High Performance Computing

Homework #3

Due: Tuesday March 22 2016 by 11:59 PM (Noon)

Email-based help Cutoff: 5:00 PM on Mon, March 22 2016

Name: Henry Ni

Description

The objective of this benchmark (using the Ackermann function with inputs 7, 10, 13 and 15) was to compare and contrast the execution behavior of Java, Python and C++ in terms of CPU and memory usage.

Experimental Platform

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

Component	***Details***
CPU Model	Core i7-4790
CPU/Core Speed	800 MHz
Main Memory (RAM) size	8082728 kB
Operating system used	Ubuntu SMP
Interconnect type & speed (if applicable)	n/a
Was machine dedicated to task (`yes/no`)	Yes, but with minor load from general purpose software and system processes
Name and version of C++ compiler	g++ -std=c++11

Name and version of Java compiler	javac
Name and version of Python enviroment	Python 2.7.6 (GCC 4.8.2)
Name and version of other non-standard software tools & components (if used)	

Runtime Observations

Record the runtime statistics collated from your experiments conducted using the specified command-line arguments below.

Command-line Argument	C++ (-O3)		C++ (-O3 and PGO)		Java		Python	
	Elapsed time (sec)	Peak memory (KB)	Elapsed time (sec)	Peak memory (KB)	Elapsed time (sec)	Peak memory (KB)	Elapsed time (sec)	Peak memory (KB)
7	.00	1264	.00	1272	.05	30080	.08	5720
7	.00	1264	.00	1268	.04	31708	.13	5716
7	.00	1268	.00	1268	.04	31956	.07	5720
7 (Avg)	.00	1265.33	.00	1269.3	.043	31248	.093	5718.67
10	.02	1336	.01	1340	.18	27616	9.77	11728
10	.03	1340	.02	1344	.17	30228	9.93	11732
10	.02	1340	.02	1344	.17	28316	10.45	11732
10 (Avg)	.023	1338.67	.016	1342.67	.173	28720	9.84	11730.67
13	1.41	1836	.93	1844	9.79	33304	988.29	60700
13	1.41	1840	.92	1840	9.25	31556	1019.54	60700
13	1.41	1840	.91	1844	9.14	31244	1013.10	60700
13 (Avg)	1.41	1838.67	.92	1842.67	9.39	32034.67	1006.98	60700
15	23.60	3554	15.37	3548	197.99	35372	Optional	Optional
15	23.68	3548	15.50	3548	199.81	35372	Optional	Optional
15	23.59	3544	15.41	3548	197.32	35352	Optional	Optional
15 (Avg)	23.62	3548.67	15.43	3548	198.37	35365.3	-	-

Inferences & Discussions

Now, using the data from tables document your inferences contrasting the Ackermann's function implemented in Python, Java and C++. Using the evidence gathered through the experiments (use as much space as you need). Compare and contrast on the performance trends and memory usage in the languages. Compare and contrast source code differences between the three different languages. In addition comment on which programming language would be more suitable for:

Big-data analytics that involves calling many methods and holding as much data as possible in memory.

For scientific visualization and gaming

Complex operations such as those used in bioinformatics, neurobiology, etc.

Energy-efficient computing for mobile devices (smartphones, tablets, laptops) and discuss their advantages.

Using the Ackermann benchmark, a few things are immediately apparent: C++ runtime is by far the quickest (especially when optimized and using PGO), Python's memory usage is extremely high compared to the other languages while Java has the most consistent memory usage.

The extremely (comparatively) fast runtimes using PGO is due to the compiler's knowledge of how the Ackermann program will run on the hardware, gathered from executing the program before for various inputs. The improved optimization and knowledge of where branching will take place enables the CPU to more efficiently perform the task at hand and as a result, bring down runtime and memory usage.

Runtime features an exponential trend upwards from inputs 7, 10, 13 and 15 for all three languages. While the times themselves differ greatly, the increases in execution speed must then be explained by the nature of benchmark, that is the Ackermann function is vastly more punishing for inputs after 10. For languages like C++ and Java, the performance impact is not felt as strongly because they are compiled into machine code at some point in time, but for an interpreted language like Python then any increase in complexity will cause a huge increase in runtime. This is consistent with what we know about the "semantic gap", that the closer a language can come to interacting with the computing hardware directly, the more its compiler can optimize instructions for that particular configuration. This is where PGO comes in and takes that a step farther using past performance as a predictor for future runs and can reduce runtime by an additional 10% (vs. non-PGO enabled C++ program executions).

Memory usage for Java as mentioned earlier is fairly constant, always hovering around 30,000 kB which is most likely due to the way it runs within the JVM (the JVM takes x amount of memory and never exceeds that amount at the cost of additional runtime). In this respect, C++ and Python are similar in that their memory usage statistics both seem exponential than constant; seeing as the function in C++ uses twice as much memory between Ackermann(13) and (15) while Python seems to demand twice as much memory, then four times that going from Ackerman(7) to A(10) to (13).

Source code wise, only Python needed to adjust its memory limit and the Java implementation did not include a default value for n

in case there were no command-line arguments passed to the function. Other than that, there were no major differences between the three programs.

As stated above, when there are no arguments passed to the Java version there is no default value for n to be initialized to. The program does not crash, but the report generated from /usr/bin/time –v java –Xssm64m ackermann" apparently reaches 110% CPU utilization. Nothing computer-breaking, but strange regardless.

For the various fields where HPC is a focus (gaming/virtualization, big data, energy efficiency) it would seem like C++ is the clear-cut choice: well-written and optimized programs run faster and with a smaller memory footprint than Java and Python. Existing game engines often make use of C/C++ because it can be optimized so heavily, as well having more control over the system as a whole (memory allocation especially). The one area is *may* be appropriate to use Java from looking at the statistics here is with Big Data, since Java uses roughly the same amount of memory regardless of what Ackermann computation is being done. However, for especially large data sets it would probably make more sense to use a language that is memory efficient instead of one that is just consistently memory-hungry. Lastly, it should be said that a language that can run quickly and with the least amount of memory used (and one that takes advantage of the hardware fully) will be one that is the most efficient which in this case is C++.