Daniel Garon
Dr Sanjay Rajopadhye
CS 475
2023-02-03

PA1: Parallelization in OpenMP

Algorithm Description:

stencil_1D.c

1. Allocate two arrays of size N
2. Initialize one of the arrays with 0s
3. Assign a predefined value to the first two and last two elements of both arrays
4. For a specified number of iterations:
    a. From 2 to N-2 in the curr array, assign the value of the average of the five contiguous values in the prev array centered at the corresponding index
    b. Swap the array pointers and increment the iteration counter

stencil_2D.c

1. Allocate two arrays of size N2, representing matrices of dimension N x N
2. Initialize one of the arrays with 0s at indices corresponding to a submatrix of size N-2 x N-2 centered within the original matrix
3. Assign a predefined value to the first and last two of both the rows and columns
4. For a specified number of iterations:
    a. In row-major order in the centered submatrix of curr, assign the value of the average of the nine contiguous values in the prev array centered at the corresponding index, extending only horizontally and vertically
    b. Swap the array pointers and increment the iteration counter

mat_vec.c

1. Allocate three arrays:
    a. A of size N x M, representing a matrix
    b. b of size N as the factor
    c. c of size M for the resulting product
2. Initialization the data structures with arbitrary values for the purposes of comparing computation time
3. Compute the dot product for each row of A with b and store the result at the appropriate index in c

Parallelization Approach

stencil_1D: I chose to parallelize the for loop that performs the computations because each iteration is independent. The repeated iterations performed by the surrounding loop cannot legally be parallelized.

stencil_2D: I chose a parallelization that is effectively the same as that in stencil_1D. I parallelized the outer for loop that performs the computations because each iteration is independent. I included a private clause for the inner loop index j. I parallelized the outer loop to avoid the repeated fork/joins which would result from parallelization of the nested for loop.

mat_vec: Every iteration of the outer loop in mat_vec is independent. I chose to parallelize the outer loop with a private clause for inner loop index j.

In each case I chose to use the default static scheduling as I expect the cost of computing each iteration to be comparable.

Experimental Setup:

Experiment data was gathered using host “steamboat” with the following specifications:

Thread(s) per core: 2
Core(s) per socket: 6

L1d cache: 32K
L1i cache: 32K
L2 cache: 256K
L3 cache: 15360K

Experiments planned:

I will execute each of the programs sequentially and then parallelized with an increasing number of processors allocated from 1 to 10. I will repeat each program execution seven times and discard both the fastest and slowest times recorded.

I will use the following input values to run each program

	N	iterations
stencil_1D	100,000	100,000
stencil_2D	1,000	10,000

I will use the following input values to mat_vec:

	N	M
mat_vec	25,000	10,000

Experimental Results:

P	Time (s)	Speedup
Seq	12.28	n/a
1	12.16	1.0
2	6.41	1.9
3	4.63	2.7
4	3.76	3.3
5	3.22	3.8
6	2.74	4.5
7	3.59	3.4
8	3.18	3.9
9	3.01	4.1
10	2.79	4.4

Table 1: stencil_1D results

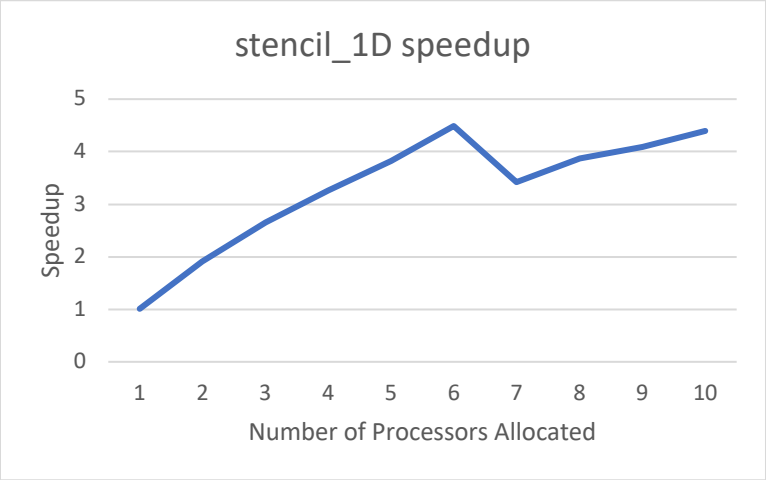

Figure 1: stencil_1D speedup

P	Time (s)	Speedup
Seq	12.32	n/a
1	12.41	1.0
2	6.61	1.9
3	4.49	2.7
4	3.62	3.4
5	3.28	3.8
6	2.87	4.3
7	3.66	3.4
8	3.26	3.8
9	3.05	4.0
10	2.89	4.3

Table 2: stencil_2D results

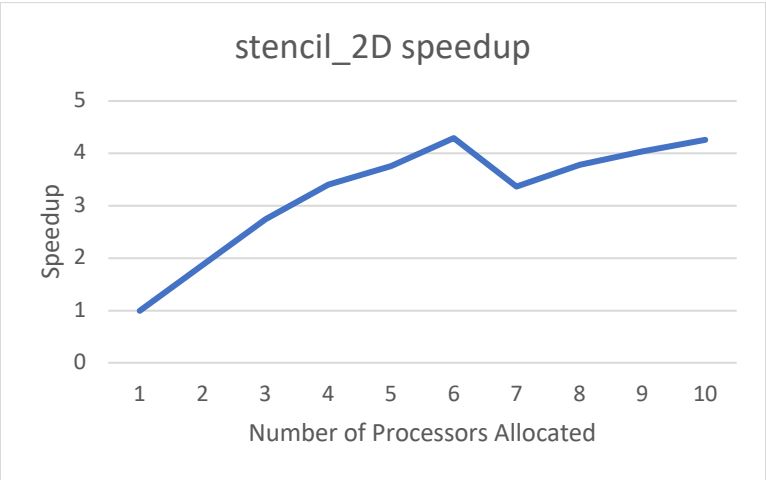

Figure 2: stencil_2D speedup

P	Time (s)	Speedup
Seq	0.216	n/a
1	0.207	1.0
2	0.123	1.8
3	0.086	2.5
4	0.069	3.1
5	0.064	3.4
6	0.062	3.5
7	0.064	3.4
8	0.061	3.5
9	0.059	3.7
10	0.059	3.7

Table 3: mat_vec results

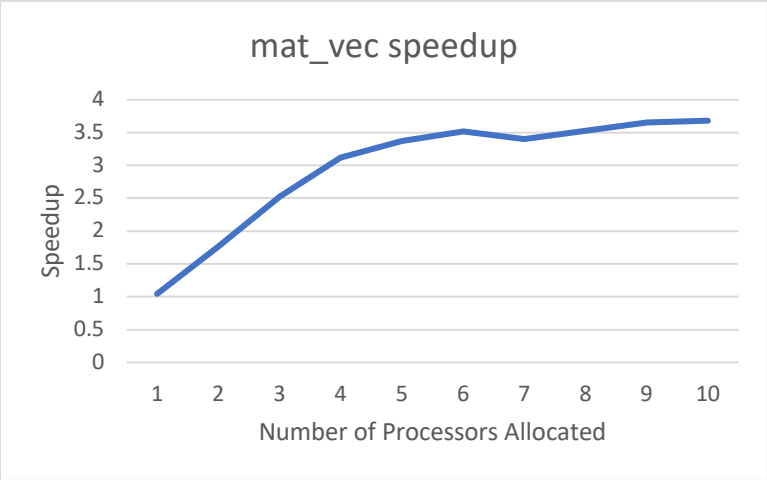

Figure 3: mat_vec speedup

Conclusion:

There is a near linear speedup as the number of allocated threads increases from one to six. Once seven threads are allocated, there is a marked decline in performance. I believe that this result is a consequence of an imbalance of work that results from one of the six physical cores being allocated two chunks. The benefit of hyperthreading is overshadowed by the increased span when one processor is allocated 2/7 share of the work.

As the number of threads allocated increases beyond seven, the speedup begins to recover, as the work imbalance is reduced.

The mat_vec program exhibits a significantly smaller speedup penalty when increasing from six to seven threads. I believe that the performance penalty experienced by the stencil programs may be due to the repeated iterations of the while loops amplifying the work imbalance of the parallelized for loops. The span of each iteration is longer and those costs add up.