Midterm Part III of III - Timing and Analysis

Adam Ross
adro4510@colorado.edu

Abstract

This report covers timing models for our Conway's game of life implementations which include estimated computation, estimated communication, estimated speed up, actual timing data, actual speed up and efficiency. MPI timing calls will be implemented in my code surrounding the communication and computation sections of the code. With the timing models multiple tables will be generated detailing square world sizes in multiples of 900x900 up to 12600x12600 for both communication methods and distribution methods. Then actual timing will be taken from Comet and Stampede systems for np=1, 9, 25 on world sizes 900,2700,5400,9000,12600. This data will be analyzed with respect to our models and prove or disprove their correctness.

1. Introduction

When writing code to be used on large scale systems where wall clock time is important it is useful to first generate timing models and then take basic timing to predict how your algorithm will perform at larger scale and to look at how well it scales over high process or data counts. In this way you may predict the total time required for a given job and decide if you need to revise.

I first generate theoretical timing models based on ta, the arithmetic time to process a byte, ts, the message overhead time, and tc, the messaging time to send an additional byte. With this model I then predict timing data for various world sizes, process counts, and distributions. Next I will develop timing code in my Conway's implementation and take timing data on Comet and Stampede. Speedup and efficiency will then be calculated for both sets of data, and graphs will be generated to highlight the scalability of my code.

2. Overview

Below is attached four charts of expected runtimes for both Comet and Stampede for the two distribution models and two communication types for asynchronous communication with and without overlap.

Distribution based performance ranges depending on the size and shape of the starting world. What we really care about is the computational overhead compared to the communication overhead. The checkerboard distribution is better at data distribution and higher arithmetic overhead whereas the row-block distribution is better at lower communication overhead data.

This comes from the number of communications done by each method. The checkerboard distribution has 4 sends and recvs per iteration whereas the row-block only has 2. This only matters in small enough world sizes where the communication time is comparable to the arithmetic time. *In general the checkerboard distribution should be better except at small world sizes.*

MT3 Expected Non-Overlapping Comet

Distribution	Communication Technique	NP	Input World Size	Expected Runtime
row	Synchronous	9	900	0.0015459562
row	Asynchronous	9	900	0.0015365862
row	Synchronous	25	900	0.0005668138
row	Asynchronous	25	900	0.0005574438
block	Synchronous	9	900	0.0015459562
block	Asynchronous	9	900	0.0015365862
block	Synchronous	25	900	0.0005668138
block	Asynchronous	25	900	0.0005574438
row	Synchronous	9	1800	0.0061423624
row	Asynchronous	9	1800	0.0061329924
row	Synchronous	25	1800	0.0022257928
row	Asynchronous	25	1800	0.0022164228
block	Synchronous	9	1800	0.0022257928
block	Asynchronous	9	1800	0.0061329924
block	Synchronous	25	1800	0.0061423624
block	Asynchronous	25	1800	0.0022164228
row	Synchronous	9	2700	0.0137985886
row	Asynchronous	9	2700	0.0137892186
row	Synchronous	25	2700	0.004986307
row	Asynchronous	25	2700	0.004976937
block	Synchronous	9	2700	0.004986307
block	Asynchronous	9	2700	0.0137892186
block	Synchronous	25	2700	0.0137985886
block	Asynchronous	25	2700	0.004976937
row	Synchronous	9	3600	0.0245146348
row	Asynchronous	9	3600	0.0245052648
row	Synchronous	25	3600	0.0088483564
row	Asynchronous	25	3600	0.0088389864
block	Synchronous	9	3600	0.0088483564
block	Asynchronous	9	3600	0.0245052648
block	Synchronous	25	3600	0.0245146348
block	Asynchronous	25	3600	0.0088389864
row	Synchronous	9	4500	0.038290501
row	Asynchronous	9	4500	0.038281131
row	Synchronous	25	4500	0.013811941
row	Asynchronous	25	4500	0.013802571
block	Synchronous	9	4500	0.013811941
block	Asynchronous	9	4500	0.038281131
block	Synchronous	25	4500	0.038290501
block	Asynchronous	25	4500	0.013802571
row	Synchronous	9	5400	0.0551261872
row	Asynchronous	9	5400	0.0551168172
row	Synchronous	25	5400	0.0198770608
row	Asynchronous	25	5400	0.0198676908
block	Synchronous	9	5400	0.0198770608
block	Asynchronous	9	5400	0.0551168172
block	Synchronous	25	5400	0.0551261872
block	Asynchronous	25	5400	0.0198676908
row	Synchronous	9	6300	0.0750216934
row	Asynchronous	9	6300	0.0750123234
row	Synchronous	25	6300	0.0270437158
row	Asynchronous	25	6300	0.0270343458
block	Synchronous	9	6300	0.0270437158
block	Asynchronous	9	6300	0.0750123234
block	Synchronous	25	6300	0.0750216934
block	Asynchronous	25	6300	0.0270343458
row	Synchronous	9	7200	0.0979770196
row	Asynchronous	9	7200	0.0979676496
row	Synchronous	25	7200	0.035311906
row	Asynchronous	25	7200	0.035302536

MT3 Expected Non-Overlapping Comet

block	Synchronous	9	7200	0.035311906
block	Asynchronous	9	7200	0.0979676496
block	Synchronous	25	7200	0.0979770196
block	Asynchronous	25	7200	0.035302536
row	Synchronous	9	8100	0.1239921658
row	Asynchronous	9	8100	0.1239827958
row	Synchronous	25	8100	0.0446816314
row	Asynchronous	25	8100	0.0446722614
block	Synchronous	9	8100	0.0446816314
block	Asynchronous	9	8100	0.1239827958
block	Synchronous	25	8100	0.1239921658
block	Asynchronous	25	8100	0.0446722614
row	Synchronous	9	9000	0.153067132
row	Asynchronous	9	9000	0.153057762
row	Synchronous	25	9000	0.055152892
row	Asynchronous	25	9000	0.055143522
block	Synchronous	9	9000	0.055152892
block	Asynchronous	9	9000	0.153057762
block	Synchronous	25	9000	0.153067132
block	Asynchronous	25	9000	0.055143522
row	Synchronous	9	9900	0.1852019182
row	Asynchronous	9	9900	0.1851925482
row	Synchronous	25	9900	0.0667256878
row	Asynchronous	25	9900	0.0667163178
block	Synchronous	9	9900	0.0667256878
block	Asynchronous	9	9900	0.1851925482
block	Synchronous	25	9900	0.1852019182
block	Asynchronous	25	9900	0.0667163178
row	Synchronous	9	10800	0.2203965244
row	Asynchronous	9	10800	0.2203871544
row	Synchronous	25	10800	0.0794000188
row	Asynchronous	25	10800	0.0793906488
block	Synchronous	9	10800	0.0794000188
block	Asynchronous	9	10800	0.2203871544
block	Synchronous	25	10800	0.2203965244
block	Asynchronous	25	10800	0.0793906488
row	Synchronous	9	11700	0.2586509506
row	Asynchronous	9	11700	0.2586415806
row	Synchronous	25	11700	0.093175885
row	Asynchronous	25	11700	0.093166515
block	Synchronous	9	11700	0.093175885
block	Asynchronous	9	11700	0.2586415806
block	Synchronous	25	11700	0.2586509506
block	Asynchronous	25	11700	0.093166515
row	Synchronous	9	12600	0.2999651968
row	Asynchronous	9	12600	0.2999558268
row	Synchronous	25	12600	0.1080532864
row	Asynchronous	25	12600	0.1080439164
block	Synchronous	9	12600	0.1080532864
block	Asynchronous	9	12600	0.2999558268
block	Synchronous	25	12600	0.2999651968
block	Asynchronous	25	12600	0.1080439164
ta	0.000000016999			
ts	0.000004685			
tc	0.000000003709			

MT3 Expected Non-Overlapping Stampede

Distribution	Communication Technique	NP	Input World Size	Expected Runtime
row	Synchronous	9	900	0.0003850462
row	Asynchronous	9	900	0.0003756762
row	Synchronous	25	900	0.0001488862
row	Asynchronous	25	900	0.0001395162
block	Synchronous	9	900	0.0003850462
block	Asynchronous	9	900	0.0003756762
block	Synchronous	25	900	0.0001488862
block	Asynchronous	25	900	0.0001395162
row	Synchronous	9	1800	0.0014987224
row	Asynchronous	9	1800	0.0014893524
row	Synchronous	25	1800	0.0005540824
row	Asynchronous	25	1800	0.0005447124
block	Synchronous	9	1800	0.0005540824
block	Asynchronous	9	1800	0.0014893524
block	Synchronous	25	1800	0.0014987224
block	Asynchronous	25	1800	0.0005447124
row	Synchronous	9	2700	0.0033503986
row	Asynchronous	9	2700	0.0033410286
row	Synchronous	25	2700	0.0012249586
row	Asynchronous	25	2700	0.0012155886
block	Synchronous	9	2700	0.0012249586
block	Asynchronous	9	2700	0.0033410286
block	Synchronous	25	2700	0.0033503986
block	Asynchronous	25	2700	0.0012155886
row	Synchronous	9	3600	0.0059400748
row	Asynchronous	9	3600	0.0059307048
row	Synchronous	25	3600	0.0021615148
row	Asynchronous	25	3600	0.0021521448
block	Synchronous	9	3600	0.0021615148
block	Asynchronous	9	3600	0.0059307048
block	Synchronous	25	3600	0.0059400748
block	Asynchronous	25	3600	0.0021521448
row	Synchronous	9	4500	0.009267751
row	Asynchronous	9	4500	0.009258381
row	Synchronous	25	4500	0.003363751
row	Asynchronous	25	4500	0.003354381
block	Synchronous	9	4500	0.003363751
block	Asynchronous	9	4500	0.009258381
block	Synchronous	25	4500	0.009267751
block	Asynchronous	25	4500	0.003354381
row	Synchronous	9	5400	0.0133334272
row	Asynchronous	9	5400	0.0133240572
row	Synchronous	25	5400	0.0048316672
row	Asynchronous	25	5400	0.0048222972
block	Synchronous	9	5400	0.0048316672
block	Asynchronous	9	5400	0.0133240572
block	Synchronous	25	5400	0.0133334272
block	Asynchronous	25	5400	0.0048222972
row	Synchronous	9	6300	0.0181371034
row	Asynchronous	9	6300	0.0181277334
row	Synchronous	25	6300	0.0065652634
row	Asynchronous	25	6300	0.0065558934
block	Synchronous	9	6300	0.0065652634
block	Asynchronous	9	6300	0.0181277334
block	Synchronous	25	6300	0.0181371034
block	Asynchronous	25	6300	0.0065558934
row	Synchronous	9	7200	0.0236787796
row	Asynchronous	9	7200	0.0236694096
row	Synchronous	25	7200	0.0085645396
row	Asynchronous	25	7200	0.0085551696

MT3 Expected Non-Overlapping Stampede

block	Synchronous	9	7200	0.0085645396
block	Asynchronous	9	7200	0.0236694096
block	Synchronous	25	7200	0.0236787796
block	Asynchronous	25	7200	0.0085551696
row	Synchronous	9	8100	0.0299584558
row	Asynchronous	9	8100	0.0299490858
row	Synchronous	25	8100	0.0108294958
row	Asynchronous	25	8100	0.0108201258
block	Synchronous	9	8100	0.0108294958
block	Asynchronous	9	8100	0.0299490858
block	Synchronous	25	8100	0.0299584558
block	Asynchronous	25	8100	0.0108201258
row	Synchronous	9	9000	0.036976132
row	Asynchronous	9	9000	0.036966762
row	Synchronous	25	9000	0.013360132
row	Asynchronous	25	9000	0.013350762
block	Synchronous	9	9000	0.013360132
block	Asynchronous	9	9000	0.036966762
block	Synchronous	25	9000	0.036976132
block	Asynchronous	25	9000	0.013350762
row	Synchronous	9	9900	0.0447318082
row	Asynchronous	9	9900	0.0447224382
row	Synchronous	25	9900	0.0161564482
row	Asynchronous	25	9900	0.0161470782
block	Synchronous	9	9900	0.0161564482
block	Asynchronous	9	9900	0.0447224382
block	Synchronous	25	9900	0.0447318082
block	Asynchronous	25	9900	0.0161470782
row	Synchronous	9	10800	0.0532254844
row	Asynchronous	9	10800	0.0532161144
row	Synchronous	25	10800	0.0192184444
row	Asynchronous	25	10800	0.0192090744
block	Synchronous	9	10800	0.0192184444
block	Asynchronous	9	10800	0.0532161144
block	Synchronous	25	10800	0.0532254844
block	Asynchronous	25	10800	0.0192090744
row	Synchronous	9	11700	0.0624571606
row	Asynchronous	9	11700	0.0624477906
row	Synchronous	25	11700	0.0225461206
row	Asynchronous	25	11700	0.0225367506
block	Synchronous	9	11700	0.0225461206
block	Asynchronous	9	11700	0.0624477906
block	Synchronous	25	11700	0.0624571606
block	Asynchronous	25	11700	0.0225367506
row	Synchronous	9	12600	0.0724268368
row	Asynchronous	9	12600	0.0724174668
row	Synchronous	25	12600	0.0261394768
row	Asynchronous	25	12600	0.0261301068
block	Synchronous	9	12600	0.0261394768
block	Asynchronous	9	12600	0.0724174668
block	Synchronous	25	12600	0.0724268368
block	Asynchronous	25	12600	0.0261301068
ta	0.0000000041			
ts	0.000004685			
tc	0.000000003709			

MT3 Expected Overlapping Comet

Distribution	Communication Technique	NP	Input World Size	Expected Runtime
row	Synchronous	9	900	0.0015459562
row	Asynchronous	9	900	0.00152991
row	Synchronous	25	900	0.0005668138
row	Asynchronous	25	900	0.0005507676
block	Synchronous	9	900	0.0015459562
block	Asynchronous	9	900	0.00152991
block	Synchronous	25	900	0.0005668138
block	Asynchronous	25	900	0.0005507676
row	Synchronous	9	1800	0.0061423624
row	Asynchronous	9	1800	0.00611964
row	Synchronous	25	1800	0.0022257928
row	Asynchronous	25	1800	0.0022030704
block	Synchronous	9	1800	0.0061423624
block	Asynchronous	9	1800	0.00611964
block	Synchronous	25	1800	0.0022257928
block	Asynchronous	25	1800	0.0022030704
row	Synchronous	9	2700	0.0137985886
row	Asynchronous	9	2700	0.01376919
row	Synchronous	25	2700	0.004986307
row	Asynchronous	25	2700	0.0049569084
block	Synchronous	9	2700	0.0137985886
block	Asynchronous	9	2700	0.01376919
block	Synchronous	25	2700	0.004986307
block	Asynchronous	25	2700	0.0049569084
row	Synchronous	9	3600	0.0245146348
row	Asynchronous	9	3600	0.02447856
row	Synchronous	25	3600	0.0088483564
row	Asynchronous	25	3600	0.0088122816
block	Synchronous	9	3600	0.0245146348
block	Asynchronous	9	3600	0.02447856
block	Synchronous	25	3600	0.0088483564
block	Asynchronous	25	3600	0.0088122816
row	Synchronous	9	4500	0.038290501
row	Asynchronous	9	4500	0.03824775
row	Synchronous	25	4500	0.013811941
row	Asynchronous	25	4500	0.01376919
block	Synchronous	9	4500	0.038290501
block	Asynchronous	9	4500	0.03824775
block	Synchronous	25	4500	0.013811941
block	Asynchronous	25	4500	0.01376919
row	Synchronous	9	5400	0.0551261872
row	Asynchronous	9	5400	0.05507676
row	Synchronous	25	5400	0.0198770608
row	Asynchronous	25	5400	0.0198276336
block	Synchronous	9	5400	0.0551261872
block	Asynchronous	9	5400	0.05507676
block	Synchronous	25	5400	0.0198770608
block	Asynchronous	25	5400	0.0198276336
row	Synchronous	9	6300	0.0750216934
row	Asynchronous	9	6300	0.07496559
row	Synchronous	25	6300	0.0270437158
row	Asynchronous	25	6300	0.0269876124
block	Synchronous	9	6300	0.0750216934
block	Asynchronous	9	6300	0.07496559
block	Synchronous	25	6300	0.0270437158
block	Asynchronous	25	6300	0.0269876124
row	Synchronous	9	7200	0.0979770196
row	Asynchronous	9	7200	0.09791424
row	Synchronous	25	7200	0.035311906
row	Asynchronous	25	7200	0.0352491264

MT3 Expected Overlapping Comet

block	Synchronous	9	7200	0.0979770196
block	Asynchronous	9	7200	0.09791424
block	Synchronous	25	7200	0.035311906
block	Asynchronous	25	7200	0.0352491264
row	Synchronous	9	8100	0.1239921658
row	Asynchronous	9	8100	0.12392271
row	Synchronous	25	8100	0.0446816314
row	Asynchronous	25	8100	0.0446121756
block	Synchronous	9	8100	0.1239921658
block	Asynchronous	9	8100	0.12392271
block	Synchronous	25	8100	0.0446816314
block	Asynchronous	25	8100	0.0446121756
row	Synchronous	9	9000	0.153067132
row	Asynchronous	9	9000	0.152991
row	Synchronous	25	9000	0.055152892
row	Asynchronous	25	9000	0.05507676
block	Synchronous	9	9000	0.153067132
block	Asynchronous	9	9000	0.152991
block	Synchronous	25	9000	0.055152892
block	Asynchronous	25	9000	0.05507676
row	Synchronous	9	9900	0.1852019182
row	Asynchronous	9	9900	0.18511911
row	Synchronous	25	9900	0.0667256878
row	Asynchronous	25	9900	0.0666428796
block	Synchronous	9	9900	0.1852019182
block	Asynchronous	9	9900	0.18511911
block	Synchronous	25	9900	0.0667256878
block	Asynchronous	25	9900	0.0666428796
row	Synchronous	9	10800	0.2203965244
row	Asynchronous	9	10800	0.22030704
row	Synchronous	25	10800	0.0794000188
row	Asynchronous	25	10800	0.0793105344
block	Synchronous	9	10800	0.2203965244
block	Asynchronous	9	10800	0.22030704
block	Synchronous	25	10800	0.0794000188
block	Asynchronous	25	10800	0.0793105344
row	Synchronous	9	11700	0.2586509506
row	Asynchronous	9	11700	0.25855479
row	Synchronous	25	11700	0.093175885
row	Asynchronous	25	11700	0.0930797244
block	Synchronous	9	11700	0.2586509506
block	Asynchronous	9	11700	0.25855479
block	Synchronous	25	11700	0.093175885
block	Asynchronous	25	11700	0.0930797244
row	Synchronous	9	12600	0.2999651968
row	Asynchronous	9	12600	0.29986236
row	Synchronous	25	12600	0.1080532864
row	Asynchronous	25	12600	0.1079504496
block	Synchronous	9	12600	0.2999651968
block	Asynchronous	9	12600	0.29986236
block	Synchronous	25	12600	0.1080532864
block	Asynchronous	25	12600	0.1079504496
ta	0.000000016999			
ts	0.000004685			
tc	0.000000003709			

MT3 Expected Overlapping Stampede

Distribution	Communication Technique	NP	Input World Size	Expected Runtime
row	Synchronous	9	900	0.0003850462
row	Asynchronous	9	900	0.000369
row	Synchronous	25	900	0.0001488862
row	Asynchronous	25	900	0.00013284
block	Synchronous	9	900	0.0003850462
block	Asynchronous	9	900	0.000369
block	Synchronous	25	900	0.0001488862
block	Asynchronous	25	900	0.00013284
row	Synchronous	9	1800	0.0014987224
row	Asynchronous	9	1800	0.001476
row	Synchronous	25	1800	0.0005540824
row	Asynchronous	25	1800	0.00053136
block	Synchronous	9	1800	0.0014987224
block	Asynchronous	9	1800	0.001476
block	Synchronous	25	1800	0.0005540824
block	Asynchronous	25	1800	0.00053136
row	Synchronous	9	2700	0.0033503986
row	Asynchronous	9	2700	0.003321
row	Synchronous	25	2700	0.0012249586
row	Asynchronous	25	2700	0.00119556
block	Synchronous	9	2700	0.0033503986
block	Asynchronous	9	2700	0.003321
block	Synchronous	25	2700	0.0012249586
block	Asynchronous	25	2700	0.00119556
row	Synchronous	9	3600	0.0059400748
row	Asynchronous	9	3600	0.005904
row	Synchronous	25	3600	0.0021615148
row	Asynchronous	25	3600	0.00212544
block	Synchronous	9	3600	0.0059400748
block	Asynchronous	9	3600	0.005904
block	Synchronous	25	3600	0.0021615148
block	Asynchronous	25	3600	0.00212544
row	Synchronous	9	4500	0.009267751
row	Asynchronous	9	4500	0.009225
row	Synchronous	25	4500	0.003363751
row	Asynchronous	25	4500	0.003321
block	Synchronous	9	4500	0.009267751
block	Asynchronous	9	4500	0.009225
block	Synchronous	25	4500	0.003363751
block	Asynchronous	25	4500	0.003321
row	Synchronous	9	5400	0.0133334272
row	Asynchronous	9	5400	0.013284
row	Synchronous	25	5400	0.0048316672
row	Asynchronous	25	5400	0.00478224
block	Synchronous	9	5400	0.0133334272
block	Asynchronous	9	5400	0.013284
block	Synchronous	25	5400	0.0048316672
block	Asynchronous	25	5400	0.00478224
row	Synchronous	9	6300	0.0181371034
row	Asynchronous	9	6300	0.018081
row	Synchronous	25	6300	0.0065652634
row	Asynchronous	25	6300	0.00650916
block	Synchronous	9	6300	0.0181371034
block	Asynchronous	9	6300	0.018081
block	Synchronous	25	6300	0.0065652634
block	Asynchronous	25	6300	0.00650916
row	Synchronous	9	7200	0.0236787796
row	Asynchronous	9	7200	0.023616
row	Synchronous	25	7200	0.0085645396
row	Asynchronous	25	7200	0.00850176

MT3 Expected Overlapping Stampede

block	Synchronous	9	7200	0.0236787796
block	Asynchronous	9	7200	0.023616
block	Synchronous	25	7200	0.0085645396
block	Asynchronous	25	7200	0.00850176
row	Synchronous	9	8100	0.0299584558
row	Asynchronous	9	8100	0.029889
row	Synchronous	25	8100	0.0108294958
row	Asynchronous	25	8100	0.01076004
block	Synchronous	9	8100	0.0299584558
block	Asynchronous	9	8100	0.029889
block	Synchronous	25	8100	0.0108294958
block	Asynchronous	25	8100	0.01076004
row	Synchronous	9	9000	0.036976132
row	Asynchronous	9	9000	0.0369
row	Synchronous	25	9000	0.013360132
row	Asynchronous	25	9000	0.013284
block	Synchronous	9	9000	0.036976132
block	Asynchronous	9	9000	0.0369
block	Synchronous	25	9000	0.013360132
block	Asynchronous	25	9000	0.013284
row	Synchronous	9	9900	0.0447318082
row	Asynchronous	9	9900	0.044649
row	Synchronous	25	9900	0.0161564482
row	Asynchronous	25	9900	0.01607364
block	Synchronous	9	9900	0.0447318082
block	Asynchronous	9	9900	0.044649
block	Synchronous	25	9900	0.0161564482
block	Asynchronous	25	9900	0.01607364
row	Synchronous	9	10800	0.0532254844
row	Asynchronous	9	10800	0.053136
row	Synchronous	25	10800	0.0192184444
row	Asynchronous	25	10800	0.01912896
block	Synchronous	9	10800	0.0532254844
block	Asynchronous	9	10800	0.053136
block	Synchronous	25	10800	0.0192184444
block	Asynchronous	25	10800	0.01912896
row	Synchronous	9	11700	0.0624571606
row	Asynchronous	9	11700	0.062361
row	Synchronous	25	11700	0.0225461206
row	Asynchronous	25	11700	0.02244996
block	Synchronous	9	11700	0.0624571606
block	Asynchronous	9	11700	0.062361
block	Synchronous	25	11700	0.0225461206
block	Asynchronous	25	11700	0.02244996
row	Synchronous	9	12600	0.0724268368
row	Asynchronous	9	12600	0.072324
row	Synchronous	25	12600	0.0261394768
row	Asynchronous	25	12600	0.02603664
block	Synchronous	9	12600	0.0724268368
block	Asynchronous	9	12600	0.072324
block	Synchronous	25	12600	0.0261394768
block	Asynchronous	25	12600	0.02603664
ta	0.0000000041			
ts	0.000004685			
tc	0.000000003709			

Below are the models for each piece of my data:

Serial:

$$T(n, p) = n^2 * ta$$

Row:

$$T(n, p) = (n * n/p) * ta + 2(ts + tc(n))$$

Grid:

$$T(n, p) = (n/sqrt(p))^2 * ta + 2(ts+ tc(n/sqrt(p) - 2)) + 2(ts+ tc(n/sqrt(p)))$$

Async Row:

$$T(n, p) = (n * n/p) * ta + 2(tc(n))$$

Async Grid:

$$T(n, p) = (n/sqrt(p))^2 * ta + 2(tc(n/sqrt(p) - 2)) + 2(tc(n/sqrt(p)))$$

Async Overlapped Row:

Given that the data we operate on can grow n^2 compared to the rows/columns we are sending but also that ta is much less than ts we have to account for each case.

If total ta > total ts & tc:

$$T(n, p) = (n * n/p) * ta + 2ts$$

Else:

$$T(n, p) = 2(ts + tc(n))$$

Async Overlapped Grid:

If total ta > total ts & tc:

$$T(n, p) = (n/sqrt(p))^2 * ta$$

Else:

$$T(n, p) = 2(ts+ tc(n/sqrt(p) - 2)) + 2(ts+ tc(n/sqrt(p)))$$

In all of the measurements I have taken across Comet and Stampede the arithmetic time has greatly outweighed the communication time, therefore the if of each if/else statement was

used when calculating my data. MPI Isend and Irecv call return instantly, as per spec, and should not add any communication overhead whereas the MPI Wait() call will bring back all or some of the communication overhead if our arithmetic time is less than the total communication time. Otherwise Wait() should return quickly given our data transferred when we were doing calculations.

Attached below is my Comet and Stampede expected and actual data charts.

The predicted performance is fairly accurate in most cases. It is usually within 10% of the actual value except the row non-overlapped asynchronous calls. I would need more time to go debug the issues here.

Below are my graphs generated from my Stampede Data. The Comet data could just as easily be generated.

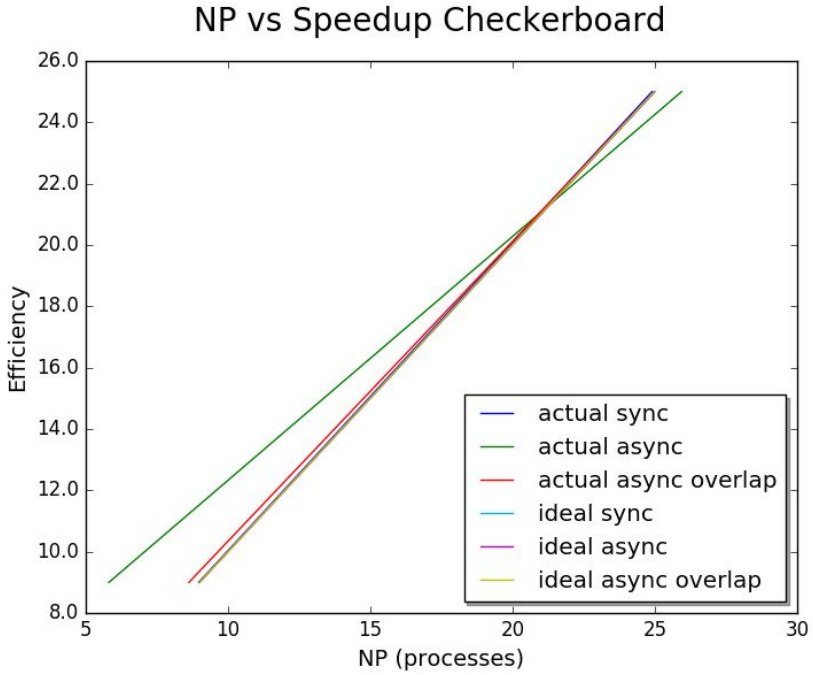

Figure 2.7 - NP vs Speedup CheckerBoard

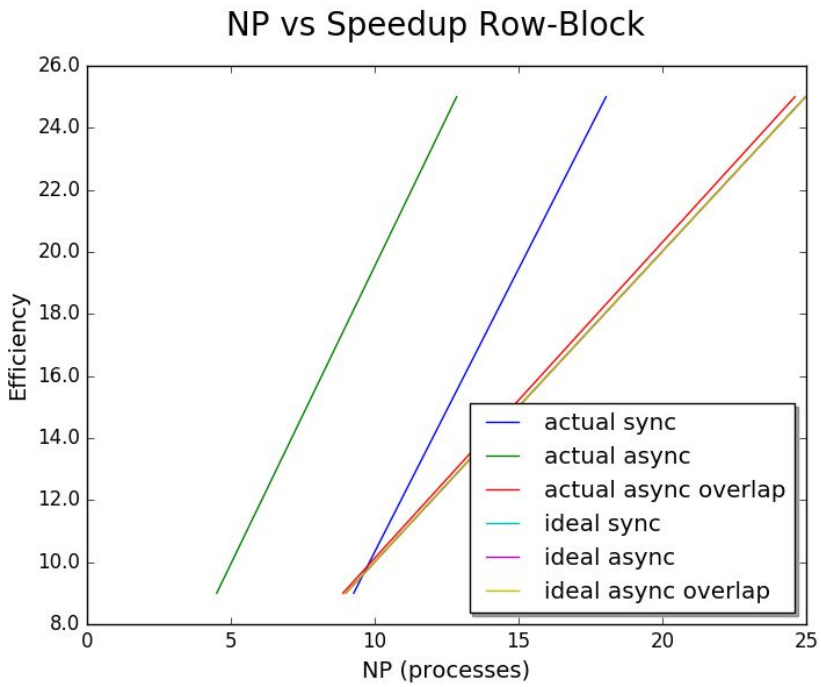

Figure 2.8 - NP vs Speedup Row-Block

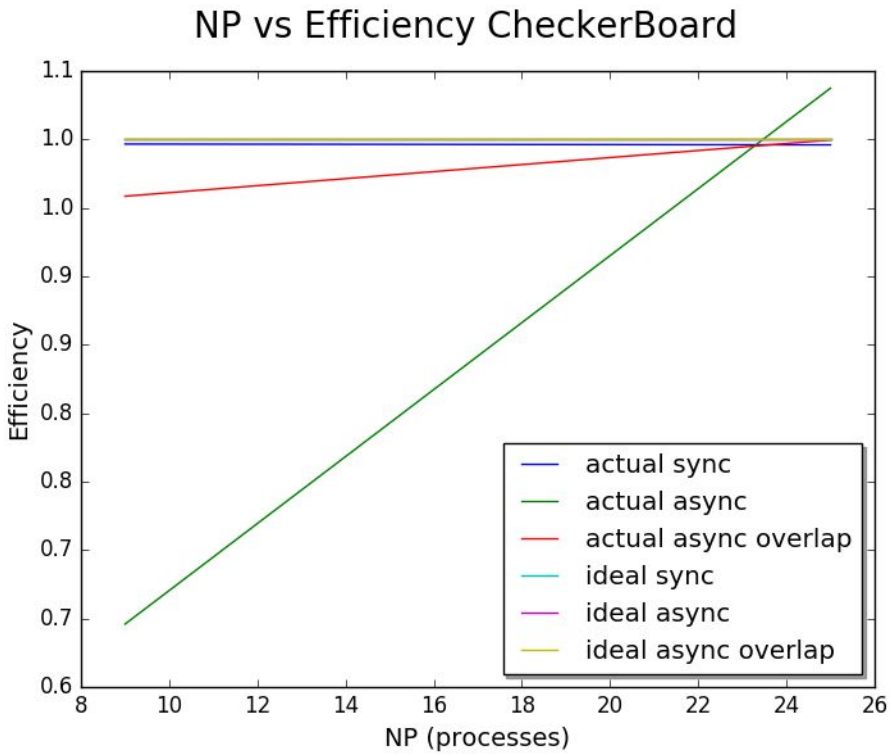

Figure 2.9 - NP vs Efficiency CheckerBoard

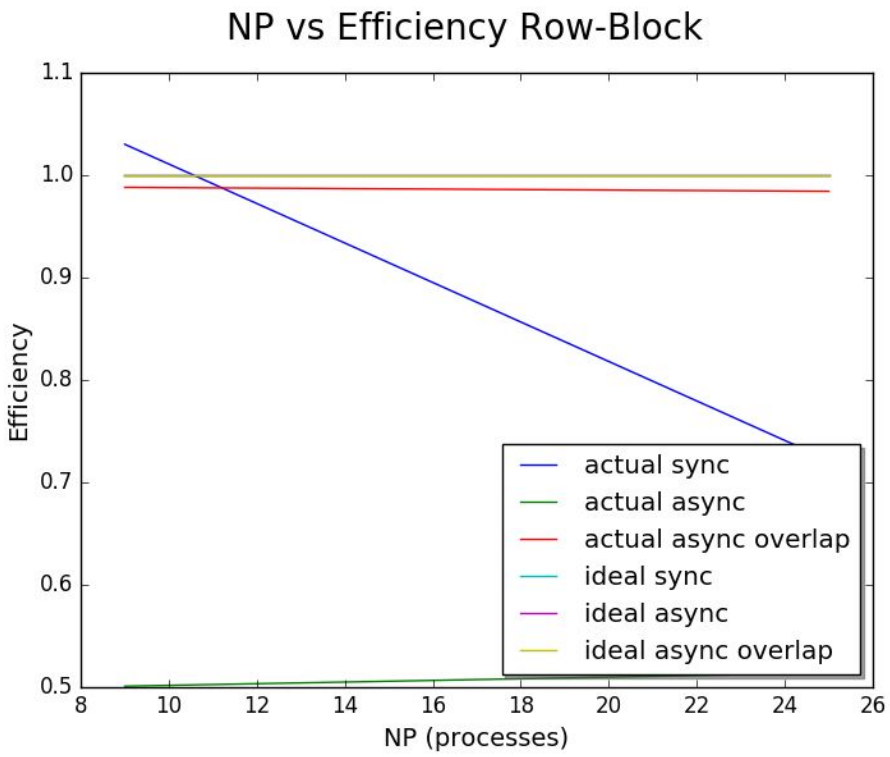

Figure 2.10 - NP vs Efficiency Row-Block

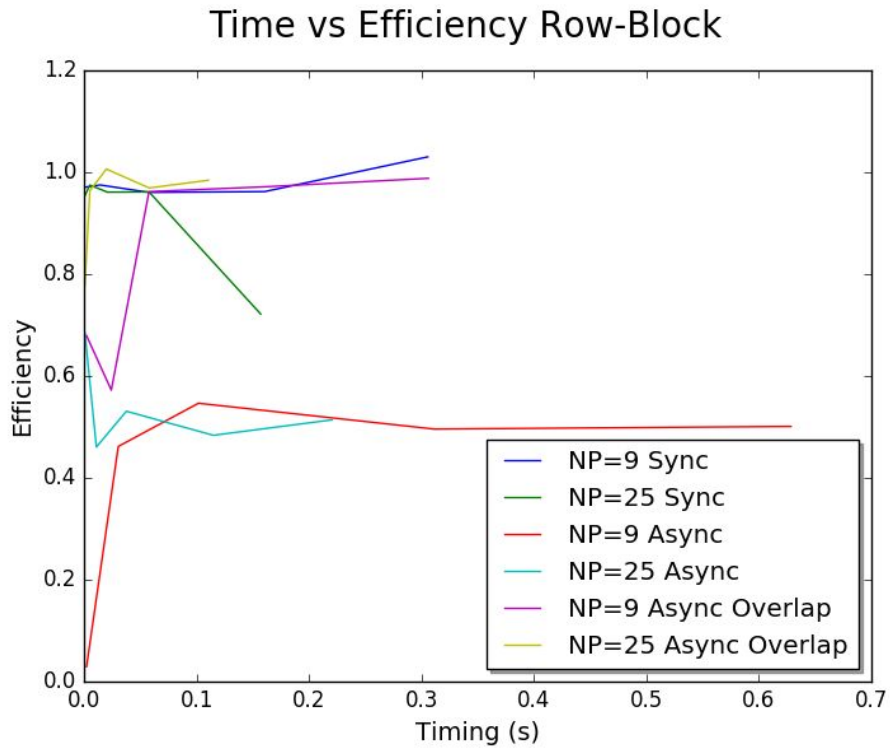

Figure 2.11 - Time vs Efficiency Checkerboard

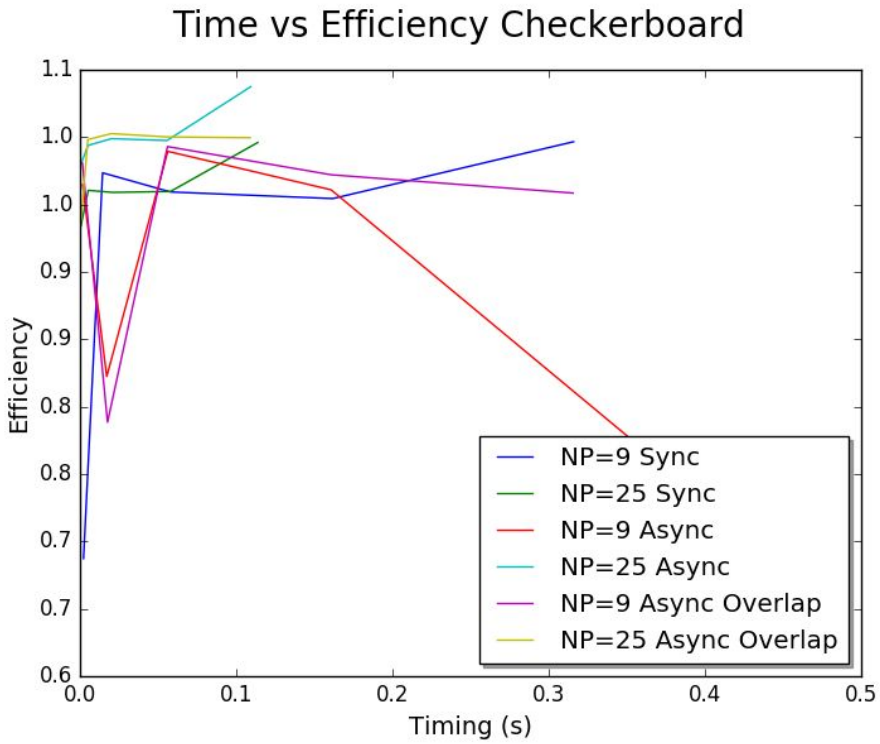

Figure 2.12 - Time vs Efficiency Row-Block

3. Conclusion/Learned

Timing is useful and important. Modeling is useful and important. I should have written this earlier.

Distribution	Communication Type	NP	Input World Size	Estimated Computation Time	Estimation Communication Time	Estimated Speedup	Actual Time	Actual Speedup	Efficiency
Serial	none	1	900	0.003321	0	-	0.003321812556	-	-
Row	sync	9	900	0.000369	0.0000160462	8.624939033	0.001032438364	3.217443938	0.3574937709
Row	sync	25	900	0.00013284	0.0000160462	22.30562671	0.0003916338996	8.481933151	0.3392773261
Block	sync	9	900	0.000369	0.000023175964	8.468137532	0.0004439029369	7.483195717	0.8314661908
Block	sync	25	900	0.00013284	0.000021395644	21.53198777	0.0004166506671	7.972656277	0.3189062511
Row	async	9	900	0.000369	0.0000066762	8.840059605	0.0007830303353	0.5321002882	0.05912225425
Row	async	25	900	0.00013284	0.0000066762	23.80368731	0.0002682395168	12.38375537	0.4953502148
Block	async	9	900	0.000369	0.000004435964	8.893090972	0.0007313724991	4.541888791	0.5046543102
Block	async	25	900	0.00013284	0.000002655644	24.51001303	0.0002236246943	14.85440848	0.5941763393
Serial	none	1	2700	0.029889	0	-	0.03343059543	-	-
Row	sync	9	2700	0.003321	0.0000293986	8.921028083	0.009229498701	3.622146393	0.4024607103
Row	sync	25	2700	0.00119556	0.0000293986	24.40000829	0.001313595301	25.4496917	1.017987668
Block	sync	9	2700	0.003321	0.000032077564	8.913900567	0.003810601855	8.77304864	0.9747831823
Block	sync	25	2700	0.00119556	0.000026736604	24.4531482	0.0013419446	24.91205331	0.9964821324
Row	async	9	2700	0.003321	0.0000200286	8.946047334	0.007272724089	4.596708883	0.5107454315
Row	async	25	2700	0.00119556	0.0000200286	24.58808844	0.002499537065	13.37471482	0.5349885927
Block	async	9	2700	0.003321	0.000013337564	8.963999423	0.006938037613	4.818451166	0.5353834629
Block	async	25	2700	0.00119556	0.000007996604	24.83389639	0.001884309348	17.74156428	0.7096625711
Serial	none	1	5400	0.119556	0	-	0.1357311835	-	-
Row	sync	9	5400	0.013284	0.0000494272	8.966636875	0.04136996374	3.280911347	0.3645457052
Row	sync	25	5400	0.00478224	0.0000494272	24.74425391	0.005155008159	26.32996481	1.053198593
Block	sync	9	5400	0.013284	0.000045429964	8.969325794	0.01711879656	7.928780683	0.8809756314
Block	sync	25	5400	0.00478224	0.000034748044	24.81965886	0.01343210379	10.10498322	0.404199329
Row	async	9	5400	0.013284	0.0000400572	8.972942566	0.03290318296	4.12516879	0.4583520878
Row	async	25	5400	0.00478224	0.0000400572	24.79233341	0.009782681245	13.8746403	0.554985612
Block	async	9	5400	0.013284	0.000026689964	8.981953627	0.03202808644	4.237879891	0.4708755434
Block	async	25	5400	0.00478224	0.000016008044	24.91659433	0.007208812213	18.8285087	0.7531403479
Serial	none	1	9000	0.3321	0	-	1.029113724	-	-
Row	sync	9	9000	0.0369	0.000076132	8.981469452	0.04731079988	21.75219457	2.416910507
Row	sync	25	9000	0.013284	0.000076132	24.85753883	0.04084522623	25.19544679	1.007817872
Block	sync	9	9000	0.0369	0.000063233164	8.984603661	0.04739566554	21.71324555	2.412582839
Block	sync	25	9000	0.013284	0.000045429964	24.91479387	0.04123370271	24.95807207	0.998322883
Row	async	9	9000	0.0369	0.000066762	8.983745993	0.09197389751	11.18919337	1.243243708
Row	async	25	9000	0.013284	0.000066762	24.87498466	0.03082291703	33.38794063	1.335517625
Block	async	9	9000	0.0369	0.000044493164	8.989161078	0.08963759382	11.48082719	1.275647465
Block	async	25	9000	0.013284	0.000026689964	24.94987119	0.0245529703	41.91402146	1.676560858
Serial	none	1	12600	0.650916	0	-	2.018465143	-	-
Row	sync	9	12600	0.072324	0.0001028368	8.987221157	0.225628498	8.945967204	0.993996356
Row	sync	25	12600	0.02603664	0.0001028368	24.90164608	0.08058301103	25.04827156	1.001930862
Block	sync	9	12600	0.072324	0.000081036364	8.98992712	0.0934044956	21.60993569	2.401103966
Block	sync	25	12600	0.02603664	0.000056111884	24.94623805	0.08086519815	24.96086313	0.9984345253
Row	async	9	12600	0.072324	0.0000934668	8.988384001	0.1811638724	11.14165378	1.237961531
Row	async	25	12600	0.02603664	0.0000934668	24.91057557	0.0603156624	33.46502489	1.338600995

Block	async	9	12600	0.072324	0.000062296364	8.992254511	0.4329709233	4.661895371	0.5179883746
Block	async	25	12600	0.02603664	0.000037371884	24.9641675	0.04772840375	42.29064843	1.691625937
Serial	none	1	900	0.003321	0	-	0.003321812556	-	-
Row	async-overlapped	9	900	0.000369	0.0000066762	8.840059605	0.001014330962	3.274880369	0.3638755966
Row	async-overlapped	25	900	0.00013284	0.0000066762	23.80368731	0.0007004451465	4.742430685	0.1896972274
Block	async-overlapped	9	900	0.000369	0.0000066762	8.840059605	0.0007373210307	4.505245907	0.5005828786
Block	async-overlapped	25	900	0.00013284	0.0000066762	23.80368731	0.0001511862567	21.97165688	0.8788662751
Serial	none	1	2700	0.029889	0	-	0.03343059543	-	-
Row	async-overlapped	9	2700	0.003321	0.0000200286	8.946047334	0.007256998354	4.606669837	0.5118522041
Row	async-overlapped	25	2700	0.00119556	0.0000200286	24.58808844	0.003309040793	10.10280547	0.4041122189
Block	async-overlapped	9	2700	0.003321	0.0000200286	8.946047334	0.02835645014	1.178941485	0.1309934983
Block	async-overlapped	25	2700	0.00119556	0.0000200286	24.58808844	0.004420017821	7.563452633	0.3025381053
Serial	none	1	5400	0.119556	0	-	0.1357311835	-	-
Row	async-overlapped	9	5400	0.013284	0.0000400572	8.972942566	0.08093484275	1.677042654	0.1863380726
Row	async-overlapped	25	5400	0.00478224	0.0000400572	24.79233341	0.009785168157	13.87111405	0.5548445618
Block	async-overlapped	9	5400	0.013284	0.0000400572	8.972942566	0.04125704364	3.289891167	0.365543463
Block	async-overlapped	25	5400	0.00478224	0.0000400572	24.79233341	0.01893297895	7.169034721	0.2867613888
Serial	none	1	9000	0.3321	0	-	1.029113724	-	-
Row	async-overlapped	9	9000	0.0369	0.000066762	8.983745993	0.09215478849	11.16723006	1.24080334
Row	async-overlapped	25	9000	0.013284	0.000066762	24.87498466	0.1135512767	9.062986822	0.3625194729
Block	async-overlapped	9	9000	0.0369	0.000066762	8.983745993	0.08944150499	11.5059974	1.278444156
Block	async-overlapped	25	9000	0.013284	0.000066762	24.87498466	0.06146253321	16.74375706	0.6697502823
Serial	none	1	12600	0.650916	0	-	2.018465143	-	-
Row	async-overlapped	9	12600	0.072324	0.0000934668	8.988384001	0.1809005713	11.15787047	1.239763386
Row	async-overlapped	25	12600	0.02603664	0.0000934668	24.91057557	0.1603833082	12.58525694	0.5034102778
Block	async-overlapped	9	12600	0.072324	0.0000934668	8.988384001	0.1768903334	11.41082786	1.267869762
Block	async-overlapped	25	12600	0.02603664	0.0000934668	24.91057557	0.1031347964	19.57113617	0.7828454467
ta	0.0000000041								
ts	0.000004685								
tc	0.000000003709								

Distribution	Communication Type	NP	Input World Size	Estimated Computation Time	Estimation Communication Time	Estimated Speedup	Actual Time	Actual Speedup	Efficiency
Serial	none	1	900	0.0140292	0	-	0.01403986202	-	-
Row	sync	9	900	0.0015588	0.0000160462	8.908298474	0.001606867478	8.737411278	0.9708234753
Row	sync	25	900	0.000561168	0.0000160462	24.30501537	0.0005892110658	23.82823887	0.953129555
Block	sync	9	900	0.0015588	0.000023175964	8.868149908	0.002269642489	6.185935491	0.6873261656
Block	sync	25	900	0.000561168	0.000021395644	24.08183234	0.0006012503688	23.35110755	0.9340443019
Row	async	9	900	0.0015588	0.0000066762	8.961618196	0.002275048075	0.2642802916	0.02936447684
Row	async	25	900	0.000561168	0.0000066762	24.70607255	0.0008207536913	17.10606017	0.6842424066
Block	async	9	900	0.0015588	0.000004435964	8.974460877	0.001616853613	8.683446611	0.9648274012
Block	async	25	900	0.000561168	0.000002655644	24.88224846	0.0005731589801	24.49558065	0.9798232261
Serial	none	1	2700	0.1262628	0	-	0.1267200522	-	-
Row	sync	9	2700	0.0140292	0.0000293986	8.981179675	0.01444039086	8.775389359	0.9750432621
Row	sync	25	2700	0.005050512	0.0000293986	24.8553193	0.005199314016	24.37245603	0.9748982412
Block	sync	9	2700	0.0140292	0.000032077564	8.979468574	0.01446238505	8.762043865	0.9735604295
Block	sync	25	2700	0.005050512	0.000026736604	24.86835092	0.005277026535	24.01353327	0.9605413306
Row	async	9	2700	0.0140292	0.0000200286	8.987169587	0.03049824951	4.15499428	0.4616660311
Row	async	25	2700	0.005050512	0.0000200286	24.90125017	0.01100568915	11.51404974	0.4605619896
Block	async	9	2700	0.0140292	0.000013337564	8.991451824	0.01712030573	7.40174003	0.8224155588
Block	async	25	2700	0.005050512	0.000007996604	24.96047944	0.00510036826	24.84527505	0.9938110019
Serial	none	1	5400	0.5050512	0	-	0.5003820885	-	-
Row	sync	9	5400	0.0561168	0.0000494272	8.992079853	0.05788604434	8.644261223	0.9604734692
Row	sync	25	5400	0.020202048	0.0000494272	24.93898321	0.02082800148	24.0244888	0.9609795521
Block	sync	9	5400	0.0561168	0.000045429964	8.992719846	0.05795468548	8.634023018	0.9593358909
Block	sync	25	5400	0.020202048	0.000034748044	24.95707319	0.02087042279	23.97565653	0.9590262612
Row	async	9	5400	0.0561168	0.0000400572	8.993580218	0.1017368341	4.918396495	0.5464884994
Row	async	25	5400	0.020202048	0.0000400572	24.95052738	0.03770358195	13.2714735	0.5308589398
Block	async	9	5400	0.0561168	0.000026689964	8.995721504	0.05618877831	8.905374053	0.9894860059
Block	async	25	5400	0.020202048	0.000016008044	24.98020576	0.02003748288	24.97230273	0.9988921093
Serial	none	1	9000	1.40292	0	-	1.390032655	-	-
Row	sync	9	9000	0.15588	0.000076132	8.995606534	0.1605425933	8.658341858	0.9620379843
Row	sync	25	9000	0.0561168	0.000076132	24.96612919	0.05783255178	24.03547159	0.9614188636
Block	sync	9	9000	0.15588	0.000063233164	8.996350605	0.1618214723	8.589914771	0.9544349746
Block	sync	25	9000	0.0561168	0.000045429964	24.97977735	0.05793522534	23.99287562	0.9597150247
Row	async	9	9000	0.15588	0.000066762	8.996147031	0.3115287974	4.461971628	0.4957746253
Row	async	25	9000	0.0561168	0.000066762	24.97029291	0.1149880169	12.08850011	0.4835400045
Block	async	9	9000	0.15588	0.000044493164	8.997431844	0.1607266602	8.648426178	0.960936242
Block	async	25	9000	0.0561168	0.000026689964	24.98811529	0.05573969975	24.93792865	0.997517146
Serial	none	1	12600	2.7497232	0	-	2.833855488	-	-
Row	sync	9	12600	0.3055248	0.0001028368	8.996971703	0.3056398251	9.271879038	1.030208782
Row	sync	25	12600	0.109988928	0.0001028368	24.97664748	0.1570677382	18.04225056	0.7216900223
Block	sync	9	12600	0.3055248	0.000081036364	8.997613503	0.315937177	8.969680348	0.9966311498
Block	sync	25	12600	0.109988928	0.000056111884	24.98725252	0.1138042295	24.90114383	0.9960457532
Row	async	9	12600	0.3055248	0.0000934668	8.997247543	0.628511613	4.508835524	0.5009817248
Row	async	25	12600	0.109988928	0.0000934668	24.97877345	0.2205768469	12.84747483	0.5138989931

Block	async	9	12600	0.3055248	0.000062296364	8.998165278	0.4874276605	5.813899616	0.6459888462
Block	async	25	12600	0.109988928	0.000037371884	24.99150842	0.1092607999	25.93661671	1.037464668
Serial	none	1	900	0.0140292	0	-	0.01386801092	-	-
Row	async-overlapped	9	900	0.0015588	0.0000066762	8.961618196	0.002268790483	6.112512822	0.6791680913
Row	async-overlapped	25	900	0.000561168	0.0000066762	24.70607255	0.0007151177934	19.3926246	0.7757049841
Block	async-overlapped	9	900	0.0015588	0.0000066762	8.961618196	0.001571201347	8.826374123	0.9807082359
Block	async-overlapped	25	900	0.000561168	0.0000066762	24.70607255	0.0005913935863	23.44971477	0.9379885908
Serial	none	1	2700	0.1262628	0	-	0.1250137977	-	-
Row	async-overlapped	9	2700	0.0140292	0.0000200286	8.987169587	0.0242733031	5.150258997	0.5722509997
Row	async-overlapped	25	2700	0.005050512	0.0000200286	24.90125017	0.005188824963	24.09289166	0.9637156665
Block	async-overlapped	9	2700	0.0140292	0.0000200286	8.987169587	0.01761688865	7.096247252	0.7884719168
Block	async-overlapped	25	2700	0.005050512	0.0000200286	24.90125017	0.005009631401	24.95468981	0.9981875926
Serial	none	1	5400	0.5050512	0	-	0.5004488457	-	-
Row	async-overlapped	9	5400	0.0561168	0.0000400572	8.993580218	0.05779953519	8.658354156	0.9620393506
Row	async-overlapped	25	5400	0.020202048	0.0000400572	24.95052738	0.01989232658	25.15788406	1.006315362
Block	async-overlapped	9	5400	0.0561168	0.0000400572	8.993580218	0.05599607686	8.937212637	0.9930236263
Block	async-overlapped	25	5400	0.020202048	0.0000400572	24.95052738	0.01996542648	25.06577289	1.002630916
Serial	none	1	9000	1.40292	0	-	1.40637909	-	-
Row	async-overlapped	9	9000	0.15588	0.000066762	8.996147031	0.1608501652	8.743410915	0.9714901016
Row	async-overlapped	25	9000	0.0561168	0.000066762	24.97029291	0.05804391553	24.22956958	0.9691827832
Block	async-overlapped	9	9000	0.15588	0.000066762	8.996147031	0.1607492257	8.748901177	0.9721001308
Block	async-overlapped	25	9000	0.0561168	0.000066762	24.97029291	0.056245958	25.00409167	1.000163667
Serial	none	1	12600	2.7497232	0	-\	2.722661213	-	-
Row	async-overlapped	9	12600	0.3055248	0.0000934668	8.997247543	0.3061578983	8.89299681	0.9881107567
Row	async-overlapped	25	12600	0.109988928	0.0000934668	24.97877345	0.1106437369	24.60745895	0.9842983579
Block	async-overlapped	9	12600	0.3055248	0.0000934668	8.997247543	0.3156058796	8.626775954	0.9585306616
Block	async-overlapped	25	12600	0.109988928	0.0000934668	24.97877345	0.1089494404	24.99013491	0.9996053964
ta	0.00000001732								
ts	0.000004685								
tc	0.000000003709								

```
// Conway's Game of Life
// Global variable include file
//
// CSCI 4576/5576 High Performance Scientific Computing
// Matthew Woitaszek

// <soapbox>
// This file contains global variables: variables that are defined throughout
// the entire program, even between multiple independent source files. Of
// course, global variables are generally bad, but they're useful here because
// it allows all of the source files to know their rank and the number of MPI
// tasks. But don't use it lightly.
//
// How it works:
//  * One .cpp file -- usually the one that contains main(), includes this file
//    within #define __MAIN, like this:
//      #define __MAIN
//      #include globals.h
//      #undef __MAIN
//  * The other files just "#include globals.h"

typedef enum { SERIAL, ROW, GRID } dist;

#ifdef __MAIN
int                     rank;
int                     np;
int                     my_name_len;
char                    my_name[255];
#else
extern int              rank;
extern int              np;
extern int              my_name_len;
extern char             *my_name;
#endif

//
// Conway globals
//
#ifdef __MAIN

int                     nrows;          // Number of rows in our partitioning
int                     ncols;          // Number of columns in our partitioning
int                     my_row;         // My row number
int                     my_col;         // My column number

// Local logical game size
int                     fake_data_size;
int                     local_width;    // Width and height of game on this processor
int                     local_height;
int                     global_width;
int                     global_height;
int                     N;

// Local physical field size
int                     field_width;        // Width and height of field on this processor
int                     field_height;       // (should be local_width+2, local_height+2)
unsigned char           *env_a;             // 1D character array to represent our 1st 2D en
vironment
unsigned char           *env_b;             // 1D character array to represent our 2nd 2D en
vironment
unsigned char           *out_buffer;        // 1D character array to represent our global 2D
 environment + padding
dist                     dist_type;

#else
extern int              nrows;
extern int              ncols;
extern int              my_row;
extern int              my_col;

extern int              fake_data_size;
extern int              local_width;
extern int              local_height;
extern int              global_width;
extern int              global_height;
extern int              N;

extern int              field_width;
extern int              field_height;
extern unsigned char    *env_a;
extern unsigned char    *env_b;
extern unsigned char    *out_buffer;

extern dist              dist_type;

#endif
```

```
/*
 * Helper function file to be included in main
 * Written by Adam Ross
 *
 */

void print_usage();
void print_matrix(unsigned char *matrix);
void print_padded_matrix(unsigned char *matrix);
void print_global_matrix(unsigned char *matrix);
void swap(unsigned char **a, unsigned char **b);
unsigned char *Allocate_Square_Matrix();
int count_alive(unsigned char *matrix);
int Calc_Confidence_Interval_stop(double *timing_data, int n);
```

```
typedef enum { false, true } bool; // Provide C++ style 'bool' type in C
bool readpgm( char *filename );
```

```
/* $Id: pprintf.h,v 1.3 2006/02/09 20:42:25 mccreary Exp $ */

/*
 * Copyright (c) 2006 Sean McCreary <mccreary@mcwest.org>. All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 *
 * 3. The name of the author may not be used to endorse or promote products
 * derived from this software without specific prior written permission
 *
 * THIS SOFTWARE IS PROVIDED ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
 * THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

// Modified by Michael Oberg, 2015/10/01 to support both C or C++

#ifdef __cplusplus
extern "C" int init_pprintf(int);
extern "C" int pp_set_banner(char *);
extern "C" int pp_reset_banner();
extern "C" int pprintf(char *, ...);
#endif

extern int init_pprintf(int);
extern int pp_set_banner(char *);
extern int pp_reset_banner();
extern int pprintf(char *, ...);
```

```
/* MT1 - Midterm Part I: Conway's Game of Line
 *
 *
 * Name: Adam Ross
 *
 * Input: -i filename, -d distribution type <0 - serial, 1 - row, 2 - grid>
 *        -s turn on asynchronous MPI functions, -c <#> if and when to count living
 * Output: Various runtime information including bug counting if turned on
 *
 *
 * Note: a Much of this code, namely the pgm reader and most of the support libraries
 * is credited to: Dr. Matthew Woitaszek
 *
 * Written by Adam Ross, modified from code supplied by Michael Oberg, modified from code su
pplied by Dr. Matthew Woitaszek
 */

#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <math.h>
#include <string.h>
#include "mpi.h"

// Include global variables. Only this file needs the #define
#define __MAIN
#include "globals.h"
#undef __MAIN

// User includes
#include "pprintf.h"
#include "pgm.h"
#include "helper.h"

int main(int argc, char* argv[]) {
    unsigned short      i, j;
    unsigned short      neighbors =         0;
    int                 half_height;
    int                 top_dest,
                        top_source,
                        bot_dest ,
                        bot_source,
                        left_dest,
                        left_source,
                        right_dest,
                        right_source =     5280;
    MPI_Status          status;
    MPI_Request         ar, br, lr, rr;
    MPI_File            out_file;
    int                 counting =          -1;
    int                 count =             0;
    int                 total =             0;
    int                 n =                 0;
    int                 option =            -1;
    bool                async =             false;
    bool                writing =           false;
    int                 iter_num =          1000;
    char                *filename;
    char                frame[47];
    int                 gsizes[2], distribs[2], dargs[2], psizes[2];
    MPI_Datatype        ext_array;
    MPI_Datatype        darray;
    MPI_Datatype        column;
    double              start;
    double              finish;
    double              *timing_data;
    double              avg =               0;

    fake_data_size = 0;

    // Parse commandline
    while ((option = getopt(argc, argv, "d:an:c:i:ws:")) != -1) {
        switch (option) {
            case 'd' :
                dist_type = atoi(optarg);
                break;
            case 'a' :
                async = true;
                break;
            case 'n' :
                iter_num = atoi(optarg);
                break;
            case 'c' :
                counting = atoi(optarg);
                break;
            case 'i' :
                filename = optarg;
                break;
            case 'w' :
                writing = true;
                break;
            case 's' :
                fake_data_size = atoi(optarg);
                break;
            default:
                print_usage();
                exit(1);
        }
    }

    // Initialize MPI
    MPI_Init(&argc, &argv);

    // Get the communicator and process information
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &np);

    // Print rank and hostname
    MPI_Get_processor_name(my_name, &my_name_len);
    printf("Rank %i is running on %s\n", rank, my_name );

    // Initialize the pretty printer
    init_pprintf(rank);
    pp_set_banner("main");

    timing_data = (double *) calloc(iter_num, sizeof(double));

    if (rank == 0) {
        pprintf("Welcome to Conway's Game of Life!\n");
    }

    //
    // Determine the partitioning
    //
    if (dist_type < GRID) {
        if (!rank)
```

```
            pprintf("Row or Serial distribution selected.\n");
        ncols = 1;
        nrows = np;
        my_col = 0;
        my_row = rank;
    } else {
        if (!rank)
            pprintf("Grid distribution selected.\n");
        nrows = (int)sqrt(np);
        ncols = (int)sqrt(np);
        my_row = rank / nrows;
        my_col = rank - my_row * nrows;

        //pprintf("Num rows%d\tNum cols %d\tMy row %d\tMy col %d\n", nrows, ncols, my_row, m
y_col);
    }

    if (np != nrows * ncols) {
        if (!rank)
             pprintf("Error: %ix%i partitioning requires %i np (%i provided)\n",
                  nrows, ncols, nrows * ncols, np );
        MPI_Finalize();
        return 1;
    }


    // Now, calculate neighbors (N, S, E, W, NW, NE, SW, SE)
    // ... which means you ...


    // Read the PGM file. The readpgm() routine reads the PGM file and, based
    // on the previously set nrows, ncols, my_row, and my_col variables, loads
    // just the local part of the field onto the current processor. The
    // variables local_width, local_height, field_width, field_height, as well
    // as the fields (field_a, field_b) are allocated and filled.
    if (!readpgm(filename)) {
        if (rank == 0)
            pprintf("An error occured while reading the pgm file\n");
        MPI_Finalize();
        return 1;
    }

    // Set half array values for async work
    half_height = (local_height / 2) + 1;

    // Set up darray create properties
    gsizes[0] = global_height; /* no. of rows in global array */
    gsizes[1] = global_width; /* no. of columns in global array*/
    distribs[0] = MPI_DISTRIBUTE_BLOCK;
    distribs[1] = MPI_DISTRIBUTE_BLOCK;
    dargs[0] = MPI_DISTRIBUTE_DFLT_DARG;
    dargs[1] = MPI_DISTRIBUTE_DFLT_DARG;
    psizes[0] = nrows; /* no. of processes in vertical dimension of process grid */
    psizes[1] = ncols; /* no. of processes in horizontal dimension of process grid */

    // Create darray and commit
    MPI_Type_create_darray(np, rank, 2, gsizes, distribs, dargs, psizes, MPI_ORDER_C, MPI_UN
SIGNED_CHAR, &darray);
    MPI_Type_commit(&darray);

    // Create data type to extract useful data out of padding
    MPI_Type_vector(local_height, local_width, field_width, MPI_UNSIGNED_CHAR, &ext_array);
    MPI_Type_commit(&ext_array);

    // Build MPI datatype vector of every Nth item - i.e. a column
    MPI_Type_vector(local_height, 1, field_width, MPI_UNSIGNED_CHAR, &column);
    MPI_Type_commit(&column);

    // allocate memory to print whole stages into pgm files for animation
    if (rank == 0) {
        out_buffer = Allocate_Square_Matrix(global_width, global_height);
    }

    // Count initial living count
    if (counting != -1) {
        count = count_alive(env_a);
        pprintf("Bugs alive at the start: %d\n", count);

        MPI_Reduce(&count, &total, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
        if (rank == 0) {
            pprintf("%i total bugs alive at the start.\n", total);
        }
    }

    // calculate pairings
    if (dist_type > SERIAL) {
        // calculate pairings
        if (dist_type == ROW) { // row distro
            top_dest = bot_source = rank - 1;
            top_source = bot_dest = rank + 1;

            if (rank == 0) { // rank 0, no need to send
                top_dest = MPI_PROC_NULL;
                bot_source = MPI_PROC_NULL;
            } else if (rank == (np - 1)) { // rank np-1 no need to send
                top_source = MPI_PROC_NULL;
                bot_dest = MPI_PROC_NULL;
            }
        } else if (dist_type == GRID) {
        // calculate pairings
            top_dest = bot_source = rank - nrows;
            top_source = bot_dest = rank + nrows;
            left_dest = right_source = rank - 1;
            left_source = right_dest = rank + 1;

            if (my_row == 0) { // top row no need to send up
                top_dest = MPI_PROC_NULL;
                bot_source = MPI_PROC_NULL;
            } else if (my_row == sqrt(np) - 1) { // rank bottom row no need to send down
                top_source = MPI_PROC_NULL;
                bot_dest = MPI_PROC_NULL;
            }
            if (my_col == 0) {
                left_dest = MPI_PROC_NULL;
                right_source = MPI_PROC_NULL;
            } else if (my_col == sqrt(np) - 1) {
                left_source = MPI_PROC_NULL;
                right_dest = MPI_PROC_NULL;
            }
            //pprintf("top: %d\tbot %d\tleft %d\tright %d\tProc %d\n", top_dest, bot_dest, l
eft_dest, right_dest, MPI_PROC_NULL);
        }
    }

    while(n < iter_num) {
```

```
        if (writing) {
            for (int k = 1; k < field_height - 1; k++) {
                for (int a = 1; a < field_width - 1; a++) {
                    if (!env_b[k * field_width + a]) {
                        env_a[k * field_width + a] = 255;
                    } else {
                        env_a[k * field_width + a] = 0;
                    }
                }
            }

            sprintf(frame, "/oasis/scratch/comet/adamross/temp_project/%d.pgm", n);
            MPI_File_open(MPI_COMM_WORLD, frame, MPI_MODE_CREATE|MPI_MODE_WRONLY, MPI_INFO_N
ULL, &out_file);

            char header[20];
            sprintf(header, "P5\n%d %d\n%d\n", global_width, global_height, 255);
            int header_len = strlen(header);

            if (rank == 0) {
                //write header
                //MPI_File_set_view(out_file, 0,  MPI_UNSIGNED_CHAR, MPI_UNSIGNED_CHAR, "nat
ive", MPI_INFO_NULL);
                MPI_File_write(out_file, &header, header_len, MPI_UNSIGNED_CHAR, MPI_STATUS_
IGNORE);
            }

            // write data
            //MPI_File_set_view(out_file, 15 + rank * local_width + local_width, MPI_UNSIGNE
D_CHAR, darray, "native", MPI_INFO_NULL);
            MPI_File_set_view(out_file, header_len, MPI_UNSIGNED_CHAR, darray, "native", MPI
_INFO_NULL);

            //MPI_File_write(out_file, env_a, (local_height * local_width), ext_array, &stat
us);
            MPI_File_write(out_file, &env_a[field_width + 12], 1, ext_array, &status);
            MPI_File_close(&out_file);

            for (int k = 1; k < field_height - 1; k++) {
                for (int a = 1; a < field_width  - 1; a++) {
                    if (!env_a[k * field_width + a]) {
                        env_a[k * field_width + a] = 0;
                    } else {
                        env_a[k * field_width + a] = 1;
                    }
                }
            }
        }

        start = MPI_Wtime();

        //Uncomment to produce pgm files per frame in serial file system
        //MPI_Gather(&env_b[field_width + 1], 1, ext_array, out_buffer, local_width * local_
height, MPI_UNSIGNED_CHAR, 0, MPI_COMM_WORLD);
        /*if (rank == 0) {
            print_global_matrix(out_buffer);
        }*/

        /*if (rank == 0) {
            for (int k = 0; k < global_height; k++) {
                for (int a = 0; a < global_width; a++) {
                    if (!out_buffer[k * global_width + a]) {
                        out_buffer[k * global_width + a] = 255;
                    } else {
                        out_buffer[k * global_width + a] = 0;
                    }
                }
            }

            sprintf(frame, "%d.pgm", n);
            FILE *file = fopen(frame, "w");
            fprintf(file, "P5\n");
            fprintf(file, "%d %d\n", global_width, global_height);
            fprintf(file, "%d\n", 255);
            fwrite(out_buffer, sizeof(unsigned char), global_width * global_height, file);
            fclose(file);
        }*/

        ////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////

        // do upper half minus edges, check if need recv
        // do lower half minus edges, check is need recv
        // do upper row
        // do columns
        // do lower row

        if (async && dist_type == ROW && n < iter_num - 1) {
            // Aschrnous enabled, receive from the last iteration or inital setup
            MPI_Irecv(&env_a[(field_height - 1) * field_width + 0], field_width, MPI_UNSIGNE
D_CHAR, top_source, 0, MPI_COMM_WORLD, &ar);
            MPI_Irecv(&env_a[0 * field_width + 0], field_width, MPI_UNSIGNED_CHAR, bot_sourc
e, 0, MPI_COMM_WORLD, &br);

            MPI_Isend(&env_b[1 * field_width + 0], field_width, MPI_UNSIGNED_CHAR, top_dest,
 0, MPI_COMM_WORLD, &ar);
            MPI_Isend(&env_b[(field_height - 2) * field_width + 0], field_width, MPI_UNSIGNE
D_CHAR, bot_dest, 0, MPI_COMM_WORLD, &br);
        } else if (async && dist_type == GRID && n < iter_num - 1) {
            MPI_Irecv(&env_a[2 * field_width - 1], 1, column, left_source, 0, MPI_COMM_WORLD
, &lr);
            MPI_Irecv(&env_a[1 * field_width + 0], 1, column, right_source, 0, MPI_COMM_WORL
D, &rr);

            MPI_Isend(&env_b[1 * field_width + 1], 1, column, left_dest, 0, MPI_COMM_WORLD,
&lr);
            MPI_Isend(&env_b[2 * field_width - 2], 1, column, right_dest, 0, MPI_COMM_WORLD,
 &rr);
        }

        ////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////

        // calulate neighbors and form state + 1 for upper half - edges
        for (i = 2; i < half_height; i++) {
            for (j = 2; j < local_width; j++) {
                neighbors = 0;
                // loop unroll neighbor checking - access row dominant
                neighbors += env_a[(i - 1) * field_width + j - 1] + env_a[(i - 1) * field_wi
dth + j] + env_a[(i - 1) * field_width + j + 1];
                neighbors += env_a[i * field_width + j - 1] +
               env_a[i * field_width + j + 1];
                neighbors += env_a[(i + 1) * field_width + j - 1] + env_a[(i + 1) * field_wi
dth + j] + env_a[(i + 1) * field_width + j + 1];

                // Determine env_b based on neighbors in env_a
```

```
                if (neighbors == 2) {
                    env_b[i * field_width + j] = env_a[i * field_width + j]; // exactly 2 sp
awn
                } else if (neighbors == 3) {
                    env_b[i * field_width + j] = 1; // exactly 3 spawn
                } else {
                    env_b[i * field_width + j] = 0; // zero or one or 4 or more die

                }
            }
        }

        ////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////

        // Receive our horizontal communication and send the vertical
        if (async && dist_type == GRID && n > 0) {
            // Need the horizontal data before we send vertically
            MPI_Wait(&lr, &status);
            MPI_Wait(&rr, &status);

            // Aschrnous enabled, receive from the last iteration or inital setup
            MPI_Irecv(&env_a[(field_height - 1) * field_width + 0], field_width, MPI_UNSIGNE
D_CHAR, top_source, 0, MPI_COMM_WORLD, &ar);
            MPI_Irecv(&env_a[0 * field_width + 0], field_width, MPI_UNSIGNED_CHAR, bot_sourc
e, 0, MPI_COMM_WORLD, &br);

            MPI_Isend(&env_a[1 * field_width + 0], field_width, MPI_UNSIGNED_CHAR, top_dest,
 0, MPI_COMM_WORLD, &ar);
            MPI_Isend(&env_a[(field_height - 2) * field_width + 0], field_width, MPI_UNSIGNE
D_CHAR, bot_dest, 0, MPI_COMM_WORLD, &br);
        }

        ////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////

        // calulate neighbors and form state + 1 for lower half - edges
        for (i = half_height; i < local_height; i++) {
            for (j = 2; j < local_width; j++) {
                neighbors = 0;
                // loop unroll neighbor checking - access row dominant
                neighbors += env_a[(i - 1) * field_width + j - 1] + env_a[(i - 1) * field_wi
dth + j] + env_a[(i - 1) * field_width + j + 1];
                neighbors += env_a[i * field_width + j - 1] +
               env_a[i * field_width + j + 1];
                neighbors += env_a[(i + 1) * field_width + j - 1] + env_a[(i + 1) * field_wi
dth + j] + env_a[(i + 1) * field_width + j + 1];

                // Determine env_b based on neighbors in env_a
                if (neighbors == 2) {
                    env_b[i * field_width + j] = env_a[i * field_width + j]; // exactly 2 sp
awn
                } else if (neighbors == 3) {
                    env_b[i * field_width + j] = 1; // exactly 3 spawn
                } else {
                    env_b[i * field_width + j] = 0; // zero or one or 4 or more die

                }
            }
        }

        ////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////

        if (async && n > 0) {
            // To avoid getting data mixed up wait for it to come through
            MPI_Wait(&ar, &status);
            MPI_Wait(&br, &status);
        }

        ////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////

        // calulate neighbors and form state + 1 for edges
        i = 1;
        for (j = 1; j < local_width + 1; j++) {
            neighbors = 0;
            // loop unroll neighbor checking - access row dominant
            neighbors += env_a[(i - 1) * field_width + j - 1] + env_a[(i - 1) * field_width
+ j] + env_a[(i - 1) * field_width + j + 1];
            neighbors += env_a[i * field_width + j - 1] +
          env_a[i * field_width + j + 1];
            neighbors += env_a[(i + 1) * field_width + j - 1] + env_a[(i + 1) * field_width
+ j] + env_a[(i + 1) * field_width + j + 1];

            // Determine env_b based on neighbors in env_a
            if (neighbors == 2) {
                env_b[i * field_width + j] = env_a[i * field_width + j]; // exactly 2 spawn
            } else if (neighbors == 3) {
                env_b[i * field_width + j] = 1; // exactly 3 spawn
            } else {
                env_b[i * field_width + j] = 0; // zero or one or 4 or more die

            }
        }

        // calulate neighbors and form state + 1 for edges
        for (i = 1; i < local_height; i++) {
            // need i = 1 and local_width + 1
            for (j = 1; j < local_width + 1; j += local_width - 1) {
                neighbors = 0;
                // loop unroll neighbor checking - access row dominant
                neighbors += env_a[(i - 1) * field_width + j - 1] + env_a[(i - 1) * field_wi
dth + j] + env_a[(i - 1) * field_width + j + 1];
                neighbors += env_a[i * field_width + j - 1] +
               env_a[i * field_width + j + 1];
                neighbors += env_a[(i + 1) * field_width + j - 1] + env_a[(i + 1) * field_wi
dth + j] + env_a[(i + 1) * field_width + j + 1];

                // Determine env_b based on neighbors in env_a
                if (neighbors == 2) {
                    env_b[i * field_width + j] = env_a[i * field_width + j]; // exactly 2 sp
awn
                } else if (neighbors == 3) {
                    env_b[i * field_width + j] = 1; // exactly 3 spawn
                } else {
                    env_b[i * field_width + j] = 0; // zero or one or 4 or more die

                }
            }
        }

        // calulate neighbors and form state + 1 for edges
        i = local_height;
        for (j = 1; j < local_width + 1; j++) {
            neighbors = 0;
```

```
            // loop unroll neighbor checking - access row dominant
            neighbors += env_a[(i - 1) * field_width + j - 1] + env_a[(i - 1) * field_width
+ j] + env_a[(i - 1) * field_width + j + 1];
            neighbors += env_a[i * field_width + j - 1] +
       env_a[i * field_width + j + 1];
            neighbors += env_a[(i + 1) * field_width + j - 1] + env_a[(i + 1) * field_width
+ j] + env_a[(i + 1) * field_width + j + 1];

            // Determine env_b based on neighbors in env_a
            if (neighbors == 2) {
                env_b[i * field_width + j] = env_a[i * field_width + j]; // exactly 2 spawn
            } else if (neighbors == 3) {
                env_b[i * field_width + j] = 1; // exactly 3 spawn
            } else {
                env_b[i * field_width + j] = 0; // zero or one or 4 or more die

            }
        }

        ////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////

        // If we are doing async we now have the data we need for the next iter, send it
        // If we are in row distrobution send vertically - thats all we need to do
        // If we are in block distrobution send horizontally first

        // sync or a async here MPI_PROC_NULs
        if (dist_type > SERIAL && !async) {
            // If we choose block decomposition send horizontally first
            if (dist_type == GRID) {
                // Send to right or recv from left
                MPI_Sendrecv(&env_b[1 * field_width + 1], 1, column, left_dest, 0,
                             &env_b[2 * field_width - 1], 1, column, left_source, 0, MPI_COM
M_WORLD, &status);
                // Send to left or recv from right
                MPI_Sendrecv(&env_b[2 * field_width - 2], 1, column, right_dest, 0,
                             &env_b[1 * field_width + 0], 1, column, right_source, 0, MPI_CO
MM_WORLD, &status);
            }
            // Send to below or recv from above
            MPI_Sendrecv(&env_b[1 * field_width + 0], field_width, MPI_UNSIGNED_CHAR, top_de
st, 0,
                         &env_b[(field_height - 1) * field_width + 0], field_width, MPI_UNSI
GNED_CHAR, top_source, 0, MPI_COMM_WORLD, &status);
            // Send to above or recv from below
            MPI_Sendrecv(&env_b[(field_height - 2) * field_width + 0], field_width, MPI_UNSI
GNED_CHAR, bot_dest, 0,
                         &env_b[0 * field_width + 0], field_width, MPI_UNSIGNED_CHAR, bot_so
urce, 0, MPI_COMM_WORLD, &status);
        }

        finish = MPI_Wtime();
        if (rank == 0 && n > 0) {
            timing_data[n] = finish - start;
        }

        // If counting is turned on print living bugs this iteration
        if (n != 0 && (n % counting) == 0) {
            count = count_alive(env_b);

            MPI_Reduce(&count, &total, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
            if (rank == 0) {
                pprintf("%i total bugs alive at iteraion %d\n", total, n);
            }
        }

        n++;
        swap(&env_b, &env_a);
    }

    if (rank == 0) {
        for (i = 1; i < n; i++) {
            avg += timing_data[i];
        }

        avg = avg / (n - 1);

        pprintf("avg: %1.20f\n", avg);
    }

    // Final living count
    if (counting != -1 && n != counting) {
        count = count_alive(env_a);
        pprintf("Per process bugs alive at the end: %d\n", count);

        MPI_Reduce(&count, &total, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
        if (rank == 0) {
            pprintf("%i total bugs alive at the end.\n", total);
        }
    }

    // Free the fields
    MPI_Barrier(MPI_COMM_WORLD);
    if (env_a != NULL) free( env_a );
    if (env_b != NULL) free( env_b );
    if (timing_data != NULL) free( timing_data );


    MPI_Finalize();


} /* end main */
```

```c
#include <stdio.h>
#include <stdlib.h>
#include "globals.h"
#include <math.h>

// Self explanitory
void print_usage() {
    printf("Usage: -i filename, -d distribution type <0 - serial, 1 - row, 2 - grid>, -s tur
n on asynchronous MPI functions, -c <#> if and when to count living\n");
}

/*
 * Helper method to print a square matrix
 * Input: a matrix and the order of that matrix
 */
void print_matrix(unsigned char *matrix) {
    unsigned char          i;
    unsigned char          j;

    //printf("local_width is: %d, local_height is: %d\n", local_width, local_height);

    for (i = 1; i < local_height + 1; i++) {
        for (j = 1; j < local_width + 1; j++) {
            printf("%u ", matrix[i * field_width + j]);
        }
        printf("\n");
    }
    printf("\n");
}

void print_padded_matrix(unsigned char *matrix) {
    unsigned char          i;
    unsigned char          j;

    //printf("local_width is: %d, local_height is: %d\n", local_width, local_height);

    for (i = 0; i < field_height; i++) {
        for (j = 0; j < field_width; j++) {
            printf("%u ", matrix[i * field_width + j]);
        }
        printf("\n");
    }
    printf("\n");
}

void print_global_matrix(unsigned char *matrix) {
    unsigned char          i;
    unsigned char          j;

    //printf("local_width is: %d, local_height is: %d\n", local_width, local_height);

    for (i = 0; i < global_height; i++) {
        for (j = 0; j < global_width; j++) {
            printf("%u ", matrix[i * global_width + j]);
        }
        printf("\n");
    }
    printf("\n");
}

/*
 * Helper function to swap array pointers
 * Input: array a and Array b
 */
void swap(unsigned char **a, unsigned char **b) {
    unsigned char          *tmp = *a;
    *a = *b;
    *b = tmp;
}

/*
 * Helper function to allocate 2D array of ints
 * Input: Order of the array
 */
unsigned char *Allocate_Square_Matrix(int width, int height) {
    unsigned char          *matrix;

    matrix = (unsigned char *) calloc(width * height, sizeof(unsigned char));

    return matrix;
}

/*
 * Helper function to clean up code duplication
 * Input: pointer to array
 */
int count_alive(unsigned char *matrix) {
    int                    count =           0;
    int                    i, j;

    for (i = 1; i < local_height + 1; i++) {
        for (j = 1; j < local_width + 1; j++) {
            if (matrix[i * field_width + j]) {
                count ++;
            }
        }
    }

    return count;
}

/* Helper function calculate the confidence interval, error margins and determine
 * if we should keep looping.
 * Returns 1 or 0 for conintue or stop.
*/
int Calc_Confidence_Interval_stop(double *timing_data, int n) {
    double              sum =               0.0;
    double              mean =              0.0;
    double              std_dev =           0.0;
    double              marg_err =          0.0;
    double              marg_perc =         100.0;
    int                 i;

    if (n > 2) {
        for (i = 0; i < n; i++) {
            sum += timing_data[i];
        }
        mean = sum / n;
        sum = 0.0;
        for (i = 0; i < n; i++) {
            sum += pow(timing_data[i] - mean, 2);
        }
        std_dev = sqrt(sum / n);
        marg_err = 1.96 * (std_dev / sqrt(n));
        marg_perc = (marg_err / mean) * 100;
    } else {
```

```
        return 0;
    }
    if (marg_perc > 5.0  && n < 20) {
        return 0;
    } else {
        printf("%d\t%1.20f\t%1.10f\t%1.10f\t%f\t", n, mean, std_dev, marg_err, marg_perc);

        return 1;
    }
}
```

```c
/*
 * HPGM helper functions to be included in main
 * Provided by Michael Oberg, Modified by Adam Ross
 *
 */

// System includes
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "mpi.h"
#include <math.h>

// User includes
#include "globals.h"
#include "pprintf.h"
#include "helper.h"

typedef enum { false, true } bool; // Provide C++ style 'bool' type in C

bool readpgm( char *filename ){
    // Read a PGM file into the local task
    //
    // Input: char *filename, name of file to read
    // Returns: True if file read successfully, False otherwise
    //
    // Preconditions:
    //  * global variables nrows, ncols, my_row, my_col must be set
    //
    // Side effects:
    //  * sets global variables local_width, local_height to local game size
    //  * sets global variables field_width, field_height to local field size
    //  * allocates global variables env_a and env_b
    int             x = 0;
    int             y = 0;
    int             start_x, start_y;
    int             b, lx, ly, ll;
    char            header[10];
    int             depth;
    int             rv;
    int             grab_width;
    int             grab_height;
    int             x_add = 1;
    int             y_add = 1;

    pp_set_banner( "pgm:readpgm" );

    // Open the file
    if (rank == 0)
        pprintf( "Opening file %s\n", filename );
    FILE *fp = fopen( filename, "r" );
    if (!fp) {
        pprintf( "Error: The file '%s' could not be opened.\n", filename );
        return false;
    }

    // Read the PGM header, which looks like this:
    //  |P5        magic version number
    //  |900 900       width height
    //  |255         depth
    rv = fscanf( fp, "%6s\n%i %i\n%i\n", header, &global_width, &global_height, &depth );
    if (rv != 4){
        if (rank == 0)
            pprintf( "Error: The file '%s' did not have a valid PGM header\n", filename );
        return false;
    }

    if (fake_data_size != 0) {
        global_width = global_height = fake_data_size;
    }

    if (rank == 0)
        pprintf( "%s: %s %i %i %i\n", filename, header, global_width, global_height, depth )
;

    // Make sure the header is valid
    if (strcmp( header, "P5")) {
        if(rank==0)
            pprintf( "Error: PGM file is not a valid P5 pixmap.\n" );
        return false;
    }
    if (depth != 255) {
        if (rank == 0)
            pprintf( "Error: PGM file has depth=%i, require depth=255 \n", depth );
        return false;
    }

    // Make sure that the width and height are divisible by the number of
    // processors in x and y directions

    if (global_width % ncols) {
        if (rank == 0)
            pprintf( "Error: %i pixel width cannot be divided into %i cols\n", global_width,
 ncols );
        return false;
    }
    if (global_height % nrows) {
        if (rank == 0)
            pprintf( "Error: %i pixel height cannot be divided into %i rows\n", global_heigh
t, nrows );
        return false;
    }

    // Divide the total image among the local processors
    local_width = global_width / ncols;
    local_height = global_height / nrows;

    // Find out where my starting range is
    start_x = local_width * my_col;
    start_y = local_height * my_row;

    grab_width = local_width;
    grab_height = local_height;

    pprintf( "Hosting data for x:%03i-%03i y:%03i-%03i\n",
        start_x, start_x + local_width,
        start_y, start_y + local_height );

    // Create the array!
    field_width = local_width + 2;
    field_height = local_height + 2;

    // allocate contiguous memory - returns a pointer to the memory
    env_a = Allocate_Square_Matrix(field_width, field_height);
    env_b = Allocate_Square_Matrix(field_width, field_height);
```

```
    // Need to handle edge cases to not grab extras
     if (dist_type == ROW ){
        grab_height = field_height;

        if (rank == 0) {
            grab_height--;
        } else if (rank == np - 1) {
            grab_height--;
            start_y--;
            y_add = 0;
        } else {
            start_y--;
            y_add = 0;
        }
    } else if (dist_type == GRID) {
        grab_width = field_width;
        grab_height = field_height;

        if (my_row == 0) {
            grab_height--;
        } else if (my_row == sqrt(np) - 1) {
            grab_height--;
            start_y--;
            y_add = 0;
        } else {
            start_y--;
            y_add = 0;
        }

        if (my_col == 0) {
            grab_width--;
        } else if (my_col == sqrt(np) - 1) {
            grab_width--;
            start_x--;
            x_add = 0;
        } else {
            start_x--;
            x_add = 0;
        }
    }

    //pprintf("start_x: %d\tstart_y: %d\tx_add: %d\ty_add: %d\t\n", start_x, start_y, x_add,
y_add);
    //pprintf("grab_width: %d\tgrab_height: %d\t\n", grab_width, grab_height);

    // Read the data from the file. Save the local data to the local array.
    if (fake_data_size == 0) {
        for (y = 0; y < global_height; y++) {
            for (x = 0; x < global_width; x++) {
                // Read the next character
                b = fgetc(fp);
                if (b == EOF){
                    pprintf( "Error: Encountered EOF at [%i,%i]\n", y,x );
                    return false;
                }

                // From the PGM, black cells (b=0) are bugs, all other
                // cells are background
                if (b == 0) {
                    b = 1;
                } else {
                    b = 0;
                }

                // If the character is local, then save it!
                if (x >= start_x &&
                    x < start_x + grab_width &&
                    y >= start_y &&
                    y < start_y + grab_height) {

                    // Calculate the local pixels (+1 for ghost row,col)
                    lx = x - start_x + x_add;
                    ly = y - start_y + y_add;
                    ll = (ly * field_width + lx);
                    env_a[ll] = b;
                    env_b[ll] = b;
                } // save local point


            } // for x
        } // for y
    }



    fclose(fp);

    pp_reset_banner();
    return true;
}
```

```
/* $Id: pprintf.c,v 1.5 2006/02/09 20:42:25 mccreary Exp $ */

/*
 * Copyright (c) 2006 Sean McCreary <mccreary@mcwest.org>. All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 *
 * 3. The name of the author may not be used to endorse or promote products
 * derived from this software without specific prior written permission
 *
 * THIS SOFTWARE IS PROVIDED ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
 * THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/* Pretty printf() wrapper for MPI processes */

#include <stdio.h>
#include <stdarg.h>
#include <string.h>

#define PP_MAX_BANNER_LEN       14
#define PP_MAX_LINE_LEN         81
#define PP_PREFIX_LEN           27
#define PP_FORMAT               "[%3d:%03d] %-14s : "

static int pid = -1;
static int msgcount = 0;
static char banner[PP_MAX_BANNER_LEN] = "";
static char oldbanner[PP_MAX_BANNER_LEN] = "";

int init_pprintf(int);
int pp_set_banner(char *);
int pp_reset_banner();
int pprintf(char *, ...);

int init_pprintf( int my_rank )
{
    pp_set_banner("init_pprintf");
    pid = my_rank;
/*
    pprintf("PID is %d\n", pid);
*/
    return 0;
}

int pp_set_banner( char *newbanner )
{
    strncpy(oldbanner, banner, PP_MAX_BANNER_LEN);
    strncpy(banner, newbanner, PP_MAX_BANNER_LEN);
    return 0;
}

int pp_reset_banner()
{
    strncpy(banner, oldbanner, PP_MAX_BANNER_LEN);
    return 0;
}

int pprintf( char *format, ... )
{
    va_list ap;
    char output_line[PP_MAX_LINE_LEN];

    /* Construct prefix */
    snprintf(output_line, PP_PREFIX_LEN+1, PP_FORMAT, pid, msgcount, banner);

    va_start(ap, format);
    vsnprintf(output_line + PP_PREFIX_LEN,
              PP_MAX_LINE_LEN - PP_PREFIX_LEN, format, ap);
    va_end(ap);

    printf("%s", output_line);
    fflush(stdout);
    msgcount++;
    return 0;
}
```