Yunming Zhang's Blog

NUMACTL notes and tutorial

Posted on July 22, 2015

This is a note summarizing the things I learned about the usage of NUMACTL. This feature can be potentially important for running parallel programs on NUMA architectures.

First, modern processors often take a NUMA (Non Uniform Memory Access) approach to hardware design. This post gives the best introduction to NUMA through comparisons with UMA and some good graphs.

https://software.intel.com/en-us/articles/optimizing-applications-for-numa

This post not only explains what NUMA is, but also the potential performance impact with migrating processes to different cores.

We sometimes want to control the way threads are assigned to cores for reasons including (1) want to use hardware threads and avoid using hyper threading (2) make sure my task doesn't migrate around frequently.

To do this, the linux operating system provides a function called NUMACTL, the documentation can be found here,

http://linux.die.net/man/8/numactl

NUMACTL gives the ability to control

1. NUMA scheduling policy
   A. for example, which cores do I want to run these tasks on
2. Memory placement policy
   A. where to allocate data

To quickly get started, you should try use

numactl –show

numactl –hardware

to checkout the NUMA architecture of your system

To understand the output, I followed this post here https://www.sharcnet.ca/help/index.php/Using_numactl

It is pretty helpful in explaining what the numbers mean. Here, I just want to make a note that if hyper threading is enabled, then you might see 48 cores, but it is actually just 24 physical cores. For more info, look up how to read /proc/cpuinfo

For me the output on Lanka compute node, which has 24 physical cores, 4 virtual cores, 2 sockets

yunming@lanka02: numactl -show

policy: default

preferred node: current

physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35

36 37 38 39 40 41 42 43 44 45 46 47

cpubind: 0 1

nodebind: 0 1

membind: 0 1

yunming@lanka02: numactl –hardware

available: 2 nodes (0-1)

node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 24 25 26 27 28 29 30 31 32 33 34 35

node 0 size: 64398 MB

node 0 free: 43311 MB

node 1 cpus: 12 13 14 15 16 17 18 19 20 21 22 23 36 37 38 39 40 41 42 43 44 45 46 47

node 1 size: 64508 MB

node 1 free: 46774 MB

node distances:

node   0   1

0: 10 21

1: 21 10

Follow

Follow "Yunming Zhang's Blog"

Get every new post delivered to your Inbox.

Enter your email address

Sign me up

Build a website with WordPress.com

This just shows that I have two memory groups, each with 64GB of memory, I have a total of 48 virtual cores. As documented in **–physcpubind**, this accepts cpu numbers as shown in the processor fields of **/proc/cpuinfo.** Thus, if hyper threading is enabled, the virtual cores representing a hyper thread, not a hardware thread will show up as well.

Now, the man page http://linux.die.net/man/8/numactl already shows some examples of using numactl such as –interleave, –physcpubind. I just want to go through a few additional examples,

One of them is from Comp 422 class back at Rice University

> numactl –localalloc –
> physcpubind=0,4,8,12,16,20,24,28,32,36,40,44,48,52,56,60,64,68,72,76,80,84,88,92,96,100,104,108,112,116,120,1

What this is doing is to use only physical threads and avoid using hyper threads. This works becausee as noted in the page, each node on BIOU (the cluster this assignment is suppose to run) has 128 threads, but it only has 32 cores. This implies that each core can use 4 threads and we want to avoid using more than 1 thread per core. Thus, we use physcpubind to force the system to allocate tasks to only hardware threads.

–localalloc is also useful because it makes sure each thread allocated on its own node.

One other example I used myself is the following to play with membind

numactl –physcpubind=0 –membind=0 ./pagerankCsr.o mediumGraph

This binds the memory allocation to the first memory group, in my case, socket 0, and using **MEMORY_BW_READS** described in my previous post https://yunmingzhang.wordpress.com/2015/07/22/measure-memory-bandwidth-using-uncore-counters/

This command gives the following output

Performance counter stats for 'system wide':

S0 1 12,955,877 uncore_imc_0/event=0x4,umask=0x3/

S0 1 14,013,046 uncore_imc_1/event=0x4,umask=0x3/

S0 1 12,935,697 uncore_imc_4/event=0x4,umask=0x3/

S0 1 14,031,470 uncore_imc_5/event=0x4,umask=0x3/

S1 1 17,244 uncore_imc_0/event=0x4,umask=0x3/

S1 1 1,142,200 uncore_imc_1/event=0x4,umask=0x3/

S1 1 14,080 uncore_imc_4/event=0x4,umask=0x3/

S1 1 1,147,476 uncore_imc_5/event=0x4,umask=0x3/

Follow

Follow "Yunming Zhang's Blog"

Get every new post delivered to your Inbox.

Enter your email address

Sign me up

Build a website with WordPress.com

notice how most of the cache line replacements are coming in in socket 0 as highlighted in bold.

numactl –physcpubind=0 –localalloc ./pagerankCsr.o mediumGraph

This command gives a similar output because we first bind cpu to 0 and 0 is in memory node 0 (socket 0) , the output looks like the following

S0 1 12,978,178 uncore_imc_0/event=0x4,umask=0x3/

S0 1 14,037,751 uncore_imc_1/event=0x4,umask=0x3/

S0 1 12,953,229 uncore_imc_4/event=0x4,umask=0x3/

S0 1 14,052,675 uncore_imc_5/event=0x4,umask=0x3/

S1 1 16,075 uncore_imc_0/event=0x4,umask=0x3/

S1 1 1,144,211 uncore_imc_1/event=0x4,umask=0x3/

S1 1 13,020 uncore_imc_4/event=0x4,umask=0x3/

S1 1 1,148,139 uncore_imc_5/event=0x4,umask=0x3/

Now we try to force the program to allocate on memory node 1

numactl –physcpubind=0 –membind=1 ./pagerankCsr.o mediumGraph

This gives the following output

S0 1 97,904 uncore_imc_0/event=0x4,umask=0x3/

```
S0      1       1,966,565 uncore_imc_1/event=0x4,umask=0x3/

S0      1          88,200 uncore_imc_4/event=0x4,umask=0x3/

S0      1       1,984,923 uncore_imc_5/event=0x4,umask=0x3/
```

S1 1 12,906,166 uncore_imc_0/event=0x4,umask=0x3/

S1 1 14,713,862 uncore_imc_1/event=0x4,umask=0x3/

S1 1 12,885,845 uncore_imc_4/event=0x4,umask=0x3/

S1 1 14,715,551 uncore_imc_5/event=0x4,umask=0x3/

As we can see, now most of the cacheline brought in are from socket 1.

To make matters worse, this particular scenarios demonstrates the importance of having the the data locally, as the running time jumps from **2.08s to 3.47s (1.6x slow down)**, showin accessing remote memory.

Follow

Follow "Yunming Zhang's Blog"

Get every new post delivered to your Inbox.

Enter your email address

Sign me up

Build a website with WordPress.com

Now this is going to be even more important for performance engineering on parallel programs.

Hope this tutorial helps!

About these ads

ESTĒE LAUDER
Join us for a makeover event by 3 Global Makeup Artists!
6th July (Wed), 8th July (Fri) & 9th July (Sat) 2016
Registration fee: $80/pax – Fully redeemable for products
Makeover sessions are every hour from 10am – 7pm.
Find Out More

Share this:

Twitter Facebook

Like

Be the first to like this.

Related

How to use srun to get an interactive node
In "Tools"

Simple ways for measuring load imbalance
In "Tools"

Reimplementing HJ-Mapper using pure Java concurrent library
In "Habanero Java"

This entry was posted in Tools and tagged tools. Bookmark the permalink.

Yunming Zhang's Blog

The Twenty Ten Theme. Create a free website or blog at WordPress.com.