Gehrig Weber

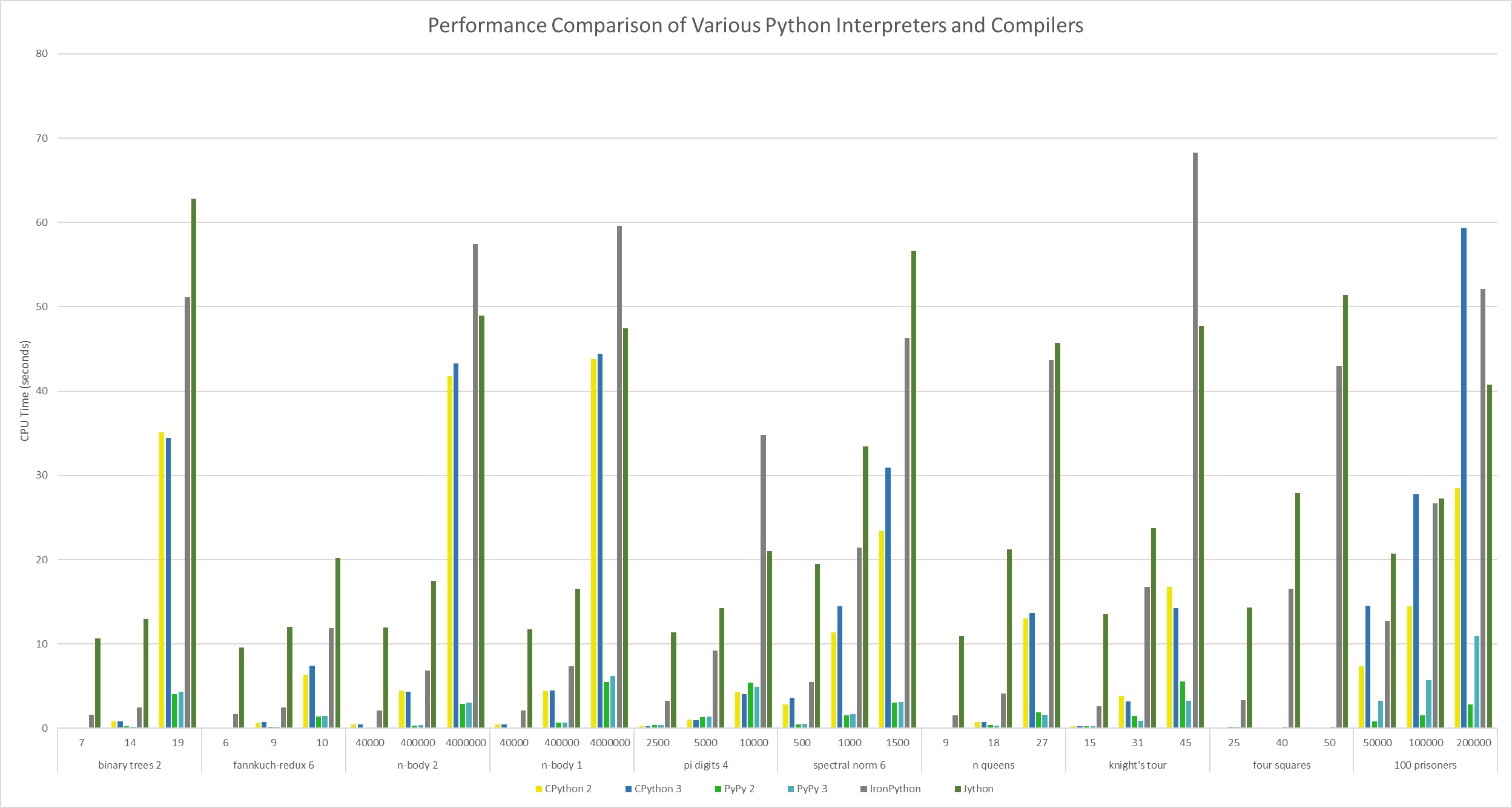
Prof. Krintz

Runtime Systems

June 8, 2020

Comparisons in Performance Between Python Implementations

This project was intended to investigate various Python implementations and evaluate them, based primarily on performance. The implementations are: CPython 2 and 3 (the default Python implementations), PyPy 2 and 3 (just-in-time (JIT) Python compilers), IronPython (an implementation of Python 2 on the .NET runtime, or Common Language Runtime (CLR)), and Jython (an implementation of Python 2 on the Java Virtual Machine (JVM)).



Ten benchmark programs were used to evaluate the implementations’ performance, as seen in the graph above. 3 sets of 5 trials were run with each program. Each set used a different value of N. The results are in CPU time, the total time spent executing the program. They were obtained by adding the user time and system time taken for each of the 5 trials, then averaging those sums. The benchmark programs used will now be detailed. Binary Trees 2 is a program that allocates and deallocates numerous perfect binary trees of increasing depth, from 4 to N. Fannkuch Redux 6 iterates through N! permutations of 1 to N. Let the first number in the sequence be X. For each permutation, the first X numbers are reversed repeatedly, until X = 1. The maximum number of reversals, or flips, is returned at the end. N-Body 2 is a simulation of stellar bodies, which involves performing a constant number of calculations N times. N-Body 1 is identical to N-Body 2, except it uses “\*\* 0.5” instead of “math.sqrt()” to calculate square roots. Pi Digits 4 is a program that prints the first N digits of pi. Spectral Norm 6 calculates the spectral norm of an N-by-N matrix. N Queens uses backtracking to return all possible solutions of the N queens problem: Can N queens be placed on an N-by-N chess board without threatening each other?. Knight’s Tour implements Warnsdorff’s algorithm to return an open tour of an N-by-N chess board with a knight, if one exists. The starting point can be changed, but it was always the top left corner of the board for the trials. Four Squares outputs all unique solutions to the four squares problem, or all combinations of *a, b, c, d, e, f, g* such that *a* + *b* = *b* + *c* + *d* = *d* + *e* + *f* = *f* + *g,* where the values are selected from the range 0 to N without repetition. This involves going through N choose 7 (N!/(N - 7)!) iterations, and doing constant work at each. 100 Prisoners involves going through N iterations of a simulation of the 100 prisoners problem, in which 100 prisoners can each open a maximum of 50 out of 100 drawers in an attempt to find their number. No communication is allowed. All prisoners must find their number in order for any of them to win.

According to the data, the Python implementations, in approximate order from best to worst performing, are PyPy 2, PyPy 3, CPython 2, CPython 3, Jython, and IronPython. The reasons for this will now be examined.

PyPy uses a tracing JIT compiler, which, like most compilers, is faster than CPython’s interpreter. It ends up being faster than IronPython and Jython’s respective JIT compilers, as well. Now, PyPy 2 is faster than PyPy 3 largely because CPython 2 is faster than CPython 3. Outside of performance, PyPy also has an efficient garbage collector that results in programs using much less memory than in CPython. Furthermore, PyPy 2 and 3 are extremely compatible with Python 2.7 and Python 3.6.9’s core languages, respectively, establishing PyPy as one of the best alternative Python implementations.

CPython’s use of an interpreter should make it the slowest Python implementation, yet it ends up being faster than IronPython and Jython, even though their runtimes make use of JIT compilers. Between the two versions, CPython 2 is faster than CPython 3 due to the fact that, in CPython 3, an int is what a long was in CPython 2, and longs do not exist. Thus, ints in CPython 3 are twice as big as ints in CPython 2, and operations with ints take much longer in CPython 3.

Jython is a Java implementation of Python that runs on the JVM, and thus, has access to all of Java’s libraries. Performance-wise, it is very similar to IronPython, a C# implementation of Python that runs on the CLR, and thus, has access to all .NET libraries. Even though they both have their respective JIT compilers, they are the two slowest implementations due to inefficient translation into bytecode. Much of the information about this topic is outdated and contradictory, but the main conclusion that was drawn is that if it was known exactly why the translation was inefficient, it would have been rectified. Between the two, Jython is a bit faster than IronPython because the JVM only compiles “hot” methods and interprets the rest, while the CLR compiles everything.

However, although each implementation is generally faster or slower than the others, there are numerous exceptions to the average order. For instance, Jython and IronPython suffer from an about 9.5 and 1.5 second JVM and CLR startup time, respectively. In practice, when one needs to run a program, their runtime has probably already been active for quite a while, effectively eliminating the startup time. This puts Jython firmly ahead of IronPython and closer to CPython performance-wise, although CPython is usually still faster.

In programs that are mostly loops and/or run for an extended period of time, such as N-Body 1 and 2, Spectral Norm 6, and 100 Prisoners, Jython and IronPython catch up to and may even exceed CPython in terms of performance. This is due to their respective JIT compilers, which prove faster than CPython’s interpreters in the long run. This, as well as Jython and IronPython’s access to the Java and .NET libraries, respectively, gives the two viability as alternative Python implementations.

In programs that involve numerous complex calculations, such as Spectral Norm 6 and especially 100 Prisoners, CPython 3 is much slower than CPython 2. This is due to the difference in the size of their ints, which results in operations with them taking longer in CPython 3 than in CPython 2. This time difference is amplified by the quantity of calculations in certain programs.

In N-Body 1, which uses “\*\* 0.5” to calculate the square root of a number, PyPy is much slower than in N-Body 2, which uses “math.sqrt()”. “\*\* 0.5” is known to be slower than “math.sqrt()” because the latter is a call to a C function, yet this performance difference is not found in any other implementation examined here. Only in PyPy is “\*\* 0.5” noticeably slower than “math.sqrt()”.

In programs that use global variables, such as Pi Digits 4, CPython ends up being faster than PyPy. Global variables cause a noticeable decrease in performance in Python in general, but this effect is exacerbated in PyPy, for reasons unknown.

In Knight’s Tour, CPython 3 and PyPy 3 are faster than CPython 2 and PyPy 2, respectively. This is due to the program’s use of “copy.deepcopy()”, which is almost twice as slow in Python 2 than it is in Python 3, as the latter uses a more efficient version of the function. Although Python 3 is generally slower than Python 2, the fact that the former is being constantly updated and improved does give it certain advantages in cases like these.

In Four Squares, execution is trivial for all of the Python implementations except for Jython and IronPython. This is because the program, which finds all unique solutions to the four squares problem, also outputs those solutions to the console. Printing to the console is extremely slow for the JVM and the CLR, which results in Jython and IronPython performing much worse than any other Python implementation. The actual implementation of Jython and IronPython themselves has nothing to do with the slowdown in this case.

From the data, it is concluded that PyPy 2 or 3 should be used whenever possible, due to their speed and respective compatibility with Python 2 and 3. However, as the default implementation, CPython is competent in terms of performance, at least when compared to other Python implementations. As for Jython and IronPython, if one requires either the Java or .NET libraries, respectively, then their performance is perfectly acceptable. Each of the Python implementations has their advantages and disadvantages, and none of them completely outclass any of the others.