Multiprocessor Systems
Assignment 2

Submitted by: Chaithanya Gadiyam

Professor: Muhammad Shaaban

TAs: Stephen Moskal, Vignesh Kothandapani

Abstract

This project is an implementation of ray tracing algorithm, which is a highly parallel problem. A parallel program is developed, which can run on multiple processes, using Message Passing Interface (MPI). Different partitioning schemes such as Static partitioning using contiguous strips of rows, Static partitioning using square blocks, Static partitioning using cyclical assignments of columns, Dynamic partitioning using a centralized task queue are implemented in the parallel program. This parallel program is tested on two scenes, a simple scene and a complex scene. And results are listed down for these scenes for all partitioning schemes to determine which scheme gives the best performance increase over a sequential version of the program.

Design Methodology

Ray Tracing is a method of generating realistic images, in which the paths of individual rays of light are followed from the viewer to their points of origin. The core concept of any kind of ray tracing algorithm is to efficiently find intersections of a ray with a scene consisting of a set of geometric primitives. Since all pixels in the rendered image can be rendered independently of each other, this is a highly parallel problem. This project includes a parallel implementation of ray tracing algorithm using MPI, with four partitioning schemes, Static partitioning using contiguous strips of rows, Static partitioning using square blocks, Static partitioning using cyclical assignments of columns, Dynamic partitioning using a centralized task queue. For static partitioning using contiguous strips of rows, total number of rows are uniformly divided as strips of rows among all processes using rank of each process. Computation time of each slave process is sent to master process along with the computation done, in the same packet using MPI_Send. Master process receives and accumulates the computation time using MPI_Recv, measures communication time and communication to computation ratio. For static partitioning using cyclical assignments of columns, total number of columns are cyclically assigned to all process based on the provided cycle size using rank of each process. Similar to Static partitioning using contiguous strips of rows, computation time of each slave process is sent to master process along with the computation done, in the same packet. Master process measures the total communication time and based on the computation times received from slave processes and its own computation time, communication to computation ratio is calculated. For static partitioning using square blocks, based on totals number of processes the rendered image is divided into square blocks, of which each process is assigned one. In case of uneven distribution, the leftover work is carried out by master process. Since the leftover work would be small compared to the total work, dividing the leftover work among all processes would increase partitioning overhead and communication overhead. Hence it is designed in such a way that leftover work is assigned to master process. The leftover work in this partitioning scheme would be a strip of columns at the extreme right of the rendered image and a strip of rows at the top of the image. For dynamic partitioning using a centralized task queue, based on the provided block dimensions each slave process is initialized with a block. After completion of initial assigned block, each slave process communicates with master process using MPI_Send to request for a new block. Master process maintains a centralized task queue and continuously polls on all slave processes to assign blocks to requesting slave processes, until the queue becomes empty. Upon reception of request from slave processes, master process assigns a new block to that slave process if the queue is not empty, by sending the block offset coordinates using MPI_Send. If the queue is empty, master process sends a termination notification to the slave process using MPI_Send. If slave process receives termination notification, it sends its computed work to master

process. Master process saves all slaves computed work after sending termination notification to all slave processes. The leftover work due to uneven distribution is carried out by master process, as dividing this work among all processes would increase partitioning overhead and communication overhead. Similar to static partitioning using square blocks, the leftover work in this partitioning scheme would be a strip of columns at the extreme right of the rendered image and a strip of rows at the top of the image. Master process measures total communication time, accumulates total computation time from all slave processes along with its own computation time and calculates communication to computation ratio. Multiple parallel configurations are used to run the program on both simple scene and complex scene and the rendered images are compared with the rendered image of sequential program for correctness.

Results and Analysis

Different configurations of four partitioning schemes are considered to run the parallel program and the results including execution time, speedup, c-to-c ratio are listed in the following tables.

For simple scene:

Table 1: Sequential runtime

Sequential	
raytrace_seq	135.53 s

From Table 2, it can be inferred that execution time initially decreases with increase in number of processes and reaches a certain point (i.e. number of processes = 16), after which execution time increases. Usually increase in number of processes reduces the execution time because of more parallelization, but with limited parallel work available and with increase in number of processes leads to more communication time and hence increases execution time.

Table 2: The effect of work unit size on computational efficiency using strip allocation

Static Strip Allocation				
Number of Processes	Number of Strips	Execution Time (s)	Speedup	C-to-C ratio
1 (mpirun –np 1)	1	165.748	0.817687	0
2 (mpirun –np 2)	2	113.842	1.19051	0.308286
4 (mpirun –np 4)	4	76.3266	1.775659	0.371876
9 (mpirun –np 9)	9	69.7289	1.94367	0.261455
16 (mpirun –np 16)	16	43.2906	3.130703	0.182218
20 (mpirun –np 20)	20	44.3398	3.056622	0.207147
25 (mpirun –np 25)	25	58.6412	2.311174	0.267863
36 (mpirun –np 36)	36	84.1643	1.610303	0.345978
49 (mpirun –np 49)	49	127.804	1.060452	0.596901
55 (mpirun –np 55)	55	164.804	0.822371	0.754442
64 (mpirun –np 64)	64	177.679	0.76278	0.82785

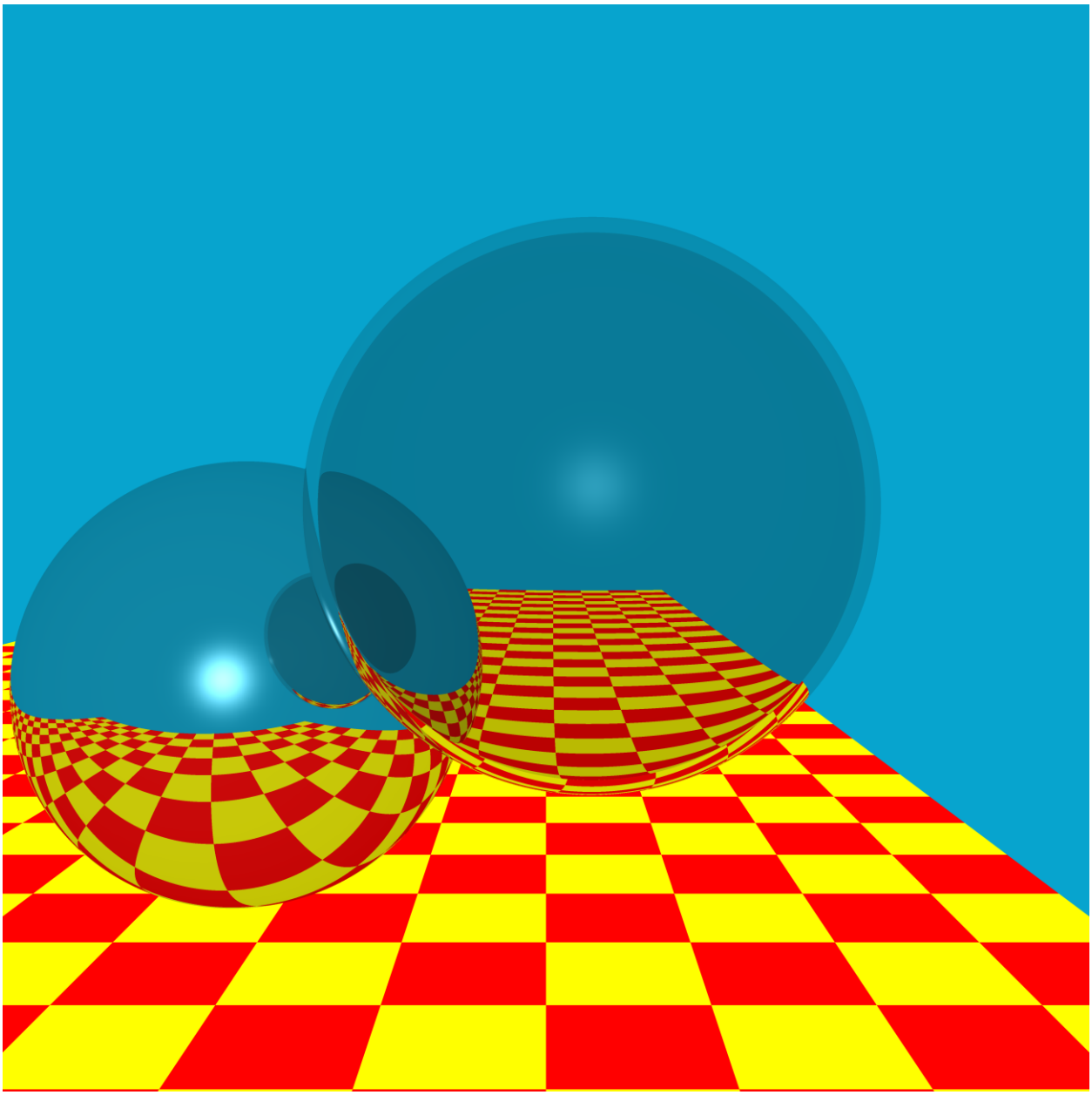

Figure 1. Sample rendered image of static strip allocation for simple scene

From Table 3, it can be seen that speedup follows a gausian curve, as it increases initially and decreases after a certain point. This is because of limited parallel computations and increasing communication time with increase in number of processes.

Table 3: The effect of work unit size on computational efficiency using block allocation

Static Block Allocation				
Number of Processes	Number of Blocks	Execution Time (s)	Speedup	C-to-C ratio
1 (mpirun –np 1)	1	168.856	0.80263657	0
4 (mpirun –np 4)	4	73.1855	1.85186956	0.237172
9 (mpirun –np 9)	9	50.7127	2.6725061	0.245044
16 (mpirun –np 16)	16	77.0212	1.7596454	0.281172
25 (mpirun –np 25)	25	57.2674	2.36661696	0.268723
36 (mpirun –np 36)	36	76.6869	1.76731619	0.368816
49 (mpirun –np 49)	49	152.088	0.89112882	0.614818
64 (mpirun –np 64)	64	181.089	0.74841652	0.832628

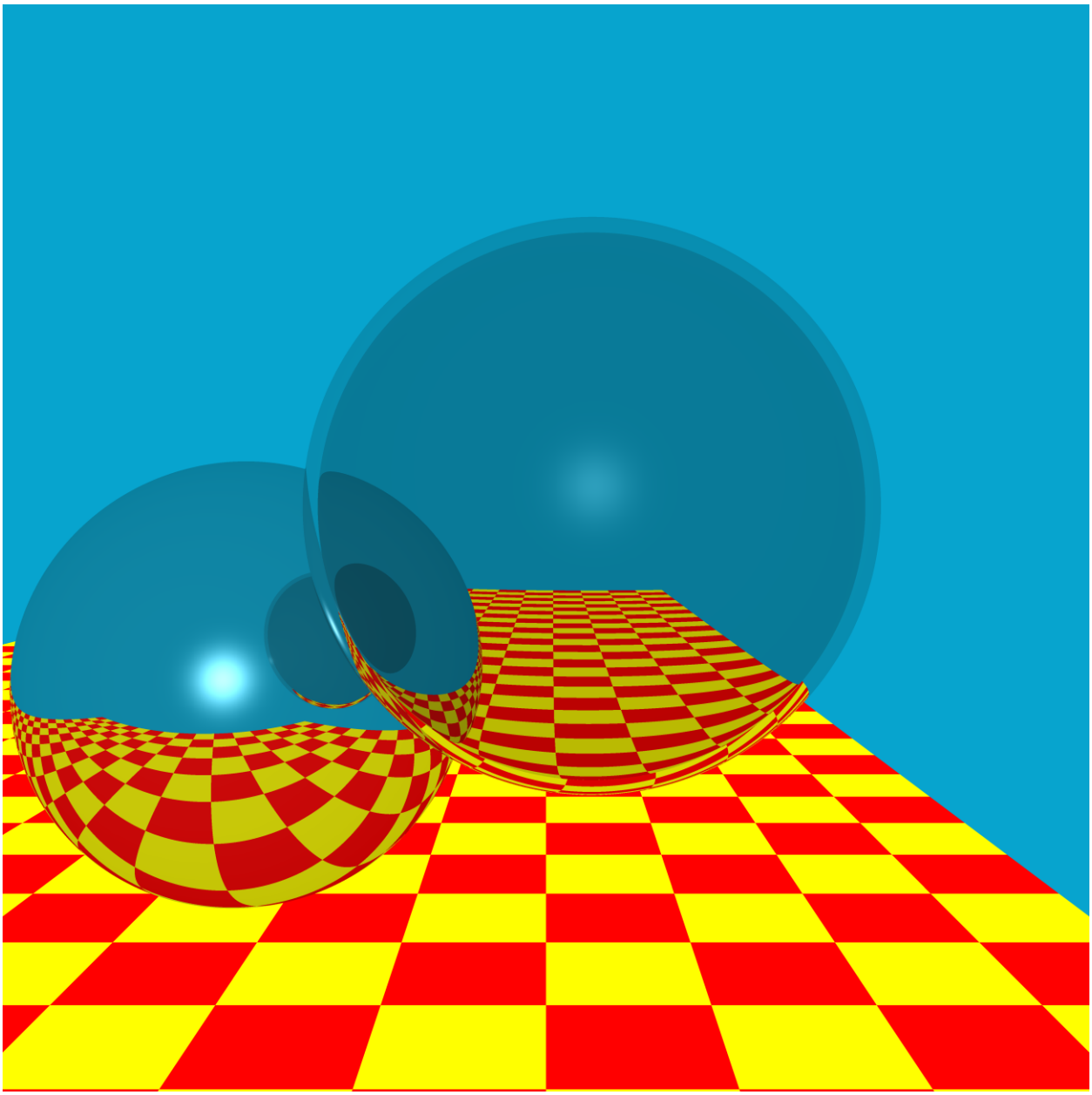

Figure 2. Sample rendered image of static block allocation for simple scene

From Table 4 and Figures 3, 4 & 5, it can be observed that the execution time for 4 processes increases gradually as the cycle size increases. Execution time for 9 processes and 16 processes does not follow a particular pattern, but average execution time for 9 processes and for 16 processes is lower compared to average execution time for 4 processes. This is because more parallelization happens with increase in number of processes.

Table 4: The effect of work unit size on computational efficiency using cyclical allocation

Static Cyclical Allocation				
Number of Processes	Width of strip in pixels	Execution Time (s)	Speedup	C-to-C ratio
4 (mpirun –np 4)	1	47.9352	2.8273586	0.00245732
4 (mpirun –np 4)	5	48.0066	2.82315348	0.00253431
4 (mpirun –np 4)	10	45.2303	2.99644265	0.00270251
4 (mpirun –np 4)	20	47.9588	2.82596729	0.00335414
4 (mpirun –np 4)	80	46.8854	2.89066532	0.00268441
4 (mpirun –np 4)	320	48.3981	2.80031654	0.0243248
4 (mpirun –np 4)	640	50.805	2.66765082	0.00264646
4 (mpirun –np 4)	1280	64.7613	2.09276219	0.161754
9 (mpirun –np 9)	1	31.5796	4.29169464	0.00793636
9 (mpirun –np 9)	5	32.6012	4.15720894	0.00848421
9 (mpirun –np 9)	10	28.0969	4.82366382	0.0195065
9 (mpirun –np 9)	20	31.8584	4.25413706	0.0280107
9 (mpirun –np 9)	40	23.7268	5.71210614	0.0070718
9 (mpirun –np 9)	160	37.9209	3.57401855	0.0381525
9 (mpirun –np 9)	400	30.4895	4.44513685	0.0202692
9 (mpirun –np 9)	600	35.9627	3.76862694	0.097928
16 (mpirun –np 16)	1	28.8184	4.70289815	0.0690923
16 (mpirun –np 16)	5	35.8383	3.7817084	0.0797994
16 (mpirun –np 16)	10	37.2249	3.64084255	0.0907926
16 (mpirun –np 16)	20	44.4251	3.05075284	0.0784461
16 (mpirun –np 16)	50	28.299	4.78921517	0.069075
16 (mpirun –np 16)	100	33.4675	4.04960036	0.0744985
16 (mpirun –np 16)	300	40.6064	3.33765121	0.108045
16 (mpirun –np 16)	400	41.414	3.27256483	0.142729

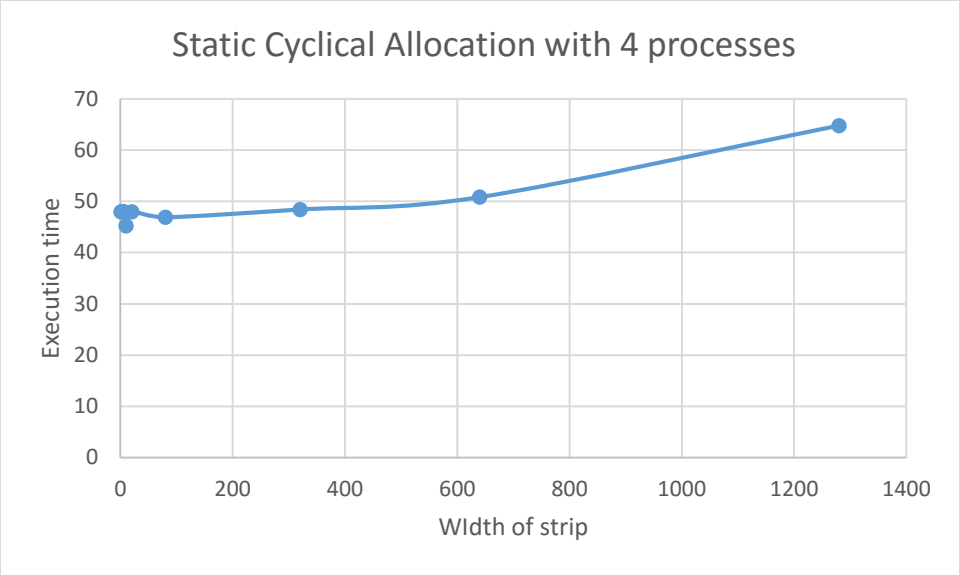

Figure 3. Plot of execution time vs width of strip for static cyclical allocation with 4 processes

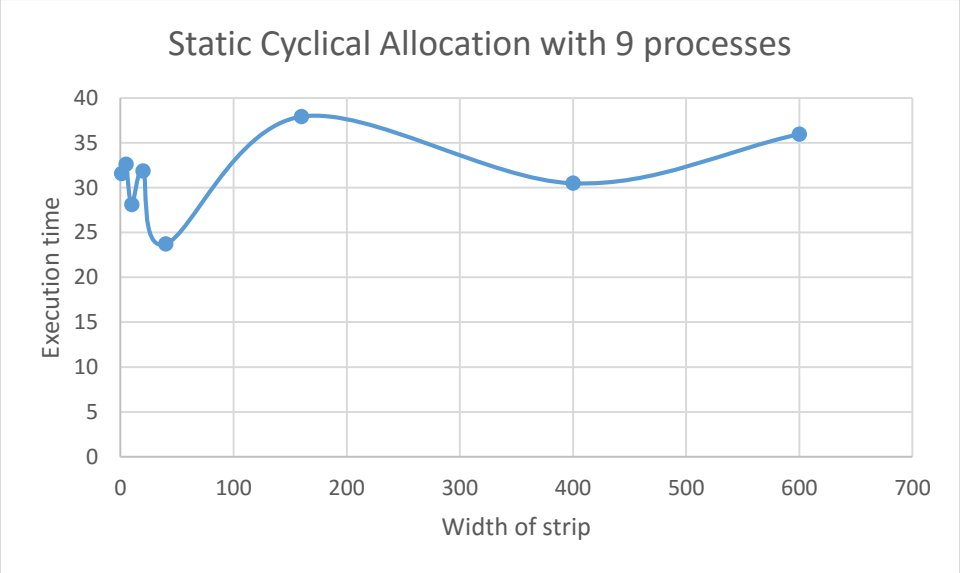

Figure 4. Plot of execution time vs width of strip for static cyclical allocation with 9 processes

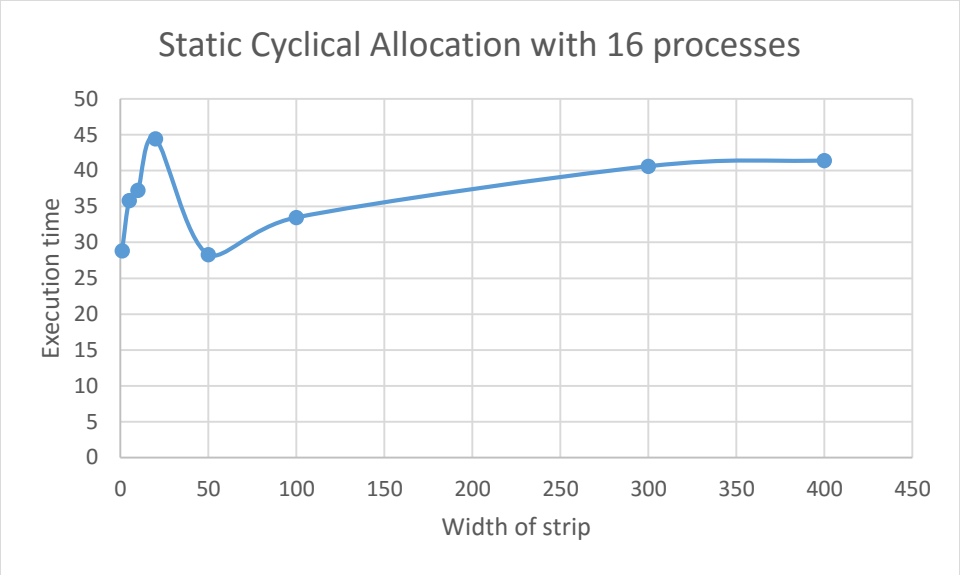

Figure 5. Plot of execution time vs width of strip for static cyclical allocation with 16 processes

As mentioned earlier, static cyclical allocation in Table 5 shows that speedup increases initially with increase in number of processes, but after reaching a certain point speedup decreases with increase in number of processes. This is because of increase in communication time with increase in number of processes and limited parallel computation.

Table 5. The effect of increasing the number of processors on a fixed allocation size

Static Cyclical Allocation				
Number of Processes	Width of strip in pixels	Execution Time (s)	Speedup	C-to-C ratio
1 (mpirun –np 1)	23	163.299	0.82994997	0
2 (mpirun –np 2)	23	88.8329	1.52567348	0.000918882
4 (mpirun –np 4)	23	47.9309	2.82761225	0.00260857
9 (mpirun –np 9)	23	24.016	5.64332112	0.00703145
16 (mpirun –np 16)	23	38.7237	3.49992382	0.0793538
20 (mpirun –np 20)	23	37.7993	3.58551613	0.126187
25 (mpirun –np 25)	23	62.3475	2.17378403	0.22013
36 (mpirun –np 36)	23	98.8243	1.37142383	0.351643
49 (mpirun –np 49)	23	157.436	0.86085775	0.630836
55 (mpirun –np 55)	23	188.031	0.7207854	0.740374
64 (mpirun –np 64)	23	180.277	0.75178753	0.70917

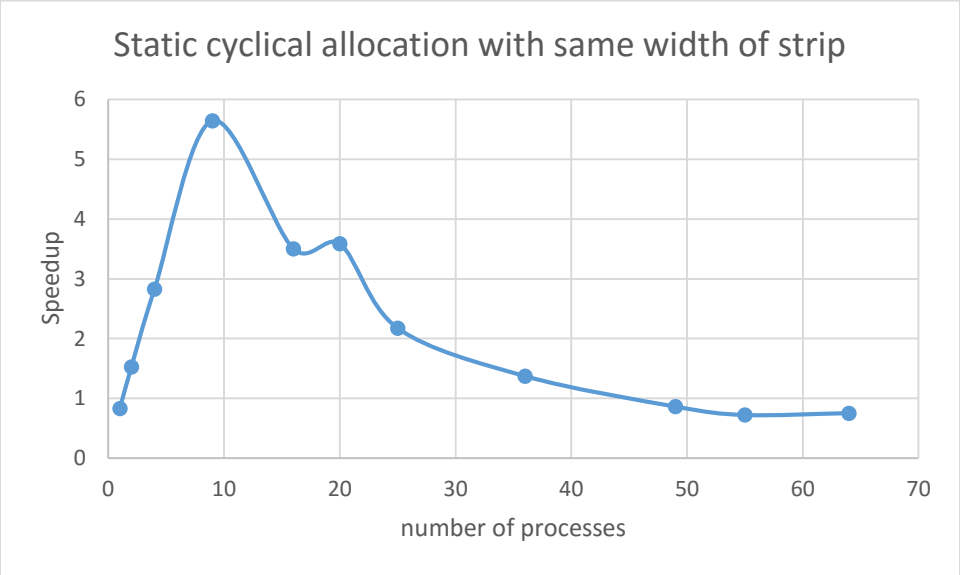

Figure 6. Plot of speedup vs number of processes for static cyclical allocation with same width of strip

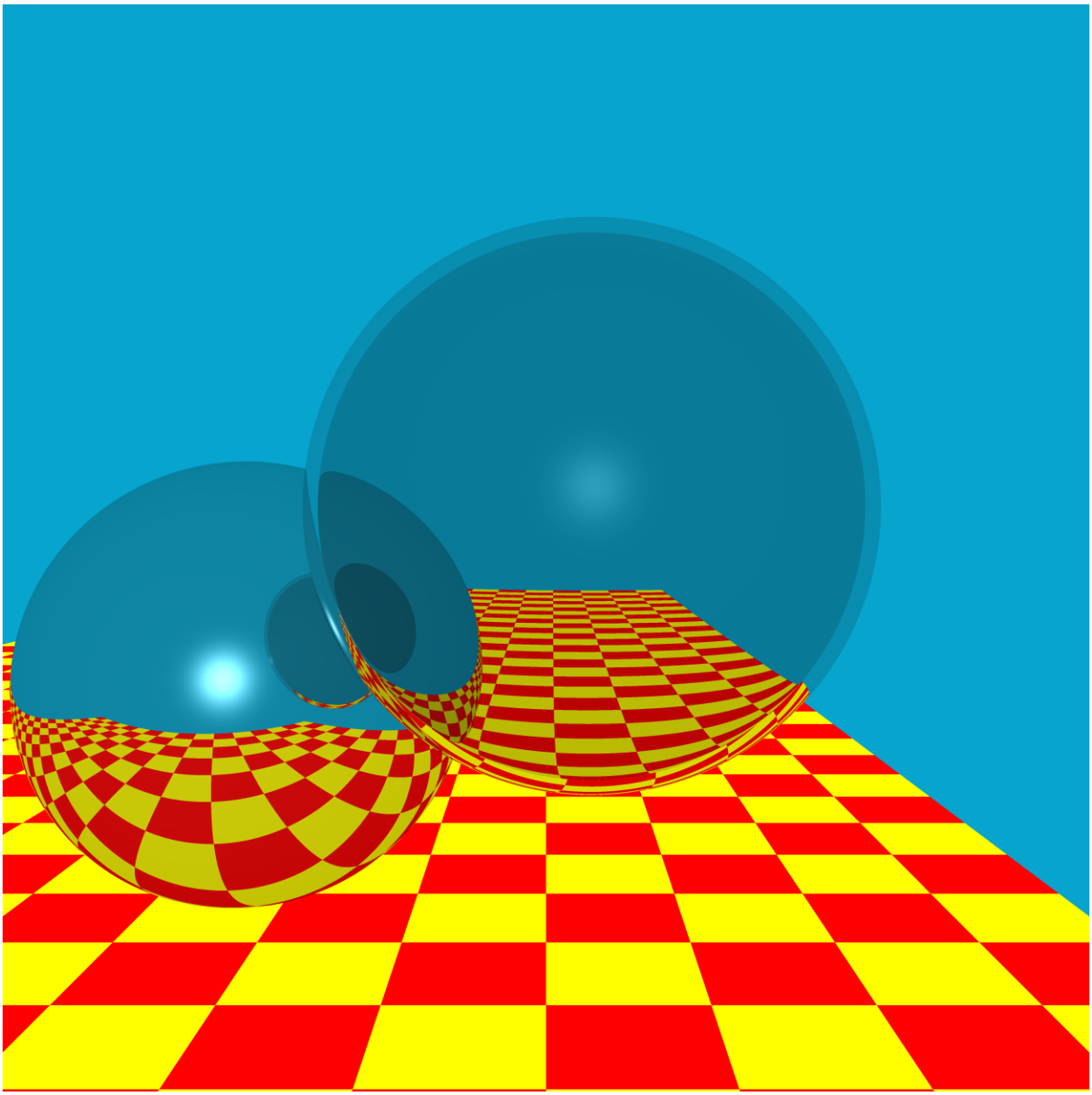

Figure 7. Sample rendered image of static cyclical allocation for simple scene

Table 4: The effect of work unit size on computational efficiency using dynamic block allocation

Dynamic Allocation				
Number of processes	Work Unit Size (rows x columns)	Execution Time (s)	Speedup	C-to-C Ratio
9 (mpirun –np 9)	1x1	271.024	0.50006641	0.497851
9 (mpirun –np 9)	10x10	30.8079	4.39919631	0.144815
9 (mpirun –np 9)	30x30	29.4996	4.59429958	0.137323
9 (mpirun –np 9)	50x50	31.0677	4.36240855	0.13792
9 (mpirun –np 9)	75x75	27.3032	4.96388702	0.130501
9 (mpirun –np 9)	100x100	34.2471	3.95741537	0.132964
16 (mpirun –np 16)	1x1	146.377	0.92589683	0.50023
16 (mpirun –np 16)	10x10	29.8639	4.53825522	0.141843
16 (mpirun –np 16)	30x30	32.945	4.11382607	0.137478
16 (mpirun –np 16)	50x50	33.6644	4.02591462	0.1427
16 (mpirun –np 16)	75x75	30.3106	4.47137305	0.137045
16 (mpirun –np 16)	100x100	35.4091	3.82754716	0.151076
25 (mpirun –np 25)	1x1	332.03	0.40818601	1.10615
25 (mpirun –np 25)	10x10	64.549	2.09964523	0.263178
25 (mpirun –np 25)	30x30	55.7344	2.43171183	0.247369
25 (mpirun –np 25)	50x50	61.2436	2.21296593	0.254519
25 (mpirun –np 25)	75x75	58.8663	2.30233597	0.258806
25 (mpirun –np 25)	100x100	116.988	1.15849489	0.372883

Table 7: The effect of work unit shape on computational efficiency as well as test for arbitrary work unit size

Dynamic Allocation				
Number of processes	Work Unit Size (rows x columns)	Execution Time (s)	Speedup	C-to-C Ratio
16 (mpirun –np 16)	11x1	136.338	0.99407355	0.310278
16 (mpirun –np 16)	10x1	52.6862	2.57240036	0.204158
16 (mpirun –np 16)	7x13	41.4086	3.2729916	0.142802
16 (mpirun –np 16)	5x29	29.8326	4.5430167	0.140651
16 (mpirun –np 16)	11x43	26.5028	5.1137993	0.130291
16 (mpirun –np 16)	67x1	29.4572	4.60091251	0.131161
36 (mpirun –np 36)	11x1	174.049	0.77868876	0.669719
36 (mpirun –np 36)	10x1	127.263	1.06495996	0.510092
36 (mpirun –np 36)	7x13	189.598	0.71482822	0.668864
36 (mpirun –np 36)	5x29	97.1505	1.39505201	0.404883
36 (mpirun –np 36)	11x43	124.893	1.0851689	0.494167
36 (mpirun –np 36)	67x1	101.376	1.3369042	0.397713

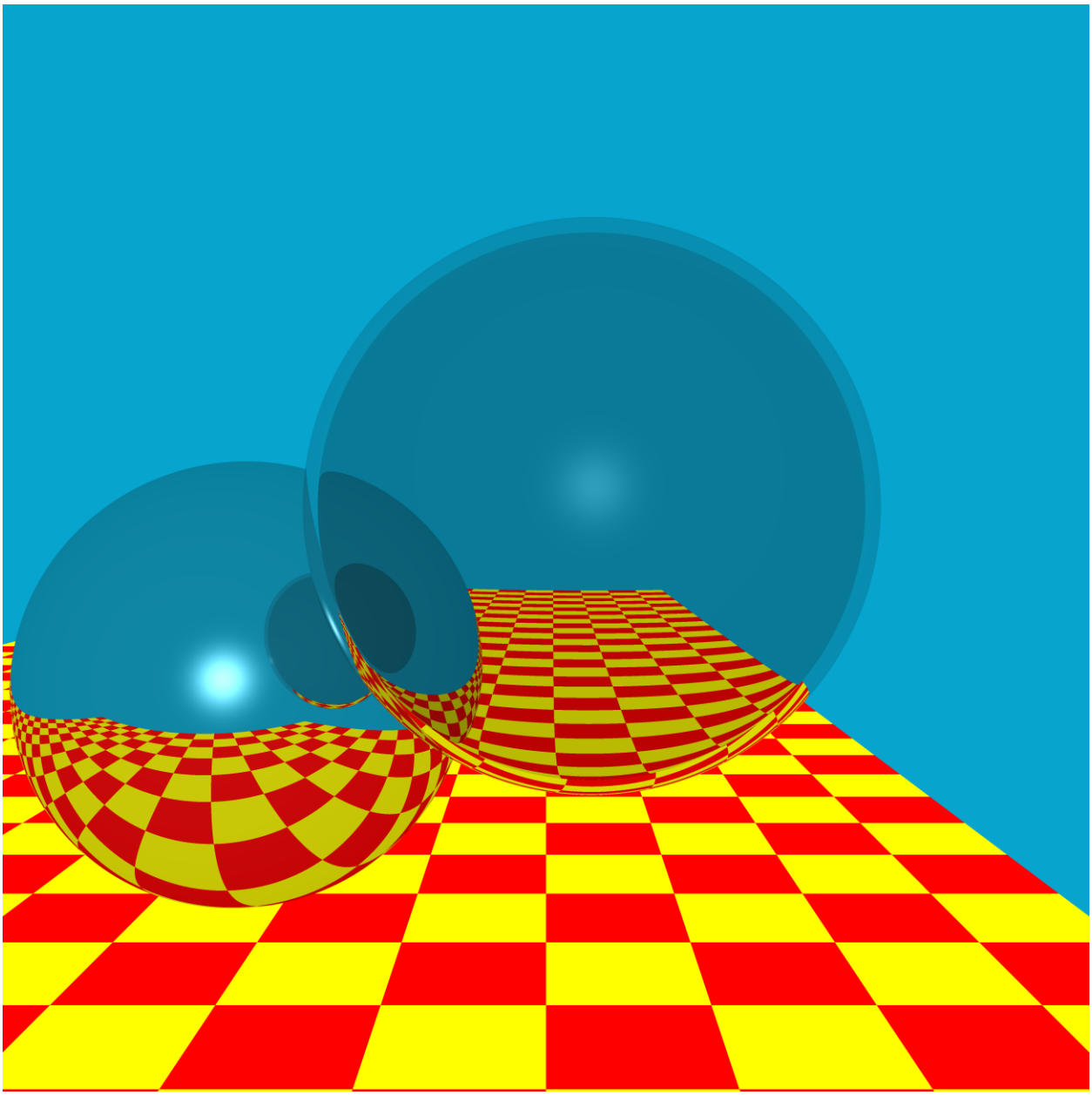

Figure 8. Sample rendered image of dynamic block allocation for simple scene

Overall, for simple scene static cyclical allocation performs better with the evidence of high speedups. Dynamic allocation will perform better if the scene is more complex.

For Complex scene:

Table 8: Sequential runtime

Sequential	
raytrace_seq	5447.42

From Table 9, it can inferred that speedup increases with increase in number of processes as more amount of work is getting parallelized with increase in number of processes. Also communication to computation ratio is low for higher number of processes.

Table 9: The effect of work unit size on computational efficiency using strip allocation

Static Strip Allocation				
Number of Processes	Number of Strips	Execution Time (s)	Speedup	C-to-C ratio
1 (mpirun –np 1)	1	5393.79	1.00994292	0
2 (mpirun –np 2)	2	5207.55	1.04606197	0.92475
4 (mpirun –np 4)	4	3162.07	1.72273859	0.568788
9 (mpirun –np 9)	9	4002.2	1.36110639	0.533488
16 (mpirun –np 16)	16	3542.03	1.53793728	0.548082
20 (mpirun –np 20)	20	1663.55	3.27457546	0.281674
25 (mpirun –np 25)	25	1617.07	3.36869771	0.27856
36 (mpirun –np 36)	36	1553.41	3.50674967	0.251085
49 (mpirun –np 49)	49	1317.16	4.13573142	0.192703
55 (mpirun –np 55)	55	1154.09	4.72009982	0.18359
64 (mpirun –np 64)	64	1296.84	4.2005336	0.182041

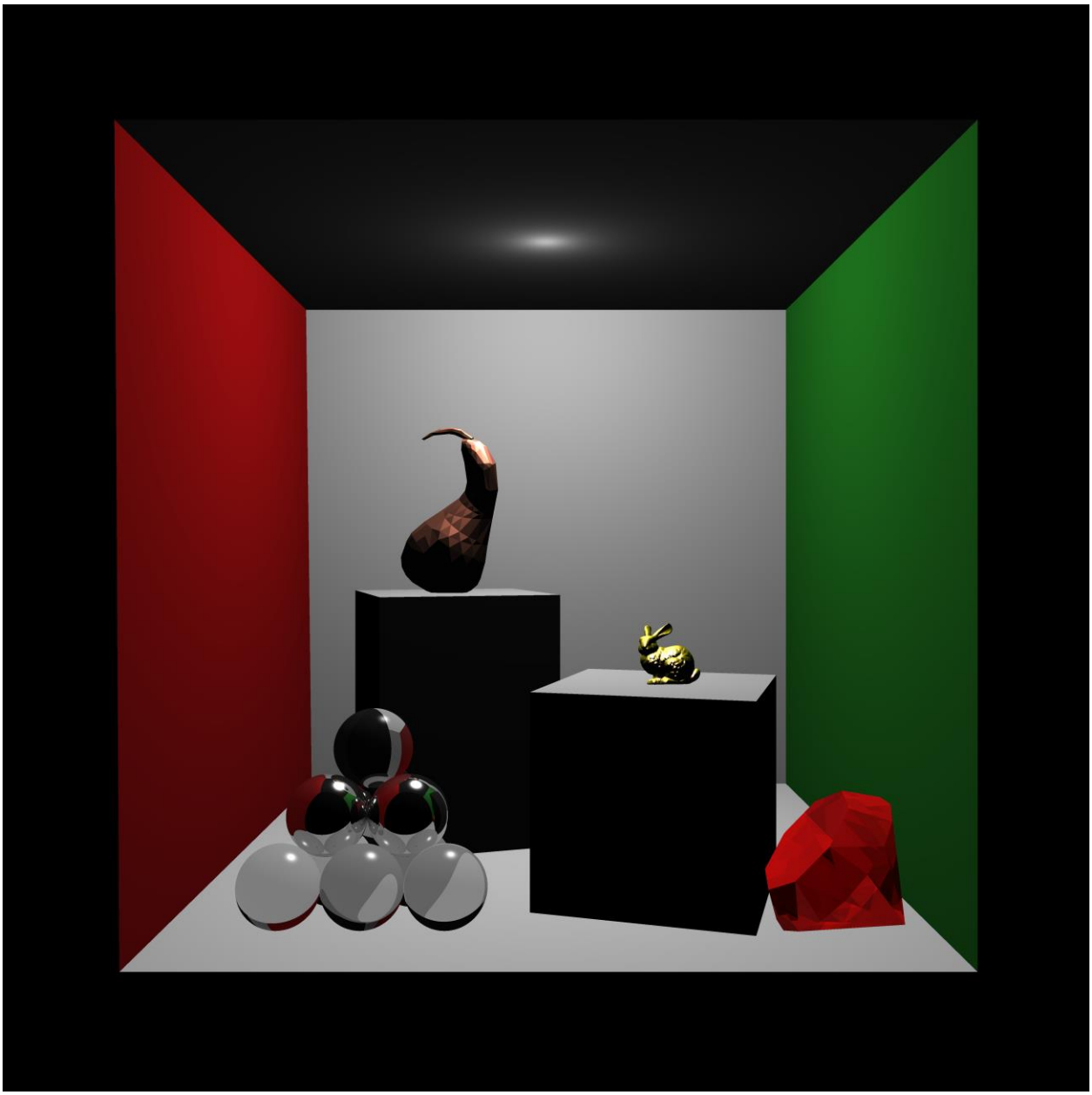

Figure 9. Sample rendered image of static strip allocation for complex scene

Table 10: The effect of work unit size on computational efficiency using block allocation

Static Block Allocation				
Number of Processes	Number of Blocks	Execution Time (s)	Speedup	C-to-C ratio
1 (mpirun –np 1)	1	5572.96	0.97747337	0
4 (mpirun –np 4)	4	4945.98	1.10138335	0.905615
9 (mpirun –np 9)	9	4191.83	1.29953266	0.54526
16 (mpirun –np 16)	16	3652.08	1.49159383	0.56461
25 (mpirun –np 25)	25	2106.79	2.58564926	0.343012
36 (mpirun –np 36)	36	5101.37	1.06783472	0.639304
49 (mpirun –np 49)	49	3166.69	1.72022522	0.489146
64 (mpirun –np 64)	64	2641.64	2.06213564	0.379507

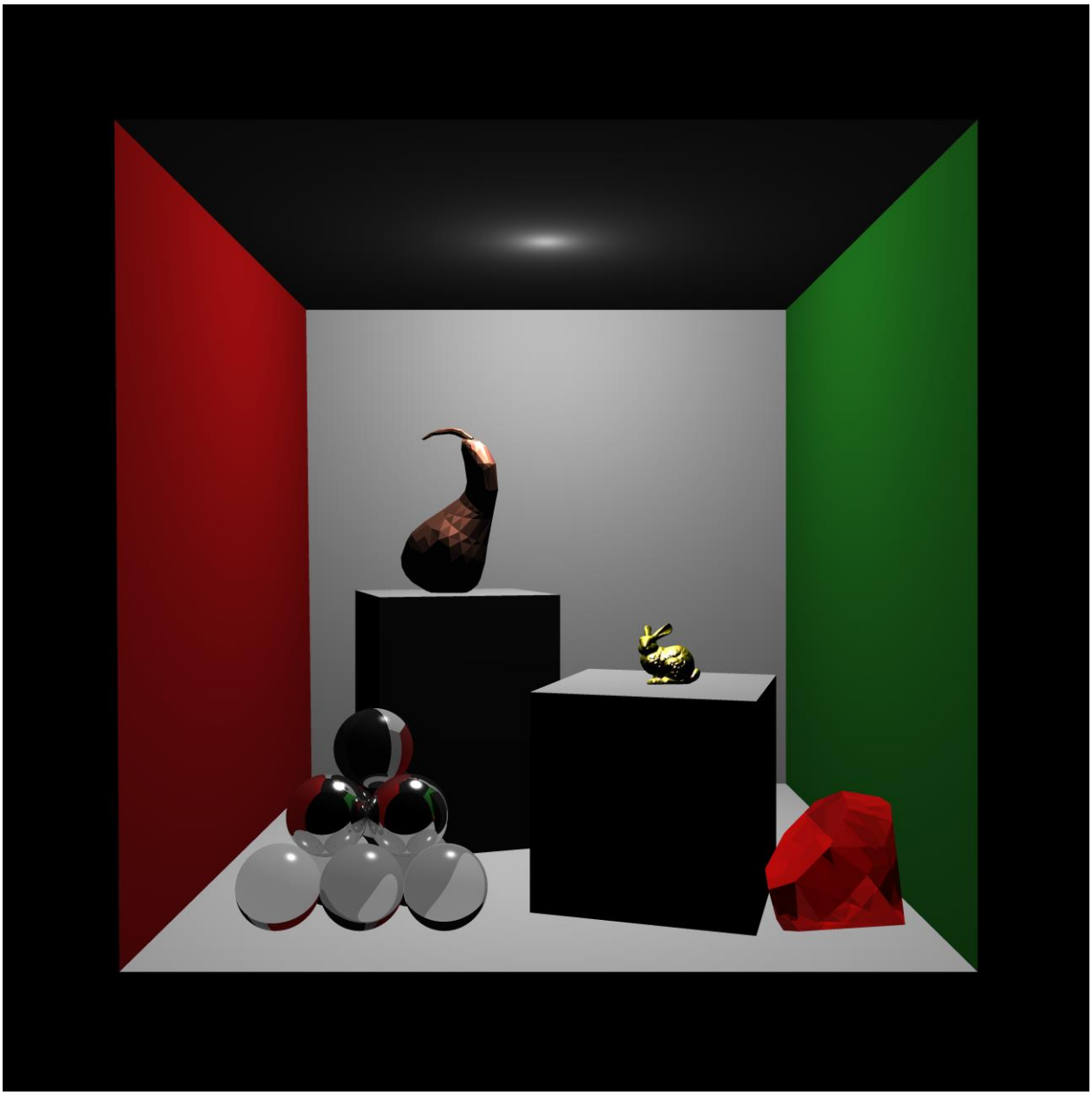

Figure 10. Sample rendered image of static block allocation for complex scene

Table 11: The effect of work unit size on computational efficiency using cyclical allocation

Static Cyclical Allocation				
Number of Processes	Width of strip in pixels	Execution Time (s)	Speedup	C-to-C ratio
4 (mpirun –np 4)	1	1449.41	3.75837065	0.00444997
4 (mpirun –np 4)	5	1480.27	3.68001783	0.00563358
4 (mpirun –np 4)	10	1529.66	3.56119661	8.3288e-05
4 (mpirun –np 4)	20	1463.47	3.72226284	8.46811e-05
4 (mpirun –np 4)	80	1911.09	2.85042567	0.129014
4 (mpirun –np 4)	320	2754.47	1.97766539	0.244539
4 (mpirun –np 4)	640	2677.8	2.03428934	8.7125e-05
4 (mpirun –np 4)	1280	3772.86	1.44384366	0.679656
9 (mpirun –np 9)	1	665.643	8.18369607	0.000706638
9 (mpirun –np 9)	5	678.438	8.02935567	0.00290821
9 (mpirun –np 9)	10	1808.51	3.01210389	0.00299003
9 (mpirun –np 9)	20	845.161	6.44542282	0.0630678
9 (mpirun –np 9)	40	1132.13	4.81165591	0.120787
9 (mpirun –np 9)	160	1674.4	3.25335643	0.0740204
9 (mpirun –np 9)	400	2828.55	1.92587015	0.186483
9 (mpirun –np 9)	600	2662.37	2.04607925	0.399746
16 (mpirun –np 16)	1	697.215	7.8131136	0.0113544
16 (mpirun –np 16)	5	577.827	9.42742378	0.00853025
16 (mpirun –np 16)	10	1304.12	4.17708493	0.0179203
16 (mpirun –np 16)	20	755.422	7.21109526	0.0315114
16 (mpirun –np 16)	50	932.894	5.83927006	0.0581104
16 (mpirun –np 16)	100	1091.23	4.99199985	0.116917
16 (mpirun –np 16)	300	2346.07	2.32193413	0.387791
16 (mpirun –np 16)	400	6896.53	0.78987839	0.438688

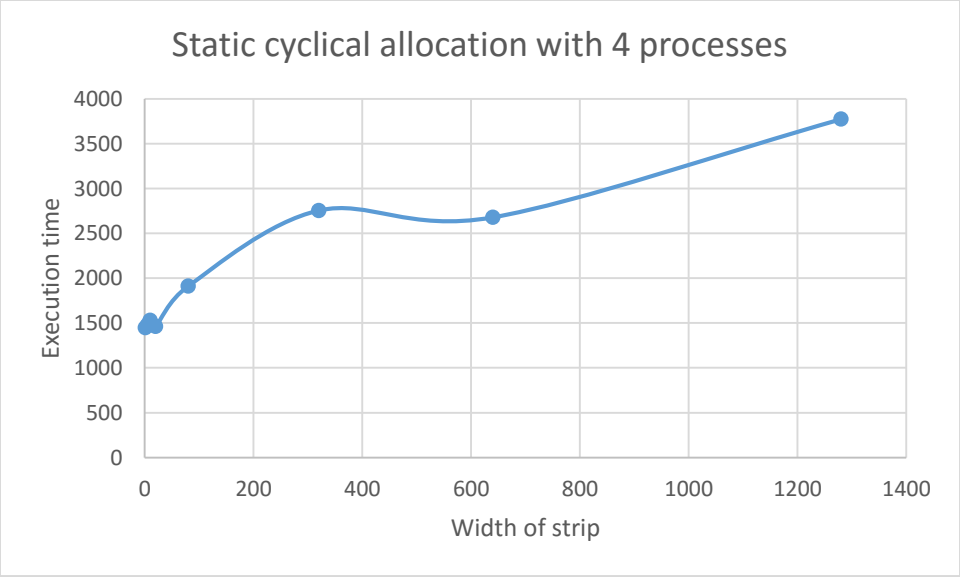

Figure 11. Plot of execution time vs width of strip for static cyclical allocation with 4 processes

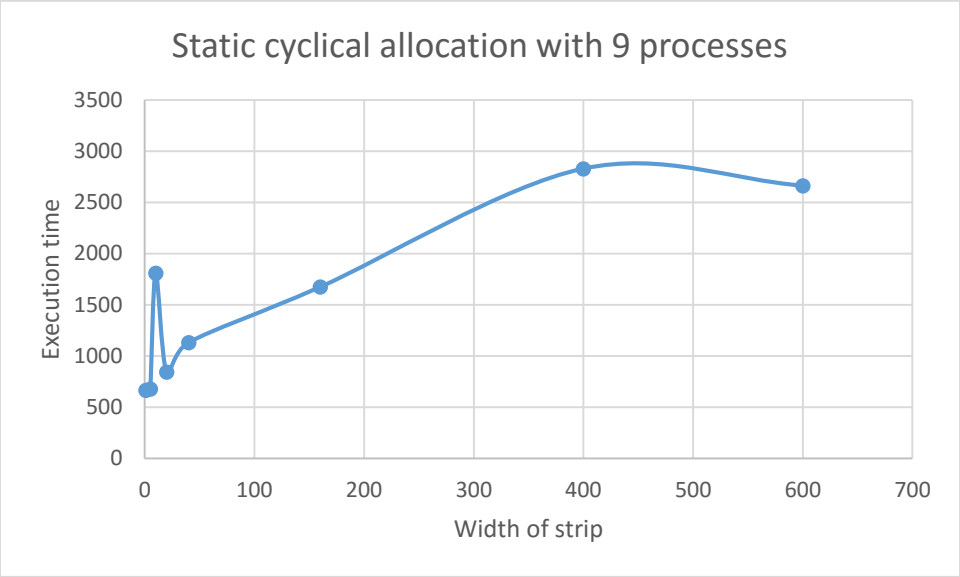

Figure 12. Plot of execution time vs width of strip for static cyclical allocation with 9 processes

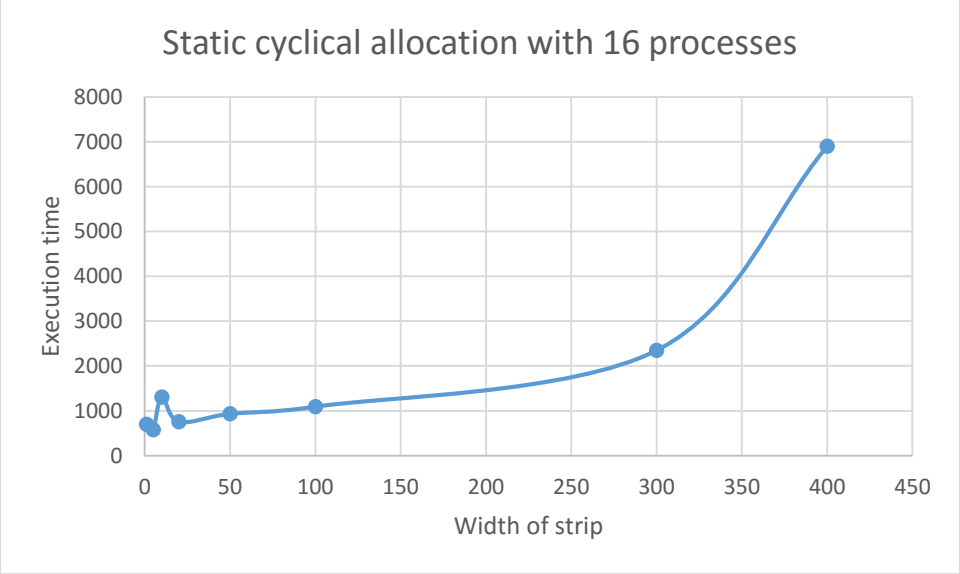

Figure 13. Plot of execution time vs width of strip for static cyclical allocation with 16 processes

Table 12. The effect of increasing the number of processors on a fixed allocation size

Static Cyclical Allocation				
Number of Processes	Width of strip in pixels	Execution Time (s)	Speedup	C-to-C ratio
1 (mpirun –np 1)	23	5680.34	0.95899541	0
2 (mpirun –np 2)	23	2735.22	1.99158386	0.0409636
4 (mpirun –np 4)	23	1546.52	3.52237281	0.0303909
9 (mpirun –np 9)	23	785.716	6.93306487	0.0573882
16 (mpirun –np 16)	23	826.196	6.59337494	0.022095
20 (mpirun –np 20)	23	590.952	9.2180414	0.0505451
25 (mpirun –np 25)	23	617.846	8.81679253	0.0497983
36 (mpirun –np 36)	23	521.623	10.4432128	0.0694589
49 (mpirun –np 49)	23	611.455	8.90894669	0.0726457
55 (mpirun –np 55)	23	757.937	7.18716727	0.0487059
64 (mpirun –np 64)	23	1079.92	5.04428106	0.0413108

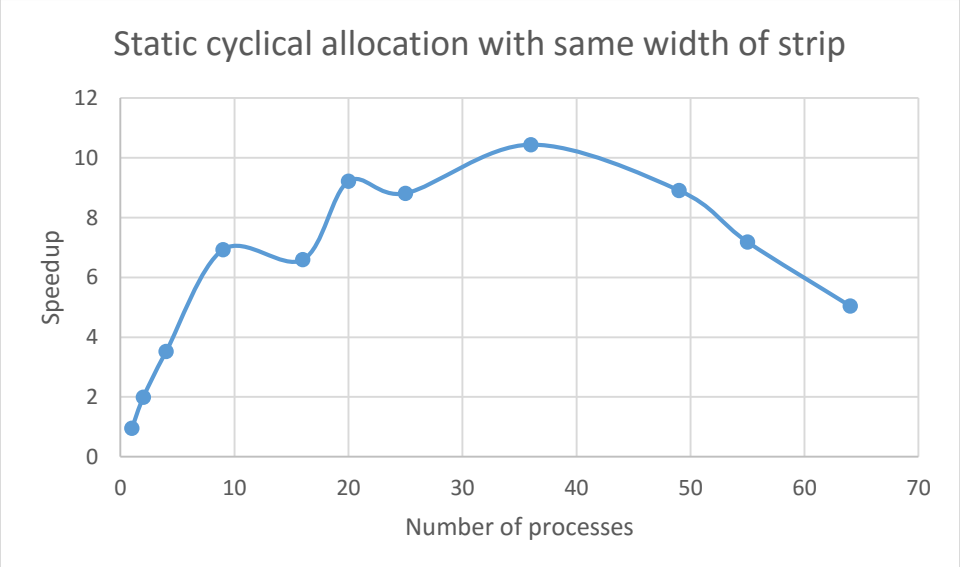

Figure 14. Plot of speedup vs number of processes for static cyclical allocation with same width of strip

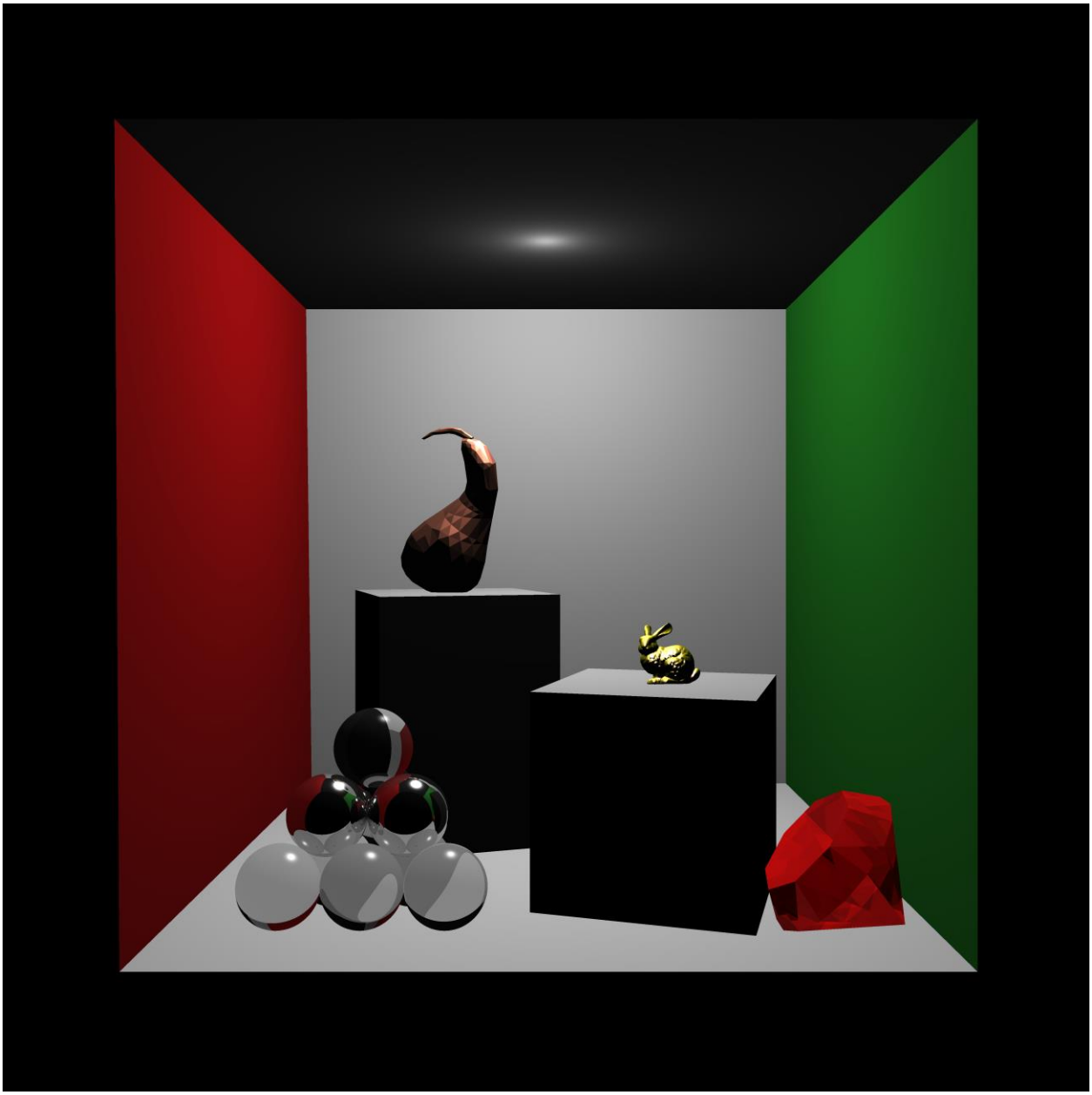

Figure 15. Sample rendered image of static cyclical allocation for complex scene

Table 13: The effect of work unit size on computational efficiency using dynamic block allocation

Dynamic Allocation				
Number of processes	Work Unit Size (rows x columns)	Execution Time (s)	Speedup	C-to-C Ratio
9 (mpirun –np 9)	1x1	1128.64	4.8265346	0.138961
9 (mpirun –np 9)	10x10	785.263	6.9370644	0.126046
9 (mpirun –np 9)	30x30	796.891	6.8358408	0.136177
9 (mpirun –np 9)	50x50	796.94	6.8354205	0.136392
9 (mpirun –np 9)	75x75	883.114	6.1684222	0.134702
9 (mpirun –np 9)	100x100	1009.91	5.3939658	0.146007
16 (mpirun –np 16)	1x1	1414.77	3.8503926	0.125456
16 (mpirun –np 16)	10x10	477.799	11.40107	0.070041
16 (mpirun –np 16)	30x30	517.8	10.520317	0.0777253
16 (mpirun –np 16)	50x50	532.184	10.235971	0.0779431
16 (mpirun –np 16)	75x75	596.407	9.1337291	0.0806633
16 (mpirun –np 16)	100x100	643.757	8.4619196	0.0943097
25 (mpirun –np 25)	1x1	611.168	8.9131303	0.0961366
25 (mpirun –np 25)	10x10	469.614	11.599782	0.0550454
25 (mpirun –np 25)	30x30	388.8	14.010854	0.0514429
25 (mpirun –np 25)	50x50	341.837	15.935724	0.0540198
25 (mpirun –np 25)	75x75	628.612	8.6657907	0.0627276
25 (mpirun –np 25)	100x100	614.974	8.857968	0.0759441

Table 14: The effect of work unit shape on computational efficiency as well as test for arbitrary work unit size

Dynamic Allocation				
Number of processes	Work Unit Size (rows x columns)	Execution Time (s)	Speedup	C-to-C Ratio
16 (mpirun –np 16)	11x1	881.817	6.17749488	0.0806639
16 (mpirun –np 16)	10x1	507.51	10.733621	0.0728244
16 (mpirun –np 16)	7x13	559.148	9.74235802	0.0697865
16 (mpirun –np 16)	5x29	425.473	12.8032096	0.0690307
16 (mpirun –np 16)	11x43	428.934	12.6999025	0.0714562
16 (mpirun –np 16)	67x1	431.672	12.6193499	0.0694314
36 (mpirun –np 36)	11x1	309.672	17.5909349	0.0433903
36 (mpirun –np 36)	10x1	326.553	16.68158	0.0450412
36 (mpirun –np 36)	7x13	289.03	18.8472477	0.0408557
36 (mpirun –np 36)	5x29	428.471	12.7136259	0.0507797
36 (mpirun –np 36)	11x43	293.561	18.5563477	0.0395009
36 (mpirun –np 36)	67x1	293.012	18.5911157	0.0409185

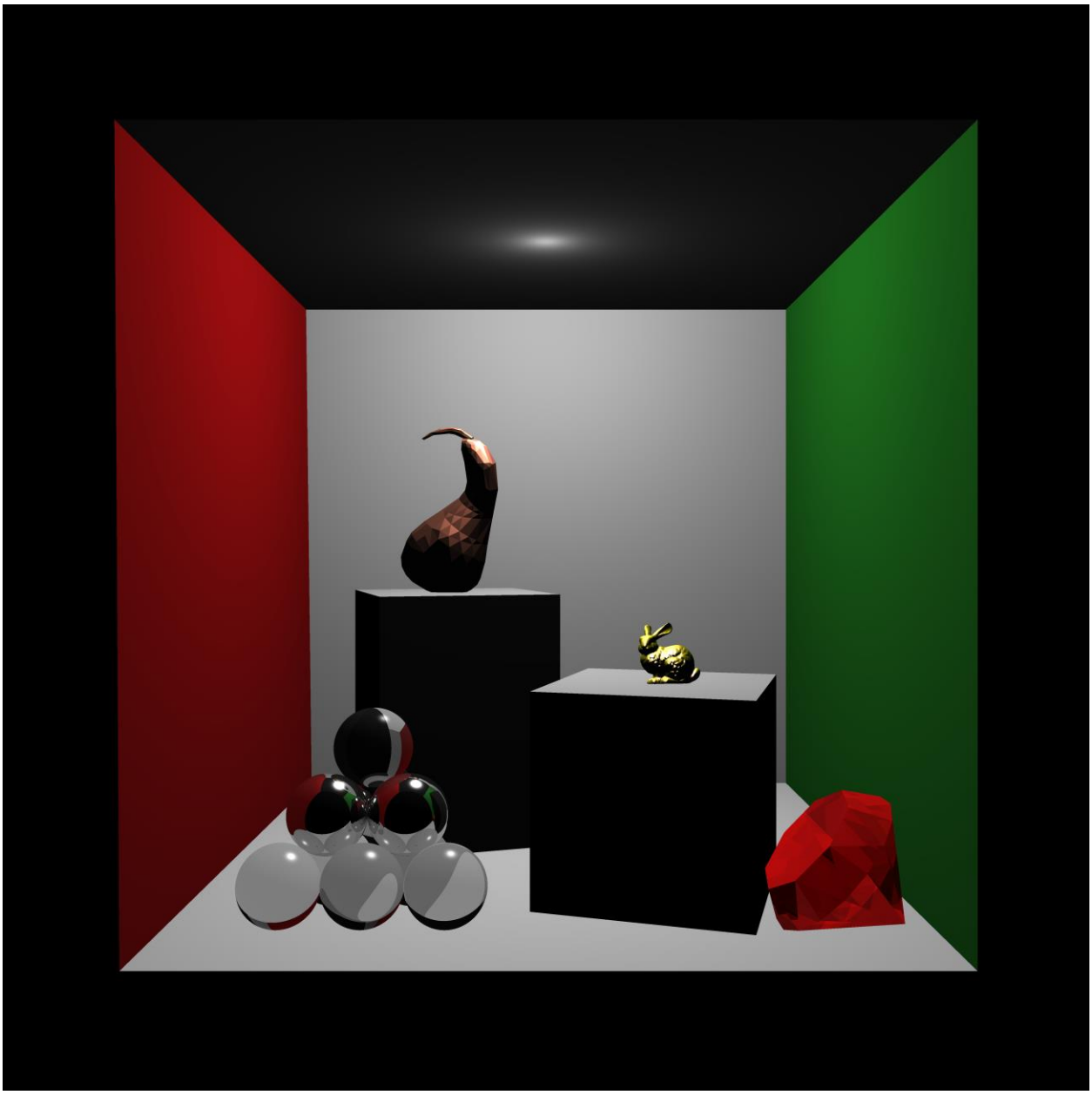

Figure 16. Sample rendered image of dynamic block allocation for complex scene

On a whole, it can observed that for complex scene dynamic allocation performs better than remaining partitioning schemes with a highest speedup of 18.84, with 36 processes. Since ray tracing is a recursive algorithm and with the presence of more number of transparent or reflective objects in the scene, execution time per ray is unpredictable. Hence in this case, dynamic allocation would be the best option.

Conclusion

Parallel implementation of ray tracing algorithm using MPI is done with four partitioning schemes: Static partitioning using contiguous strips of rows, Static partitioning using square blocks, Static partitioning using cyclical assignments of columns, Dynamic partitioning using a centralized task queue. Since ray-tracing algorithm is recursive in nature, execution time per ray is unpredictable. Although static cyclical allocation performed well for simple scene, in most cases dynamic allocation of blocks would be the best option among all four partitioning schemes. This is evident from the results of all four partitioning schemes for complex scene.