Basic MPI Optimizations

MPI Derived Types (FAILED)

Given the **AoS** structure of the work grid, a MPI *cell_type* struct that resembles the C *t_speed* struct was introduced.
The appeal of MPI_Datatypes is the ability to pack non-contiguous data, in making best use of the bandwidth given the constant latency induced by communication. In hindsight, given the data to be communicated (halos/edges) were contiguous in memory, this was an unseen pitfall given by the additional overhead in packing/unpacking to/from C and MPI struct types.

MPI Reduction Operations (SIGNIFICANT SUCCESS)

Each rank computed its partial *av_vels* for the iteration space (*maxIters*). Subsequently after the timestep loop the master performs a MPI_SUM reduction to obtain the real average velocities. This is comparatively much faster than if at each iteration of the timestep loop, the grid was gathered on master and the average velocity of that iteration computed.
Each rank is allowed to work independent(apart from the necessary halo exchange).
Reductions are tree-based as compared with gathers and so scale well O(log_2 n) with increasing ranks.

RUNTIMES SO FAR

N	128x128	256x256	1028x1028
1 Node	4.79s	12.72s	51.60s
2 Nodes	4.42s	35.02s	33.03s
4 Nodes	4.48s	32.91s	24.16s

Additional MPI Optimizations

MPI File IO Operations (FAILED)

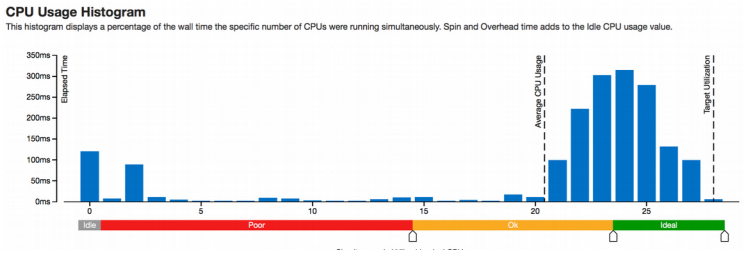
Previously implemented as a scatter/gather from/to the master rank and following read/write operations were replaced by parallel MPI FILE-IO operations. Given that individual ranks can independently read/write from opened files, any overhead of communication and synchronization is escaped. Since this wasn't a major source of overhead (given profiler statistics) as compared with the timestep loop, no distinguishable improvement was seen.

MPI Neighbour operations (FAILURE)

MPI 3.0 introduced MPI_Graph and MPI_Neighbour collectives that define a topologies or communication graphs and protocols of communication based on them. We replaced previous MPI_Sendrecv point-to-point communications with a single MPI_Neighbour collective call. No improvements were seen.

Fair Allocation (SIGNIFICANT SUCCESS)

Previously each rank was given the same amount of rows, any additional rows would be given to the last rank. This created load imbalance, slowing the execution of each iteration to the speed of the slowest rank. Instead round robin allocation of grid rows was introduced for fair allocation.



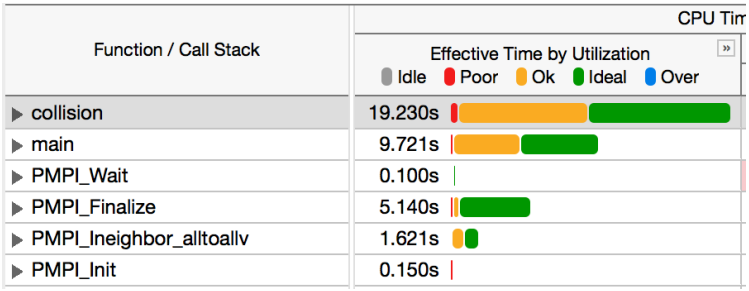
As seen above load imbalance is still (and always) present but significantly reduced.

Serial Optimizations

Remove Redundancy (MARGINAL SUCCESS)

The *collision* routine was at each iteration computing constants using a division.
These inverse constants were collected as C MACROS (escaping expensive redundant computations). We then could replace divisions in the collision routines by multiplications by these constants. Given the memory bandwidth bound nature of the CPU, this provided only a marginal improvement.

Loop Fusion (SIGNIFICANT SUCCESS)



With Profiler guidance *Collision* was located as the execution hotspot. The fusion of *Propagation*, *Rebound*, *Collision* and *Average-Velocity* allowed the removal of redundant computations present in *Average-Velocity* previously performed during *Collision*. Noticeable improvements were recorded.

N	128x128	256x256	1028x1028
1 Node	+0.35s	-1.17s	-4.21s
2 Nodes	+0.55s	-0.51s	-1.77s
4 Nodes	-0.47s	-0.00s	-1.11s

Register/Cache Memory (FAILED)

An intermediate buffer (located in local memory) was being copied into at propagation before final assignment at either rebound or collision. This was replaced with a single local t_speed variable that held the intermediate cell value after propagate. No improvement in speed was noted. Most notably due to the memory location and associated cache line being cached during assignment.

RUNTIMES SO FAR

N	128x128	256x256	1028x1028
1 Node	1.12s	7.19s	28.7s
2 Nodes	1.04s	4.47s	10.2s
4 Nodes	1.03s	3.77s	8.51s

Structure of Arrays (SOA)

The grid was transformed in SoA, previous compiler detected dependencies under AoS that disallowed vectorization did not exist. Sadly, vectorization currently was poor due to non-linearly computed indexed loads at propagate and the masked vectorization operations as a consequence of the rebounds-collision conditional. Consequently, smaller grids suffer from larger runtimes due to the lack of spatial locality of SoA as opposed to AoS. As the grid and thus inner-loop trip-count increases the speedup of vectorization outweighed the benefits of caching as seen in lower runtimes for 1024x1024 across all node counts.

N	128x128	256x256	1028x1028
1 Node	+0.51s	-0.55s	-1.5s
2 Nodes	+0.75s	+0.12s	-0.7s
4 Nodes	+1.5s	+0.75s	+0.16s

Not Masked Vector Operations (FAILED)

Basic vectorization preliminaries were made including alignment of memory and prefixing of pointers are restricted.

The rebound-collision conditional was replaced by a clever multiplication by the mask obstacle[cell_index], removing the need for masked vector operations. As per vectorization report a 2.2x speedup was estimated.

N	128x128	256x256	1028x1028
1 Node	+0.37s	-0.19s	-10.2s
2 Nodes	-2.5s	-2.2s	-3.9s
4 Nodes	+3.4s	+4.3s	-2.3s

Masked Vectorization(SIGNIFICANT)

As compared to before the conditional was removed but replaced by a mask during assignment to the final grid.
Ie.

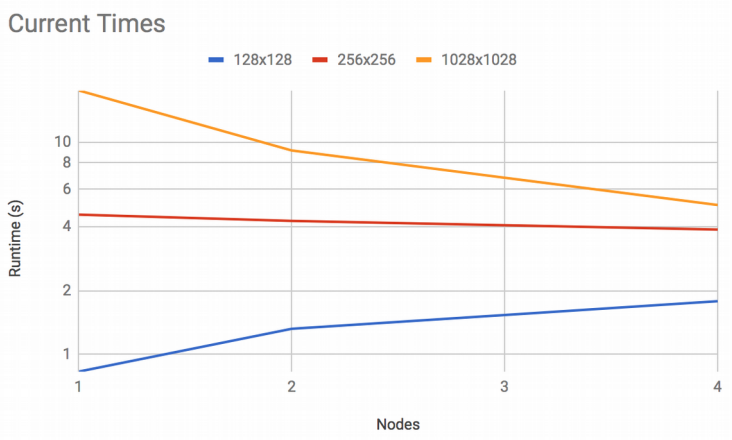
```
final_cells[cell_index] =
    mask ? rebound_cells[cell_index]
         : collision_cells[cell_index]
```

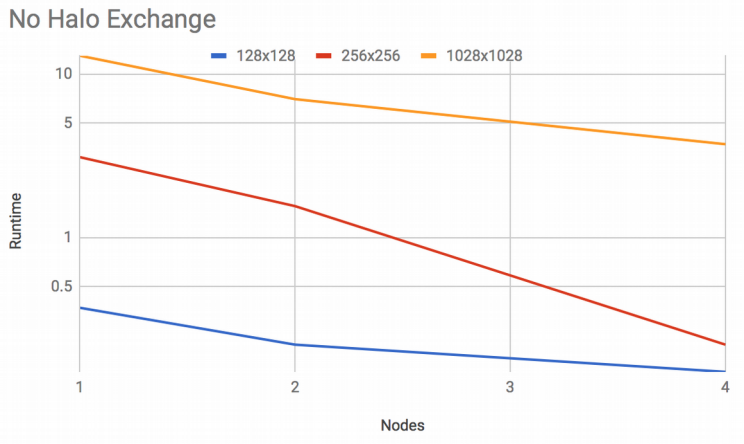
This provides the explicit advantage that the final value is conditionally loaded from memory as opposed to always loaded, as previously, when we loaded both values and multiplied by the mask. Distinct improvements were seen given the memory bandwidth bound nature of the CPU.

N	128x128	256x256	1028x1028
1 Node	-0.11s	-2.5s	-10.5s
2 Nodes	-2.5s	-3.2s	-4.1s
4 Nodes	-2.5s	-3.8s	+2.3s

Reduced MPI Communication

The current bottleneck was found to be the communication time between ranks.





As seen in the *current times*, the cost of communication dampened the scalability of higher node runs. In removing the halo exchange the runtimes experienced a linear speedup with increasing node counts.

In maintaining correctness of the program a different strategy was used.
In sending the top and bottom halos, we are only required in sending the top and bottom speeds respectively. 1/3rd the data is only sent. Communication of data less than the bandwidth still incurs the cost of communication latency, but in the case of large data exchanges (as in 1024x1024) benefits are noticeable.

N	128x128	256x256	1028x1028
1 Node	-0.78s	-1.58s	-1.2s
2 Nodes	-1.3s	-6.8s	-2.1s
4 Nodes	-2.6s	-3.4s	-1.9s

Potential Further Optimizations

Packing using MPI_VectorType

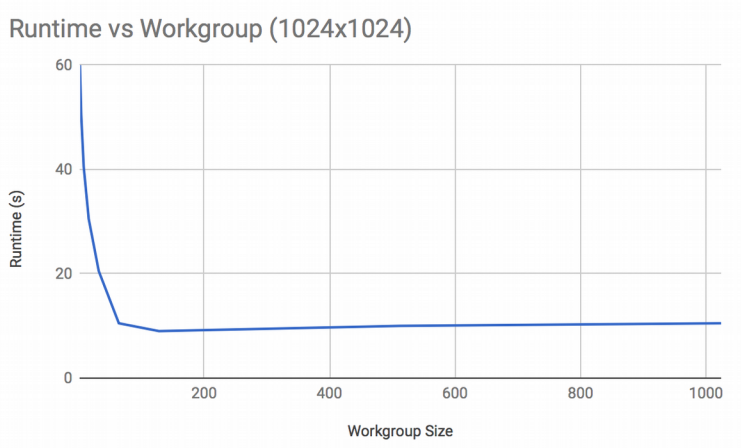
In communication of the 3 speeds for top/bottom edges/halos during halo exchange. 3 synchronized communications are made, each for a given speed of the cell of a edge/halo.
A vector MPI_Datatype could be used to pack the 3 speeds into a single transmission.
Providing improvements especially at lower grid sizes when bandwidth becomes a non-issue.

Basic OpenCL Optimizations

Initial Times

Host buffers are copied into device buffers during initialisation of device context.
Further-after no CPU-GPU communication is present apart from copy of halo/edges back and forth each iteration. Each work-item computes the final value of its specific cell.

Workgroup sizes are analogous to OpenMP blocked iteration scheduling. Given a small workgroup size significant overhead is incurred in managing/scheduling workgroups. Regardless
fairly fine grain tasks are best to achieve load balancing between compute units of the GPU.



As an ideal grain size, 128 work-items per workgroup was chosen. A multiple of 32 is best chosen st. each 32 chunk of a workgroup can be mapped to a CPU hardware thread or CUDA warp.

Workgroups compute their partial average velocity which are written to a buffer on the device for each iteration. The host reads this buffer and computes the final average velocities after the timestep loop.

N	128x128	256x256	1028x1028
1 Node	2.56s	9.58s	20.76s
2 Nodes	2.60s	7.72s	12.2s
4 Nodes	3.04s	8.19s	8.47s

Data Map (FAILED)

clEnqueueMapBuffer slowed runtimes as mapping allocated memory each invocation, as opposed to copying to an existing host buffer with *clEnqueueReadBuffer*.

Private Memory (SIGNIFICANT SUCCESS)

The removal of the intermediate buffer provided a great speedup as compared to the CPU version.
Given work-items are executed in parallel. GPU Compute-Unit caches sizes are not large enough to store the intermediate cell value of all work-items of its workgroup.

N	128x128	256x256	1028x1028
1 Node	-0.64s	-4.05s	-22.2s
2 Nodes	-0.37s	-2.81s	-14.75s
4 Nodes	-0.1s	-0.6s	-2.1s

Structure of Arrays (SIGNIFICANT SUCCESS)

Given the SIMD execution of GPU warps. In leveraging the hardware CMTs (coalesced memory transactions), the grid was transformed into SOA. The aligned, unit-stride memory access pattern of adjacent work-items require significantly CMTs in reading all data required by work-items of the warp to continue execution. The irregular memory access pattern of propagate given the nonlinear computation of the indices,

```
int x_e = (ii + 1) % comm_info->local_x;
int x_w = (ii - 1) % comm_info->local_x;
```

may generates a sub-optimal number of CMTs or generate a higher number of aborts due to conflicting CMTs given that CMT instructions are generated at compile time under static analysis and not dynamically when the memory access happens.

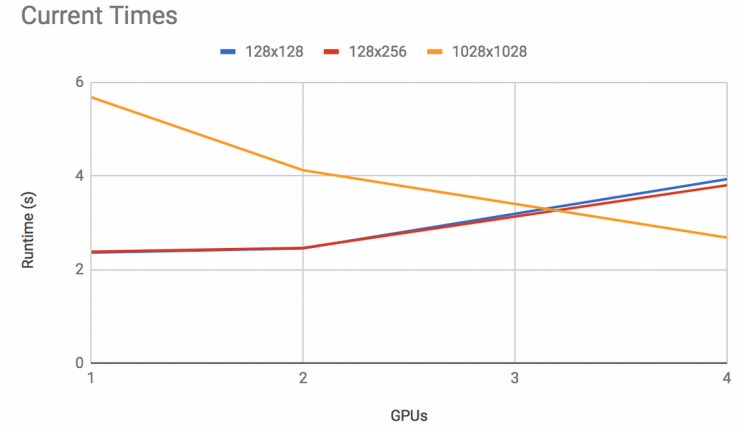
N	128x128	256x256	1028x1028
1 Node	+0.32s	-1.7s	-6.7s
2 Nodes	-1.00s	-0.6s	-3.1s
4 Nodes	-3.1s	-5.2s	-1.1s

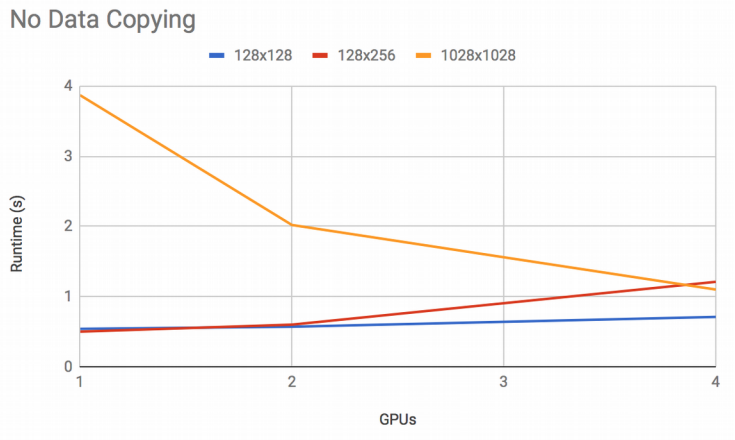
Masked Conditionals (FAILURE)

Given the agonizing useless branch predictors of GPU execution engines, branches generally invoke the performance penalty of pipeline stalls. In additional, given work-items of a warp execute in lockstep, diverging branches cannot be executed at the same time.
Given the small ratio of obstacles to non-obstacles and the largely differing workload between branches, the conditional performed better than computing both branches and masking the final store.

Reduced Data Copying (SIGNIFICANT SUCCESS)

As with the MPI Communication, CPU-GPU copying of only top/bottom speeds of top/bottom halo/edges respectively.





Removal of data copying between host and device, improved timings but not scaling (as seen above). Performance analysis brought to attention that data copying took constant time, emphasizing that latency, not bandwidth of the PCIe bus connection was the limiting factor.

N	128x128	256x256	1028x1028
1 Node	-0.52s	-0.88s	-1.5s
2 Nodes	-0.5s	-0.62s	-1.3s
4 Nodes	-0.2s	-0.40s	-0.8s

Tree-Based AV_VELS Reduction (FAILURE)

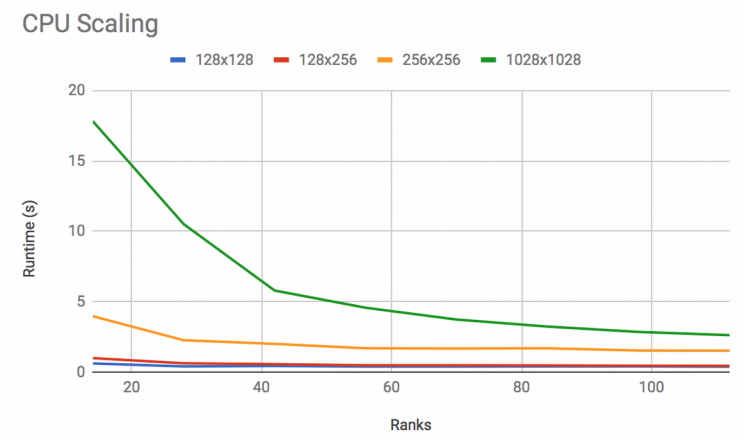
According to profiler statistics. The final reduction done by a single work item of the workgroup incurred negligible overhead. Likely since the array being summed over is locally allocated to the workgroup.

PRE-COMPUTATION (MARGINAL SUCCESS)

The acceleration kernel performed significantly less memory accesses as compared with the remainder of the timestep and given the high memory bandwidth of the GPU, the accelerate kernel was assumably compute bound. To optimise for this many constants were precomputed. Improvements were noted under all sizes and number of GPUs.

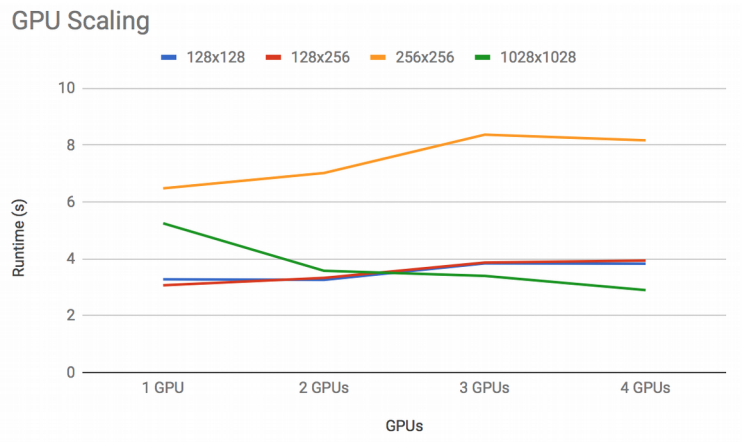
APPENDIX

FINAL TIMES (CPU)



Ranks	128x128	128x256	256x256	1028x1028
14	0.61	0.99	3.97	17.83
28	0.4	0.62	2.26	10.51
42	0.43	0.56	2	5.79
56	0.39	0.48	1.69	4.57
70	0.39	0.48	1.67	3.73
84	0.4	0.47	1.69	3.23
98	0.4	0.45	1.53	2.85
112	0.38	0.44	1.52	2.62

FINAL TIMES(GPU)



GPUs	128x128	128x256	256x256	1028x1028
1	3.28	3.07	6.48	5.25
2	3.26	3.33	7.02	3.58
3	3.84	3.87	8.37	3.4
4	3.83	3.94	8.17	2.95