Parallel and Distributed Computing – Spring 2021

Momina Atif Dar

P18-0030

Intel Smart Cache

Intel Smart Cache was a technological advancement in the field of processors as it provided functionalities that are needed to this day and will be needed in the future as well.
Before Smart Cache the caches worked some other way. For example, there were 4 cores and every core had its dedicated cache, that core could only use their own cache. This way the cache was being underutilized and was limited for every core. No core could cache something more than its designated space. Smart Cache was revolutionary because it allowed the four caches to be adjustable according to the needs of cores. Let’s say if core 1 needed more cache than what it cache was offering than some other cache’s space was given to it. This was the needy cache’s space was increased and the giver cache’s space was decreased as it wasn’t using that space anyway. This solved the problem for needy cache and the other cache was also not underutilized.
This was a good step for improving the performance, efficiency and reducing latency. Now, more data could be put in cache without fetching it and no cache was being underutilized – win-win situation!

In Shared caches, there’s a cache (L2) that is shared among multiple cores for them to access the data at the same time.
Some of the prominent advantages of shared cache architecture are

- Shared cache is used efficiently as if one core is not using the shared cache then the other core can use it all. This way the shared cache can be used to its full and doesn’t face underutilization.
- More options for programmers as the data can be shared more easily via shared cache among threads that are running on different cores.
- Cache-coherency is reduced as the shared cache will contain the updated data to be used by different cores.

- Data redundancy is reduced as shared cache has to store some data only once for it to be accessible to multiple cores unlike in private cache architecture where same data needed by different cores had to be fetched and stored in their caches at the same time.
- Data fetching from system memory is reduced as data is effectively shared to allow data requests to be resolved at shared-level cache.

There are some techniques that allow to use caches effectively (for multi-core systems with shared cache architecture):

- **Processor Affinity:** the process to assign threads that need to use cache but don't need to share data to cores that are not sharing the cache and to assign threads that need to share data to cores that are sharing the cache. It is important to categorize and assign threads accordingly to avoid performance degradation.
- **Cache blocking:** in this technique the large data (mostly a large dataset) is broken down into blocks and stored in cache, this allows the smaller blocks to stay in cache for longer time period which results in less fetching from main memory. This comes handy when one wants to iterate over a large data.
- **Hold approach:** in this technique, the data in shared cache is updated only when it is necessary. Otherwise the data is put on 'hold' in dedicated caches of cores.
- **Delayed approach:** this technique improves cache hit rate by introducing delay in sending a signal. Thread 0 gets the data from memory and modifies it, shared it with shared-cache and then sends the signal – delayed signal – to Thread 1 to access it, when Thread 1 checks shared-cache for data it gets the data so cache hit. On the contrary, if Thread 0 sends signal right after modifying the data and before sharing it with shared-cache then Thread 1 may check shared-cache and find no data so cache miss.