Programming Techniques for Supercomputers
Assignment – 0

Boran Gündogan – Mehmet Arif Bagci – Nedimcan Aytemur

Task 1)

a)

- The following figure shows the execution units/port structure of AVX512 Code: 512 bit instructions for Intel Ice Lake.

Execution Units/Ports					
					AVX512 ADD
				AVX512 MULT	AVX512 MULT
AVX512 LD	AVX512 LD	AVX512 ST		AVX512 FMA	AVX512 FMA
1 Load	1 Load				1 Add
1 Load		1 Store		1 Mult	1 Add

The inner loop consists of the following instructions:

- 3 Load operations
- 1 Store operation
- 2 Additions + 1 Multiplication = 3 Floating Point (FP) operations

Since one AVX-512 vector is 512 bits (8 doubles), 8 iterations are computed per AVX-512 instruction. For a double-precision stencil that performs **2 ADDs and 1 MULT**, all three floating-point operations can be executed concurrently in totally two cycle: the two ADDs are handled by last port and the MULT by another. However, the 3 load operations require 2 cycles due to only 2 load ports being available. The single store can be performed in 1 cycle.

Thus, the bottleneck is the AVX-512 **load operations**.

In order to calculate the P_{max}, the following formula can be used.

$$P_{max} = \frac{W}{T_{inst}^{max}}$$

Where:

W is work → number of FP operation = 8 iteration x 3 FP operation

T_{inst}^{max} is the number of cycles needed to execute that work = 2 cycle as the bottleneck is load.

$$P_{max} = \frac{8\ iteration\ x\ 3\ FP}{2\ cycle} = \mathbf{12\ F/cy}$$

In order to convert this value to GF/s, we should multiply it with frequency (Gcy/s).

$$\frac{\mathbf{12\ F}}{\mathbf{cy}}\ \boldsymbol{x}\ \frac{\mathbf{1.9\ Gcy}}{\mathbf{s}} = \mathbf{22.8\ GFlop/s}$$

- The following figure shows the execution units/port structure of AVX(2) Code: 256 bit instructions for Intel Ice Lake.

Execution Units/Ports					
					AVX ADD
				AVX MULT	AVX MULT
AVX LD	AVX LD	AVX ST	AVX ST	AVX FMA	AVX FMA

The expected usage of the ports remains the same, as the number of operations is consistent with AVX-512. However, on this architecture, one vector length is 256 bits, which corresponds to 4 doubles. As a result, the workload changes: only 4 loop iterations are performed per vector instead of 8.

Therefore, the maximum performance Pmax can be calculated as:

$$P_{max} = \frac{4\ iteration\ x\ 3\ FP}{2\ cycle} = \mathbf{6\ F/cy}$$

$$\frac{\mathbf{6\ F}}{\mathbf{cy}}\ \boldsymbol{x}\ \frac{\mathbf{1.9\ Gcy}}{\mathbf{s}} = \mathbf{11.4\ GFlop/s}$$

- The following figure shows the execution units/port structure of Scalar Code: 64-bit instructions for Intel Ice Lake.

Execution Units/Ports					
					ADD
				MULT	MULT
LD	LD	ST	ST	FMA	FMA

The expected usage of the ports remains the same, as the number of operations is consistent with AVX-512. However, on this architecture, one vector length is 64 bits, which corresponds to a double. As a result, the workload changes: only 1 loop iteration is performed per vector instead of 8.

Therefore, the maximum performance Pmax can be calculated as:

$$P_{max} = \frac{1\ iteration\ x\ 3\ FP}{2\ cycle} = \mathbf{1.5\ F/cy}$$

Programming Techniques for Supercomputers
Assignment – 0

Boran Gündogan – Mehmet Arif Bagci – Nedimcan Aytemur

$$\frac{\mathbf{1.5}\,\boldsymbol{F}}{\boldsymbol{cy}}\ \boldsymbol{x}\ \frac{\mathbf{1.9}\,\boldsymbol{Gcy}}{\boldsymbol{s}} = \mathbf{2.85}\ \boldsymbol{GFlop/s}$$

b) Nontemporal stores or cache line prefetching cannot be used. So, write-allocate structure can be used.

In each inner iteration:

• **3 loads** from b[i][j-1], b[i][j], b[i][j+1]: As one double is equal to 8 byte.

8 byte * 3 = 24 byte is transferred for loading.

• **1 store** to a[i][j]: store operation also triggers load operation as write-allocate is used.

1 Store * 8 byte = 8 byte and 1 load * 8 byte = 8 byte → in total 16 byte is transferred.

24 byte + 16 byte = **40 byte** is transferred in each loop iteration.

• 2 Add + 1 Multiplication operation is performed in each iteration → 3 Flops

The code balance (B_C) quantifies how much data is transferred over a given data path per unit of work and defined as:

$$B_C = \frac{data\ transfer(byte)}{amount\ of\ work(flops)}$$

So the code balance is calculated as

$$B_C = \frac{40\ byte}{3\ Flops} = 13.3333\ bytes/flop$$

Task 2)

a) In Hockney's model the transfer time and effective bandwidth are calculated as:

• Transfer Time → $T = T_l + \frac{V}{b}$ where T_l (s) is latency, V is data volume in byte, and b is bandwidth in Byte/seconds.

• Effective bandwidth → $b_{eff} = \frac{V}{T} = \frac{V}{T_l + \frac{V}{b}}$

T_l is given as 130 ns = 130 x 10^{-9} seconds

Bandwidth is given as 250GB/s = 250 x 10^9 Byte/seconds

Data volume is given as 4096 Byte

So transfer time can be calculated as:

$$T = 130 \text{ x } 10^{-9}\text{seconds}_l + \frac{4096 \text{ Byte}}{250 \text{ x } 10^9 \text{ Byte/seconds}} = \mathbf{146.38\ ns}$$

And effective bandwidth can be calculated as:

$$b_{eff} = \frac{4096\ Byte}{146.384 \times 10^{-9}\ \text{s}} = \mathbf{27.98}\ \boldsymbol{GB/s} \approx \mathbf{28.0}\ \boldsymbol{GB/s}$$

b) Message Size

$N_{1/2}$ for Half Asymptotic Bandwidth

$$\frac{N}{T_\ell + \frac{N}{b_S}} = \frac{b_S}{2}$$

When, we solved this equation;

$$N_{1/2} = b_S . T_\ell = 250 . 10^9 * 130 . 10^9 = \mathbf{32500}\ \boldsymbol{byte}$$

c) **Outstanding Prefetches and Data In-Flight**

$$K = \left\lceil \frac{b_S \cdot T\ell}{cache\ line\ size} \right\rceil$$

- Cache line size = 64 bytes
- $N_{1/2} = b_S . T_\ell = 250 . 10^9 . 130 . 10^9 = \mathbf{32500}\ \boldsymbol{byte}$

Number of Prefetches:

$$K = \left\lceil \frac{32{,}500}{64} \right\rceil = \mathbf{507.8125}$$

Data In-Flight:

$$507.8125 \times 64 = \mathbf{32,500}\boldsymbol{bytes}$$

Programming Techniques for Supercomputers
Assignment – 0

Boran Gündogan – Mehmet Arif Bagci – Nedimcan Aytemur

Task 3)

(a) Performance vs. Power-of-2 Strides

To investigate the effect of stride length on memory performance, a double-precision vector update kernel was benchmarked for various power-of-2 strides, with M ∈ {1, 2, 4, 8, ..., 2^20} and $N = 10^8$. The update loop iterated over the array as a[i] = s * a[i] with increasing stride values M, measuring the number of updates per second.

As shown in the performance graph, the performance drops rapidly as the stride increases. This phenomenon is due to the reduced spatial and temporal locality: as the stride increases, fewer elements within a cache line are utilized, and cache lines are evicted before they can be reused. Furthermore, because power-of-2 strides tend to map memory addresses to the same cache sets, cache conflicts become frequent, leading to cache thrashing and degraded memory throughput. After a certain point (around $M = 2^{13}$), performance saturates at a low level due to the complete loss of locality.

(b) Non-Power-of-2 Strides ($M = 8 \times 1.2^n$)

To evaluate whether the performance degradation observed in part (a) is primarily due to regular stride patterns, a second set of experiments was conducted using non-power-of-2 strides defined as $M = 8 \times 1.2^n$ for integer n, with $M \leq 10^6$.

The performance results show a striking difference: in contrast to the power-of-2 case, the performance remains more stable and generally higher across a wide range of stride values. This is because non-regular stride values lead to a more uniform distribution of memory accesses across cache sets, minimizing cache conflicts and allowing better reuse of cache lines. In particular, sudden drops in performance, as seen in the power-of-2 case, are mostly avoided.

These results demonstrate that avoiding powers of two in stride lengths can significantly improve performance due to better cache behaviour. This observation is particularly relevant when designing memory-bound kernels or stencil-based loops in high-performance computing.

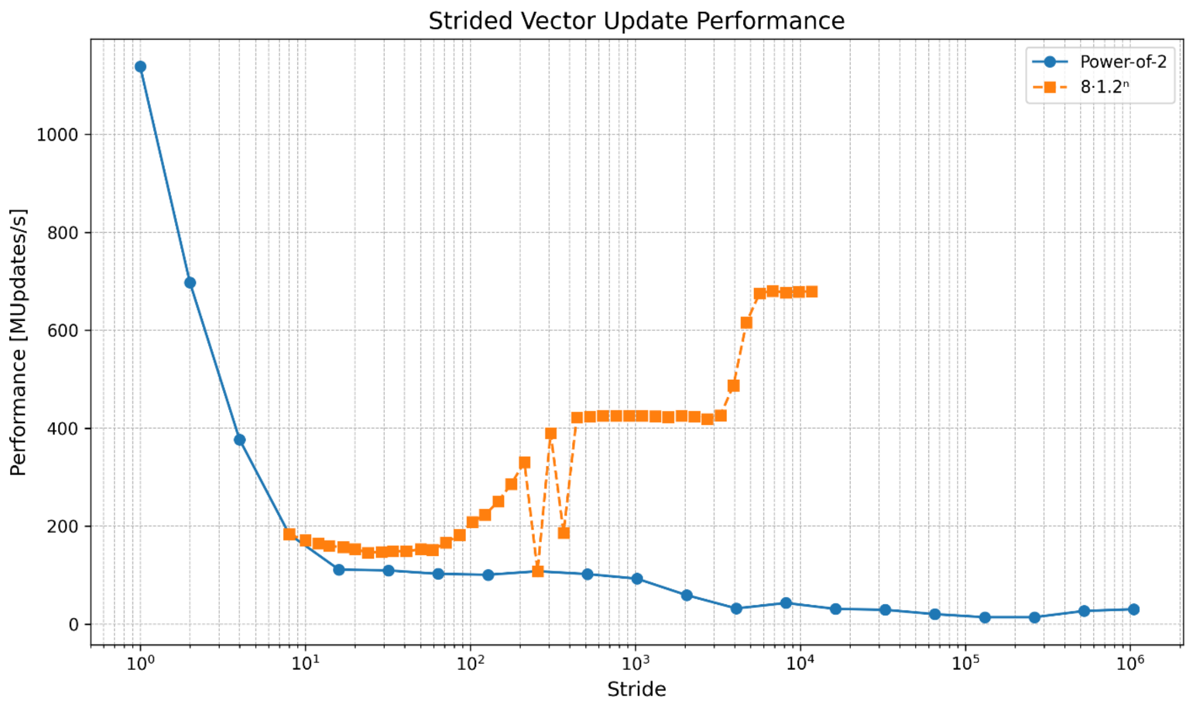

Figure 1. Performance Measurement of Strided Vector Updating