High Performance Computing: Homework 2

Dmitriy (Tim) Kunisky [dk3105]

- Processor: Intel(R) Core(TM) i5-5257U CPU @ 2.70GHz
  - Maximum flop rate: 19.76 Gflop/s, or 4.94 Gflop/s per core
  - Maximum main memory bandwidth: 25.6 GB/s
- Compiler: Apple LLVM version 9.1.0 (clang-902.0.39.2)

Problem 2: Matrix-Matrix Multiplication

Loop orderings. The fastest loop order appears to be (j, p, i), corresponding to traversing columns with temporal locality as much as possible (since traversing the i index traverses a column of both $\boldsymbol{A}$ and $\boldsymbol{C}$, traversing the p index traverses a column of $\boldsymbol{B}$ and a row of $\boldsymbol{A}$, and traversing the j index traverses a row of both $\boldsymbol{B}$ and $\boldsymbol{C}$). We then expect the most harmful changes to this ordering to be those that move column traversals earlier in the loop order. Thus, the costliest ordering should be the reverse, (i, p, j), while orderings like (j, i, p) should be intermediate. The table below illustrates that this is indeed the case.

	Ordering (j, p, i)		Ordering (j, i, p)		Ordering (i, p, j)	
Dimension	Flop Rate (Gflop/s)	Bandwidth (GB/s)	Flop Rate (Gflop/s)	Bandwidth (GB/s)	Flop Rate (Gflop/s)	Bandwidth (GB/s)
16	5.93	94.82	2.53	40.51	2.62	41.94
112	9.62	154.00	1.83	29.23	1.54	24.70
304	7.79	124.64	0.91	26.98	0.77	12.34
640	5.66	90.64	0.82	13.15	0.65	10.43
976	4.38	70.05	0.87	13.90	0.38	6.05
1408	4.17	66.71	0.69	11.12	0.19	3.07

Block size tuning. It appears that the optimal block size is around 60. One possible justification for this is that the processor used for these tests has a Level 1 cache of size 64KB. If the block size is B, then the number of bytes required to store two $B \times B$ arrays of doubles (the size of the data on which arithmetic is being performed in one block matrix operation) is $2 \cdot 8 \cdot B^2$, which is smaller than 64KB only while $B \leq 60$ (over values of B divisible by 4). The figures in the table below support this heuristic calculation.

	$B = 40$		$B = 60$		$B = 80$	
Dimension	Flop Rate (Gflop/s)	Bandwidth (GB/s)	Flop Rate (Gflop/s)	Bandwidth (GB/s)	Flop Rate (Gflop/s)	Bandwidth (GB/s)
240	4.69	75.05	8.95	143.26	9.12	145.91
480	5.37	85.93	8.79	140.78	7.92	126.78
720	5.28	84.55	8.68	138.85	8.38	134.07
960	5.26	84.20	8.80	140.82	6.52	104.37
1200	5.41	86.48	8.33	133.21	6.84	109.41
1440	5.19	82.97	8.56	137.01	7.44	119.04
1680	5.31	85.03	8.16	130.55	6.87	109.91

Parallelization. Finally, we parallelize our implementation by simply using the parallelized for loop directive of OpenMP on the top-level block iteration. This roughly doubles the flop rate, which corresponds to this processor having two cores. With this optimization, the matrix multiplication reaches a flop rate of slightly more than 17 Gflop/s, which is approximately 86% of the peak flop rate of 19.76 Gflop/s.

	$B = 60$, Serial		$B = 60$, Parallel	
Dimension	Flop Rate (Gflop/s)	Bandwidth (GB/s)	Flop Rate (Gflop/s)	Bandwidth (GB/s)
240	8.95	143.26	19.68	314.86
480	8.79	140.78	18.41	310.60
720	8.68	138.85	18.68	298.84
960	8.80	140.82	17.01	272.16
1200	8.33	133.21	17.85	285.60
1440	8.56	137.01	17.59	281.39
1680	8.16	130.55	17.24	275.79

Problem 4: Jacobi/Gauss-Seidel Smoothing

We present timing results for varying problem sizes of the Jacobi and Gauss-Seidel smoothers, for one, two, and four threads. The machine used has two cores, so we expect to see a benefit from two threads versus one, but no additional benefit from four threads versus two. The results below support this claim; for large problems, the speed roughly doubles from moving from one to two threads, but there is no effect from moving from two to four threads.

	Jacobi			Gauss-Seidel		
N	1 Thread	2 Threads	4 Threads	1 Thread	2 Threads	4 Threads
50	0.15	0.19	0.18	0.25	0.20	0.28
100	0.55	0.34	0.35	0.90	0.57	0.65
150	1.35	0.68	0.84	2.03	1.10	1.22
200	2.28	1.13	1.51	3.54	2.01	2.20
300	5.03	2.70	3.39	7.89	4.18	4.38
400	10.46	7.86	7.64	17.31	7.74	8.05

(Note that running the parallel version of either algorithm is implemented by passing an arbitrary third argument to the executable, as shown in the top comment of either source file.)