**High Performance Computing**

# Homework #2: Part B

**Due: Saturday February 17 2015 by 11:59 PM (Midnight)**

**Email-based help Cutoff: 5:00 PM on Mon, Feb 16 2015**

Maximum Points For This Part: 10

**Objective**

The objective of this part of the homework is used to use a given benchmark program to assess the performance impact of using the following two API methods to access individual elements in a `std::vector`:

- Using the `std::vector::at()` method (tip does bounds checking like Java/Python)
- Using the `std::vector::operator[]` method (tip does not do bounds checking)

## *Instructions:*

1. Download the supplied benchmark program and study the code carefully. See if you are able to answer the following questions:
   a. How and why was the test vector size chosen?
   b. Why does the benchmark repeat the test many times?

2. Ensure the benchmark is calling the appropriate method, i.e., sum or sum_at, depending on the API method you would like to test.

3. Compile the program with optimizations enabled (eg: `-O3` for `gcc` or `-fast` for `icpc`)

4. Ensure you use an interactive job on Red Hawk to record timings and fill in this report.

5. Once you have filled-in the report, save it as a PDF file and submit.

**Name:** Yan Yu

## *Experimental Platform*

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

| ***Component*** | ***Details*** |
|---|---|
| CPU Model | Intel<R> Xeon<R> CPU |
| CPU/Core Speed | 2.67 GHz |
| Main Memory (RAM) size | 247225392k |
| Operating system used | Linux 2.6.32-279.14.1.e16.x86_64 |
| Interconnect type & speed (if applicable) | |
| Was machine dedicated to task (yes/no) | |
| Name and version of C++ compiler (if used) | gcc-4.9.2 |
| Name and version of Java compiler (if used) | Javac 1.7.0_13 |
| Name and version of other non-standard software tools & components (if used) | |

## *Performance Analysis*

The benchmark program mainly uses a vector of size 1000000 to test the performance of the hardware, how different compiling options can affect the runtime performance. And at the same time, to monitor the difference between vector::at() and vector::operator[] methods.

Document the statistics collated from your experiments conducted in the table below. Delete the first row with fictitious data included just to illustrate an example.

| **`std::vector` Element Access Mode** | **User Time (sec)** | **Elapsed Time (sec)** | **%CPU** | **Max resident size (KB)** |
|---|---|---|---|---|
| Using `at` method (#1) | 0.4 | 00.41 | 98 | 20416 |
| Using `at` method (#2) | 0.38 | 00.39 | 98 | 20400 |
| Using `at` method (#3) | 0.40 | 00.41 | 99 | 20416 |
| **Averages (of 3 runs)** | **0.39** | **0.40** | **98.3** | **20411** |

| **`std::vector` Element Access Mode** | **User Time (sec)** | **Elapsed Time (sec)** | **%CPU** | **Max resident size (KB)** |
|---|---|---|---|---|
| Using `operator[]` (#1) | 0.38 | 00.39 | 98 | 20416 |
| Using `operator[]` (#2) | 0.38 | 00.38 | 98 | 20416 |
| Using `operator[]` (#3) | 0.38 | 00.39 | 99 | 20416 |
| **Averages (of 3 runs)** | **0.38** | **0.387** | **98.3** | **20416** |

Using the above chart develop a report (10 sentences) discussing the following performance aspects (use as much space as needed):

- What is the functional difference between the use of at() method versus operator[]?
- What is the performance difference between the two approaches?
- When should a programmer use one versus the other?
- What are the implications on other languages (such as Java/Python) with references to accessing values in a `vector`-like data structure (such as: `ArrayList` in Java)

The functional difference between at() and operator[] methods is that at() method will do boundary check but [] operator will not. So it would take additional time to do the boundary check job. This can be seen from the two tables above, when we use at() method, it takes a little bit longer to finish the task, while if we switch to [] operator, the time is consistent and shorter than at() method.
To use these two methods properly, programmers must be aware of the context, if we know that our program will not cause boundary problem, or performance is the primary goal, we should use [] operator, on the other hand, if safety is primary concern and we don't know if we will cause any boundary problem, we should use at() method.
Other language like java which provides ArrayList data structure that is pretty much the same with vector in C++, doesn't provide operator [] method for you, and we need to use

the method get() which is like at() method to access element to avoid the boundary problem, which slows down the performance sometimes.