AMD

针对AMDEPYC™7002系列的高性能计算(HPC)调优指南

处理器

出版单位 56827

修订版次 2.0

发布日期 2020年11月

使用目的

本文档提供了开始调整AMD2的指导[编号]基于HPC工作负载的基于一代EPYC™处理器的系统。这不是一个全面的指南，有些项目可能有类似的，但不同的，名称，在特定的OEM系统(例如。特定于OEM的BIOS设置）。每个HPC工作负载的性能特性都有所不同。虽然本指南是一个很好的起点，但我们鼓励您执行您自己的性能测试，以进行额外的调整。本指南还提供了关于哪些项目应作为特定于应用程序的附加调整的重点的建议。

产品目录

使用 日期	修订版 次	说明
2020年11月	2.0	与以前发布的版本合并
2020年01月	1.0	首次公开发布

Chapter 1 产品概述

HPC工作负载具有独特的要求。OEM平台的默认硬件和BIOS配置可能无法为HPC工作负载提供最佳性能。要在超级平台和工作负载级别上启用优化，本指南指出

- 可能会影响性能的BIOS设置
- 硬件配置的最佳实践
- 受支持的操作系统版本和优化
- 针对BIOS和操作系统设置的工作负载特定的建议

还讨论了AMDEPYC软件开发环境，包括有关如何安装和运行HPL、HPCG、DGEMM和流基准测试的信息。本指南提供了一个良好的起点，但并没有在所有编译器中进行详尽的测试。

1.1 前提条件

要使用本文档并对HPC执行调整，技术受众应具有配置服务器的经验，以及以下内容：

- 以管理方式访问服务器的管理界面(BMC)
- 强烈建议您熟悉OEM服务器的管理界面(BMC)
- 对操作系统的管理访问权限
- 强烈建议您熟悉操作系统特定的配置、监控和故障排除工具

Chapter 2 微架构和 设置

2.1 AMDEPYC™7002系列处理器

AMDEPYC™7002系列处理器基于AMD“Zen2”核心和微架构，采用前沿7纳米技术构建。AMDEPYC™SoC基于x86，提供了一套跨8到64个核心的一致功能，包括128条PCIe®Gen4,8个内存通道和高达4TB的高速内存。AMDEPYC™7002系列处理器采用以下规范构建：

AMDEPYC™7002系列	
处理器技术	7纳米
最大核数	64
最大内存速度	3200兆赫兹
最大内存容量	4tb
外围设备组件互连	128车道（最大车道）PCIe®第4代

下图显示了一个九模AMDEPYC7002系列处理器的高级方框图。图中间有一个中央I/O模具，IO模具周围有8个不同的CCDs（计算复合模具）。内存通道、核心和缓存的布局细节对性能有影响，本节将进行解释。

图1EPYC7002配置有8个核心复合模具(CCD)和中央输入/输出模具(IOD)

2.2 Zen2核心产品

AMDEPYC7002系列处理器基于新的Zen2处理器核心，其中包括L1回写缓存和专用的512KBL2缓存。每个核心都可以支持同时执行多线程(SMT)，允许每个核心同时执行2个执行线程。

2.3 核心复合体模具(CCD)

每隔2个[编号]EPYC代处理器由一个输入/输出模具(IOD)和多达8个核心复合模具(CCD)组成。CCD包含CPU的核心和缓存。CCD使用AMD的无限结构™连接到输入/输出模具。CCD连接到IOD以访问内存、I/O和彼此。每个插槽最多支持8个内存通道，PCIeGen4支持128个通道。

每个CCD最多包含两个核心复合物(CCX)，如下一节所述。

2.4 核心复合体(CCX)

核心综合体(CCX)由最多4个核心和共享的16MB（末级）L3缓存组成。每个CCD最多包含两个核心复合物(CCX)。虽然这两个CCXs及其相关的L3缓存在同一个芯片上，但它们是分开的。

图2核心复合物模具(CCD)上的两个核心复合物CCX）

可以使用BIOS中的以下一种或两种方法来禁用内核：

- 将每L3的岩芯从4减至3、2或1，保持CCD数量不变。
    - 这种方法增加了每个核心的有效缓存比率。它还减少了共享高速缓存的核心数量。
- 减少活动的CCD的数量，保持每个CCD的核心恒定。
    - 这种方法保持了在核心之间共享缓存的优势，同时保持了每个核心相同的缓存比率。

2.5 无限数据结构(DF)

无限结构提供了处理器的所有主要组件之间和2插座系统中的CPU之间的连贯内存连接。它支持高达1467MHz(FCLK)的速度。

2.6 统一内存控制器(UMC)

每个存储器控制器可以在高达1600MHz（内存时钟）下运行，因此可以支持高达3200MHz的DDR4主存储器。

如果使用的内存是DDR4-2933，则内存控制器将在1467MHz下运行。这与数据结构相同，称为耦合模式。这提供了最低的内存延迟。然而，使用DDR4-3200R2DIMMs仍然可以实现最大的内存带宽。

其他操作模式和“内存P状态”存在，并在工作负载调优指南中讨论，但对于HPC系统，最高性能与刚才描述的叙述有关。

2.7 内存和输入输出布局

每个AMDEPYC7002系列处理器都支持8个内存通道。每个内存通道最多支持2个DIMMs。系统的每个处理器最多可访问4TB的DDR4-3200内存。PCIGen4子系统提供多达128条高速输入/输出通道。

当所有内存和I/O连接到单个I/O死亡时，它们可以被抽象为逻辑象限，每个象限有2个内存通道和32条I/O通道。内存通道可以在象限内（双向），一直通过16通道交错，跨越2套接字系统的所有内存通道（更多信息请参阅NUMA部分）。

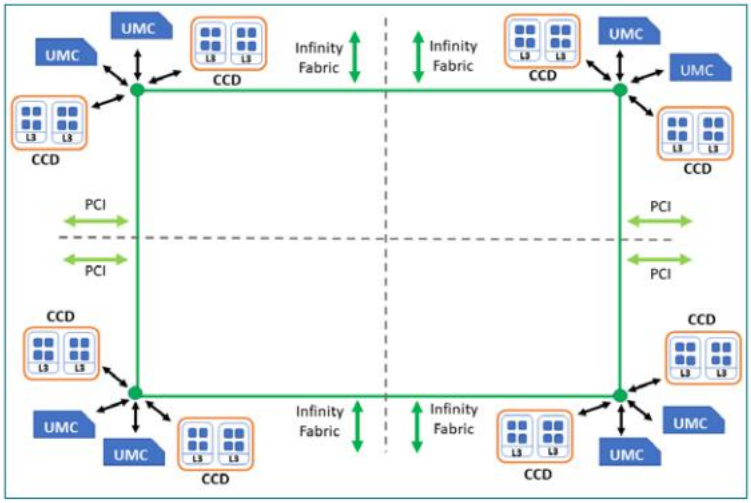


图3EPYC7002逻辑布局为IO模具周围的象限。

Chapter 3 编号

AMDEPYC7002系列处理器使用非统一内存访问(NUMA)内存体系结构。如上一节所述，AMDEPYC7002系列处理器中的四个逻辑象限允许将该处理器划分为不同的NUMA域。这些域被指定为每个插座的NUMA(NPS)。

使用BIOS设置，每个服务器可以配置为NPS1、NPS2、NPS4或NPS0（不推荐），并通过附加选项将L3缓存配置为NUMA(L3CAN)。

AMDEPYC7002系列处理器具有每个处理器的不同核心计数，并非所有它们都可支持所有NPS设置（对于每个插槽具有6个CCD的CPU，NPS=4不可用，只有NPS=1或2）：

https://developer.amd.com/wp-content/resources/56338_1.00_pub.pdf

3.1 小=1

NPS1表示每个套接字都有一个NUMA节点。此设置将处理器上的所有内存通道配置为单个NUMA域，即。处理器上的所有核心、连接到它的所有内存以及连接到处理器的所有PCIE设备都在一个NUMA域中。然后交错每个处理器上的所有8个内存通道。

3.2 小=2

在NPS2中，处理器被划分为两个NUMA域。每个处理器的一半的核心和一半的内存通道被分组为一个NUMA域，其余的核心和内存通道被分组为第二个域。内存交错在每个NUMA域中的四个内存通道上。

3.3 小=4

NPS4将处理器划分为四个NUMA域。处理器的每个逻辑象限都被配置为其自己的NUMA域。内存交错每个象限中的两个内存通道。PCIe设备将位于处理器上四个NUMA域之一，这取决于具有该设备PCIe根的IO模的象限。

3.4 节点数=0

NPS=0插入2插接字系统中的所有内存通道。此配置不应用于HPC工作负载，因为它为每个内存访问添加了套接字间延迟。

3.5 L3缓存为NUMA域

使用L3缓存为NUMA(L3CAN)，每个L3缓存都作为其自己的NUMA节点公开。在双处理器系统上，每个处理器最多有16xL3个缓存，此设置将显示多达32个NUMA域。

3.6 每个套接字的NUMA(NPS)和内存带宽

内存带宽随着CCD的更少而减少，并通过合成测试流清楚地演示。然而，对于具有更多随机内存存取模式的实际HPC工作负载，NPS=1或2可以提供与NPS=4非常相似的峰值性能（参见下章“描述7002分子动力学应用NAMD的策略”）

3.7 了解硬件信息和硬件信息

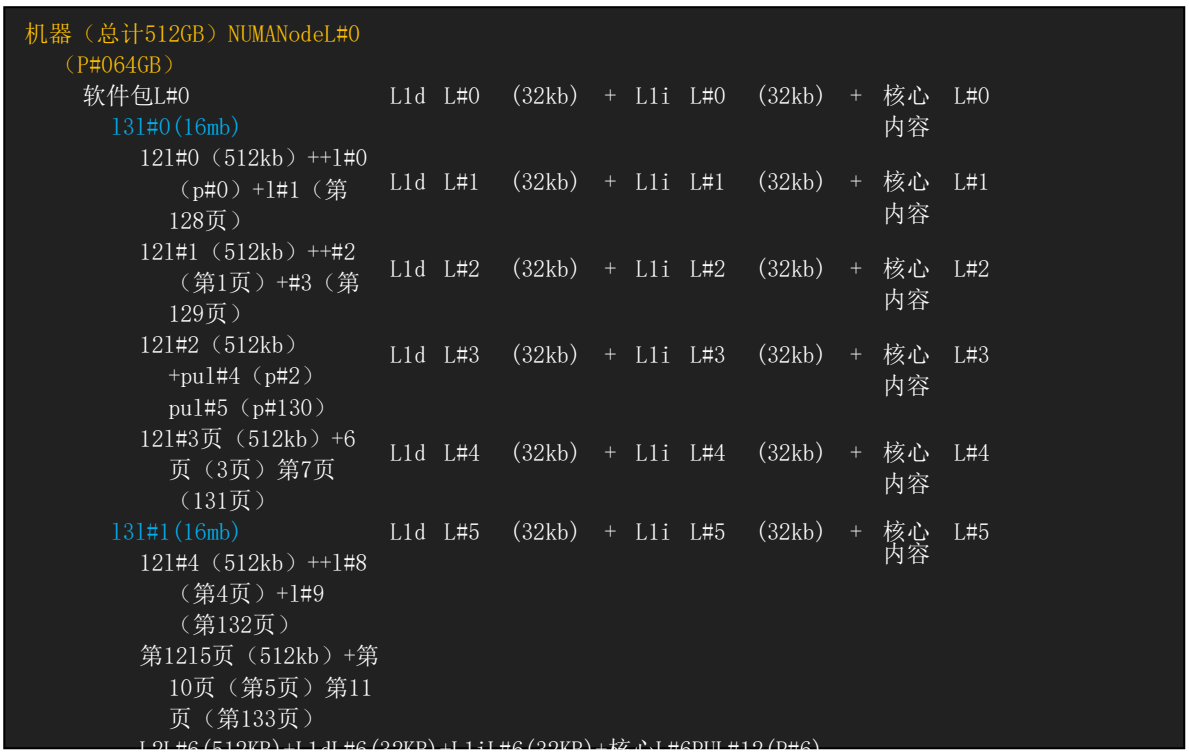
本节介绍操作系统如何查看NUMA节点、缓存和核心。下面的示例显示了双套接字系统上的hwloc-ls的输出，每个套接字有64核，配置为NPS=4，运行CentOS7.6。

本示例说明了几个要点：（请注意，此系统上未启用SMT）

- 每个L3缓存的4个核与与每个16MBL3缓存相关的CPUID分组在一起。32,33,34,35.每个L3的4个核的分组代表一个CCX。
- 每个CCD有2个CCX（由输出左边的蓝括号分组）不按hwloc-ls逻辑分组，因为它们在逻辑上是分开的，尽管它们物理上位于同一个CCD上。
- 4个CCX在逻辑上显示在每个NUMA节点下的分组。
- 每个NUMA节点由64GB的内存组成。
- 与此NUMA节点相关的PCIe车道上附加了一个MellanoxHCA。CPUID32-47在逻辑上“最接近”此Mellanox网卡。

hwloc-ls是一个非常有用的工具，在获得CPUID时，您需要知道将什么核心固定到您的工作的CPUID。您也可以使用numactl-H，它将提供每个NUMA节点的核心列表。

上面的hwloc-ls输出演示了每个核心的单个线程的情况。下面的示例显示了在BIOS中启用同时多线程(SMT)时的输出，从而在每个核心上启用第二个线程。



通常，在启用SMT的2x64核心CPU的双套接字系统上，每个核心上的第一个线程在0-127范围内，而第二个线程的CPU在128-255范围内具有CPUID。在上面的示例中，您可以看到core-0有两个与线程相关的线程：一个附加CPUID0，另一个附加CPUID128。

如果安装旧的Linux内核，还需要确保它能识别系统中正确的缓存层次结构，请使用hwloc-info：

```
$ hwloc-info
depth 0:        1 Machine (type #1)
 depth 1:       2 Package (type #3)
  depth 2:      8 NUMANode (type #2)
   depth 3:     32 L3Cache (type #4)
    depth 4:    128 L2Cache (type #4)
     depth 5:   128 L1dCache (type #4)
      depth 6:  128 L1iCache (type #4)
       depth 7: 128 Core (type #5)
        depth 8:        128 PU (type #6)
Special depth -3:       11 Bridge (type #9)
Special depth -4:       8 PCI Device (type #10)
Special depth -5:       12 OS Device (type #11)
```

旧的内核可能会不对齐或完全忽略L3，这一点使用hwlocinfo或hwloc-ls都会很明显。

3.8 C州

CPU可以在中空闲多个核心状态或C状态。根用户可以控制处理器使用哪种CPU状态。

- C0：处于激活状态。这是运行应用程序时的活动状态。
- C1：闲置状态
- C2：怠速和电源门控。这是一个更深的睡眠状态，当移动回C0状态时，与CPU退出C1时相比，它将有更大的延迟。

用户根目录可以启用和禁用C型计算机状态。例如，这里是双功率监视器的输出
所有的铁芯通常在C2中空转。

```
#c功率监视器
                 |兆器                     ||Idle_Stats
   0|     0|     0|    0.03| 99.97|     1937||  0.00|     0.00|  99.97
   0|     0|     8|    0.22| 99.78|     1973||  0.00|     0.00|  99.80
   0|     0|    16|    0.01| 99.99|     1935||  0.00|     0.00|  100.0
   0|     0|    24|    0.00|100.00|     1703||  0.00|     0.00|  100.0
   0|     0|    64|    0.00|100.00|     1854||  0.00|     0.00|  100.0
   0|     0|    72|    0.00|100.00|     1649||  0.00|     0.00|  100.0
   0|     0|    80|    0.00|100.00|     1694||  0.00|     0.00|  100.0
   0|     0|    88|    0.00|100.00|     1712||  0.00|     0.00|  100.0
   0|     1|     1|    0.01| 99.99|     1824||  0.00|     0.00|  100.0
   0|     1|     9|    0.01| 99.99|     1720||  0.00|     0.00|  100.0
   0|     1|    17|    0.00|100.00|     1758||  0.00|     0.00|  100.0
……等
```

以下是为核心0到3（-c0-3）禁用C2（-d2）的例子，它防止它们闲置进入C2，使它们处于C1或更高状态。

c0-3空闲集d2

```
#c0-11监视器
                |兆器                    ||Idle_Stats
    0|    0|    0|    0.01| 99.99|    1996||    0.00|   99.99|    0.00
    0|    0|    8|    0.20| 99.80|    1938||    0.00|    0.00|   99.75
    0|    1|    1|    0.00|100.00|    1896||    0.00|   99.96|    0.00
    0|    1|    9|    0.00|100.00|    1675||    0.00|    0.00|   99.95
    0|    2|    2|    0.00|100.00|    1856||    0.00|   99.96|    0.00
    0|    2|   10|    0.00|100.00|    1658||    0.00|    0.00|   99.97
    0|    3|    3|    0.00|100.00|    1872||    0.00|   99.96|    0.00
    0|    3|   11|    0.00|100.00|    1680||    0.00|    0.00|   99.97
    0|    4|    4|    0.00|100.00|    1690||    0.00|    0.00|   99.96
    0|    5|    5|    0.00|100.00|    1665||    0.00|    0.00|   99.96
    0|    6|    6|    0.00|100.00|    1661||    0.00|    0.00|   99.96
……等
```

在这4个核心上使用cpuc0-3空闲集-e2重新启用电源门控C2空闲状态（-e2）

可以通过在c功率空闲集指令中省略-c参数，将这些设置应用于所有核心。启用SMT，如果线程在C0中，核心不能进入C2（激活）状态或C1空闲状态。因此，如果在任何逻辑CPU上禁用C2，您也应该在SMT兄弟系统上禁用C2。在一个2x32的核心系统中，如果你在CPU7上禁用C2，你也应该在CPU71上禁用C2)。

禁用C2对于运行高性能、低延迟的网络非常重要。从电源门控C2状态移动到活动C0状态所需的延迟可以增加网络IO的延迟。所有核心必须在C1中闲置（禁用C2）以支持高性能网络。

3.9 P型状态、频率和提升

P状态是在内核活动时管理电源使用情况的执行电源状态，如C状态在内核空闲时管理电源级别的方式。只有当处理器处于C0“活动”状态时，才能实现更高的P状态。

- P型状态是执行权力状态
- C型状态为空闲节能状态

一旦在C0状态下激活，核心就可以根据其利用率在其各种P状态之间移动，以平衡性能和电源使用情况。当一个核心被充分利用时，它将针对P0，即其引用的基准频率。

用户根用户可以通过执行来观察这些p状态

脉冲功率频率信息

下面的示例显示了7742CPU的输出，它显示了在1.5GHz（P2）、2.0GHz（P1）和2.25GHz（P0）频率下的三个P状态。

```
c功率频率信息分析CPU0：
  驱动程序：计算机程序
  以相同的硬件频率运行的CPU：0
  需要通过软件协调其频率的CPU：0最大转换延迟：无法确定或不受支持。硬件限制：
  1.50GHz-2.25GHz
  可用频率步长：2.25GHz、2.00GHz、1.50GHz
  可用的cpufreq调速器：保守的用户空间电源节省按需性能
  现行策略：频率应在1.50GHz和2.25GHz以内。
                        调速器的“性能”可决定在此范围内使用哪个速度。
  当前CPU频率：2.25GHz（通过硬件呼叫断言）提升状态支持：
    支持：是的，活动：
    没有启动状态：0
    总状态：3个个人状态，
    P0：          2250MHz
    的P状态-P1：  2000MHz
    的P状态-P2：  1500兆赫
    兹
```

启用增强允许一个核心实现另一个，更高的频率。此频率作为CPU的启动频率而被引用。最大提升是任何核心可以提升的最大频率，只要浮点单元中有足够的热和计算净空间。

从上例中可以看出，7742CPU的报价基准频率为2.25GHz。如果启用了提升，7742的核心可以提高到其引用的提升频率的3.4GHz。如果CPU内的所有核心都试图提高这个高，整个CPU可能会达到热功率限制，从而降低有效频率到基础和提升频率之间的范围。

若要从启动中获益，必须启用它。这可以在BIOS中设置，例如，在红帽Linux命令行上设置为根命令（启动需要在BIOS中设置为启用，以便允许从命令行切换和关闭）：

echo1>、系统、设备、系统、cpu、cpufreq/boost

可返回：

echo0>、系统、设备、系统、cpu、cpufreq/boost

在BIOS中更改“启动”设置将需要重新启动系统。

对于HPC工作负载，我们建议将CPU调速器设置为性能。用户可以通过运行cpupower监视器或AMD微探查器(uProf)命令行作为根界面来观察到这一点。（有关详细信息，请参阅后面的CPU控制器部分。）

3.10 CPU控制器

AMDEPYC支持几个CPU控制器。不同的调速器可以应用于不同的机芯。对于高性能计算环境，“性能”调节器经常被广泛使用：

- 性能：这将核心频率设置为P0内的最高可用频率。当启动器设置为关闭时，它将在基底频率下工作，例如。在7742CPU上的2.25GHz。如果启动，则它将尝试将频率提升至3.4Ghz的最大启动频率。当在提升的频率下工作时，这仍然代表着P0p的状态。
- 按需应变：根据尾部负载设置核心频率。这有利于快速上升至最高工作频率，随后在怠速时缓慢降至P2。这可能会惩罚那些短命的线程。
- 保守：类似于按需应变，但有利于更优雅的斜坡到最高频率，并在空闲时快速返回P2。
- 功率节约：设置最低支持的核心频率，并将其锁定为P2。

管理员可以通过cpupower命令设置CPU调节器。下面是一个将CPU调节器设置为性能的示例：

大功率频率设置g性能

关于Linux中的CPU控制器的更广泛的讨论和解释可以在kernel.org，https://上找到 www.kernel.org/doc/Documentation/cpu-freq/governors.txt

3.11 有用的“电源”命令示例

cpupower命令是一个非常有用的实用工具，用于查询和设置CPU上的一系列条件。我们在这里列出了一些例子：

c0-15监视器

显示核心0到15上的频率。如果用户需要在打开和关闭启动功能时观察到更改，则很有用。

脉冲功率频率信息

列出提升状态、CPU调速器以及有关CPU配置的其他有用信息。

大功率频率设置g性能

将CPU调速器更改为“性能”。

空闲组d2

禁用CPU0到15上的C2空闲状态。

请参阅https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt以获取更多关于CPU控制器的详细信息。

Chapter 4 快速参考-高性能性能 设置

本节包含了调优提示的快速参考摘要，它可以作为HPC系统的BIOS和OS设置的快速起点。

不熟悉AMDEPYC处理器体系结构的阅读器可能会发现在继续之前阅读本文档的其他部分有益。

4.1 快速参考：BIOS和OS

BIOS设置：

- SMT=OFF|ON（是否依赖于应用程序而提供优势）
- iommu=关闭
  注意：2x64核SMT=(即。256个螺纹）：
  - 打开IOMMU=，并推荐iommu=pt作为内核引导参数
  - x2apic=自动版
- APBDIS=1；和固定的SOCP状态=P0
- 核心性能增强，=开启
- 确定性滑块=功率
- cTDP=240(设置为处理器的最大CTDP设置，例如。7742的值为240W）
- PPL=240(设置为处理器的最大cTDP设置，例如。7742的值为240W）
- NPS=4
  - 或对6个CCD使用NPS=2（如果CPU有6个CCD，则禁止NPS=4）
- DFC-状态=已禁用
- 首选-IO控制=手册
  - 首选-IO设备=<总线号从无限波段卡：使用lspci>
  - 增强的首选IO模式=已启用
- 核心C型状态=已启用
- tsme=关闭

平台供应商可能包括针对HPC的工作负荷配置文件或系统调整。这些功能可能会禁用C状态（不推荐）和/或禁用核心性能增强（也不推荐）。如果无法在工作负荷配置文件中恢复这两个设置，请使用自定义设置。

操作系统设置：

- 对于具有高性能低延迟互连的HPC集群，如Mellanox，将禁用C2空闲状态。假设有一个双插座2x64核心系统

  空闲集d2

- 将CPU调节器设置为“性能”：

  大功率频率设置g性能

使用这些设置可建立性能基线。一旦建立起来，就会鼓励开发人员基于这个基线进行进一步的系统调整。

4.2 快速参考：基本系统检查

- 检查SMT是否启用了[线程(s)=1表示SMT=OFF；线程（2）=2表示SMT=ON]：

  云器

- 检查无限波段卡或其他外围设备的连接：

  高计

- 检查增压器是否打开（1）或关闭（0）：

  猫、系统、设备、系统、cpu、cpufreq/boost

- 检查CPU调速器和其他有用的设置：

  脉冲功率频率信息

- 目视检查哪些核心/线程都很忙

  高压机顶

- 检查核心上的频率和空闲状态

  脉冲功率监视器

- 运行流：双CPU插槽，DDR4-3200双级，每个通道1个DIMM插槽(64个核心CPU上每L31核)，使用Intel或AOCC编译器应该产生大约350GB/s

4.3 其他提示

使用“seq”生成CPUID列表

可能需要在不同时间构建以逗号分隔的CPUID列表。建议使用Linux命令序列号：

具有2x64核CPU的系统，每L3有2核：

第二段，0 2 127

系统具有2x64核CPU，每L3有3核（加入2个列表，每秒核心+每四核）：

秒，0 2 127 |tr-d ‘\n’；　　回声，|　tr-d ‘\n’；　回声 “$（秒秒，1 4 127）”

核心封装和内存位置：

您可以将二进制文件固定在一个特定的核心上运行，例如。核心内容7：

/my二进制文件

这可以放置线程，但仍然允许内核自由选择要使用的内存插槽。为了确保内存在逻辑上最接近核心，然后包括“--模绑定=”

模因绑定=0。/my二进制文件

要确定核心属于哪个NUMA节点，请使用numactl-H的输出

```
节点 0 会计单 0 1 2 3 4 5  6 7 8  9 10 11 12 13 14 15
       位：
节点 0 外形尺 65422mb
       寸：
节点 0 免费： 296mb
节点 1 会计单 16 17  18 19  20 21  22 23  24 25  26 27  28 29  30 31
       位：
节点 1 外形尺 65535  mb
       寸：
节点 1 免费： 40mb
节点 2 会计单 32 33  34 35  36 37  38 39  40 41  42 43  44 45  46 47
       位：
节点 2 外形尺 65535  mb
       寸：

节点 3 会计单 48 49  50 51  52 53  54 55  56 57  58 59  60 61  62 63
       位：
节点 3 外形尺 65523  mb
       寸：
节点 3 免费： 41mb
节点 4 会计单 64 65  66 67  68 69  70 71  72 73  74 75  76 77  78 79
       位：
节点 4 外形尺 65535  mb
       寸：
节点 4 免费： 321mb
```

使用“make-j”更快地进行“make”

从源焦油球构建代码的一种常见方法是使用

。/配置、制造、制造、安装

“make”步骤将隐式使用一个核心。使用AMDEPYC，您可以通过传递-j标志并表示希望编译器使用的最大线程数来显著加快编译。例如，在带有SMT=的双套接字2x64核心系统上，现在有256个线程，可以使用以下内容：

。/配置；make-j256；进行安装

例如：CPUID为7的核心位于NUMA域0中

这允许编译器利用最多256个线程进行编译。对于大型代码源，这将极大地减少构建时间。

Chapter 5 生物系统 设置

本节介绍了高性能计算环境中的通用BIOS设置。有关这些设置的更多详细信息，请参见《工作量调整指南》。

5.1 裸金属工作负载的BIOS设置

下面的指导方针旨在让您开始进行调优。确保评估您的需求，并应用最适合您的要求的适当设置。如果找不到以下确切的选项，请查看服务器制造商指南。这些选项表示AMD参考平台上默认BIOS设置的更改，但不一定在每个服务器平台上。

- x2APIC 已启用
- SMT 已被禁用
- *NPS 4（正确设置内存频率）*
- 苹果培 1
- 固定的SOCP状态 P0
- 首选的IO

- 确定性滑块=性能|功率
  电源和执行确定性模式（系统管理单元(SMU)策略选择。每个核心将运行的有效GHz由SMU控制。这将跟踪处理器所有部分的功率、电流绘制和温度，并了解cTDP的功率预算。

  通过使用该SKU范围内所有CPU通用的最低基准参考设置，为群集的相同SKU提供可重复的性能。

  “功率”：利用产率变化，提高运行时性能，保证各部分的功率消耗达到但不超过cTDP功率限制。如果要将cTDP设置为其最大值，则必须设置“确定性滑块=电源”

- cTDP（可配置的热设计点）
  确保在BIOS中使用CTDP=PPL。每个CPU都有一个最小和最大的cTDP阈值。

- PPL（成套设备的功率限制）
  每个CPU都有一个最大的PPL限制。确保PPL=cTDP。PPL可以设置为低于CTDP的最小值
- SMT（同时使用多线程）
  启用后，每个核心将显示两个线程。用户可以参考命令行上的hwloc-ls，以获取每个线程的CPUID列表。在HPC工作负载中，SMT通常会被关闭。如果您不处于计算绑定的场景中，您可能会看到SMT的一些好处。如果您的代码没有获得每个核的授权，或者对启用SMT没有金钱影响，您可能需要进行尝试，看看它是否有利于您的工作量。

- 启用：这允许1个核心执行2个线程。
- 禁用：这允许1个core执行1个线程。

此设置的好处取决于应用程序。注意，对于具有2x64和SMT=的双套接字平台，用户还需要设置以下设置：

- 打开IOMMU=，并推荐iommu=pt作为内核引导参数
- x2apic=自动版

- x2顶端
  此选项可帮助操作系统在高核心计数配置中更有效地处理中断。如果使用>255线程，则必须启用此选项，并且只有在需要64核CPUSKU时才需要启用此选项。

- APBDIS（算法性能提升禁用）1=启用(即。已
  禁用APB）
  0=被禁用(即。APB已启用）

  此设置控制了EPYC上无限结构的提升行为。对于HPC的工作负载，我们建议使用一个固定的频率，即。不提升，因此将APBDIS状态设置为“1”。在某些OEMBIOS中，一旦设置为1，用户将看到新的SOCP状态，需要设置为P0（内存“p状态”不是罗马。最高性能内存状态）

- 核心性能提升=关闭|开启
  打开或关闭所有核心上的增压功能。例如，这也可以通过Linux命令行作为RHEL/CentOS中的根目录进行切换(这需要在启动时在BIOS中已经启用/打开核心性能提升。如果在BIOS中设置为OFF，则不能在Linux命令行上切换）。

- 内存速度=自动运行
  自动系统将允许系统自动训练到给定的DIMM人口和内存等级的正确的速度设置。如果用户愿意，他们可以按此时钟计时，例如。对于对内存速度不敏感的应用程序，因此节省电源并提供更大的提升净空间

- 核心C州=已启用
  参考CPUC状态。保持这些选项已启用。如果需要，用户应通过命令行作为根目录
  禁用C2（请参阅下文）

- DFC状态（数据结构C状态）
  在长时间空闲时间时，无限结构可以进入较低的c状态以节省电力。对于HPC工作负载，应该禁用此功能。如果从操作系统中禁用了核心C2，则此设置并不重要。

- NPS=1|2|4

通过插入内存通道对来设置每个套接字的NUMA域。。在许多HPC应用程序中，秩和内存可以固定在核心和NUMA节点上，此例中的典型建议是使用NPS4选项。如果您的工作量不是很清楚NUMA，或者在NUMA复杂性增加时遭受影响，您可以尝试使用NPS1。

- 首选的输入输出控制装置
  对于具有单个MellanoxPCI卡的系统，在使用高性能低延迟互连如Mellanox的Infini带结构时，需要启用此功能。此设置将会发生
  1) 为单个PCI设备提供增强的优先级[在BIOS中：首选IO设备]
  2) 增加PCI时钟“LCLK”[在BIOS：增强首选IO模式]
  根据OEMBIOS，它将要求使用PCI设备或PCI插槽。提前使用lspci来确定Mellanox卡位于哪个PCI设备上，例如：

  这需要一个基于AMDsAGESABIOSv1.0.0.5及更高版本的BIOS。对于基于早期AGESA的BIOS，请咨询您的系统集成商。

```
[杰森@我的系统~]$lspci|grep-iMel
c1：00.0英频带控制器：MT28908系列c1：英频带控制器：Mellanox技术MT28908系列[ConnectX-6]
```

- CCD控制装置
  此选项允许用户修改处理器中活动CCD的数量。它可与核心控制系统结合使用，以改变零件的有效布局。
- 取芯控制
  此选项允许您修改CCX中活动内核的数量。这些选项列为（x+x），其中x是每个ccx的活动内核数。例如：将其设置为（2+2）意味着每个CCX有2个活动核，每个CCD有4个活动核。如果零件有8个CCD，那么您总共有32个核心。每个CCD有4个芯*8个CCD，=有32个总芯。

  注意：CCD和核心控制选项通常用于模拟工作负载下不同部件配置的行为。此实验可以帮助您确定适合工作量和需要的最佳部件配置。

  设置每个L3高速缓存的活动核心数。对于64核部件，自动每L3留下4个核活动。其他选项还包括：
  - 每CCX有3-3：3核激活，即。关闭每CCX1芯
  - 每个CCX激活2-2：2个核，即。每CCX关闭2芯
  - 每CCX1-1：1核心激活，即。每CCX关闭3芯

  用户可能希望禁用核心，以最大化某些代码上的每核心13缓存比率。

- 内存频率、无限结构频率、耦合与非耦合模式
  - 存储器时钟和无限结构时钟既可以在同步频率下运行，也可以被称为耦合模式，也可以在异步频率下运行，也可以被称为非耦合模式。

- AMDEPYC支持高达3200MT/s的DDR4频率，但是结构时钟可以同步至2933MT/s（对于低功率组B基础设施部件，或2667MT/s）。
- 如果内存的时钟为或低于2933MT/s，则内存和结构将始终以耦合模式运行，从而提供最低的内存延迟。
- 如果您以3200MT/s的速度运行DDR4内存，则内存和结构时钟将以非耦合模式运行。这提供了稍高的带宽，但代价是增加了内存延迟。
- 如果系统支持3200MT/s内存，您可以尝试2933MT/s的耦合模式和3200MT/s的非耦合模式，以确定哪一种最适合您的工作负载。
- 在BIOS中，将内存频率设置为所需速度，并确保APBDIS设置为1，并且固定的SOCP状态设置为P0。

- 首选的IO
  首选IO允许将系统中的一个PCIE设备配置为首选状态。该装置在无限大织物上得到了优先处理。这通常为提供系统之间互连的结构适配器启用。
- tsme=关闭

  安全内存加密（加密所有内存）

Chapter 6 操作系统

6.1 Linux内核的注意事项

自从EPYC发布以来，已经发布了一些专门与EPYC相关的补丁。您可以选择手动应用这些打补丁程序，也可以选择部署具有适当的最新内核的操作系统，以避免手动打补丁程序。

红帽/中心

至少使用RHEL/Centosv7.6和内核3.10-957。

Kernel.org

至少是内核4.19。与幽灵和崩溃相关的补丁在4.15版本中进入了内核。

假设

SLES12P4和115P1都支持AMDEPYC罗马。

6.2 /proc和/系统

在系统正常运行时间内可以消耗内存。以下总结了有关NUMA系统应熟悉的一些领域，它们会影响性能并启用内存“清理”。关于Linux虚拟内存系统的讨论可以通过内核文档(https://www.kernel.org/doc/Documentation/sysctl/vm.txt)

/proc/sys/vm/zone_reclaim_mode

在kernel.org中，“Zone_reclaim_mode允许用户设置积极的方法，以在区域内存耗尽时恢复内存。如果设置为零，则不会进行区域回收。
系统将满足其他区域/节点的分配。”

内核行为由以下三位来控制：

1=区域回收时间

2=区域回收写入脏页4=区域回收交换页

这些可以在逻辑上或得到。允许设置为3（1+2）。

对于从缓存文件系统数据而受益的工作负载，区域回收通常会关闭；但这是一种平衡，如果作业大小超过了NUMANode的内存，和/或您的作业是多代码的，并扩展到NUMANode之外，那么打开它可能是明智的。阅读这篇文章了解更多细节， ：
https://www.kernel.org/doc/Documentation/sysctl/vm.txt

注意，所有2PEPYC系统都对远程套接字实现区域回收。此设置的建议设置的值为1或3。

/proc/sys/vm/drop_caches

在Linux中运行应用程序后，您可能会发现，可用内存减少，而缓冲区内存增加，尽管没有运行任何应用程序，例如。

```
[root@mun-smc05 ~]# free -g
              total        used        free      shared  buff/cache   available
Mem:            251           0         233           0          17         248
Swap:             3           0           3
```

发出numactl-H将显示内存缓冲的NUMANode（可能是所有）。在Linux中，用户可以通过3

种方式清理缓存，以将缓冲或缓存内存返回为“空闲”：

echo1>/proc/sys/vm/drop_caches[释放页面缓存]
echo2>/proc/sys/vm/drop_caches[释放板对象。数据项，内部段
echo3>/proc/sys/vm/drop_caches[清理页面缓存和板对象]用户需要是根用

户或具有SUDO权限才能执行上述操作。

```
[root@mun-smc05 ~]# free -g
              total        used        free      shared  buff/cache   available
Mem:            251           0         250           0           0         249
Swap:             3           0           3
```

在HPC系统上，在作业完成后，在为下一个用户分配同一个节点之前，清理内存通常很有用。例如，SLURM可以使用尾声脚本来完成这一点。如果使用BIOS选项NPS=4运行，这一点尤其重要，其中旧的I/O内存缓存缓冲区可以很容易地消耗一个NUMA区域中的所有内存，并且未来的分配溢出到非本地内存。

cat、程序、系统、vm、交换
性10

建议禁用交换，以防止任何不需要的交换使用。如果您需要使用交换，您需要为节点购买更多的内存容量。

请确保您的节点有足够的内存来处理工作负载。禁用没有足够内存的交换可能会产生不期望的效果。

浅a

6.3 透明的大页面(THP)

如果需要大页面，用户可以禁用透明的大页面

echo“从不”>/sys/内核/mm/transparent_hugepage/启用echo“从不”>/sys/

内核/mm/transparent_hugepage/碎片整理

6.4 页页

如果应用程序提供了对显式主页的支持，则可以禁用透明的大型页面。按照应用程序的指南在启动时分配显式主页。例如，对于HPL，通过禁用THP和启用主页，可以实现一个很小的增益。

要保留240GB和200万大页：

```
echo3>/proc/sys/vm/drop_cachesecho1>/proc/s

ys/vm/compact_memory

$number_huge_2m_pages=240*1024/2

echo$number_huge_2m_pages>/proc/sys/vm/nr_hugepages
```

其中，$number_huge_2m_pages=240*1024/2。使用壁虎（保留主页）执行mybinary.bin

```
壁力预加载堆mybinary.bin
```

这比THP还有另一个优势：如果大页面不足或者无法分配二进制文件所需的主页，代码会警告您

6.5 Randomize_va_space

地址空间随机化是一个安全功能，并防止安全黑客攻击。已禁用的功能

```
echo0>/proc/sys/内核/randomize_va_space
```

6.6 NUMA平衡

NUMA平衡是一个功能，它允许操作系统扫描内存并尝试迁移到逻辑上更接近访问它的核心的DIMM。但是，扫描“错误NUMA位置差”页面存在开销；操作系统正在猜测用户的NUMA分配。如果NUMA本地访问权限非常差，这可能非常有用。

对于大多数HPC代码来说，禁用NUMA平衡是有利的，因为它可能引入跨节点引入变异性。它也具有一个性能开销。例如，启用NUMA平衡时，流内存带宽基准运行速度稍慢。禁用与使用

```
echo0>/proc/sys/内核/numa_balancing
```

更多详情可在这里找到：

https://documentation.suse.com/sles/15-SP1/html/SLES-all/cha-tuning-numactl.html

https://access.redhat.com/documentation/enus/red_hat_enterprise_linux/7/html/virtualization_tuning_and_optimization_guide/sectvirtualization_tuning_optimization_guide-numa-auto_numa_balancing

6.7 幽灵和崩溃

谷歌项目0(GPZ)在2018年初宣布了几个关于投机执行的漏洞，需要三种变体。AMDEPYCCPU不受“变体-3”的影响，也被称为“崩溃”。AMDEPYCCPU受变体1和变体2“幽灵”的影响。AMD首席技术官对这些漏洞的更完整讨论：

https://网址为www。放大的。com/en/corporate/speculative-execution-previous-updates#paragraph-337801

较新的内核（参见本指南中Linux内核章节）将自动应用补丁以防止这些。然而，它确实对性能有很小的影响（只有几个%）。
一些客户正在选择让其“边缘节点”或“头”节点[从组织连接到外部的节点]针对这些漏洞进行修补，但正在选择关闭那些在组织内（逻辑）安全的计算节点上的修补程序。

红帽已经详细描述了此过程：https://access.redhat.com/articles/3311301总

之，root可以通过在两个文件中将单个条目值设置为“0”来关闭这些过程：

echo0>/sys/内核/调试/x86/retp_enabledecho0>/sys/内

核/debug/x86/ibpb_enabled

然后，可以查看两个文件，检查幽灵补丁是否已被禁用：

cat/sys/内核/调试/x86/retp_enabledcat/sys/内

核/调试/x86/ibpb_enabled

Chapter 7 图书馆和 编译器

7.1 AOCC-AMD编译器

AMD库、编译器、用户指南在接受条款并开始下载时，下载表单不需要登录。除了OpenBLAS之外，本章中的所有内容都可从AMD开发人员门户网站获得。

AOCC指的是AMD编译器套件；AOCL指的是AMD数学库套件。两者都在积极开发中，是AMDs战略的焦点。AOCC和AOCL的最新主要更新，2.0版本，在罗马发布前不久发布。2.1于2020年1月发布，包括2.0的几个关键更新

AOCC由C/C++编译器(clang)和一个Fortran编译器(flang)组成，可以从中下载

- AOCC(AMD编译器) https://developer.amd.com/amd-aocc/
- AOCL(AMD库) https://developer.amd.com/amd-aocl/

AOCC编译器系统是一个高性能、生产质量的代码生成工具。在构建和优化针对32位和64位Linux®平台的C、C++和Fortran应用程序时，AOCC环境为开发人员提供了各种选项。AOCC编译器系统提供了高水平的高级优化、多线程和处理器支持，其中包括全局优化、矢量化、过程间分析、循环转换和代码生成。AMD还提供了高度优化的库，它在利用时从每个x86处理器核心中提取最佳性能。

- 调谐了AMD系列17h处理器
- AMDEPYC7xx2系列性能
- 增强了对AMDEPYC7xx2系列体系结构的高级优化
- 改进的Flang-作为默认的Fortran前端，添加了F2008功能
- 基于LLVM9.0发行版(llvm.org，19第2019年9月)与错误修复
- 优化的库，包括AMDLibM(libM数学库v3.3)
- LLVM链接器(lld)作为默认的链接器
- 在RHEL7.4、SLES12sp3、Ubuntu18.04LTS、Ubuntu19.04上进行测试

在附录中的DGEMM和流的生成文件中提供了传递给clang（因此是标志）的有用编译器选项的示例。

7.1.1 AOCC克隆

clang是一个C、C++和目标-C编译器，它包括预处理、解析、优化、代码生成、汇编和链接。

Clang支持-marcy=znver2标志，以为AMD的Zen27002系列基于“罗马”的x86架构实现最佳代码生成和调优。

7.1.2 AOCCFlang语言系统

Flang是为与LLVM集成而设计的福特兰前端，适合与Clang/LLVM的互操作性。它支持所有的clang编译器选项和一些特定于喇叭的选项。

AMD扩展了flang(https://github.com/flang-compiler/flang)的Github版本，后者又基于Nvidia/PGI商业Fortran编译器。

7.1.3 有用的AOCC编译器选项

在附录中的DGEMM和流的生成文件中提供了传递给clang（因此是标志）的有用编译器选项的示例。

概述	
生成在2上运行的指令编号第EPYC代	-3月=znver2
为本地机器生成说明	-三月=本地版
OpenMP线程和亲和关系（N个核数）	导出OMP_NUM_THREADS=N 导出GOMP_CPU_AFFINITY=”0-(N1)：1”
启用向量库	-向量库=libmvec
连接到向量库的链接	-L/libm-安装/lib-lmvec
链接到AMD存储库	安装，安装
其他选项	
禁用所有的优化功能	-00
最小级别速度和代码优化	-01
中等水平的优化（默认值）	-02
积极性的优化	-03
最大化性能	-快速

积极性的优化	
启用主动式优化	-lv-功能专门化 -展开阈值=[50|100] 资金循环
启用更快、更不太精确的数学运算	数学数学 自由数学
自由压缩形式	表形式
启用链接时间优化	氟托
启用展开功能	资金循环
启用积极的循环优化	启用循环版本控制许可证 -可启用的循环分发 启用-部分解除开关 -展开攻击性
启用积极的内联优化	-功能专业化 -细线聚合物
启用主动式矢量化	记忆中积极的矢量 -可启用的条状带向量 化技术
启用内存布局优化	-框架结构-布局=[1|2|3|4|5] -fremap-数组（用于 氟）
配置文件引导下的优化	-f配置文件-实例生成（1st调用）
开放式M器	-孔
启用流媒体存储区	存储店
启用删除未使用的阵列计算	-减少阵列计算=3

7.2 GCC编译器

RHEL/CentOS7.6附带的默认编译器是4.8.5版本。对于HPC用户来说，这个GCC编译器版本通常不能提供超级计算环境中所需的性能。我们从后来的GCC版本7.3、8.1和9.1版本开始进行了测试，并使用这些测试在HPL、HPCG和DGEMM上运行和获得了良好的性能。

7.3 英特尔股份有限公司

AMD已经使用Intel编译器v18、v19和v20的初始版本对一些代码进行了有限的测试。我们在附录中演示了如何在两个编译器上的合成基准测试构建几个二进制文件。

若要使Intel数学内核库（MKLv19及更早版本）能够向AMDCPU发出AVX2指令，将需要在运行时环境中设置以下环境变量。

导出MKL_DEBUG_CPU_TYPE=5

导出MKL_ENABLE_INSTRUCTIONS=AVX2

没有这些，MKL就不会在EPYC7002系列上使用AVX2代码路径，DGEMM的运行速度比它慢很多倍。（IntelMKL库无法正确检查x86ISA标准CPUIDAVX/AVX2ISA功能位--中报告了处理器x86ISA功能
/proc/cpuinfo。）

7.4 AOCL---AMD数学库

AOCL是一组专门为AMDEPYC™处理器系列而调优的数值库。他们有一个简单的界面来利用最新的硬件创新。

更多关于AOCL、预构建库的焦油球以及如何从源代码构建AMD库的细节可以在https://developer.amd.com/amd-aocl/上获得

有关AOCL的问题，请联系您当地的AMD现场应用工程师或发送电子邮件至toolchainsupport@amd.com

7.4.1 斜叶

BLIS是一个便携式的开源软件框架，用于实例化高性能的基本线性代数子程序(BLAS)——类似于密集的线性代数库。该框架的设计目的是分离基本的计算核心，当优化时，可以立即使其大多数常用和计算密集型操作的优化实现。已针对AMDEPYC处理器系列优化了选定的内核。

GitHubhttps://github.com/amd/blis上提供了源代码

7.4.2 唇灯

libFLAME是一个用于密集矩阵计算的便携式库，提供了线性代数包(LAPACK)中存在的许多功能。它包括一个兼容层，折叠接口，其中包括完整的LAPACK实现。该库为科学和数值计算社区提供了一个现代的、高性能的密集线性代数库，可扩展，易于使用，并在开源许可下可用。与包括优化AMDEPYCTM处理器系列的BLIS库相结合，libFLAME允许在AMD平台上运行高性能的LAPACK功能。

GitHubhttps://github.com/amd/libflame上提供了源代码

7.4.3 太时间

FFTW是一个用于计算离散傅里叶变换(DFT)的快速C例程及其各种特殊情况的综合集合。它是快速傅里叶变换算法的一个开源实现。它可以计算任意大小和维数的实数和复值数组的变换。提供AMD优化FFTW，包括为AMDEPYC™处理器系列优化的选择性内核和例程。

GitHubhttps://github.com/amd/amd-fftw上提供了源代码

7.4.4 利米

AMDLibM是一个软件库，包含基本数学函数的优化
x86-64台基于处理器的机器。它提供了来自标准C99数学函数列表中的许多例程。

应用程序可以链接到AMDLibM库，并调用数学函数，而不是编译器的数学函数，以获得更好的准确性和性能。

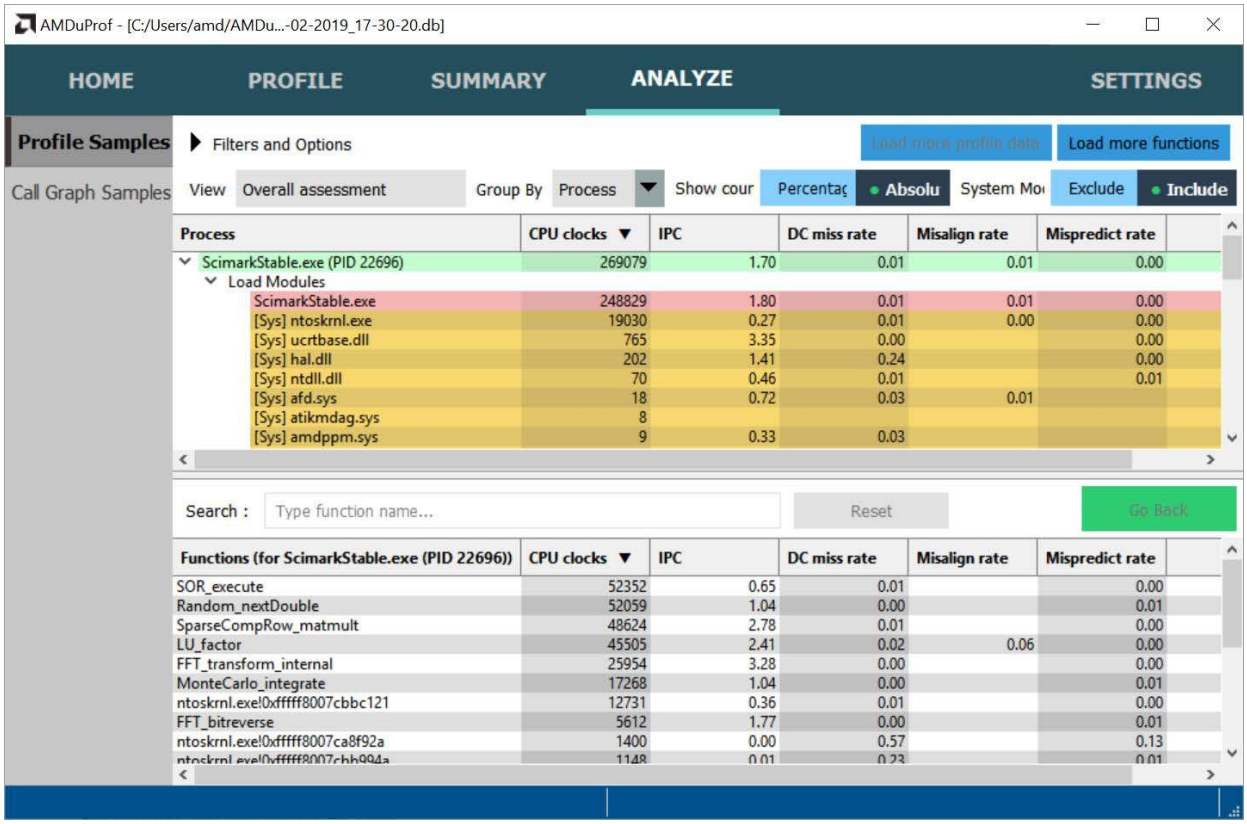
7.4.5 稻草包

ScaLAPACK是一个用于并行分布式内存机器的高性能线性代数例程库。它依赖于外部库，包括BLAS和LAPACK来进行线性代数计算。AMD的优化版本的ScaLaPACK支持使用BLIS和libFLAME库，它们为AMDEPYC处理器系列CPU优化了密集的矩阵函数和求解器。

ScaLAPACK可以从源文件或预构建的二进制文件中安装。

可以从https://github.com/amd/scalapack.中克隆出AMD源的ScaLAPACK可以从AOCL主安装程序tar文件中安装预构建的AMD优化的ScaLAPACK。

7.5 大学教授



AMDuProf是在Windows和Linux操作系统上运行的应用程序的性能分析工具。它允许开发人员更好地了解其应用程序的运行时性能，并确定提高其性能的方法。用户可以从https://developer.amd.com/amd-uprof/免费下载

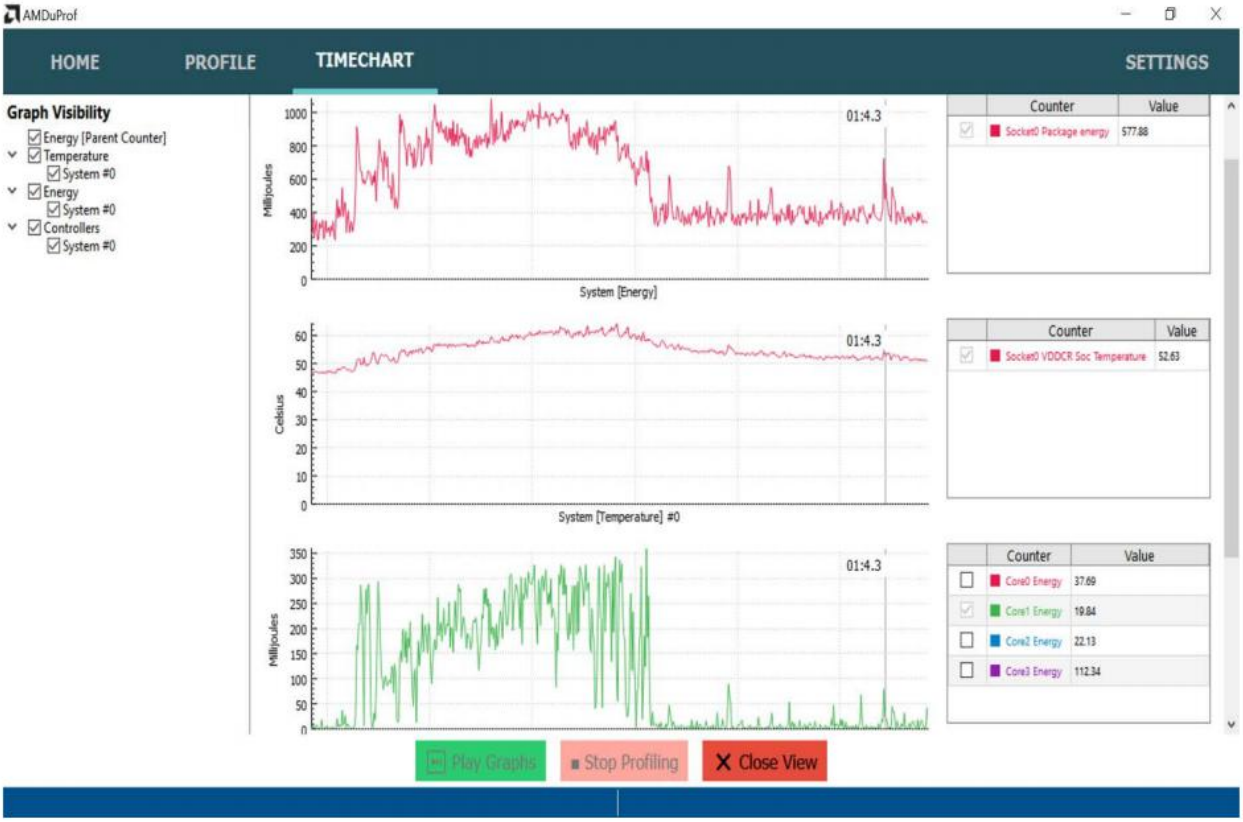
AMDuProf提供：

- 性能分析
  - CPU分析—以识别应用程序的运行时性能瓶颈。
- 能量分析
  - 电源应用程序分析——识别应用程序中的能源热点（仅限窗口）。
- 功率分析
  - 全系统的功率分析——监控系统的热特性和功率特性。
- 系统分析
  - 性能计数器监视器实用程序—用于监控系统性能指标（仅限Linux和FreeBSD）

AMDuProf可有效地用于：

- 分析一个或多个进程或整个系统的性能
- 跟踪源代码中的性能瓶颈（热点和微体系结构）
- 确定优化源代码以提高性能和电源效率的方法
- 检查内核、驱动程序和系统模块的行为
- 分析线程并发性
- 观察频率、热特性和功率特性（功率轮廓）
- 观察系统指标，如IPC、核心有效频率、内存带宽等。

AMDuProf分析仪可用于监测系统中各部件的频率、热量和能量指标。GUI提供了时间艺术页面中各种度量的实时时间线图。



Chapter 8 正在执行 应用程序 开启 放大日期 电电 7002 一系列
处理器

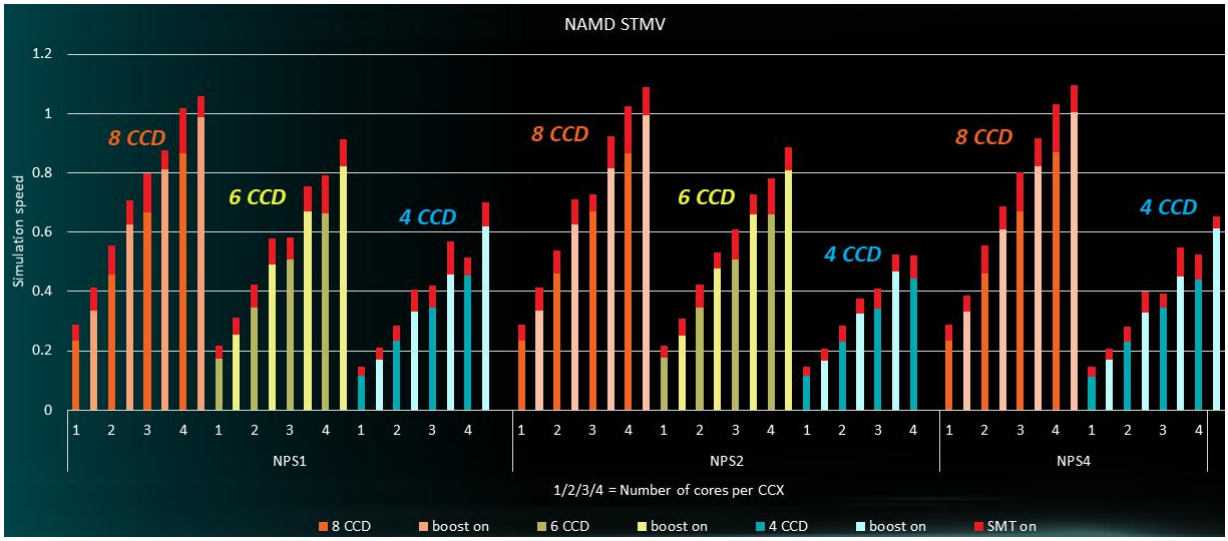
8.1 特征策略

在决定哪些系统设置最适合应用程序时，可能需要考虑多个设置来进行调整，以获得最佳的HPC服务器。以下示例表显示了跨4个不同CPU核心计数/配置的SMTON/OFF、BOOstN/OFF的矩阵。这种类型的表格可以帮助指导您的调查和测试计划。

芯芯	芯芯	smt=关闭		smt=已打开	
		启动=并关闭	启动=电源	启动=并关闭	启动=电源
1	16				
2	32				
3	48				
4	64				

此具体示例提供了一种方法来了解SMT的好处，并了解每L3增加核心计数的好处，例如可能超过3核的性能落后，这可以了解执行该应用程序的最合适的设置，甚至是整个采购（然而，更多的核心也可以在关键时刻提供更大的爆发吞吐量能力）。对于NPS的不同设置和CCD的数量，也可以重复上表。

以下是用STMV测试用例描述分子动力学应用NAMD的例子（更高更好）：



具有2x64核7742CPU的服务器用于上述特性研究，每个L3使用1/2/3/4核(即。每个插座64核)在具有2x7742CPU的系统中。使用这种CPU，我们可以灵活地测试NPS=1/2/4，并将CCD计数从8个减少到4个CCD。这说明我们建议推荐64、48还是32核心CPU来提供最佳性能。

对于一个给定的核心计数。每L31芯）有2杆：左杆提升=，右杆提升=。酒吧顶部的红色部分显示了SMT=ON的增量优势

通过NPS=4、8xCCD、每L3（64核CPU）Boost=ON和SMT=ON实现最佳性能。

8.2 配对策略和混合代码

以OpenMPI为例，我们通常找到最可靠的固定核心的方法是通过应用文件。

我们发现，如果我们依靠OPenMPI的接口来获取CPUID（“-cpu-list”），则性能略有降低

我们还发现，当使用套接字中的所有核心/线程时，仍然强烈建议显式钉扎，以确保最小的执行时间。

对于混合MPI+OpenMP代码，通常在执行代码时可采用三种策略进行选择：

1. 每个可用线程的所有MPI排名
2. 按L3绘制：每L3的MPI等级
    a. 如果SMT=OFF，则OMP_NUM_THREADS通常设置为驻留在该L3缓存上的内核数。如果它是一个64核的CPU，那么它是OMP_NUM_THREADS=4

    b. 如果为SMT=ON，则OMP_NUM_THREADS通常设置为驻留在该L3缓存上的线程数。如果是4核部分，则是OMP_NUM_THREADS=8

3. 按核心映射：如果SMT=ON，则用户可以选择每个核心+OMP_NUM_THREADS=1MPI排名2。

例如，对于使用2x7742的OpenMPI执行my二进制文件。2x64个核心CPU，具有：

- smt=关闭
- 每L3有1个MPI排名，没有OpenMP线程

```
出口计算机=$（秒，0 4 127）

绑定到无cpu列表$c列表-mcapmlucx-mcaoscucx\\

^自我，vader，开放的-mcacoll_hcoll_enable0\\
-xUCX_NET_DEVICES=mlx5_2：1-xUCX_TLS=自我，sm-xPMIX_MCA_gds=^ds21\\
——应用程序myappfile-1rankperL3.txt
```

本例中的无限波段卡是一个表示为mlx5_2的双端口设备，我们使用端口1，因此可以使用UCX_NET_DEVICES=mlx5_2：1

随附附加文件myappfile-1rankperL3.txt如下：

```
-np  1  数字数    --植物菌毛=0肌二元平行
-np  1  数字数    --门菌属=4my二元平行
-np  1  数字数    ——门菌骨=8肌二元平行
-np  1  数字数    --门耻骨=12肌二元平行
-np  1  数字数    门耻=16肌二元平行
-np  1  数字数    门菌=20二元平行
-np  1  数字数    --门耻骨=24肌二元平行
-np  1  数字数    --门耻骨=28肌二元平行
-np  1  数字数    --门耻骨=32肌二元平行
-np  1  数字数    --门耻骨=36肌二元平行
-np  1  数字数    ——门耻骨=40my二元平行
-np  1  数字数    --门耻骨=44肌二元平行
-np  1  数字数    --门耻骨=48肌二元平行
-np  1  数字数    --门菌属=52肌二元平行
-np  1  数字数    --门耻骨=56肌二元平行
-np  1  数字数    ——门耻=60my二元平行
-np  1  数字数    --门耻骨=64肌二元平行
-np  1  数字数    门耻=68肌二元平行
-np  1  数字数    --门菌属=72肌二元平行
-np  1  数字数    --门耻骨=76肌二元平行
-np  1  数字数    ——门耻=80my二元平行
-np  1  数字数    --门菌属=84my二元平行
-np  1  数字数    --门阴茎=88肌二元平行
-np  1  数字数    --门菌属=92肌二元平行
-np  1  数字数    --门菌属=96肌二元平行
-np  1  数字数    --门菌属=100肌二元平行
-np  1  数字数    --门菌属=104肌二元平行
-np  1  数字数    --门菌属=108肌二元平行
-np  1  数字数    --门菌属=112肌二元平行
-np  1  数字数    门耻=116肌二元平行
-np  1  数字数    --门菌属=120my二元平行
-np  1  数字数    --门耻骨=124肌二元平行
```

Chapter 9 附录

9.1 双姆

它的作用是：强调计算周期。计算从以下位置提供的核心/CCX/NUMA节点/套插字的故障：

http://portal.nersc.gov/project/m888/apex/

您还需要从https://developer.amd.com/amd-aocl/blas-library/下

载BLIS多线程(MT)库

生成文件：

```
CFLAGS=-Ofast-USE_CBLASUSE_CBLAS-mavx2-乐趣-循环-lomp\\
          =快速=znver2月=znver2
LIBS=/home/software/aocl/aocl-2.1-1910/amd/aocl/2.1-1910/amd-blis-mt/lib/libblis-mt.a INC=-
```

执行情况

如果用户希望在双套接字2x7742系统中运行套接字-0上的所有64个内核，则使用AOCC编译时：

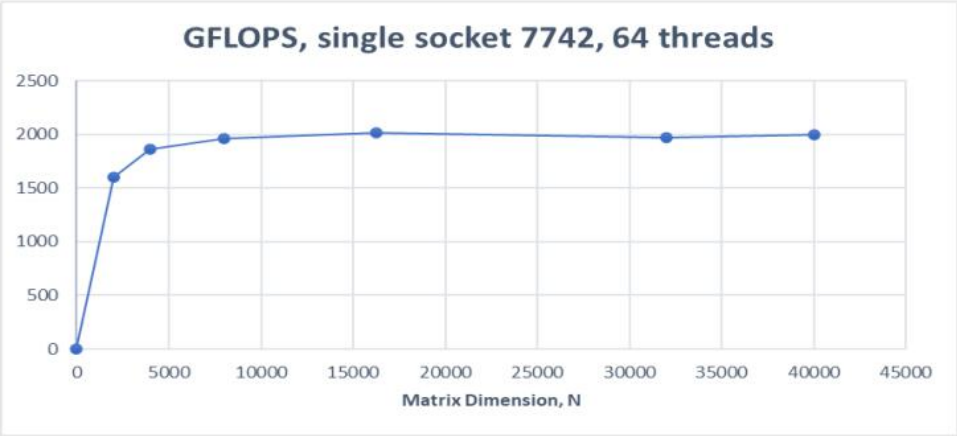
omp_num_threads=64gomp_cpu_affinity=”0-63：1”　　\\

模因绑定=0-3。**/mt-dgemm.aocc8000**

通过仔细选择针芯，可得出其他组合（根据L3，根据CCD）。测试结果

在具有2x7742和16x64GBDDR4-3200R2的系统上，加大=Off、SMT=OFF

理论峰值为每周期2.25GHz*16次下降*64芯=2304GFLOPS，即。大约2000/2304=87%的效率。



9.2 hpl

项目简介：

HPL是一个软件包，它在分布式内存计算机上以双精度（64位)运算解决(随机）密集线性系统。HPL可以从其官方网站https://下载www.netlib.org/benchmark/hpl/先决条件：

对于使用AOCC和AOCL在AMD平台上构建和运行HPL，我们所需要的。

1. 安叶片
2. 玉马
3. 屈骨板
4. 打开的MPI

这些都是构建一个优化的OpenMPI库所必需的。AMDBLIS的预构建库可以从 https://developer.amd.com/amd-aocl下载

为了“快速启动”，用户可以构建没有knem或jemalloc，然后使用下面的Make.zen文件构建HPL：使用HPL绑定到核心运行。/xhpl

设置以下环境变量：

出口CC=clang出口FC=法

朗出口CXX=clang++出口

F90=法朗出口F77=法朗

出口AR=llvm-ar

导出RANLIB=llvm-牧场导出

NM=llvm-nm

导出编译器根=<路径到AOCC编译器根目录>导出OMPI=<路径导出到AOCC包括目录>

将OMPL=<路径导出到AOCC库目录>

现在，请设置一些编译标志：

导出CFLAGS=”-O3-数学-三月=znver2-我${OMPI}“导出cxx标志=”-O3-数学-三月=2-

i${OMPI}“导出标志=”-o3-数学-三月=2f-i${OMPI}“导出标志=”-L${OMPL}”

导出包括=$OMPI：$包括

导出路径=$编译器根/bin：$路径

导出LD_LIBRARY_PATH=$OMPL：$LD_LIBRARY_PATH

生成Jemalloc

Jemalloc是一个通用的malloc（3）实现，它强调避免碎片化和可扩展的并发支持。

导出内存分配根目录=<JEMALLOC_ROOT_PATH>

git克隆

https://github.com/jemalloc/jemalloc.githttps://github.com/jemalloc/je

malloc.git

。/autogen.sh

c标志=$标志　　。/配置--前缀=$内存分配根制造-j

进行安装

生成KNEM

KNEM是一个Linux内核模块，可为大型消息进行高性能的节点内MPI通信。KNEM通过Linux内核中的单个副本将数据从一个进程传输到另一个进程。

导出十根根=<KNEM_ROOT_PATH>

git克隆，https://gforge.inria.fr/git/knem/knem.git膝盖，膝盖

。/autogen.sh

。/配置--前缀=$knemrootCFLAGS=“$CFLAGS　　\\

-I$“$=”$\\

-L$”主机=x86_64清除

制造

进行安装

生成OpenMPI

一个高性能的消息传递库。开放的MPI项目是一个开源的消息传递接口实现。官方网站：https://www.open-mpi.org/

获取https://download.open-mpi.org/release/open-mpi/v4.0/openmpi4.0.0.tar.bz2

tar-xvfopenmpi-

4.0.0.tar.bz2cdopenmpi-4.0.0

。/配置-前缀=$打开根-与knem=$开根

\\

cc=${cc}cxx=${cxc}fc=${fc}c标志=“${c标志}”　　\\

cxx标志=“$”=“${fc旗帜}”　　\\

启用共享=是，启用静态=是　　\\

-启用mpi1兼容性制造-j

进行安装

ln-s$OPENMPIROOT/lib/libmpi.so.40.20.0$OPENMPIROOT/lib/libmpi.so.20

导出路径=$打开管道根/bin：$路径

导出LD_LIBRARY_PATH=$OPENMPIROOT/lib：$LD_LIBRARY_PATH

我们现在可以开始构建HPL了

导出hplroot=<HPL_ROOT_PATH>

获取https://www.netlib.org/benchmark/hpl/hpl-2.3.tar.gz焦油-xvfhpl-2.3.tar.gz

mvhpl-2.3hpl使拱门=zen

使用以下示例Make.zen文件

```
壳牌            =/压缩机箱/sh
CDcpln          =cd
_smkdi          =cp
rrm触           =ln-s
摸拱门          =毫米号
例如，          =/压缩机箱/rm-f
LAdirLA         =触摸屏
incLAli         =$（拱门）
b               = ../../..
                =$（目录）/包括
                =$(TOPdir)/bin/$(ARCH)
                =$(TOPdir)/lib/$(ARCH)
                =$（目录）/目录。a
                =${开放根}
                =-I$(MPdir)/包括
DF77_INTEGER=短：Fortran77整数是一个C短。f2cdefs        =-Dadd-
DF77_INTEGER=标准样式
HPL_INCLUDES=-I$（插入）-I$（插入）/$(ARCH)$(LAinc)$(MPinc)HPL_LIBS
                =$(HPLlib)$(LAlib)$(MPLib)-lm
hpl_opts        =-dhpl_progress_report
hpl_defs        =$(f2cdefs)$(hpl_opts)$(hpl_includes)
抄送器          =叮声
计算机公司      =$(hpl_defs)
cc标志          =$(HPL_DEFS)-O3--数学-有趣的循环-三月=znver2-fopenmp\\
    -I$包含/包含/I$包含}/包含链接器      =叮声
链接标志        =-o3-数学-乐趣循环-3月=znver2\\
    -L$/lib$(ccflagos)-L$-$-lm存档       =有限公司
管道子
```

如果尚未设置，请进行以下操作系统更改

回声1 <42>/系统、设备、系统、cpu、计算机/提升echo性能>/sys/devices/system/cpu/cpu0/cpufreq/scaling_governorecho0

<randomize_va_space2>/程序/sys/内核/randomize_va_space

回声“从不” <transparent_hugepage2>/系统/内核/mm/transparent_hugepage/启用的echo“从不” <transparent_hugepage2>/系统/内核/mm/transparent_hugepage/碎片整理回声0 <ptrace_scope2>/proc/sys/内核

/yyama/ptrace_scope

echo0 <numa_balancing2>/程序/sys/内核/numa_balancing

回声3 <drop_caches2>/proc/sys/vm/drop_caches

回声1 <compact_memory2>/proc/sys/vm/compact_memory

对于HPL的情况，我们发现通过禁用透明的大页面并使用拥抱页来提供一个提升。在一个拥有256GB主内存的系统上，你可以设置240GB的2MB主页

回波122880 <nr_hugepages2>/proc/sys/vm/nr_hugepages

对于HPL的情况，我们发现通过禁用透明的大页面并使用拥抱页来提供一个提升。

我们现在可以开始运行HPL了。同样，用户可以执行一个简单的操作（假设为OpenMPI）：

米润连到核心。/xhpl

但为了进一步获得最大的性能，我们需要

1. 设置BLIS的内部循环
2. 使用混合的MPI+OpenMP
3. 显式地将每个线程固定到一个特定的CPUID上

确保加载膝盖骨模块并执行（需要root/sudo执行拥抱）：

导出mpi_options=“——mcampi_leave_pinned1——绑定到无

报告绑定mca自我，映射：1：l3cache缓存

-xOMP_NUM_THREADS=4-xOMP_PROC_BIND=真机-xOMP_PLACES=核心“

mpirun$mpi_options-app。/appfile_ccx

```
-np 1  。/xhpl_ccx.sh  0  0-3 4
-np 1  。/xhpl_ccx.sh  0  4-7 4
-np 1  。/xhpl_ccx.sh  0  8-11 4
-np 1  。/xhpl_ccx.sh  0  12-15  4
-np 1  。/xhpl_ccx.sh  1  16-19  4
-np 1  。/xhpl_ccx.sh  1  20-23  4
-np 1  。/xhpl_ccx.sh  1  24-27  4
-np 1  。/xhpl_ccx.sh  1  28-31  4
-np 1  。/xhpl_ccx.sh  2  32-35  4
-np 1  。/xhpl_ccx.sh  2  36-39  4
-np 1  。/xhpl_ccx.sh  2  40-43  4
-np 1  。/xhpl_ccx.sh  2  44-47  4
-np 1  。/xhpl_ccx.sh  3  48-51  4
-np 1  。/xhpl_ccx.sh  3  52-55  4
-np 1  。/xhpl_ccx.sh  3  56-59  4
-np 1  。/xhpl_ccx.sh  3  60-63  4
-np 1  。/xhpl_ccx.sh  4  64-67  4
-np 1  。/xhpl_ccx.sh  4  68-71  4
-np 1  。/xhpl_ccx.sh  4  72-75  4
-np 1  。/xhpl_ccx.sh  4  76-79  4
-np 1  。/xhpl_ccx.sh  5  80-83  4
-np 1  。/xhpl_ccx.sh  5  84-87  4
-np 1  。/xhpl_ccx.sh  5  88-91  4
-np 1  。/xhpl_ccx.sh  5  92-95  4
-np 1  。/xhpl_ccx.sh  6  96-99  4
-np  1  。/xhpl_ccx.sh  6 100-103   4
-np  1  。/xhpl_ccx.sh  6 104-107   4
-np  1  。/xhpl_ccx.sh  6 108-111   4
-np  1  。/xhpl_ccx.sh  7 112-115   4
-np  1  。/xhpl_ccx.sh  7 116-119   4
-np  1  。/xhpl_ccx.sh  7 120-123   4
-np  1  。/xhpl_ccx.sh  7 124-127   4
```

为此，您需要使用以下文件：appfile_cxx（执行钉扎）和xhpl_ccx.sh（在内部BLIS循环上按每L3设置核心）

上面的应用文件调用了xhpl_ccx.sh：

```
#! /bin/bash#
#将内存绑定到节点$1，并将四个子线程绑定到$2中指定的CPU#内核并行化在第二个最内层循环
(IC)中执行

导出LD_LIBRARY_PATH=$BLISROOT/lib：$LD_LIBRARY_PATH导出
LD_LIBRARY_PATH=$OPENMPIROOT/lib：$LD_LIBRARY_PATH
导出OMP_NUM_THREADS=$3导出
GOMP_CPU_AFFINITY=“$2”导出
OMP_PROC_BIND=TRUE

#BLIS_JC_NT=1（无外环并行化）：
导出BLIS_JC_NT=1
#BLIS_IC_NT=每个L3的核心数（二级线程数-#共享L3缓存域中的每个核心1个）：
导出BLIS_IC_NT=$OMP_NUM_THREADS
#BLIS_JR_NT=1（没有第四级线程）：
导出BLIS_JR_NT=1
#BLIS_IR_NT=1（没有第五级线程）：
导出BLIS_IR_NT=1
```

以下是一个用于罗马架构的HPL示例，具有256GB内存和2x64AMDEPYC774264核处理器。请根据系统的配置更改此文件：更改NB、N、P和Q。

```
HPLinpack基准测试输入文件
创新计算实验室，田纳西大学HPL.out       输出文件名（如有）
6              设备输出（6=支架，7=支架，文件）
1              问题大小的数量(N)
171776         Ns
1              NBs的数量
244            问题大小的数量(N)
0              MAP流程映射（0=行，1=主列）
1              工艺网格的数量(PxQ)
4              Ps
8              Qs
16.0           阈值
1              面板事实的#<
2              PFACT（左0=，1=螺纹，右2=）
1              递归停止标准的数量
4              纳米薄荷（>=1）
1              递归过程中的面板的数量
2              NDIVs
1              递归式面板事实的数量。
1              RFACT（左0=，1=螺纹，右2=）
1              广播的次数
1              蓄电池（0=1rg、1=1rM、2=2rg、3=2rM、4=Lng、5=LnM）
1              超前观测深度数
0              删除程序（>=0）
2              交换（0=箱，1=长，2=混音）
```

使用上述过程在2套接字测试服务器上的AOCLv2.1，我们推导出(boost=on，cTDP和PPL设置为最大，确定性滑块=Power)

- 2x7742（2x64芯）：3.81TFLOPS
- 2x7532（2x32芯）：2.41TFLOPS

9.3 小溪

它的功能：测试核心的最大内存带宽，或整个CPU，例如可从：

http://www.cs.virginia.edu/stream/

安装程序：我们在这里提供了一些使用3种不同的编译器的例子：GCC，Intel，AOCC

在64核CPU上，每L3缓存1核可实现最大内存带宽，即。每个CPU有16个核。因此，在具有2x64核的双套接系统上，我们需要设置以下环境变量。如果系统已经随所有核心一起启动，活动用户可以通过适当的核心钉扎来实现最大内存带宽。需要以下操作系统设置（根/子操作权限）：

```
#! /bin/bash
echo0>/proc/sys/内核
/randomize_va_spaceecho0>/proc/sys/vm/nr_hugepages
echo0>/proc/sys/内核/numa_balancing
echo “从不”>/sys/内核/mm/transparent_hugepage/启用echo “从不”>/sys/内核
```

使用相应编译器的以下环境变量来运行

海湾合作条款	导出OMP_NUM_THREADS=32 导出GOMP_CPU_AFFINITY=0-127：4
英特尔国际广播公司	导出OMP_PROC_BIND=真导出 OMP_NUM_THREADS=32 导出OMP_PLACES=“$(回声“{”；seq-s}，{0 4 127；回声“}”)”
宏克	导出OMP_SCHEDULE=静态导出 OMP_DYNAMIC=假导出 OMP_THREAD_LIMIT=256导出 OMP_NESTED=假导出 OMP_STACKSIZE=192M 导出OMP_NUM_THREADS=32 导出GOMP_CPU_AFFINITY=0-127：4

生成文件：

```
整数：
        icc-ostream.intel-2500000000            dn时间=10                                \\
          -dstream_array_size=2500000000          -mcmodel=大型共享信息                  \\

铿锵
        clang流。c-03            -mcmodel=介质-DSTREAM_TYPE=双mavx2                      \\
          -3月=znver2        -故障展开循环              -洛普       打开存储\\
```

注意：由于编译器不启用流式存储，因此不会从GCC获得最大性能。

试验结果：
现在，我们比较2x7742系统上的流三合会结果(MB/s)，并检查将每个套接字的NUMA域从4到2更改为1(NPS)(系统在512GBDDR4-3200R2中使用64GB的效果；我们用Intel编译了流，但AOCC编译器提供了同等的性能)

这些结果表明：

CCD=8 NPS=4		SMT = OFF			SMT = ON		
		THREADS	Boost=OFF	Boost=ON	THREADS	Boost=OFF	Boost=ON
Cores/L3	1	32	354,045	354,326	64	339,322	340,233
	2	64	339,420	340,511	128	324,179	325,024
	3	96	331,466	332,242	192	313,955	314,683
	4	128	324,828	325,397	256	307,286	308,198

CCD=8 NPS=2		SMT = OFF			SMT = ON		
		THREADS	Boost=OFF	Boost=ON	THREADS	Boost=OFF	Boost=ON
Cores/L3	1	32	322,523	323,551	64	305,656	305,698
	2	64	305,658	306,082	128	293,386	293,879
	3	96	295,923	296,221	192	282,972	283,383
	4	128	285,429	286,518	256	281,536	281,792

CCD=8 NPS=1		SMT = OFF			SMT = ON		
		THREADS	Boost=OFF	Boost=ON	THREADS	Boost=OFF	Boost=ON
Cores/L3	1	32	300,768	302,505	64	287,146	287,659
	2	64	287,123	287,694	128	282,037	283,127
	3	96	282,868	283,034	192	277,052	277,656
	4	128	279,416	280,622	256	276,206	276,686

- 带宽随着我们减少每个套接字的NUMA域的数量而减少
- 设置Boost=具有边际积极影响
- 设置SMT=开启会对此合成测试产生负面影响

我们还展示了当CCD从CCD=8降低到CCD=4时性能如何变化：

ccd=4nps=4		smt=关闭			smt=已打开		
		螺纹	启动=并关闭	启动=电源	螺纹	启动=并关闭	启动=电源
芯芯	1	32	312,953	317,666	64	314,067	315,701
	2	64	311,103	312,668	128	294,007	295,190
	3	96	297,717	299,107	192	283,904	285,086
	4	128	291,366	292,545	256	278,066	279,472

9.4 Mellanox配置

本章介绍在AMDEPYCHPC群集上配置Mellanox无限带结构时的一些基本注意事项。梅拉诺克斯使用OFED中间件堆栈来操作他们的结构。虽然openfabrics.org有一个社区OFED版本，但我们建议使用Mellanox捆绑并在他们的网站上免费提供的OFED版本。

您必须在AMDEPYCRomeHPC群集上使用OFEDv4.7-3.2.9或更高版本。早期的OFED版本将不会产生正确的带宽配置文件。我们的连接X-6卡上的固件是20.26.1814.用户还需要在BIOS中启用首选IO模式（请参见前面的“BIOS设置”）。

一旦安装了OFED，系统管理器可能会发布多个测试。例如，为了确保在集群中的任意2个计算节点之间存在正确的带宽配置文件，请打开2个终端窗口并连接到节点3和节点5，例如：

- **在节点3上：数字-C20ib_write_bw-a-report_gbits**
- **在节点-5上：数字-C15ib_write_bw-a-report_gbits-dmlx5_0节点-3**

在这个例子中，我们有

- 已测试了从节点5到节点3的连接
- 使用数字来确保我们选择了“逻辑上”最接近Mellanox主机通道适配器(HCA)PCI卡的核心，或者核心是承载ConnectX-6卡的PCI插槽的“本地”（对于本地到远程，请参见附录OSU）。回顾一下，我们可以通过事先发布核心并注意核心来确定为此选择哪些核心
- 确保核心在C1状态下闲置（参见以上讨论）

我们在两个节点上都有双端口连接X-6Mellanox卡，并通过HDR200MellanoxIB网络明确声明了通过-d标志在卡上使用哪个端口，应该显示以下配置文件：

-

#bytes	#iterations	BW peak[Gb/sec]	BW average[Gb/sec]	MsgRate[Mpps]
2	5000	0.067164	0.066476	4.154756
4	5000	0.13	0.13	4.187184
8	5000	0.27	0.27	4.187324
16	5000	0.54	0.54	4.197941
32	5000	1.08	1.07	4.178680
64	5000	2.15	2.14	4.176621
128	5000	4.31	4.27	4.169307
256	5000	8.61	8.57	4.182211
512	5000	17.07	17.02	4.156448
1024	5000	34.07	33.98	4.148378
2048	5000	67.39	67.28	4.106271
4096	5000	132.84	132.15	4.032974
8192	5000	186.65	186.43	2.844643
16384	5000	191.88	191.75	1.462959
32768	5000	196.80	196.78	0.750658
65536	5000	196.44	196.44	0.374673
131072	5000	197.07	197.06	0.187927
262144	5000	197.13	197.13	0.093999
524288	5000	197.14	197.14	0.047001
1048576	5000	197.13	197.13	0.023500
2097152	5000	197.15	197.14	0.011751
4194304	5000	197.11	197.10	0.005874

也就是说，我们看到了一个200Gb/s的单向峰值带宽，随着包大小的增加，带宽稳步增加，然后随着包大小的增加而保持在峰值带宽。这是正确的配置文件，核心需要处于C1空闲状态才能实现这一点。如果您的核心在C2中空闲，那么您会在包带中间的带宽配置文件中看到一个可能短暂的最大值，之后带宽会随着包大小的增加而减小。

网络延迟也可以像上面使用ib_write_bw一样使用ib_write_lat测试网络延迟。用户应该期望在使用EPYCRome的MellanoxHDR200网络上看到大约1微秒的延迟。我们用Boost=打开2字节大小的包测量0.99微秒。

这些测试非常有用，因为它们证明没有损坏的系统组件（电缆、卡、PCI插槽等），并且系统配置正确。系统经理也应进行监控
对于任何与Mellanox相关的警告/错误的消息。

Mellanox还有效地编译了一个OpenMPI版本，并将其作为其OFED版本的一部分捆绑起来。如果您希望使用一个替代的MPI库，我们建议您阅读Mellanox是如何在其最新版本的文档中构建OpenMPI的。我们提请您注意以下应该传递给您的旗帜
。/如果MPI库支持它们，请配置：

```
。/配置与hcoll=启用兼容hcoll1/opt/mellanox/hcoll\
 --with-knem=/opt/knem-1.1.3.90mlnx1--with-ucx=/opt/ucx/1.5.1                \
--with-xpmem=/opt/xpmem/2.6.5
```

或发行的

```
。/配置-h
```

以检查MPI库是否支持这些标志。

9.5 俄亥俄州立大学的网络测试

它的作用：俄亥俄州立大学(OSU)微型基准是一套MPI、SHMEM和UPC测试，可以衡量广泛的并行消息交换操作的性能。

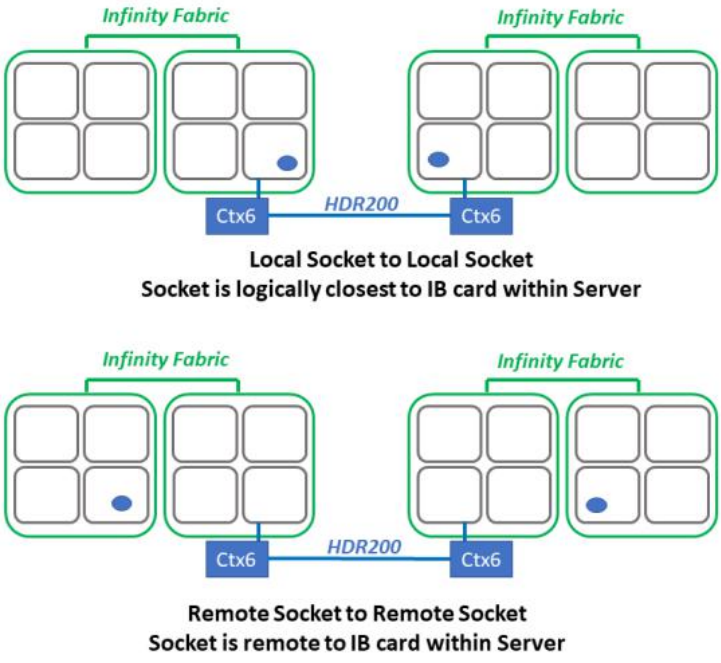
可用于：http://mvapich.cse.ohio-state.edu/benchmarks/

安装：在使用ConnectX-6卡时，有关OFED和固件的最低正确版本，请参阅附录“Mellanox配置”。我们在这里演示了3个如何运行的例子及其后续的结果：

1. 延迟：此测试在2个节点上运行，每个节点有一个核心。一个核心是发送者，而另一个节点上的核心是接收器。发送方向接收方发送具有一定数据大小的消息，并等待来自接收方的回复。接收人接收来自发送人的消息，并发送具有相同数据大小的回复。对该乒乓球测试进行了多次迭代，得到了平均单向延迟数。在测试中使用了MPI函数(MPI_Send和MPI_Recv)的阻塞版本。

2. “本地到本地”：使用CPU上对无限波段HCA为本地的所有核心。另一个CPU上的核心处于空闲。由于没有消息可以穿越无限结构，这应该会提供最高的带宽。

3. “远程到远程”：将使用CPU插接字上未连接到无限波段HCA的所有核心。这是一个最坏的情况，因为所有消息都需要首先使用服务器的无限结构。



用于测试的系统配置：

BIOS	American Megatrends Inc. Version: 5.14 Release Date: 08/15/2019
CPU	EPYC 7742 64-Core
Speed	2.25 GHz
NPS	4
System profile	Performance determinism
SMT	disabled
Boost	ON
Preferred I/O	C1
C2	disabled
CPU governor	performance
OS	CentOS 7.6
Kernel	3.10.0-957.10.1.el7.x86_64
HCA	Mellanox ConnectX-6
Speed	HDR200
OFED	MLNX_OFED_LINUX-4.7-3.2.9.0
HCA Firmware	20.26.1814

生成文件：

```
! #/bin/bash

#   延迟
米庇润与无核细胞母蛋白结合
    --mcabtlˆ自我，vader--mcacoll_hcoll_enable0-x\UCX_NET_DEVICES=mlx5_2：1-xUCX_TLS=自我，sm，
    rc_x主机节点02：1，节点03：1\
    节点-np2任务集-c92osu_latency

#从本地到本地

米庇润与无核细胞母蛋白结合
    --mcabtlˆ自我，vader--mcacoll_hcoll_enable0-x\UCX_NET_DEVICES=mlx5_2：1-xUCX_TLS=自我，sm，
    rc_x主机节点02：64，节点03：64\
    局部细胞-植物细胞=64-127\osu_mbw_mr

#远程到远程

米庇润与无核细胞母蛋白结合
    --mcabtlˆ自我，vader--mcacoll_hcoll_enable0-x\UCX_NET_DEVICES=mlx5_2：1-xUCX_TLS=自我，sm，
    rc_x主机节点02：64，节点03：64\
    数字-局部细胞-植物细胞=0-63\osu_mbw_mr
```

应选择CPUID与连接X-6卡位于相同的NUMA域中

#Size	Latency (us)	Local to Local		Remote to Remote	
		MB/s	Messages	MB/s	Messages
0	1.05	-	-	-	-
1	1.04	125.1	125069945.2	70.4	70412887.4
2	1.04	246.3	123166738.7	136.6	68285326.0
4	1.05	486.3	121561838.7	273.6	68395917.5
8	1.12	996.1	124511971.0	549.7	68707453.4
16	1.05	1943.7	121477972.2	1095.1	68446331.2
32	1.21	3924.2	122631907.4	2216.4	69261295.3
64	1.26	6531.6	102055478.6	5366.3	83847775.7
128	1.33	9216.7	72005515.4	7387.1	57711369.5
256	1.69	14971.9	58484084.7	13564.0	52984404.0
512	1.90	19486.1	38058783.0	17002.5	33208058.0
1024	2.30	22529.4	22001373.7	19901.4	19434928.7
2048	2.34	24768.5	12093987.4	21817.2	10652915.4
4096	2.92	24030.2	5866750.7	22443.3	5479313.4
8192	3.92	24300.7	2966392.3	22714.4	2772754.5
16384	4.86	24393.9	1488883.3	22832.7	1393598.1
32768	6.84	24824.9	757595.2	22840.5	697036.9
65536	10.20	24768.4	377936.4	23427.0	357467.5
131072	16.66	24725.8	188643.2	23415.2	178643.7
262144	15.64	24712.3	94269.8	23411.3	89307.1
524288	26.83	24706.3	47123.5	23407.0	44645.2
1048576	49.36	24700.3	23556.1	23404.9	22320.7
2097152	94.64	24696.9	11776.4	23403.1	11159.5
4194304	185.20	24695.7	5887.9	23405.2	5580.2

Chapter 10 资源来源

有关AMD平台上的所有开发人员的参考资料

http://developer.amd.com

AMD处理器文档和指南：针对基于AMDEPYC7002系列处理器的服务器的

https://developer.amd.com/resources/developer-guides-manuals/

工作负载调整指南

https://developer.amd.com/wp-content/resources/56745_0.80.pdf

《AMD优化CPU库用户指南》

https://developer.amd.com/wp-content/resources/AOCL_User_Guide_2.1.pdf软件优化指南为AMD系列17h型号30h处理器和更大的处理器https://developer.amd.com/wp-content/resources/56305_SOG_3.00_PUB.pdf

社区论坛关于AMD，讨论技术问题，并联系服务器大师

https://community.amd.com/community/server-gurus

其他资源

来自PRACE的最佳实践指南。那不勒斯调优指南与有用的参考为编译器选项和移植代码：

http://www.prace-ri.eu/best-practice-guide-amd-epycSLES调谐指南

https://www.suse.com/documentation/suse-best-practices/optimizing-linux-for-amd-epyc-with-sle-12-sp3/data/optimizing-linux-for-amd-epyc-with-sle-12-sp3.html

Linux虚拟内存：

https://www.kernel.org/doc/Documentation/sysctl/vm.txt红帽系列中的调谐指南：

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/performance_tuning_guide/

幽灵和崩溃：

https://access.redhat.com/articles/3311301

AMD关于幽灵和崩溃的声明：

https://网址为www。放大的。com/en/corporate/speculative-execution-previous-updates#paragraph-337801

Linux内核，CPU管理器文档：

https://www.kernel.org/doc/Documentation/cpu-

freq/governors.txt