CSCI 440 Final Project

Evan Alba

Introduction

In this final project, we decided to answer the question of: How does the performance of threads in C compare to Python (multiprocess)? The machine used to conduct the experiments has the following specifications: [4]

Table 1: System Specifications

Operating system: Ubuntu-2204-jammy-v20240829
CPU: Intel Broadwell CPU @ Max. Turbo Frequency 3.3GHz
Memory: 1GiB

Next, the experiments that aided us in answering our question was DNS Lookup, Matrix multiplication, and Sieve of Eratosthenes. Finally, we were aware of the side effects that might affect the experiments to help us answer this question, such as internal system noise and cache. For these reasons, we cleared the cache before each experimental run. [1]

Results

DNS Lookup

This was the CSCI340-DNS-Name-Resolution-Engine assignment which was a multithreaded application that resolves domain names to IP addresses. [2] Each column consisted of 24 experimental runs. We resolved 100, 250, and 500 domain names.

Table 2: DNS Lookup C Implementation

Number of domain names:	**100**	**250**	**500**
	0.001626	31.258247	60.898789
	0.002586	24.08784	53.091423
	0.002706	29.220294	58.26787
	0.002003	22.162608	55.086981
	0.001628	31.50231	52.242414
	0.001674	24.610356	47.188876
	0.002422	26.533417	57.701956
	0.001808	25.395634	51.489152
	0.001431	21.345517	59.505514
	0.001636	20.62462	42.106229
	0.001443	24.151229	43.106307
	0.001683	22.053887	52.78473
	0.001568	19.374727	44.032324
	0.001854	24.723573	47.806795
	0.001986	20.790119	55.73729
	0.001566	20.247729	42.625379
	0.001455	25.073108	39.249132
	0.001432	26.444592	39.768378
	0.001275	29.549054	41.153848
	0.002686	22.099261	55.599254
	0.001474	21.272223	42.8616
	0.001431	21.826123	39.315407
	0.001539	27.555455	40.706192
	0.001993	21.63888	44.521351
Margin of Error:	0.0001679285835347728	1.4118871521353595	2.887812114303134
Acceptable Margin of Error $\leq$ average * 10%:	0.0001787708333333338	2.4314200125000003	4.861863295833334
Average Time:	0.001787708333333336	24.314200125000003	48.61863295833334

Table 3: DNS Lookup Python Implementation

Number of domain names:	**100**	**250**	**500**
	4.932794	8.339633	4.108472
	4.145971	4.128107	4.107842
	4.370909	4.108125	4.112662
	4.113415	4.824197	4.677667
	4.131743	4.098138	4.111409
	4.80851	4.203419	4.116013
	4.115453	4.429539	4.191529
	4.327283	4.121566	4.543122
	4.108901	4.303205	4.065968
	4.630792	4.958995	4.132608
	4.133618	4.125891	4.773285
	4.136486	5.122046	5.033929
	4.452006	4.137458	4.181003
	4.06628	4.549626	4.146522
	4.461041	4.086675	4.519937
	4.15336	4.130646	4.242975
	4.342236	4.8438	5.549705
	4.06658	4.141427	4.102941
	4.684498	4.114	4.782037
	4.147773	4.120738	4.491736
	4.321837	4.578629	4.49196
	4.100163	4.176146	4.208603
	4.10631	4.088927	4.115384
	4.119253	4.128361	6.738168
Margin of Error:	0.10035182284972993	0.35108946560038423	0.24226165757497728
Acceptable Margin of Error $\leq$ average * 10%:	0.42907171666666666	0.4494137250000001	0.44810615416666677
Average Time:	4.290717166666666	4.4941372500000005	4.4810615416666675

Matrix multiplication

A binary operation in which we used 2 matrices of the same dimensions. [5] Each column consists of 84 experimental runs. The dimensions of the matrices used were 100x100, 200x200, and 300x300.

Table 4: Matrix multiplication C Implementation

Matrix Sizes:	**100x100**	**200x200**	**300x300**
	0.394969	1.690205	3.993498
	0.404226	1.689778	4.077157
	0.425958	1.652995	3.871786
	0.440114	1.687249	3.857128
	0.433929	1.705113	3.94071
	0.410754	1.665852	3.883086
	0.417993	1.718835	3.888072
	0.415225	1.707002	24.972984
	0.424647	1.732494	33.288035
	0.408873	1.674096	33.90175
	0.390507	1.679832	34.247047
	0.411774	1.65244	33.89068
	0.409365	1.646764	34.480162
	0.433612	1.667292	31.962507
	0.420058	1.647772	31.797253
	0.407031	1.651649	32.775433
	0.41905	2.656281	32.444163
	3.4218	14.227188	34.382554
	3.4204	12.728747	32.864539
	3.412056	12.627916	33.293203
	3.442656	12.60855	32.504931
	3.422323	13.363216	33.909453
	3.430985	12.7679	33.378241
	3.435784	12.928591	32.758995
	3.427992	13.503356	32.791293
	3.428859	13.518243	35.689955
	3.406812	13.305958	33.641977
	2.435378	13.241968	33.283904
	3.418573	14.117609	32.963915
	3.419945	13.941507	32.804076
	3.41199	13.641521	33.761077
	3.416315	13.044364	32.456827
	2.438214	13.546812	32.77485
	3.42109	13.755871	32.553975
	2.422787	13.32128	33.448292
	3.417267	12.896016	32.800403
	3.406661	13.525686	33.036788
	3.713358	13.934137	31.932579
	3.273535	13.605903	32.153253
	2.580482	14.004891	32.252599
	3.534424	13.960209	31.65132
	3.486824	14.221988	31.659916
	3.090642	14.179464	32.141451
	2.832374	13.492524	32.0661
	3.260651	13.58283	30.639449
	2.884308	12.887464	31.440991
	3.516899	13.072716	32.799433
	2.838167	13.079075	31.211379
	3.446327	13.384934	31.973352
	3.5205	13.100596	30.975288
	3.133764	13.327015	31.840815
	2.412926	12.978311	31.379224
	3.478346	12.858566	30.719405
	3.268191	13.349511	31.333308
	2.661868	13.306078	30.538301
	3.262711	12.842792	32.63881
	2.808087	12.866996	33.997689
	3.494577	13.448867	31.20123
	3.102262	12.600632	31.695154
	2.329224	13.471893	32.069844
	3.735384	12.539168	31.859955
	3.257417	13.751832	31.367346
	2.262884	13.385627	31.72295
	3.509799	13.065172	30.602562
	3.487052	13.535572	31.92846
	2.929791	13.075933	31.112131
	3.20075	13.364084	31.048175
	3.264382	12.980086	31.997641
	3.220841	13.074844	31.388694
	3.099396	13.47998	31.365162
	3.506054	13.53147	30.746564
	3.513133	13.084541	30.815993
	3.020911	12.861897	29.946358
	3.22173	12.057999	30.378768
	3.326679	13.015198	31.11753
	3.085201	12.964403	32.744399
	3.499273	13.506278	31.365184
	2.992052	13.393748	30.110316
	2.973974	13.836998	30.930488
	3.492059	12.762801	30.614399
	2.654794	12.748895	30.226891
	3.023299	13.558652	31.214147
	3.755137	12.714867	32.001152
	3.494769	13.128906	32.4109

Table 5: Matrix multiplication Python Implementation

Matrix Sizes:	**100x100**	**200x200**	**300x300**
	0.634618	4.226565	80.884085
	0.686314	4.169464	76.252826
	0.611927	4.17978	13.470691
	0.609929	4.139194	13.398051
	0.628872	4.230381	13.436706
	0.591303	4.123574	104.212466
	0.601589	10.157258	32.188974
	0.59802	32.993084	92.485045
	0.594539	31.698813	13.440018
	0.592593	32.779596	46.202237
	0.592781	32.217474	70.099361
	0.601899	32.444125	71.144054
	0.589982	32.637196	93.505406
	0.582742	31.433561	80.906047
	0.582709	31.337688	102.14281
	0.583396	32.359451	80.542982
	0.591556	32.13135	87.200445
	0.594789	32.472461	89.668143
	0.594805	32.836974	13.40146
	4.487732	32.699618	13.410983
	4.400547	32.142966	13.728436
	4.759758	33.099557	52.029933
	4.607481	32.893426	100.89493
	4.539068	32.644641	83.091854
	4.938422	33.172667	100.886375
	4.785919	32.386692	92.766221
	4.735507	33.047479	68.421573
	4.623052	32.476885	93.950026
	4.666353	32.070907	85.750144
	4.861641	32.5059	13.493572
	4.548224	32.111707	13.567254
	4.693067	32.647043	13.434095
	4.483967	33.010436	97.262246
	4.562177	32.203399	46.554875
	4.22807	32.371466	89.173944
	3.390078	32.876745	13.398946
	4.628255	32.974315	13.429061
	4.237638	32.534272	13.564468
	4.884654	32.246569	13.490892
	4.664318	32.099926	42.49791
	4.901105	33.073887	13.6075
	4.639144	33.041186	13.451657
	4.591811	32.738337	87.622907
	4.852883	31.925123	111.265894
	4.354151	32.235769	110.111745
	4.478949	32.084823	109.512837
	4.783421	32.50655	110.102198
	3.861483	32.630704	109.584503
	4.733594	33.322924	107.183834
	4.062916	32.149921	109.448405
	4.776892	32.438519	109.269848
	4.472756	32.715268	111.125856
	4.533182	31.543919	109.382151
	4.53521	32.644323	110.657246
	4.897007	32.753574	111.972132
	4.576198	33.432135	110.400754
	4.684645	33.045823	112.242894
	4.663233	30.923299	111.100571
	4.626494	31.998998	109.609064
	4.511574	32.485449	110.132318
	4.403948	32.814843	110.006356
	4.264034	32.670913	111.787046
	4.271005	31.798889	110.702547
	4.259259	33.101019	110.299827
	4.308773	32.416743	107.843771
	4.815232	31.945146	110.095276
	4.421798	32.629735	110.236272
	4.721168	32.495045	111.028202
	4.520051	32.282932	110.604044
	4.305917	32.161209	110.559552
	4.542406	32.396507	109.253615
	4.168897	31.711119	108.85602
	4.178622	31.697509	108.715919
	4.524082	32.515688	90.195175
	4.518607	31.139948	108.68562
	4.309967	32.021363	108.660868
	4.305158	32.683298	109.48112
	4.498335	33.00131	109.163032
	4.240917	32.395981	109.172816
	4.595933	31.497395	109.416129
	4.47828	30.980008	109.009582
	4.515908	32.394425	110.244817
	4.605611	32.19121	111.378844
	4.278671	33.43001	109.602217

Table 6: Matrix multiplication C Implementation

Margin of Error:	0.2511505879209499	1.0015398132216697	1.6975345837714586
Acceptable Margin of Error $\leq$ average * 10%:	0.2642656904761906	1.0941836797619047	2.9710687547619052
Average Time:	2.642656904761906	10.941836797619047	29.710687547619052

Table 7: Matrix multiplication Python Implementation

Margin of Error:	0.3552941680421538	1.6369015871802428	8.027252050329059
Acceptable Margin of Error $\leq$ average * 10%:	0.3628327595238096	3.0129135488095233	8.157336340476192
Average Time:	3.628327595238096	30.129135488095233	81.57336340476192

Sieve of Eratosthenes

An algorithm invented by Eratosthenes (3rd century BC) for finding the prime from natural numbers. [3] Each column consists of 84 experimental runs. We calculated the primes up to 100, 10,000, and 100,000.

Table 8: Sieve of Eratosthenes C Implementation

Prime number limits up to:	**1000**	**10000**	**100000**
	0.000496	0.000665	0.002709
	0.000373	0.000568	0.002618
	0.000436	0.000656	0.002627
	0.000365	0.000597	0.002703
	0.000459	0.00059	0.00274
	0.000427	0.000552	0.002678
	0.00048	0.000518	0.002692
	0.000396	0.00054	0.002554
	0.000407	0.000556	0.00287
	0.000424	0.000672	0.002699
	0.000454	0.000574	0.002821
	0.000436	0.000668	0.002877
	0.000578	0.000528	0.002676
	0.000586	0.000548	0.002756
	0.000387	0.000585	0.002697
	0.000462	0.000644	0.002942
	0.000388	0.000531	0.002952
	0.000509	0.000659	0.002648
	0.000368	0.000617	0.002621
	0.00042	0.000609	0.00275
	0.000517	0.000587	0.002899
	0.000417	0.000517	0.002945
	0.000407	0.000528	0.00266
	0.000453	0.000533	0.002704
	0.000331	0.000597	0.002841
	0.000432	0.000588	0.002743
	0.000495	0.000659	0.002688
	0.000424	0.000563	0.002675
	0.000348	0.000608	0.002777
	0.000405	0.000656	0.002673
	0.0004	0.000559	0.002963
	0.000434	0.000543	0.00266
	0.000425	0.000569	0.003006
	0.000564	0.000555	0.002727
	0.000412	0.000595	0.002918
	0.000378	0.000623	0.002649
	0.000437	0.000598	0.002654
	0.00046	0.000848	0.002657
	0.000377	0.000624	0.002694
	0.000351	0.000585	0.002947
	0.000355	0.000587	0.002978
	0.000469	0.000634	0.002685
	0.000424	0.000555	0.002678
	0.000529	0.000547	0.00277
	0.000376	0.000692	0.002858
	0.00041	0.000536	0.002788
	0.000435	0.000609	0.002746
	0.000352	0.000566	0.002599
	0.000394	0.000545	0.002624
	0.000601	0.000557	0.002709
	0.000353	0.000669	0.002616
	0.000492	0.00057	0.002568
	0.000677	0.000573	0.002676
	0.000503	0.000693	0.002656
	0.000413	0.000496	0.002683
	0.000452	0.000624	0.002775
	0.000424	0.000596	0.002895
	0.000511	0.000641	0.002567
	0.000439	0.000505	0.002892
	0.000371	0.000556	0.002679
	0.000364	0.000777	0.002681
	0.000393	0.000836	0.002662
	0.000462	0.000608	0.002721
	0.000405	0.000527	0.002596
	0.00044	0.00072	0.002641
	0.000411	0.00055	0.002708
	0.000439	0.000651	0.002546
	0.000446	0.000632	0.002729
	0.000513	0.000544	0.002548
	0.000466	0.000612	0.002693
	0.000461	0.000602	0.002766
	0.000628	0.000545	0.003215
	0.000729	0.000591	0.002782
	0.000542	0.000504	0.002869
	0.000436	0.000561	0.002739
	0.00057	0.000595	0.00266
	0.000426	0.000641	0.002728
	0.000618	0.000936	0.002691
	0.000432	0.000607	0.002633
	0.000511	0.000618	0.002684
	0.000476	0.00057	0.002806
	0.000488	0.000611	0.002674
	0.000384	0.000727	0.002657
	0.000457	0.000696	0.002714

Table 9: Sieve of Eratosthenes Python Implementation

Prime number limits up to:	**1000**	**10000**	**100000**
	0.012336	0.018171	0.060367
	0.012516	0.017776	0.058329
	0.012829	0.017749	0.057031
	0.012931	0.017756	0.057639
	0.013498	0.017562	0.057627
	0.013043	0.017077	0.057529
	0.013599	0.017725	0.109176
	0.013076	0.017315	0.058129
	0.012875	0.021639	0.058451
	0.012703	0.017566	0.05696
	0.012322	0.016953	0.057092
	0.012365	0.018536	0.056962
	0.012407	0.017291	0.059976
	0.013095	0.017973	0.058373
	0.013737	0.016923	0.056995
	0.012719	0.017072	0.058502
	0.013017	0.017402	0.057923
	0.012484	0.01751	0.057728
	0.012621	0.017953	0.058134
	0.012528	0.017991	0.056661
	0.013134	0.01784	0.057313
	0.013174	0.017551	0.127003
	0.012765	0.017016	0.057632
	0.012627	0.017306	0.057513
	0.013461	0.017054	0.059638
	0.012975	0.018402	0.057822
	0.013698	0.017439	0.058008
	0.013325	0.016853	0.057771
	0.013621	0.017066	0.057684
	0.013933	0.017467	0.056563
	0.013637	0.017063	0.058607
	0.012938	0.050214	0.056585
	0.012794	0.017073	0.057318
	0.012845	0.017538	0.057598
	0.013423	0.01708	0.057809
	0.013064	0.018533	0.056999
	0.012729	0.018914	0.056024
	0.012973	0.017229	0.056392
	0.013114	0.017222	0.057246
	0.012457	0.017005	0.057621
	0.013003	0.018465	0.058412
	0.012917	0.017378	0.058667
	0.013678	0.017769	0.058287
	0.013708	0.017995	0.058638
	0.013542	0.017269	0.056684
	0.013042	0.017883	0.060188
	0.012774	0.01688	0.058789
	0.013739	0.017287	0.059181
	0.012882	0.017976	0.05805
	0.012721	0.018297	0.057998
	0.012956	0.018188	0.058718
	0.014635	0.01846	0.058473
	0.013342	0.017974	0.057203
	0.014024	0.017732	0.056863
	0.012655	0.016717	0.057332
	0.014428	0.019058	0.058908
	0.012726	0.017598	0.061428
	0.012971	0.016975	0.058122
	0.012229	0.018216	0.058816
	0.012935	0.017517	0.058003
	0.013569	0.016884	0.058571
	0.012632	0.017014	0.059094
	0.012482	0.017533	0.056195
	0.013286	0.017717	0.05869
	0.013942	0.017598	0.058055
	0.012884	0.017656	0.056993
	0.01248	0.017558	0.057398
	0.013255	0.017087	0.057227
	0.012868	0.017318	0.057716
	0.012507	0.017338	0.057724
	0.013167	0.016822	0.057409
	0.013831	0.048732	0.05825
	0.013263	0.017231	0.057737
	0.01323	0.017226	0.057723
	0.013052	0.01725	0.05809
	0.014635	0.017358	0.060059
	0.012825	0.016998	0.057152
	0.013209	0.017312	0.058215
	0.013214	0.017181	0.058404
	0.012843	0.018318	0.057461
	0.013049	0.0174	0.057309
	0.01407	0.017603	0.057286
	0.012884	0.017116	0.05915
	0.012987	0.017354	0.058267

Table 10: Sieve of Eratosthenes C Implementation

Margin of Error:	0.000016257592056160246	0.000016281622560404245	0.000025822709542425057
Acceptable Margin of Error $\leq$ average * 10%:	0.0004499404761904763	0.0006050476190476191	0.002734464285714284
Average Time:	0.0004499404761904763	0.0006050476190476191	0.002734464285714284

Table 11: Sieve of Eratosthenes Python Implementation

Margin of Error:	0.00011048513869001597	0.001055886804452669	0.0020052305178177273
Acceptable Margin of Error $\leq$ average * 10%:	0.0013100702380952382	0.0018345750000000002	0.005937696428571432
Average Time:	0.013100702380952381	0.01834575	0.05937696428571432

Conclusion

Overall, typically the performance of threads in C completed faster than Python (multiprocess). One thing that I discovered important was my implementation of DNS Lookup was that it was faster in Python than C. From this I learned that your implementation in some programming language matters a lot. Especially in the world of threads were your implementation affects your performance greatly. Also, I learned that it takes a good amount of work to convert from C to Python because you have to learn the libraries and also structuring syntactically is different when trying to program what you want to do. Last, one of my favorite things in this project I learned was the importance of performance measuring and the great affect a system noise can have when timing preformance.

References

[1] *Can we clear memory cache in Ubuntu.* URL: `https://askubuntu.com/questions/1333592/can-we-clear-memory-cache-in-ubuntu`.

[2] *CSCI340-DNS-Name-Resolution-Engine.* URL: `https://github.com/CSUChico-CSCI340/CSCI340-DNS-Name-Resolution-Engine`.

[3] *Eratosthenes, sieve of.* URL: `https://encyclopediaofmath.org/wiki/Eratosthenes,_sieve_of`.

[4] *Google Cloud VM e2-micro Specifications.* URL: `https://gcloud-compute.com/us-central1/e2-micro.html`.

[5] *Matrix multiplication.* URL: `https://encyclopediaofmath.org/wiki/Matrix_multiplication`.