Assembly Sorter Project

Simon Munch
simun15@student.sdu.dk
DM548

November 6, 2016

Contents

1 My Quicksort

I chose to make a quicksort in this project due to its renown as a fast sorting algorithm, while also bieng a relativly simply and easy to code. It also bieng an in situ algorithm made it the obvious choice.

I origionally planned to get the middle element as the pivot for my sorter, by i ran into errors i couldnt fix, and so chose to use the topmost element as the pivot. As i wanted to limit the potential memory usage, i chose to have my sorter be called in a loop, instead of calling it recursivly, as i that way could keep a close eye on the stack and avoid it bieng expanded in a way i didn,t want it to.

I did run into problems with my pointers, as they seemed to leave the allocated memory and try to sort elements beyond the location of the input numbers, causing me to add more 'cmp' operations to ensure that they don,t crash the program. This does decrease my sorters performance, but it was a simple and easy fix, and so i accepted the resulting performance decrease.

2 Performance

The Quicksort algorithm is known to be slower than more simple sorts on smaller inputs due to its overhang, which can be seen on my sorter too, as the MCIPS value is very low when the amount of input numbers are small, and hardly increase untill the amount of numbers given as input reach the thousands.

Sorting million of numbers in only a few seconds (between 5 and 6 seconds) even when my code isnt the most clean and optimized. So even thought my Sorter is slower than more optimized code, for my first sorter in .asm, i think it did pritty well.

3 To Improve

My code makes too many comparisons, due to my carefullness with pointers. If i could decrease the amount of cmp i do to make sure pointers stay within the bounds of the allocated memory, my code would get a large perfomance increase. The pivot function could also be fixed, to give me the middle element as a pivot to avoid some of the worst case scenarios which might happen.

Also, in my code, I make a comparison a second time on elements which have just been swapped. This is unecessary, and could be avoided if a move

the pointers too when i swap, however as i fear out of bounds pointers, i decided not to, even thougth this might have given a decent perfomance increase when the inpus is very large.

My memory usage could also be improved, as once the function parse_number_buffer has been called, the memory for the origional input could be freed.