A Toolbox for Load Balancing Development and Analysis in WarpX/AMReX Applications

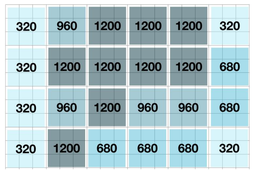
Jessica Imlau Dagostini[1], Sowmya Yellapragada[2], Kevin Gott[3], Rebecca Hartman-Baker[3]
[1]University of California Santa Cruz; [2]University of Utah; [3]Lawrence Berkeley National Laboratory

Abstract

In this work, we present a new toolbox designed to streamline the collection and analysis of load balancing data from WarpX/AMReX simulations. Our tools enable running simulations and efficiently collect and parse load balancing data from WarpX runs, and a mechanism to re-run collected data with different load balancing strategies for posterior analysis and comparison. We demonstrate the applicability of our tool by performing the full-step process of collection and data analysis with a laser-ion simulation.

Background and Motivation

AMReX is a software framework designed for solving partial differential equations using adaptive mesh refinement (AMR) techniques.
WarpX, an advanced time-based 1D/2D/3D/RZ electromagnetic & electrostatic Particle-In-Cell code to generate realistic input sets. WarpX is based on AMReX.



Workload on AMReX/WarpX is mainly defined on **Boxes**. Each box has a "weight" assigned to it, and this is used to balance the load across ranks.

Testing and comparing their efficiency using realistic data can be challenging, particularly when done outside the context of the entire application.
Our toolbox simplifies the extraction of data from WarpX and enables developers to conduct statistical load balancing inferences over real data efficiently.

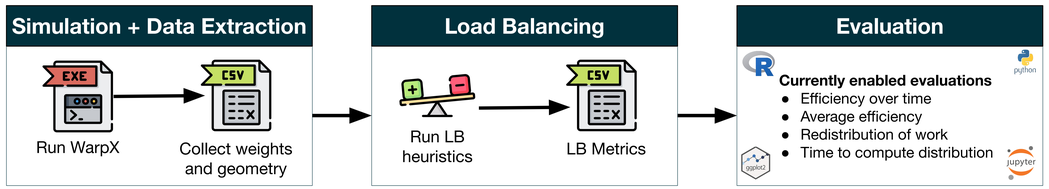
The Load Balancing Extractor Toolbox

- Our toolbox leverages one of WarpX's "Reduced Diagnostics" to collect load balancing costs
- With just two commands a developer can generate + run load balancing.

To demonstrate the applicability of our tool, we run WarpX on A100 GPU nodes from NERSC's Perlmutter, varying from 4 to 96 GPUs

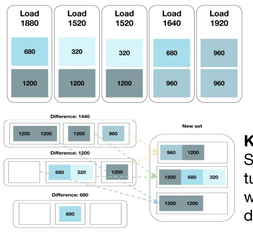
We ran the **Laser Ion Acceleration with a Planar Target** [1], with grid sizes of
- (2688, 3712) and a maximum grid size of 512, creating 48 total boxes
- (7488, 14720) and a maximum grid size of 512, creating 435 total boxes

The balancing metric used in this run was WarpX's *timers* to weight each box



LB Heuristics

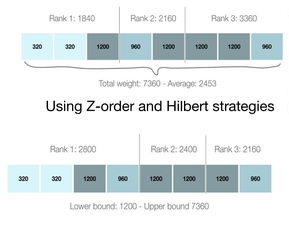
Multi-way objective/partitioning solutions



Knapsack
Order boxes by weight. Assigns the next heaviest box to the lightest rank.

Karmarkar-Karp
Split boxes into subsets and tuples. Mix the two subsets with the most significant difference.

Communication-approximation solutions



Using Z-order and Hilbert strategies

Space Filling Curve
Order boxes by their proximity in the grid. Cut this sorted list by maximum allowable weight (proportional average)

Painters Partition
Order boxes by proximity in the grid (as SFC). Binary search for the optimal cutting weight.

Combinations of algorithms
These algorithms can be mixed and test interesting combinations, e.g. **SFC+Knapsack** combines an SFC algorithm across nodes, optimizing inter-node communication, with Knapsack on each individual node to evenly distribute the workload node-locally.

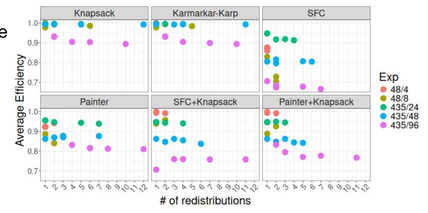
A Laser-Ion Acceleration Story

With collected metrics, the user can start asking questions:
- What is the efficiency behavior of each algorithm over time?

Knapsack and Karmarkar-Karp show the highest efficiencies. From SFC/comms-aware solutions, Painter's Partition gets good results.

We easily run same simulation with varying redistribution algorithms

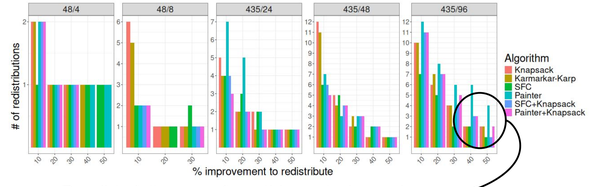
Correlating efficiency and redistributions, we can observe that Knapsack, Karmarkar-Karp, and Painter+Knapsack present most of results in the desired quadrant (top-left)



Finally, the time to run the balancing distribution is also an important aspect to consider when analysing load balancing strategies
What is the overhead each algorithm brings to the simulation?

What are the peaks over the execution?
- Redistribution of workload to improve efficiency
- WarpX uses a ratio efficiency to decide when to redistribute
- User-defined variable (default value is 10% of improvement)

How many redistributions happens with different % of improvement?

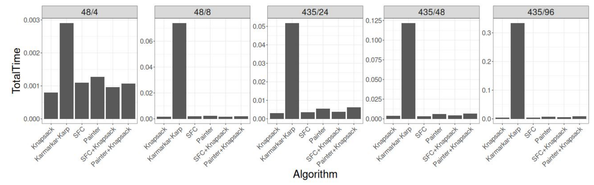


Bigger ratios to redistribution impacts on the number of redistributions each algorithm performs

Possible follow-ups from this analysis can be
- What is the impact of less redistribution on performance?
- What algorithm does less redistribution and keeps high efficiency?

Our toolbox easily enables testing different improvement ratios and analysis of their impact efficiency without having to re-run full experiment

The toolkit collect multiple metrics that allows different analysis to find 'sweet spots'



Our toolbox allows users to focus on the statistical analysis of load balancing, empowering them to explore the effects of different strategies and find the optimal solutions.

Conclusions & Future Work

This toolbox simplifies testing multiple load-balancing strategies to find optimal configurations and reduce simulation runtime; Enables targeted statistical analysis, leading to significant performance gains.
Future work:
- Extend this toolbox to consider memory limitations inside the balancing algorithms;
- Extend to use with other AMReX-based applications;
- Use this toolbox to test power-aware load balancing strategies focusing on energy-efficient distributions.

ACKNOWLEDGMENTS

This work was supported by the Computing Sciences Summer Program from NERSC and LBL. Special thanks to Kevin Gott, Rebecca Hartman-Baker, and Sowmya Yellapragada.

References:
[1] https://warpx.readthedocs.io/en/latest/usage/examples/laser_ion/README.html
[2] https://github.com/jessdagostini/amrex_LB

Scan to access the toolbox code and see more resources