HASC

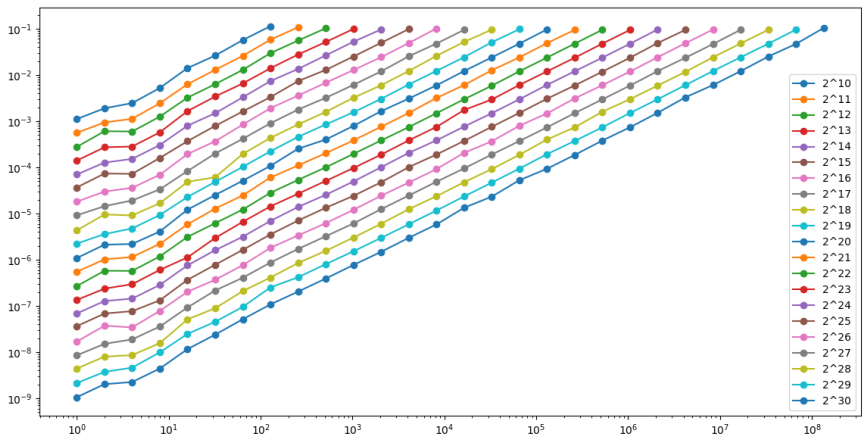
Sheet 1 —

1. (Pointer chasing)

   (a) In the main experiment, we run the pointer chasing with a certain size of vector and a certain stride. This certain size is from 256 integers (1024 bytes) to $N = 512 \times 1024 \times 1024$(max size) integers. The stride is from 1 to $n/2$. (e.g., If $n = 256$, s ranges from 1 to 128.)

   The time that empty experiment takes won't be affected by the cache size because it won't visit any memory. At the same time, the time every loop takes can be different with different n and s. So, we should calculate time these loop takes and eliminate their effects.

Figure 1: Result with empty loop: the speed of a for loop varies with its length.



   For each n, we can keep the total amount of pointer chasing loops invariant.

   (b) See result_a.txt and result_b.txt.

   (c) See figure (a), (b).

   (d) Comparing the total bytes of the vector (show in the legend) with the size of caches, we can find 3 obvious gaps among these lines in figure (b). They correspond to the size of L1, L2 and L3 cache.

   The first gap is between 2^{15} and 2^{16}, which corresponds to L1 cache (48KB). The second gap is between 2^{15} and 2^{16}, which corresponds to L2 cache (48KB). The third gap is between 2^{22} and 2^{23}, which corresponds to L3 cache (8MB). In figure (a), there is an obvious gap between 2^{24} and 2^{25}, which corresponds to L3 cache (24MB). At the data points that take the longest time, we have to visit the main

Figure 2: (a): Laptop with L1: 480KB*10, L2: 1280KB*10, L3: 24MB*1

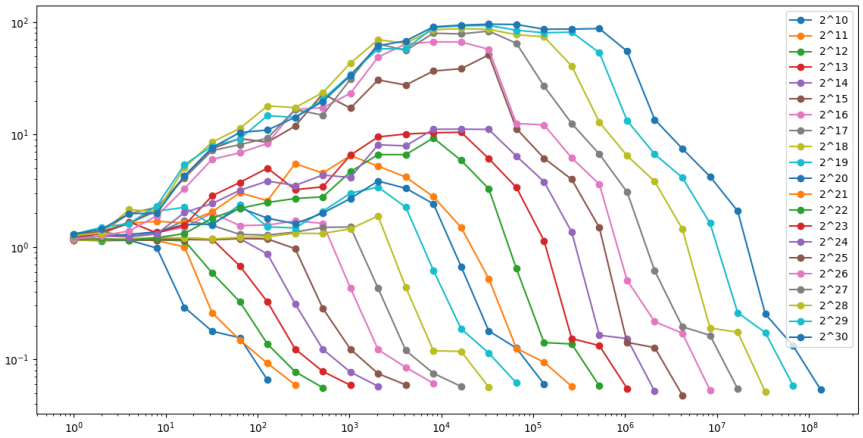
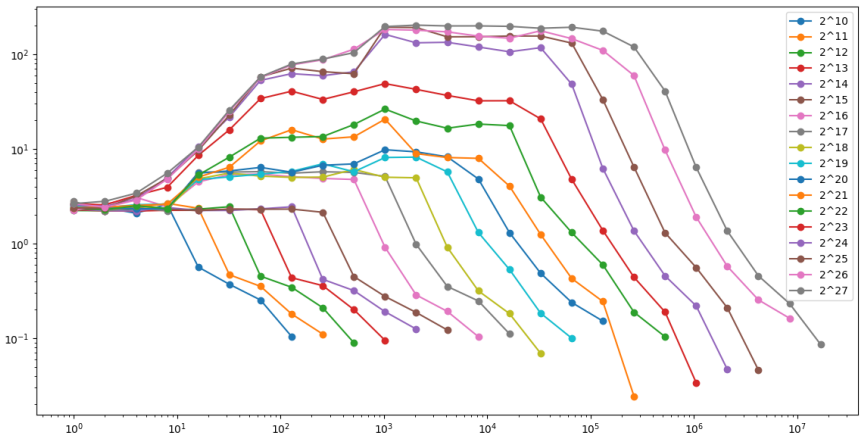
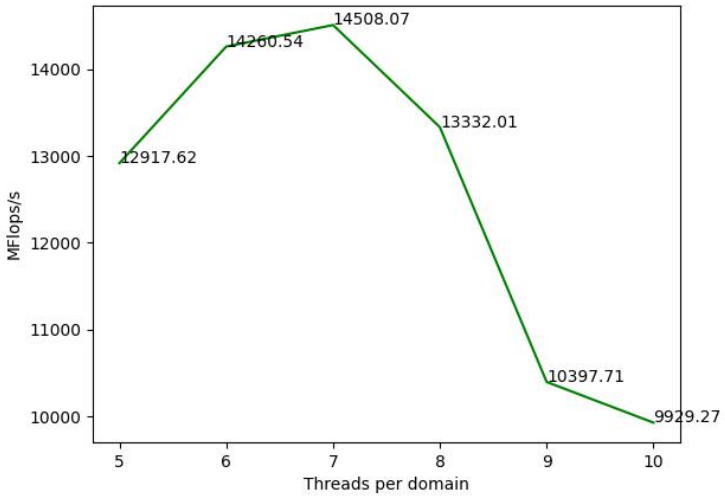


Figure 3: (b): Laptop with L1: 48KB*4, L2: 1280KB*4, L3: 8MB*1



memory. Therefore, the time it takes is close to the speed of main memory (about 100ns).

Figure 5: Plot of MFlops/s and thread number



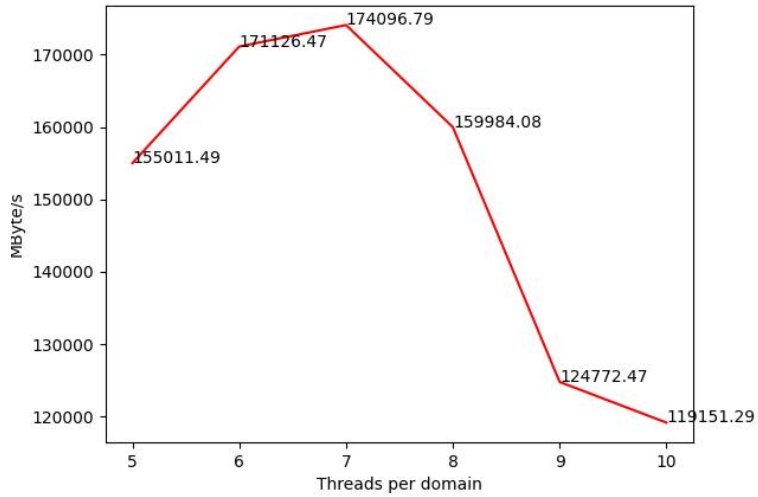
2. (Peak Performance)

   (a) Stats for CPU performance of laptop owned by group members (data source: `https://www.passmark.com/`, `https://www.intel.com/content/dam/support/us/en/documents/processors/APP-for-Intel-Core-Processors.pdf`).

	Liu	Zhang	Jia
CPU	Intel i7-12700H	Intel i9-13500H	Intel i5-8250U
GOps/s (Floating point math)	67	74.9	12.9
GFlops/s	515.2	560	102.4
GByte/s (Data Encryption)	18.1	20.5	2

   Result for running likwid stream test on an ancient Intel i5-8250U fluctuates with the number of threads as expected, with peak achieved at 7 rather than 8 on an 8-thread CPU (?). For a single thread, the number goes down to 4329.15 MFlops/s and 51949.80 MByte/s.

Figure 4: Plot of MByte/s and thread number



(b) Calculation formula for peak performance in GFlops:

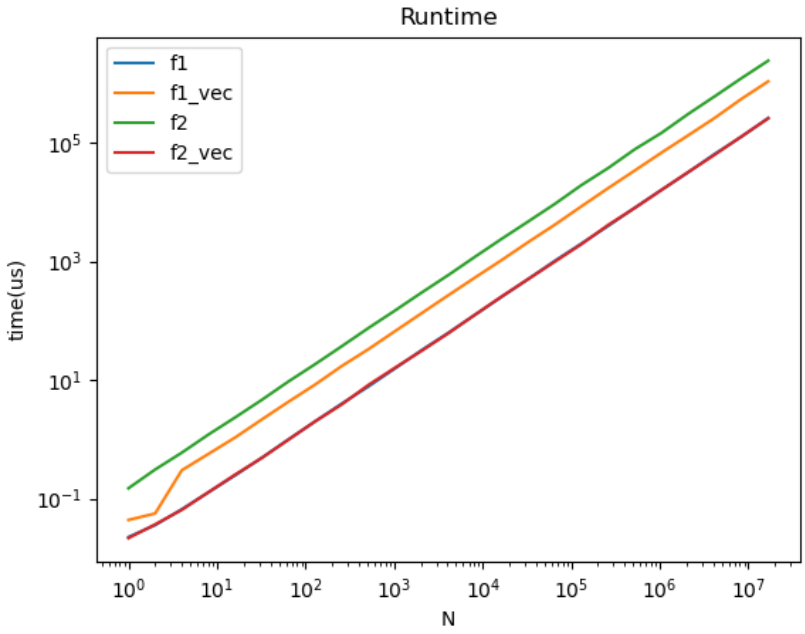
(CPU speed in GHz) × (number of cores) × (CPU instruction per cycle) × (Operation per instruction).

The theoretical maximum performances calculated with above formula are as listed: (I did not find specification for the IPC value regarding each type, therefore set it to a rough 4. OPI is regularly set to 2, but not without exceptions.)

CPU	Intel i7-12700H	Intel i9-13900H	Intel i5-8250U
GHz	4.70/3.50	5.40/4.10	3.40
Cores	$14 = 6 + 8$	$14 = 6 + 8$	4
GFlops/s	449.6	521.6	108.8

Which corresponds roughly to the given number by Intel.

Figure 6: Runtime comparison between vectorized and vanilla implementation.



3. (Vectorized Numerical Integration)

   (a) The value of vectorized integral align with the vanilla one. Since the integration interval is not specified, please see implementation for details.

   (b) See implementation.

   (c) The vectorized version is implemented by using VCL::Vec4d.

   $f1$ in the graph means the first function $f(x) = x^3 - 2x^2 + 3x - 1$, $f2$ means the sum of x^i. The blue line($f1$) is almost fully covered by the red line($f2_{vec}$).

   For $f1$, the regular version is faster than the vectorized version. This might because the number of operations per integration is too small, so that the cost of vectorization is higher than the benefit.

   For $f2$, the vectorized version is much faster than the regular version.

   (d) We estimate the number of operations per integration for f1 and f2 by:

$$ops_{f1} = n \times 8 + 3 + (n - 1)$$

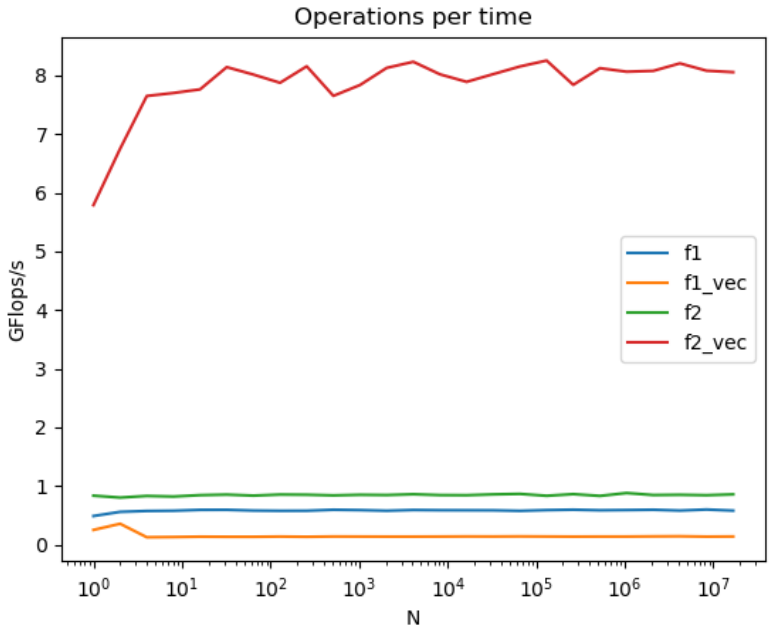
$$ops_{f2} = n \times 122 + 3 + (n - 1)$$

   The f1 regular version has higher operations per time than the vectorized version.

   The f2 vectorized version has significantly higher operations per time than the regular version.

   The tricky thing of estimating the number of operations is that we don't know how many operations it really takes for a vectorized version when doing power

Figure 7: GFlops/s comparison between vectorized and vanilla implementation.



operation. We just estimate that it takes $n-1$ times of multiplication to calculate x to the power of n.