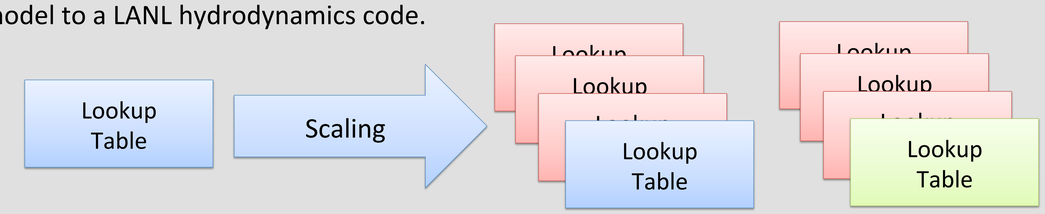
# Data Redundancy in HPC

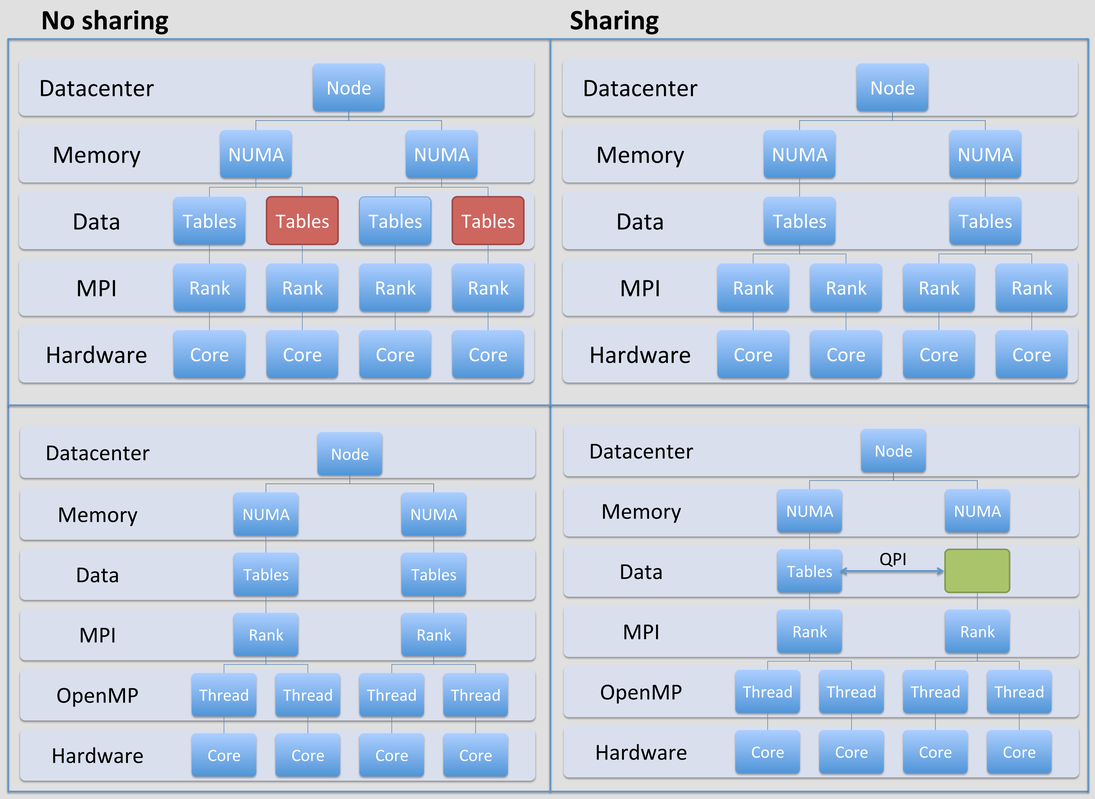
HPC codes often make use of lookup tables or other static data. When these codes are ran at scale, those lookup tables are often duplicated across various levels of parallelism. Sometimes, this redundancy is unavoidable because some parallel tasks do not share memory (like those on separate nodes). Other times, a lookup table is duplicated between parallel tasks that could share memory.

These redundancies can waste memory, limiting the granularity of the model. When there is excessive redundancy, programmers may disable some a machine's parallel capabilities in order to increase the available memory per task. This increases time-to-solution. We show how different programming models can effectively address this issue, and apply one such model to a LANL hydrodynamics code.




# Programming Models

New architectures demand new programming models. While MPI + X models reduce redundancy on their own, combining MPI + X models with shared memory can take advantage of memory hierarchy and increasingly faster NUMA interconnects/QPIs.

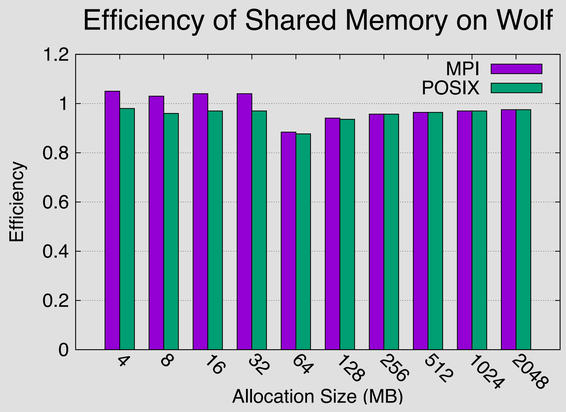



# Shmem Techniques

Two shared memory APIs were evaluated:

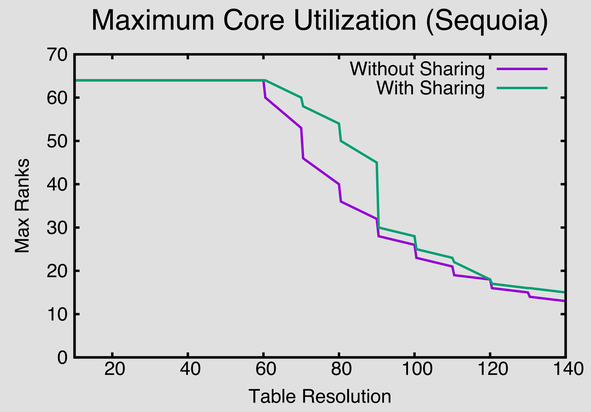
- MPI3's shared windows
- classic POSIX shared memory.

The graph below shows the ratio between observed and expected memory usage for a small test applications. Values above one can be attributed to a lack of granularity in the measurements.

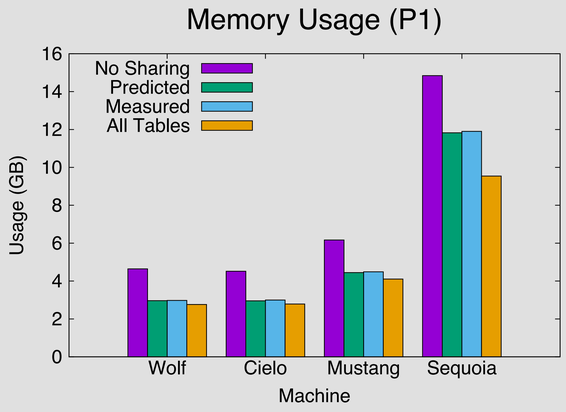



# Utilization

When a code's per-task memory footprint exceeds the machine's RAM-per-core, cores must go unused. The graph below shows maximum core utilization on Sequoia with and without shared memory. Increased core utilization decreases time-to-solution. New architectures with memory hierarchy will make the isolation of shared/static data even more important.




# Memory Savings




To the left, we show memory usage for a representative hydrodynamics problem. We predicted memory usage (green bar) by instrumenting the size of the shared allocations and subtracting from the original memory usage (blue bar). The cyan bar shows memory usage for the same code when two large lookup tables are shared. The orange bar represents our predicted memory usage if all lookup tables were shared.

The memory savings per node is larger for machines with more cores.

To the right, we show how our shared memory API scales across multiple nodes. Since memory is only shared within each node, the per-node memory usage should remain fairly constant (modulo MPI overhead and load balancing code). The optimal usage (blue bar) is calculated by subtracting the size of the shared region from the original. The POSIX measurements (green bar) show actual memory usage with the POSIX shared memory API. The "Off" measurements (cyan bar) represent memory usage when allocating one shared region per parallel task.