**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:
   1. How and why was the test vector size chosen?
   2. 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:** |  |

## Experimental Platform

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

|  |  |
| --- | --- |
| Component | Details |
| CPU Model |  |
| CPU/Core Speed |  |
| Main Memory (RAM) size |  |
| Operating system used |  |
| Interconnect type & speed (if applicable) |  |
| Was machine dedicated to task (yes/no) |  |
| Name and version of C++ compiler (if used) |  |
| Name and version of Java compiler (if used) |  |
| Name and version of other non-standard software tools & components (if used) |  |

## Performance Analysis

|  |
| --- |
| Instead of this text, briefly (4 to 5 sentences) describe the functionality of the benchmark program based on your analysis. |

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) |  |  |  |  |
| Using at method (#2) |  |  |  |  |
| Using at method (#3) |  |  |  |  |
| **Averages (of 3 runs)** |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **std::vector Element Access Mode** | **User Time**  **(sec)** | **Elapsed Time**  **(sec)** | **%CPU** | **Max resident size**  **(KB)** |
| Using operator[] (#1) |  |  |  |  |
| Using operator[] (#2) |  |  |  |  |
| Using operator[] (#3) |  |  |  |  |
| **Averages (of 3 runs)** |  |  |  |  |

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)

|  |
| --- |
|  |