Comparison of MPI and GA on Glenn1 and Glenn2

Final Project Report

CSE – 721

Introduction to Parallel Computing

Karthik Raj Saanthalingam

Abstract:

The project's aim is to compare the Message passing model - MPI and the Partitioned Global Address Space model – Global Arrays, using benchmarks such as Gaussian solvers, Matrix solvers etc. The project aims at contrasting the difference between the coarse grained parallelism approaches in terms of communication pattern, range of applications, scalability, interoperability with other libraries etc. The differences can be highlighted by identifying the bottlenecks in the benchmarks that are more suitable to be implemented in one model than the other.

MPI VS GA

Message Passing Interface

Message Passing Interface (MPI) is an API specification that allows processes to communicate with one another by sending and receiving messages. Data is explicitly associated with each processor, accessing data requires specifying the location of the data on the processor and the processor itself. The message passing model is typically used for parallel programs running on clusters where cost of accessing non-local memory is high and can be applied to a wide range of data structures. Data locality is explicit but data access is complicated.

Message Passing requires cooperation from the processors on both sides. The processor sending the message and the processor receiving the message must both participate, thus enforcing an implicit synchronization. Alternatively one-sided get/put data access from/to local data at remote process is also possible. The explicit partitioning of data ensures that the communication overhead between processors can be assessed easily and can be minimized to achieve higher performance.

Partitioned Global Address Space - Global Arrays

Distributed dense arrays that can be accessed through a shared memory-like style i.e. it provides an abstraction of a shared memory model for physically distributed data. PGAS model can be applied only to arrays and it gains by exploiting the 'locality of reference'. These differences become vivid from the performance analysis of the benchmarks, the implicit/explicit bottlenecks introduced by the parallel programming models, communication and memory overheads etc.

One sided communication typically involves message transfer to be initiated on sending processor and the sending processor can continue computation. Receiving processor is not involved. Data is copied directly from switch into memory of the recipient processor. This mandates that data consistency must be managed explicitly. Global view of data ensures higher productivity and local compute can be used to assess the load sharing. The disadvantage of global arrays is that it can be used only for array data structures.

Glenn1 VS Glenn2

Glenn1:

Glenn1 comprises of four processors per node with 8 GB of shared memory i.e. 2 GB per processor. The nodes directly connect to an intermediate switching node. Many such intermediate switching nodes are connected to a root switching node. The switching nodes are organized like a 'semi-fat' tree, with n/2 links between an intermediate switching node and the root switching node. Thus the diameter of the cluster is 4 and the maximum contention per link is 2.

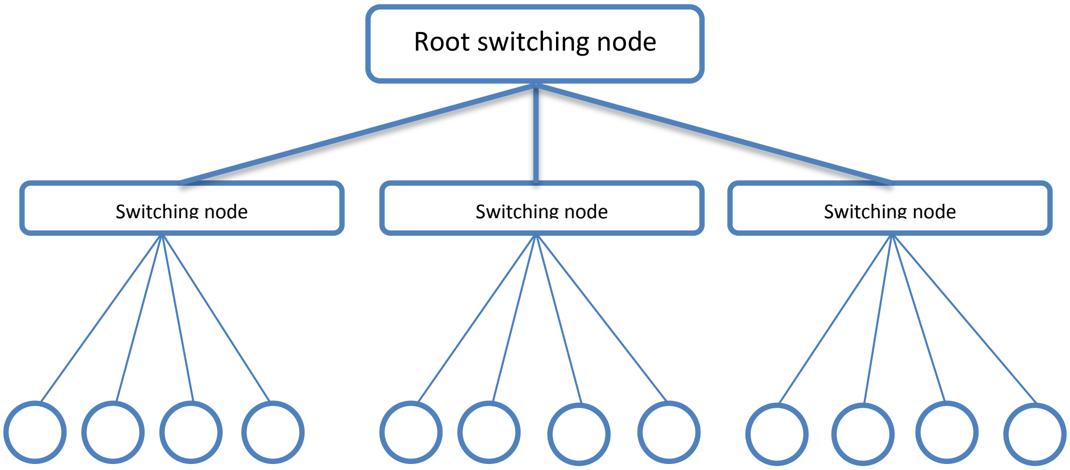


Fig 1: Glenn1 cluster

Glenn2:

Glenn1 comprises of eight processors per node with 24 GB of shared memory i.e. 3 GB per processor. The nodes connect to an intermediate switching node; the intermediate switching nodes connect to a root switching node to form a semi-fat tree as in the case of Glenn1. The diameter and contention factor remains the same.

Since the number of processors per node and shared memory are higher in Glenn2, it can be expected to perform better than Glenn1 in most cases as intra-node communication is faster as any processor within a node is at a distance of 1 and can cooperate through shared memory.

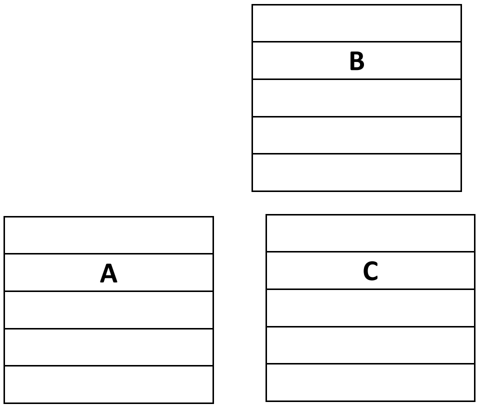
Inter-node communication involves communication with processors at a distance of four which is four times higher than intra-node communication and is also subject to contention in the network depending on the number of processors in a node that are communicating across the network.

Since the performance of multi-node systems consisting of identical processing nodes significantly depends on the amount of inter-node communication, the single node profiles, amount of communication being the bottleneck, Glenn2 should have a better processing time given the higher availability of shared memory and lesser inter-node communication given a uniform communication pattern between processors in a given parallel application.

Benchmarks:

1. **Matrix multiplication**
   MPI implementation using asynchronous communication:

   The input and output matrices have a row-wise block partition among the P processors. A master process divides the work and distributes it to all the other processes and accumulates the results from the other processes in an asynchronous fashion.



The computation begins with a one-to-all broadcast of input matrix B to all processes. There is no more communication between the processors and they perform the computation locally and transfer the results back to the master processor as a one-one communication.

The amount of communication required by the program is $3*N^2$. i.e.

N^2 - for broadcasting array B

N^2 - for sending out each interval of rows ($P*(N^2/P) = N^2$)

N^2 - for receiving the results back from the worker processors.

The number of calculations required by the algorithm is N^3.

Communication – computation ratio = $T_{COMM}*3*N^2 / T_{CALC}*N^3$.

As N approaches infinity, we get ZERO. This implies that the communication overhead reduces significantly with increase in problem size.

Let the ratio T_{COMM}/T_{CALC} is 1. Then, taking into account the cost of communication, the number of operations (both calculations and communications) performed by each of the P processors is

$N^3/P + N^2 + 2N^2/P$

Where, N^3/P – computation cost, $N^2 + 2N^2/P$ - communication cost

The cost of communication is the cost of receiving the broadcast plus the costs of receiving and sending the rows it's responsible for. If we compare this value with the conventional cost of computation in the case of a single processor (N^3), we get:

$$(3N^3 + (P+2)*N^2) / \quad P*N^3$$

The limit as N approaches infinity is 1/P i.e. if the overhead of communication can be factored out, then as the size of the matrix increases then using P processors will speed up the problem by P times.

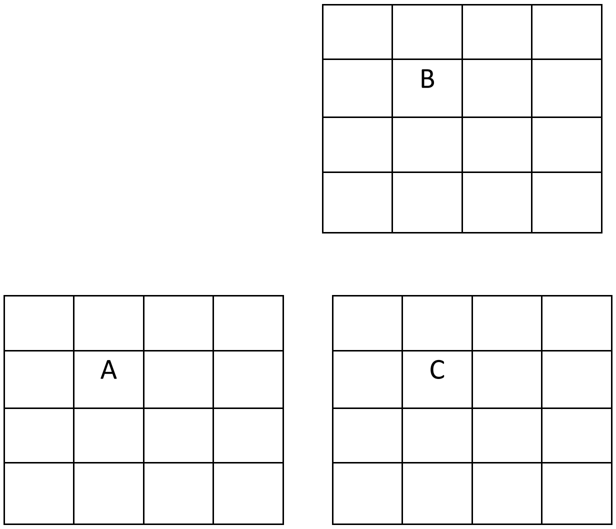
Let us consider the addition of two matrices. This algorithm is only N^2. The ratio of communication over computation on P processors is

$$3N^2/ N^2 = 3$$

This implies that the parallel implementation of the algorithm is efficient if and only if T_{COMM}/T_{CALC} is less than 1/3.

Global Arrays implementation using get-compute-put model:

The input and output matrices are transformed into global arrays with 2D block partitioning being carried out using regular distribution model. The master process initializes the input matrices and other processes are synchronized to wait till the data is copied to the global array.



The processes 'get' the corresponding row and columns from the global arrays of A and B to compute the block in C that they own. Since this involves one-sided 'get' calls no synchronization is required. The results are copied back to global array C using the 'put' call.

The following are the results of the two different runs on Glenn1 and Glenn2.

	Execution Time (sec.)			
	Glenn 1		Glenn 2	
Processes	MPI	GA	MPI	GA
1	2.630	2.610	1.940	2.070
2	1.330	2.600	1.080	2.050
4	0.705	1.460	0.593	0.587
8	0.379	0.617	0.311	0.226
16	0.234	0.257	0.220	0.094
32	0.179	0.105	0.171	0.046
64	0.227	0.043	0.173	0.024

	Execution Time (sec.)			
	Glenn 1		Glenn 2	
Processes	MPI	GA	MPI	GA
1	2.180	2.000	1.910	2.010
2	1.150	2.029	1.110	2.040
4	0.588	0.695	0.588	0.637
8	0.309	0.224	0.309	0.225
16	0.181	0.094	0.220	0.095
32	0.136	0.044	0.148	0.049
64	0.185	0.022	0.173	0.026

	Speedup*			
	Glenn 1		Glenn 2	
Processes	MPI	GA	MPI	GA
1	1.0	1.0	1.0	1.0
2	2.0	1.0	1.8	1.0
4	3.7	1.8	3.3	3.5
8	6.9	4.2	6.2	9.2
16	11.2	10.2	8.8	22.0
32	14.7	24.9	11.3	45.0
64	11.6	60.7	11.2	86.3

	Speedup*			
	Glenn 1		Glenn 2	
Processes	MPI	GA	MPI	GA
1	1.0	1.0	1.0	1.0
2	1.9	1.0	1.7	1.0
4	3.7	2.9	3.2	3.2
8	7.1	8.9	6.2	8.9
16	12.0	21.3	8.7	21.2
32	16.0	45.5	12.9	41.0
64	11.8	90.9	11.0	77.3

	Efficiency			
	Glenn 1		Glenn 2	
Processes	MPI	GA	MPI	GA
1	100%	100%	100%	100%
2	99%	50%	90%	50%
4	93%	45%	82%	88%
8	87%	53%	78%	114%
16	70%	63%	55%	138%
32	46%	78%	35%	141%
64	18%	95%	18%	135%

	Efficiency			
	Glenn 1		Glenn 2	
Processes	MPI	GA	MPI	GA
1	100%	100%	100%	100%
2	95%	49%	86%	49%
4	93%	72%	81%	79%
8	88%	112%	77%	112%
16	75%	133%	54%	132%
32	50%	142%	40%	128%
64	18%	142%	17%	121%

	Relative Performance			
	GA/MPI		Glenn 2/Glenn 1	
Processes	Glenn 1	Glenn 2	MPI	GA
1	99%	107%	74%	79%
2	195%	190%	81%	79%
4	207%	99%	84%	40%
8	163%	73%	82%	37%
16	110%	43%	94%	37%
32	59%	27%	96%	44%
64	19%	14%	76%	56%

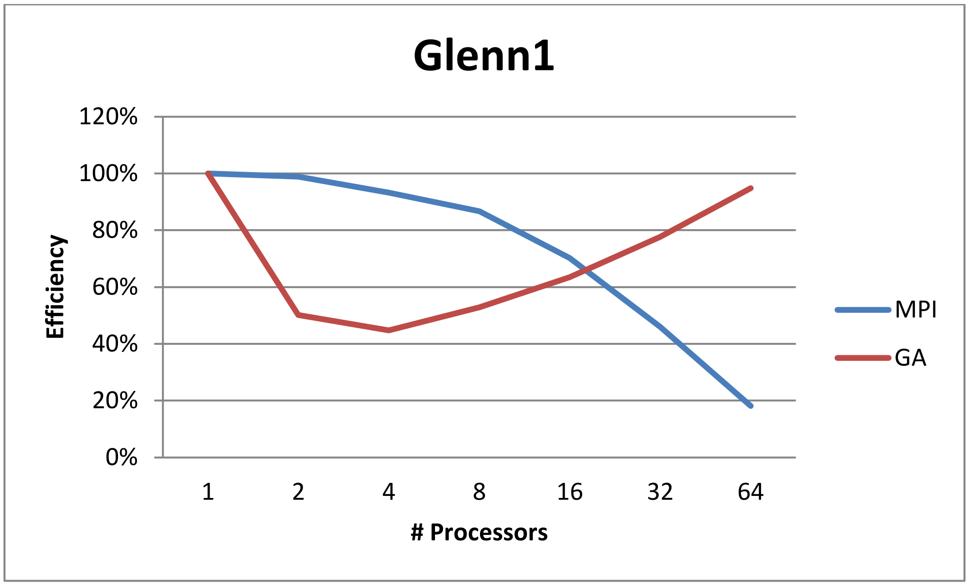
	Relative Performance			
	GA/MPI		Glenn 2/Glenn 1	
Processes	Glenn 1	Glenn 2	MPI	GA
1	92%	105%	88%	101%
2	176%	184%	97%	101%
4	118%	108%	100%	92%
8	72%	73%	100%	100%
16	52%	43%	122%	101%
32	32%	33%	109%	111%
64	12%	15%	94%	118%

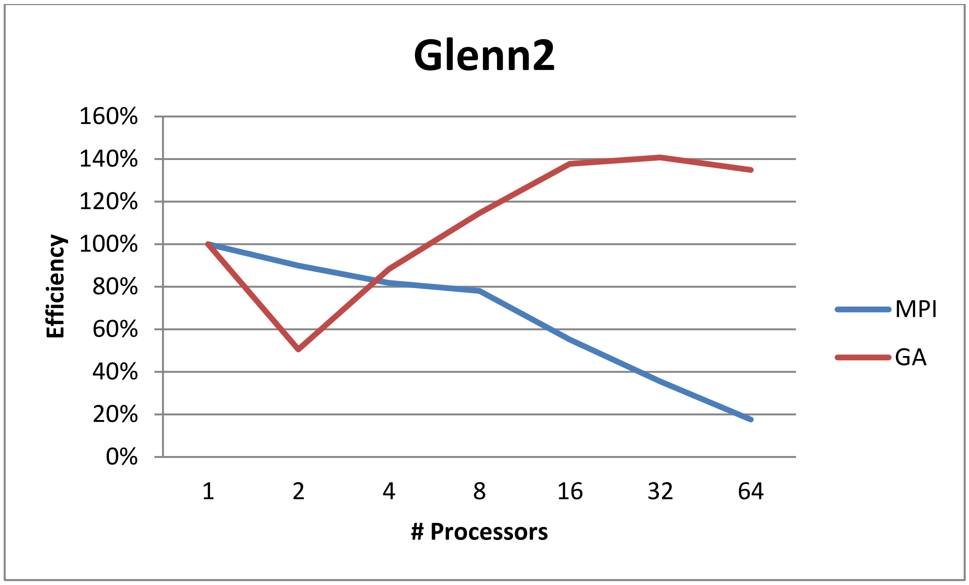
The size of the matrices were fixed at 1000 and the applications were run on identical nodes.

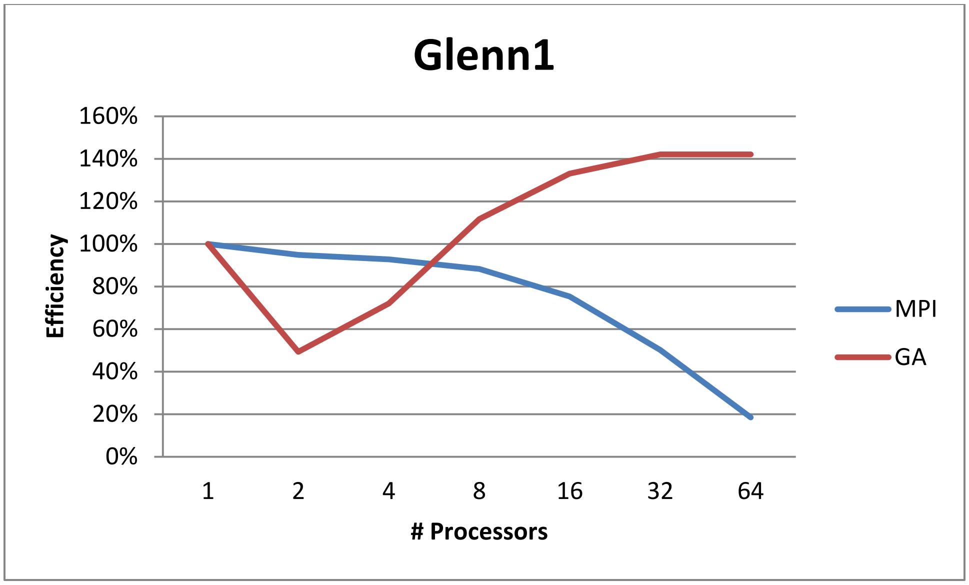
Speedup =Run time on single processor/Parallel runtime

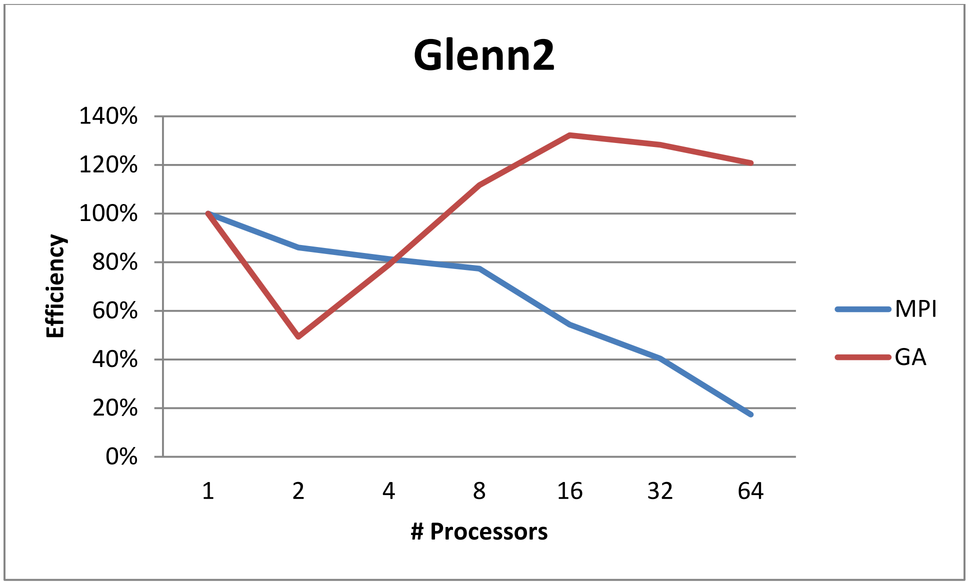
Efficiency =Speedup/ # Processors

Efficiency plots

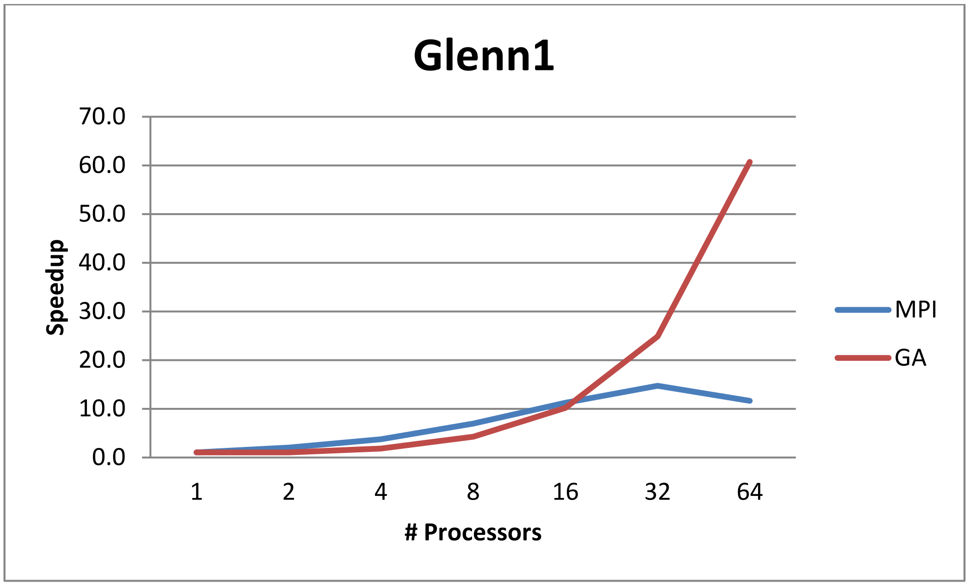


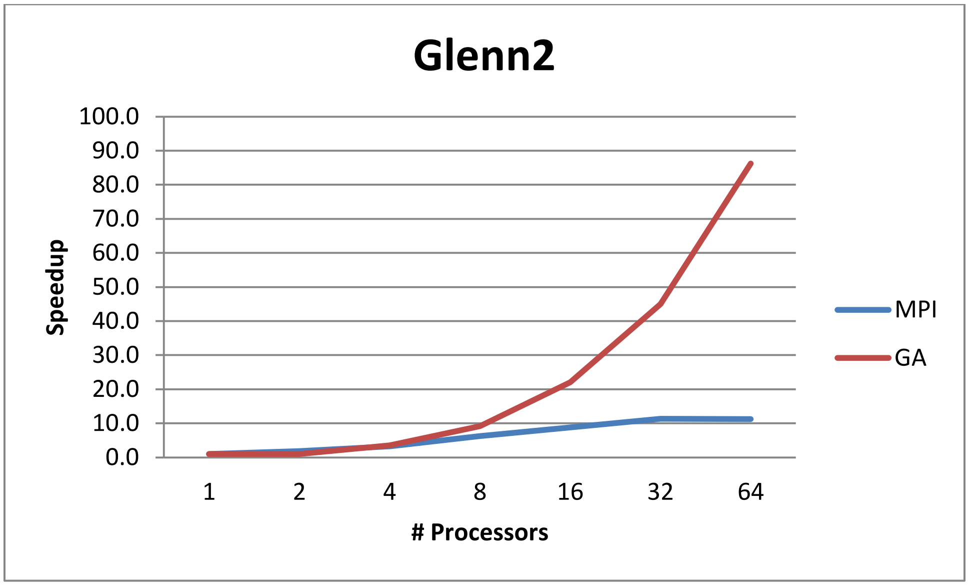


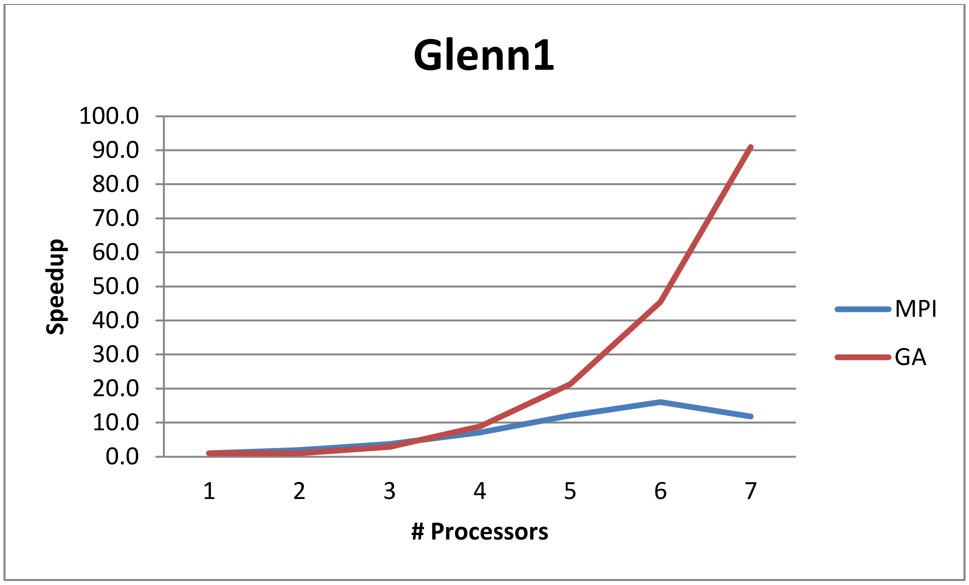

Glenn1
Efficiency
160%
140%
120%
100%
80%
60%
40%
20%
0%
1
2
4
8
16
32
64
# Processors
MPI
GA

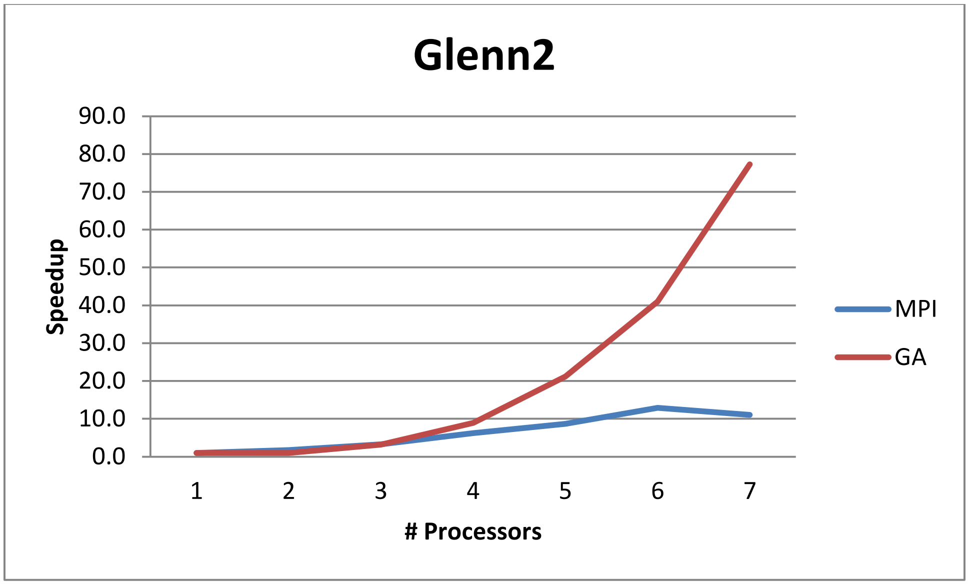

Glenn2
Efficiency
140%
120%
100%
80%
60%
40%
20%
0%
1
2
4
8
16
32
64
# Processors
MPI
GA

Speedup plots









Inferences:

For the fixed problem size MPI model was not able to maintain the efficiency and the efficiency dropped due to the increase in communication overhead with the increase in the number of processors. The efficiency remains relatively constant for intra-node communication i.e. Four processors on Glenn1 where as it dips in case on Glenn2. This can be attributed to the fact that the runtime on a single processor in case of Glenn2 is faster than the one on Glenn1 due to the availability of a larger cache and memory. The communication overhead that is incurred is not balanced by the cost incurred in parallel computation.

The one sided communication model requires lesser overhead for a fixed problem size as there is no explicit synchronization between the processes. The ‘dip’ in efficiency for lower number of processors can be attributed to the fact that the communication overhead dominates the performance achieved through the parallel computation but the cost is compensated as the parallel computation time decreases with the increase in the number of processors.

Conclusion:

In general, the performance of the one-sided ‘get-put’ model proved to be more efficient on both the clusters. The efficiency and speedup achieved in higher in Glenn2 than in Glenn1 due to the higher incidence of intra-node communication and a higher availability of shared memory and cache. The higher costs incurred with the two sided communication model is due to the relatively lower size of the problem, thus the parallel computation cost could not account to the extra overhead accounted for due to communication . The efficiency could be increased by increasing the problem size at a higher rate than that required by the one sided communication model.