Multiple Processor Systems

Project Assignment 2

Vignesh Govindarajulu Kothandapani

University ID: 831001854

November 23, 2014

Abstract:

Ray tracing is an image rendering technique in whoch all the pixels in the rendered image can be rendered independent of each other. It is to be noted that there may be millions of pixels in a rendered image. Each requiring its own ray tracing result. If the scenes captured are reflective or transparent, large number of rays are generated and traced, thus leading to large execution time. A sequential program thus would take large amount of execution time.

Here since each pixel is rendered independently, the program can be parallelized to run on a number of processors, thus leading to the faster execution times. The parallelization is done by using static partitioning that include strips, square blocks, cyclical assignment of rows and also by dynamic portioning using a centralized task queue. The various design methodology are described in the later sections.

Design Methodology:

Static partitioning using contiguous strips of columns:

In this the scene is divided into a number of vertical strips depending on the number of processors. Each strip is processed by a single processor. In certain cases the scene cannot be exactly divided among the processors, it results in the remainder portion. The remainder portion is processed by the master in these cases. Once the required portions of the image are processed by the slaves, the values obtained are passed to the master. The master places all the values in the respective positions in the array and thus the required image is obtained.

Static partitioning using square blocks:

In this the whole scene is divided into a number of square blocks equal to the number of processors, thus, each block is processed by a single processor. In certain cases remainder can occur on the top and right side of the scene, these extra pixels are processed by the processor handling the block right next to the remainder. This shows that remainder is also almost divided among processors. Once the required portions of the image are processed by the slaves, the values obtained are passed to the master. The master places all the values in the respective positions in the array and thus the required image is obtained.

Static partitioning using cyclical assignment of rows:

In this the whole scene is portioned into a number of strips assigned to each processor in a cyclic manner till the whole scene is rendered. In certain cases the scene cannot be exactly divided among the processors, the remaining strips are processed by the master in these cases. Once the required portions of the image are processed by the slaves, the values obtained are passed to the master. The master places all the values in the respective positions in the array and thus the required image is obtained.

Dynamic portioning using a centralized task queue:

The whole image is divided into blocks of the required size which is given as an input. In this all the computations is done by the slaves. The master performs the task of assigning blocks to the slave which is free at that time. As the blocks are dynamically assigned to the slaves, the efficiency of the program is increased. The remainder portion is also dynamically assigned to the slave that is free at that time. The master places all the values in the respective positions in the array and thus the required image is obtained.

Result and Analysis:

The observations for the rendering simple and complex images for different number of processors and processing types are discussed below.

1. Readings for Simple Image:

Sequential	
raytrace_seq	135.53 (s)

Table 1: Sequential runtime

Static Strip Allocation				
Number of Processes	Number of Strips	Execution Time (s)	Speedup	C-to-C ratio
1 (mpirun –np 1)	1	254.723	0.53	0
2 (mpirun –np 2)	2	154.625	0.87	0.00215638
4 (mpirun –np 4)	4	103.624	1.30	0.601402
9 (mpirun –np 9)	9	57.5527	2.35	0.840775
16 (mpirun –np 16)	16	41.1582	3.29	0.9514
20 (mpirun –np 20)	20	115.142	1.17	0.9219
25 (mpirun –np 25)	25	71.0336	1.90	0.8417278125
36 (mpirun –np 36)	36	87.212	1.55	0.8258732
49 (mpirun –np 49)	49	116.101	1.67	0.76937
55 (mpirun –np 55)	55	123.803	1.09	0.734935
64 (mpirun –np 64)	64	156.421	0.866	0.9098224

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

Static Block Allocation				
Number of Processes	Number of Blocks	Execution Time (s)	Speedup	C-to-C ratio
1 (mpirun –np 1)	1	143.435	.94	0
4 (mpirun –np 4)	4	66.0877	2.05	0.567534
9 (mpirun –np 9)	9	49.8755	2.717	0.902811
16 (mpirun –np 16)	16	34.8108	3.8933	0.932309
25 (mpirun –np 25)	25	21.0826	6.4285	0.96699
36 (mpirun –np 36)	36	20.1223	6.735	0.981315
49 (mpirun –np 49)	49	14.2238	9.52	0.986164
64 (mpirun –np 64)	64	13.5656	9.99	1.02069

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

Static Cyclical Allocation				
Number of Processes	Height of strip in pixels	Execution Time (s)	Speedup	C-to-C ratio
4 (mpirun –np 4)	1	38.3343	3.535	0.00338457
4 (mpirun –np 4)	5	38.0422	3.56	0.00347454
4 (mpirun –np 4)	10	38.0215	3.56	0.00336048
4 (mpirun –np 4)	20	38.7133	3.50	0.00305333
4 (mpirun –np 4)	80	41.1163	3.29	0.00348307
4 (mpirun –np 4)	320	62.4211	2.17	0.00167544
4 (mpirun –np 4)	640	101.384	1.33	0.000596341
4 (mpirun –np 4)	1280	E		E
9 (mpirun –np 9)	1	20.8761	6.49	0.0265346
9 (mpirun –np 9)	5	21.1067	6.42	0.0714828
9 (mpirun –np 9)	10	22.2201	6.099	0.00816373
9 (mpirun –np 9)	20	23.0451	5.881	0.00773106
9 (mpirun –np 9)	40	27.0316	5.01	0.00636308
9 (mpirun –np 9)	160	36.9311	3.66	0.00409244
9 (mpirun –np 9)	350	73.186	1.85	0.00195562
9 (mpirun –np 9)	650	E		E
16 (mpirun –np 16)	1	12.95	10.46	0.122828
16 (mpirun –np 16)	5	14.0667	9.63	0.0623454
16 (mpirun –np 16)	10	14.1221	9.59	0.061928
16 (mpirun –np 16)	20	17.8231	7.60	0.0463885
16 (mpirun –np 16)	50	17.4357	7.77	0.0473178
16 (mpirun –np 16)	100	18.1646	7.46	0.0459831
16 (mpirun –np 16)	250	31.1494	4.350	0.0214002
16 (mpirun –np 16)	400	E		E

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

Static Cyclical Allocation				
Number of Processes	Height of strip in pixels	Execution Time (s)	Speedup	C-to-C ratio
1 (mpirun –np 1)	27	139.338	0.97	0
2 (mpirun –np 2)	27	85.9453	1.5769	0.00746372
4 (mpirun –np 4)	27	45.4236	2.98	0.0120992
9 (mpirun –np 9)	27	23.4216	5.786	0.00910804
16 (mpirun –np 16)	27	18.8229	7.200	0.0433655
20 (mpirun –np 20)	27	14.399	9.41	0.0892749
25 (mpirun –np 25)	27	16.3549	8.28	0.102197
36 (mpirun –np 36)	27	10.9681	12	0.203879
49 (mpirun –np 49)	27	33.0865	4.09	0.0528191
55 (mpirun –np 55)	27	20.0842	6.74	0.106686
64 (mpirun –np 64)	27	55.2984	2.45	0.0306284

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

Note: As the dynamic code is not fully implemented, readings have not been taken.

- Readings for Complex Image:

Sequential	
raytrace_seq	5349.11 (s)

Table 6: Sequential runtime

Static Strip Allocation				
Number of Processes	Number of Strips	Execution Time (s)	Speedup	C-to-C ratio
1 (mpirun –np 1)	1	6780.32	0.788	0
2 (mpirun –np 2)	2	7233.96	0.739	0.0000231
4 (mpirun –np 4)	4	3509.4	1.524	0.499835
9 (mpirun –np 9)	9	2848.3	1.878	0.985162
16 (mpirun –np 16)	16	3287.36	1.627	0.999961
20 (mpirun –np 20)	20	2144.12	2.49	1.0111
25 (mpirun –np 25)	25	2025.78	2.6405	1.00885
36 (mpirun –np 36)	36	1422.43	3.76	1.01908
49 (mpirun –np 49)	49	1093.42	4.89	1.04301
55 (mpirun –np 55)	55	971.425	5.05	1.04954
64 (mpirun –np 64)	64	868.461	6.15	1.07139

Table 7: The effect of work unit size on computational efficiency using strip allocation

Static Block Allocation				
Number of Processes	Number of Blocks	Execution Time (s)	Speedup	C-to-C ratio
1 (mpirun –np 1)	1	5722.09	0.9348	0
4 (mpirun –np 4)	4	5405.38	0.9895	0.973169
9 (mpirun –np 9)	9	3474.23	1.5396	0.989479
16 (mpirun –np 16)	16	2865.55	1.8666	0.994115
25 (mpirun –np 25)	25	1913.61	2.7952	0.995469
36 (mpirun –np 36)	36	2999.82	1.7831	0.998262
49 (mpirun –np 49)	49	2695.46	1.9844	0. 998682
64 (mpirun –np 64)	64	1959.08	2.7304	0.998842

Table 8: The effect of work unit size on computational efficiency using block allocation

Static Cyclical Allocation				
Number of Processes	Height of strip in pixels	Execution Time (s)	Speedup	C-to-C ratio
4 (mpirun –np 4)	1	1518.47	3.5226	0.015519
4 (mpirun –np 4)	5	1506.8	3.5499	0.053157
4 (mpirun –np 4)	10	1595.01	3.3536	0.0724886
4 (mpirun –np 4)	20	1763.13	3.0338	0.194451
4 (mpirun –np 4)	80	1792.58	2.9840	0.000391284
4 (mpirun –np 4)	320	2631.76	2.0325	0.0855661
4 (mpirun –np 4)	640	5502.11	0.97	0.0000142893
4 (mpirun –np 4)	1280	E	E	E
9 (mpirun –np 9)	1	692.382	7.725	0.0682048
9 (mpirun –np 9)	5	698.459	7.658	0.0967171
9 (mpirun –np 9)	10	752.352	7.10	0.184962
9 (mpirun –np 9)	20	794.256	6.73	0.011087
9 (mpirun –np 9)	40	858.022	6.23	0.274525
9 (mpirun –np 9)	160	1608.52	3.32	0.000238782
9 (mpirun –np 9)	350	2877.33	1.85	0.121662
9 (mpirun –np 9)	650	E	E	E
16 (mpirun –np 16)	1	397.538	13.455	0.0758588
16 (mpirun –np 16)	5	452.823	11.812	0.142835
16 (mpirun –np 16)	10	455.278	11.74	0.00208433
16 (mpirun –np 16)	20	486.886	10.98	0.0603923
16 (mpirun –np 16)	50	742.284	7.206	2.59258
16 (mpirun –np 16)	100	1073.83	4.98	0.95349717
16 (mpirun –np 16)	250	1578.93	3.38	0.8486587
16 (mpirun –np 16)	400	E	E	E

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

Static Cyclical Allocation				
Number of Processes	Height of strip in pixels	Execution Time (s)	Speedup	C-to-C ratio
1 (mpirun –np 1)	27	5653.76	0.94611	0
2 (mpirun –np 2)	27	2919.28	1.83	0.0275092
4 (mpirun –np 4)	27	1542.49	3.46	0.186516
9 (mpirun –np 9)	27	863.229	6.196	0.113109
16 (mpirun –np 16)	27	819.489	6.52	0.116114
20 (mpirun –np 20)	27	526.47	10.16	0.841978
25 (mpirun –np 25)	27	518.068	10.32	0.74864
36 (mpirun –np 36)	27	430.989	12.41	0.103903
49 (mpirun –np 49)	27	1487.92	3.59	0.00109798
55 (mpirun –np 55)	27	319.161	16.7599	0.00594735
64 (mpirun –np 64)	27	2504.42	2.13586	0.000610403

Table 10. The effect of increasing the number of processors on a fixed allocation size

Note: As the dynamic code is not fully implemented, readings have not been taken.

Effect of Parallel implementation:

- Unbalanced load leads to lower speedup.
- Minimizing the number of sends and receives in-order to obtain higher speedup.
- Execution time is reduced.

Efficiency:

Simple Image

1) Static Strip Allocation:

   The speedup increases with the

2) Static Block Allocation:

   Speedup in this case increases with the increase in the number of processors. Overall the speedup is higher as compared to static strip allocation as all pixels as well as remainders are equally divided among the processors.

3) Static Cyclical Allocation:

   Effect of work unit size:

   It is to be noted that for the same number of processors the speedup decreases with the increase of strips. It is due to the fact that with increase in the number of strips the remainders might be high .since the remainder are processed by a single processor, the execution time becomes high.

   Effect of increasing processors for strip size:

   The speedup increases with the increase in the number of processors. The efficiency of parallel processing increases with increase in the number of processors for the same strip size.

Complex image:

1) Static Strip Allocation:

   Speedup in this case increases with the increase in the number of processors. The computations in case of complex image is higher as compared to simple image. However, the c-c ratio is on the higher side.

2) Static Block Allocation:

   In this case the speedup achieved is very less as compared to static strip allocation.

3) Static Cyclical Allocation:

   Effect of work unit size:

   This is similar to the results obtained with simple image. Here for the same number of processors the speedup decreases with the increase of strips. It is due to the fact that with increase in the

number of strips the remainders might be high .Since the remainder are processed by a single processor, the execution time becomes high.

Effect of increasing processors for strip size:

The speedup increases with the increase in the number of processors. The efficiency of parallel processing increases with increase in the number of processors for the same strip size.

From observing the above results it can be concluded that cyclic processing with large number of processors and less number of strips ideal for rendering images.

Charts:

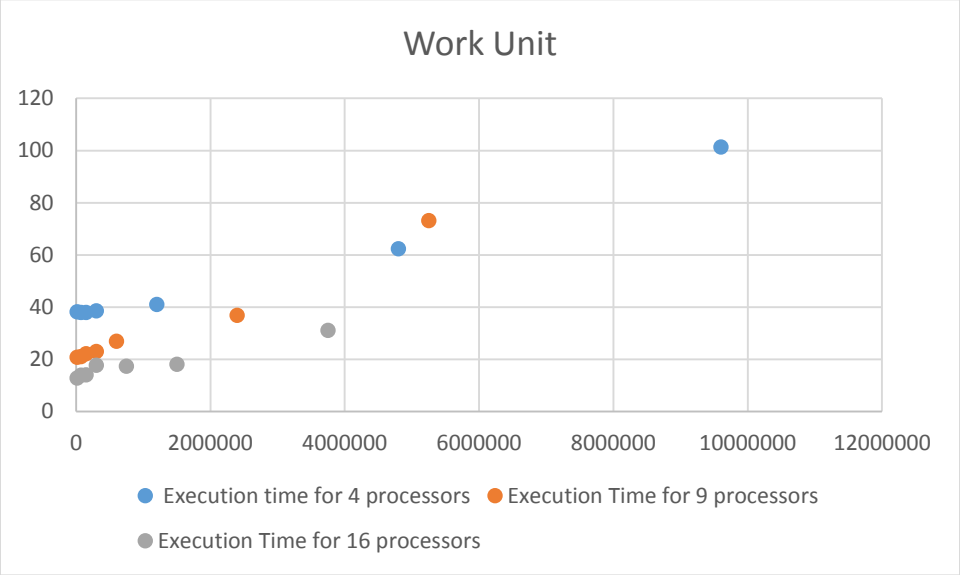

Fig 1: WorkUnit VS Execution time (Simple Image)

The execution time increases with the increase in work unit size. For the same work unit size execution time increases in no of processors decreases the execution time

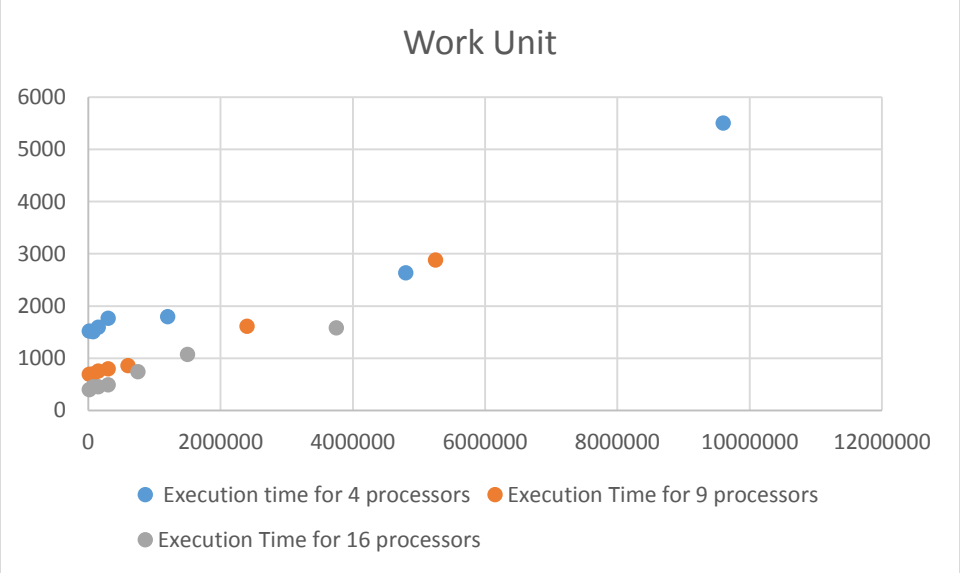

Fig 2:WorkUnit VS Execution time (Complex Image)

The execution time increases with increase in work unit size. For the lower unit size the increases in no of processors decreases the execution time but as unit size increases execution time decreases.

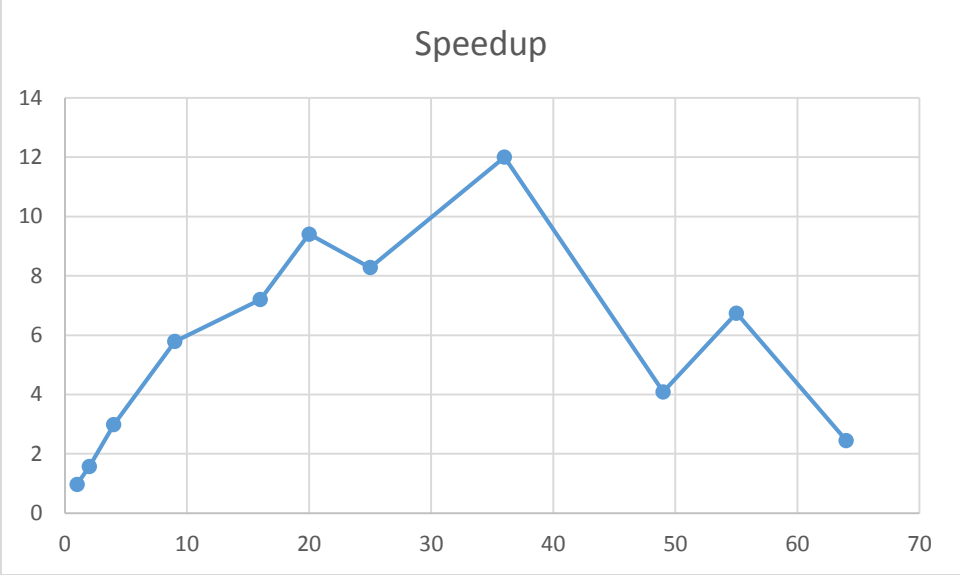

Fig 3: Speedup VS No of processors(Simple)

The speedup increases initially with the increase in no of processors but later decreases.

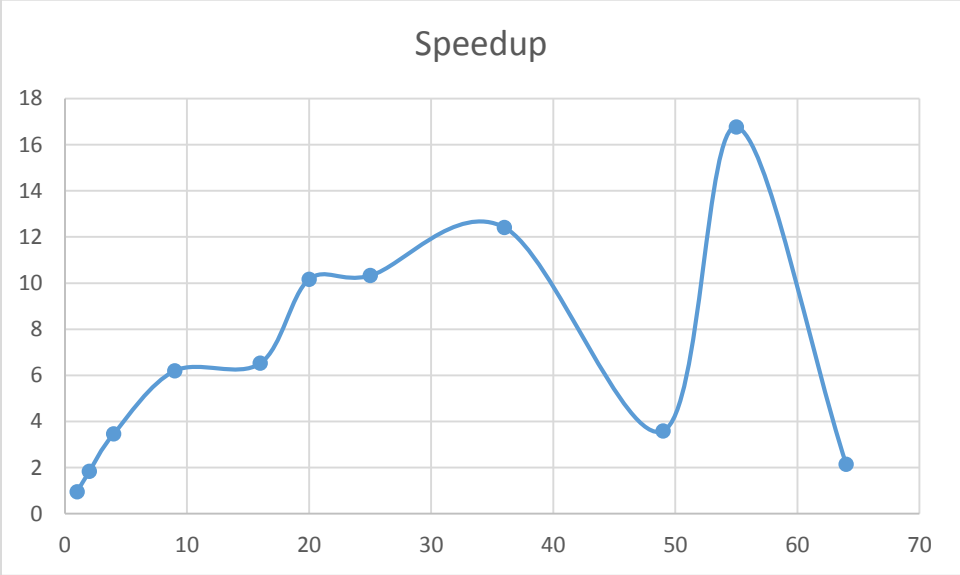

Fig 4: Speedup VS No of processors

The speedup increases initially with increase in no of processors.

Conclusion:

The cyclic implementation is found to be the most effective method for parallel processing. A good parallel code should have load balancing as well as good communication to computation ratio (less than 1).

Drawbacks:

-Remainder is processed by the master in static strip allocation and static cyclical allocation which leads to more execution time and lesser speedup

-Cyclic implementation does not work for conditions in which each processor doesn't have the pixels to process for the given number of strips.