Reducing Total Run Time of Complex Simulations

Delsey Sabu[1], Amelia Hu[1], Elise Bishoff[1], Wai Lau[1], Ann Almgren[2], Andy Nonoka[2], Cy Chan[2]

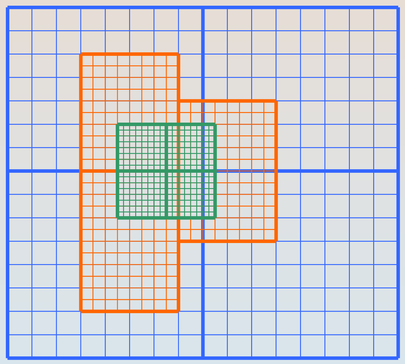
[1]Seattle Pacific University; [2]Lawrence Berkeley National Laboratory

Abstract

Large-scale simulation of physical phenomena can be very expensive, computational-wise. The research problem that I have explored in 2017 Summer is reducing total run time of complex simulations, especially in the area of large - scale parallel computation. Adaptive Mesh Refinement (AMR) is an approach that uses parallel computation to simulate physical phenomena that can span a wide range of scales. The goal of the research group was to explore strategies to reduce total runtime of complex simulations that implement the block-structured AMR and design new techniques to minimize total time to solution. The research group started with identifying various algorithms and load-distributions among processors. This research focused on mostly computational cost when parallel programming and looked for effective load distribution strategies. We analyzed the algorithms and their effect on simulations that runs with existing AMReX code frame work. We tested the algorithms on NERSC's CORI for real-time results and discovered that with the algorithms and distributions we had knapsack algorithm performed best.

AMR

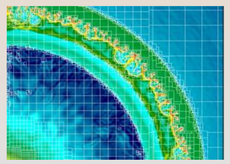
In block-structured AMR applications the domain is broken into multiple grids in such a way that specified data has better resolution where there is more data. The domain over a single-level AMR application is decomposed into regions that vary in size depending on the analysis that is needed. One or more grids are assigned to each MPI process when operating on an "owner computes" rule and thus, the MPI process only operates on the specified data in those grids. The distribution of the grids vary and hence, the communication costs can also vary depending on the amount of communication needed between the processes with varying grid distribution.



AMR approach uses rectangular grids with successively finer grids at each level.

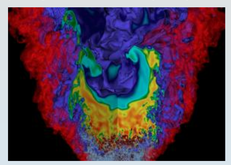
https://cs.lbl.gov/careers/summer-student-program/computing-sciences-summer-students-2017-talks-and-events/

- AMR parallelize code distributing the grids on each level across processors.
- AMR uses parallel computation to simulate physical phenomena that can span a wide range of scales.
- AMR gives the finer details needed, while also decreasing the amount of work done.



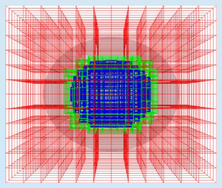
AMR grids in calculation of astrophysical mixing.

http://crd.lbl.gov/departments/applied-mathematics/center-for-computational-sciences-and-engineering/



Simulation of practical scale combustion (laboratory scale flames)

http://crd.lbl.gov/departments/applied-mathematics/center-for-computational-sciences-and-engineering/



Full star simulations of convection preceding the explosion phase of supernovae.

https://cs.lbl.gov/careers/summer-student-program/computing-sciences-summer-students-2017-talks-and-events/

Methodology

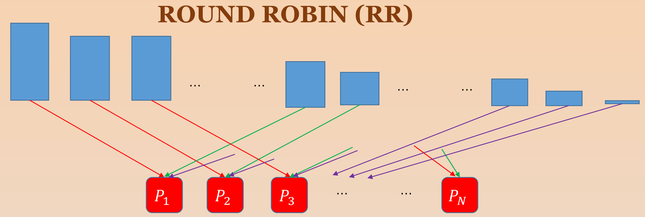
The research was divided into four parts; 1) identifying various algorithms and load-distribution among processors, 2) analyzing the algorithms and their effect on simulations that runs with existing AMReX code frame work, 3) testing the algorithms on NERSC's CORI for real-time results, 4) collecting and analyzing the results from earlier parts to form conclusion

Algorithms & Workload Distributions

Different load balancing strategies exists with unique pros and cons, but models and predictions often do not match up with reality when it comes to running complex simulations as there are varying hardware issues that arises for different runs of applications. The research group explored algorithms to reduce total runtime of complex simulations that implement the block-structured AMR and reports on the results of simulations run with different known workload distributions.

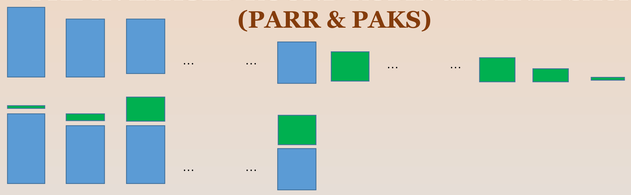
Algorithms

ROUND ROBIN (RR)



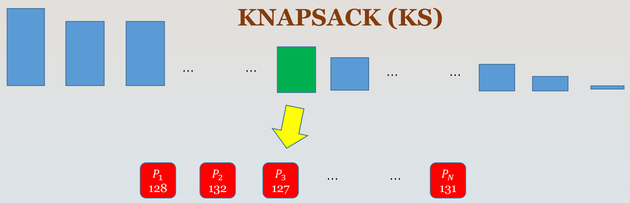
Loads in descending order are distributed to each processor until all loads have been placed on a processor

PRE-AVERAGED ROUND ROBIN and KNAPSACK (PARR & PAKS)



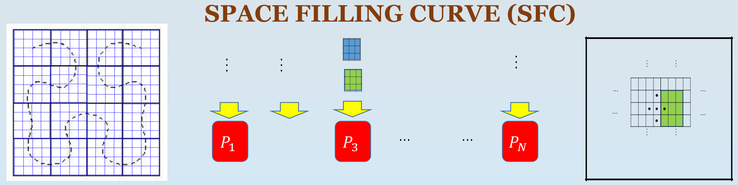
Loads in descending order and large loads are paired with smaller loads. The mapping is then taken up by Round Robin or Knapsack.

KNAPSACK (KS)



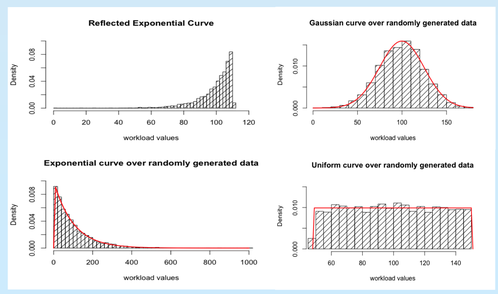
Loads in descending order are distributed to the processor that has the least amount of workload.

SPACE FILLING CURVE (SFC)



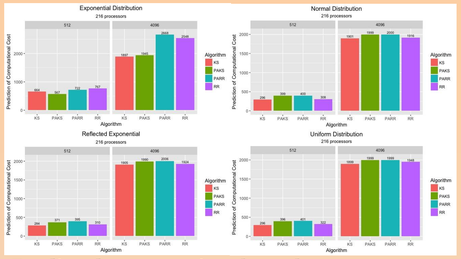
Preserve partial adjacencies between grids. Neighboring grids go to the same processor to minimize the communication cost

Distributions



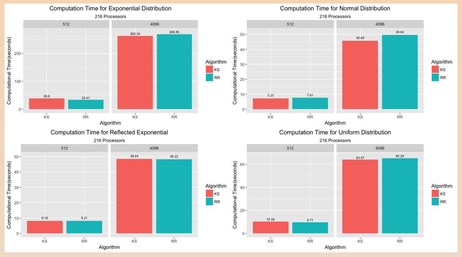
Generate random workloads according to various workload distributions; Reflected Exponential Curve (upper left), Exponential Curve (lower left), Gaussian Curve (upper right), Uniform Curve (lower right)

Results



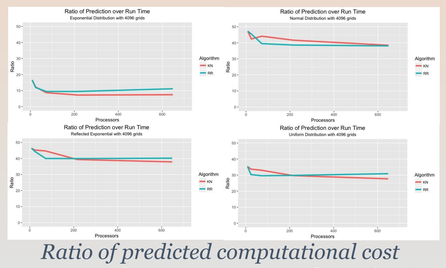
Predicted computational cost between 4 algorithms

- Predicted computational cost for distributions given 216 processors with 64, 512, and 4096 total grids.
- Predicted outcome for which algorithm would be best to use depending on the distribution and number of grids.



Real time computational time between KS and RR

- Real computation time for all distributions for 64, 512, and 4096 grids.
- Real computational time using KS and RR algorithms given 216 processors.



Ratio of predicted computational cost over real time computational cost for KS and RR

- Ratio of predicted values over our computational time for all distributions with 4096 grids.
- The graphs display lines with slopes close to zero, meaning that the predicted results matched up with the real computational time.

Conclusion

The research group predicted that the KS algorithm would perform better than RR, PARR, and PAKS. Confirming the predictions, the results showed that KS algorithm took less time for computation than the other algorithms when tested. The research group compared the predicted computational cost with the actual computational time and found that predictions generally matched the real time results, confirming the idea that the mathematical models used to predict the computational cost for RR and KS is accurate. The workloads that followed the exponential distribution took longer to run than uniform, normal and reflected exponential, which all had almost equal computational run times. The research group has concluded that given KS and RR, KS algorithm should be used to distribute workload among processors.

Future work will entail of analyzing other load balancing strategies that not only include computational time, but also consider communication time.

Acknowledgments

This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the Visiting Faculty Program (VFP) program. Special thanks to the researchers at Center for Computational Sciences and Engineering (CCSE) and Computing Research Division (CRD).