Kyle Hersey

0851973

CIS\*3190

Assignment 4: Report

To test the performance of each Ackermann program I ran them with the parameters M=4 N=1. This particular calculation is extremely computationally heavy, and is a better judge of performance for efficiency. With that in mind here are the perfomance test results for the fortran Ackermann programs. I ran each program using: time ./runme, and I take the user result as this does not take into account the time the program is waiting for me for input, and is closest to actual execution time.

Fortran

Recursive Ackermann time to run: 0m55.100s

Iterative Ackermann time to run: 1m28.991s

Although Fortran does not by default support recursion, it offers recursive functions which allow recursion to be very easily implemented. This also has the added benefit or making it very obvious when recursion is being used, as all recursion is labeled. The iterative algorithm was more difficult to implement, as Fortran doesn't have any prebuilt data structure for implementing a stack, and as such I had to use an implementation similar to the provided C code, using a large integer array to simulate the stack, and an integer variable to point to the top of the stack. As with all the languages, the Recursive algorithm is faster as it has less total function calls, and does not use any complex data types. In overall performance Fortran's recursive program comes in last behind C and Ada. In iterative timings Fortran comes in second behind C. While Fortran is not a language I am used to working with, I had a pretty easy time to throwing together these two programs. Fortran is not quite as efficient as C and Ada but it is very close. From a usability perspective Fortran is much better than Cobol, but is not as usable as either Ada or C, as even newer releases still contain a lot of outdated constructs.

C

Recursive Ackermann time to run: 0m48.556s

Iterative Ackermann time to run: 1m23.819s

C is a language I am quite used to and have a lot of experience programming with, and as such, throwing together the recursive Ackermann algorithm was quite easy. C is a language that supports recursion by default and does not add any extra hastle to writing recursive functions. The already provided iterative version makes use of C structs to create an array based stack implementation. Performance for C code is always good, and in this case the C recursive algorithm was just milliseconds faster than the Ada recursive algorithm. For the iterative algorithm, C comes in first, being the fastest again, beating Fortran by a couple seconds. C is definitely the best language performance-wise and I find it to be very usable, but this is probably because I have the most experience with it.

Ada

Resursive Ackermann time to run: 0m48.814s

Iterative Ackermann time to run: 8m59.296s

Ada is a language that supports recursion by default, and as such the provided Ada recursive algorithm and the C recursive algorithm are almost identical. In order to run my test I had to modify the provided code to run with M=4 N=1, and in the end it ran milliseconds faster than the C implementation. When writing my iterative implementation I used the built in Ada linked list package, which I used in a previous project as an easy way to implement a stack. While this made implementing the code extremely easy and fast, it had an extreme negative impact on the performance of the iterative algorithm, taking almost 9 minutes to complete in my test. This really demonstrates the how much faster the an array based stack implementation is, despite how a dynamic linked list implementation may be a “better” solution under many other circumstances. I didn't know what to expect from Ada performance wise, but based on the recursion algorithms it seems to be very, very close to that of C. Out of all the legacy programming languages I definitely enjoyed programming in Ada the most, and if I had even half the experience coding in Ada as C I would probably find it more usable than C.

Cobol

Iterative Ackermann time to run: didn't finish

As we know Cobol does not have any support for recursion which makes a recursive algorithm not an option. Cobol also doesn't seem to have any way to implement a stack as a data structure, but I really couldn't care to find out. My implementation of the iterative algorithm, ended up being pretty ugly looking, as I avoided using any functions, due to the amount of extra Cobol code I would have to write. In theory this should lead to a more efficient algorithm as function calls have a large overhead on performance. I used an array to simulate the stack, which another variable holding the position of the top of the stack. I hated every painstaking second of working with this horrible language, and hopefully I will never have to use it again in my life. Performance wise I have to put Cobol in dead last, as it was not able to complete my test after allowing it to run for over 20 minutes (Note: it does complete when M is 3 or less). This means that the Cobol implementation, which should have been the fastest of all the iterative ackermann programs due to a lack of function calls, is in fact slower than an algorithm using a dynamic linked list stack. Cobol was the least efficient and the least usable of all the languages used in this project, and I look forward to not using it in the future.