Parallel Sparse and Conventional FFTs, Applications and Implementation

Date: 2/14/2020
Time: 10:55 AM - 12:35 PM & 3:20 PM - 5:00 PM
Room: 505

Aim of this minisymposium :

- The fast Fourier Transform (FFT) is an algorithm used in a wide variety of applications, yet does not make optimal use of many current and emerging platforms such as many-core processors, GPUs, and distributed-memory systems.
- Hardware utilization performance on its own does not, however, imply optimal problem-solving.
- The purpose of this minisymposium is to enable an exchange of information between people working on FFT algorithms such as sparse and conventional FFTs, to those working on FFT implementations, in particular for parallel hardware.

Samar Aseeri, PhD
Computational Scientist
KAUST

Benson Muite, PhD
Research Fellow of Distributed Systems
University of Tartu

Meetings To Date

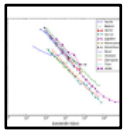
- Birds of Feathers: SC17 and ISC18
- Presentations: SIAM PP 18 and IXPUG Middle East Conference 2018

Press

- SIAM-NEWS blog: State-of-the-Art FFT: Algorithms, Implementations, and Applications
- SIAM-NEWS blog: Next Generation FFT Algorithms in Theory and Practice: Parallel Implementations, Sparse FFTs, and Applications

Experience

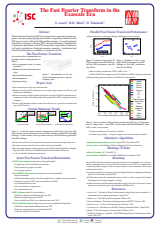
- Benchmarking numerical solution of the Klein-Gordon equation using FFTs
- Period (2014 –present)



Research (selected)

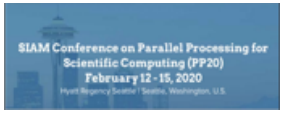
- S. Aseeri, O. Batrasev, M. Icardi, B.Leu, A. Liu, N. Li, B.K. Muite, E. Müller, B. Palen, M. Quell, H. Servat, P. Sheth, R. Speck, M. Van More, J. Vienne, “Solving the Klein-Gordon equation using Fourier spectral methods: A benchmark test for computer performance”, ACM DL - Proceedings of the 23rd High Performance Computing Symposium (HPC 2015), held in Conjunction with 2015 Spring Simulation Multi-Conference, April 2015.
- Aseeri, Samar; Chatterjee, Anando; Keyes, David; Verma, Mahendra. Implementing some Strategies to Reduce Communication Time of FFT Algorithm. (Paper -YNP)

Project Poster



- **proj125:**
(PP05) The Fast Fourier Transform in the Exascale Era-ISC 2018

Upcoming Meeting



- Parallel Fast Fourier Transforms (PFFT) mini-symposium at SIAMPP 2020, Seattle - Feb 14th
- SC20.....possible workshop Submission: Deadline is soon.

Outreach

- wep page http://www.fft.report
- forum https://www.forum.fft.report
- mailing list fft@lists.ut.ee

Implementing some Strategies to Reduce Communication Time of FFT Algorithm

Presenter:

Samar A. Aseeri, PhD

Computational Scientist, ECRC

King Abdullah University of Science & Technology (KAUST), SA

Definition of Fourier Transform

- In 1807 Jean Fourier invented a technique to solve the heat diffusion equation in a conducting plate with arbitrary forcing. This technique was the Fourier Transform.

$$x \to k : \; f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{ikx} F(k) dk$$

$$k \to x : F(k) = \int_{-\infty}^{\infty} e^{-ikx} f(x) dx$$

Cooley and Tukey, 1965

- Cooley and Tukey published a paper on the Fast Fourier Transform an O(n log n) method for the calculation of the Discrete Fourier Transform which is an approximation of the Fourier Transform which originally is derived from the Fourier Series.
- The FFT algorithm was developed by Gauss 1805 but this was not recognized until recent time.
- Time complexity of FFT according to DFT reduces from $O(n^2)\ to\ O(n\ logn)$.

Why is it Important?

- It is an important tool for image and signal processing, radio astronomy, wave propagation (such as seismic inversion), diffusion (such as hydrocarbon reservoirs), solid and fluid mechanics and electromagnetism.
- It is an accurate and low computational cost algorithm
- It solves multiscale problems
- Derivatives are simply calculated $\widehat{f^{(n)}} = (ik)^n \hat{f}$

Limitations

- Efforts to optimize the performance of 3D parallel FFT libraries have tended to focus on slab and pencil decompositions.
  - Slab decompositions tend to perform well on small process counts.
  - pencil decompositions scale better on large core counts.
- Applications that rely on FFTs adopt different data decomposition strategies:
  - 1D decompositions give each process a complete 2D slab
  - 2D decomposition give each process a complete 1D pencil
  - 3D decompositions give each process a block that does not span the global domain in any dimension
- The main performance bottleneck of parallel 3D FFTs is the communication. Once 3D data is distributed over MPI processes, all-to-all communications are unavoidable.

Parallel Libraries

- FFTW
- FFTE
- FFTK
- 2decomp&fft
- P3DFFT
- PFFT
- OpenFFT
- AccFFT
- GPU FFT and Xion phi FFT
- Hybrid FFT

Work in Progress

- Introduction
- FFTK Library
- Shaheen Topology
- Allocation Control
- Scaling and Conclusion

Introduction

- Aim is to speed up FFTs on Cray XC40 machine by using the full bandwidth offered by the cluster
- For testing we used the FFTK parallel library developed by collaborators of this work
- Several job placements and reordering cases were examined and some findings will be highlighted here

FFTK Library

- Chatterjee, Verma, and group members of Kanpur
- Scaled up to 196608 cores of Shaheen II of KAUST for 3072^3
- Tested up 6144^3 grid
- Fluid solver TARANG uses it
- 2D pencil decomposition is typically used for large core counts

FFTK Library

- The forward transform is given by

$$\hat{f} = \sum_{k_x k_y, k_z} f(x,y,z) e^{-ik_x x}\, e^{-ik_y y} e^{-ik_z z}$$

- Algorithm
  - FFT along Z axis
  - Communicate Z pencils to Y pencils.
  - FFT along Y axis
  - Communicate Y pencils to X pencils.
  - FFT along X axis

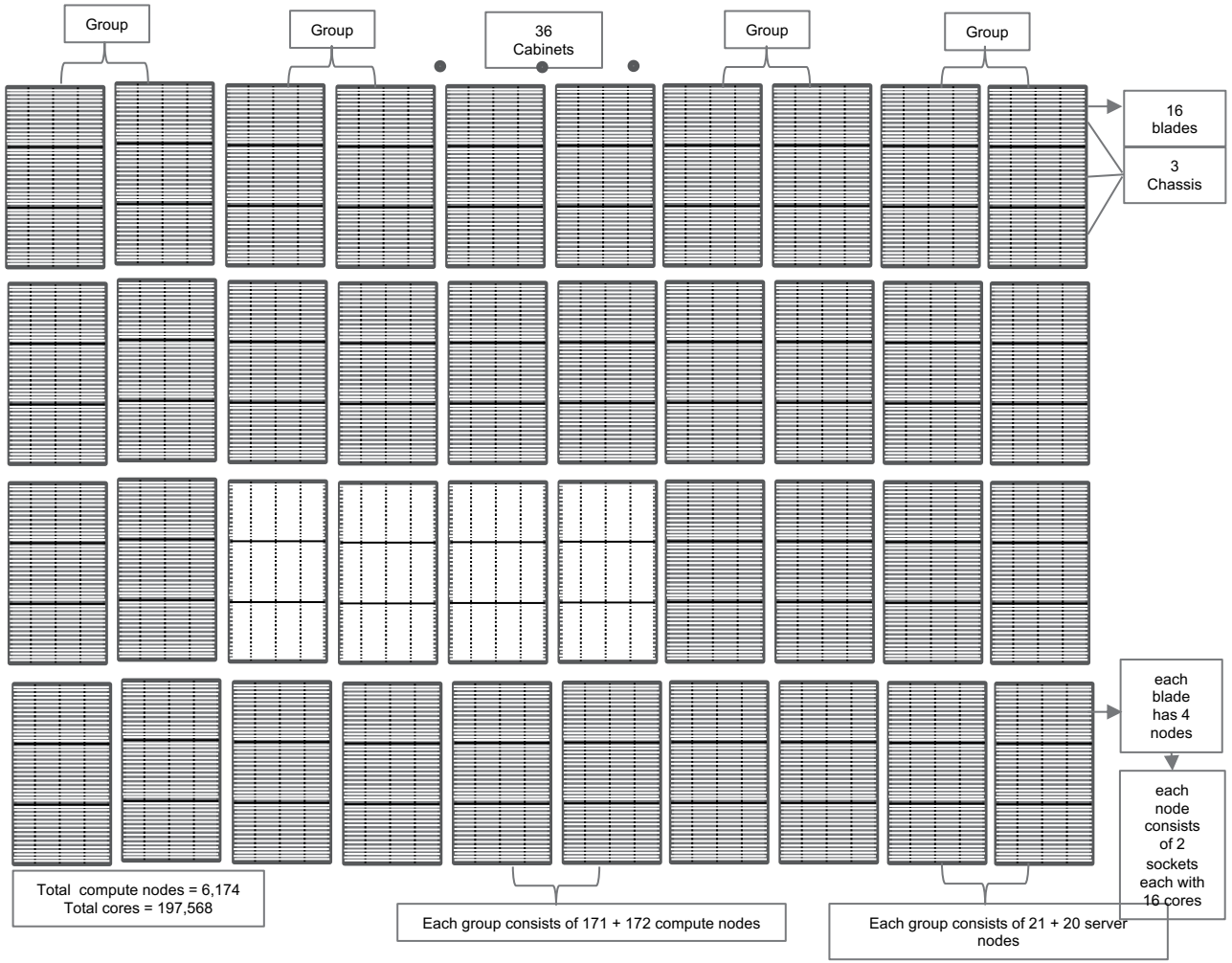
FFTK Library

- Works also for 2D data by setting $N_y = 1$
- Slab FFT can be performed by setting $p_{row} = 1$
- Available basis: FFF, SFF, SSF, SSS and ChFF

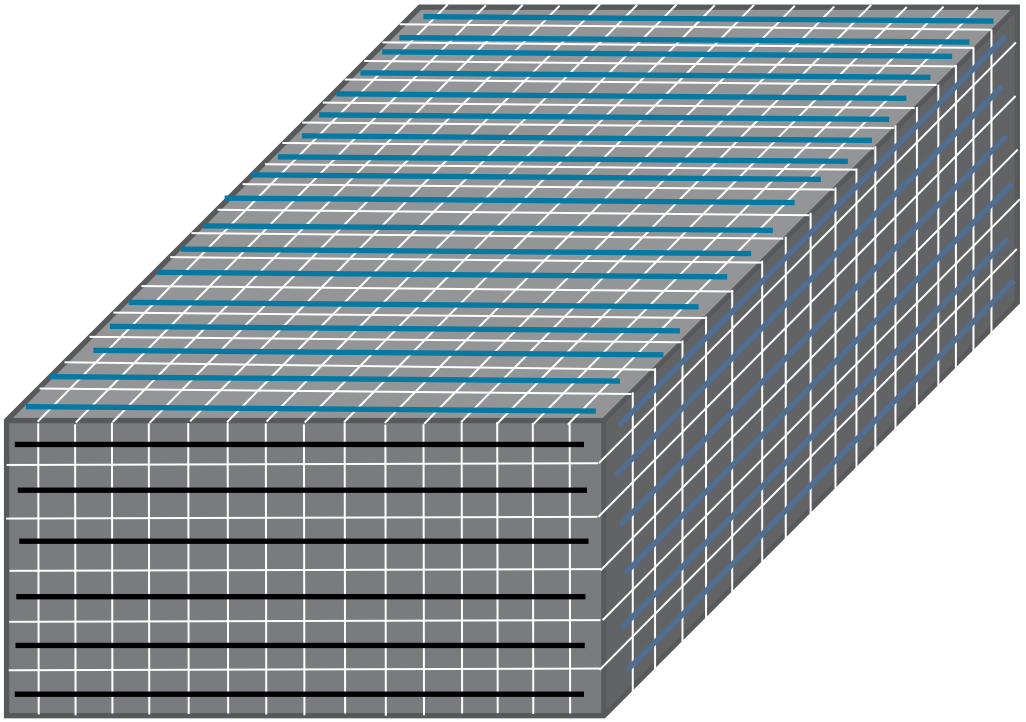
Shaheen Topology

- Cray XC40
- 38 fastest supercomputer in the world
- It consists of about 200000 CPU cores
- It manages a speed of about 7 Petaflops/s theoretical peak and 5.5 Petaflops/s of Linpack performance
- It uses Dragonfly Topology

Shaheen Topology



Shaheen Topology



Allocation Control

- Optimize Communication
  - Grid ordering
  - Contiguous Nodes
  - Job Placement

Allocation Control

- Grid Ordering
  - grid_order is a tool by which we can manually specify how MPI ranks will be distributed in nodes.
  - There are four types of rank placements that can be specified by the system using an environment variable MPICH_RANK_REORDER
  - Our primary tests shows ordering only controls MPI rank distribution after nodes have already been allocated by the system.
  - We might investigate it further even though it is not what we are looking for.

Allocation Control

- Contiguous Nodes
  - Slurm can provide us with contiguous set of nodes from within a chassis which are directly connected with each other in all-to-all fashion.
  - Since FFT also requires all-to-all communication we expect it to give better performance in this case.
  - The "$--contiguous$" flag of slurm does not necessarily assign contiguous nodes within a chassis if we for instance run tests on 48 nodes.
  - It assigns any contiguous nodes according to its digit order so could be within two chassis or within a group or within two groups.
  - Even though testing it on 32 ppn gave some improvement but still this is not what we are looking for.
  - We use SLURM_JOB_NODELIST environment variable to check the nodes allocated for the jobs.

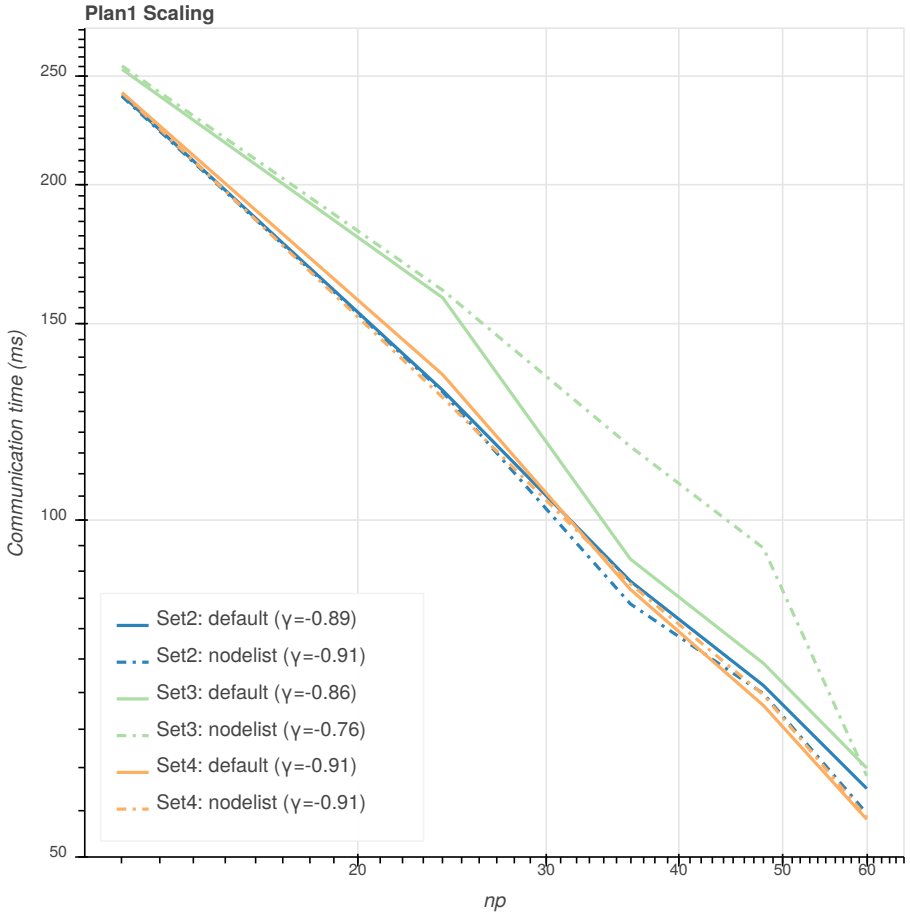
Allocation Control

- Job Placement
  - Using the Slurm flag "$--nodelist = nid0[1284 - 1343]$" we can determine specific nodes for the jobs.
  - For this a useful tool called "NID Marker" has been designed in this work specifically for Shaheen to help us visualize node locations and find node number order to be used for the above Slurm flag.

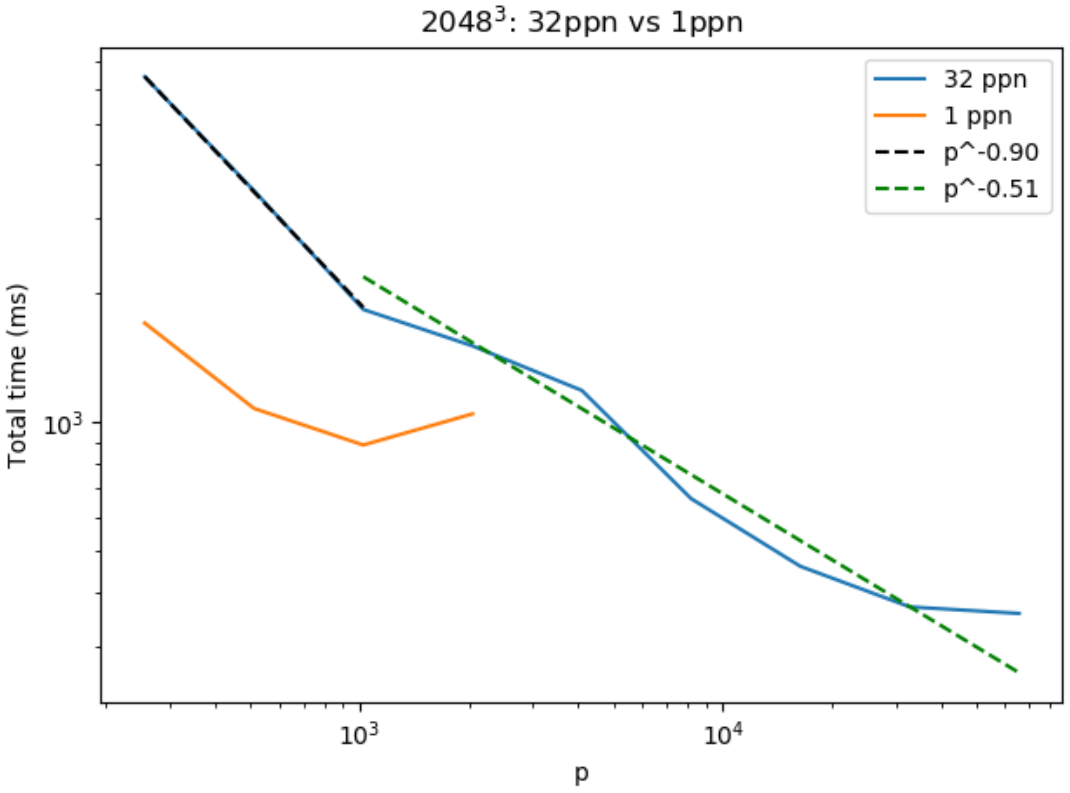
Scaling and Conclusion

- Test Case
  - Examined grid size is 512^3
  - We run up to 60 cores using 1ppn
  - A Chassis contains a max of 60 compute nodes
  - We compare performance of nodes within one Chassis verses default choice of the nodes by the system

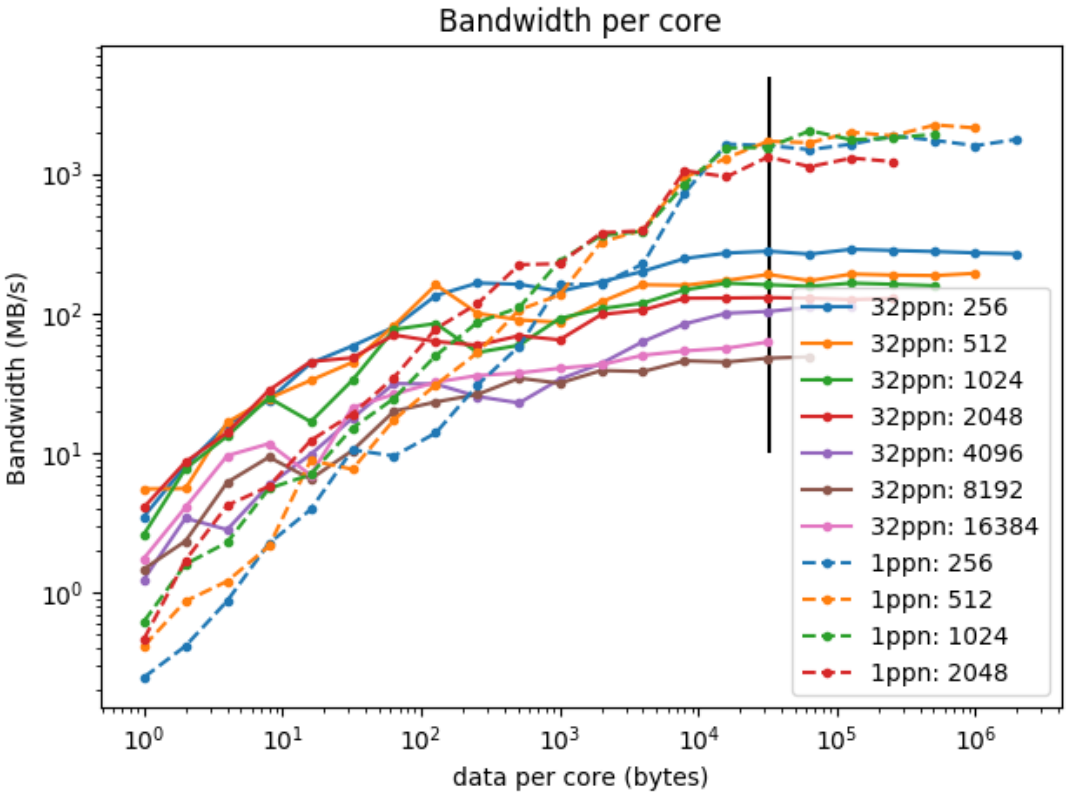
Scaling and Conclusion



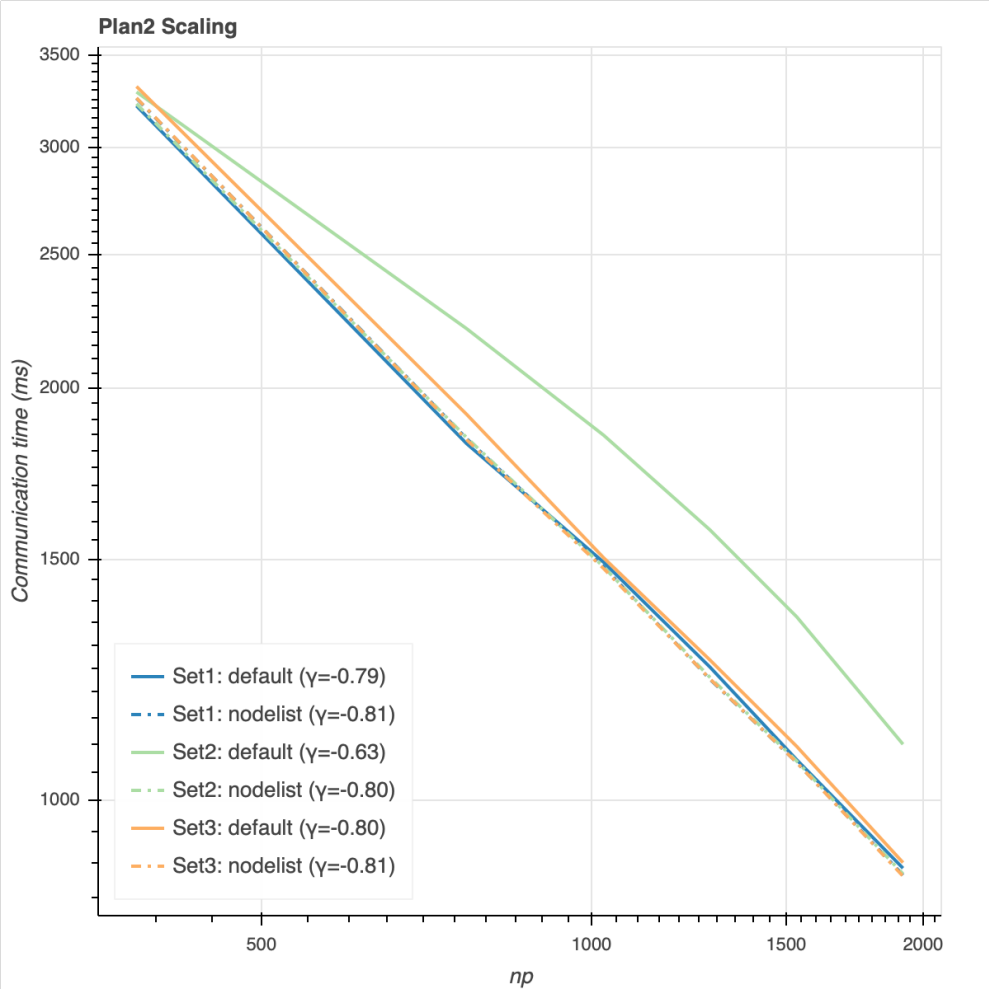
Scaling and Conclusion



Scaling and Conclusion

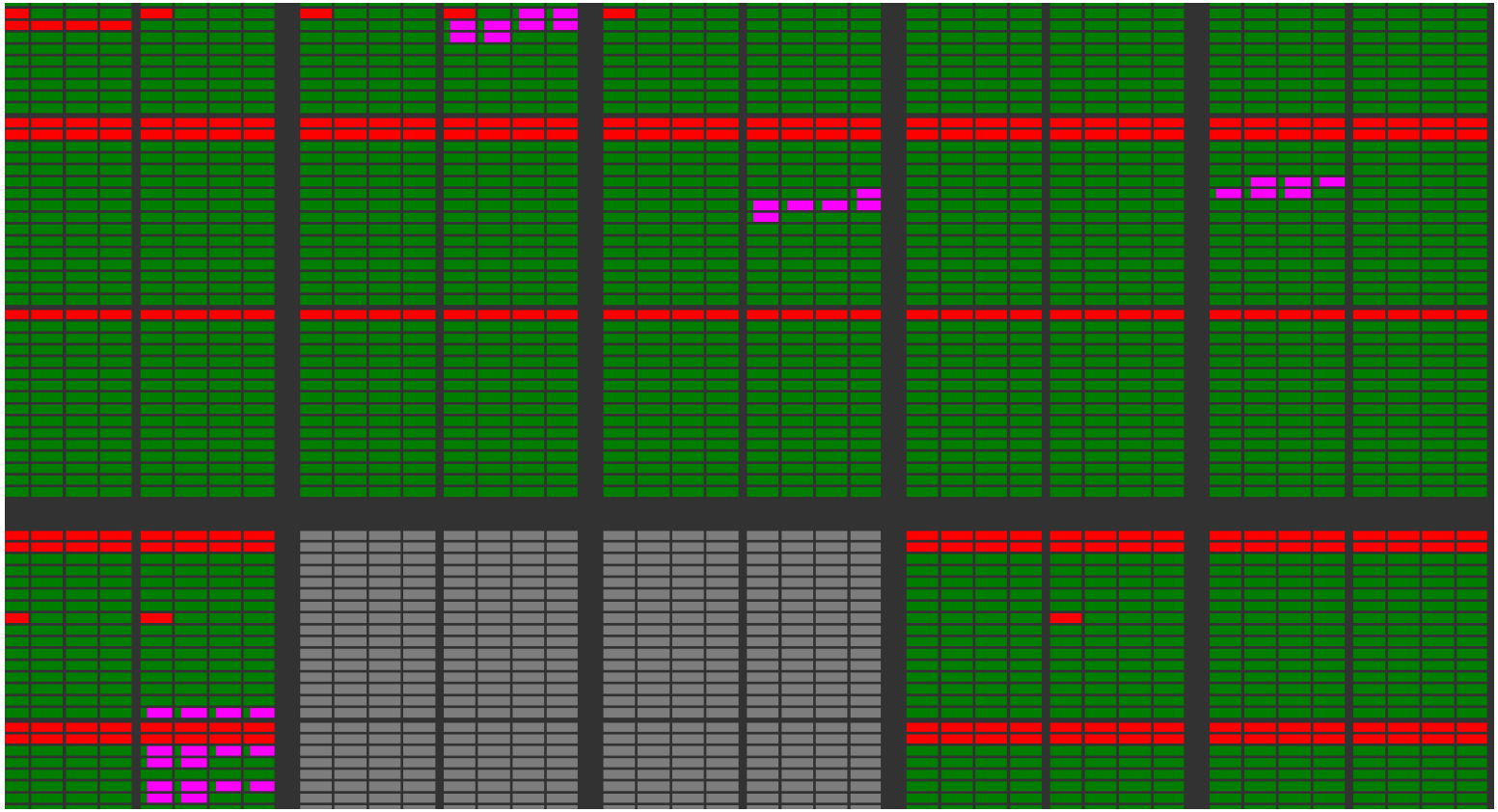


Scaling and Conclusion

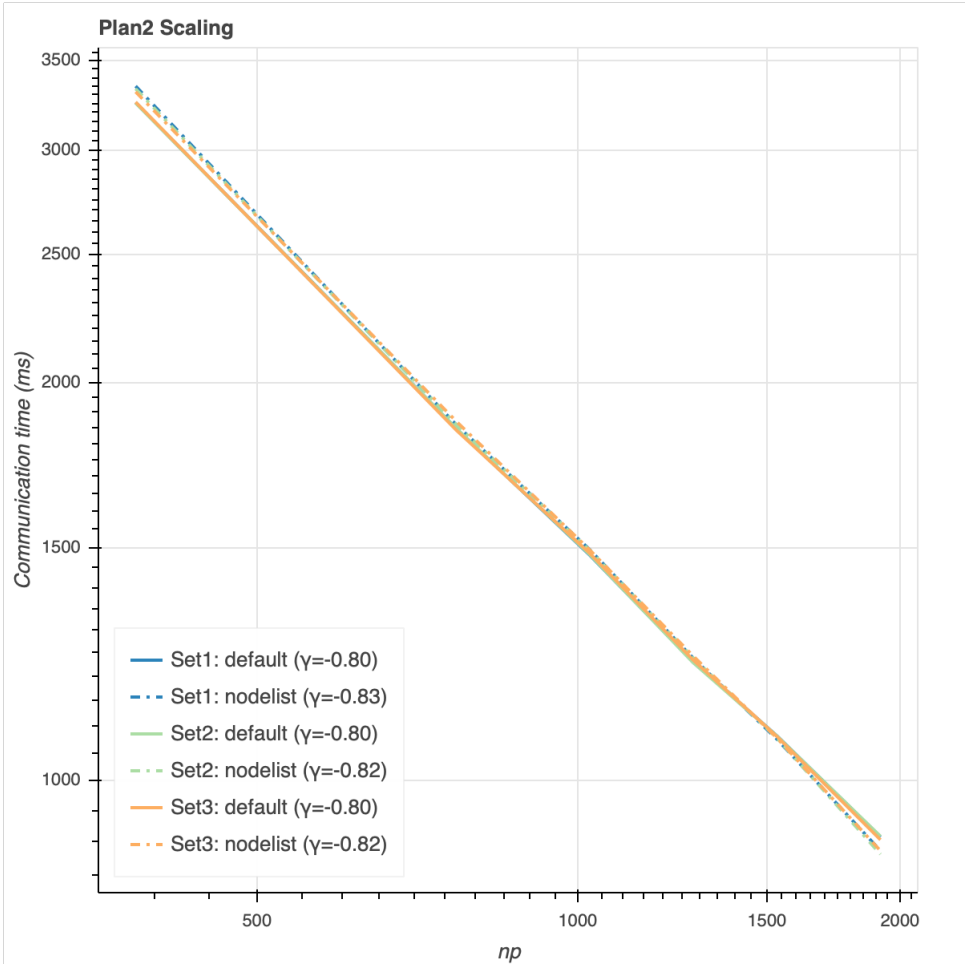


Scaling and Conclusion

Set2: Nid-Marker

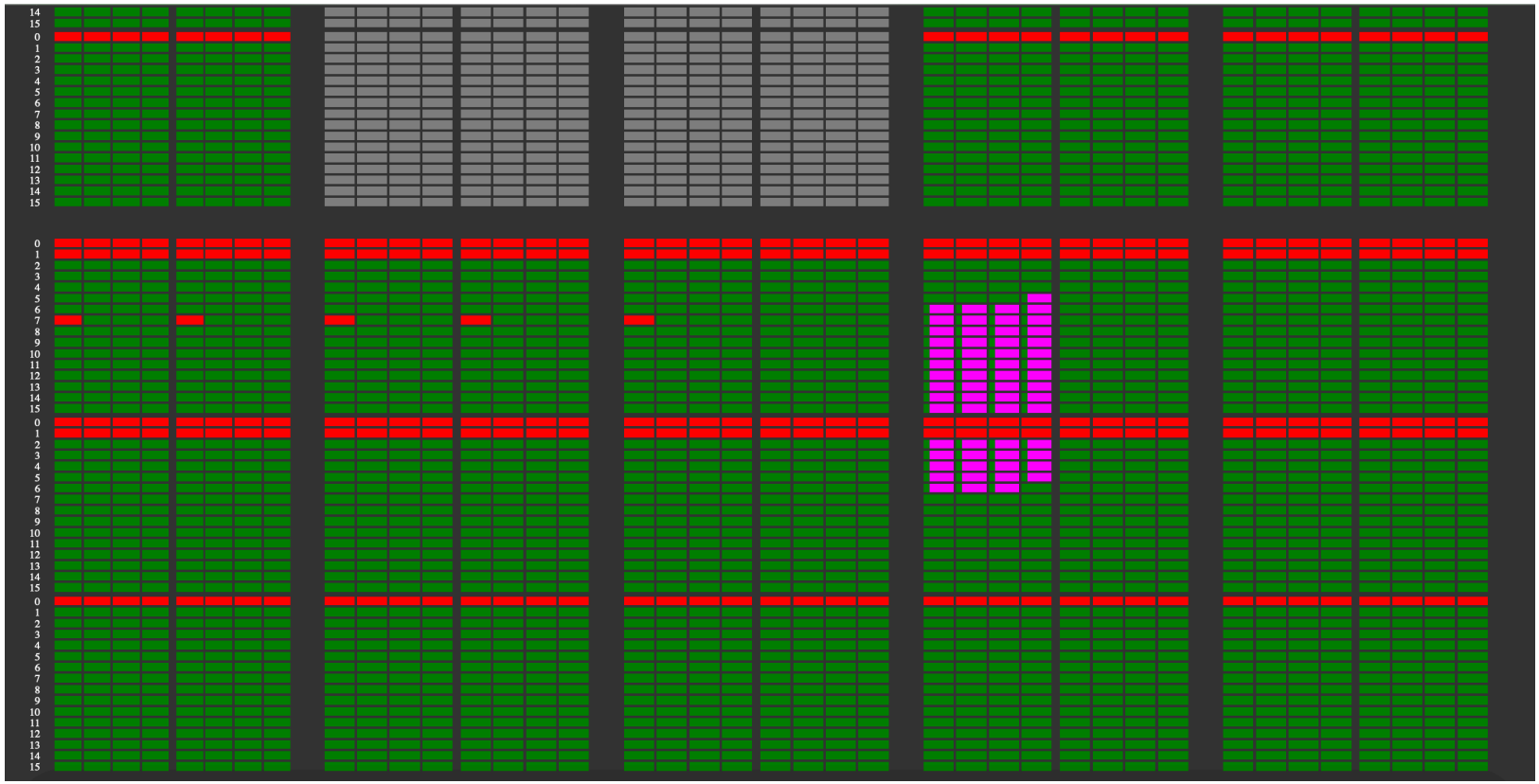


Scaling and Conclusion



Scaling and Conclusion

Set, Set2, Set3: Nid-Marker



Scaling and Conclusion

- Full core utilization
- Test large grid size to detect behavior with high core counts.
- Experiments on reserved racks
- Examining impact of dragonfly topology plugin available at slurm

Scaling and Conclusion

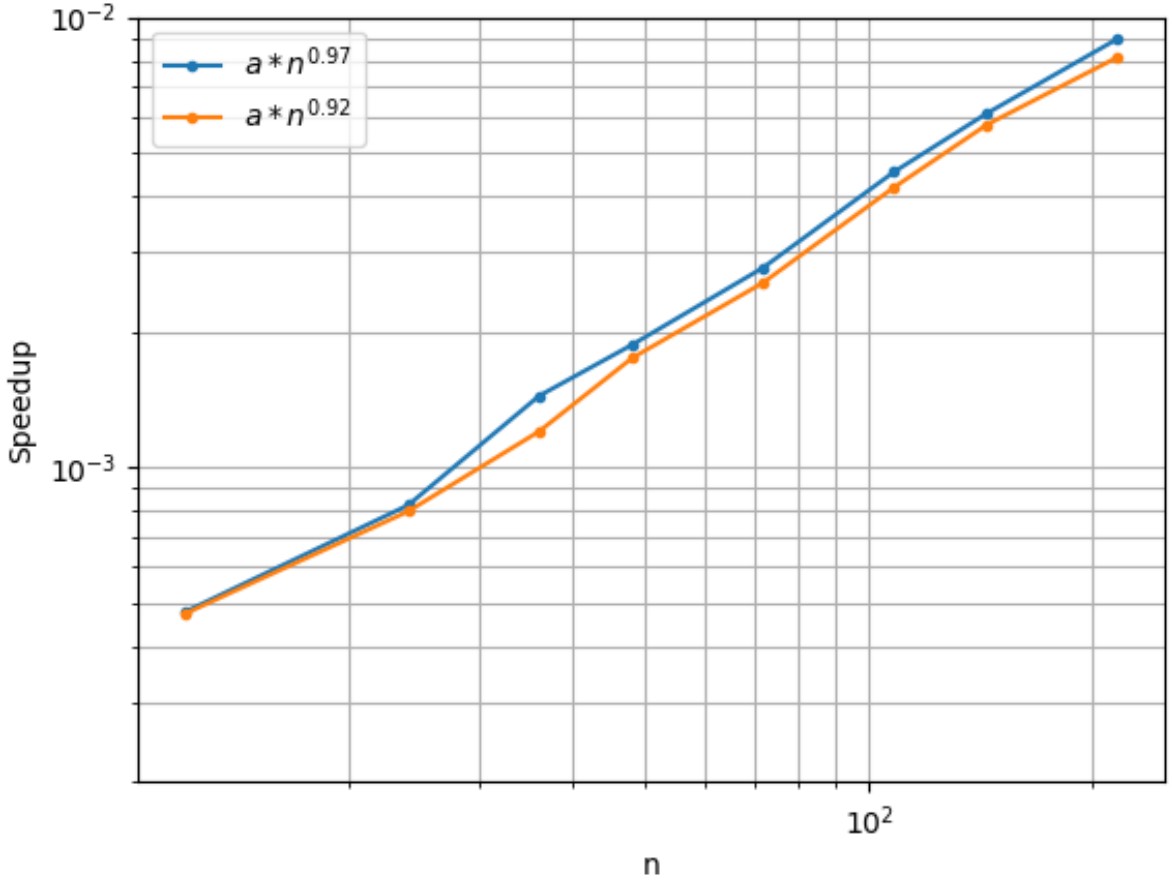
Full:
We ran 1080^3

We ran from 12 to 216 cores

Optimized:
We used 1 core per blade (4 nodes)
We used 1 chassis per group (two racks)
We used all the 18 groups

Default:
We used all core in blade (4 nodes)
We used all 6 chassis per group (two racks)
We used from 12 to 216 cores.

Scaling and Conclusion



Scaling and Conclusion

Chassis: We ran 1080^3

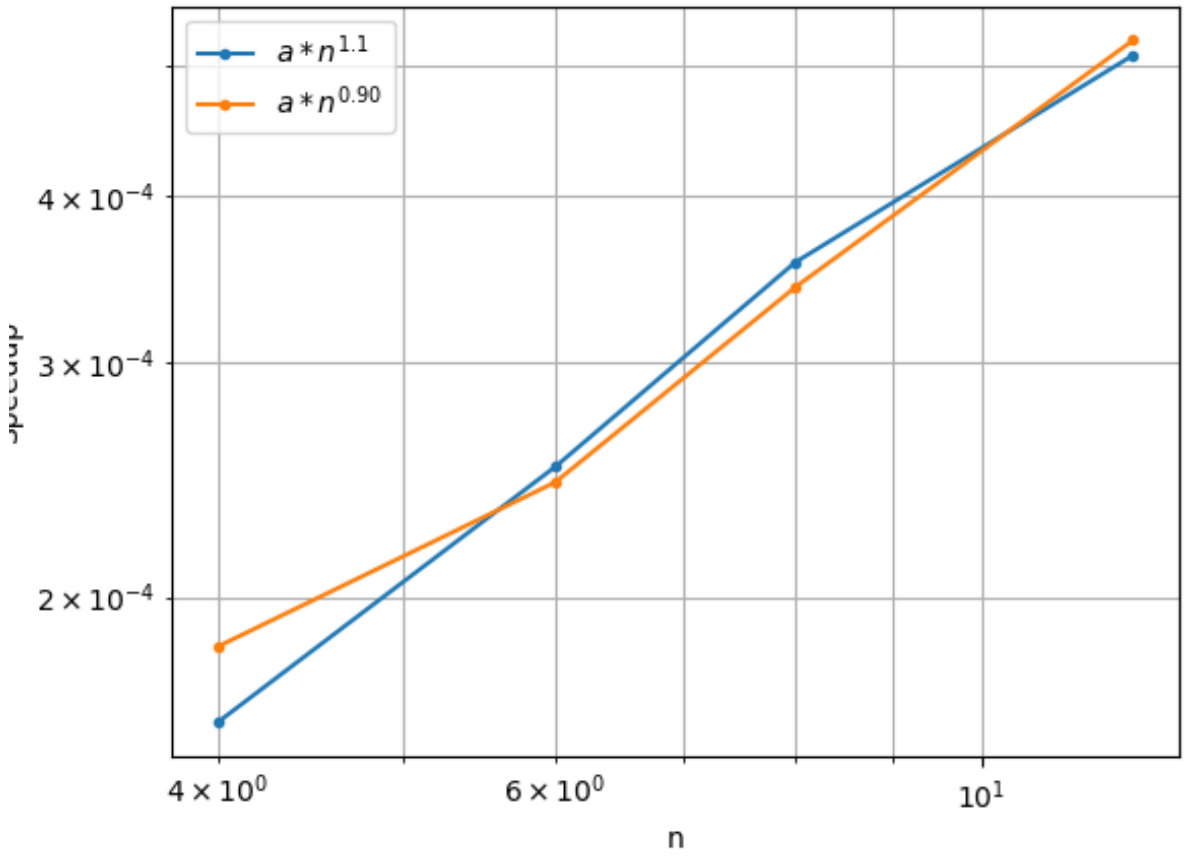
We ran from 2 to 12 cores

12 because it has many factors thus we get many points in scaling plot.

Optimized:
We used 1 core per blade (4 nodes)

Default:
We used all core in blades (4 nodes)

Scaling and Conclusion



Scaling and Conclusion

- To see if we align MPI communication with physical connections, then how much do we gain.
- From previous runs, we have seen that when we use all the cores (199608) of the system gamma was close to 0.8. From these runs we see that up to 216 nodes spread across 18 groups, we are getting very good scaling.
- Our next step will be to tweak communication pattern of Tarang to make use of this observation.
- We can use
  - 1st chassis of all groups to solve for x component of velocity.
  - 2nd chassis of all groups to solve for y component of velocity.
  - 3rd chassis of all groups to solve for z component of velocity.
  - 4th chassis of all groups to solve for temperature

Collaborators

- David Keyes, KAUST (Project PI)
- Mahendra Verma and Anando Chatterjee, IITK (FFTK developers)

Tank You

- Questions/Comments and Collaborations are welcomed!
- Email: samar.aseeri@kaust.edu.sa
- Upcoming HPC venues:
  - BID’20 at PPoPP’20
  - HPBench’20 at HPCS’20