High Performance Computing

Homework #3: Part B

Due: Saturday February 24 2015 by 11:59 PM (Midnight)

Email-based help Cutoff: 5:00 PM on Mon, Feb 23 2015

Maximum Points For This Part: 10

Objective
The objective of this part of the homework is to develop a simple benchmark C++ program to compare the performance of using methods vs. lambdas in algorithms.

Background:

In C++ many standard algorithms such as: `std::sort`, `std::count_if` accept predicates to which you may pass methods or specify lambdas. Recollect that in computer science, a "predicate" is a mathematical function that takes zero or more parameters and returns a Boolean value (i.e., true or false). Predicates that accept one parameter are called unary predicates while predicates that accept two parameters are called binary predicates.

Scientific question:

Is there a performance difference between using a function versus supplying a lambda for a predicate?

Hypothesis:

Using a lambda would be faster as lambdas can be effectively inlined (thereby avoiding overhead of a function call) and the compiler can better optimize lambda.

Instructions:

This homework requires you to design an experimental benchmark to test the aforementioned hypothesis and conduct experiments to accept/reject the hypothesis. Complete the report (in this document), save it as a PDF file, and upload it to Niihka.

Here are a few tips to help design your benchmark:

1. Working with your `std::sort` algorithm would be effective (because of its time complexity arising from the number of times the predicate is used for comparisons)
2. Ensure your benchmark runs for about 10 seconds with optimizations (`-O3`).
3. Don't overcomplicate the benchmark. A few lines of code can be more useful than 100s of lines of code.

Name: Yan Yu

Experimental Platform

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

Component	***Details***
CPU Model	Intel core i7
CPU/Core Speed	2.5 GHz
Main Memory (RAM) size	16GB
Operating system used	Mac OS X Yosemite
Interconnect type & speed (if applicable)	n/a
Was machine dedicated to task (`yes/no`)	Yes
Name and version of C++ compiler (if used)	clang
Name and version of Java compiler (if used)	Javac 1.8.0_20
Name and version of other non-standard software tools & components (if used)	

Benchmark Design

The benchmark program I designed is basically generating 150,000,000 random integers which are stored inside a vector, and then sort all of them by using std::sort method.

Experiments and Observations

Briefly describe how the experiments were conducted (indicate commands used and details on the jobs that were used to conduct the experiments)

The experiment is conducted by recording the user & elapsed times of the program, the command being used is /usr/bin/time –v ./benchmark. Several metrics were recorded – user time, elapsed time, % CPU, Max Resident size.
Each version of the program were ran 3 times in order to get a average time measurement.

Observations:

Document the statistics collated from your experiments conducted (you may use tables from previous reports or create your own):

`std::sort` with Lambda	User Time (sec)	Elapsed Time (sec)	%CPU	Max resident size (KB)
Using `lamda` (#1)	17.34	17.58	99	2348288
Using `lamda` (#2)	17.29	17.43	99	2348288
Using `lamda` (#3)	17.28	17.39	99	2348288
Averages (of 3 runs)	**17.30**	**17.47**	**99**	**2348288**

`std::sort` with function pointer	User Time (sec)	Elapsed Time (sec)	%CPU	Max resident size (KB)
Using `function ptr` (#1)	24.58	24.77	99	2348272
Using `function ptr` (#2)	25.17	25.55	99	2348288
Using `function ptr` (#3)	25.08	25.42	99	2348272
Averages (of 3 runs)	**24.94**	**24.25**	**99**	**2348277**

Results and Conclusions

Using the above data develop a report (about 10 sentences) discussing the performance aspects and conclude if you accept or reject the hypothesis.

Based on the observation from the above charts, using lambda is much faster than using function pointer, which is abut 30% performance boost seen from the use & elapsed times. The reason why lambda is faster is because that compiler will insert the piece of lambda function code into the std::sort function, which actually modified the sort function, and using function pointer is just adding an extra function call, which in assembly is another jump. So the extra jump made the sort function by using function pointer slower than by using lambda which doesn't add an extra function call.

Submission

Once you have completed your report upload the following onto Niihka:

1. This report document (duly filled) and saved as a PDF file with the naming convention `MUID_Homework3_PartB.pdf`.
2. The benchmark program that you have developed named with the convention `MUID_Homework3_PartB.cpp`.