First individual report

-, SUMUKHA SHRIDHAR, S221654

1 DATABAR EXERCISE 1

1.1 Observing cache performance

By observing the time measurements, one can see that, the HPC system has 3 cache levels. The size of L3 cache is 64Megabytes, L2 cache is 4 Megabytes. L1 is 512 Kilobytes. Until we can observe a time, in the order of 0.1 ns, its safe to assume that, we are accessing the L1 level of cache. Since L1 is the smallest and nearest to the CPU, it takes less time to perform read and write to this cache.

While moving from L1 to L2 level, we can observe a small variations in the time measurement. Its a very close bet that has been speculated here that the L2 size is 4 Megabytes!!! There is time measurement of 14.7 ns, which indicates the largest and farthest level of cache. This should be L3, with size 64 Megabytes.

By experiments done with loop unrolling, there was speedup of approximately 5x was observed. When we have unrolled a loop, we hide the latency in read and write of the data, reduce the exit condition checks which reduce branch penalties. Less exit condition checks have a great impact on the performance, since these checks would hinder instruction pipe-lining.

To suit the memory hierarchy, we can make use of xmm and ymm registers. This would also lead to vectorisation, which are implemented with SIMD intrinsic.

*The measurements were averaged over 5 observations.

1.2 Working with SIMD instructions

In SIMD programming we are utilising the vector registers available in the CPU to gain performance. In SIMD same instructions are run in parallel on multiple data elements. For the current task, SSE extensions have been used. Our vector registers are 128 bits long and can hold four 32-bit integers. So, we are expected to get a performance gain of 4x with the current implementation.

In the current task, a speedup of 3.3x was gained while doing vectorisation with unrolled loops compared to naive implementation. This is due to the utilisation of 128-bit vector registers. In every loop iteration four integers are loaded to the SSE registers, instead of one. So, there are four times less load operations in sum vectorized unrolled than in the naive implementations. Also, when loop unrolling is done there are four times less loop iterations, which means four times less exit condition checks for the loop.

In the code there is speedup gain of 3.23x with the vectorized unrolled version, for n=777777, when compared with the naive implementation. Also, there is speedup gain of 3.03x with the vectorized version, for n=777777, when compared with the naive implementation.

There is a very negligible speedup going from vectorized to vectorized unrolled unrolled version, since it is a very low level of optimisation technique compared to SIMD technique. When we have unrolled a loop, we hide the latency in read and write of the data, reduce the end of loop conditions which reduce branch penalties.

For n=7777, the speedup observed was around 2.7x for both vectorized unrolled and vectorized version of the code. With n=777777 we have a much bigger problem size and all the cache and registers are filled and utilised much effectively than n=7777 case.

Author's address: -, Sumukha Shridhar, s221654, s221654@dtu.dk.

2 DATABAR EXERCISE 2

2.1 Working with performance counters

Task A:

The cache-references by a-v2 is 45148, where as for a-v1, it is 215344. This measure has high impact on the performance of the current code. These are very expensive calls and the numbers differ by order of magnitude when we compare a-v1 and a-v2. Also, the amount of time spent by a-v2 in main is less than the amount of time spent by a-v1. a-v2.c is faster than a-v1.c since there a-v2 has a sorted array, thus there are less and compare and mov operations in the main function.

Also, the branch misses in a-v1 is order of magnitude more than than the branch-misses for a-v2. To conclude, cache-references and branch misses in a-v1 are much higher than a-v2, which makes a-v1 slower than a-v2.

Task B:

The cache-references for b-v1 is almost twice the cache-references for b-v2. As discussed earlier, these are very expensive calls and they greatly influence the performance of the program at the given scale. b-v1 had around 75% load misses, which are one more expensive operation. Also, b-v2 has roughly 33% less ref-cycles and instructions compared to b-v1, which adds upto the improved performance of b-v2.

3 DATABAR EXERCISE 3

In the code the stencil can be parallelised for shared memory architecture. But for distributed systems, we would have to add halo regions in order to handle the boundaries between adjacent domains. Also, when parallelising we would have to keep in mind that more the number of domains, more is the communications needed between them, which is much more costlier than computations.

By Amdahl's law, we have

$$speedup = \frac{1}{1 - f + \frac{f}{p}} \tag{1}$$

where,

f = fraction of the parallelisable part of the program,

p = number of processors

We get a speedup of 20x, considering 95% of the program to be parallelisable, with $p \longrightarrow \infty$.