Load balance & rank placement

Motivation for load imbalance analysis

- **Increasing system software and architecture complexity**
  - Current trend in high end computing is to have systems with tens of thousands of processors
    - This is being accentuated with multi-core processors
- **Applications have to be very well balanced In order to perform at scale on these MPP systems**
  - Efficient application scaling includes a balanced use of requested computing resources
- **Desire to minimize computing resource "waste"**
  - Identify slower paths through code
  - Identify inefficient "stalls" within an application

Example load distribution

Imbalance time

- **Metric based on execution time**
- **It is dependent on the type of activity:**
  - User functions

    Imbalance time = Maximum time – Average time
  - Synchronization (Collective communication and barriers)

    Imbalance time = Average time – Minimum time
- **Identifies computational code regions and synchronization calls that could benefit most from load balance optimization**
- **Estimates how much overall program time could be saved if corresponding section of code had a perfect balance**
  - Represents upper bound on “potential savings”
  - Assumes other processes are waiting, not doing useful work while slowest member finishes

Imbalance %

$$\text{Imbalance\%} = 100 \times \frac{\text{Imbalance time}}{\text{Max Time}} \times \frac{N}{N - 1}$$

- **Represents % of resources available for parallelism that is "wasted"**
- **Corresponds to % of time that rest of team is not engaged in useful work on the given function**
- **Perfectly balanced code segment has imbalance of 0%**
- **Serial code segment has imbalance of 100%**

MPI sync time

- Measure load imbalance in programs instrumented to trace MPI functions to determine if MPI ranks arrive at collectives together
- Separates potential load imbalance from data transfer
- Sync times reported by default if MPI functions traced
- If desired, PAT_RT_MPI_SYNC=0 deactivates this feature

Causes and hints

- **Need CrayPAT reports: What is causing the load imbalance?**
- **Computation**
  - Is decomposition appropriate?
  - Would reordering ranks help?
- **Communication**
  - Is decomposition appropriate?
  - Would reordering ranks help?
  - Are receives pre-posted?
  - Any All-to-1 communication?
- **I/O – synchronous single-writer I/O will cause significant load imbalance already with a couple of MPI tasks**

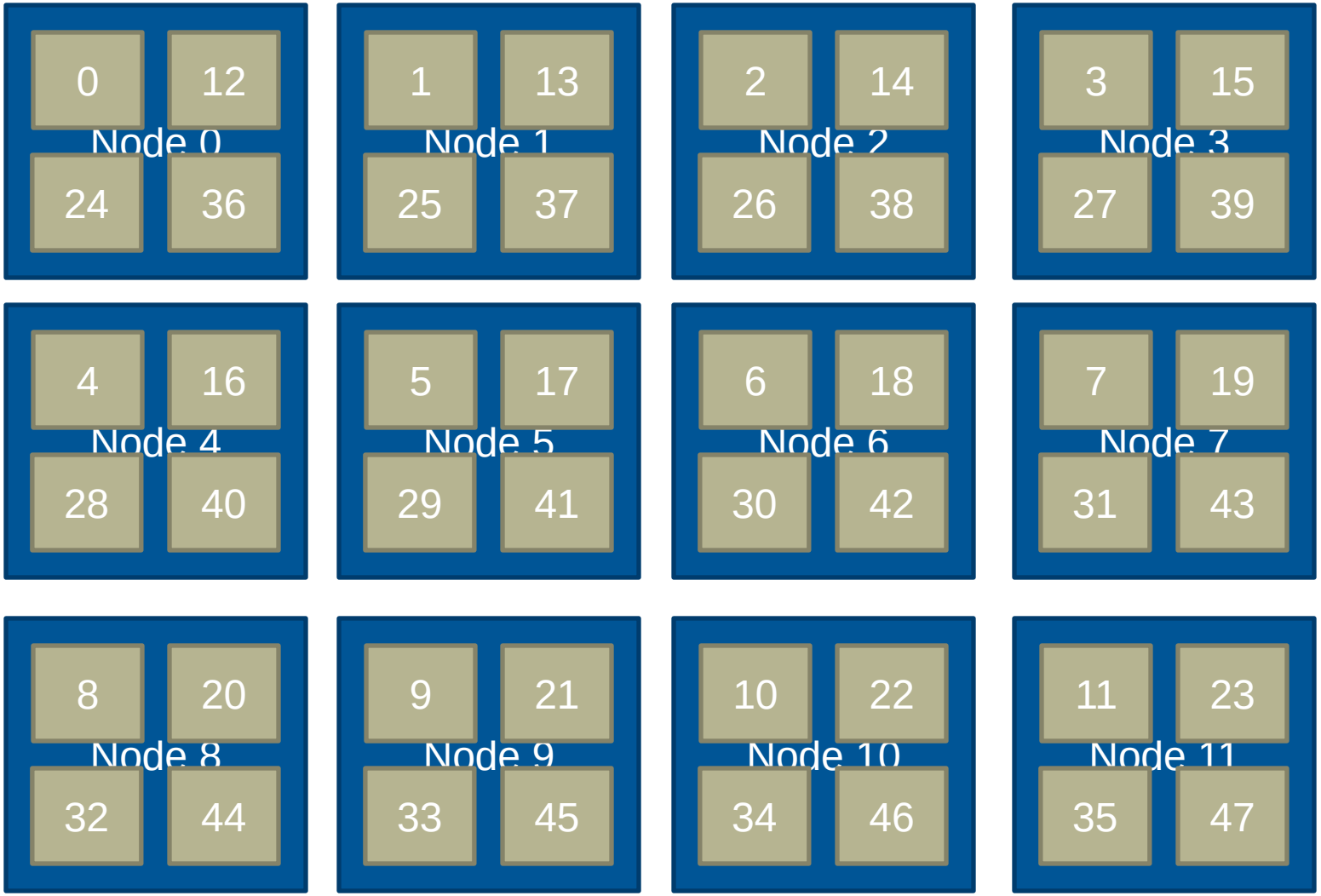
Rank Placement

- **The default ordering can be changed using the following environment variable:**
  - MPICH_RANK_REORDER_METHOD=n
- **These are the different values that you can set it to:**
  - 0: Round-robin placement – Sequential ranks are placed on the next node in the list. Placement starts over with the first node upon reaching the end of the list.
  - 1: (DEFAULT) SMP-style placement – Sequential ranks fill up each node before moving to the next.
  - 2: Folded rank placement – Similar to round-robin placement except that each pass over the node list is in the opposite direction of the previous pass.
  - 3: Custom ordering. The ordering is specified in a file named MPICH_RANK_ORDER.

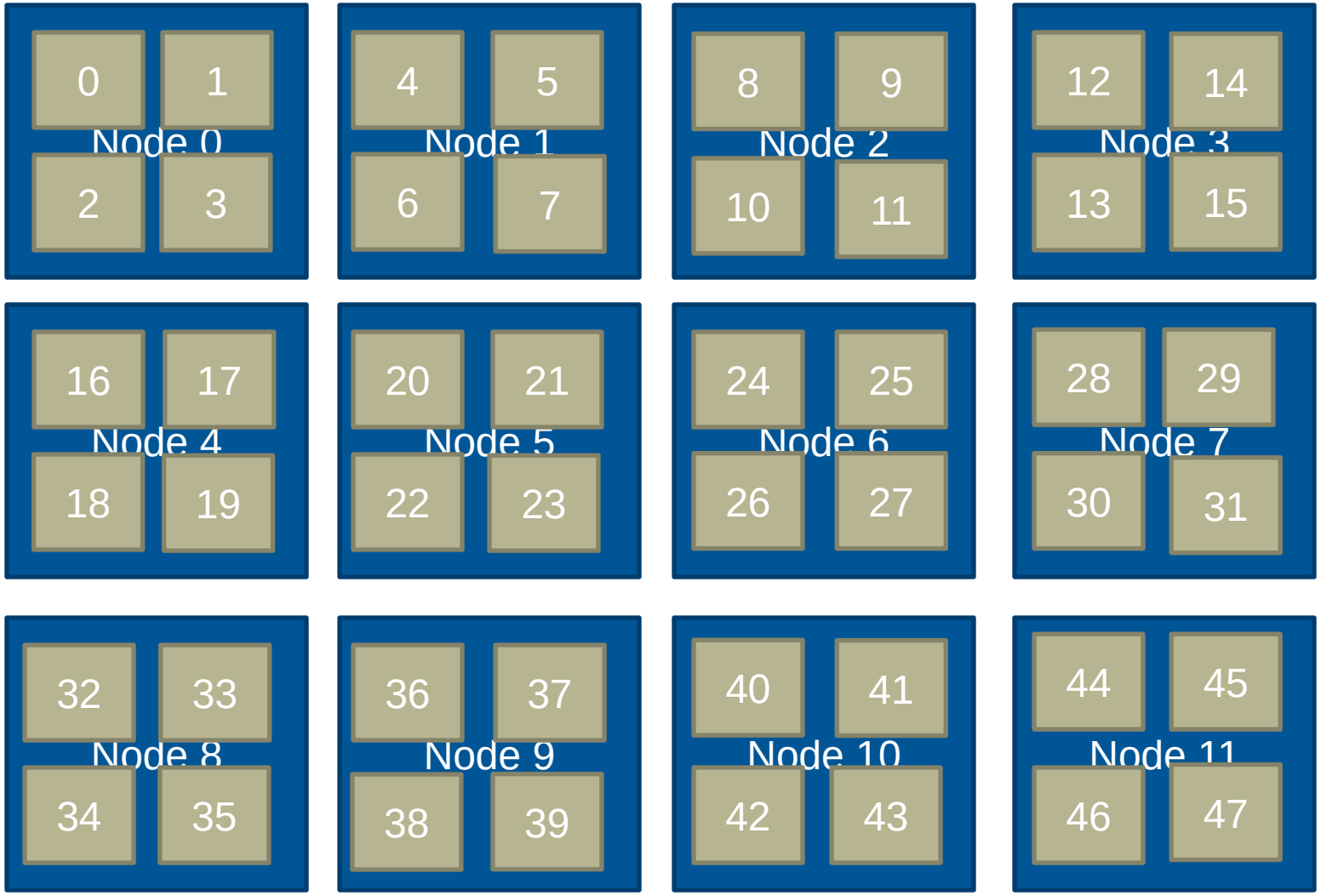
Rank Placement

- **When is this useful?**
  - Point-to-point communication consumes a significant fraction of program time and a load imbalance detected
  - Also shown to help for collectives (alltoall) on subcommunicators
  - Spread out IO across nodes

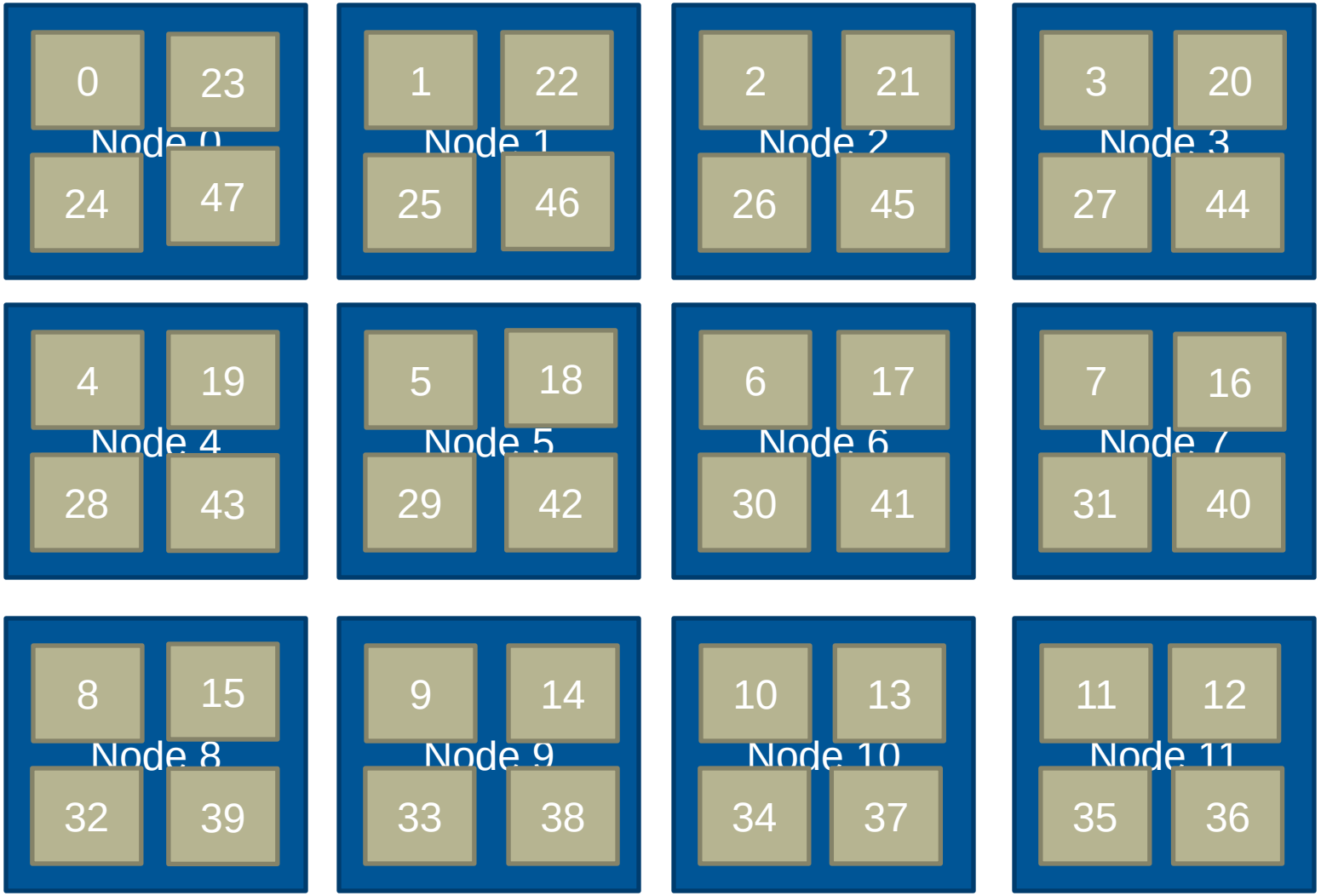
0: Round Robin Placement



1: SMP Placement (default)



2: Folded Placement



Rank Placement

- From the man page: The grid_order utility is used to generate a rank order list for use by an MPI application that uses communication between nearest neighbors in a grid. When executed with the desired arguments, grid_order generates rank order information in the appropriate format and writes it to stdout. This output can then be copied or written into a file named MPICH_RANK_ORDER and used with the

  MPICH_RANK_REORDER_METHOD=3

  environment variable to override the default MPI rank placement scheme and specify a custom rank placement.
- Craypat will also make suggestions

Rank placement

- **So easy to experiment with that it should be tested with every application…**
- **When is this a priori useful?**
  - Point-to-point communication consumes a significant fraction of program time and a load imbalance detected
  - Also shown to help for collectives (alltoall) on subcommunicators
  - Spread out I/O servers across nodes

Hybrid MPI + OpenMP?

- **OpenMP may help**
  - Able to spread workload with less overhead
  - Large amount of work to go from all-MPI to (better performing) hybrid - must accept challenge to hybridize large amount of code
- **When does it pay to add OpenMP to my MPI code?**
  - Add OpenMP when code is network bound
  - Adding OpenMP to memory bound codes may aggravate memory bandwidth issues, but you have more control when optimizing for cache
  - Look at collective time, excluding sync time: this goes up as network becomes a problem
  - Look at point-to-point wait times: if these go up, network may be a problem
  - If an all-to-all communication pattern becomes a bottleneck, hybridization often overcomes this
  - Hybridization can be used to avoid replicated data

OpenMP thread placement

- **When running a hybrid MPI+OpenMP application, the optimal number of threads/MPI task depends on the application and even input**
  - On the XE, one should try at least with 32x1, 16x2, perhaps also with 8x4, even 4x8 (MPI tasks x OpenMP threads per node)
- **The XE system is able to place OpenMP threads appropriately when the code is compiled with the Cray, PGI or GNU compiler**
  - Just do e.g. ”aprun -n 64 -d 32 -N 1 ./a.out” (for a 64x32=2048 core job)
  - You can use the aprun switch -S to force a certain number of MPI tasks per a numa node (=CPU) and -ss to have the threads to allocate memory only in the local numa node

Summary

- **Load imbalance is very often the very reason for non-scalability of an application**
- **It can be due to imbalanced computation or communication, with the usual suspects being**
  - Bad decomposition
  - All-to-one communication patterns
  - Single-writer I/O
- **Usually needs fixing at the source code level**
- **Some things for non-severe load imbalances can be done on the environment level: try to adjust the rank placement**
- **Hybrid MPI+OpenMP approach often useful for overcoming load balance problems**
  - Mind the thread placement when using hybrid codes!