**Project #4**

**Functional Decomposition**

**CHI CHIEH WENG**

[**wengchic@oregonstate.edu**](mailto:wengchic@oregonstate.edu)

1. **Machine:** OSU flip (Linux) (**Intel intrinsics**)

2.

|  |  |  |  |
| --- | --- | --- | --- |
| **ARRAY SIZE** | **SIMD Speedup** | **SIMD Performance** | **Non-SIMD Performance** |
| 1024 | 3.202 | 800.227 | 248.786 |
| 2048 | 3.249 | 1194.148 | 108.613 |
| 4096 | 3.258 | 351.773 | 249.542 |
| 8192 | 3.142 | 373.317 | 334.503 |
| 16384 | 3.155 | 373.928 | 395.852 |
| 32768 | 3.157 | 783.581 | 373.755145 |
| 65536 | 3.154 | 474.298 | 388.548212 |
| 131072 | 3.172 | 441.324 | 139.124 |
| 262144 | 3.145 | 504.183 | 160.319 |
| 524288 | 2.95 | 597.707 | 202.602 |
| 1048576 | 2.939 | 751.794 | 255.766 |
| 2097152 | 3.013 | 1029.232 | 341.282 |
| 4194304 | 3.057 | 1191.098 | 389.673 |
| 8388608 | 2.934 | 1093.6 | 372.746 |

3.

4. **What patterns are you seeing in the speedups?**

The array size from 1,024 to 8,388,608 is the fastest at the beginning and then starts to decline linearly. When the array size is 2097152, it rises slowly.

5. **Are they consistent across a variety of array sizes?**

In various array sizes, the speed-up gap is not big at first, but it can be clearly seen from the graph that when the array exceeds a certain size, the speed-up will decrease.

6. **Why or why not, do you think?**

In the SIMD vectorization lecture, it was mentioned that as the data size increases, time consistency problems usually occur, which leads to a decrease in acceleration. I think if the amount of data is enough, the graph of acceleration reduction can be seen more clearly.

**Extra Credit**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ARRAYSIZE | 1 core alone | | 2 cores alone | | 4 cores alone | | SIMD alone | 2 cores SMID | 4 cores SMID |
| 1024 | 0.887 | 1.316 | | 3.015 | | 2.363 | | 2.063 | 2.525 |
| 2048 | 0.941 | 1.325 | | 3.152 | | 2.806 | | 3.893 | 4.073 |
| 4096 | 0.967 | 1.348 | | 3.344 | | 3.074 | | 4.506 | 4.383 |
| 8192 | 0.985 | 1.257 | | 3.568 | | 3.215 | | 5.004 | 8.308 |
| 16384 | 0.991 | 1.268 | | 3.892 | | 3.36 | | 6.103 | 6.441 |
| 32768 | 0.998 | 2.263 | | 4.125 | | 3.368 | | 6.466 | 6.741 |
| 65536 | 0.997 | 2.348 | | 4.138 | | 3.349 | | 6.32 | 6.669 |
| 131072 | 0.995 | 1.956 | | 4.156 | | 3.352 | | 3.925 | 12.427 |
| 262144 | 0.997 | 1.857 | | 4.23 | | 3.389 | | 6.51 | 12.67 |
| 524288 | 0.998 | 1.889 | | 4.354 | | 3.313 | | 6.588 | 7.299 |
| 1048576 | 0.997 | 1.455 | | 4.315 | | 3.246 | | 6.298 | 6.712 |
| 2097152 | 1 | 1.435 | | 4.231 | | 3.236 | | 6.357 | 12.153 |
| 4194304 | 1 | 1.45 | | 4.214 | | 3.177 | | 6.26 | 11.71 |
| 8388608 | 1 | 1.433 | | 4.256 | | 3.157 | | 6.294 | 10.052 |

1. **Combine multithreading and SIMD in one test. In this case, you will vary both the array size and the number of threads (NUMT). Show your table of performances. Produce a graph similar to the one on Slide #19 of the SIMD Vector notes, using your numbers. Add a brief discussion of what your curves are showing and why you think it is working this way.**

The speedups of the chart are with respect to a for-loop with no multicore or SIMD. The special thing is that using 4 cores alone will be faster than when using SIMD alone. However, using 2 cores + SMID will be faster than using 4 cores alone. Therefore, using core alone will be faster than using SMID alone after a certain amount, but using core + SMID will be better than in all cases. In addition, the reason for this is because each core has its own SMID, but there is no way to achieve what we expected Speedup equal to 16 (4 cores \* 4 SSE\_WIDTH) can only be reached approximately. Moreover, we can also find that the array size be increase and the speedup will drop down, because the cache can't keep up.