NumaPerf Design

Xin Zhao (ID:32216254)

October 9, 2020

SeriousScore

There are two kinds of cache miss: capacity cache miss and conflict cache miss.Since a cache miss leads to a main memory access, we can compute main memory access number by the following formular.
$MainMemoryAccessNumber = AccessNumber * CapacityCacheMissRate + CacheInvalidationNumber$

Since we only care about the absolute value of remote main memory access number, so
$RemoteMemoryAccessNumber = RemoteAccessNumber * CapacityCacheMissRate + RemoteCacheInvalidatio$

For serious score, we can not simply use the absolute value of remote memory access number without considering total running time or total number of memory accesses.You can consider a case that in application A, 98% of the codes are running sequentially and the left 2% are running parallelly, so the max improving percentage we can get by fix the NUMA issues is 2% , based on the consumption that NUMA problems usually happened in the parallel parts.That is why I use the following formular as the serious score.
$SeriousScore = RemoteMemoryAccessNumber/applicationRunningTime$

The problem of this formular is that we do not know the CapacityCacheMissRate, which heavily depends on applications and machines.So that I can just use a heuristic number. Currently the number is $\frac{1}{100}$.

Contributions Of NumaPerf

1. NumaPerf is machine in-dependent.
2. NumaPerf can detect NUMA issues caused by false sharing.
3. NumaPerf proves that thread binding is also a necessary strategy for NUMA issues.Besides, NumaPerf can detect whether an application should bind threads or not and how to do the binding.
4. NumaPerf could help to give advices about which strategy can be applied for each NUMA issue detected.

Strategies For NUMA Issues

1. Threads binding.
2. Memory duplication over nodes.
3. Interleaved memory.
4. Memory and threads co-location.
5. Splitting a big chunk memory to multiple pieces that each piece will only be accessed by one thread.
6. Strategies for False-Sharing problem like pending.

How NumaPerf Detect Which Strategy Should Be Applied

1. Thread binding.
   There are two situations that we will need thread binding:
   The first one is that threads frequently requiring locks, so that threads are highly possible to be shifted to another node, which turned local accessing to remote accessing.
   The second one is that thread based accessing is not balanced, so binding some threads together could reduce inter-node congestion.

   To detect the first situation, we will intercept all the lock functions, and record how many times each thread requires locks.If the number exceeds a threshold, threads binding will be needed.
   For the second situation, we will provide thread based accessing number, like for each thread, how many time it accesses memories from each other thread.So that we can compute the minimal remote access number and the maximal remote access number from all possible threads-nodes binding situations.If the maximal and minimal remote access numbers have a huge gap, applications could get some benefits from a proper thread binding.

2. Memory duplication over nodes.
   To get benefits from memory duplication, we have to confirm that remote invalidation number is super low, which means the memory is not copied by other threads when it is updated.So that duplication in this situation will not bring too much extra overheads.Meanwhile, accessing number of this memory from all other threads should be high enough, which gives enough reason to do duplication.

3. Interleaved memory.
   If access numbers of a allocated memory are almost even and super high for all threads, interleaved memory is a good strategy.But we should also notice the size of this memory, since only a big enough memory can be interleaved–more than $nodes * pagesize$.

4. Memory and threads co-location.
   The sign of this strategy is super clear. If the accessing is most from a certain thread, you can simply bind this memory with this thread.

5. Splitting a big chunk memory to multiple pieces that each piece will only be accessed by one thread.
   The sign of this is still very clear. If the access pattern of this memory is regular, you can use this strategy. More specifically, if each small part of this memory is only accessed by one thread, the memory can be splitted.