Semi-Static and Dynamic Load Balancing for Asynchronous Hurricane Storm Surge Simulations

Max Bremer[1,2], John Bachan[1], and Cy Chan[1]
max@ices.utexas.edu

[1]Lawrence Berkeley National Laboratory [2]The University of Texas at Austin

PAW-ATM Workshop, Dallas,TX
November 16, 2018

INSTITUTE FOR COMPUTATIONAL ENGINEERING & SCIENCES

Problem Statement

Simulation of Hurricane Storm Surge

Motivation

- Hurricane simulation plays a key role in coordinating evacuation efforts.
- This is part of a project to port this application to a task-based programming model.
- We are interested in assess potential performance improvements *without yet having a task-based implementation.*

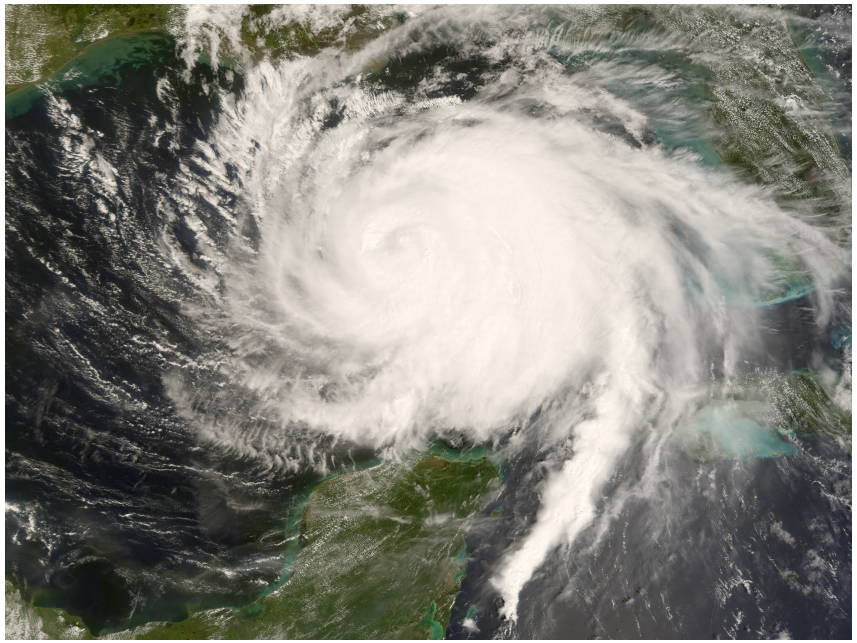


Figure: NASA image created by Jesse Allen, using data provided courtesy of the MODIS Rapid Response team.

Simulation of Hurricane Storm Surge

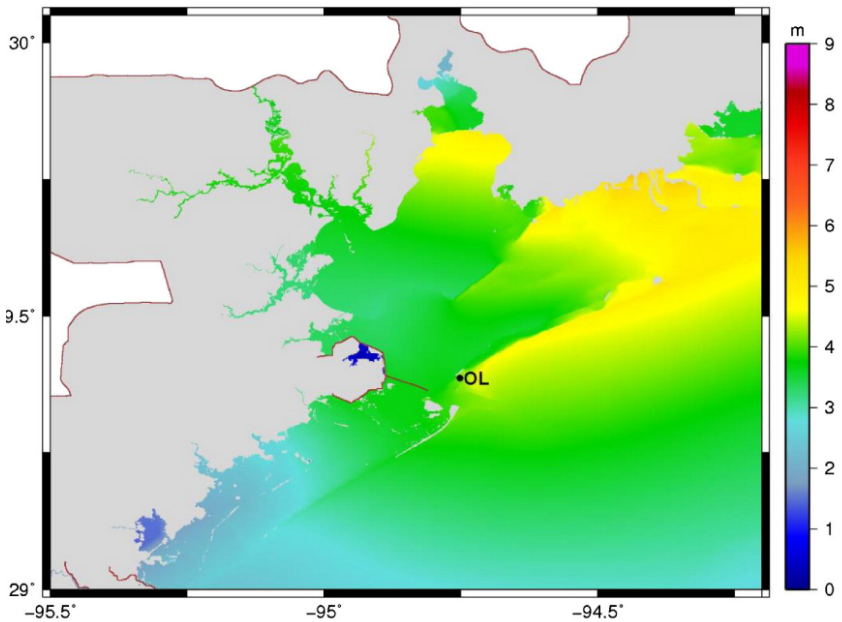


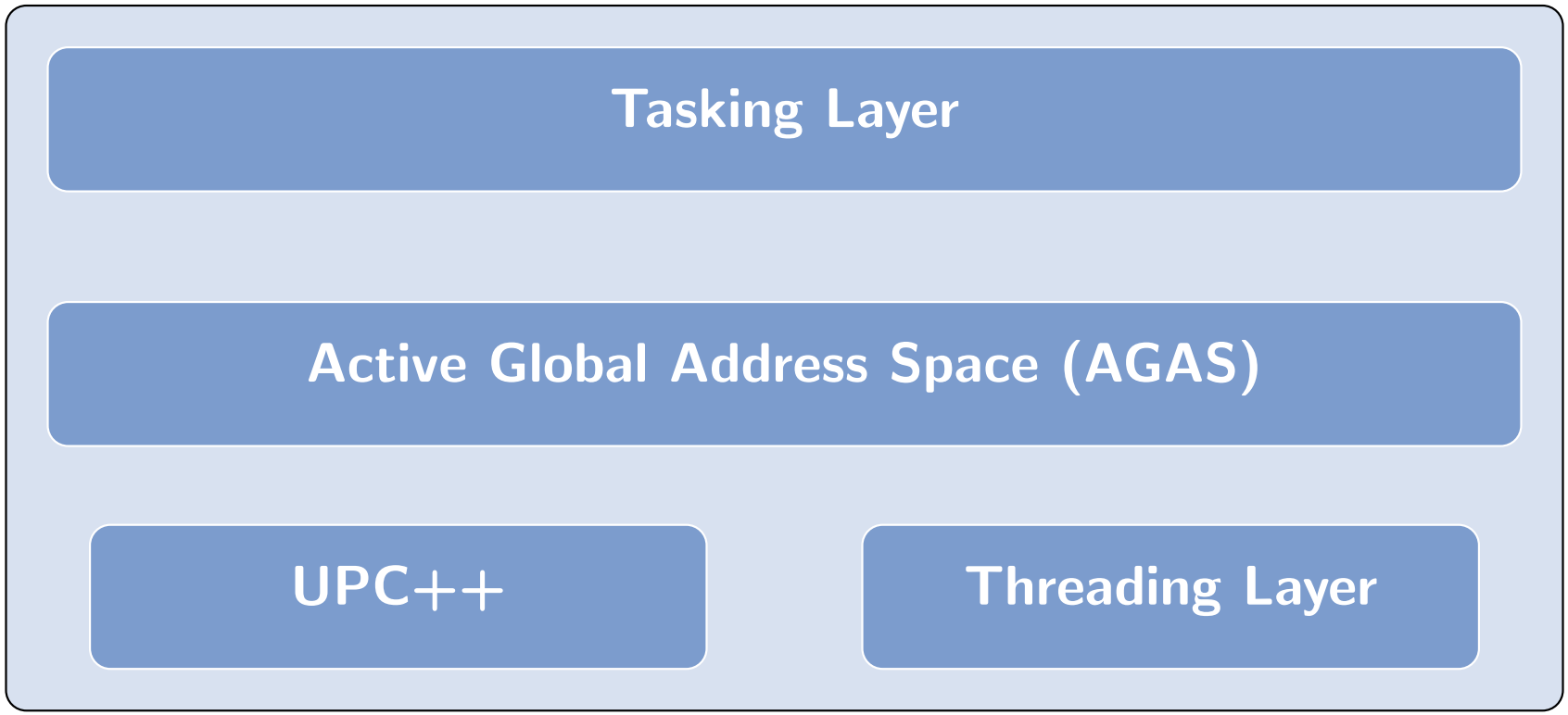
Figure: Surge profile generated by ADCIRC+SWAN for Hurricane Ike [Sebastian et al., 2014].

Computational kernel

- Simulate the ocean surface using the shallow water equations.
- Using a discontinuous Galerkin discretization; can be thought of as an explicit stencil kernel.
- The key source of irregularity is the computational difference between the simulation of wet and dry regions of the mesh.

Programming Model

Programming Model



Cost Model

Discrete Event Simulation—the Application

- Simulate storm using DGSWEM to determine wet/dry status of each element for a given element and timestep.
- Use of discrete simulation uses a factor of 5,200x fewer core hours.

Figure: Sample partitioning of mesh into *tiles*.

Discrete Event Simulation—the Hardware

- Intra-NUMA messages based on bandwidths with conservative contention estimates.
- Inter-NUMA messages use latency-bandwidth model for Cray Aries interconnect.
- Threading overheads measured via self-timing.

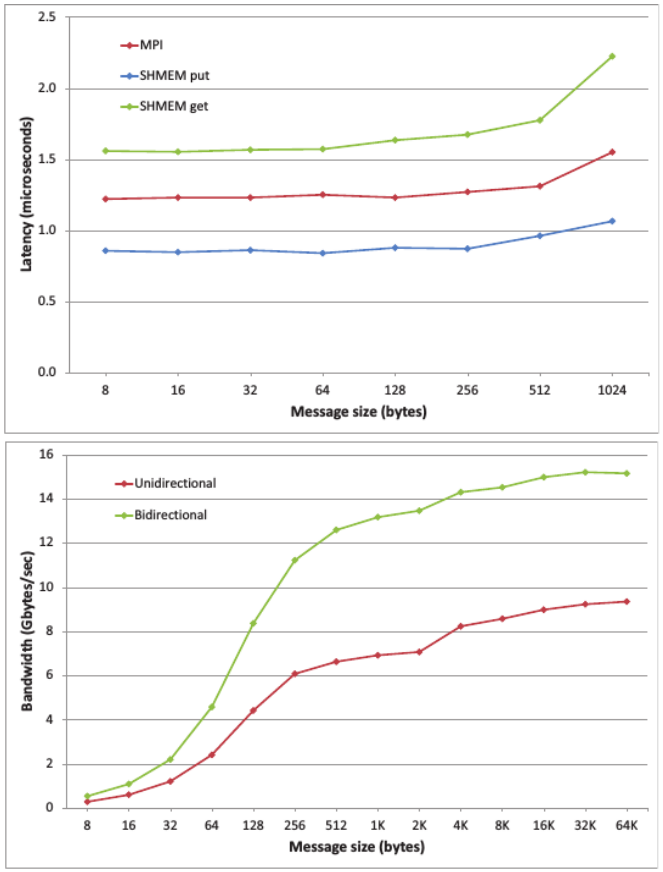


Figure: Latency (top) and bandwidth (bottom) for Cray Cascade [Faanes et al., 2012]

Load Balancing Algorithms

Static Approach

- Traditionally METIS is applied to the tile graph, distributing tiles across ranks.

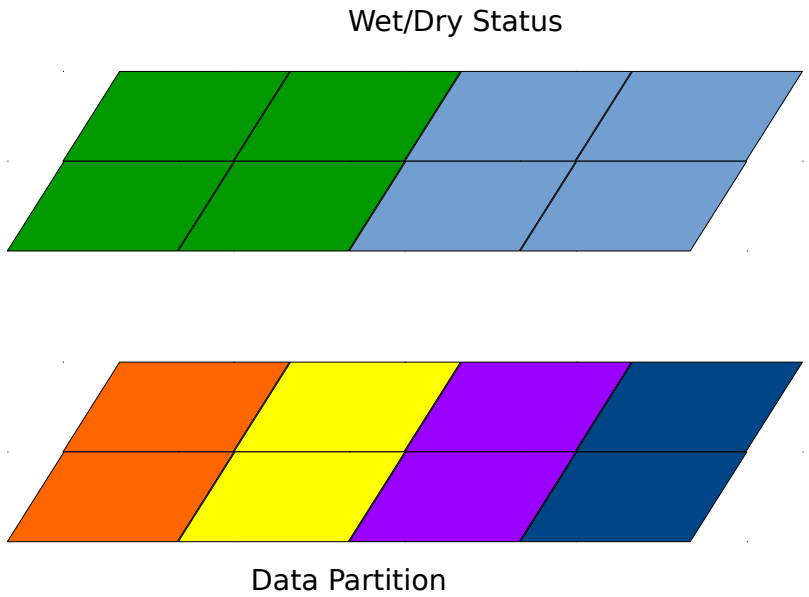


Figure: Sample tile graph with wet and dry. The data is partitioned onto 4 ranks: Rank 0, Rank 1, Rank 2, and Rank 3.

Static Approach

- Traditionally METIS is applied to the tile graph, distributing tiles across ranks.
- This leads to poor resource utilization. Half of the ranks are assigned no work.

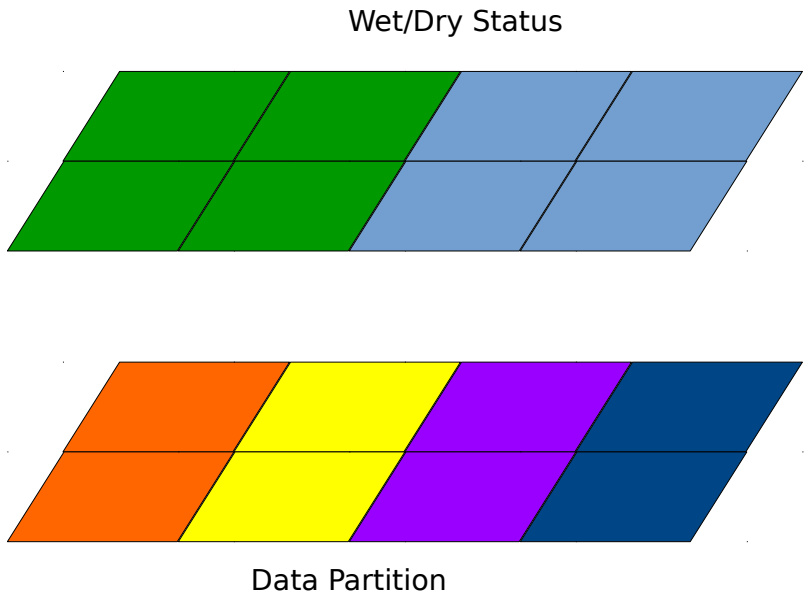


Figure: Sample tile graph with wet and dry. The data is partitioned onto 4 ranks: Rank 0, Rank 1, Rank 2, and Rank 3.

Static Approach

- Traditionally METIS is applied to the tile graph, distributing tiles across ranks.
- This leads to poor resource utilization. Half of the ranks are assigned no work.
- This partitioning strategy does not take into account the load balance associated with each rank.

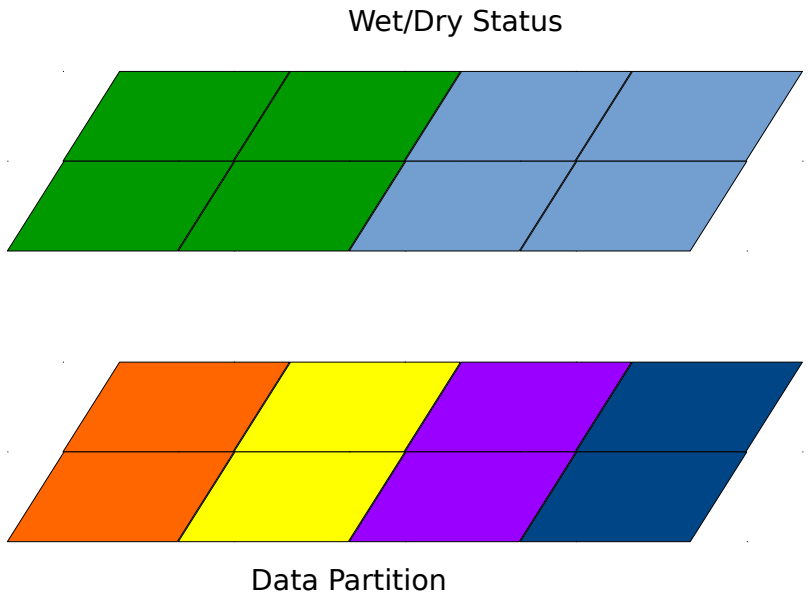


Figure: Sample tile graph with wet and dry. The data is partitioned onto 4 ranks: Rank 0, Rank 1, Rank 2, and Rank 3.

Multi-Constraint Static Approach

- We must balance wet and dry tiles simultaneously.

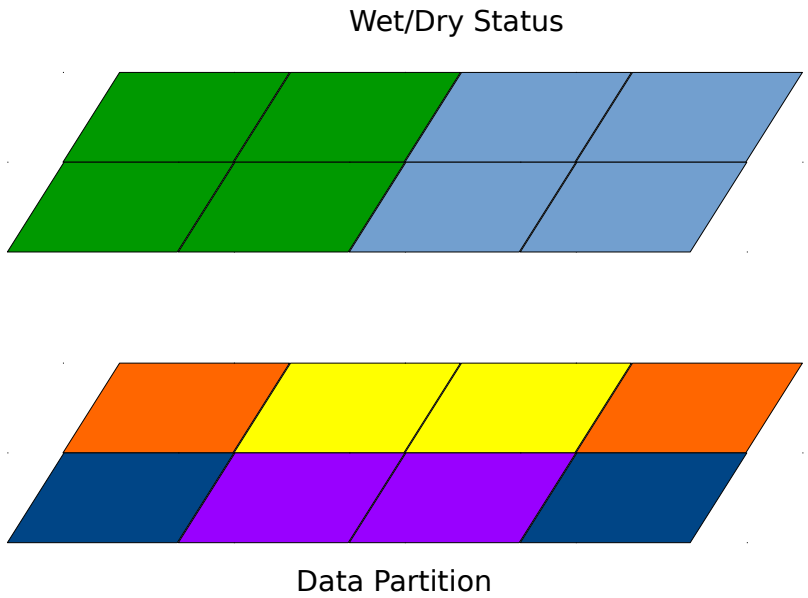


Figure: Sample tile graph with wet and dry. The data is partitioned onto 4 ranks: Rank 0, Rank 1, Rank 2, and Rank 3.

Multi-Constraint Static Approach

- We must balance wet and dry tiles simultaneously.
- METIS supports calls to multi-constraint graph partitioning.

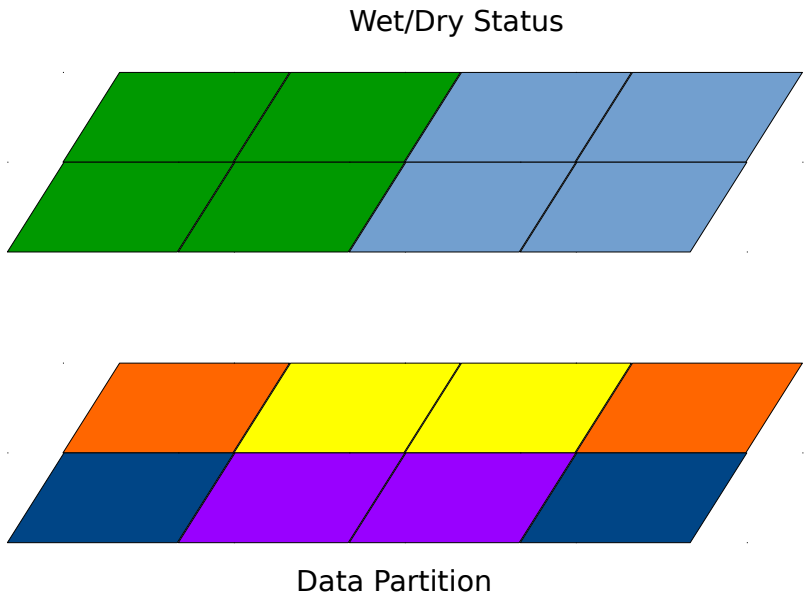


Figure: Sample tile graph with wet and dry. The data is partitioned onto 4 ranks: Rank 0, Rank 1, Rank 2, and Rank 3.

Semi-Static Load Balancing

- Periodically aggregate tile states and repartition *tile graph* using METIS.

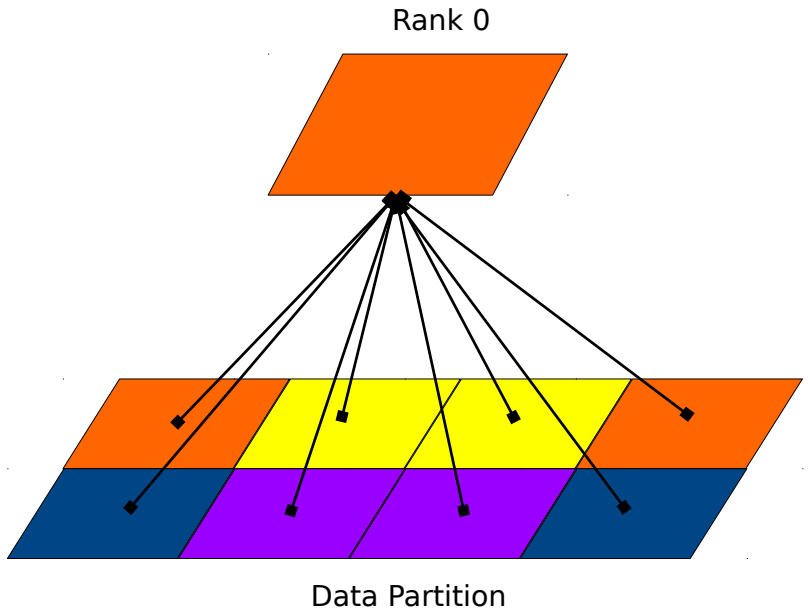


Figure: The data is partitioned onto 4 ranks: Rank 0, Rank 1, Rank 2, and Rank 3.

Semi-Static Load Balancing

- Periodically aggregate tile states and repartition *tile graph* using METIS.
- Repartitioning overlaps with application task execution.

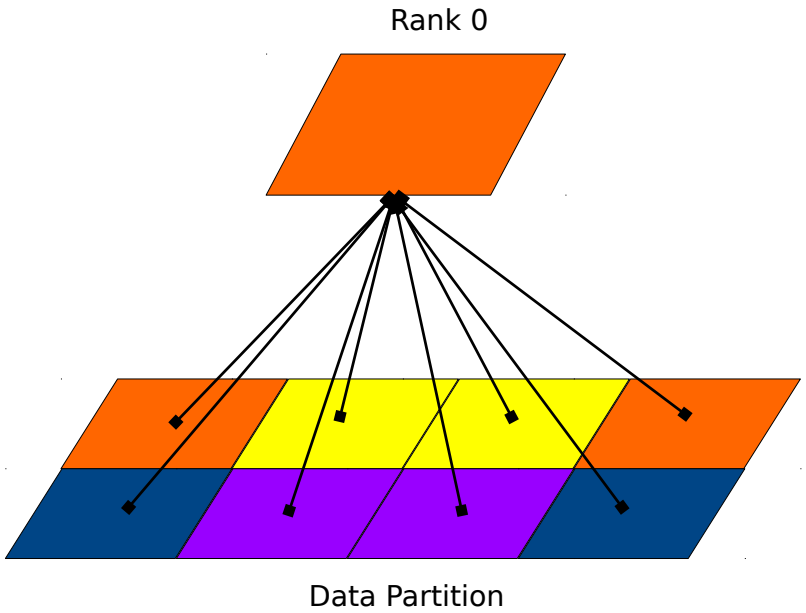


Figure: The data is partitioned onto 4 ranks: Rank 0, Rank 1, Rank 2, and Rank 3.

Semi-Static Load Balancing

- Periodically aggregate tile states and repartition *tile graph* using METIS.
- Repartitioning overlaps with application task execution.
- Rebalance frequency set to 5 s (every $\mathcal{O}(10^4)$ RK-stages).

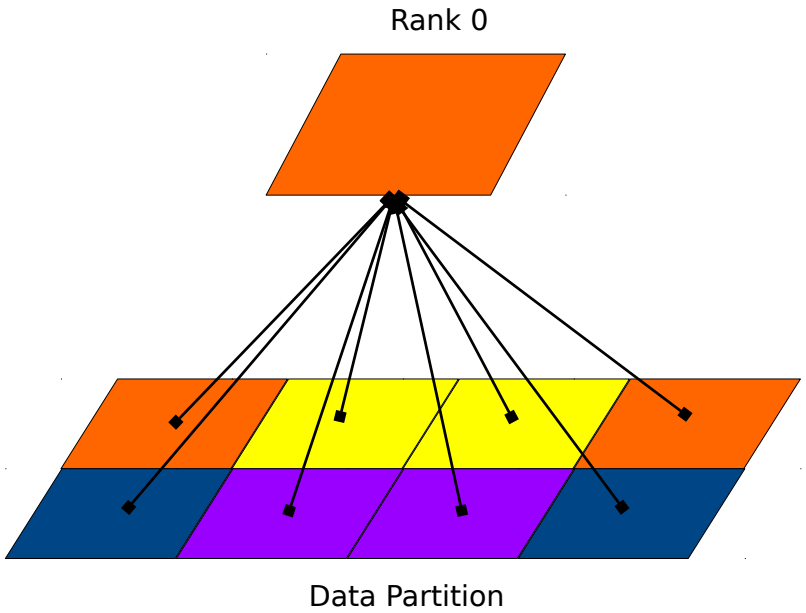


Figure: The data is partitioned onto 4 ranks: Rank 0 , Rank 1 , Rank 2 , and Rank 3 .

Asynchronous Diffusion Approach

- Ranks aggregate tile information and steal from neighbors.
- Steal heuristic taken from multi-constraint METIS refinement strategy.
- Rebalance frequency set to 40 ms (every $\mathcal{O}(100)$ RK-stages).

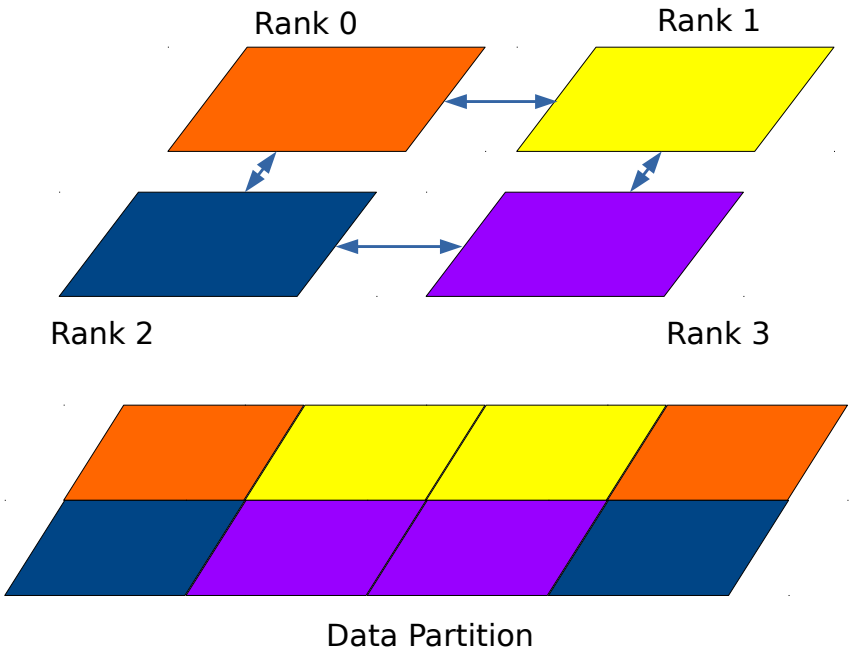
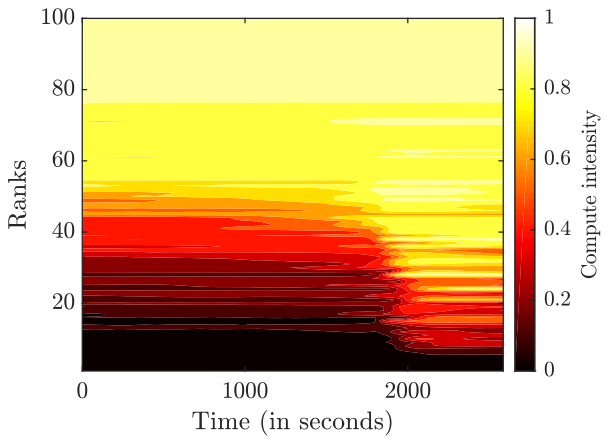


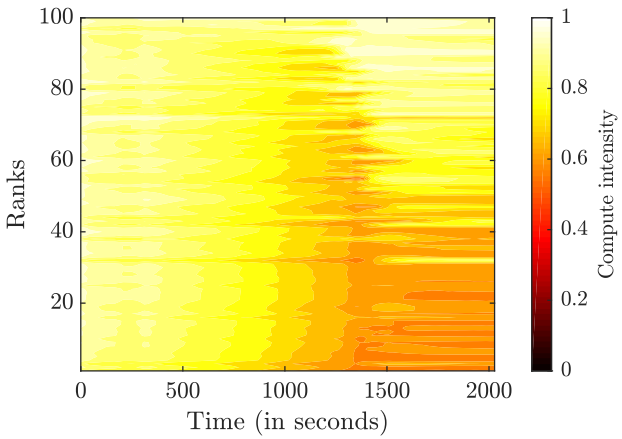
Figure: The data is partitioned onto 4 ranks: Rank 0 , Rank 1 , Rank 2 , and Rank 3 .

Results

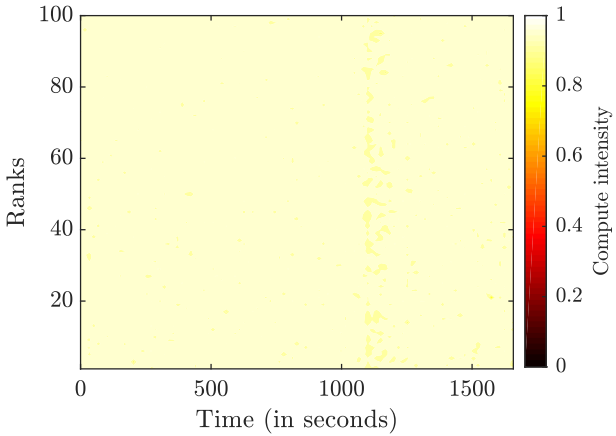
Compute Intensities



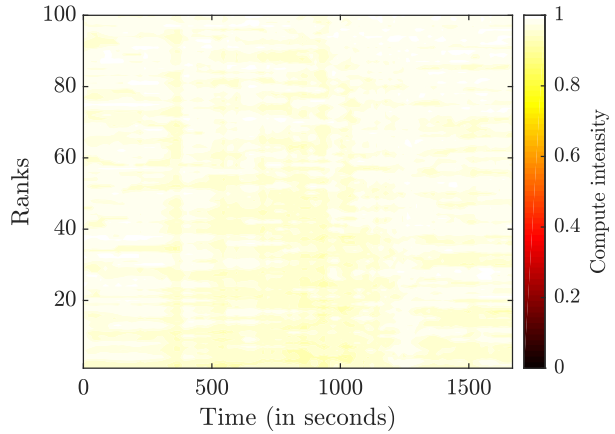
(a) Static



(b) Multi-constraint Static



(c) Semi-Static



(d) Asynchronous Diffusion

Load imbalance

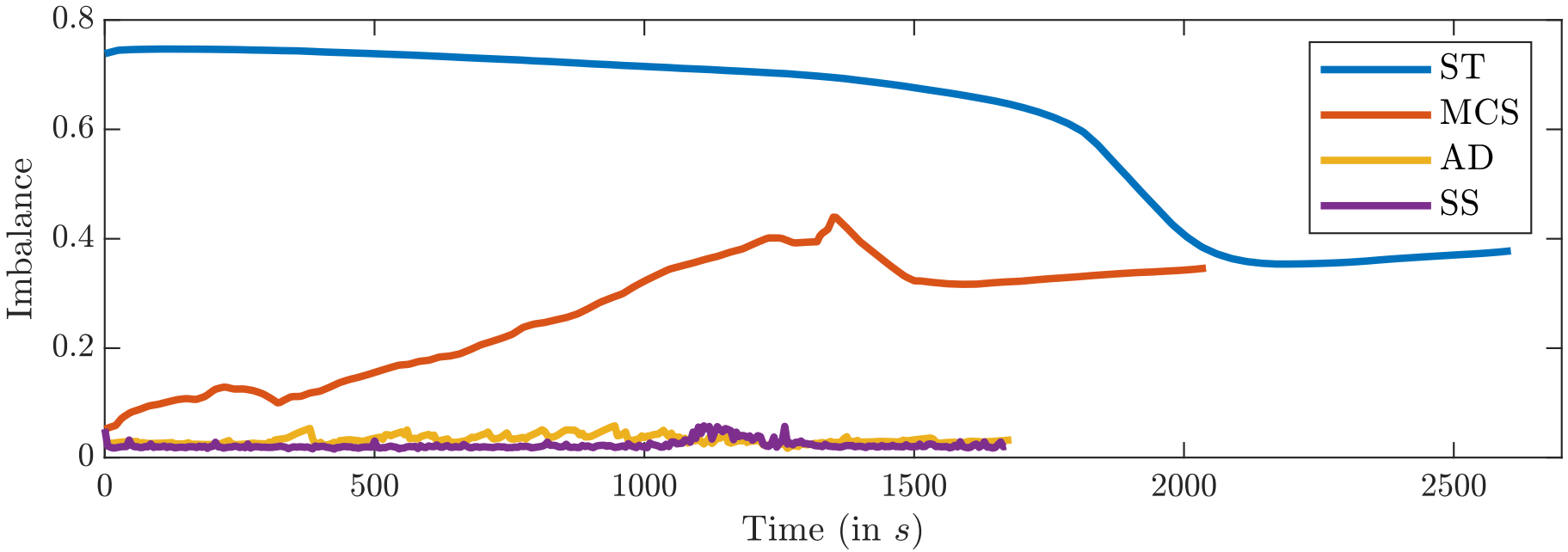


Figure: Load imbalance for Storm 36 using 1200 cores on an Edison-like machine. The following load balancing strategies were evaluated: static (ST), multi-constraint static (MCS), asynchronous diffusion (AD), and semi-static (SS).

Parallel Efficiencies

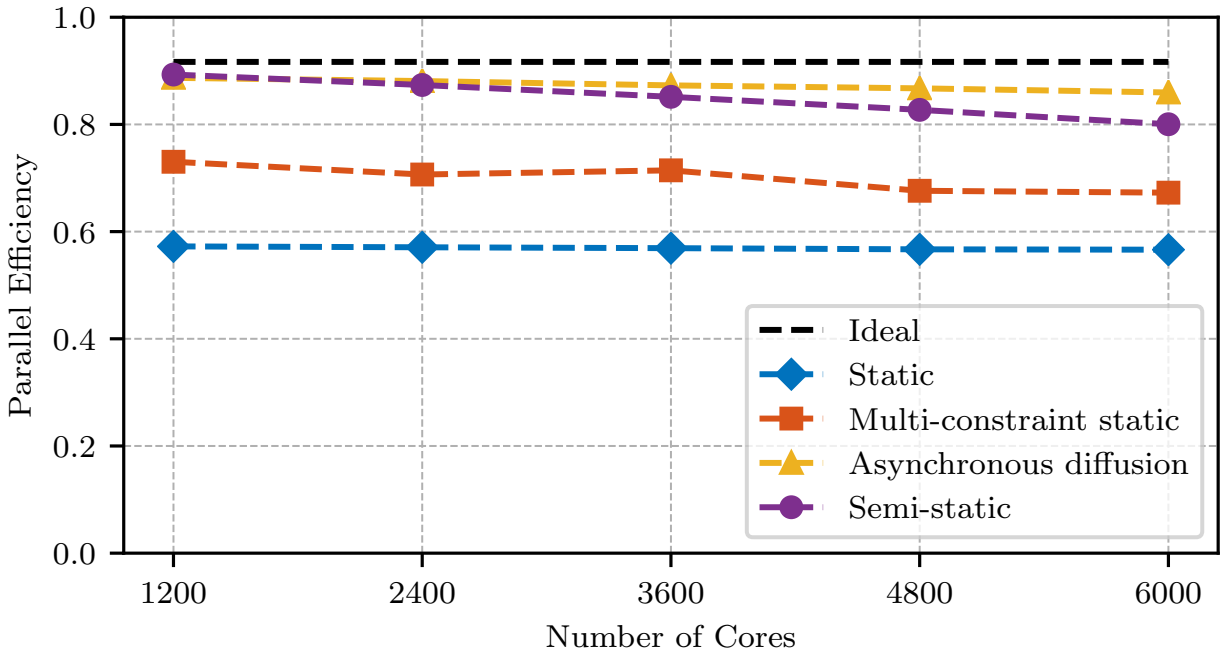


Figure: Parallel Efficiency $E(n) = T^*/(nT^n)$ simulated on an Edison-like machine for 1200 to 6000 cores. For asynchronous diffusion, $E(6000)/E(1200) = 96.9\%$.

Conclusions

Conclusion

- At 50 nodes, we achieve a speed-up of 1.55 versus traditional static load balancing and 1.22 versus multi-constraint load balancing.
- At 50 nodes, the semi-static approach moves 720,000 tiles, and the asynchronous diffusion approach moves 3,600 tiles.
- Achieve 93.7 % of maximum attainable performance at 6,000 cores; Algorithms are scalable over operational core counts.
- Going forward the coastal modeling community is looking to generate meshes containing all flood plains; this corresponds to an element dry fraction of 75%.
- On going work to implement these load balancing strategies using HPX; code can be found at `https://github.com/UT-CHG/dgswemv2`.

Thank you for your attention!

Acknowledgements

M.B. was funded by the Department of Energy through the Computational Science Graduate Fellowship, Grant DE-FG02-97ER25308. This manuscript has been authored by authors at Lawrence Berkeley National Laboratory under Contract No. DE-AC02-05CH11231 with the U.S. Department of Energy. Furthermore, this research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231.

DGSim Validation

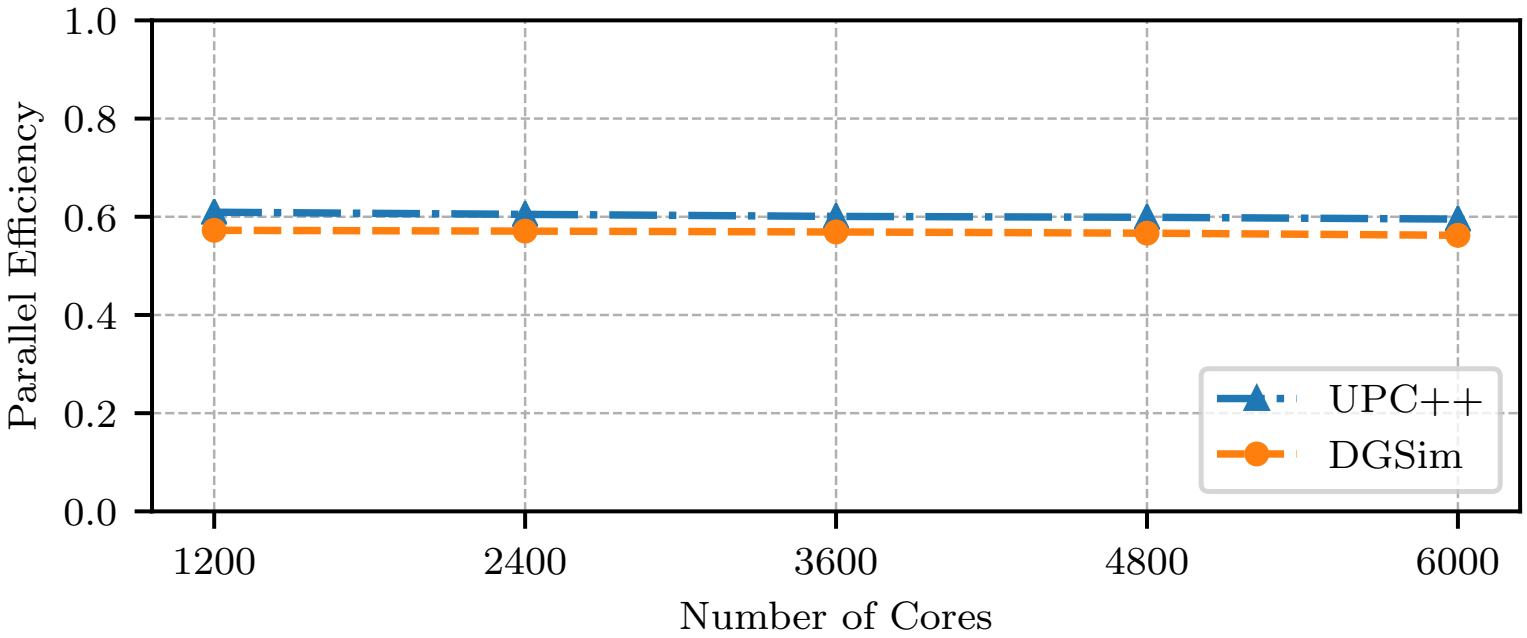


Figure: Validation of DGSim against a skeletonized application code on NERSC's Edison. The skeletonized application uses UPC++ for all inter-process communication. The simulated results are within 7% of the measured results.

Load Balancing Statistics

	Static	Multi-constraint static		Asynchronous diffusion					Semi-static				
Cores	T_{ST}	T_{MC}	S_{ST}	T_{AD}	TM	σ_{TM}	S_{ST}	S_{MC}	T_{SS}	TM	σ_{TM}	S_{ST}	S_{MC}
1200	2,613	2,048	1.28	1,686	3,654	145	1.55	1.21	1,675	723,109	2,325	1.56	1.22
2400	1,310	1,058	1.24	849	9,532	189	1.54	1.25	856	681,654	557	1.53	1.24
3600	876	698	1.26	571	16,614	319	1.53	1.22	585	641,950	568	1.50	1.19
4800	659	553	1.19	431	24,402	319	1.53	1.28	452	593,709	3,932	1.46	1.22
6000	528	444	1.19	348	36,955	761	1.52	1.28	373	574,109	1,255	1.41	1.19

Table: All times are reported in seconds. For each case, 5 runs were performed. The standard deviation of all execution times was found to be below 2% of the mean. T corresponds to the execution time in seconds. TM refers to the number of relocated tiles. The speed-up S_X is the speed-up of the simulation relative to strategy X at the given core count. Due to non-determinism in our simulation, we've reported standard deviations of tiles moved σ_{TM} for a sample size of 5.