::basic_ifstream (inlined)
+    1.64%     0.00%  homework5  libstdc++.so.6.0.25    [.] std::istreambuf_iterator<char, std::char_traits<char> >::operator* (inlined)
+    1.64%     0.00%  homework5  libstdc++.so.6.0.25    [.] std::char_traits<char>::assign (inlined)
```

We can see from the perf report that the runtime performance for memory allocation has reduced to 3.68%. Now, matrix::dot operation takes the highest percentage of runtime.

From the runtime statistics for code before and after enhancement implementation, it is observed that the code was optimized successfully when the data from perf report was taken into consideration. A runtime average of 20.678 seconds was achieved after optimization which is a drop of about 10 seconds. This is because of the use of vector data structure instead of a matrix to store the data. Using a vector data structure reduces the amount of cache miss that occurs in case of matrix. Also, the number of for loops in some of the method implementation also decreases which might have contributed to the decrease in the runtime statistics. However, this is achieved with a tradeoff with memory usage which is increased by about 13%.