Algorithm Design

stencil_1D

This algorithm computes the average of five elements within an array. These elements are located two indexes away from the current index. This algorithm has a time complexity of O(n).

stencil_2D

This algorithm computes the average of nine elements within a matrix. These elements are located two cells away from the current cell. This algorithm has a time complexity of O(n^2).

mat_vec

This algorithm creates a matrix where each cell is computed using the sum of the cell's coordinates multiplied by a scalar from an array. This algorithm has a time complexity of O(n*m).

Experimental Setup

Experiment Conditions

This experiment was conducted on the "cooper" lab machine. This machine has 12 cores, the L1d cache and L1i is 192 kB, the L2 cache is 1.5 MB, and the L3 cache is 15MB. The experiments will run sequentially then be conducted serially with up to 10 cores.

Experimental Results

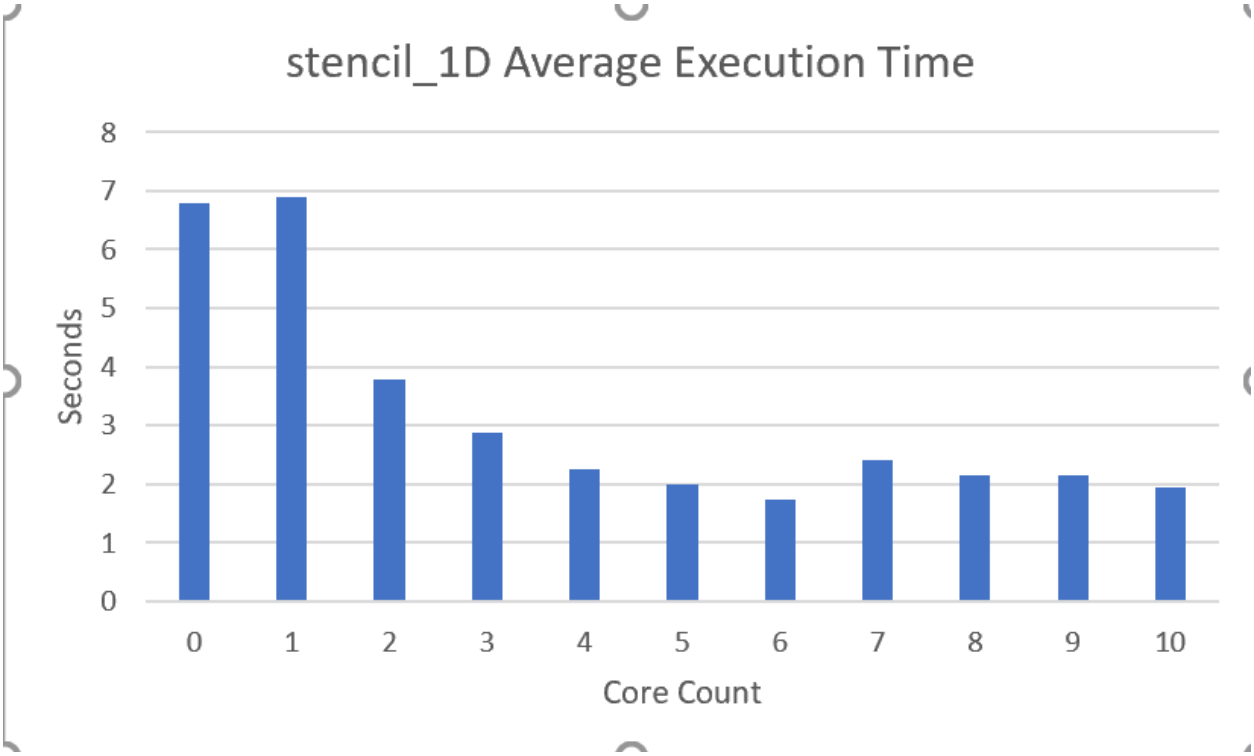
Results

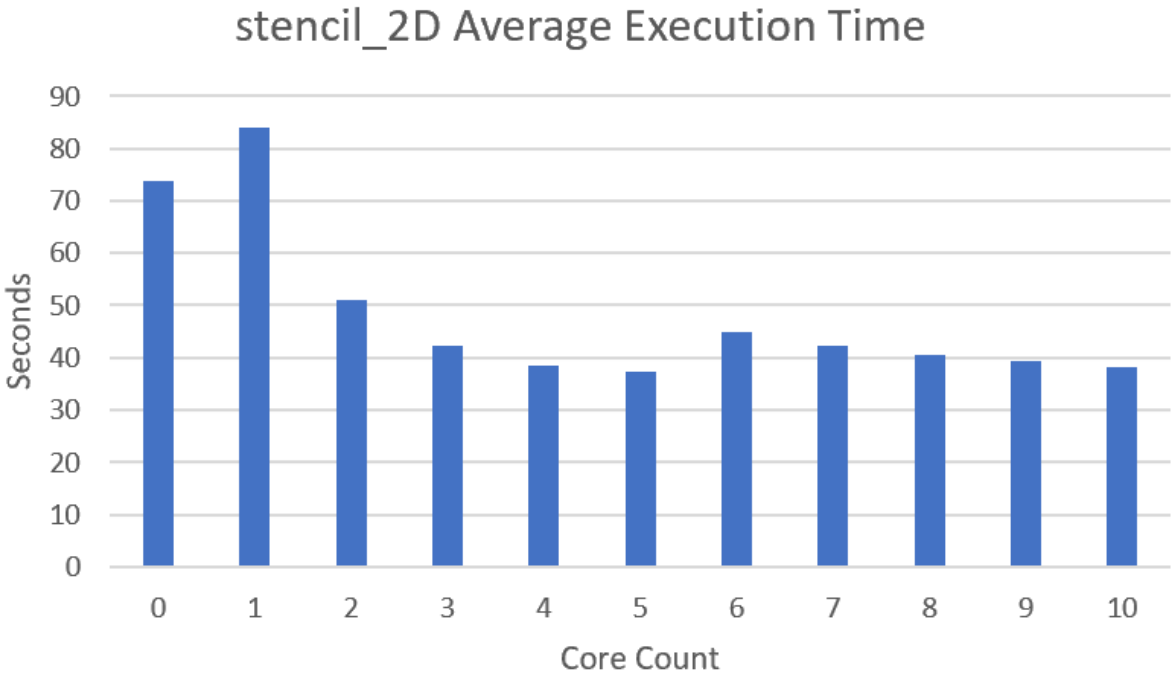
After conducting the experiment, I found serial processing to be much faster than the sequential execution. However, stencil_1D and stencil_2D had diminishing returns. For stencil_1D, 6 cores was the fastest. For stencil_2D, 5 cores was the fastest. Mat_vec's execution speed increased asymptotically toward 0.06 seconds with increasing core count.

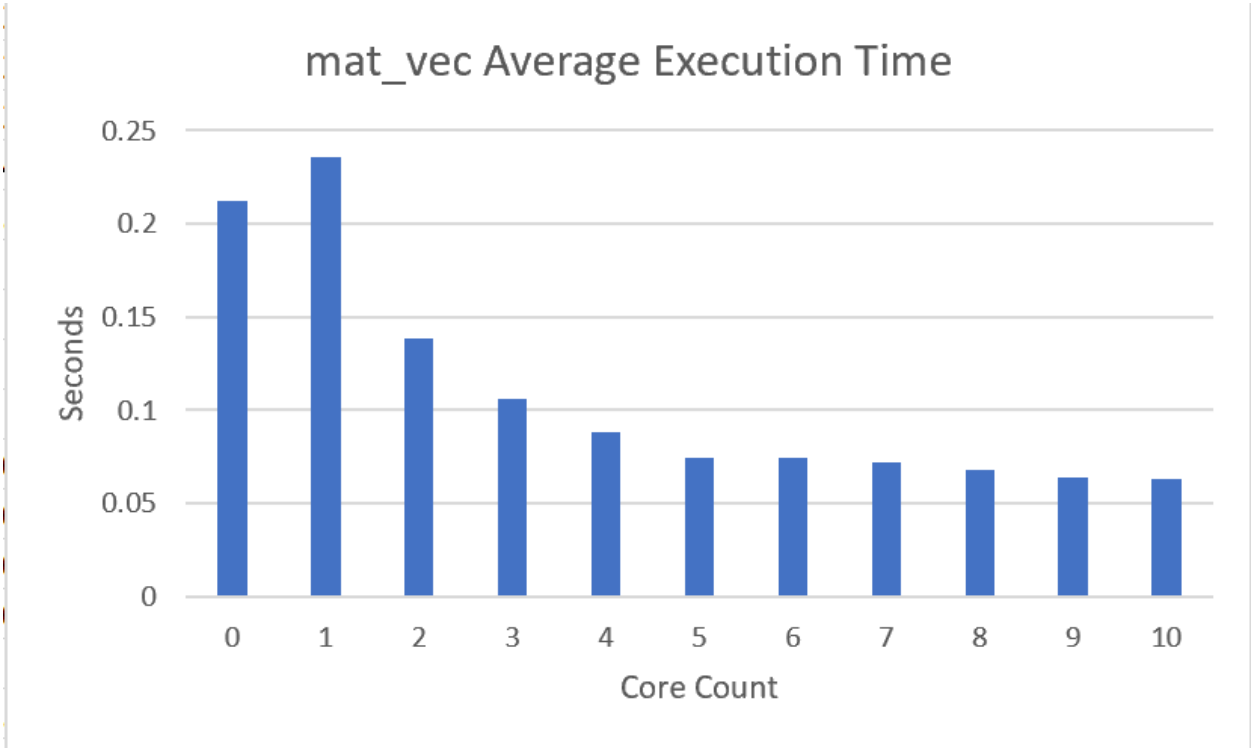
Tables

Normalized Data											
stencil_1D(20000, 30000)											
threads	0	1	2	3	4	5	6	7	8	9	10
Trial 1	6.698504	6.875848	3.770194	2.897284	2.257068	1.97691	1.778284	2.386033	2.154689	2.105642	1.962264
Trial 2	6.662934	6.835967	3.780858	2.891175	2.234435	1.980332	1.726455	2.431737	2.153381	2.332913	1.916382
Trial 3	6.668206	6.908062	3.789988	2.877085	2.239169	1.998384	1.719989	2.388874	2.139496	2.069139	1.927761
Trial 4	7.286092	6.923407	3.798227	2.868087	2.237067	1.973059	1.726747	2.416973	2.153264	2.165382	1.908783
Trial 5	6.683921	6.903272	3.766552	2.875344	2.233483	1.983846	1.717222	2.39283	2.175254	2.096624	1.93553
Average	6.7999314	6.8893112	3.7811638	2.881795	2.2402444	1.9825062	1.7337394	2.4032894	2.1552168	2.15394	1.930144
Speedup		0.987026308	1.79836996	2.35961663	3.03535248	3.42996728	3.92211851	2.82942678	3.15510319	3.15697345	3.52301766
stencil_2D(3500, 3500)											
threads	0	1	2	3	4	5	6	7	8	9	10
Trial 1	73.907448	83.993457	50.896791	42.175412	38.505236	37.383571	44.799806	42.43877	40.473919	39.299397	38.176329
Trial 2	73.692024	83.889944	51.038808	42.057318	38.410355	37.358339	44.691584	42.226077	40.45724	39.282562	38.210663
Trial 3	73.582444	83.876001	50.909254	42.100861	38.446694	37.345701	45.018208	42.27762	40.534163	39.289924	38.34292
Trial 4	73.820851	83.753578	51.039858	42.182656	38.460977	37.310342	44.724315	42.282106	40.540812	39.279079	38.275045
Trial 5	73.647337	83.742131	51.102609	42.10403	38.4286	37.382068	44.736752	42.290507	40.497696	39.316883	38.172587
Average	73.7300208	83.8510222	50.997464	42.1240554	38.4503724	37.3560042	44.794133	42.303016	40.500766	39.293569	38.2355088
Speedup		0.879297817	1.44575857	1.7503068	1.91753724	1.97371272	1.64597495	1.74290223	1.82045991	1.876389	1.92831279
mat_vec(25000, 10000)											
threads	0	1	2	3	4	5	6	7	8	9	10
Trial 1	0.21146	0.234989	0.141128	0.106394	0.087675	0.074919	0.074075	0.071632	0.067781	0.063291	0.06341
Trial 2	0.211777	0.234521	0.137704	0.104905	0.088721	0.072385	0.074327	0.07139	0.068367	0.063231	0.062366
Trial 3	0.214024	0.237034	0.13571	0.105851	0.088826	0.072298	0.075093	0.071456	0.066532	0.064012	0.063006
Trial 4	0.212273	0.234364	0.136586	0.109653	0.086609	0.075295	0.074411	0.074167	0.067024	0.064362	0.063013
Trial 5	0.210893	0.237253	0.141698	0.10327	0.087431	0.078059	0.074226	0.07189	0.068189	0.065275	0.063272
Average	0.2120854	0.2356322	0.1385652	0.1060146	0.0878524	0.0745912	0.0744264	0.072107	0.0675786	0.0640342	0.0630134
Speedup		0.900069685	1.53058199	2.00053012	2.41411049	2.84330323	2.84959907	2.94125952	3.13835149	3.31206449	3.36571904

Graphs

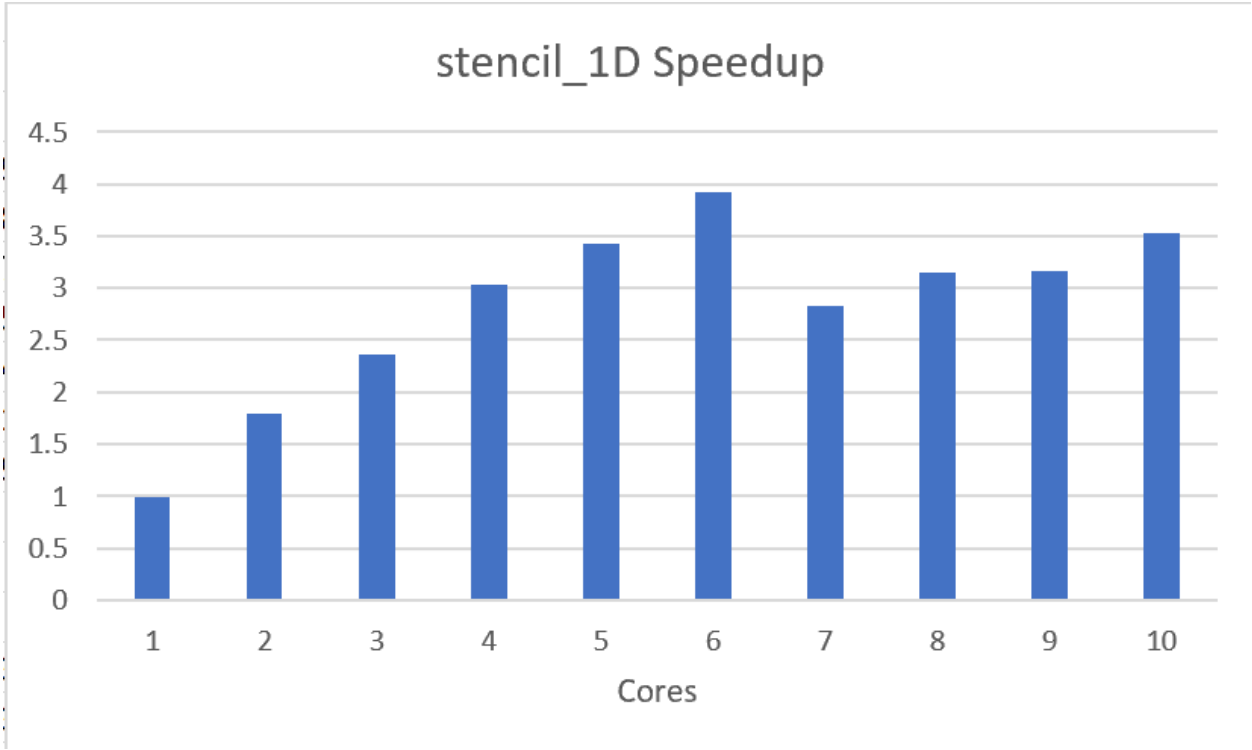


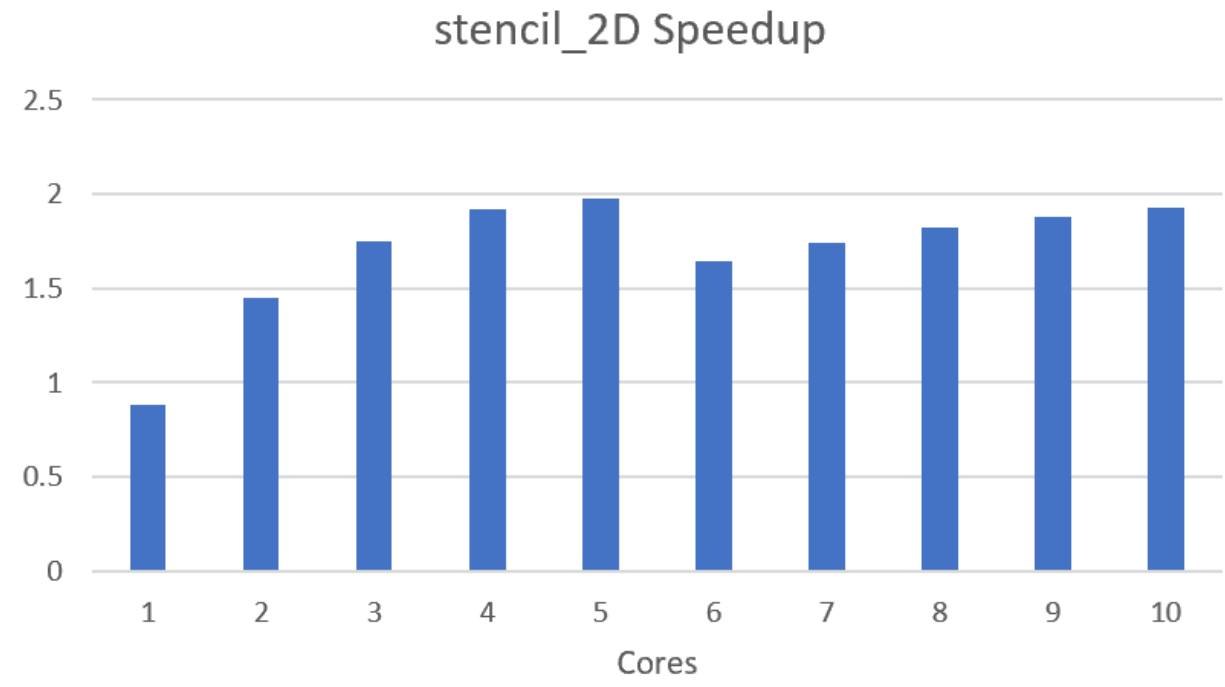

stencil_2D Average Execution Time
Seconds
90
80
70
60
50
40
30
20
10
0
0
1
2
3
4
5
6
7
8
9
10
Core Count

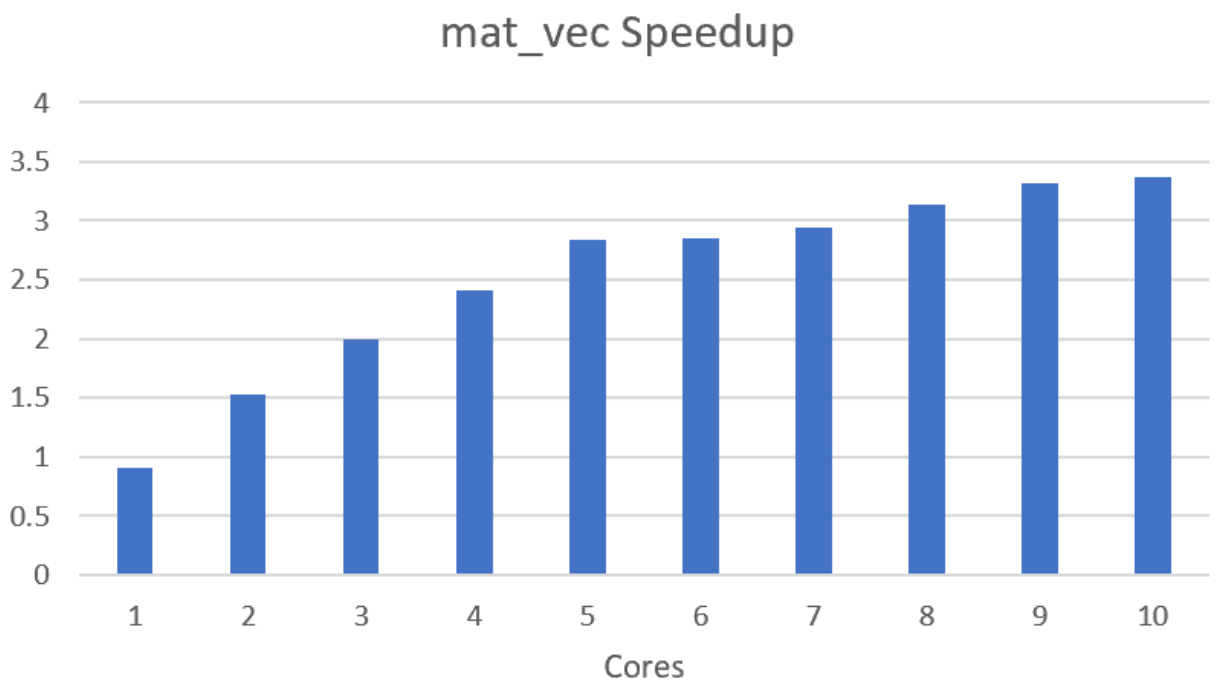

mat_vec Average Execution Time
Seconds
0.25
0.2
0.15
0.1
0.05
0
0
1
2
3
4
5
6
7
8
9
10
Core Count

Speedup

Speedup was calculated using average sequential time divided by average parallel time.



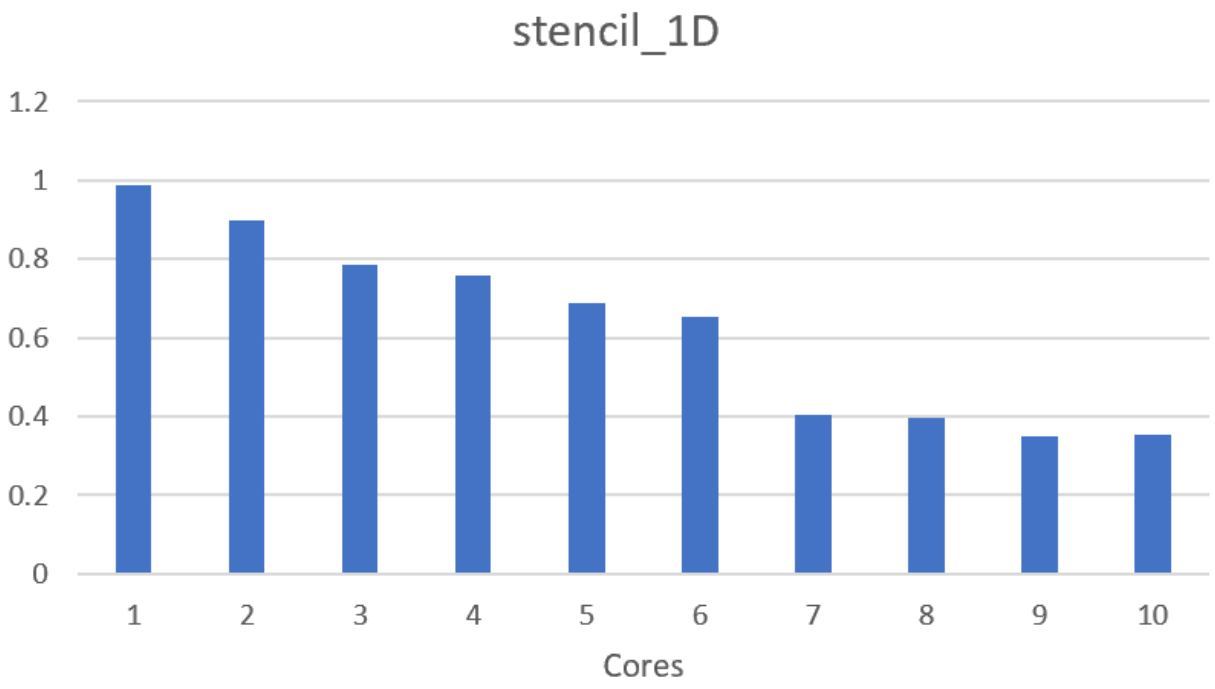


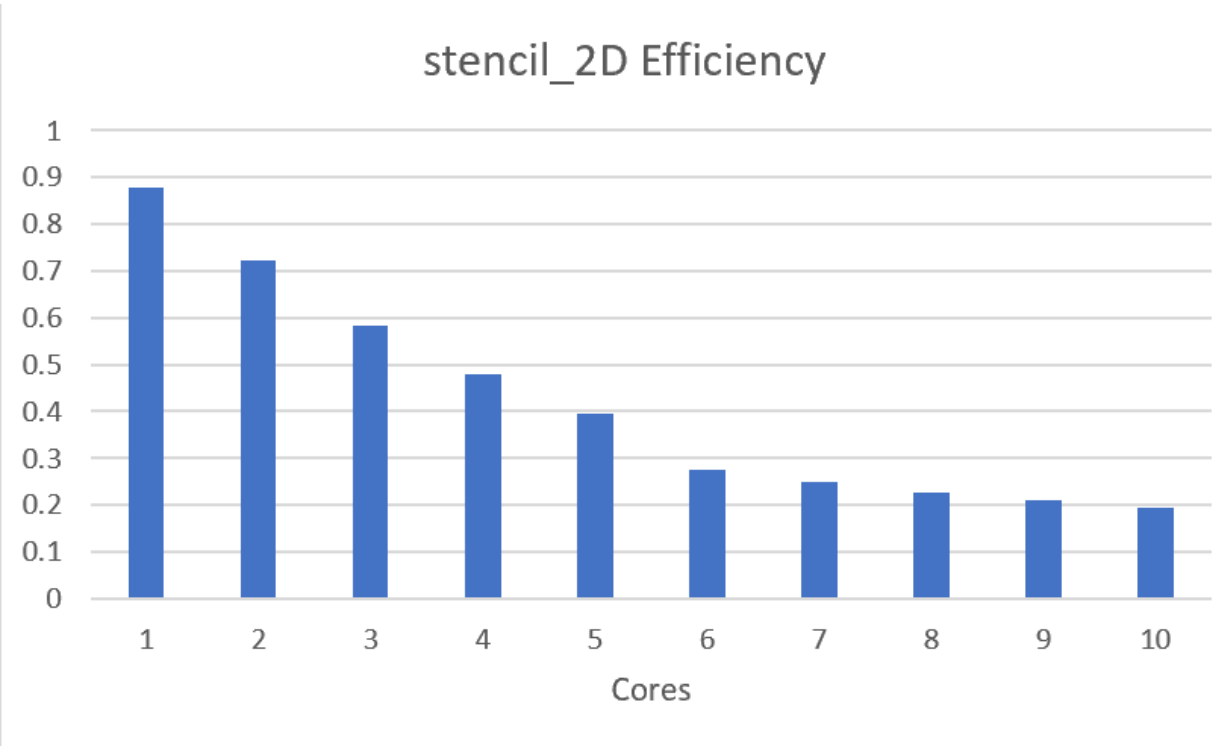


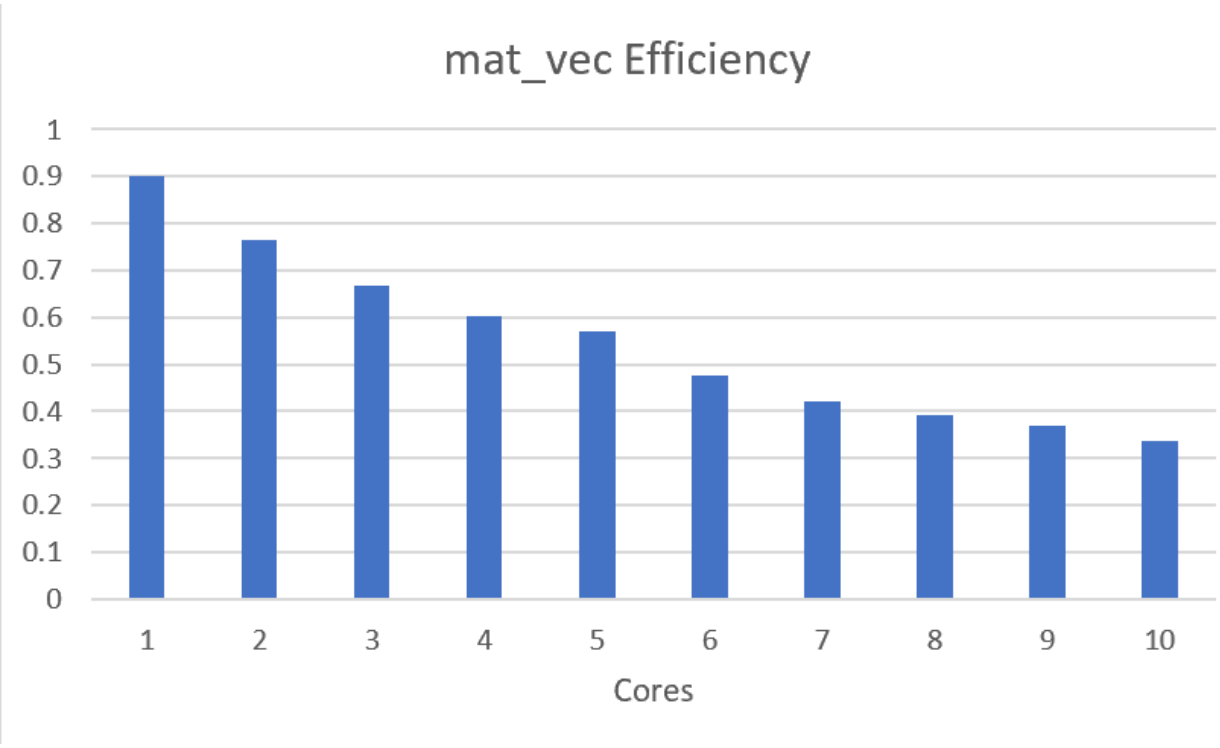
Here we can see how performance improved almost linearly for mat_vec, while stencil_1D and stencil_2D had diminishing returns as cores increased.

Efficiency

Efficiency is calculated by dividing speedup by the number of cores used.




stencil_2D Efficiency
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1
2
3
4
5
6
7
8
9
10
Cores


mat_vec Efficiency
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1
2
3
4
5
6
7
8
9
10
Cores

Conclusion

In conclusion, we can see that speedup is correlated to the number of cores. However, this relationship is not always positively correlated. For some problems, such as stencil_1D and stencil_2D, there is a “sweet spot” for core count. This is seen in the efficiency graphs. After the “sweet spot”, efficiency stagnates.