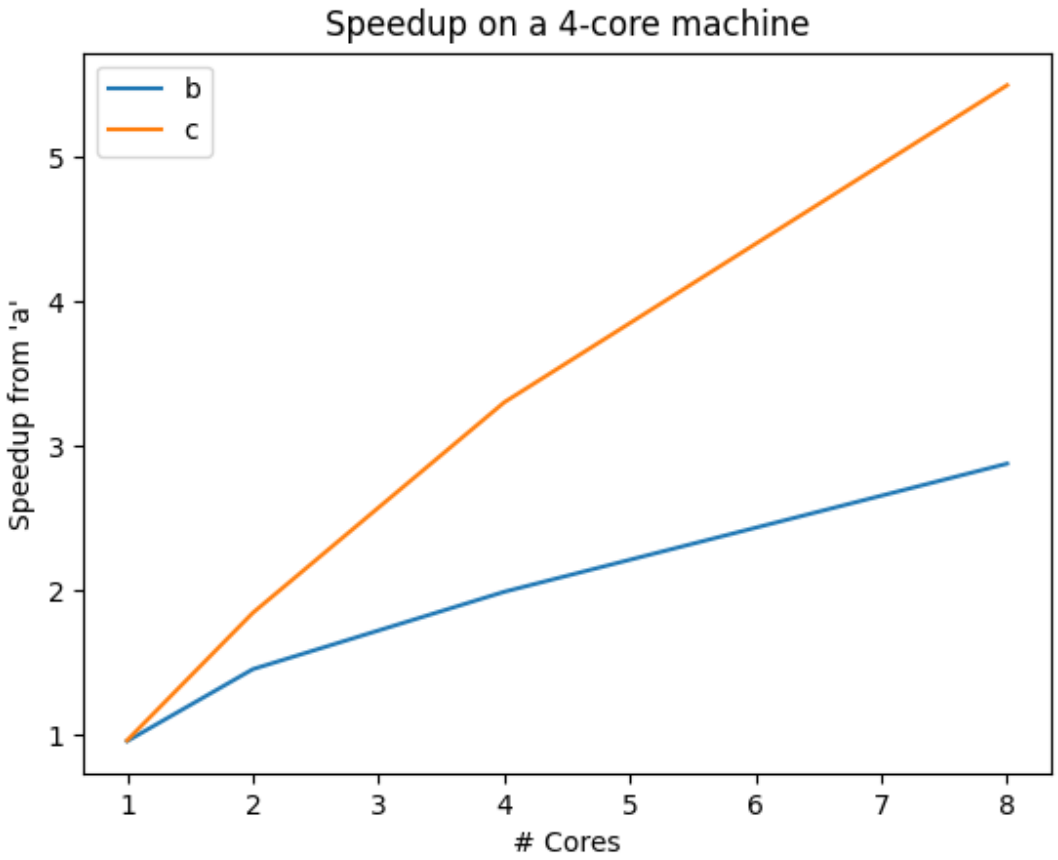
Ari ‘aith’ I.



In part 3 I use C++ threads to perform millions of addition operations in parallel. I tested 3 versions: 'a' having no multi-threading, 'b' having threads executing dependent addition instructions, and 'c' having threads utilizing the size-4 'chunking' used in part 2, as well as utilizing spatial locality to reduce cache misses.

As expected, the single-threaded version, 'a', is slowest. Version 'b,' breaks up the sequential addition found in 'a' by having multiple threads execute the parts of the array additions at the same time. With 8 threads on my 4-core machine I achieve a significant speed of 2.8x from 'a.'

'b' is constrained by its addition operations being dependent on the prior addition operation. 'c' utilizes chunks the addition operations into groups of 4 independent instructions. This resulted in a sizeable speedup from 'b,' but the real winner was considering spatial locality when accessing each array element. Adding array elements that were one address space away, rather than de-localized, resulted in a speedup of 4x from 'a' alone. With both optimizations, 'a' resulted in a whopping 5.5x speedup from 'a' (on a 4-core machine!)