Adam Wright

wrighada@oregonstate.edu

CS-475 Project-2

Written commentary

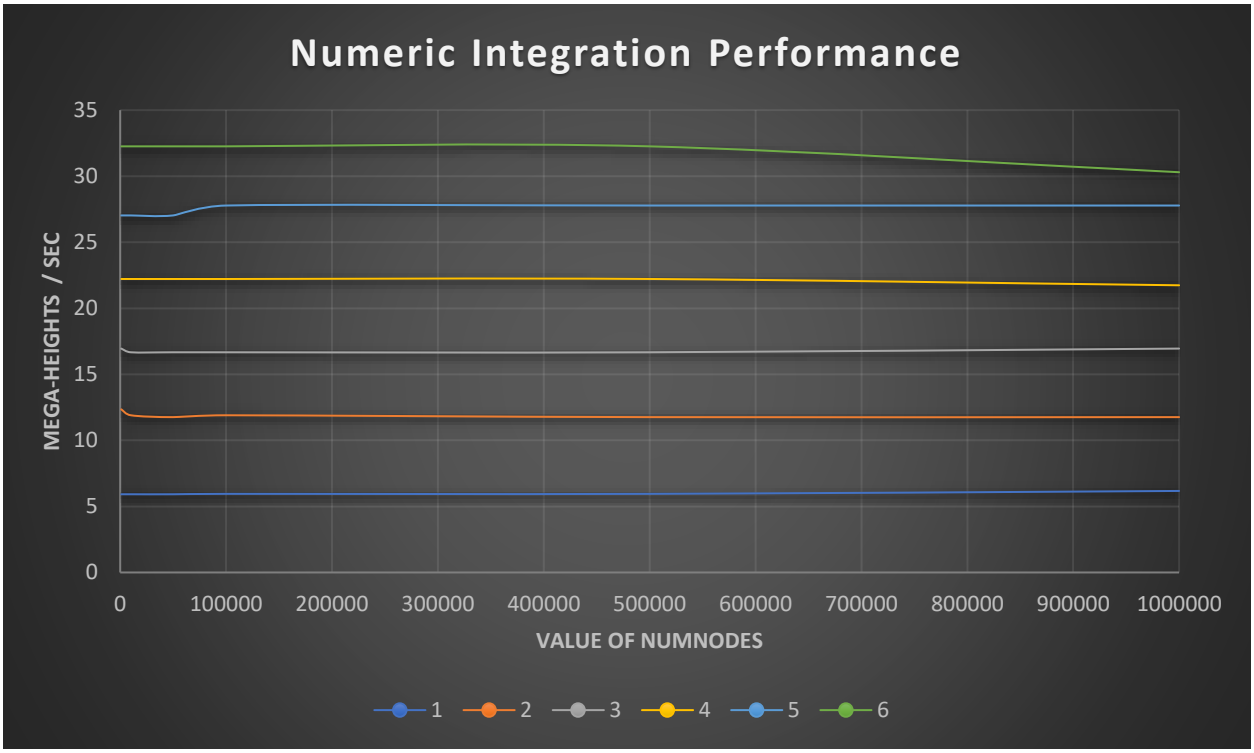
1. What machine was the test conducted on?

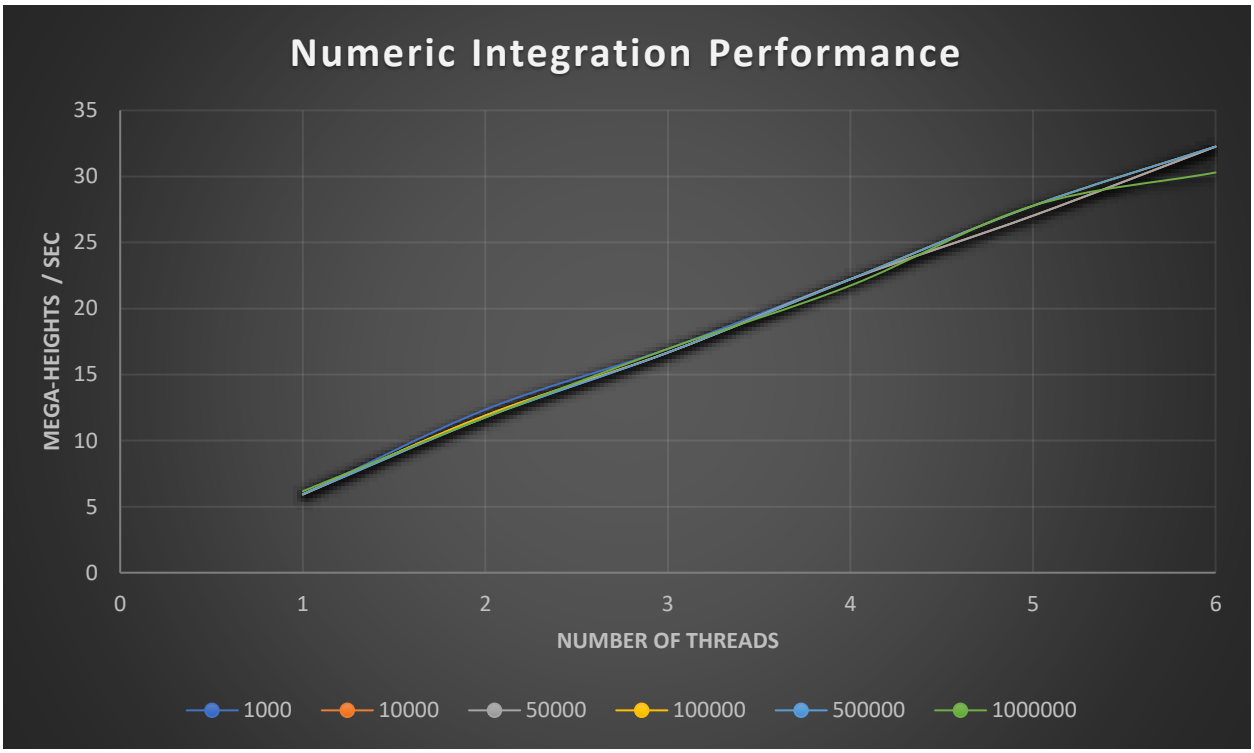
I used my local machine for the test. It is a 6 core/12 thread 3.33 GHz Xeon with 12MB of L3 cache. The operating system is Windows 10. The program was written in VS code with Git Bash set as the integrated Terminal. Git Bash uses mingw-64 as a compatibility layer for unix programs and this allowed me to script the running of the test in Bash and to compile each iteration of the program in the script with GCC set of -O3 optimization.

2. What do you think the actual volume is?

The volume number that I calculated was 6.48. This number seems reasonable because the volume of a sphere of radius 1 is 4.18 and the volume of a cube with the sides equal to 2 equals 8. This leaves the superquadratic volume of 6.48 right in between those numbers, as we would expect.

3. Show the performance in graphs





4. What patters are you seeing in the graphs?

> The performance of OpenMP with this algorithm is very stable. The performance does not depend on the value of NUMNODES, even when it's value goes as low as 10. The array size seems to have virtually no effect on the performance. The performance also scales very predictably with the number of cores. Each new core doubles the performance of the algorithm.

5. Why do you think that it is behaving this way?

> One notable difference between this test and the two previous tests is that there was no need to allocate random arrays at the beginning of the program's running. That array filling was being done sequentially and would be a drag on the parallel fraction of the algorithms. I believe that the performance is stable in this program, because each of the calculations are independent, and so there is no need for any blocking to bring a thread to a halt as it awaits the result of a calculation which is a dependancy. There are only small deviations in performance vs. NUMNODES and the most notible appeared at the top end of the 6 core result. It is likely that since I only have 6 pyhsical cores, that some system process was scheduled and was responsible for the small decline. It is also possible that the 6-core largest NUMNODES result is suffering from a lack of temporal coherence and could benefit from prefetch.

6. What is the Parallel Fraction using inverse Amdahl?

> - 1 thread = 5.92 Mega-Heights / sec
>
> - 6 threads = 33.33 Mega-Heights / sec
>
> - The 6-core speedup result = 33.33 / 5.92 = 5.63

- Entering the speedup into Amdahl's parallel fraction equation we get:

$\frac{6}{6-1}(1-\frac{1}{5.63}) = 1.2 * (1-0.1776) = 1.2 * (0.822) = 0.987$

- This shows that the algorithm is 98.7% parallelizable.

7. What is the maximum performance possible, given the Parallel Fraction?

Given the parallel fraction result, the maximum possible speedup would be 98.7%. This would mean that the runtime could be reduced to only 1.3% of it's single-core runtime, given a large enough number of cores applied to the program. The speed-up could never be greater than that, as the remaining 1.6% would be inherently sequential and would always need to run on a single core.