

stream四种测试都差不多；

但是确实没有多线程测试的细节；

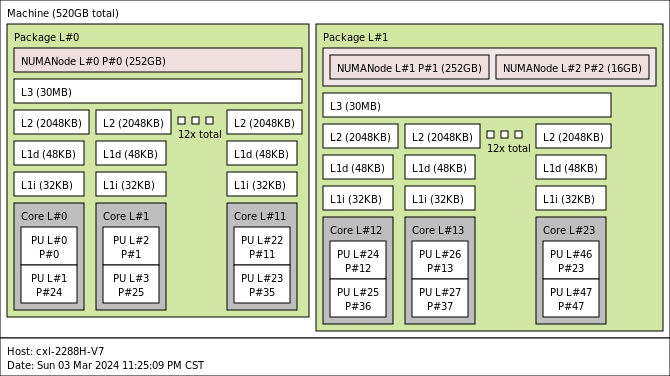
现在看到的大多数论文并没有体现出多线程访问后性能下降的问题，有一个就是考虑多线程访问CXL memory达到CXL接口的带宽上限之后，就保持不变，不会出现下降的情况。但是现在如何模拟多个host访问一个设备的性能测量呢？

[~~https://blog.yufeng.info/archives/1511~~](https://blog.yufeng.info/archives/1511)

~~详解服务器内存带宽计算和使用情况测量~~

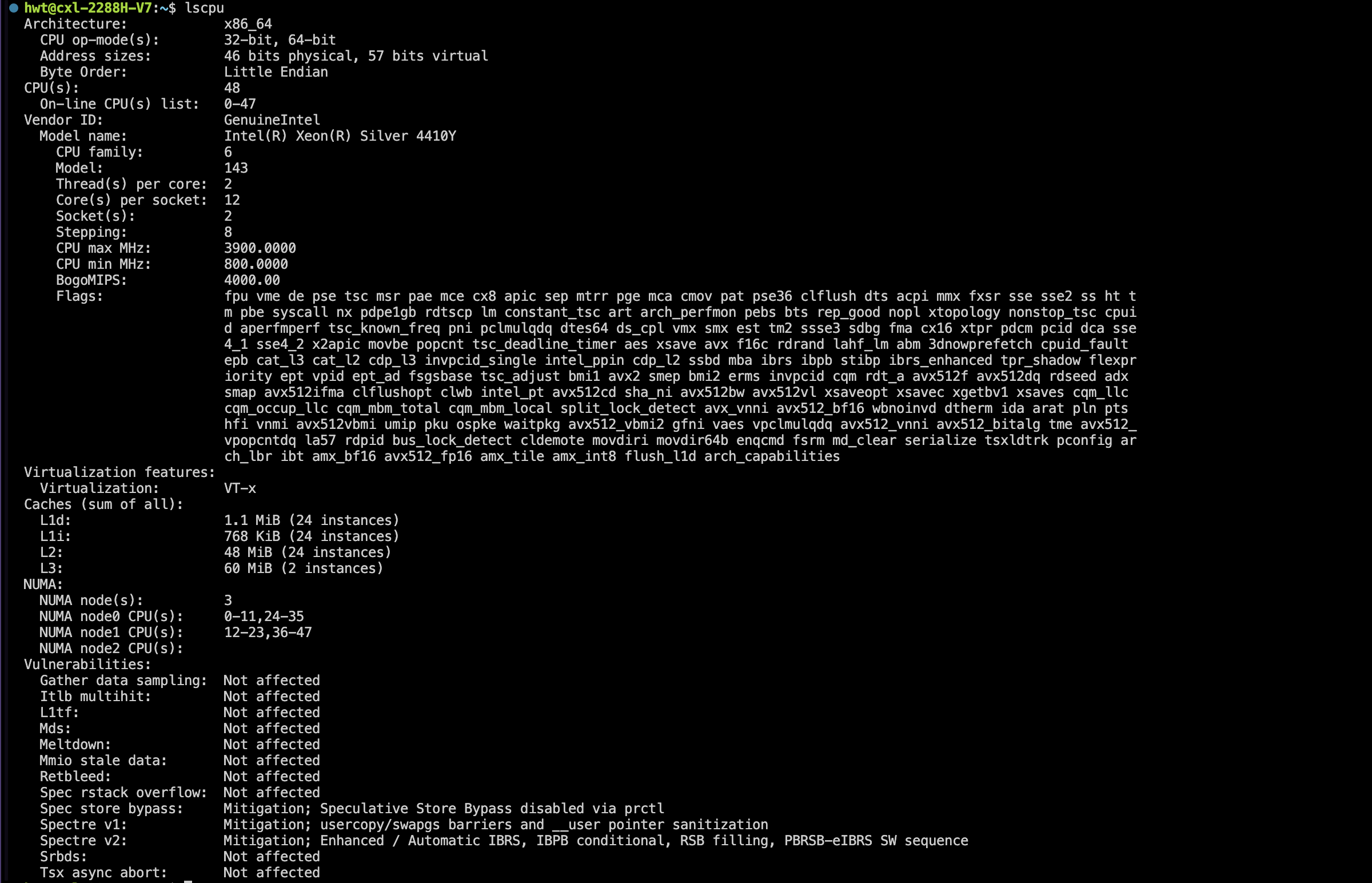
在Linux下CPU的使用情况有top工具, IO设备的使用情况有iostat工具，就是没有内存使用情况的测量工具。 我们可以看到大量的memcpy和字符串拷贝（可以用systemtap来测量），但是像简单的数据移动操作就无法统计，我们希望在**硬件层面**有办法可以查到**CPU在过去的一段时间内总共对主存系统发起了多少读写字节数**。

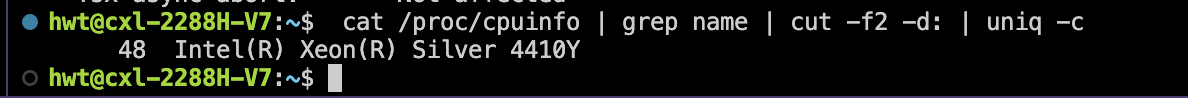
ubuntu查看当前v7服务器配置



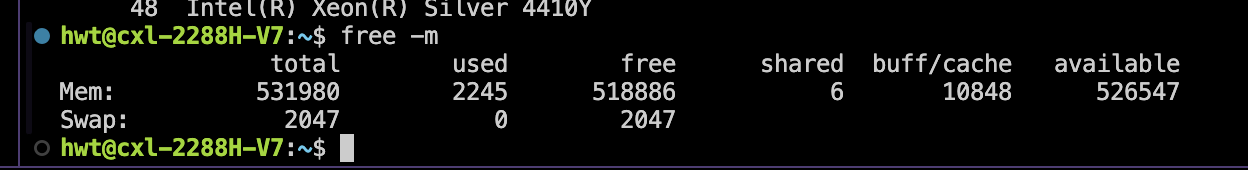
但是不知道现在的Intel I-series Agilex FPGA接到的DDR4内存控制器是几通道的？

查看CPU型号lscpu





查看内存情况 free -m

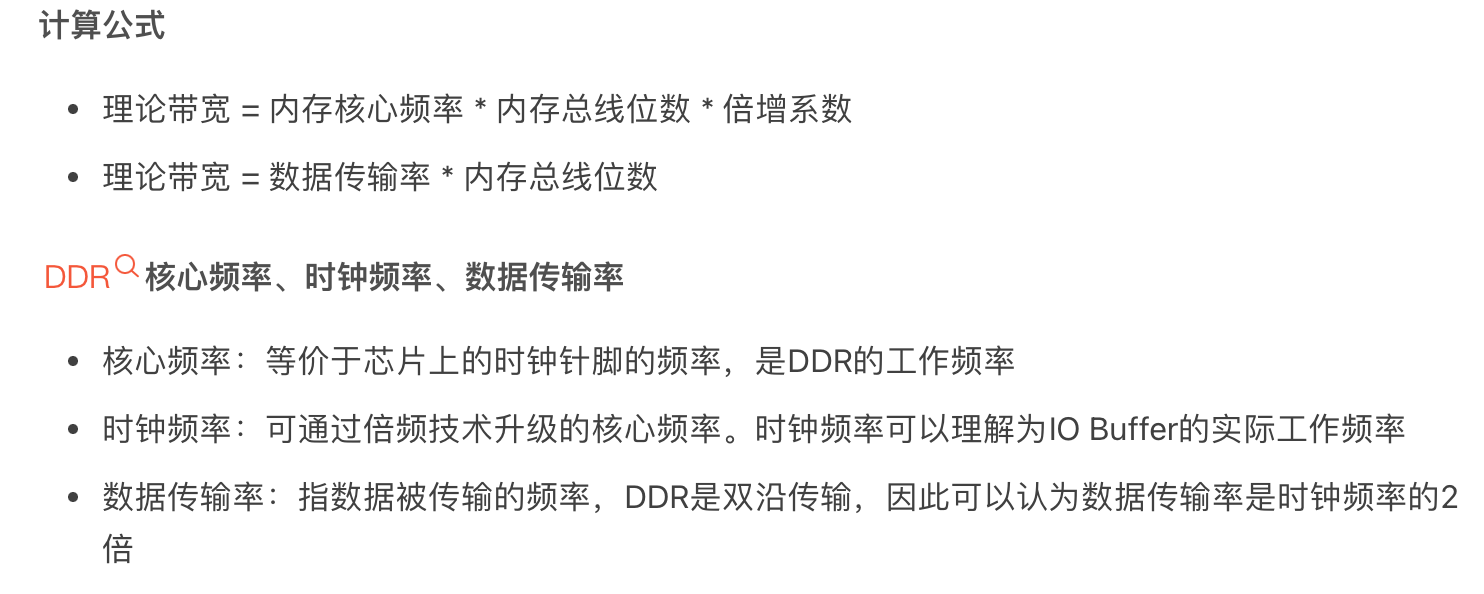


现有为numa架构，每个CPU都有自己的内存控制器直接连接到内存去，而且有8个通道，CPU直接通过QPI连接，内存控制器和QPI上面都会流动数据。

服务器使用的是DDR5内存，FPGA上使用的是DDR4内存。

需要计算在这样架构下的内存带宽。

内存条带宽如何计算？



内存带宽计算公式=内存核心频率x内存总线位数x倍增系数

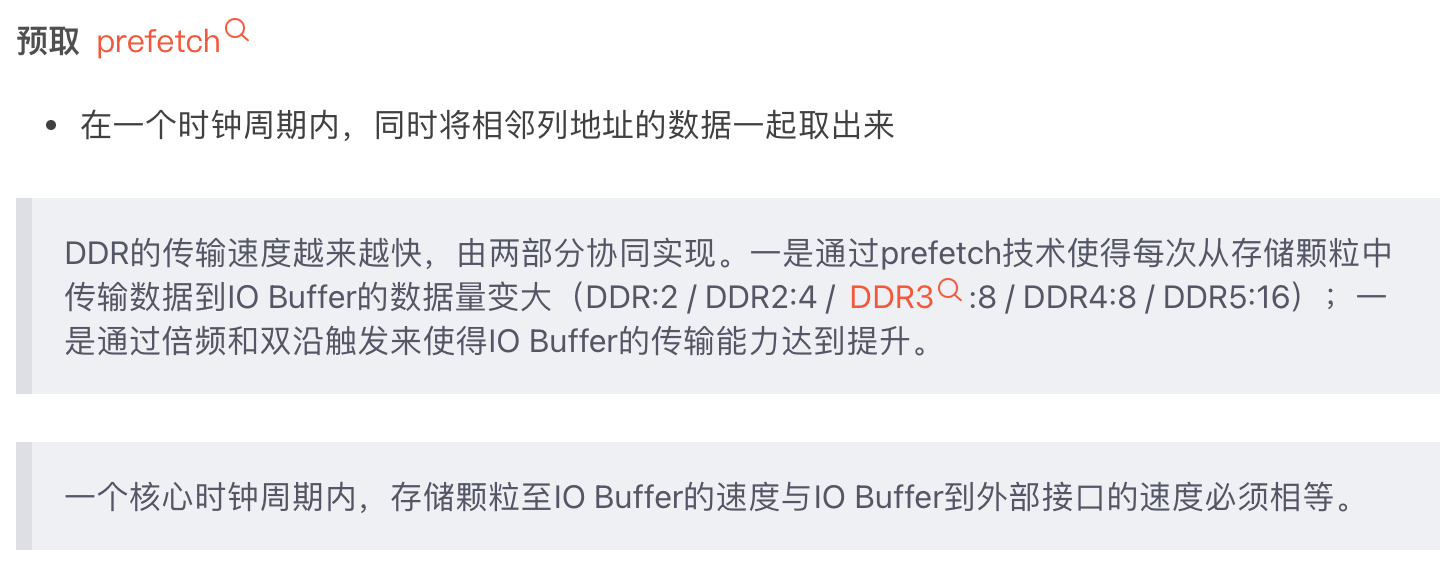
内存频率分为3种：内存核心频率，时钟频率（I/O Buffer的传输频率），有效数据传输频率；核心频率是内存Cell阵列（Memory Cell Array）的工作频率，是内存的真实运行频率

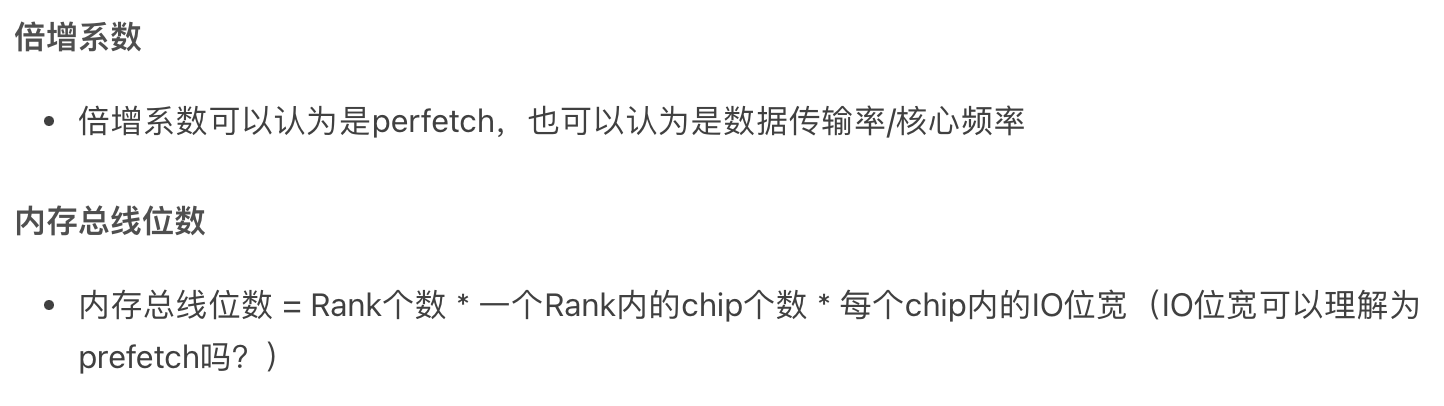
DDR1采用时钟脉冲上升、下降沿各传一次数据，1个时钟信号可以传输2倍于SDRAM的数据，所以DDR也称为双倍SDRAM，倍增系数=2；

DDR2一次预读4bit数据，是DDR一次预读2bit的2倍，倍增系数=4；

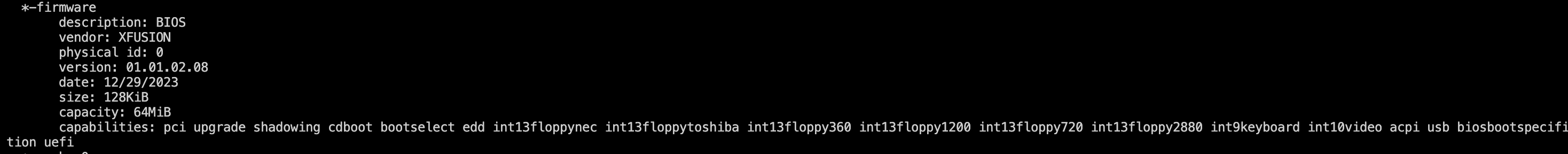
DDR3 8bit，倍增系数=8；

有效数据传输频率则是指数据传送的频率。DDR3内存一次从存储单元预取8Bit的数据，在I/OBuffer（输入/输出缓存）上升和下降中同时传输，因此有效的数据传输频率达到了存储单元核心频率的8倍。同时DDR3内存的时钟频率提高到了存储单元核心的4倍。（也就是先预取8bit，后面上升沿和下降沿必须把这8bit数据传完）





Memory内部结构？





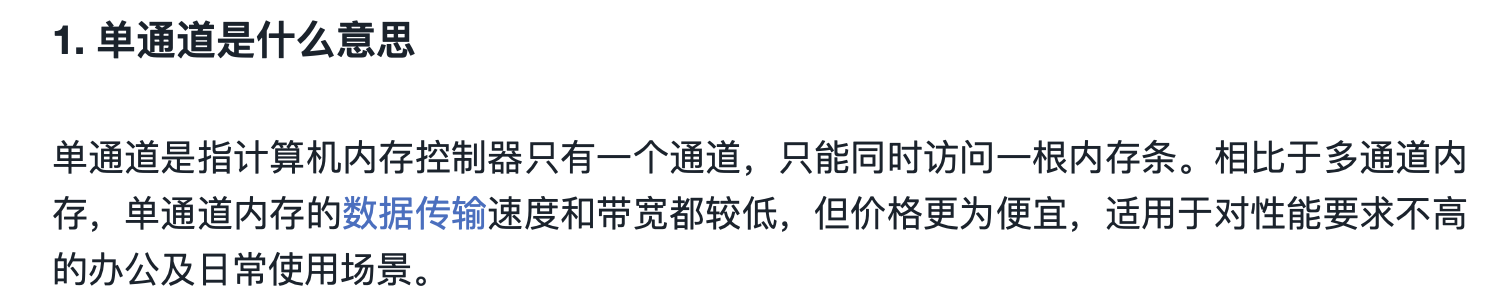
共32个bank

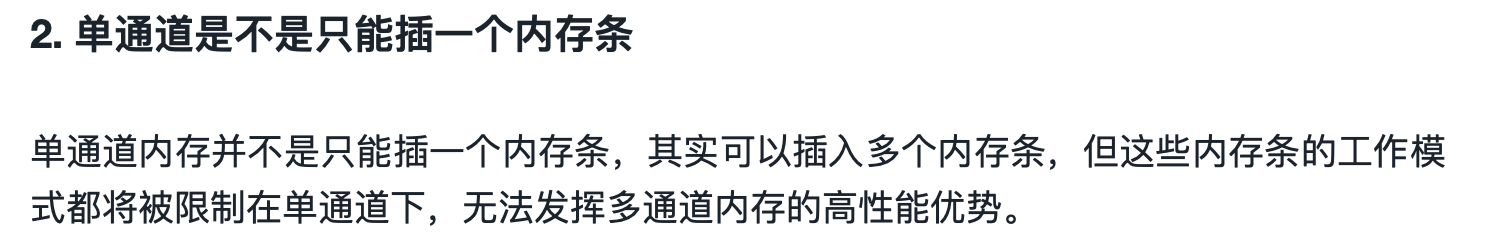
~~DDR5 4800MHz datawidth=64bit; 2CPU 8channels;~~

~~单通道内存：4800/16\*64bit\*16/8=38.4GB/s;~~

~~CPU是8个通道的，即单CPU的内存带宽是38.4\*8=307.2GB/s;~~

~~服务器有2个CPU，总带宽是614.4G(理论)；~~





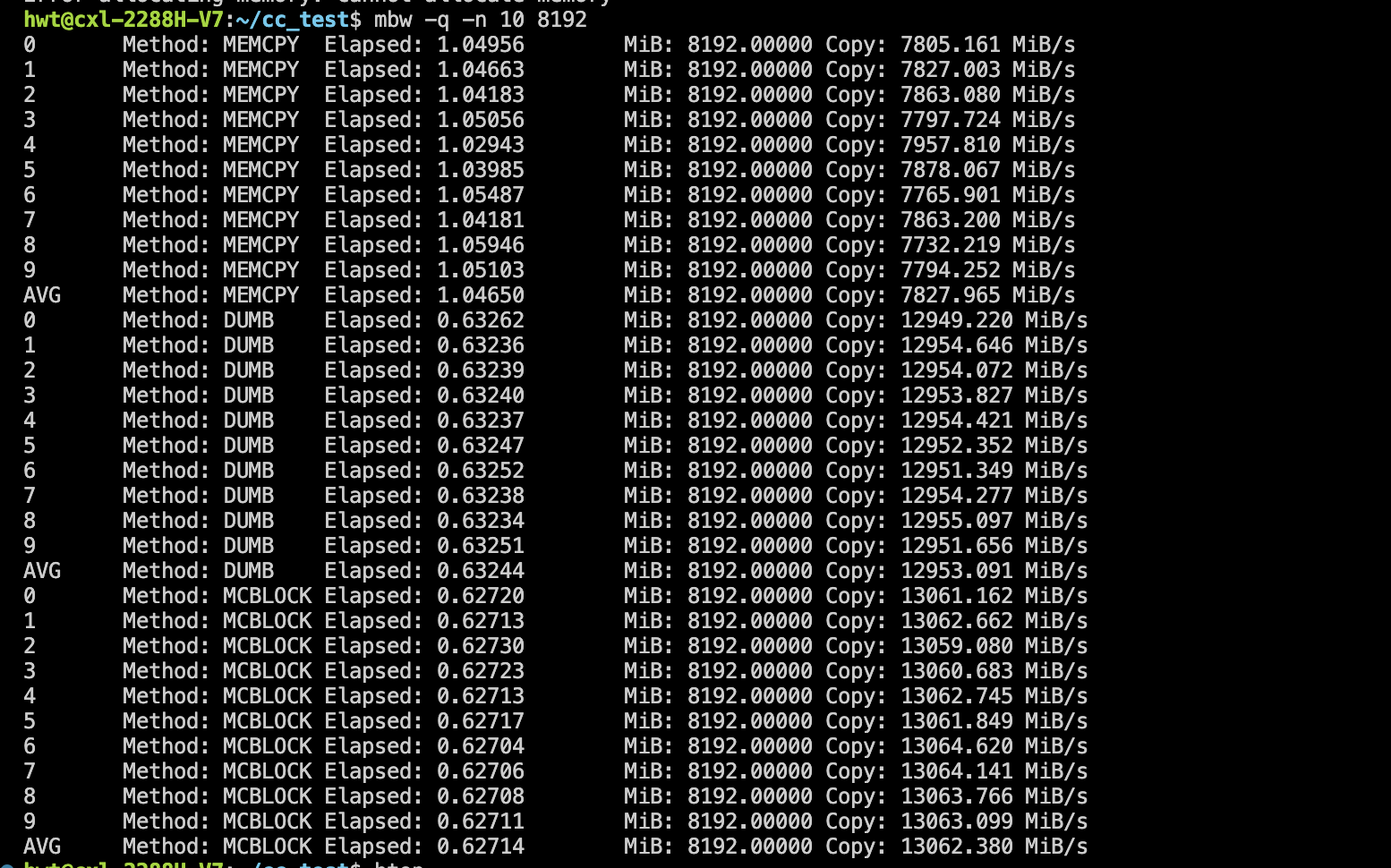


这里的configuration:latency = 0 是不是延迟可配？

使用一个简单的mbw(Memory Bandwidth benchmark)测试一下现有内存带宽。

内存带宽测试工具mbw https://github.com/raas/mbw

是一个带宽测试工具（通常用来评估用户层应用程序进行内存拷贝操作所能够达到的带宽），可以测试在内存拷贝memcpy, 字符串拷贝dumb, 内存块拷贝mcblock三种不同方式下的内存拷贝速度，程序代码比较简单，便于理解。



-q 隐藏日志 10 测试次数 8192内存大小

但是为什么测试结果这么奇怪？

PTU（Intel-performance-tuning-utility）

<https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html>

<https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler-download.html?operatingsystem=linux&distributions=online>

wget https://registrationcenter-download.intel.com/akdlm/IRC\_NAS/56d0db2b-1ff1-4abe-857a-72ca9be22bd3/l\_oneapi\_vtune\_p\_2024.0.1.14.sh

sudo sh ./l\_oneapi\_vtune\_p\_2024.0.1.14.sh

使用以上指令下载并手动安装；

<https://www.intel.com/content/www/us/en/docs/vtune-profiler/get-started-guide/2024-0/linux-os.html>

使用指南

可以使用vtune在Linux上采集，然后放到Windows上来分析；

很好的使用笔记：

<https://seekstar.github.io/2021/09/19/%E5%9C%A8%E6%9C%8D%E5%8A%A1%E5%99%A8%E4%B8%8A%E4%BD%BF%E7%94%A8vtune/>

可以使用PTU，硬件层面的带宽使用测量工具，主要是分析系统瓶颈；

ptu现在应该已经用不了了，但是又没有找到其他的工具；原文结果如下：



CPU的topology结构，每个操作系统的CPU对应到哪个CPU哪个核心的哪个超线程去。有了这个信息，才能使用taskset将内存测试工具绑定到指定的CPU，才能精确的观察内存使用情况。

参考：<https://blog.yufeng.info/archives/666>

修改用户权限：

<https://blog.csdn.net/younger_china/article/details/23349249>

查看系统各类硬件的信息

sudo apt install hardinfo

hardinfo

<https://cloud.tencent.com/developer/article/2305756>

hwloc指令使用指南：<https://benjr.tw/98373>

如何识别Linux上的CPU处理器架构

如今，多核处理器架构变得越来越流行。支持高性能计算应用程序、硬件虚拟化和数据中心服务器整合的需求加速了这一趋势。如果您是服务器管理员和云架构师，您必须充分了解服务器的CPU处理器架构，以便部署的应用程序能够充分利用底层硬件能力。

高核心密度硬件的趋势也引导了软件开发的发展，引入了新型并行编程模型。在这些模型下开发的多线程应用程序必须能够利用跨不同内核的并行执行、多级缓存、CPU/内存亲和性等。

本教程介绍**如何在 Linux 上通过命令行识别 CPU 处理器架构**。 CPU处理器架构的特点是物理插槽/处理器的数量、每个处理器的核心数量、多级（L1/L2/L3）缓存、NUMA（非均匀内存访问）配置等。

方法一：likwid

likwid ((Like I Knew What I'm Doing) is a suite of command line tools that are designed to support application designers for multi-threaded application development. likwid works with Linux kernel 2.6 and higher, and is regularly updated to support the latest generations of Intel/AMD processors, such as Intel's Sandy, Ivy, Haswell, Broadwell, Skylake processors, and AMD K8, K10, and Bulldozer (Interlagos).

在 Linux 上安装 likwid

$ tar xvfvz likwid-3.0.0.tar.gz

$ cd likwid-3.0.0

$ sudo make install

likwid 附带了几个命令行工具：

* likwid-topology：显示 NUMA 和缓存拓扑。
* likwid-perfctr：显示处理器的硬件性能计数器。
* likwid-features：显示和更改 Intel Core 2 处理器上的硬件预取控制位。
* likwid-pin：将多线程应用程序固定到特定的 CPU。
* likwid-bench：用于线程汇编内核快速原型设计的基准测试工具。
* likwid-mpirun：支持 MPI 和 MPI/线程混合应用程序的 CPU 固定的脚本。
* likwid-perfscope：likwid-perfctr 的前端，允许实时绘制性能指标。
* likwid-powermeter：用于访问 RAPL 计数器并查询 Intel 处理器上的 Turbo 模式步骤的工具。
* likwid-memsweeper：清理 ccNUMA（缓存一致性 NUMA）内存域的工具。

可视化 CPU 处理器架构：

$ likwid-topology -g

-------------------------------------------------------------

CPU type: Intel Core Westmere processor

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Hardware Thread Topology

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Sockets: 2

Cores per socket: 4

Threads per core: 2

-------------------------------------------------------------

HWThread Thread Core Socket

0 0 0 0

1 0 0 1

2 0 10 0

3 0 10 1

4 0 1 0

5 0 1 1

6 0 9 0

7 0 9 1

8 1 0 0

9 1 0 1

10 1 10 0

11 1 10 1

12 1 1 0

13 1 1 1

14 1 9 0

15 1 9 1

-------------------------------------------------------------

Socket 0: ( 0 8 4 12 6 14 2 10 )

Socket 1: ( 1 9 5 13 7 15 3 11 )

-------------------------------------------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Cache Topology

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Level: 1

Size: 32 kB

Cache groups: ( 0 8 ) ( 4 12 ) ( 6 14 ) ( 2 10 ) ( 1 9 ) ( 5 13 ) ( 7 15 ) ( 3 11 )

-------------------------------------------------------------

Level: 2

Size: 256 kB

Cache groups: ( 0 8 ) ( 4 12 ) ( 6 14 ) ( 2 10 ) ( 1 9 ) ( 5 13 ) ( 7 15 ) ( 3 11 )

-------------------------------------------------------------

Level: 3

Size: 12 MB

Cache groups: ( 0 8 4 12 6 14 2 10 ) ( 1 9 5 13 7 15 3 11 )

-------------------------------------------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NUMA Topology

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NUMA domains: 2

-------------------------------------------------------------

Domain 0:

Processors: 0 2 4 6 8 10 12 14

Relative distance to nodes: 10 20

Memory: 4207.48 MB free of total 8181.75 MB

-------------------------------------------------------------

Domain 1:

Processors: 1 3 5 7 9 11 13 15

Relative distance to nodes: 20 10

Memory: 4020.77 MB free of total 8192 MB

-------------------------------------------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Graphical:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Socket 0:

+-----------------------------------------+

| +-------+ +-------+ +-------+ +-------+ |

| | 0 8 | | 4 12 | | 6 14 | | 2 10 | |

| +-------+ +-------+ +-------+ +-------+ |

| +-------+ +-------+ +-------+ +-------+ |

| | 32kB | | 32kB | | 32kB | | 32kB | |

| +-------+ +-------+ +-------+ +-------+ |

| +-------+ +-------+ +-------+ +-------+ |

| | 256kB | | 256kB | | 256kB | | 256kB | |

| +-------+ +-------+ +-------+ +-------+ |

| +-------------------------------------+ |

| | 12MB | |

| +-------------------------------------+ |

+-----------------------------------------+

Socket 1:

+-----------------------------------------+

| +-------+ +-------+ +-------+ +-------+ |

| | 1 9 | | 5 13 | | 7 15 | | 3 11 | |

| +-------+ +-------+ +-------+ +-------+ |

| +-------+ +-------+ +-------+ +-------+ |

| | 32kB | | 32kB | | 32kB | | 32kB | |

| +-------+ +-------+ +-------+ +-------+ |

| +-------+ +-------+ +-------+ +-------+ |

| | 256kB | | 256kB | | 256kB | | 256kB | |

| +-------+ +-------+ +-------+ +-------+ |

| +-------------------------------------+ |

| | 12MB | |

| +-------------------------------------+ |

+-----------------------------------------+

上面是 HP ProLiant DL380 G7 的示例输出，其中显示了两个物理插槽、每个插槽中支持超线程的四核 CPU、32kB L1 缓存、256kB L2 缓存和 12MB L3 缓存。

方法二：hwloc

hwloc 是一个命令行套件，它收集了底层处理器架构的各种属性，例如 NUMA 内存节点、多级缓存、处理器插槽、处理器内核、PCI 设备/桥等。

在 Debian、Ubuntu 或 Linux Mint 上安装 hwloc

$ sudo apt-get install hwloc

在 Fedora、CentOS 或 RHEL 上安装 hwloc

$ sudo yum install hwloc

一旦安装了hwloc软件包，您就可以使用lstopo来显示处理器架构，如下所示。

$ lstopo --no-io

如果您在 Linux 桌面环境中运行 lstopo，它将弹出一个窗口，其中可以很好地显示底层处理器架构和缓存层次结构。

如果在无桌面服务器环境中调用lstopo，它将以文本格式显示输出，如下所示。

Machine (16GB)

NUMANode L#0 (P#0 8182MB) + Socket L#0 + L3 L#0 (12MB)

L2 L#0 (256KB) + L1 L#0 (32KB) + Core L#0

PU L#0 (P#0)

PU L#1 (P#8)

L2 L#1 (256KB) + L1 L#1 (32KB) + Core L#1

PU L#2 (P#2)

PU L#3 (P#10)

L2 L#2 (256KB) + L1 L#2 (32KB) + Core L#2

PU L#4 (P#4)

PU L#5 (P#12)

L2 L#3 (256KB) + L1 L#3 (32KB) + Core L#3

PU L#6 (P#6)

PU L#7 (P#14)

NUMANode L#1 (P#1 8192MB) + Socket L#1 + L3 L#1 (12MB)

L2 L#4 (256KB) + L1 L#4 (32KB) + Core L#4

PU L#8 (P#1)

PU L#9 (P#9)

L2 L#5 (256KB) + L1 L#5 (32KB) + Core L#5

PU L#10 (P#3)

PU L#11 (P#11)

L2 L#6 (256KB) + L1 L#6 (32KB) + Core L#6

PU L#12 (P#5)

PU L#13 (P#13)

L2 L#7 (256KB) + L1 L#7 (32KB) + Core L#7

PU L#14 (P#7)

PU L#15 (P#15)

您可以通过指定输出文件，让 lstopo 将处理器架构可视化导出到单独的图像文件，如下所示。

$ lstopo --no-io topo.png

方法三：numactl

numactl 是一个命令行工具，用于调整 NUMA 硬件（例如将进程或线程固定到特定的物理核心或 ccNUMA 节点）。

在 Debian、Ubuntu 或 Linux Mint 上安装 numactl

$ sudo apt-get install numactl

在 Fedora、CentOS 或 RHEL 上安装 numactl

$ sudo yum install numactl

如果您想使用 numactl 检查可用的 NUMA 节点，请执行以下操作：

$ numactl --hardware

available: 2 nodes (0-1)

node 0 cpus: 0 2 4 6 8 10 12 14

node 0 size: 8181 MB

node 0 free: 4235 MB

node 1 cpus: 1 3 5 7 9 11 13 15

node 1 size: 8191 MB

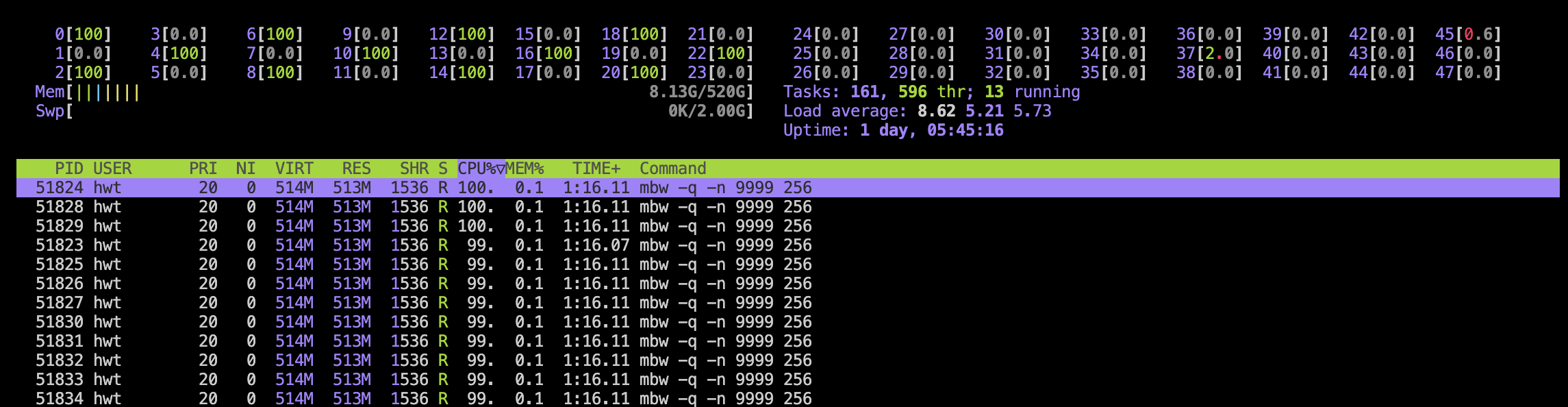
node 1 free: 4048 MB

node distances:

node 0 1

0: 10 20

1: 20 10



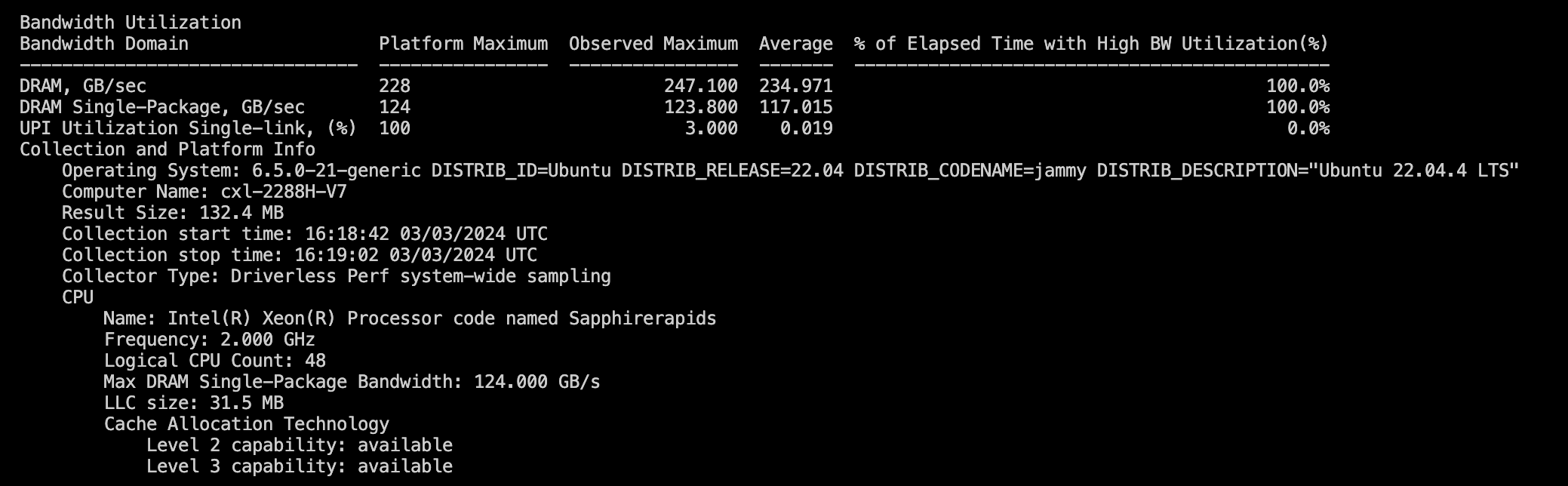
第一区域：CPU、内存、Swap的使用情况；

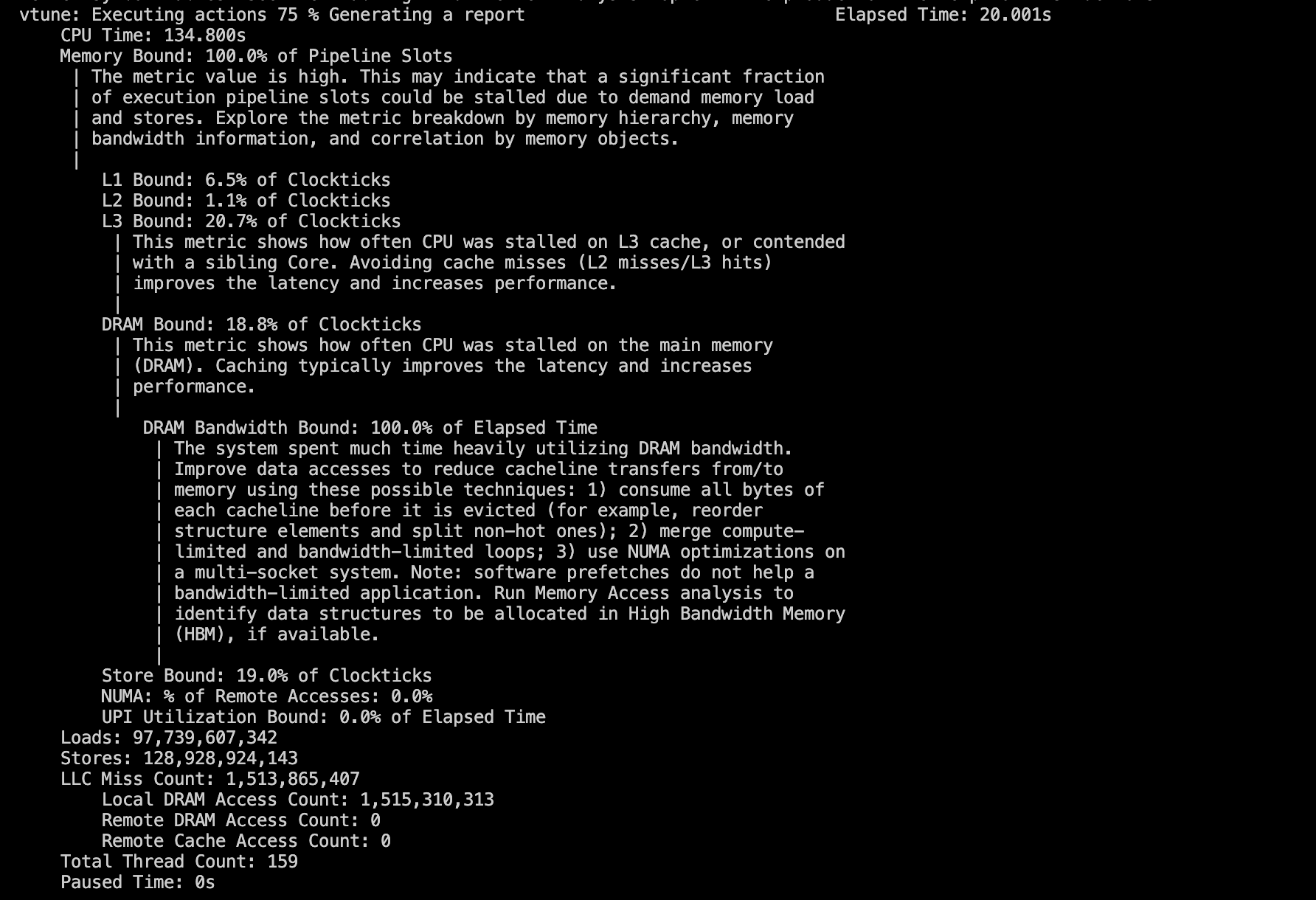
第二区域：任务、线程、平均负载及系统运行时间的信息。平均负载部分提供了三个数字，这仅仅表示的是过去的5分钟、10分钟和15分钟系统的平均负载而已，在单核的系统中平均负载为1表示的是百分之百的 CPU 利用率。最后，运行时间 （uptime）标示的数字是从系统启动起到当前的运行总时间。

第三区域：当前系统中的所有进程。各列说明：  
PID：进程标志号，是非零正整数  
USER：进程所有者的用户名  
PR：进程的优先级别  
NI：进程的优先级别数值  
VIRT：进程占用的虚拟内存值  
RES：进程占用的物理内存值  
SHR：进程使用的共享内存值  
S：进程的状态，其中S表示休眠，R表示正在运行，Z表示僵死状态，N表示该进程优先值是负数  
%CPU：该进程占用的CPU使用率  
%MEM：该进程占用的物理内存和总内存的百分比  
TIME+：该进程启动后占用的总的CPU时间  
COMMAND：进程启动的启动命令名称

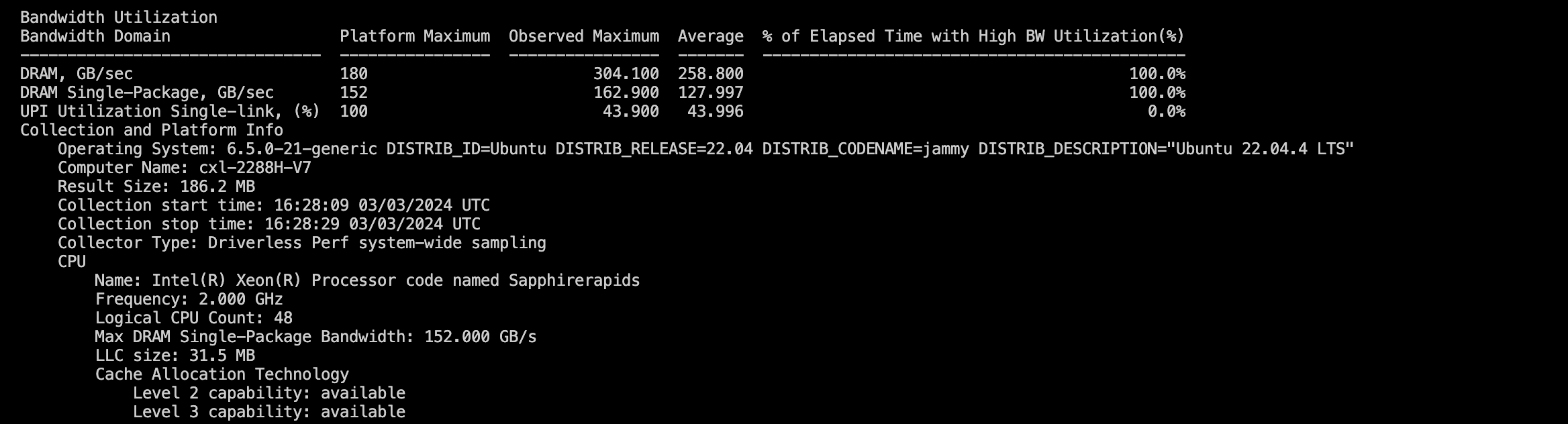


需要加入到bashrc文件中自启动；



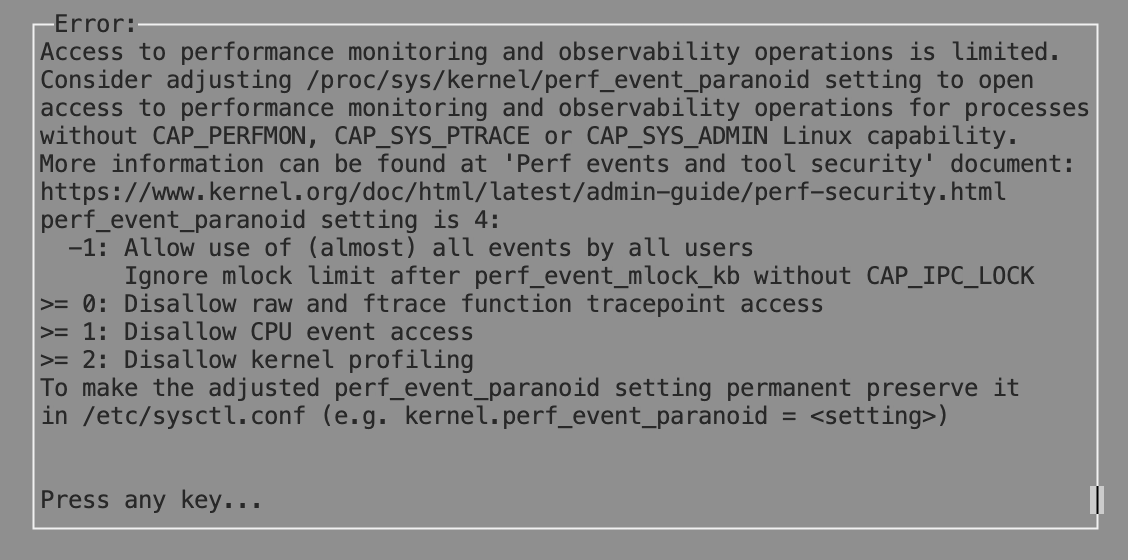


跑满1个cpu的24个核：

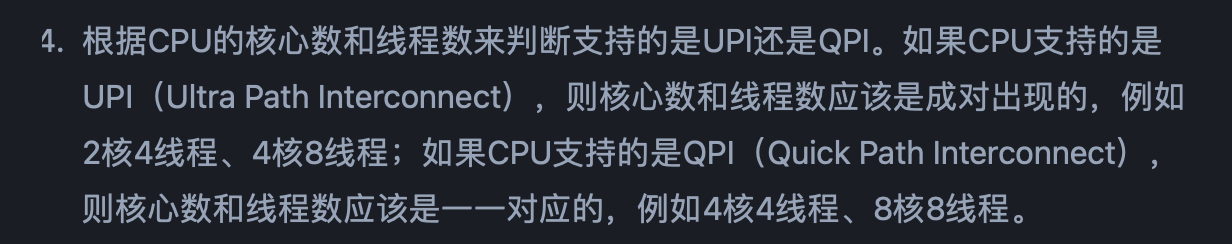


最大带宽差不多能达到理论带宽

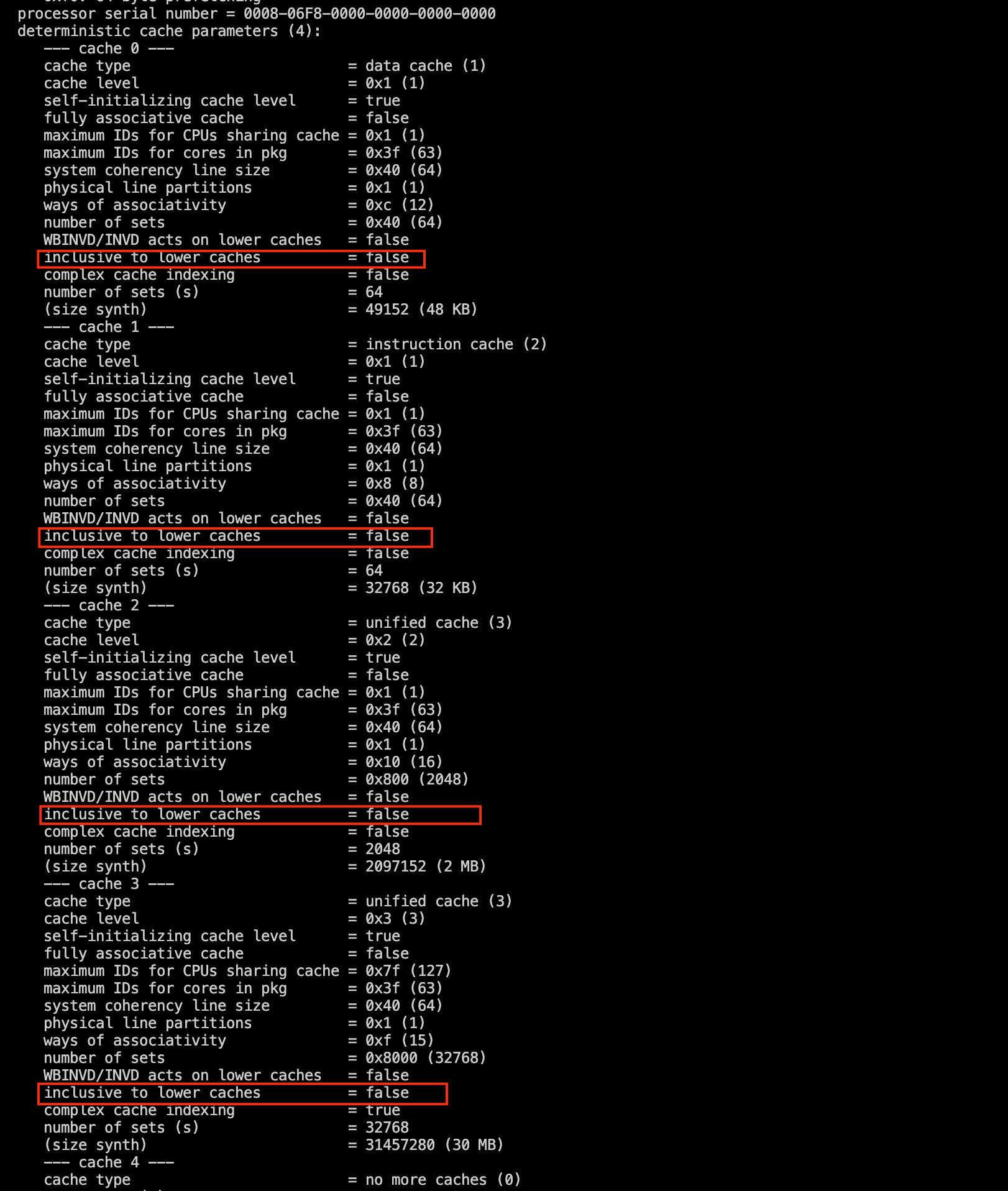
通过命令行来绑定核，但是要考虑NUMA是否开启，禁止当前节点访问另一个NUMA节点的内存才行；



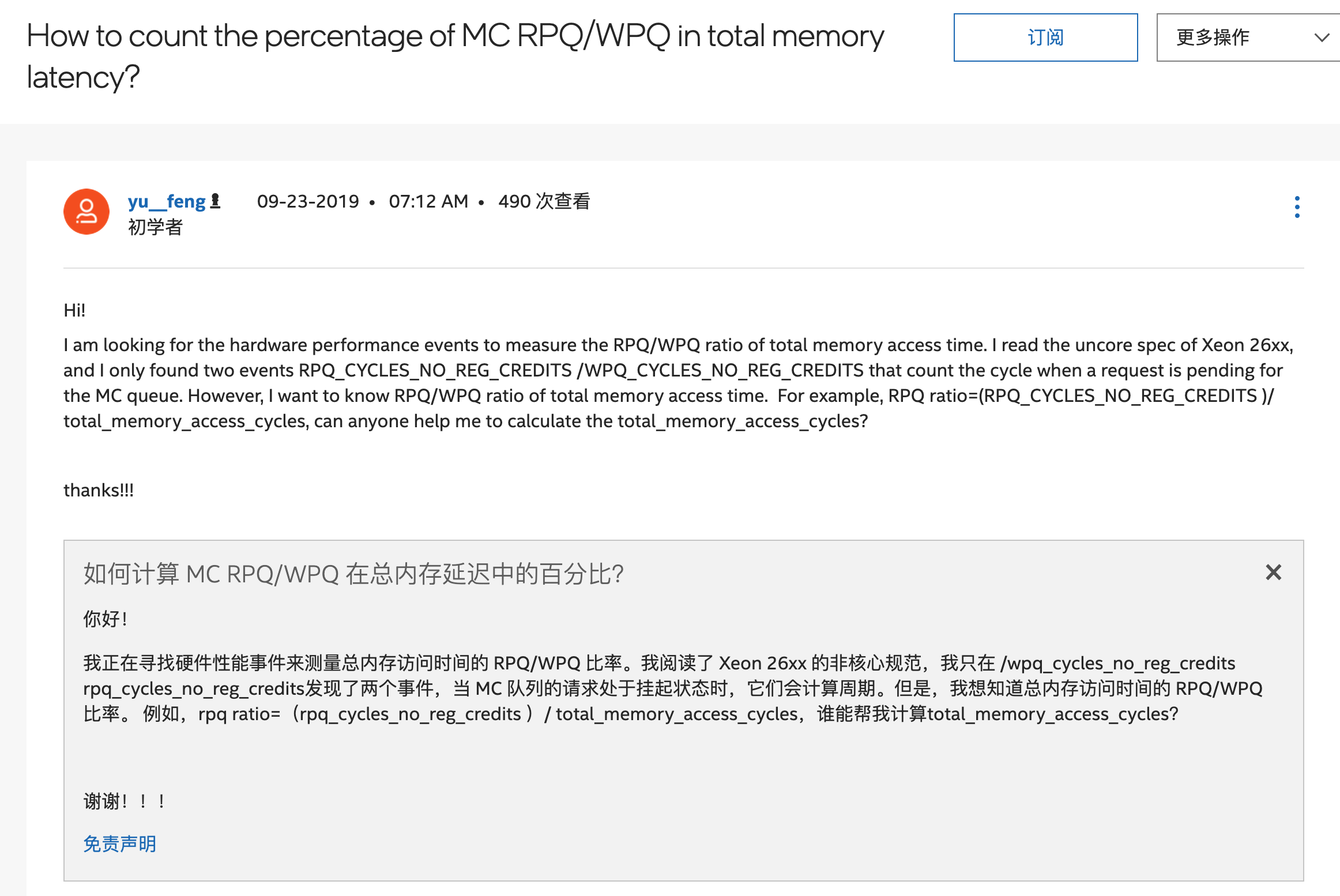
相关的寄存器没有设置导致当前perf无法使用；



这种判断方法对吗？







这两个如何看？