CS 5220 – Sep. 01 Preclass Questions

Eric Gao – emg222

Question 1

We have 60 cores per accelerator and 15 Xeon Phi accelerator boards. Each core on the Xeon Phi board has a clock speed of 1.053 GHz (http://ark.intel.com/products/71992/Intel-Xeon-Phi-Coprocessor-5110P-8GB-1_053-GHz-60-core). These accelerators use the IMCI instruction set, which can perform 2 FLOPs per FMA and have 512 bits per instruction, which means we can perform 8 double-precision FMAs per Vector FMA, and we can perform 1 Vector FMA per cycle, since we only have 1 thread per core. The theoretical peak FLOP rate for just the accelerator boards is then:

$$\begin{aligned} 15 \text{ accelerators} \times 60 \frac{\text{cores}}{\text{acclerator}} \times 1.053 \times 10^9 \frac{\text{cycles}}{\text{second core}} \\ \times 1 \frac{\text{Vector FMAs}}{\text{cycle}} \times 8 \frac{\text{FMAs}}{\text{Vector FMA}} \\ \times 2 \frac{\text{FLOPs}}{\text{FMA}} = 15.16 \text{ TFLOP/s} \end{aligned}$$

We then also have 8 nodes, with 12 cores per node. Any core on these 8 nodes has a theoretical maximum clock rate of 3.5 GHz (http://ark.intel.com/products/75283/Intel-Xeon-Processor-E5-2697-v2-30M-Cache-2_70-GHz). These processors use the SIMD (AVX) instruction set, which can perform 2 FLOPs per FMA and have 256 bits per instruction, which means we can perform 4 double-precision FMAs per Vector FMA. In addition, we have 2 threads per core, which means we can perform 2 Vector FMAs per core. Assuming these specs, the FLOP rate of our 8 compute nodes is given by:

$$\begin{aligned} 8 \text{ nodes} \times 12 \frac{\text{cores}}{\text{node}} \times 3.5 \times 10^9 \frac{\text{cycles}}{\text{second core}} \\ \times 2 \frac{\text{Vector FMAs}}{\text{cycle}} \times 4 \frac{\text{FMAs}}{\text{Vector FMA}} \\ \times 2 \frac{\text{FLOPs}}{\text{FMA}} = 5.38 \text{ TFLOP/s} \end{aligned}$$

This gives us a total and theoretical maximum FLOP rate of: $15.16 + 5.38 = 20.54$ TFLOPs, or about 20.5 TeraFLOPs per second.

Question 2

My machine has a Haswell Intel Core i7-4770K (http://ark.intel.com/products/75123/Intel-Core-i7-4770K-Processor-8M-Cache-up-to-3_90-GHz). There are 4 cores. It uses a SIMD (AVX) architecture, which uses 256 bit instructions, resulting in at most 4 double-precision FMAs per Vector FMA. We have 2 threads per core, so we can perform 2 Vector FMAs per core. Assuming these specs, we get a peak FLOP rate of:

$$4 \text{ cores} \times 3.9 \times 10^9 \frac{\text{cycles}}{\text{second core}} \times 2 \frac{\text{Vector FMAs}}{\text{cycle}}$$
$$\times 4 \frac{\text{FMAs}}{\text{Vector FMA}} \times 2 \frac{\text{FLOPs}}{\text{FMA}} = 249.6 \text{ GFLOP/s}$$

Question 3

Speedup is found by:

$$S(p,t) = \frac{\text{Pipelined time}}{\text{Serial time}}$$

Assuming we have t tasks that take τ time each, we know that the serial time is $t\tau$.

We have a pipeline with p stages, assuming we don't have any branching and don't have to flush the pipeline at any point. We also assume that each stage in the pipeline takes an equivalent amount of time. That is if a task takes τ time to complete in the pipeline, then each stage in the pipeline takes $\frac{\tau}{p}$ time if there are p stages in the pipeline. Therefore, after we finish the first task in τ time, we finish the each of the remaining tasks in $\frac{\tau}{p}$ time. Our pipelined speed is then:

$$time_{pipelined} = \tau + (t-1)\frac{\tau}{p} = \tau\left(1 + \frac{t-1}{p}\right)$$

Our speedup is then given by:

$$S(p,t) = \frac{1 + \frac{t-1}{p}}{t}$$

Question 4

The minimum serial time, assuming no pipelining, is just the sum of the times:

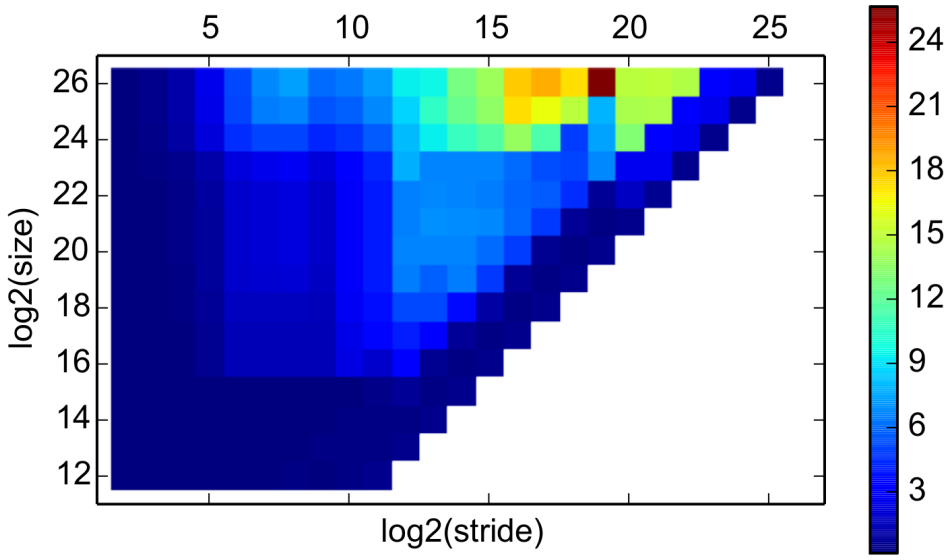
$$time_{serial} = 1 + 0.5 + 0.25 + 0.5 + 0.5 = 2.75 \text{ hr}$$

The minimum parallel time can be seen in the following table.

time	0 hr	1 hr	1.25 hr	1.5 hr	1.75 hr	2 hr	2.25 hr
	GCC	GCC done					
		MPI	MPI	MPI done			
		BLAS	BLAS done				
			LAPACK	LAPACK	LAPACK done		
					App	App	App done

The minimum parallel time is therefore 2.25 hrs.

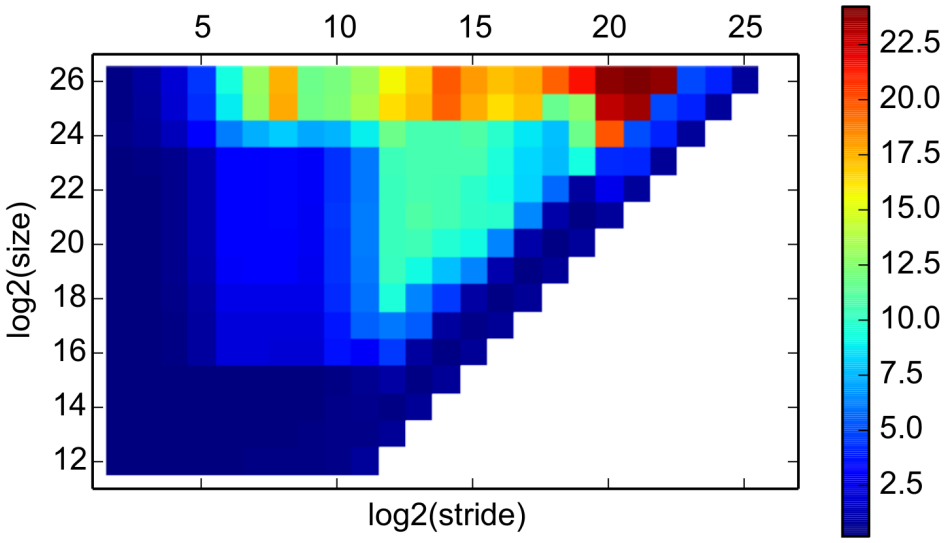
Question 5



We first notice that membench runs extremely quickly for any array with size less than or equal to 2^{15} bits. This indicates that perhaps the L1 cache is 32 KB large. We then get even larger increases in time when the size of the array increases above approximately 2^{17} or 2^{18} bits, or 128 KB or 256 KB. This indicates that the L2 cache may be around 256 KB large. The final large increase in time occurs when we try to load an array of size 2^{24} bits or 16 MB, indicating that the L3 cache may be around 8 MB large.

We also notice that no matter the size of the array, we have good times when the stride is small, indicating that spatial locality improves times. Similarly, we have good times when the stride is large relative to the size the array. But this might be mainly because we have to perform fewer lookups since the stride is so large.

Question 6



Again we notice that membench runs extremely quickly for any array with size less than or equal to 2^{15} bits or 32 KB. So we can assume that the L1 cache may be roughly 32 KB large. Any size larger than 2^{18}, we get an increase in time, indicating that the L2 cache may be either 128 KB or 256 KB large. Finally, we get another increase in time when we have an array larger than 2^{23}, indicating that the L3 cache maybe about 8 MB large. We notice a similar pattern when compared to the previous heat map with regards to stride size.

Question 7

Based on the code I wrote, the first method was the fastest, most likely because of spatial locality. The third method was the second fastest and the second method was the slowest. This was run on my local machine.