Add cache and die affinity

Author	Owning-sig	Date	Status
ranchochen@tencent.com	sig-node	2021.5.1	Draft

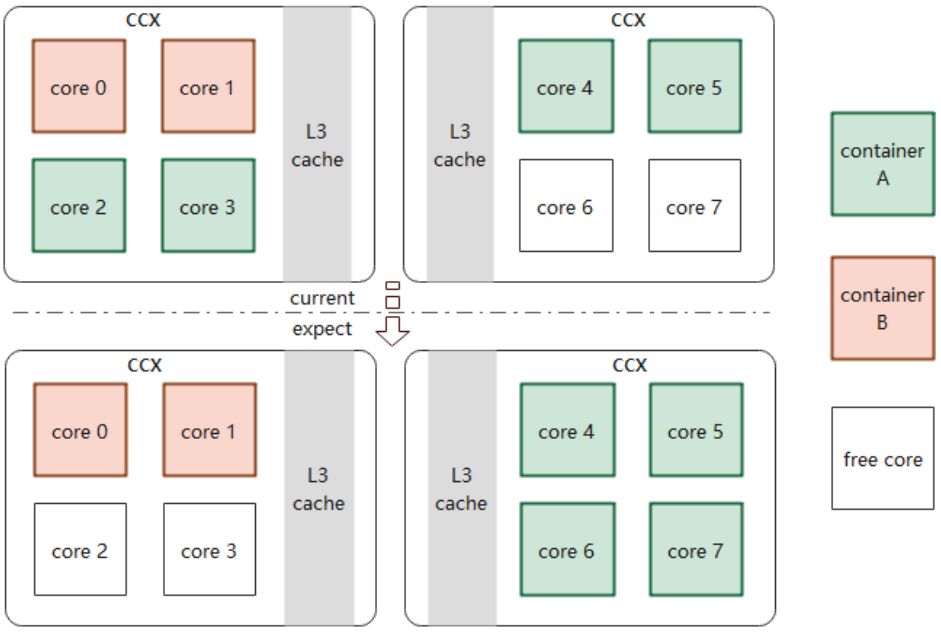
Summary

Caches are not considered in current Kubernetes cpu-manager, in some architectures, each socket/package owns more than one L3 cache, containers may encounter performance degradation for L3 cache interference and lower hit rate.

We propose to support for L3 cache affinity during container cpu allocation. While in the same package/socket, try to use cpus sharing L3 cache for container demand but not just choose from all cpus in the package/socket.

Motivation

Kubernetes cpu-manager tries to allocate cpus in the same core, socket/package, gaining better performance. In traditional architecture, L3 cache is shared between the whole socket, current cpus allocator works well.



However, the allocation algorithm may encounter problem in processors like `2nd Gen AMD EPYC™`, each `ccx`(a term used by AMD to describe a cluster of physical cores along with the shared L3 cache) owns its L3 cache, more than one L3 cache exists in a socket/package, we call L3 caches like this as uncore-cache all this design). Depending on current cpu allocation may face uncore-cache interference. For example, 4 cores with HT in ccx, a container demand for 8 cpus may not get the whole ccx, but get some cpus in other ccx(see figure below), container A and B may affect each other while the other flush uncore-cache. In our opinion, container's cpu locality should be considered.

Goals

Support uncore-cache affinity in cpu allocation in architecture.

Future work

Cross-die may also decrease process performance. We will add die affinity future, and corresponding cpu assignment algorithm implemetation.

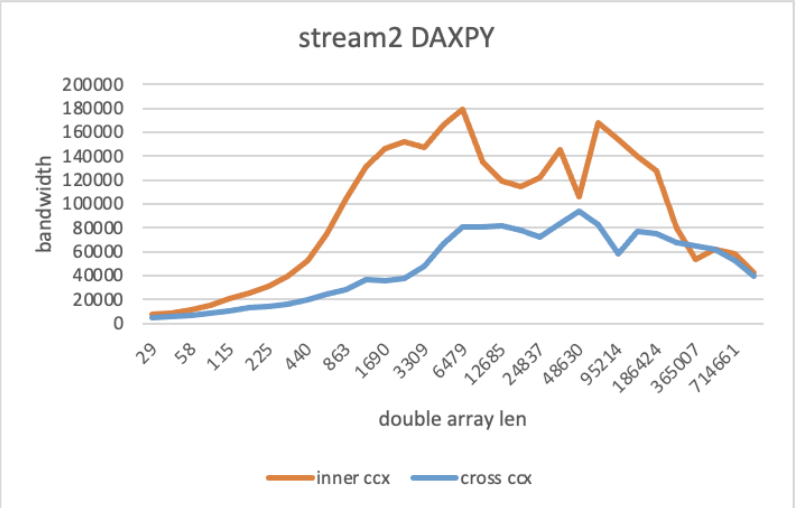
Proposal

In order to make a decision to allocate cpu with uncore-cache affinity, we should be aware of the uncore-cache information in kubelet, current kubelet gets cpu topology with cadvisor, which does not support the related details. So, we add cache id and uncore-cache items to cadvisor(all merged).

- Add cache id to cadvisor
  In cadvisor PR(https://github.com/google/cadvisor/pull/2847/), use /sys/devices/system/cpu/cpu*/cache/index3/id to get L3 cache id of current cpu, and store it as cpu topology.
- Add uncore cache to cadvisor
  In cadvisor PR(https://github.com/google/cadvisor/pull/2849), add L3 cache not shared among the whole socket(uncore cache) to core info in cpu topology. And we can get core->uncore-cache mappings.

User Stories (Optional)

Before change, when kubelet allocates cpus for containers, uncore-cache is not considered, and may get cpus across caches even there're free cpus shared uncore-caches.
Also, we make a bench with `stream2` DAXPY, as we can see, cross ccx(cross uncore-cache) gets lower bandwidth.



When workload is memory sensitive, this feature can improve memory bandwidth significantly(20% above).

Risks and Mitigations

- Currently no risks was found.
- Feature is enbled by a gate - a new kube feature with default false, potential risk effects could be limited.

Drawbacks

L3 cache affinity will not always get a better performance, however, we do think, workload in containers should not influence other containers. Decreasing L3 cache-miss in individual containers should be taken into consideration during programming workload or use other L3 cache allocation and isolation technology.

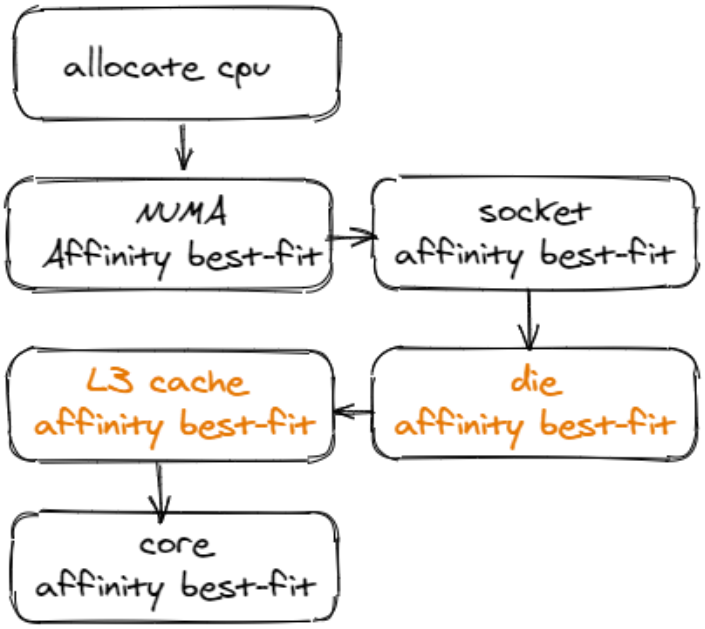
Design Details

- **Feature Gate**
  - Add CPUManagerLLCAlign to kubelet's feature-gates to enable(true)/disable(false) the feature.
  - Also, more than one l3 cache should exist in a single socket/package.
- **General Design**
  - **Logic Elaboration**

    Try to allocate cpus sharing the same cache if demand is larger than one core. Add L3 cache affinity before tring core affinity best-fit.
    If we cannot find llc-satisfied cpus, continue the original process(find available cores).

- **feature-gates CPUManagerLLCAlign**

  CPUManagerLLCAlign should set false in defaultKubernetesFeatureGates. And make a judge in takeByTopology, enable->(do l3 cache affinity best-fit),disable->(skip).



- Data Structure Design

pkg/kubelet/cm/cpumanager/topology/topology.go

```
type CPUTopology struct {
        NumCPUs            int
        NumCores    int
        NumSockets         int
        NumUnCoreCaches int
        CPUDetails   CPUDetails
}

// CPUInfo contains the socket and core IDs associated with a CPU.
 type CPUInfo struct {
        SocketID      int
```

```
        CoreID        int
        UnCoreCacheID int
}
```

- Core logic in cpu assignment

pkg/kubelet/cm/cpumanager/cpu_assignment.go

```
// Return free uncore cache IDs as a slice sorted by:
// - the number of whole available cores share the same uncore
cache, ascending
// - socket ID, ascending
// - uncore cache id, ascending
func (a *cpuAccumulator) freeUncoreCacheGroups() []int {
      uncoreCacheIDs := a.details.UncoreCaches().ToSlice()
      if len(uncoreCacheIDs) == 0 {
            return uncoreCacheIDs
      }
      sort.Slice(uncoreCacheIDs,
      func(i, j int) bool {
                  iUncoreCache := uncoreCacheIDs[i]
                  jUncoreCache := uncoreCacheIDs[j]
                  iCPUsInUncoreCache :=
a.details.CPUsInUncoreCaches(iUncoreCache).Filter(a.isUncoreCacheG
roupFree)
                  jCPUsInUncoreCache :=
a.details.CPUsInUncoreCaches(jUncoreCache).Filter(a.isUncoreCacheG
roupFree)

                  if iCPUsInUncoreCache.IsEmpty() ||
```

```
jCPUsInUncoreCache.IsEmpty() {
                            return iCPUsInUncoreCache.Size() <
jCPUsInUncoreCache.Size() || iUncoreCache < jUncoreCache
                } else {
                            iSocketID :=
a.details[iCPUsInUncoreCache.ToSlice()[0]].SocketID
                            jSocketID :=
a.details[jCPUsInUncoreCache.ToSlice()[0]].SocketID

                            return iCPUsInUncoreCache.Size() <
jCPUsInUncoreCache.Size() || iSocketID < jSocketID || iUncoreCache
< jUncoreCache
                }
            })
      return uncoreCacheIDs
}
```

```
func isUncoreCacheAlignEnabled() {
    return
utilfeature.DefaultFeatureGate.Enabled(kubefeatures.CPUManagerUnco
reCacheAlign)
}
```

Test Plan

Test should work on two scenarios:

- For AMD rome/milan or other architectures with more than one L3 cache in a socket, cpu allocation for a container should always try to get all demand cpus sharing one L3 cache. Check containers' cpuset.cpus for verification.
- For other architectures, cpu allocation should be the same as before.

Dependencies

High version cadvisor is in need, in which cache id and uncore cache info stored in cpu topology.