High Performance Computing

Homework #3

Due: Wednesday Sept 22 2021 Before 11:59 PM (Midnight)

Email-based help Cutoff: 5:00 PM on Tue Sept 21 2021

Name: **John Doll**

Description

The objective of the experiment and what the micro-benchmarks are designed to do is to compare the computational semantic gap of Java, C++, and Python. These benchmarks are specifically designed to compare the overhead of method calls. This is important because if a language does not support efficient method calls, it does not make much sense to build complex programs or use them for complex problems.

Experimental Platform

The experiments documented in this report were conducted on the following platform (**Note**: commands to obtain the information are provided in the template version of this report):

Component	***Details***
CPU Model	Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz
CPU/Core Speed	2.40GHz
Operating system used	Linux pitzer-login01.hpc.osc.edu 3.10.0-1160.36.2.el7.x86_64 #1 SMP Thu Jul 8 02:53:40 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux
Name and version of C++ compiler	10.3.0
Name and version of Java compiler	11.0.8
Name and version of Python environment	3.7.4

Runtime Observations

Record the runtime statistics collated from your experiments conducted using the specified command-line arguments below. Job information should be placed in the appendix at the end of this report.

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)**
10	0.08	2300	0.04	2320	0.26	42228	10.71	14656
10	0.03	2296	0.04	2320	0.24	44396	9.33	14536
10	0.03	2300	0.04	2320	0.25	58704	9.34	14576
10 (Avg)	**0.046**	**2298.67**	**0.04**	**2320**	**0.25**	**48442.67**	**9.79**	**14589.33**
13	2.01	2640	1.75	2864	5.00	64124	15:28.40	64848
13	1.99	2640	1.73	2864	4.95	42552	15:28.85	64844
13	1.99	2640	1.73	2864	5.06	50308	15:30.56	64848
13 (Avg)	**1.99**	**2640**	**1.73**	**2864**	**5.00**	**52328**	**15:29.27**	**64846.67**
15	32.11	4344	30.11	5256	1:20.68	45200	4:10:04	240172
15	32.20	4344	30.09	5256	1:20.72	41872	4:06:30	240152
15	32.17	4344	30.06	5256	1:20.42	47604	4:03:41	240300
15 (Avg)	**32.16**	**4344**	**30.09**	**5256**	**1:20.61**	**44892**	**4:06:45**	**240208**
16	2:09.69	6620	2:05.19	8440	6:10.58	65964	Optional	Optional
16	2:09.74	6620	2:05.40	8440	6:02.53	53416	Optional	Optional
16	2:09.71	6620	2:05.23	8440	5:52.58	58672	Optional	Optional
16 (Avg)	**2:09.71**	**6620**	**2:05.27**	**8440**	**6:01.90**	**59350.67**	Optional	Optional

Inferences & Discussions

Now, using the data from tables and the discussion points from the project grading rubric, document your inferences in the space below. Ensure you address all of the 12 aspects mentioned in the grading rubric. The discussion will be critically graded and only the comprehensive / complete discussions will be assigned full score.

We can see that the C++ PGO times for 15 are slightly faster than the C++ times with just O3 optimization. This is because we run the C++ PGO program a couple times with different inputs, then send it off to be time tested after. This means that the PGO optimizer knows how the program runs and can optimize based on that, increasing the runtime efficiency, which O3 optimization cannot do.

The run time and memory usage for C++ with both compilation settings and for Python all are exponential. As the argument input increases by just a couple, the time it takes to run the program and the memory needed in order to do the program are both increased exponentially in order to meet the demands. For Java, the story is a bit different. Java run time, like C++ and Python, is exponential. However, the memory used by Java as the command line argument increases is not exponential. It could be regarded as either linear or polynomial. I think with more test runs, we could narrow it down. It seems to have spikes in memory usage when running programs which make interpreting the pattern more difficult.

The semantic gap is clearly visible between C++, Java and Python even in the first run with the command argument of 7. The C++ language is written very close to the computer architecture which means that it has a higher efficiency when running the code because it doesn't have to do as many operations. It is compiled before it is run meaning that it translates the object code to assembly and optimizes it before the program is run. Python does not compile before it is run so there is no optimization that happens when it is run. Java does what is called Just In Time compilation which means that it compiles the .class file right before it is run meaning it still has the overhead of an interpreter in order to go straight from compilation to running the code.

C++ and Java are written very similarly for the microbenchmarks being performed. Python has an extra statement to change the stack sizes in powers of 2 in order to make it consistent with C++ and Java. Otherwise, the programs are all written very simply and look identical.

To run the C++ -O3 and PGO benchmarks, we just run the C++ file name. To run the Java benchmarks, we must specify the amount of memory we would like to be allocated for that run of the program. To run python, we must run it specifically with /usr/bin/time and cannot use the for loop that is used for the C++ and Java files. Because of this, we must also specify the input number manually for each run. Additionally, because the Python program takes so long to run, we must also specify that the default time before time out is much longer since it will not complete before then.

If we are looking to run the quickest program and save the most time, then C++ with O3 and PGO compilation is what we should use. If we want the language that makes the best use of memory, then we should use C++ with O3 compilation. If we are looking at the most energy efficient then we should go with C++ with O3 and PGO optimization because it takes the least amount of time to run and only a little bit more of memory than C++ with just O3 optimization.

Python is clearly the worst environmental language of the three we studied in this homework. It takes and incredibly long time to run and a massive amount of memory. Java is a step better in both time taken and memory used, but C++ is the clear winner in both time spent running and memory used. This seems backwards from the industry practices of promoting Java and Python. Python gets credit for having an extensive amount of libraries that can be imported in order to make creating your program much simpler. Java has been widely used you a while and is generally thought of as easier to learn than C++. So while C++ programming is clearly superior to Java and Python in terms of program duration and memory used, it gets stereotyped as hard to learn and not as versatile compared to Java and Python.

Appendix

Copy-paste the SLRUM job script that you used to collect the runtime statistics in the space below:

The echo lines with the run commands were commented in and out as I needed when running the tests, but this was my final script.

```bash
#!/bin/bash

#SBATCH --account=PMIU0184
#SBATCH --job-name=hw3_cpp_java
#SBATCH --time=06:00:00
#SBATCH --mem=4GB
#SBATCH --nodes=1
#SBATCH --tasks-per-node=1

# To keep the benchmarks more upto date, we load the more recent
# versions of the tools available on the cluster
module load gcc-compatibility/10.3.0 java/11.0.8 python/3.7-2019.10

# Function to run given command 3 times
function run3() {
    cmd="$*"
    for arg in 10 13 15 16; do
        echo "----------------------------------------------"
        echo "Running 3 reps of ${cmd} ${arg}"
        for rep in `seq 1 3`; do
           /usr/bin/time -v ${cmd} ${arg}
        done
    done
}

# This script assumes the benchmarks have been precompiled.
```

```
#echo "==============================================="
#run3 ./ackermann

#echo "==============================================="
#run3 ./ackermann_pgo

#echo "==============================================="
#run3 java -Xss16m ackermann

# echo "==============================================="
#Submit python jobs seperately to finish sooner.
/usr/bin/time python3 ackermann.py 15

# End of script
```