Russian Peasant Multiplication

by Matt Breckon

Build and Run

To build the project for Fortran, Ada, Cobol, and C, run:

```
$ make
```

Two Python scripts were included to benchmark the compiled programs: **benchmark.py** and **benchmark_py3.py**. Originally, the benchmark.py script was developed on my local version of python 2.7.11. However, the lab computers run Python 2.6.6 which does not support the function check_output() within the subprocess module; the lab computers also run Python 3.1.3 which does support this function.

To run the benchmarking script on lab computers, run:

```
$ python3 benchmark_py3.py
```

To run the benchmarking script on python 2.7.x, run:

```
$ python benchmark.py
```

Within the main execution of the script, variables m and n can be modified to test different values to be multiplied.

Lastly, to simply execute Fortran, Ada, Cobol, or C after compilation, run:

```
$ lang/recursive m n
$ lang/iterative m n
```

Discussion

Benefits and Limitations

Working with legacy languages over the course of this semester, C over the course of years, and Python over the past year, very obvious benefits and limitations begin to appear when jumping from one language to another.

Initially, I see little benefit from my view point when it comes to using Ada or Fortran in comparison to C except for memory management. This particular project did not require dynamic use of variables and as such, this benefit was not immediately relevant. However, one benefit I enjoy about Fortran in comparison to Ada or C is how many more functions are immediately available without the need to import libraries or modules. For example, the use of getarg(), read(), and write() within Fortran out of the box is a small but nice benefit

where C requires the #include header and Ada requires the import of modules for this functionality.

A benefit of C compared to Ada is the use of pointers. Ada functions, at least with the compiler used for this course, only allow parameters as immutable variables.

Cobol has no benefits and I would love to see someone try to explain to me one; I can't even utilize functions without making them external modules. However, with this project I used the paragraph syntax in a "hacky" way to make an out-of-reach paragraph beyond the stop run statement act as a function requiring a perform statement to call upon it.

Also, many people would argue that this is a benefit, but I prefer dynamically typed language myself. So in my personal opinion, all of the languages aside from Python suffer from type declarations. Not being able to utilize certain modules in Ada such as bitwise shift because it does not support Ada's built in long_integer type is an immediate hindrance.

Overall, Python is the most beneficial language due to ease of use. Writing this algorithm in Python resulted in extremely short and straightforward code. However, as is talked about below, I am fully aware of the overhead Python has.

Efficiency

As far as efficiency of programs goes, the results were exactly what I expected. C, on average, was the fastest version of the algorithm for both recursive and iterative implementations. Ada, Fortran, and even Cobol were either just shy of C's performance or only a few milliseconds off. Python, on the other hand, was extremely slow compared to all other versions of the algorithm. I had already known Python had performance issues but it was interesting to see C recursively finish an execution in 5.56ms compared to Python finishing in 23.32ms.

Initially, I had developed the benchmark completely naïve importing the Python algorithms into the benchmark script. This resulted in the fastest execution times. However, to fairly compare the results the algorithms needed to be called identically to the other programs through Python's subprocess module acting as a shell. It is worth noting that Python is not compiled and instead ran with the Python interpreter while the other languages are completely compiled down to an executable.

Use of Large Numbers

After a bit of digging, I found no discernable difference in languages utilizing large numbers for the implementation of this project. For example, C has type "long long", Ada has type "long_integer", Fortran allows declaration of different kinds of integer, and Cobol utilizes variable declaration of digit length. Python, being dynamically typed, requires no

declaration of a long integer and instead dynamically converts integers that exceed 32 bits to long integers.

Slowest Language

As previously mentioned, Python was the slowest language due to lack of compilation and instead is ran as a script through the Python interpreter. However, when it comes to the slowest language *to program in*, first place prize goes to Cobol.

Results

		Fortran	Ada	Cobol	C	Python
m = 23958233 n = 5830	**Rec.**	8.33ms	7.46ms	N/A	5.56ms	23.32ms
	Iter.	6.33ms	5.96ms	8.52ms	5.01ms	24.54ms
m = 2345 n = 789	**Rec.**	5.24ms	4.87ms	N/A	5.63ms	24.29ms
	Iter.	6.97ms	5.04ms	5.98ms	5.26ms	24.21ms
m = 52395802 n = 423485329	**Rec.**	5.45ms	5.28ms	N/A	5.51ms	24.39ms
	Iter.	8.18ms	5.00ms	5.34ms	4.79ms	24.59ms
m = 2395825 n = 5830593	**Rec.**	6.99ms	6.44ms	N/A	6.36ms	25.44ms
	Iter.	6.65ms	7.99ms	8.70ms	8.17ms	21.45ms
m = 99999 n = 123	**Rec.**	5.12ms	5.10ms	N/A	6.73ms	24.14ms
	Iter.	8.68ms	6.15ms	7.38ms	4.99ms	24.42ms
m = 11 n = 3	**Rec.**	5.60ms	5.24ms	N/A	5.91ms	34.4ms
	Iter.	7.02ms	6.89ms	6.73ms	4.87ms	24.61ms

Interesting to note that Ada out performed C when multiplying smaller values recursively in one instance. Repeat occurrences of the same m and n values (m = 2345 and n = 789) has consistent results with Ada being the fastest program.