Determining Optimal MPI Process Placement for Large-Scale Meteorology Simulations with SGI MPIplace

James Southern, Jim Tuccillo

SGI

25 October 2016

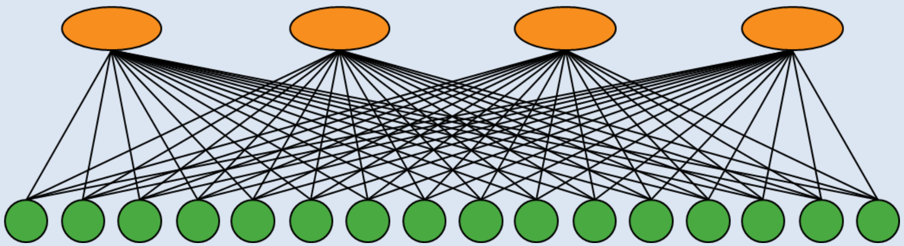
Motivation

- Trend in HPC continues to be towards more cores, slower clock speed.
  - Applications will require increasing parallelism.
  - Some parallelism will be soaked up within a node, but there will also be a requirement for inter-node communication.
  - Some of today's best performing interconnect technologies do not scale linearly with increasing system size – may no longer be feasible.
  - Future capability systems likely to have relatively sparse interconnects compared to today.
  - Applications may need to place processes carefully across topology to maximize performance.

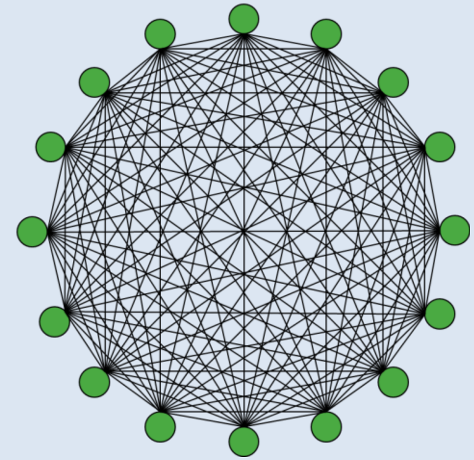
Common Topology Options

Supported on SGI ICE

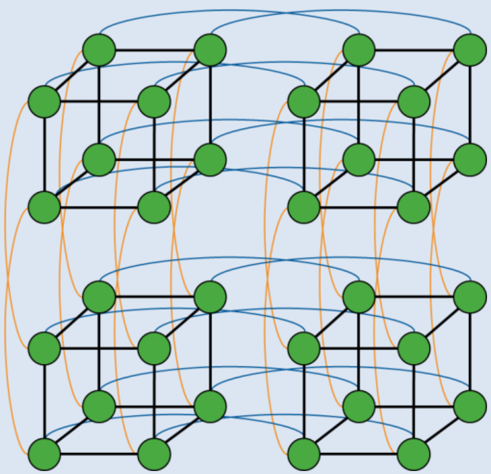
Fat Tree (Clos Networks)



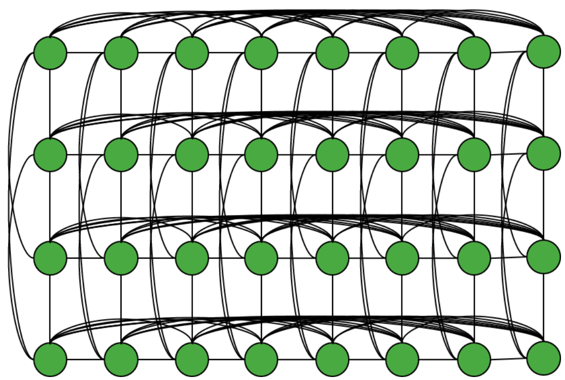
All-To-All



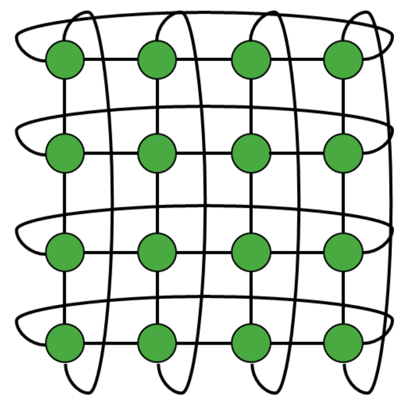
Hypercube



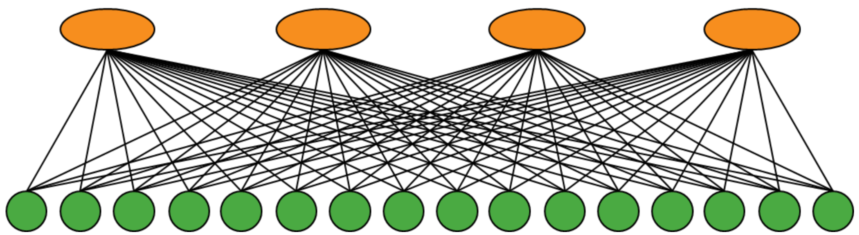
Multi-Dimensional All-To-All



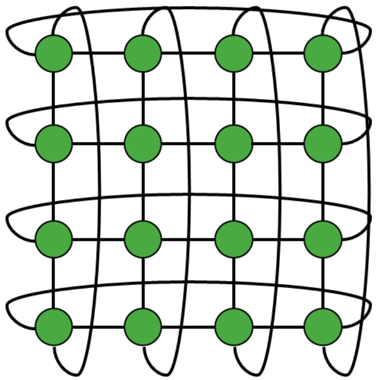
Torus



Pros and Cons

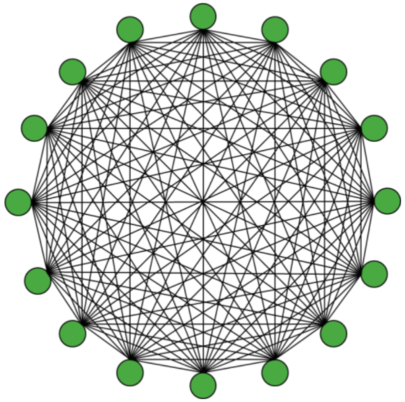


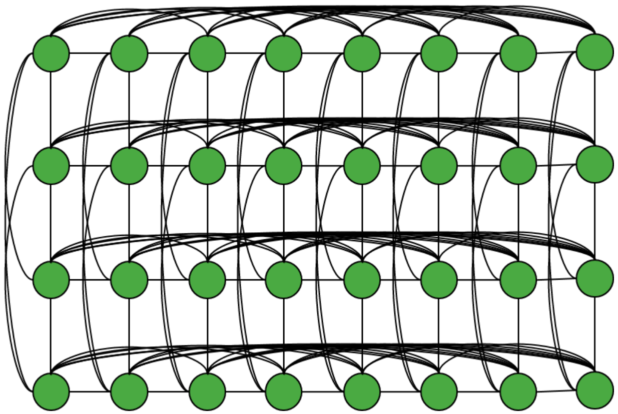
- **Fat Tree:**
  - Strong, general purpose fabric.
  - Effective for small systems.
  - Expensive for large systems.
  - Rich path fabrics and consistent hop-counts: predictable job latency.
  - Cost does not scale linearly: switching and cabling becomes increasingly expensive with size.
- **Torus:**
  - Highly scalable: switching and cabling scales linearly with size.
  - Multiple paths between two nodes: good load balancing, fault resilience.
  - Can lead to large hop-count for some messages on larger systems (poor latency).



Pros and Cons

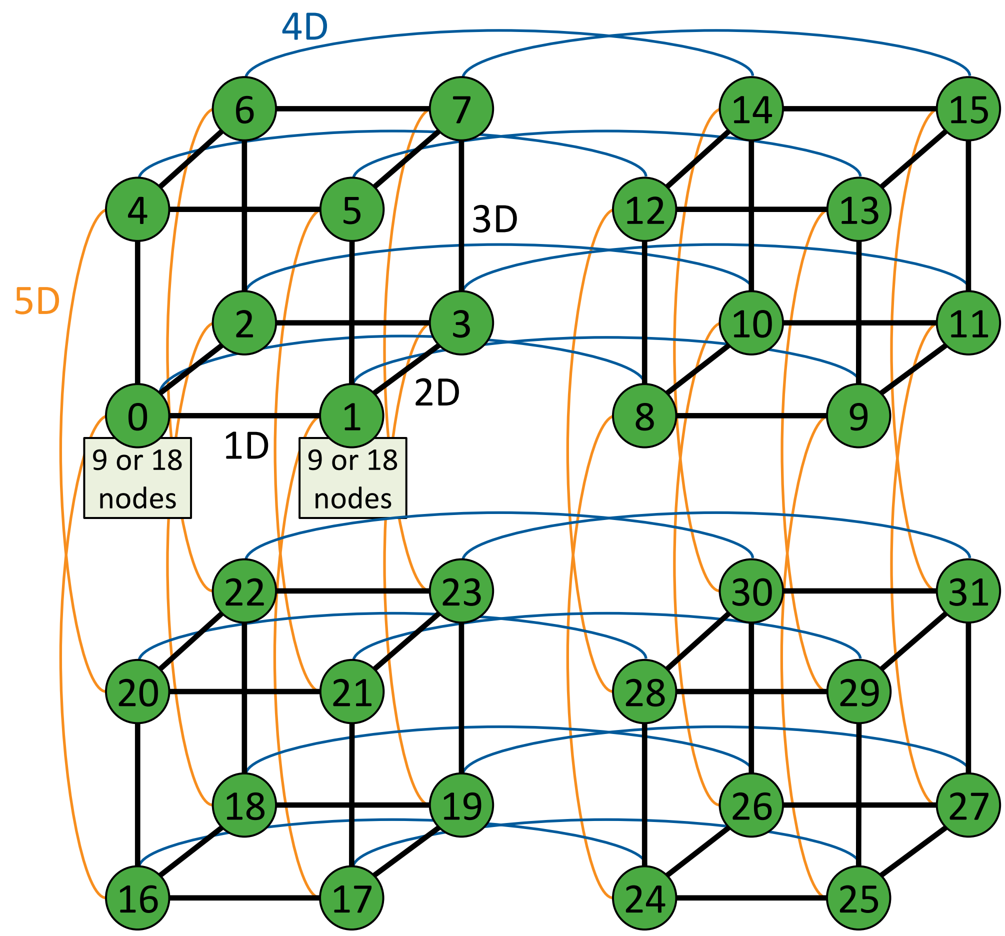
- **All-To-All:**
  - Ideal for apps that are highly sensitive to MPI latency.
  - Limited to small systems by port counts on switches.
- **Multi Dimensional All to All:**
  - Low latency interconnect: maximum hop count is *D*+1.
  - Fewer cables than All-To-All, but number of cables still does not scale linearly with system size.
  - Connectivity “islands”: sudden discontinuity in latency for jobs above a certain size or spanning two islands.





SGI's Hypercube Topology

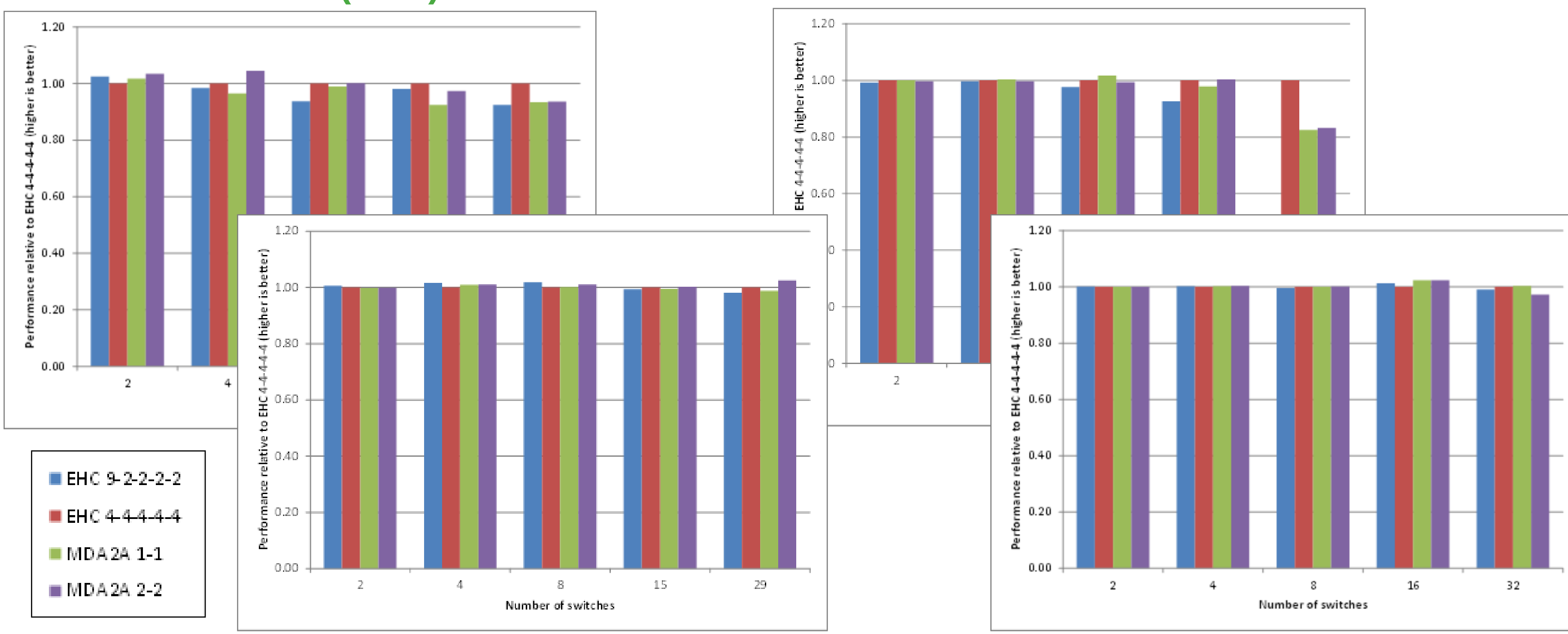
- Add orthogonal dimensions of interconnect to grow the system.
- Cost grows linearly with system size.
- Easily optimized for both local and global communication.
- Rich bandwidth capability that scales easily from small to very large systems.



Enhanced Hypercube

- At small scale, real applications tend to show limited sensitivity to interconnect topology.

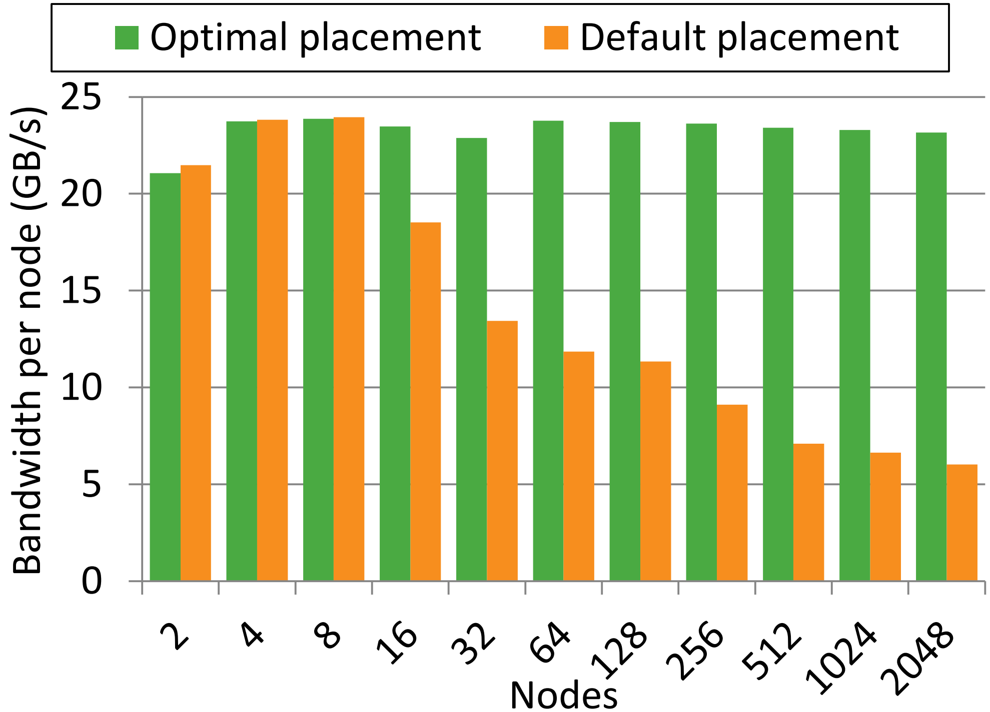
Harmonie (DKE) OFAM3



BQCD GAMESS

Process Placement

- EHC provides good off-the-shelf performance for most applications.
- It is possible to extract even better performance for some applications by placing processes optimally.
  - Put MPI processes that communicate heavily on nearby nodes.
  - This is increasingly important as the number of nodes (and switches) increases.
- SGI provide tools to assist with placement.

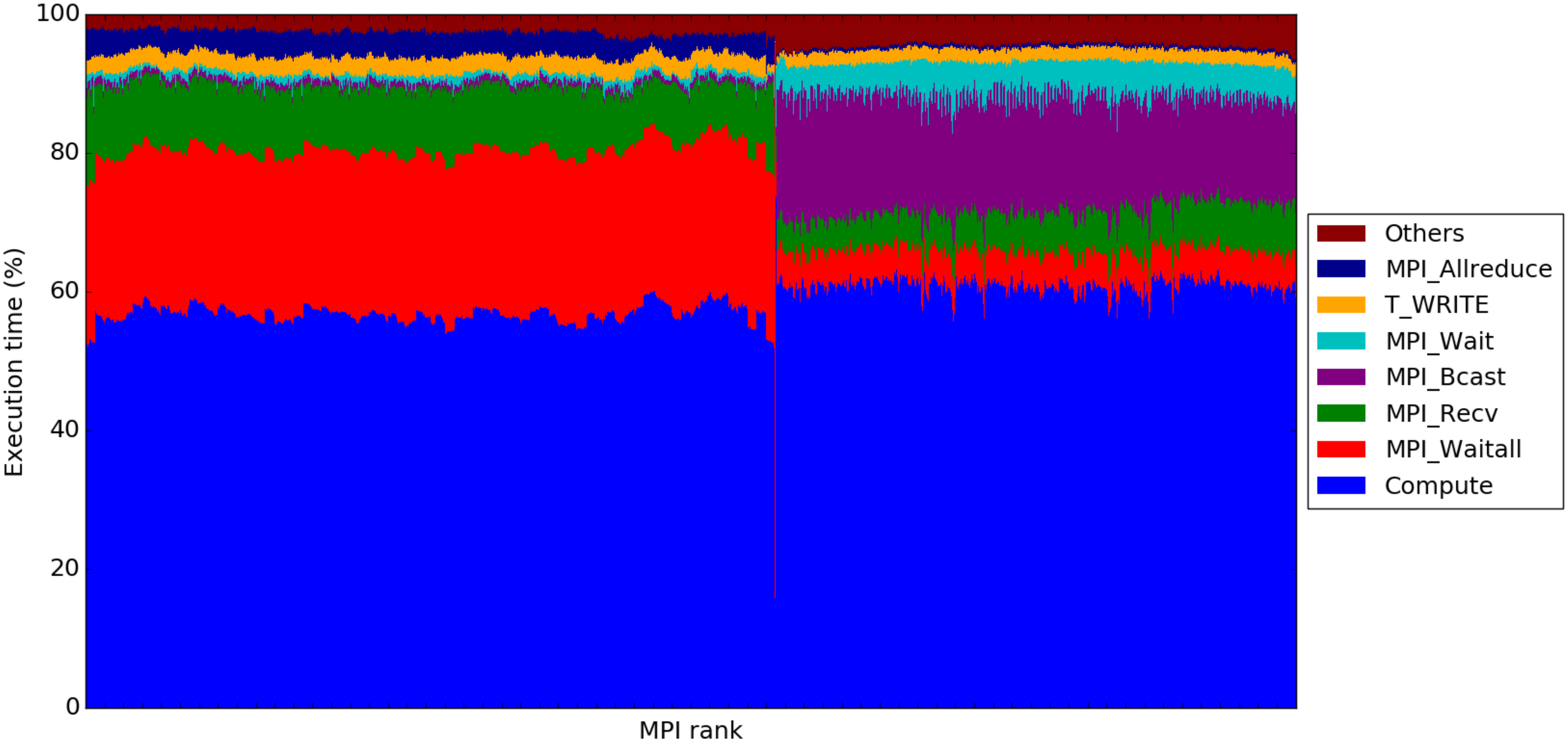


Bandwidth per node for a QCD application with a 4-d grid stencil communication pattern. Placing processes optimally makes all communications one-hop, increasing scalability.

SGI MPInside

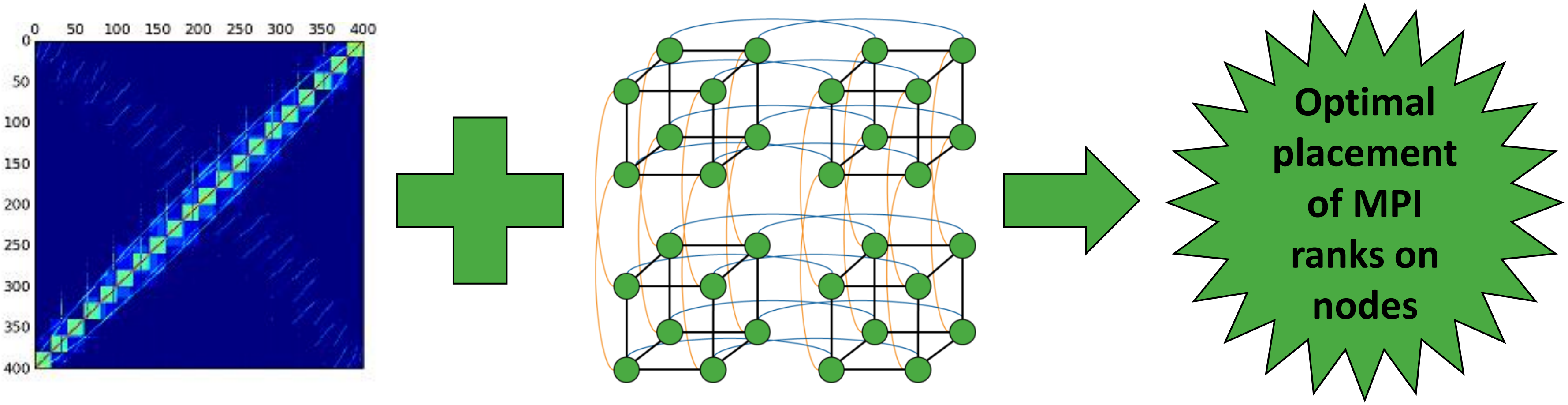
- **SGI's MPI profiling tool**
  - Useable with thousands of MPI ranks.
  - No need to re-compile or re-link.
  - Break down how much time application spends in each MPI routine.
  - Generate communications matrices to assist understanding of complex applications.
    - Amount of data transferred, number of requests, wait time.
  - Simple performance modelling.
    - Use virtual clocks to perform on-the-fly "what-if" experiments.
    - E.g. Investigate the impact of an interconnect with different performance characteristics.

MPInside Example Profile



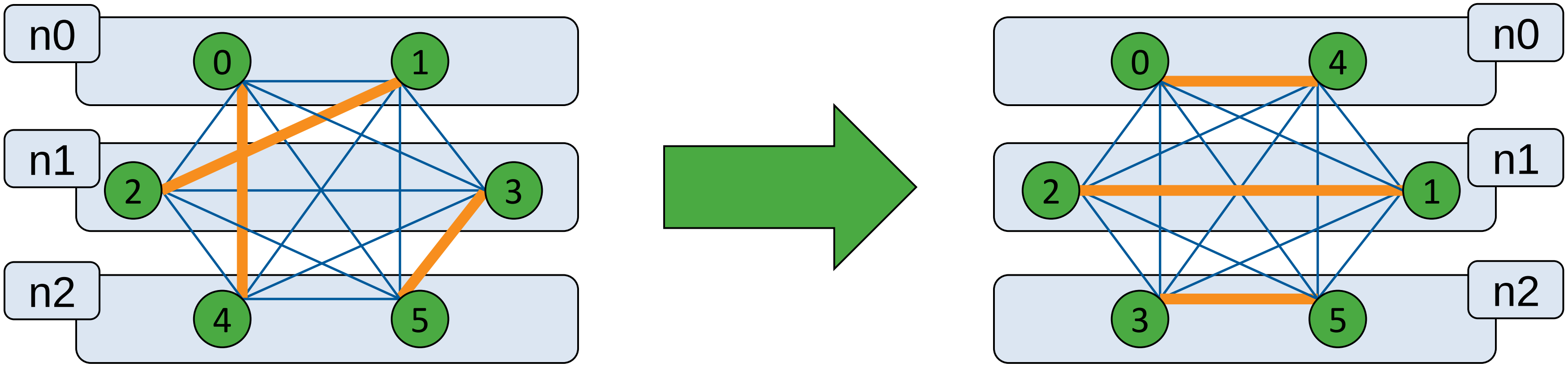
SGI MPIplace

- **A profile-guided placement tool for MPI**
  - Map MPI ranks to nodes using knowledge of underlying interconnect topology and MPInside communication matrix.
  - Minimize inter-node and inter-switch transfer costs.



A Toy Example

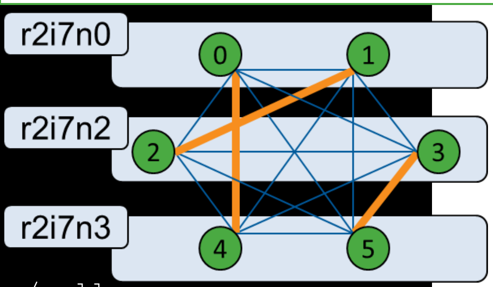
- Six MPI processes on three nodes (or on six nodes, three switches).
- Each rank sends some data to every other rank.
  - Some ranks send more data to others.
  - Three pair of “partners”.
- Optimal communication pattern is to have partners located on the same node (or switch).



Three Node Example

```
jsouthern@cy002: $ qsub -I -lselect=3:ncpus=56:mpiprocs=2
qsub: waiting for job 49679.cy002 to start
qsub: job 49679.cy002 ready
jsouthern@r2i7n0: $ mpiexec_mpt ./test | sort -k 7
MPI rank 0 runs on host r2i7n0
MPI rank 1 runs on host r2i7n0
MPI rank 2 runs on host r2i7n2
MPI rank 3 runs on host r2i7n2
MPI rank 4 runs on host r2i7n3
MPI rank 5 runs on host r2i7n3
jsouthern@r2i7n0: $ mpiexec_mpt MPInside ./test > /dev/null
MPInside 3.6.1 standard:
MPInside... Writing Reports. Please wait
jsouthern@r2i7n0: $ mpiexec_mpt ${PWD}/mpiplace_compute -n pbs.nodefile
-p MPINSIDE_MAT_DIR/ -o mpiplace.out -v
This job will use 3 hosts and 1 switches
jsouthern@r2i7n0: $ export MPI_WORLD_MAP="mpiplace.out"
jsouthern@r2i7n0: $ mpiexec_mpt ./test | sort -k 7
MPI rank 0 runs on host r2i7n0
MPI rank 4 runs on host r2i7n0
MPI rank 1 runs on host r2i7n2
MPI rank 2 runs on host r2i7n2
MPI rank 3 runs on host r2i7n3
MPI rank 5 runs on host r2i7n3
jsouthern@r2i7n0: $
```

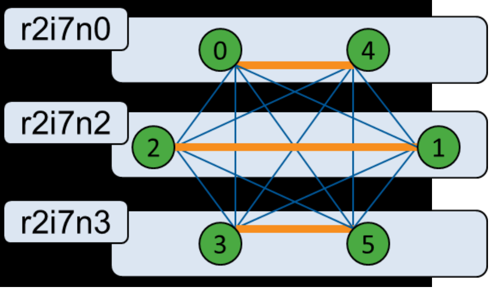
1. Run with default placement to get baseline.



2. Generate profiling data with MPInside.

3. Calculate optimal process placement with MPIplace.

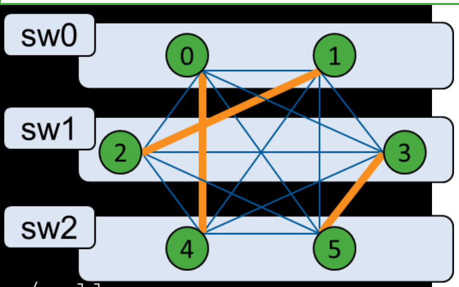
4. Run with optimal placement.



Three Node Example

```
jsouthern@cy002: $ qsub -I -lselect=36:ncpus=56:mpiprocs=1
qsub: waiting for job 49680.cy002 to start
qsub: job 49680.cy002 ready
jsouthern@r2i2n0: $ mpiexec_mpt ./test | sort -k 7
MPI rank 0 runs on host r2i0n0
MPI rank 1 runs on host r2i0n1
MPI rank 2 runs on host r2i0n5
MPI rank 3 runs on host r2i0n6
MPI rank 4 runs on host r2i0n18
MPI rank 5 runs on host r2i0n19
```

1. Run with default placement to get baseline.




```
jsouthern@r2i2n0: $ mpiexec_mpt MPInside ./test > /dev/null
MPInside 3.6.1 standard:
MPInside... Writing Reports. Please wait
```

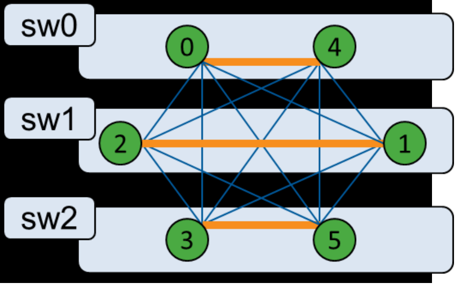
2. Generate profiling data with MPInside.

```
jsouthern@r2i2n0: $ mpiexec_mpt ${PWD}/mpiplace_compute -n pbs.nodefile
-p MPINSIDE_MAT_DIR/ -o mpiplace.out -v
This job will use 6 hosts and 3 switches
```

3. Calculate optimal process placement with MPIplace.

```
jsouthern@r2i2n0: $ export MPI_WORLD_MAP="mpiplace.out"
jsouthern@r2i2n0: $ mpiexec_mpt ./test | sort -k 7
MPI rank 0 runs on host r2i0n0
MPI rank 4 runs on host r2i0n1
MPI rank 2 runs on host r2i0n5
MPI rank 1 runs on host r2i0n6
MPI rank 5 runs on host r2i0n18
MPI rank 3 runs on host r2i0n19
jsouthern@r2i7n0: $
```

4. Run with optimal placement.



Real Weather Applications

- MPAS Atmosphere (MPAS-A):
  - Atmospheric component of the MPAS (Model for Prediction Across Scales) Earth system modelling package.
  - Flat MPI. Chose a problem size such that there was a reasonable amount of MPI time.
- IFS:
  - RAPS14 benchmark cases.
  - Hybrid MPI+OpenMP.
  - Chose benchmark case and number of MPI ranks so that there was a reasonable amount of MPI time when run on the benchmark system.

Benchmark System

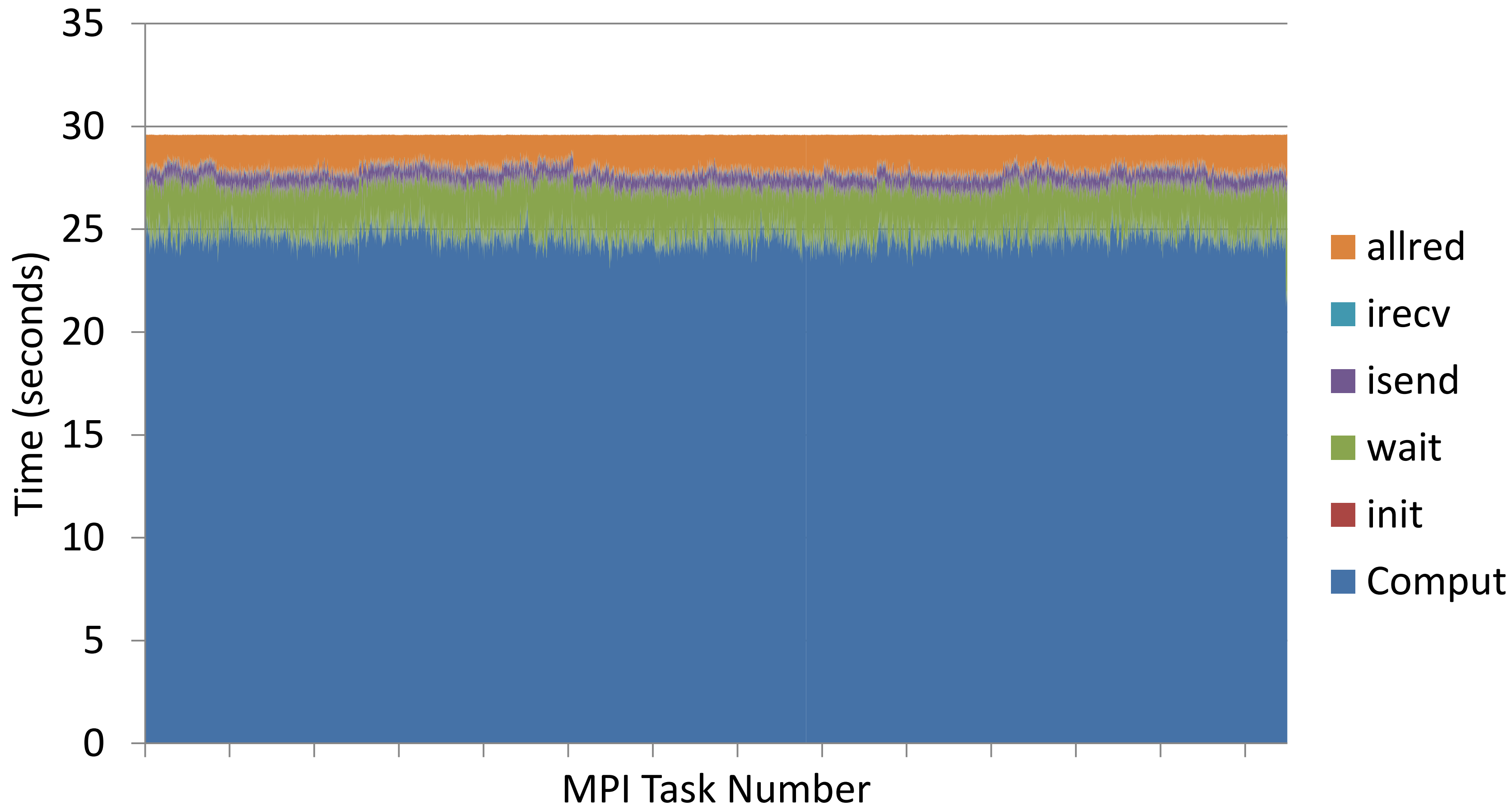
- SGI ICE XA.
  - 288 dual-socket compute nodes.
    - 2 x Intel E5-2690 v4 CPU (14 core, 2.6 GHz).
    - 128 GB memory.
  - 5D enhanced hypercube interconnect.
    - 4-4-4-4-4 topology.
    - EDR InfiniBand.
    - Dual plane.
    - Premium switch blades.
  - SUSE Linux Enterprise Server 11.3.
    - Intel compilers (version 16.0.3)
    - SGI MPT (version 2.14).
    - SGI Performance Suite.

15

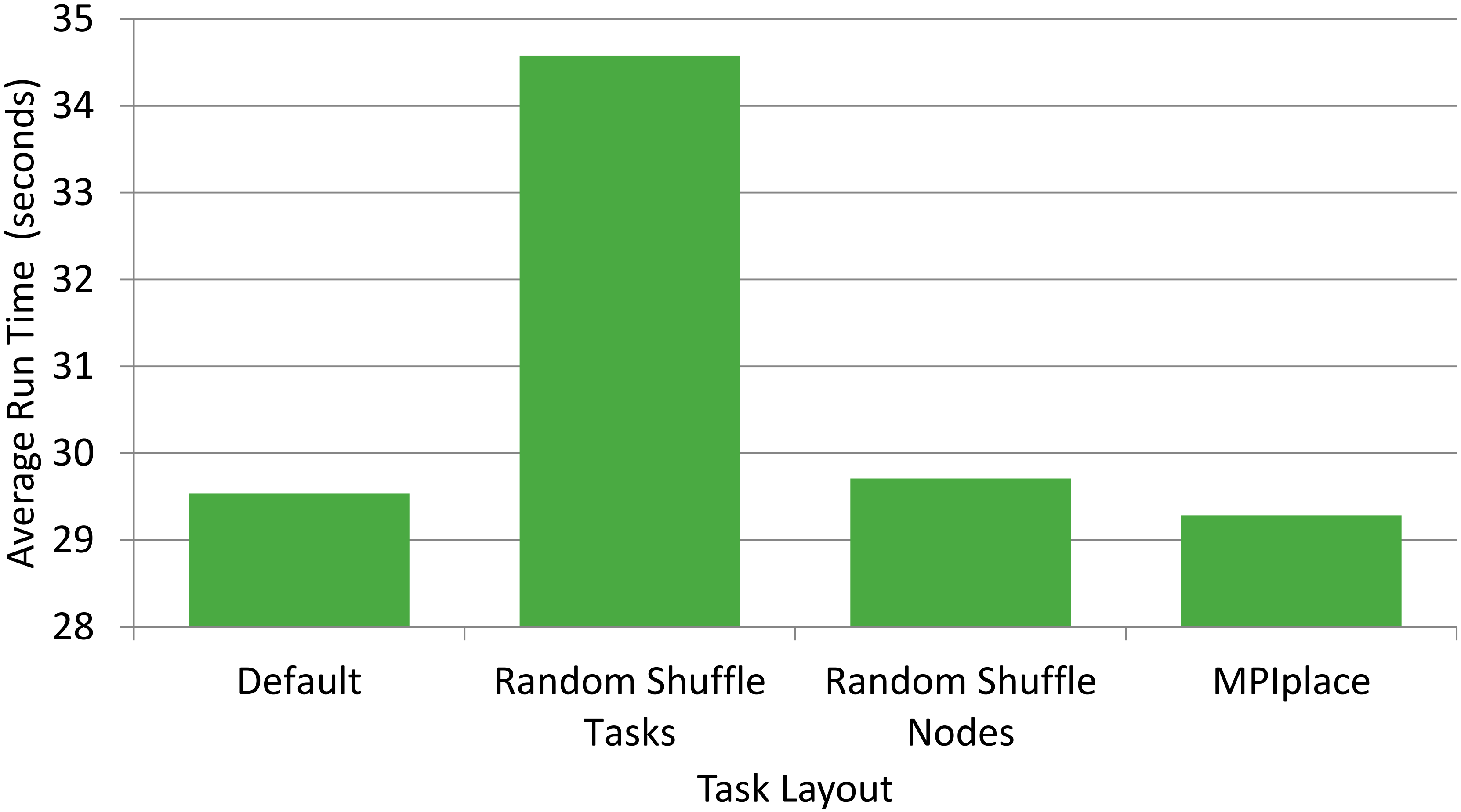
MPAS-A From NCAR

- Atmospheric component at 30 km.
- Exhibits good scalability – approximately 86% parallel efficiency at 6912 cores relative 1152 cores.
- 30 km resolution chosen so that we would see a noticeable amount of MPI time on the system we were using (8064 Broadwell cores). Experiments were performed using 6912 cores.
- Only the time integration is considered.
  - The simulation was for 3 hours and required 720 timesteps.
  - Typically a first timestep will take longer as pages of memory are allocated – we used a large enough number of timesteps to reduce that impact.

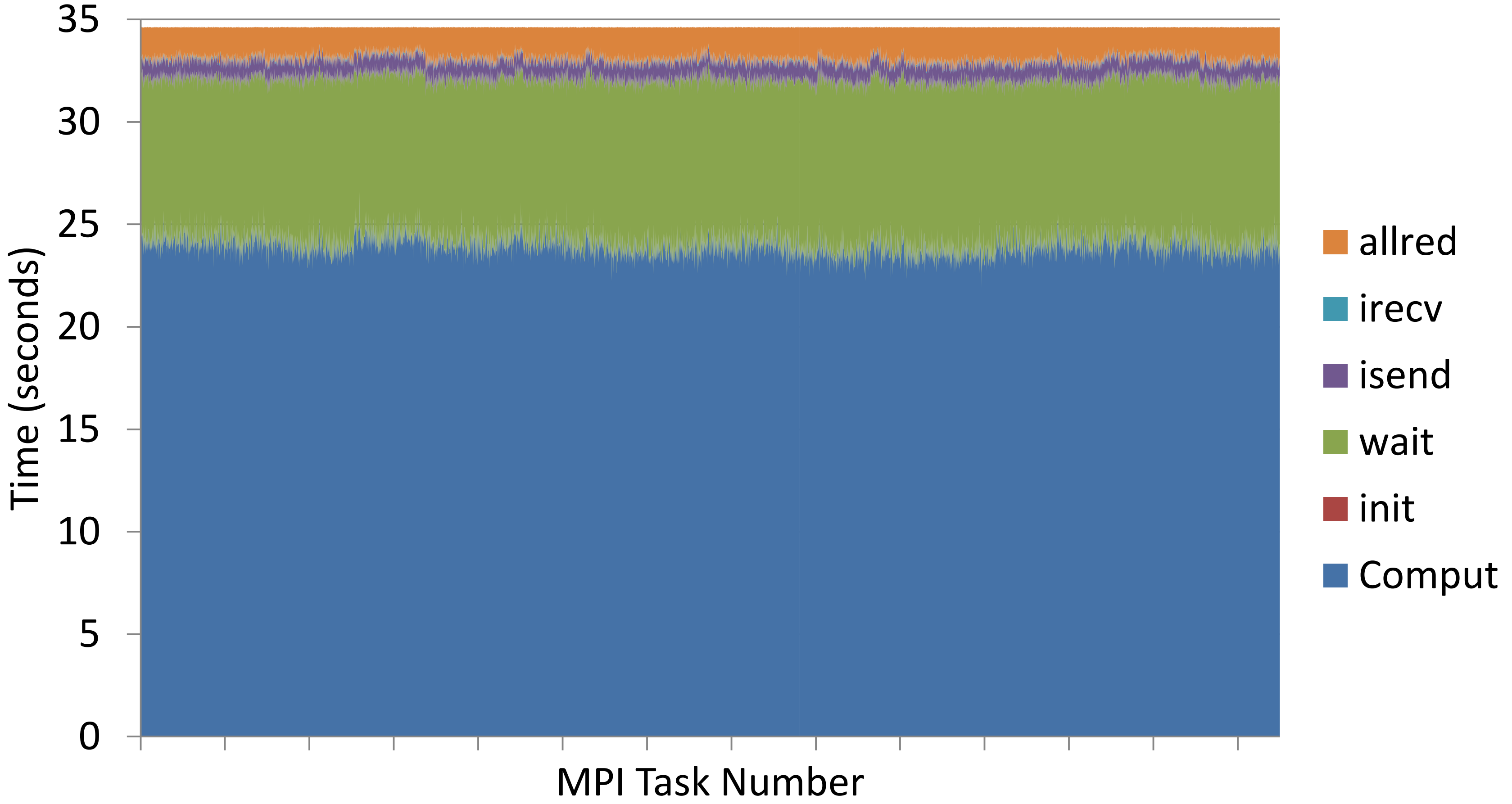
MPAS-A MPI Instrumentation: Default Task Layout



Performance of MPAS-A



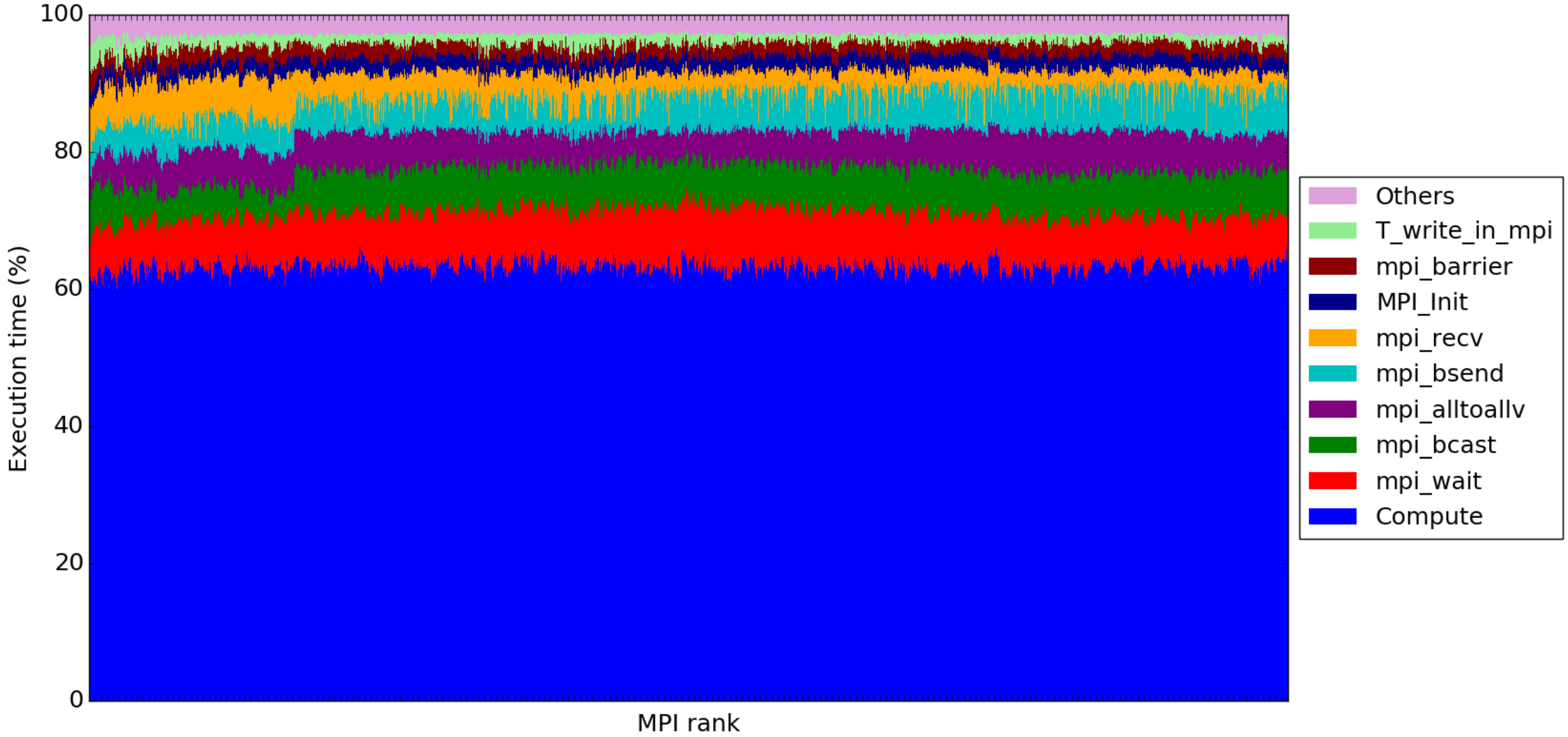
MPAS-A MPI Instrumentation: Random Shuffle of the Tasks



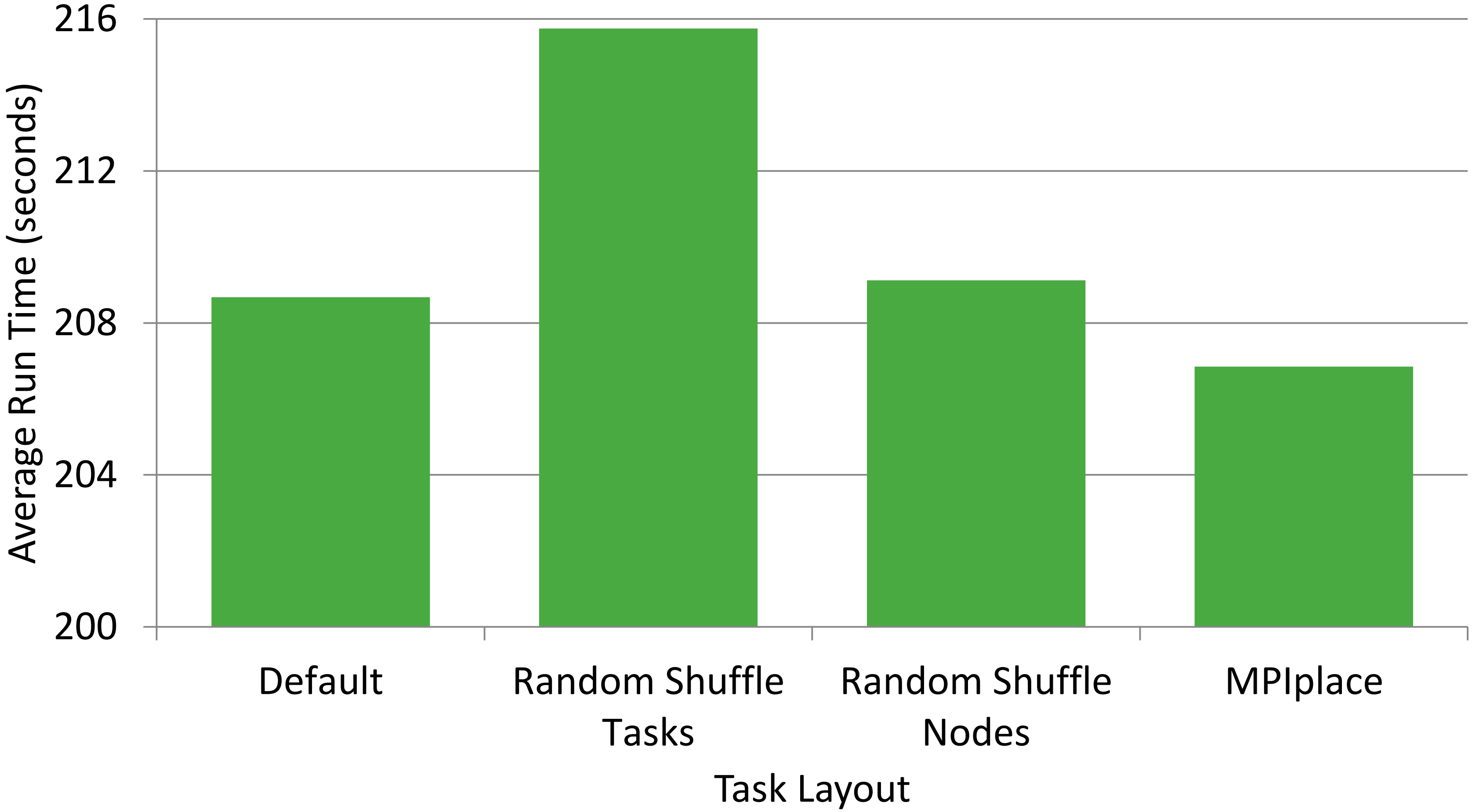
IFS Benchmark

- Ran the TCo639 dataset.
  - Scales relatively well to 200 nodes, but MPI time is beginning to grow.
  - Run with 28 MPI processes per node.
  - Hyper-threading is enabled, so two OpenMP threads per task.
- Use the "short" version of the benchmark.
  - Simulates two days forecast modelling.
  - Runs for approximately 210 seconds.
  - As for MPAS-A, first time step takes longer, we ran for long enough to reduce that impact.

IFS Instrumentation: Default Task Layout



Performance of IFS



Conclusions

- SGI provide tools (MPInside, MPIplace) to assist with optimal placement over relatively sparse topologies.
- Both MPAS-A and IFS showed around a 1% improvement in run-time when running the selected test cases.
  - In the case of MPAS-A (~15% MPI time) this improvement is clearly more than simply random fluctuations. Reduced MPI time by more than 5%.
  - IFS (~35% MPI time for this test case) shows limited sensitivity to process placement overall: a completely random process placement is <5% slower than optimal.
- On larger systems, with more MPI processes and more latency optimal process placement would be expected to become more important.

24