**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:** | **Maciej Wozniak** |

## 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 | Unix |
| 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 of the starter code:

|  |  |  |  |
| --- | --- | --- | --- |
| **Rep** | **User time (sec)** | **Elapsed time (sec)** | **Peak memory (KB)** |
| 1 | 31.49 | 34.13 | 3516 |
| 2 | 31.52 | 34.14 | 3516 |
| 3 | 31.12 | 34.73 | 3516 |
| 4 | 31.59 | 34.23 | 3516 |
| 5 | 31.63 | 34.30 | 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:

|  |
| --- |
|  |

## 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 will focus on improving the parts which has the highest self-percentage. We can see that these are **matrix.dot,**  Some functions in neuralnetwork classify part    I will focus on improving these functions. It is hard to say what exactly will be improvements but we can expect 20-30% improvement if we decrease value of all 3 functions from 15-20% to 5%. Also if we check the whole **perf** report we can see that a lot of time is spent on allocation. |
|  |
|  |
|  |
|  |

## 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 | 39.49 | 129.06 | 3696 |
| 2 | 39.64 | 129.02 | 3696 |
| 3 | 38.82 | 126.39 | 3696 |
| 4 | 39.24 | 128.89 | 3696 |
| 5 | 40.88 | 130.73 | 3696 |

## Comparative runtime analysis

Compare the runtimes (*i.e.*, before and after your changes) by fill-in the [Runtime Comparison Template](https://docs.google.com/spreadsheets/d/1XURuABw8MFCNCC8p_ced5Z0enwzx63Ec-FVpLXrYdfc/edit#gid=0) and copy-paste the full sheet in the space below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| User time comparison | | |  | elapsed time | | |
| **Replicate#** | **Original code** | **Edited code** |  | **Replicate#** | **Original code** | **Edited code** |
| 1 | 31.49 | 39.49 |  | 1 | 34.13 | 129.06 |
| 2 | 31.52 | 39.64 |  | 2 | 34.14 | 129.02 |
| 3 | 31.12 | 38.82 |  | 3 | 34.73 | 126.39 |
| 4 | 31.59 | 39.24 |  | 4 | 34.23 | 128.89 |
| 5 | 31.63 | 40.88 |  | 5 | 34.3 | 130.73 |
| **Average:** | 31.47 | 39.614 |  | **Average:** | 34.306 | 128.818 |
| **SD:** | 0.2033469941 | 0.7728389224 |  | **SD:** | 0.2470425065 | 1.553663413 |
| **95% CI Range:** | 0.2524886417 | 0.9596062664 |  | **95% CI Range:** | 0.3067437863 | 1.929127925 |
| **Stats:** | **31.47 ± 0.25** | **39.614 ± 0.96** |  | **Stats:** | **34.306 ± 0.31** | **128.818 ± 1.93** |
| **T-Test  (H0: μ1=μ2)** | 0.000007250503305 |  |  | **T-Test  (H0: μ1=μ2)** | 0.000000008539177646 |  |

|  |
| --- |
| 1. In **Matrix::dot** we can remove placeholder called sum and change the value directly in the matrix cell.   This unfortunately **increased** time by 2sec…   1. Correcting types (**does auto slow down the process?)** 2. Another improvement we can try is suing PGO – profile guided optimization.   Even though it reduced *self-percentage* for some of the methods (no method has % higher than 10) it increased self % for other parts of the program and consequently increased the time to 35sec, also not good   1. Methods inlining. We are going to put the train function into **main** improve the performance this way.   Didn’t help much and also increased time   1. Changing the matrix::dot by: 2. Changing input as a matrixB and resutlPlaceholder |
| 1. Each matrix multiplication part of the code changed into |
| It did not improve performance much, the average is around 33 seconds because we are doing exactly the same thing but outside the loop   1. Copying just twice in NeuralNetworks::learn and NeuralNetworks::Classify       And overwriting matix::dot method     1. That make the algorithm to run 38s on average on in user time and **up to 2 minutes in elapsed time** |

## 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.

|  |
| --- |
| Even though I used various approached to improve the methods, none of them seems to be working. Using predefined matrix as a results template, slowed down the whole process significantly, probably by constant need of resizing it. Honestly, we could have expected it a bit... What is more surprising, is that we did not see any decrease in memory. Even though we dropped some of the variables the overall memory increased. Another surprise is a significant increase in elapse time, by almost 4 times! That is very bad! I do not know how else it can be improved. |