英特尔® 至强® CPU Max 系列
配置和调优指南

2023 年 2 月

一般提示和法律声明

英特尔技术可能需要启用硬件、软件或激活服务。没有任何产品或组件是绝对安全的。

具体成本和结果可能不同。

您不得将此文件用于或协助用于任何关于英特尔产品的侵权或其他法律分析的文件。对于后续起草的包含本文所披露标的物的任何专利权利要求，您同意授予英特尔非排他的、免许可费的许可。

所有关于英特尔最新产品规格和路线图的信息可在不通知的情况下随时发生变更。

描述的产品可能包含可能导致产品与公布的技术规格有所偏差的、被称为非重要错误的设计瑕疵或错误。一经要求，我们将提供当前描述的非重要错误。

英特尔未做出任何明示和默示的保证，包括但不限于，关于适销性、适合特定目的及不侵权的默示保证，以及在履约过程、交易过程或贸易惯例中引起的任何保证。

代号用于英特尔识别研发中且尚未上市的产品、技术或服务。其并非“商业用”名称且并无意用作商标。

本文并未（明示或默示，或通过禁止反言或以其他方式）授予任何知识产权许可。如有例外须同时满足以下条件：a）发布未经修改的版本，及 b）文档中所包含的代码已获许可且受 Zero-Clause BSD 开源许可证（0BSD）之约束， https://opensource.org/licenses/0BSD。您可以在本文档基础上，按照上述规定创建旨在在本文档所提及的英特尔产品上执行的软件实现，但无权就本文档进行任何修改或创建衍生文档。

修订历史

日期	修订	描述
2023 年 2 月	001	首次发布文档

目录

表格

图表

第 1 章 简介

1.1 目标受众

本文主要面向使用英特尔® 至强® CPU Max 系列产品运行和优化应用的系统管理员和应用工程师。

1.2 术语表

表 1. 缩略词定义

缩略词	术语	定义
BIOS	Basic Input Output Service（基本输出输入系统）	
HBM	High Bandwidth Memory（高带宽内存）	
1LM	1-Level Memory Mode or FLAT Mode（一级内存模式或“Flat”模式）	HBM 和 DDR 作为独立地址空间对软件可见的模式。
2LM	2-Level Memory (2LM) mode or Cache Mode（二级内存模式或“缓存”模式）	HBM 作为 DDR 内存侧缓存的模式。在该模式下，仅 DDR 地址空间对软件可见，而 HBM 则作为 DDR 的透明内存侧缓存。
	“fake” NUMA Node（fake-NUMA 节点）	使用 Linux 内核启动项 (numa=fake) 启用该功能，使系统物理内存分为多个“fake”（假的）NUMA 节点。换句话说，通过使用 fake-NUMA，一个物理 NUMA 节点（即一个统一的物理内存区域）可以以多个 NUMA 节点的形式对应用可见。

1.3 参考

表 2. 参考

描述	URL
英特尔® 架构指令集扩展编程参考 (Intel® Architecture Instruction Set Extensions Programming Reference)	https://software.intel.com/content/www/us/en/develop/download/intel-architecture-instruction-setextensions-programming-reference.html
memkind 库	http://memkind.github.io/memkind/
libnuma API	https://man7.org/linux/man-pages/man3/numa.3.html
hbwmalloc API	http://memkind.github.io/memkind/man_pages/hbwmalloc.html
英特尔® Memory Latency Checker（英特尔® MLC）	https://www.intel.cn/content/www/cn/zh/developer/articles/tool/intelr-memory-latency-checker.html

STREAM 基准测试	https://www.cs.virginia.edu/stream/
英特尔® oneAPI 数学核心函数库（Intel® oneAPI Math Kernel Library，英特尔® oneMKL）	https://www.intel.cn/content/www/cn/zh/developer/tools/oneapi/onemkl-download.html
面向 Linux 的英特尔® oneAPI 数学核心函数库开发人员指南*	http://www.intel.com/content/www/us/en/develop/documentation/onemkl-linux-developer-guide/top/intel-oneapi-math-kernel-library-benchmarks/intel-distribution-for-linpack-benchmark-1/overview-intel-distribution-for-linpack-benchmark.html

第 2 章
英特尔® 至强® CPU Max 系列

2.1 处理器示意图

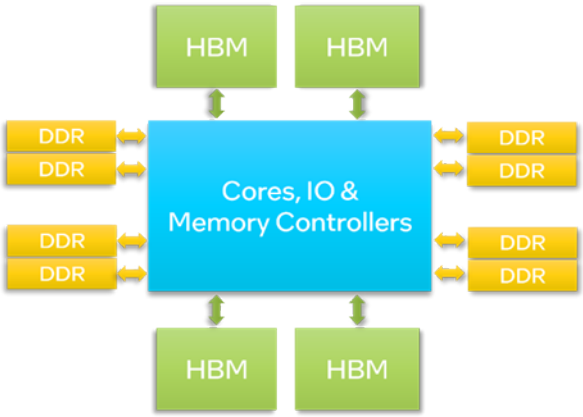

图 1. 处理器示意图

处理器包含 4 个 HBM2e 堆栈，每个处理器的高带宽内存（HBM）总容量为 64 GB，另外还有 8 通道 DDR 内存。

双路系统中，两个处理器通过最多 4 个英特尔® 超级通道互联（Intel® Ultra Path Interconnect，英特尔® UPI）链路连接。双路系统的 HBM 总容量为 128 GB。

2.1.1 HBM 堆栈与规格

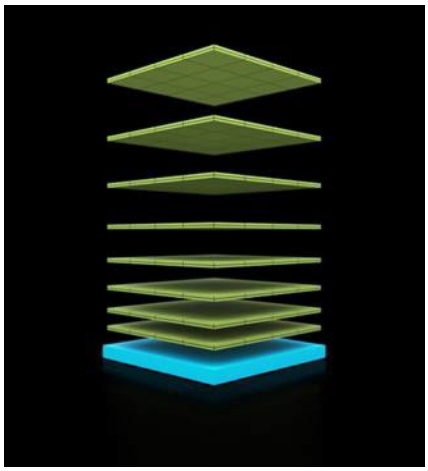

图 2. HBM 中的内存堆栈

HBM 内存由多个 DRAM 内存堆栈组成，通过宽总线连接。每个堆栈包含 8 个堆叠在底层逻辑芯片上的 DRAM。英特尔® 至强® CPU Max 系列处理器配备 4 个堆栈，HBM 总容量为 64 GB。

第 3 章
硬件配置

3.1 CPU 配置

英特尔® 至强® CPU Max 系列（封装或插槽）处理器中的 HBM 和 DDR 内存可以采用三种内存模式和两种集群模式进行配置。

本节将从硬件角度对每种模式进行介绍。有关每种模式如何与操作系统进行配置，以及应用如何使用这些模式分别见下文第 5 章和第 6 章内容。

3.1.1 内存模式

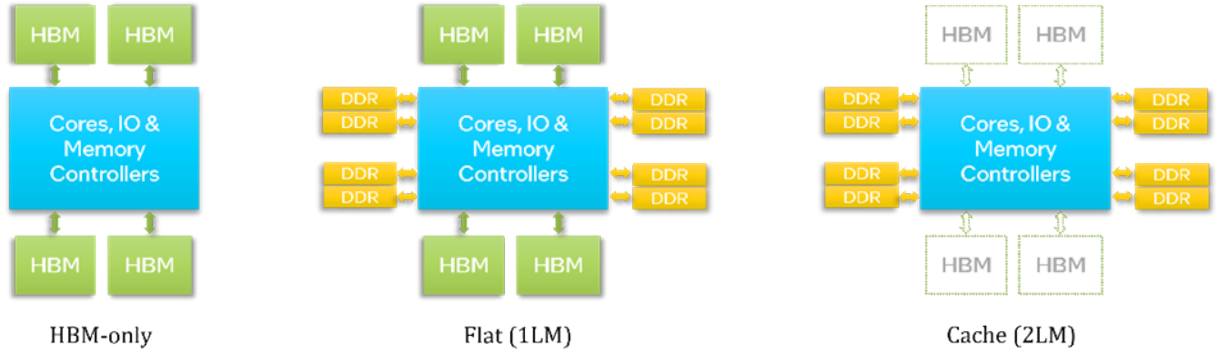

图 3. HBM 内存模式

处理器可使用三种内存模式使 HBM 对软件（操作系统和应用）可见。

3.1.1.1 “仅 HBM”内存模式

若未安装 DDR，可选择“仅 HBM”内存模式。在此模式下，HBM 是操作系统和应用唯一可用的内存。所有已安装的 HBM 都对操作系统可见，而应用则会看到操作系统暴露的部分。因此，操作系统和应用均可随时使用 HBM。但操作系统、后台服务和应用必须共享可用的 HBM 容量（每处理器 64 GB）。

3.1.1.2 “Flat”模式或一级内存 (1LM) 模式

若已安装 DDR 内存，可在系统启动时于 BIOS 菜单中选择“Flat”（也称 1LM）模式，使 HBM 和 DDR 对软件可见。在此模式下，HBM 和 DDR 各自作为独立的地址空间对软件可见，即 DDR 作为一个独立的地址空间（NUMA 节点）为软件所见，而 HBM 作为另一个地址空间（NUMA 节点）为软件所见。如下文第 6.2 节所述，用户需要使用 NUMA 感知工具（例如 numactl）或库，才能在该模式下使用 HBM。其他操作系统配置需在 HBM 作为常规内存池一部分被访问前完成（参见第 5.2 节）。

3.1.1.3 “缓存”模式或二级内存（2LM）模式

若已安装 DDR，可在系统启动时于 BIOS 菜单中选择“缓存”（也称 2LM）模式，使 HBM 作为 DDR 的内存侧缓存。在该模式下，仅 DDR 地址空间对软件可见，而 HBM 则作为 DDR 的透明内存侧缓存。因此，应用和命令行无需修改即可使用“缓存”模式。HBM 属于直接映射缓存，可能需要额外采取配置步骤才能尽量减少冲突未命中（conflict miss）这种情况（参见第 5.2.1 节）。

3.1.2 集群（分区）模式

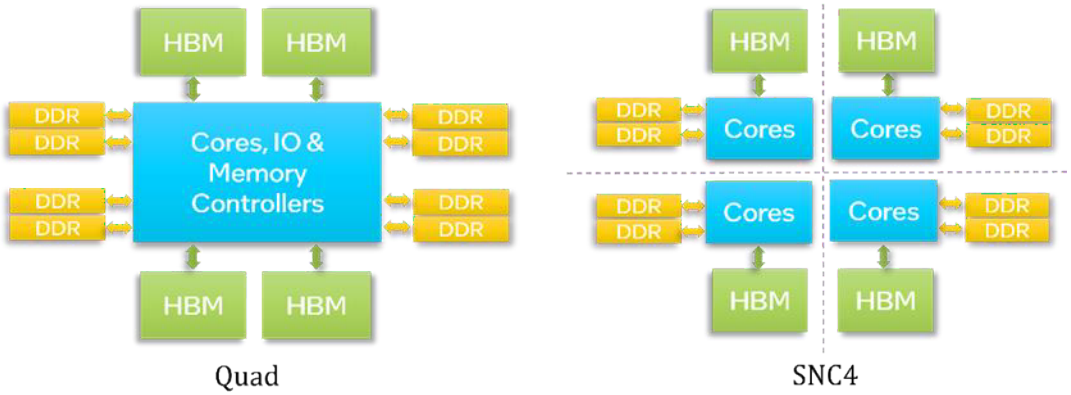

图 4. 集群模式

集群模式决定了如何对处理器进行分区，以形成不同的地址空间（NUMA 节点）。

通过集群（分区），内核可在同一个分区内实现更高带宽和更低时延。集群模式与内存模式为正交关系。英特尔® 至强® CPU Max 系列处理器具备两种集群模式。

3.1.2.1 “Quadrant”模式

“Quadrant”模式代表了对软件可见的单个地址空间（NUMA 节点）。在该模式下，应用不必采取额外操作步骤即可实现 NUMA 感知，因此更适用于在处理器所有内核间共享大型数据结构的应用（例如，运行在所有内核上并共享大型数据结构的 OpenMP 应用）。

3.1.2.2 “SNC4”（子 NUMA 集群-4）默认集群模式

该模式会对每个 CPU 进行分区，形成 4 个子 NUMA 集群分区。每个分区可作为 1 个或多个 NUMA 节点对软件可见。因此，每个处理器至少有 4 个 NUMA 节点。要使用该模式，应用须具备 NUMA 感知能力。相比“Quadrant”模式，这一模式可实现更高带宽和更低时延，更适合 NUMA 感知应用（例如 MPI 或 MPI + OpenMP 应用）。

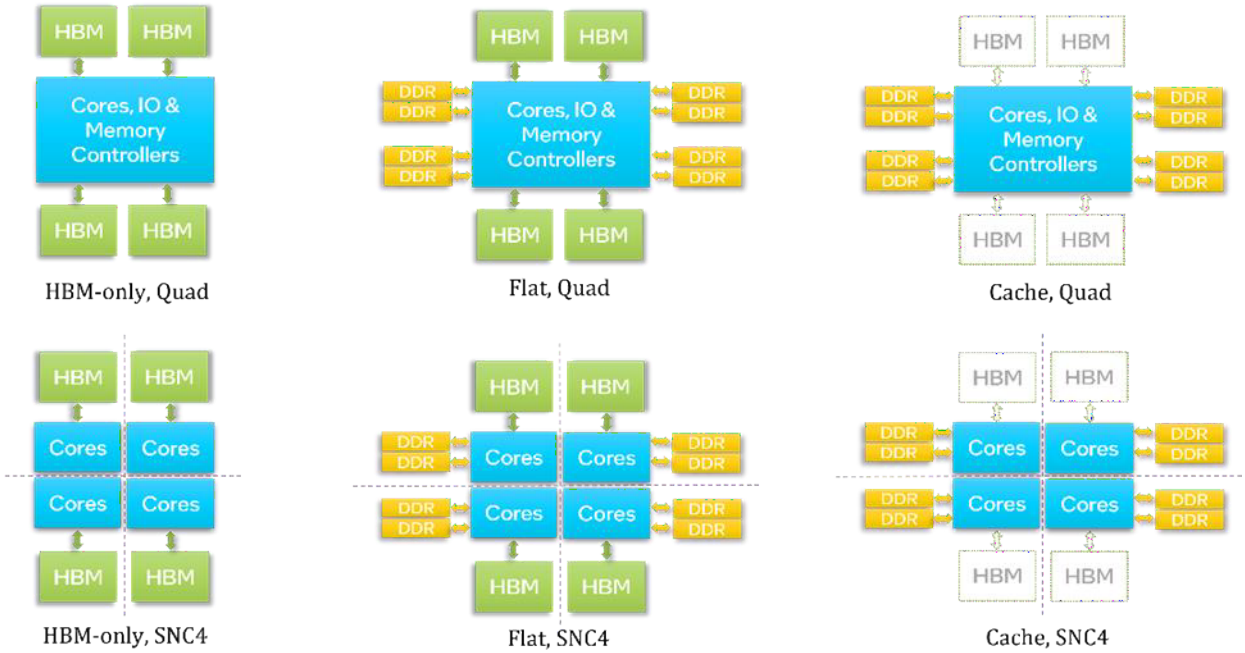

图 5. 内存模式配置

如图 5 总结所示，英特尔® 至强® CPU Max 系列处理器结合 3 种内存模式和 2 种集群模式，共有 6 种配置可供选择。

3.2 多路配置

英特尔® 至强® CPU Max 系列支持双路配置，最多由 4 个英特尔® UPI 链路连接。每路（处理器）为一个独立的地址空间（NUMA 节点）。因此，双路系统在“Quadrant”模式下至少有 2 个 NUMA 节点，在“SNC4”模式下至少有 8 个 NUMA 节点。

3.3 DIMM 配置

每个处理器配备 4 个 DDR 内存控制器，每个内存控制器支持 2 条 DDR 通道，每个处理器共支持 8 条通道。

3.3.1 “仅 HBM”模式

使用“仅 HBM”模式时，无需安装 DIMM。在部分 BIOS 调试版本中，可能会通过在 BIOS 选项中禁用 DIMM，而无需对其进行物理移除。

3.3.2 “Flat”模式

图 6 所示为单个 CPU 采用“Flat”模式时所有 DIMM 的配置选项，以及每个 DIMM 配置选项是否支持“SNC4”模式(使用 HBM、DDR 和 HBM + DDR)。

	iMC3				iMC2					iMC0				iMC1				SPR+HBM					
	Chan 1 (7/H)		Chan 0 (6/G)		Chan 1 (5/F)		Chan 0 (4/E)			Chan 0 (0/A)		Chan 1 (1/B)		Chan 0 (2/C)		Chan 1 (3/D)							
DDR5 + CPS	Slot0	Slot1	Slot0	Slot1	Slot0	Slot1	Slot0	Slot1		Slot1	Slot0	Slot1	Slot0	Slot1	Slot0	Slot1	Slot0	SNC2	SNC4 (DDR & HBM)	Hemi (DDR only)	Quad (HBM) * Exclusive with SNC4	Quad (DDR) * Exclusive with SNC4	All2All (DDR only)
0+0																			Y		Y		
1+0											DDR5										Y		Y
							DDR5														Y		Y
2+0			DDR5								DDR5										Y		Y
							DDR5								DDR5						Y		Y
4+0			DDR5				DDR5				DDR5				DDR5				Y		Y	Y	
8+0	DDR5		DDR5		DDR5		DDR5				DDR5		DDR5		DDR5		DDR5		Y		Y	Y	
16+0	DDR5	DDR5	DDR5	DDR5	DDR5	DDR5	DDR5	DDR5		DDR5	DDR5	DDR5	DDR5	DDR5	DDR5	DDR5	DDR5		Y		Y	Y	

CPU

图6. DIMM 配置选项

所有方案均支持“Quadrant”或“SNC4”模式使用 HBM。但只有具备对称 DIMM 配置的最后三种方案支持“SNC4”模式同时使用 HBM 和 DDR。在非对称 DIMM 配置方案中，DDR 空间采用名为“All-to-All”的特殊集群模式进行配置，该模式会产生更低的非对称带宽和更高的非对称时延。

与每通道仅填充一个插槽（倒数第二行）的方案相比，每通道填充 2 个 DDR 插槽（最后一行）的方案会导致更低带宽和更高时延。

3.3.3 “缓存”模式

“缓存”模式要求在 4 个内存控制器间采用对称 DIMM 配置。因此，上表中仅最后三行所列的方案支持“缓存”模式。

而为获得更出色的性能，必须使用 dual-rank DIMM。因为相比 single-rank DIMM，dual-rank DDR DIMM 提供的带宽更高。

3.4 BIOS 设置

本节介绍了在选择内存模式和集群模式时的 BIOS 选项。本节所示的菜单选项均针对的是英特尔® 软件开发平台（Intel® Software Development Platform，英特尔® SDP），您系统中的菜单选项**或有不同**，具体视 BIOS 提供商而定。

如您系统中的 BIOS 版本较旧，请升级至最新 BIOS 版本，因为最新版本可能包含新增功能和性能优化工具。

3.4.1 选择内存模式

使用以下菜单选项，在 BIOS 中选择内存模式：

EDKII Menu -> Socket Configuration -> Memory Configurations -> Memory Map -> Volatile Memory Mode -> 1LM/2LM（EDKII 菜单 -> 插槽配置 -> 内存配置 -> 内存映射 -> 易失性内存模式 -> 1LM/2LM）

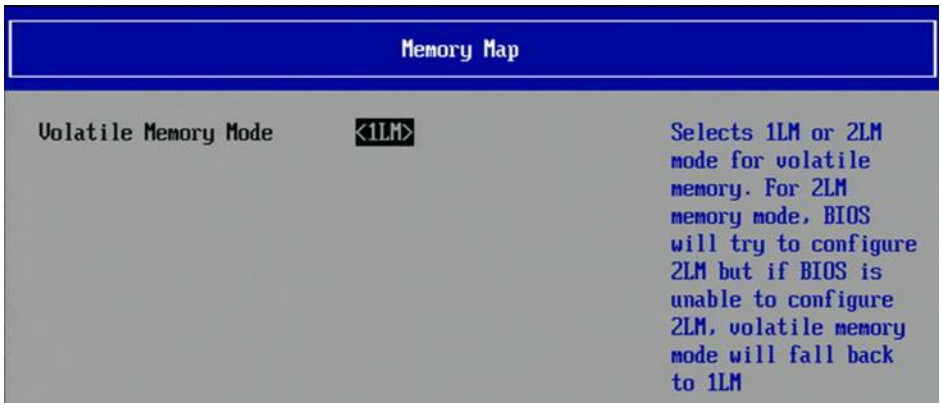

图7. 选择内存模式

3.4.2 选择集群模式

使用以下菜单选项，可在 BIOS 中选择集群模式：

EDKII → Socket configuration → Uncore configuration → Uncore General Configuration → SNC (Sub Numa)
[EDKII → 插槽配置 → Uncore 配置 → Uncore 常规配置 → SNC（子 NUMA）]

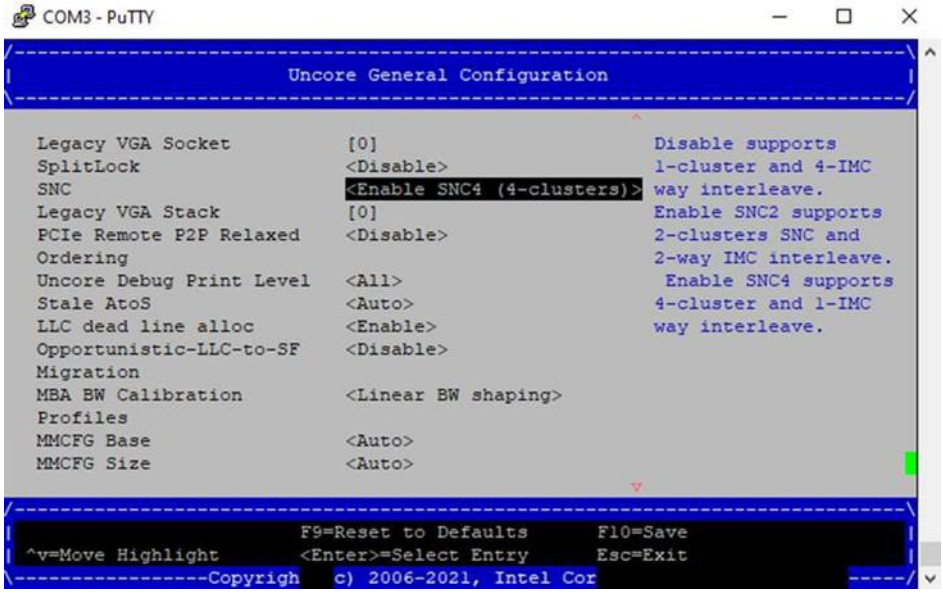

图8. 选择集群模式

注：可使用英特尔® 系统配置实用程序 (Intel® SYSCFG utility) 保存/恢复和检查英特尔® 服务器平台的 BIOS 配置。

第 4 章
Linux 系统配置

4.1 常见配置选项

以下配置选项可作为设置所有内存模式时的参考：

- 禁用内存交换。这项尤其适用于容量有限的“仅 HBM”模式。内存交换会严重影响性能。如运行应用导致内存交换，可以考虑释放内存（例如，清理文件系统缓存）或扩展至更多节点。

- 启用“zone_reclaim_mode”减少 NUMA 失中。该模式非常适合 NUMA 节点规模较小的情形（例如 “SNC4” 集群模式）。启用该模式时，Linux 页面分配器先在请求的 NUMA 节点上回收容易用的页面，然后再从其他 NUMA 节点上获得内存。这可减少不必要的 NUMA 交叉，从而避免性能下降。但回收活动可能会导致性能发生小幅波动。使用以下命令即可启用“zone_reclaim_mode”选项。由于该操作必须在每次重启后执行，因此建议使用初始化脚本进行自动设置。

```
echo 2 > /proc/sys/vm/zone_reclaim_mode
```

- 每次运行前，（如果此前运行过程中缓存的内容无使用价值）建议清理文件系统缓存，并使用以下命令规整内存。由于这些命令需要 root 权限（即根权限），系统管理员应考虑将其纳入批处理系统作业的前期处理工作（Job Prologue），或将其以具有 setuid 权限的二进制文件形式提供。

```
sync; echo 3 > /proc/sys/vm/drop_caches;
echo 1 > /proc/sys/vm/compact_memory
```

- 建议启用透明大页（THP）。大多数科学计算应用都会从使用 THP 获益。虽然创建 THP 等大页时可能会因需要进行内存规整产生开销，但管理员可按照上文介绍的方法在每次运行前规整内存，以此减少这种开销。

- 避免使用 /dev/shm（tmpfs）来存储文件，因为这样做会减少可用的内存。建议系统管理员将清除 /dev/shm 纳入作业的前期处理工作，以减少作业之间相互影响。

- 建议使用最新且稳定的 Linux 内核（当前为 5.15）。

4.2 Linux 实用工具

4.2.1 numactl

```
$ numactl -H
available: 8 nodes (0-7)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 112 113 114 115 116 117 118 119 120 121 122 123 124 125
node 0 size: 128734 MB
node 0 free: 128333 MB
node 1 cpus: 14 15 16 17 18 19 20 21 22 23 24 25 26 27 126 127 128 129 130 131 132 133 134 135 136 137 138 139
node 1 size: 129017 MB
node 1 free: 128847 MB
node 2 cpus: 28 29 30 31 32 33 34 35 36 37 38 39 40 41 140 141 142 143 144 145 146 147 148 149 150 151 152 153
node 2 size: 129017 MB
node 2 free: 128834 MB
node 3 cpus: 42 43 44 45 46 47 48 49 50 51 52 53 54 55 154 155 156 157 158 159 160 161 162 163 164 165 166 167
node 3 size: 129017 MB
node 3 free: 128848 MB
node 4 cpus: 56 57 58 59 60 61 62 63 64 65 66 67 68 69 168 169 170 171 172 173 174 175 176 177 178 179 180 181
node 4 size: 128976 MB
node 4 free: 128807 MB
node 5 cpus: 70 71 72 73 74 75 76 77 78 79 80 81 82 83 182 183 184 185 186 187 188 189 190 191 192 193 194 195
node 5 size: 129017 MB
node 5 free: 127974 MB
node 6 cpus: 84 85 86 87 88 89 90 91 92 93 94 95 96 97 196 197 198 199 200 201 202 203 204 205 206 207 208 209
node 6 size: 129017 MB
node 6 free: 128801 MB
node 7 cpus: 98 99 100 101 102 103 104 105 106 107 108 109 110 111 210 211 212 213 214 215 216 217 218 219 220 221 222 223
node 7 size: 129005 MB
node 7 free: 128834 MB
node distances:
node   0   1   2   3   4   5   6   7
  0:  10  12  12  12  21  21  21  21
  1:  12  10  12  12  21  21  21  21
  2:  12  12  10  12  21  21  21  21
  3:  12  12  12  10  21  21  21  21
  4:  21  21  21  21  10  12  12  12
  5:  21  21  21  21  12  10  12  12
  6:  21  21  21  21  12  12  10  12
  7:  21  21  21  21  12  12  12  10
```

图9. numactl -H 示例

Linux 工具 numactl 常用于查看系统的 NUMA 配置以及在特定 NUMA 节点上执行应用。要查看系统的 NUMA 配置情况，可使用 numactl -H。下图所示为 numactl -H 的输出结果，分别显示了 NUMA 节点数，CPU 内核数和每个 NUMA 节点的内存容量，随后的矩阵描述了每个节点与其他节点之间的距离。更多信息请见 man numactl。

4.2.2 numastat

Linux 工具 numastat 提供有关 NUMA 内存使用情况的多种数据。以下命令尤其有用（更多详情请见 man numastat）。

```
$ numastat -p python

Per-node process memory usage (in MBs)
PID                         Node 0          Node 1          Node 2
----------------  --------------- --------------- ---------------
5132 (tuned)                10.54            1.21            1.21
68100 (python3)             11.88            0.00            0.00
----------------  --------------- --------------- ---------------
Total                       22.42            1.21            1.21
```

图10. numastat -p 示例

- 如图 11 所示，numastat -p <process_name 可提供特定进程的内存使用情况。
- numastat -m 可显示整个系统的内存使用情况。

```
$ numastat
                           node0           node1
numa_hit                7038233469      7552949520
numa_miss                 31491495               0
numa_foreign                     0        31491495
interleave_hit           254896206       254893381
local_node              6905622525      7430407252
other_node               164102439       122542268
```

图11. numastat -m 示例

- numastat（不含参数）显示 NUMA 的命中和失中情况（自启动之初开始积累）。这对于识别可能导致性能意外下降的 NUMA 节点交叉（numa_miss）很有用。由于这些数据从启动之初就开始积累，因此应在程序运行前后分别运行一次 numastat 来确定当次程序运行是否发生 NUMA 失中情况。

4.2.3 turbostat

turbostat 可在作为 root（或作为具有 setuid 权限的二进制文件）执行时用于检查 x86 架构处理器的功耗、频率和温度。turbostat 对于识别系统散热问题很有帮助。例如，下图就显示了系统功耗、频率和温度：

```
• turbostat -qS                    # average for both CPUs
• turbostat -qS --cpu package      # separately for each CPU
```

```
$ turbostat -qS
Avg_MHz Busy%   Bzy_MHz TSC_MHz IPC     IRQ     SMI     POLL    C1ACPI  C2ACPI  POLL%   C1ACPI% C2ACPI% CPU%c1  CPU%c6  CoreTmp PkgTmp  PkgWatt RAMWatt PKG_%   RAM_%
1337    49.92   2679    1800    1.68    599693  0       5       669     8110    0.00    0.05    49.92   50.08   0.00    75      75      697.92  172.62  190.03  0.00
1341    49.93   2676    1807    1.70    601152  0       7       1038    8791    0.00    0.07    50.07   50.07   0.00    73      73      702.72  174.93  191.08  0.00
```

图12. turbostat 示例

4.2.4 lscpu

这一标准 Linux 工具显示了系统高级配置详情，包括 NUMA 节点、内核数、基频、缓存大小和 CPU 标志（功能）。

4.2.5 dmidecode 与 lshw

这两项 Linux 工具可（在具备 root 权限时）用于检测已安装的硬件组件，包括 HBM 和 DDR 模块。

4.2.6 htop

htop是一种类似于 Linux top的工具，但它会以可视化格式显示单个 CPU 内核/线程的使用情况，有助于识别 NUMA 使用情况和 MPI 等级，或 OpenMP 线程布局。此外，htop 还可显示系统的内存使用情况。

4.2.7 lstopo

lstopo 工具隶属于 hwloc 库，后者能够通过标准包管理器（如 dnf install hwloc）以包的形式进行安装。lstopo 工具（或 lstopo-no-graphics）可显示系统的硬件拓扑结构。

第 5 章
内存模式特定配置

本章介绍了每种内存模式的操作系统配置方案。

5.1 “仅 HBM”内存模式

在“仅 HBM”模式下使用 HBM，无需额外的配置操作。但由于 HBM 容量有限，管理员可采取一些额外操作来减少内存容量开销。以下为操作建议：

- 考虑减少系统启动时即开始启用的非必要服务（守护进程）和驱动程序[例如，虚拟网络计算（VNC）服务器，打印或邮件服务等守护进程，以及性能分析驱动程序]
- 考虑缩减操作系统文件缓存容量和 MPI 缓冲区
- 每次运行前，可以考虑清理文件系统缓存并规整内存（参见上文第 4.1 节）

5.1.1 NUMA 节点枚举

图 13 所示为“仅 HBM”模式下的双路系统在设置“Quadrant”和“SNC4”模式时的 NUMA 节点配置。使用“Quadrant”模式会产生 2 个 NUMA 节点（每路 1 个节点）；使用“SNC4”模式会产生 8 个 NUMA 节点（每路 4 个节点）。每个 NUMA 节点均包含内核与 HBM 内存。

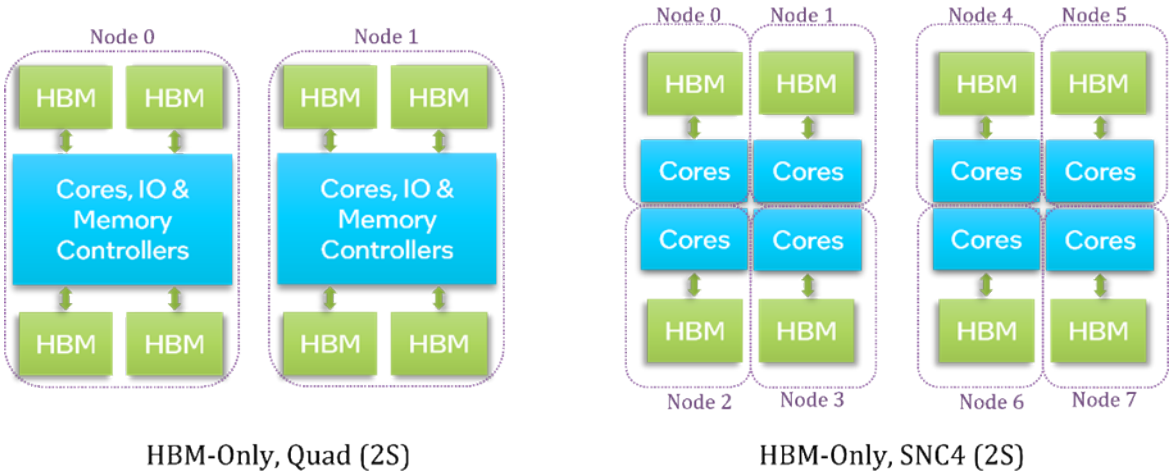

图 13. “仅 HBM”模式下的 NUMA 节点配置

使用 numactl -H 验证 NUMA 节点配置和每个 NUMA 节点的总内存或可用内存容量。

5.2 “Flat”内存模式

在完成 DDR 安装后，可如第 3.4.1 节所述，在 BIOS 设置中选择 1LM 以启用“Flat”模式。但是仅仅这样操作并不能使 HBM 通过默认内存池对操作系统和应用可见，必须采取额外的操作步骤才能使 HBM 进入默认内存池。

在 BIOS 菜单中选择 1LM 后，系统启动时仅有 DDR 对操作系统和应用可见。此时，HBM 因为被标记为专用内存，所以仍然无法在默认内存池中可见。这样的设计选择是为了防止操作系统在启动过程中分配和预留宝贵的 HBM 内存资源。由于 HBM 在启动过程中处于“隐藏”状态，操作系统无法对 HBM 内存资源进行分配和预留。

以下是在“Flat”模式下启动并使 HBM 可见的操作步骤：

1. 在 BIOS 菜单中选择 1LM（参见第 3.4.1 节）并启动操作系统。系统启动后，默认内存池中仅有 DDR 可见。可使用‘numactl -H’观察到实际情况。

2. 安装以下 Linux 包：

```
dnf install daxctl ndctl
```

3. 针对双路系统执行下列 daxctl 命令（“Quadrant”模式仅需前两项命令，“SNC4”模式要求完成所有命令）。所有这些命令均需 root 权限。

```
## Base commands for both Quadrant andSNC4 cluster modes
##
daxctl reconfigure-device -m system-ram dax0.0
daxctl reconfigure-device -m system-ram dax1.0

## For SNC4 cluster mode, use the following additional
commands: ##
daxctl reconfigure-device -m system-ram dax2.0
daxctl reconfigure-device -m system-ram dax3.0
daxctl reconfigure-device -m system-ram dax4.0
daxctl reconfigure-device -m system-ram dax5.0
daxctl reconfigure-device -m system-ram dax6.0
daxctl reconfigure-device -m system-ram dax7.0
```

每次启动系统时，都需要执行第 3 步。因此，将上述命令写入脚本，在操作系统启动时自动执行，会更为便利。

使用‘numactl -H’验证 HBM 节点是否可见，以及整个 HBM 容量是否可用。

5.2.1 “Flat”模式 NUMA 节点枚举

图 14 所示为“Flat”模式下的双路系统在设置“Quadrant”和“SNC4”模式时的 NUMA 节点配置情况。

- 使用“Quadrant”模式会产生 4 个 NUMA 节点（2 个连接内核的 DDR 节点和 2 个未连接内核的 HBM 节点）。
- 使用“SNC4”模式会产生 16 个 NUMA 节点（8 个连接内核的 DDR 节点和 8 个未连接内核的 HBM 节点）。

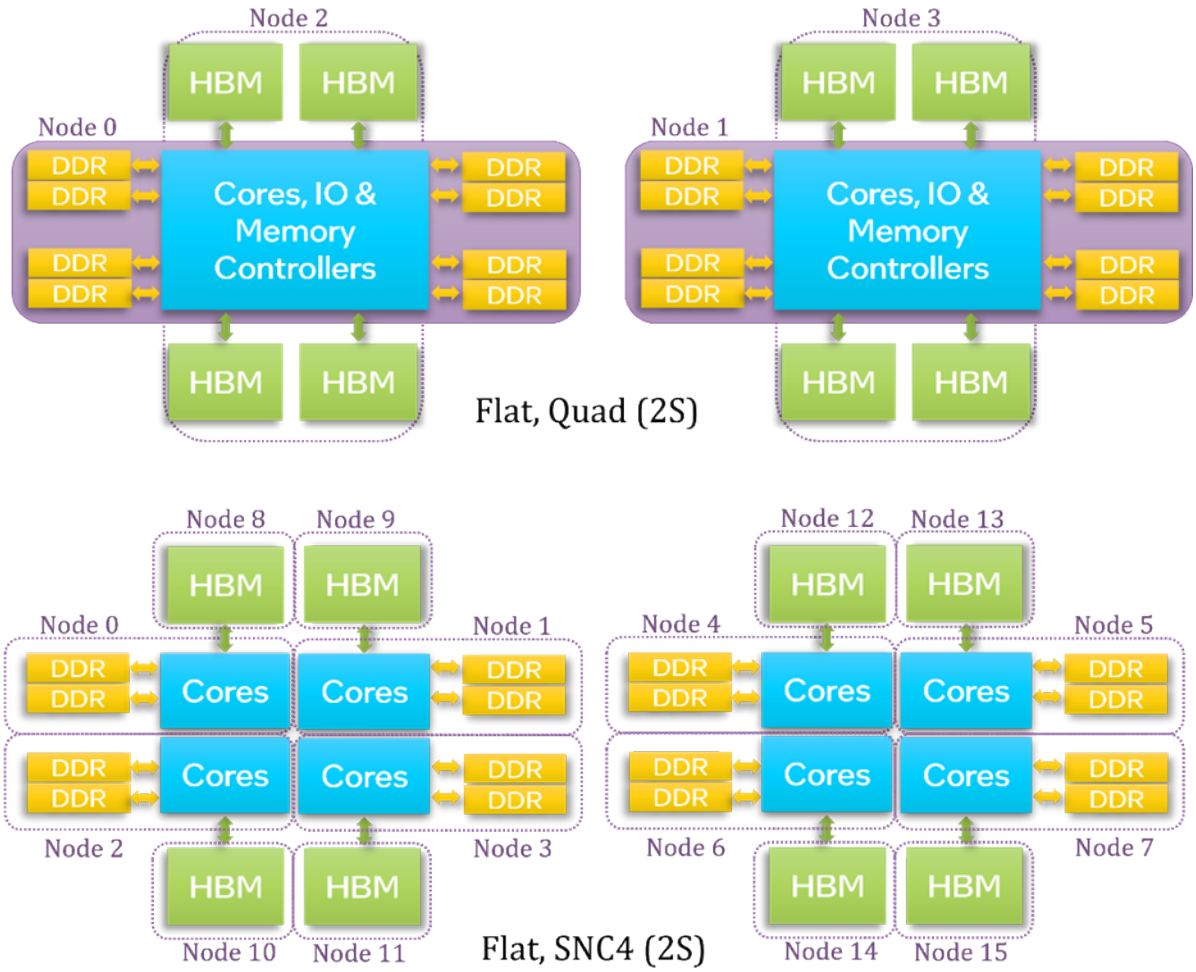

图 14. “Flat”模式下的 NUMA 节点配置

使用 numactl -H 验证 NUMA 节点配置和每个 NUMA 节点的总内存或可用内存容量。

5.3 “缓存”内存模式

使用“缓存”模式不需要额外进行配置。但由于 HBM 缓存是直接映射的内存侧缓存，因此强烈建议使用 fake-NUMA 另行配置操作系统，以缓解冲突未命中对应用的影响。

5.3.1 在“缓存”内存模式下使用 fake-NUMA

使用 Linux 内核启动项 (numa=fake) 启用该功能，使系统物理内存分为多个“fake”（假的）NUMA 节点。换句话说，通过使用 fake-NUMA，一个物理 NUMA 节点（即一个统一的物理内存区域）可以以多个 NUMA 节点的形式对应用可见。

以 128 GB DDR 内存配置“缓存”模式下的 64 GB HBM 为例。下图（左侧）显示了 128 GB DDR 地址空间中两个缓存行如何映射到 HBM 的相同位置，导致在 64 GB HBM 缓存中产生冲突。换句话说，因为 HBM 缓存是直接映射的，所以两个缓存行中只有一行可以存在于缓存中。

图 15（右侧）显示了创建两个 fake-NUMA 节点的效果。如果应用可以适配 fake-NUMA 节点（即节点 0），则可以保证不会在 HBM 缓存中遇到任何冲突未命中的情况。

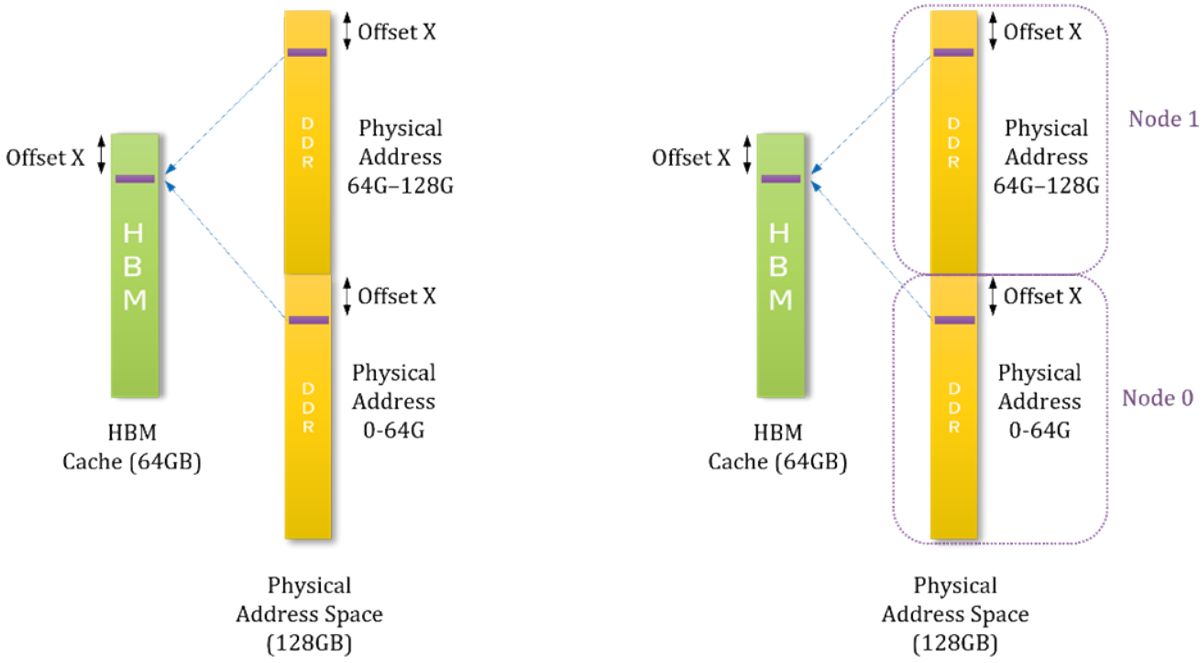

图15. fake-NUMA 节点示例

要在配备 128 GB DDR 的系统上创建两个 fake-NUMA 节点（每节点容量为 64 GB），可以使用内核启动项 numa=fake=2U。这样，每个物理 NUMA 节点就创建了两个 fake-NUMA 节点。

如果没有 fake-NUMA，即使应用所占内存小于 HBM 的容量，也可能会因为物理内存碎片化导致 HBM 缓存中产生冲突未命中的情况。启用 fake-NUMA 后，Linux 内核会按顺序填充 fake-NUMA 节点。也就是说，内存首先在 NUMA 节点 0 上分配，然后在 fake-NUMA 节点 1 上分配，依此类推。这样做可确保适配 fake-NUMA 节点的应用在 HBM 缓存中进行分配时不会产生冲突，因此使这类应用具备更高的性能和更低的波动性。

在选择 fake-NUMA 启动项启动内核后，使用‘numactl -H’验证节点分割是否合适。如 fake-NUMA 节点不可见，务必使用内核配置选项‘CONFIG_NUMA_EMU=y’来构建内核。

在“Quadrant”集群模式下，一个 fake-NUMA 节点的大小应约为 64 GB；在“SNC4”模式下，则应在 16 GB 左右。

属于同一物理 NUMA 节点的所有 fake-NUMA 节点共享相同的 CPU 内核。因此，fake-NUMA 不会影响应用启动命令，不过 NUMA 节点的数量会按 DDR 总容量与 HBM 总容量的比率增加。

强烈建议使用 fake-NUMA 时**禁用内存交换**，因为 fake-NUMA 可能会在一个 fake-NUMA 节点填满时进行内存交换操作。

由于 fake-NUMA 引入了容量更小的节点，因此当一个 fake-NUMA 节点填满时，通过 fake-NUMA 启用 zone_reclaim 可能会产生更频繁的回收活动，从而导致性能小幅波动。

所有标准 NUMA 工具都可在 fake-NUMA 节点上使用。例如，‘numactl -m 2 ./a.out’可启动使用 fake-NUMA 节点 2 的内存的应用。与此类似，numastat 会显示 fake-NUMA 节点的属性。

5.3.2 页面随机分配（页面随机化）

Linux 提供了一种随机分配页面的功能。不使用 fake-NUMA 时，页面随机分配功能可能有助于（例如，在新启动的系统和已经运行很长一段时间的系统之间）获得更一致的性能结果。在页面分配随机进行时，页面会被随机分配到物理内存中的页面地址，每次应用启动时，HBM 缓存中相互冲突的页面都会改变。

此功能可在 Linux 内核 v5.4 或更新的版本中通过使用内核启动项 page_alloc.shuffle=y 启用。可使用文件 /sys/module/page_alloc/parameters/shuffle 来检查此功能是否存在。

5.3.3 “缓存”内存模式 NUMA 节点枚举

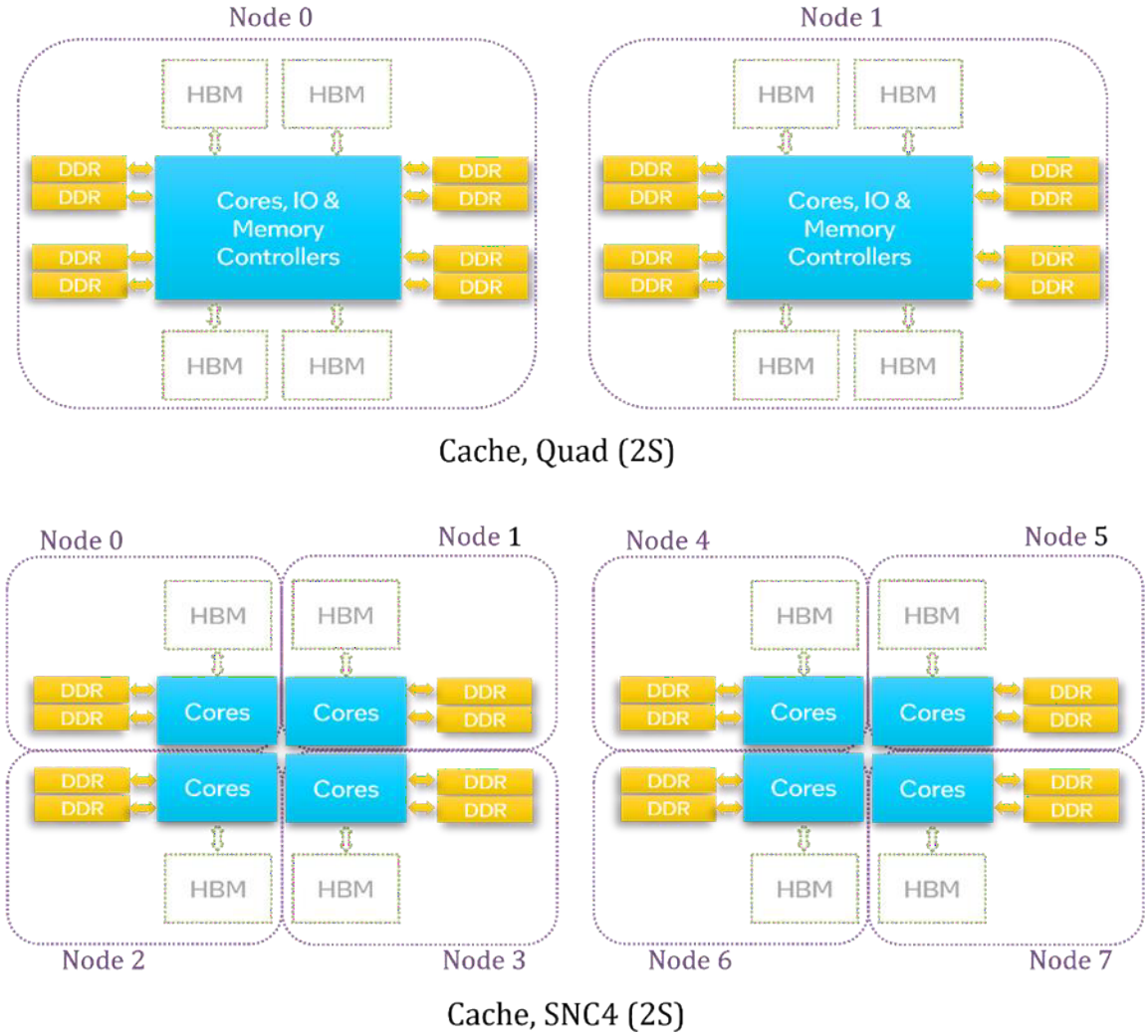

图16. “缓存”模式下的NUMA 节点配置

图 16 所示为“缓存”模式下的双路系统在设置“Quadrant”和“SNC4”模式时的 NUMA 节点配置。“Quadrant”模式会产生至少 2 个 NUMA 节点（每路 1 个节点），“SNC4”模式会产生至少 8 个 NUMA 节点（每路 4 个节点）。每个 NUMA 节点均包含内核与内存。

使用 numactl -H 验证 NUMA 节点配置和每个 NUMA 节点的总内存或可用内存容量。

第 6 章
使用内存模式和集群模式

本章介绍终端用户如何使用内存模式和集群模式。

6.1 “仅 HBM”内存模式

使用“仅 HBM”模式无需更改源代码或命令行语法。操作系统和应用两者都使用 HBM 内存，这是唯一可用的选项。但是，应用若要适配可用的内存，可能需要另外采取下文提供的一些步骤。这些步骤是第 5.1 节提供的操作系统配置步骤的补充。

- 在 OpenMP 线程数与 MPI 等级之间取得平衡。由于 OpenMP 线程共享内存，因此多使用 OpenMP 线程可以减少内存总占用空间
- 如果应用无法适配 HBM 容量，需要适当调整 OpenMP 堆栈大小和 MPI 通信缓冲区大小
- 每次运行前，按照第 4.1 节所述释放文件系统缓存并规整内存
- 避免使用 /dev/shm（tmpfs）来存储文件，因为这样做会减少可用的内存。如果先前的作业已经产生了文件，需要清除 /dev/shm 中的文件
- 确保没有 NUMA 失中（通过在运行程序前后运行 numastat 来实现）
- 如在执行上述步骤后应用仍无法适配 HBM 容量，可以考虑扩展到更多节点

6.2 “Flat”内存模式

如上所述，“Flat”模式能使 DDR 和 HBM 作为独立的地址空间（NUMA 节点）对用户可见。图 17 所示为“SNC4”集群模式下采用“Flat”内存模式的系统的‘numactl -H’输出结果。

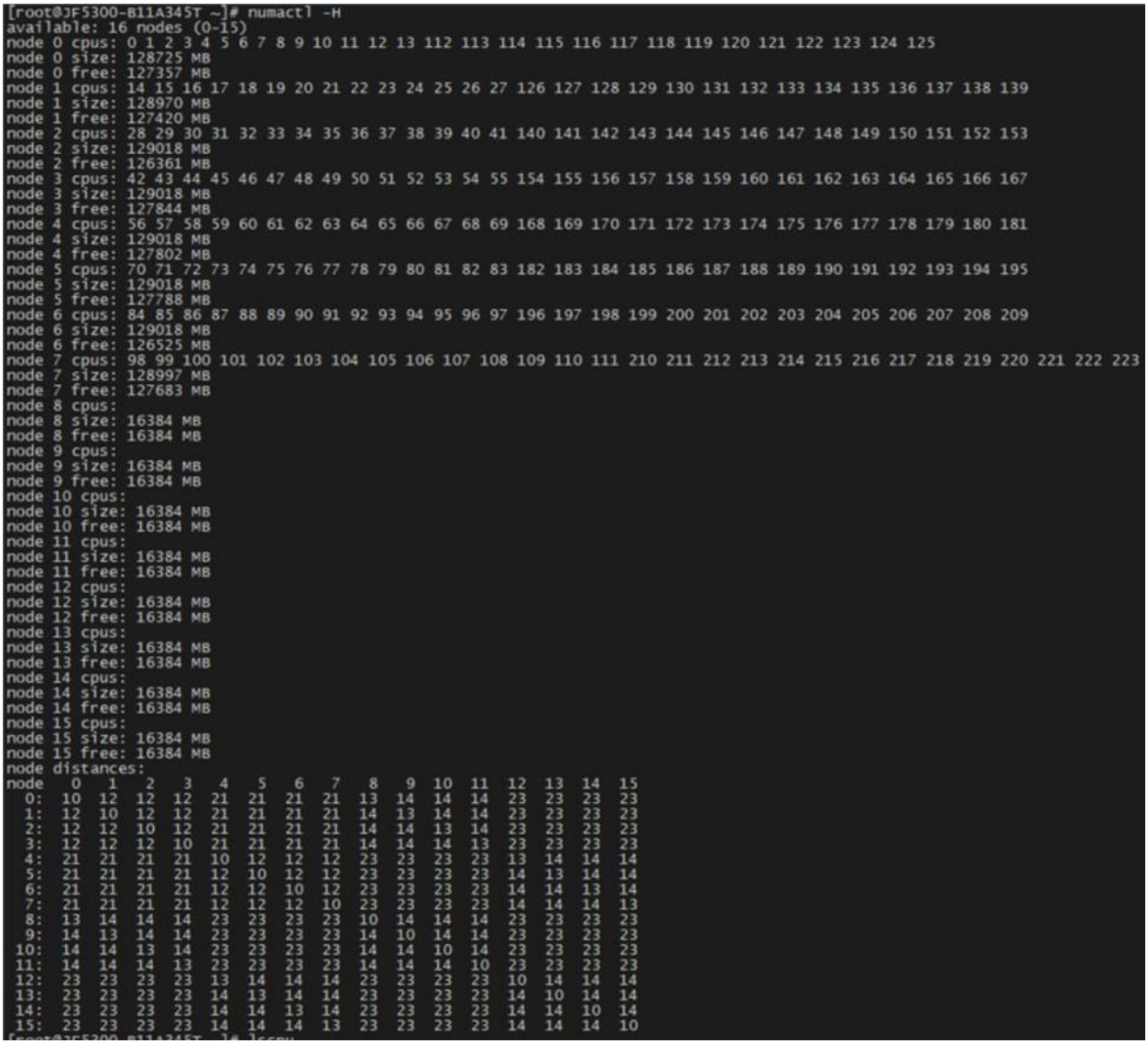

图17. numactl -H 示例

如 'numactl -H' 输出结果所示，系统每路有以下两种节点：

- 连接 CPU 的 DDR 节点（节点 0-7）
- 未连接任何 CPU 的 HBM 节点（节点 8-15）

应用在 CPU 内核上启动时，内存分配会先从离它们最近的 NUMA 节点开始，即"节点距离"决定内存分配。例如，在上图中，对于节点 0 中的 CPU 而言，最近的内存（距离为 10）在节点 0，是连接到节点 0 的 DDR。要使用不同类型的内存，用户需要使用 libnuma 的功能。libnuma 功能可通过以下四种方式获得：

- numactl 工具
- 英特尔® MPI 库
- OpenMP 库

- libnuma API
- memkind 库

使用前两种方法可将整个应用（程序代码、静态数据、堆、栈）放置在 HBM 中，使用后两种方法则可将动态分配（堆分配）的数据结构放置在 HBM 中。

6.2.1 使用 numactl 进行整个应用的 HBM 布局

标准的 Linux 工具 numactl 可用于将应用所需的内存放入 NUMA 节点。要进行这样的操作有多个策略可供考虑：

- membind (numactl --membind hbm_node1, hbm_node2, .../a.out)： 强制从指定节点分配应用所需的所有内存。如果应用超出指定节点的容量，应用将终止。因此，用户必须保证应用不会超出可用的 HBM 容量（该容量可能小于最大 HBM 容量）。

- preferred (numactl --preferred hbm_node ./a.out)： 使应用首先从指定的首选节点开始分配内存，直到节点填满。一个节点填满后，后续分配将转到默认节点，且默认节点始终为“Flat”内存模式下的 DDR 节点。请注意，只能指定一个首选节点。由于 Linux 使用 first-touch 策略，因此应用需要分配和访问（例如，初始化）页面，以便将它们放置到首选节点中。举例来说，在“Quadrant”集群模式下的双路系统上，用户可以使用 MPI 冒号语法在 mpiexec 命令中针对不同的 MPI 等级指定不同的 HBM 节点：

```
mpiexec -n 1 numactl ---preferred hbm_node1 ./a.out : -n 1 numactl --
preferred hbm_node2 ./a.out
```

- preferred-many (numactl --preferred-many hbm_node1, hbm_node2,...../a.out)： 类似于 --preferred 选项，但允许有多个首选节点。因此，这一选项对于 SNC4 和多路系统特别有用。但是，运用这个策略需要 Linux 内核 5.15 以上和 numactl 2.0.15 以上版本。

- interleaved (numactl --interleave hbm_node,DDR_node)： 允许在任意两个 NUMA 节点之间交叉存取内存，在内存占用空间大约是 HBM 容量的两倍时特别有用。如果我们在 DDR 和 HBM 之间进行交叉存取，可以预期的最大带宽是 DDR 带宽的两倍。

最好的方法通常是在应用适配 HBM 容量的情况下使用 membind 策略将整个应用放置在 HBM 中。如果应用不适配 HBM 容量，则应考虑扩展到更多节点，之后再使用 preferred 或 interleaved 策略。

6.2.1.1 使用 SNC4 时需特别注意的事项

在“SNC4”模式下使用 numactl 支持多 NUMA 节点配置时，有几点需要特别注意，具体取决于使用的是 membind 还是 preferred 策略：

- membind： 在“SNC4”模式下运行 MPI 应用时，用户可以将所有 HBM 节点指定为 numactl 的参数（例如，mpiexec -np 8 numactl -m 4-7 ./a.out），而 HBM 内存将根据每个等级（进程）从最近的节点开始分配。

- preferred 或 interleaved：在“Flat”模式下使用 SNC4 时，如果我们想运用上面讨论的 preferred 或 interleaved 策略将应用的一部分放到 HBM 中，就必须使用 MPI 的冒号语法，因为 preferred 策略只接受 1 个 NUMA 节点（除非 -- preferred-many 可用），而 interleaved 策略须借助相应的 HBM 和 DDR 节点来完成。例如，要在采用“SNC4”内存模式的单路系统上运用 preferred 策略，在运行 56 个 MPI 等级的情况下，我们可以使用以下命令：

```
      mpirun -n 14 numactl -N 0 -p 4 ./a.out : -n 14 numactl -N 1 -p 5 ./a.out : -n 14
numactl -N 2 -p 6 ./a.out : -n 14 numactl -N 3 -p 7 ./a.out
```

与此类似，要在同一系统上运用 interleaved 策略，我们可以使用以下命令：

```
      mpirun -n 14 numactl -N 0 -i 0,4 ./a.out : -n 14 numactl -N 1 -i 1,5 ./a.out :
-n 14 numactl -N 2 -i 2,6 ./a.out : -n 14 numactl -N 3 -i 3,7 ./a.out
```

6.2.2 使用英特尔® MPI 进行整个应用的 HBM 布局

对于 MPI 应用来说，环境变量 I_MPI_HBW_POLICY（而不是 numactl）可以针对 MPI 等级分配 HBM。 有关此环境变量的更多信息，请见 I_MPI_HBW_POLICY 的参考页。

```
      mpirun -genv I_MPI_HBW_POLICY hbw_bind -n 2 ./a.out
      mpirun -genv I_MPI_HBW_POLICY hbw_preferred -n 2 ./a.out
      mpirun -genv I_MPI_HBW_POLICY hbw_interleave -n 2 ./a.out
```

I_MPI_HBW_POLICY 环境变量也接受由 MPI 本身分配内存（例如 MPI 缓冲区）的分配策略。例如，以下命令在进行用户和 MPI 库分配时都使用了 hbw_bind 策略。

```
      mpirun -genv I_MPI_HBW_POLICY hbw_bind,hbw_bind -n 2 ./a.out
```

6.2.3 （“Flat”模式下）HBM 中数据结构的布局

要实现更精细的控制，可以采用以下方法将动态分配的数据结构放置在 HBM 中。只有当用户需要细粒度控制并且无法使用 numactl 或 MPI 环境变量将整个应用放入 HBM 时（例如，当应用超过 HBM 总容量时），才可使用这些方法。

使用这些方法需要修改源代码。只有动态分配的数据结构（即在堆上分配的数据结构）才可以放置在 HBM 中。栈、静态数据和代码不能使用这些方法放于 HBM 中。

6.2.3.1 使用 OpenMP 进行 HBM 布局

OpenMP 在英特尔各个经典编译器（任何最近的版本）和 2021.3 及以上版本的英特尔® oneAPI 编译器中均有提供。编译器的 OpenMP 编译指示和指令依托的是作为 libnuma 接口的 memkind 库。

GCC 11 或更高版本中也提供了此 OpenMP 功能。

C/C++

```
      #include <omp.h>

      float *x = (float *)omp_aligned_alloc(64, N*sizeof(float), omp_high_bw_mem_alloc);

      omp_free(x, omp_high_bw_mem_alloc);
```

要获得“membind”行为，需要将回退操作设置为 null_fb or abort_fb

```
      omp_alloctrait_t traits[2] = { {omp_atk_alignment, 64}, {omp_atk_fallback,
omp_atv_null_fb} };

      omp_allocator_handle_t my_high_bw_mem_alloc = omp_init_allocator(omp_high_bw_mem_space,
2, traits);

      float *x = (float *)omp_alloc(N*sizeof(float), my_high_bw_mem_alloc);

      omp_free(x, my_high_bw_mem_alloc);

      omp_destroy_allocator(my_high_bw_mem_alloc);
```

FORTRAN

```
real, allocatable ::x(:)
!dir$ omp allocate(x) allocator(omp_high_bw_mem_alloc) align(64)
allocate(x(N))
```

6.2.3.2 使用 memkind 库中的 hbwmalloc 进行 HBM 布局

您可以使用 memkind 库提供的 hbwmalloc API 在 HBM 中分配数据结构。

与 -lmemkind 连接

```
#include <hbwmalloc.h>
float* x = (float *)hbwmalloc(N * sizeof(float));
hbw_free(x);
```

还有 hbw_posix_mem_align，

```
#include <hbwmalloc.h>
float* x; hbw_posix_memalign((void**) &x, 64, N * sizeof(float));
hbw_free(x);
```

以及分配器，

```
#include <hbw_allocator.h>
std::vector<float, hbw::allocator<float>> x;
```

两个 FORTRAN 示例：

```
!dir$ attributes memkind:hbw :: x

real, allocatable :: x(:)

allocate(x(N))


real, allocatable :: x(:)

!dir$ memkind : hbw, align:64

allocate(x(N))
```

6.3 “缓存”内存模式

使用“缓存”模式无需更改源代码或命令行语法。由于 HBM 缓存对软件透明，因此应用只能看到 DDR 内存空间。这样一来，用户就可以像使用仅配备 DDR 的设备一样运行他们的应用。

不过，要获得更好的性能，用户需要知悉：HBM 是充当 DDR 的缓存，并且此 HBM 缓存是直接映射的。因此，应用在 HBM 缓存中可能会遇到冲突未命中情况。当两个 DDR 地址映射到 HBM 缓存中同一位置（集）时就会发生冲突未命中情况。由于 HBM 是直接映射的内存侧缓存，因此在给定时间只能缓存其中一个地址。这会导致频繁出现缓存失中情况（即在缓存中找不到所需的缓存行）。缓存失中会增加内存时延并降低有效带宽。

由于 HBM 容量为每路 64 GB，因此理想情况下，可以适配此容量的应用应该不会导致任何冲突未命中。但实际上，由于物理内存碎片化的缘故，情况并非如此。操作系统可以从物理地址范围内的任何位置分配物理内存。也就是说，物理内存并不总是从地址 0 开始连续分配。频繁的分配和释放操作会导致已分配的内存不连续。这种情况被称为“内存碎片化”。当物理内存呈现碎片化时，即使应用占用的内存总量小于 HBM 的容量大小，应用也有可能会出现内存地址在 HBM 直接映射缓存中相互冲突的情况（即地址映射到缓存中相同的位置）。

如第 5.3.1 节所述，对于内存占用量小于 64 GB 的应用，我们可以使用名为 fake-NUMA 的 Linux 内核功能来避免因物理内存碎片化而产生这些不必要的冲突。通过使用 fake-NUMA，我们可以将物理内存地址空间划分为几个连续的 64 GB 区域（fake-NUMA 节点），确保位于给定的 64 GB NUMA 节点内的地址不会发生冲突（即它们不会映射到 HBM 缓存中相同的位置）。因此，如果应用可以在单个 fake-NUMA 节点内运行，就可以避免出现冲突未命中情况。

如果系统配置了 fake-NUMA，对于内存占用空间适配 HBM 大小的应用来说，不需要额外采取措施来避免产生冲突未命中情况。应用启动时会自动从 fake-NUMA 节点 0 开始分配内存，然后再转到其他 fake-NUMA 节点进行分配。不过，用户可以使用 numactl 将应用放置在任何 fake-NUMA 节点中。例如，在应用会占用将近 64 GB 内存这种较为罕见的情况下，将应用放置在 fake-NUMA 节点 1（而非节点 0）中可以略微提高性能。这是由于 fake-NUMA 节点 0 的可用内存通常略小于其他 fake-NUMA 节点，因为操作系统会从每路的第一个 fake-NUMA 节点预留一些内存。使用 numactl -H 可以查看所有 fake-NUMA 节点。

SNC4 增加了 fake-NUMA 节点的数量。但用户无需额外采取措施就能使用 fake-NUMA，因为默认的操作能够确保适配 HBM 容量的应用进行内存布局时不会发生冲突。

如果应用所占的内存超过 HBM 容量大小，fake-NUMA 会继续按顺序从 fake-NUMA 节点分配内存。这会使分配行为更可预测。

第 7 章
应用配置

本章介绍如何为英特尔® 至强® CPU Max 系列配置常用基准测试和工具。

7.1 软件环境

英特尔® oneAPI 工具套件，如英特尔® oneAPI 基础工具套件（基础套件）和英特尔® oneAPI HPC 工具套件（HPC 套件），提供编译器、分析器（如英特尔® VTune™、英特尔® Advisor）和库[如英特尔® 数学核心函数库（Intel® Math Kernel Library，英特尔® MKL）、英特尔® MPI 库]，支持英特尔® 至强® CPU Max 系列。

7.2 冒烟测试

以下基准测试可作为冒烟测试，用于验证系统的性能是否合适。它们应在系统启动后运行才能进行性能验证。此外，这些测试在各批作业前后的运行速度要足够快才能验证各系统的预期性能。

前两个测试[英特尔® Memory Latency Checker（英特尔® MLC）]和 STREAM 测的是内存系统性能（带宽和时延），HPL 测的浮点计算性能（GFLOPS）。HPCG 对内存带宽最为敏感。

7.2.1 英特尔® Memory Latency Checker（英特尔® MLC）

英特尔® MLC 提供的是颇为详细的单个系统的时延和带宽测量结果。测试系统时可使用以下命令：

峰值带宽：mlc --peak_injection_bandwidth -Z -X -t60

带宽矩阵和时延：mlc

7.2.2 STREAM

STREAM 基准测试提供的是单个节点上不同例程（如 Copy、Scale、Add、Triad）的带宽测量结果。

为了在英特尔® 至强® CPU Max 系列处理器上实现出色性能，请使用以下命令行启用软件预取：

```
icc -O3 -xCORE-AVX512 -qopt-zmm-usage=high -mcmodel=large -qopenmp -qopt-streaming-stores=always -fno-builtin -qopt-prefetch-distance=128,16 -DSTREAM_ARRAY_SIZE=500000000 -DNTIMES=500 stream.c -o stream
```

使用以下命令执行生成的二进制文件：

```
KMP_HW_SUBSET=1t KMP_AFFINITY=balanced,granularity=core,verbose ./stream
```

7.2.3 HPL

英特尔® LINPACK* 分发版基准测试以对高性能 LINPACK（HPL）的修改和补充为基础，与英特尔® oneAPI 数学核心函数库（oneMKL）公共库版本[或作为英特尔® oneAPI 基础工具套件（基础套件）的一部分]一同提供，并附有说明。测试中会测量分解和求解双精度随机稠密线性方程组所需的时间，将该时间转换为速率（GFLOPS），并对结果的准确性进行检验。

该基准测试可以在单节点或多节点上运行。有关如何运行该基准测试的说明可通过上面的链接获取。例如，要在采用双路系统的单节点上以“仅 HBM”模式和“SNC4”集群模式运行，须更改 runme_intel64_dynamic 文件中的以下定义：

```
export MPI_PROC_NUM=2
export MPI_PER_NODE=2
export NUMA_PER_MPI=4
```

然后运行以下命令：

```
./runme_intel64_dynamic -p 2 -q 1 -b 384 -n 120000
```

注：在采用 fake-NUMA 的“缓存”内存模式下，NUMA_PER_MPI 应等于单路系统上的 fake-NUMA 节点数。

7.2.4 HPCG

面向英特尔® CPU 优化的 HPCG 基准测试（源代码和预构建的二进制文件）可以与最新的英特尔® oneAPI 数学核心函数库（英特尔® oneMKL）公共库版本[或作为英特尔® oneAPI 基础工具套件（基础套件）的一部分]一起下载，其中附有开发人员指南。

要从源代码构建，请使用以下命令行：

```
# source C/C++ compiler, MPI compiler, and MKL library
#

export MKLROOT=/path/to/mkl

export LD_LIBRARY_PATH=${MKLROOT}/lib/intel64:${LD_LIBRARY_PATH}

# build binary for Intel AVX-512 -- bin/xhpcg_skx will be created
#

./configure IMPI_IOMP_SKX
make -j4 MKLROOT=${MKLROOT} MKL_INCLUDE=${MKLROOT}/include
```

要以“仅 HBM”内存模式和“SNC4”集群模式在采用英特尔® 至强® CPU Max 系列处理器（每个处理器有 56 个内核）的双路系统上运行，您可以使用以下命令行：

```
      # Note: Select the best MPI x OMP decomposition for your case
      #       Following assumes SNC4 (8 NUMA nodes on 2S),
      #       and 14 cores (28 threads) on a NUMA node
      #

      export MKL_NUM_THREADS=28
      export OMP_NUM_THREADS=28

      nprocs_per_node=8
      nnodes=1
      nprocs=$((nnodes*nprocs_per_node))

      problem_size=168          # options: 168, 192, 256
      run_time_in_seconds=100   # 100 used as smoke test.
                                # 1800 is min for official HPCG submission

      export I_MPI_SHM=spr-hbm
      export I_MPI_FABRICS=shm:ofi
      export I_MPI_PIN_DOMAIN=numa
      export I_MPI_DEBUG=10     # print out mpi configuration mapping data

      # for 1 hyper-thread, use 'compact,1,0' instead of 'compact'
      #
      export KMP_AFFINITY=granularity=fine,compact

      echo " ===          nnodes: ${nnodes}"
      echo " ===             ppn: ${nprocs_per_node}"
      echo " ===          nprocs: ${nprocs}"
      echo " === n_omp_per_proc: ${MKL_NUM_THREADS}"
      echo " ===       prob_size: ${problem_size}"
      echo " ===        run_time: ${run_time_in_seconds}"

      # run bin/xhpcg_skx binary (either prebuilt or built by user)
      #
      mpiexec.hydra -genvall -n ${nprocs} -ppn ${nprocs_per_node} bin/xhpcg_skx -
n$problem_size -t$run_time_in_seconds
```

7.3 明确应用的内存使用情况

为了获得更好的性能，应用通常需要适配 HBM 容量。为此，了解应用和工作负载的内存占用情况非常重要。以下工具可用于此目的：

- top 和 htop（提供系统在某一给定时间的已用/可用总内存情况）
- numastat -m（提供各 NUMA 节点在某一给定时间的总内存情况）
- numastat -p <binary_name>（提供特定进程在给定时间的内存消耗情况）
- /usr/bin/time -v <app_cmd_line>（提供关于应用的多种统计数据，包括*最大常驻内存集*）。这与 bash 内置的'time'命令不同，因此必须指定路径。

如果应用占用的内存大小超过可用的 HBM 容量，可以考虑扩展到更多节点或使用"缓存"模式。

7.4 面向内存带宽优化应用

由于英特尔® 至强® CPU Max 系列处理器提供的内存带宽远高于前代英特尔® 至强® 处理器，因此在前代处理器上受内存带宽限制的应用（或例程）在英特尔® 至强® CPU Max 系列处理器上可能不会遇到这种情况。

确定应用的实际带宽利用率的好办法是使用英特尔® VTune™ Profiler 的内存访问分析功能。如果分析显示应用的给定阶段或例程未充分利用内存带宽，则可以考虑采用以下选项针对内存带宽进行优化：

优化计算：如果例程的计算速度不足以产生足够的内存带宽，可以考虑优化计算（如地址计算），并通过矢量化来提高计算吞吐量。英特尔® Advisor 可用于识别矢量化机会并提高矢量化质量。

优化内存时延：如果例程产生多次未命中 L3 缓存的内存访问，但仍未充分利用 HBM 带宽，则例程可能会受到内存时延限制。降低硬件预取器有效性的不规则访问模式（如间接访问、收集、分散）往往会导致高内存时延。可以考虑针对此类访问模式使用软件预取。英特尔® oneAPI 编译器提供编译器标志（如 -qopt-prefetch）并支持显式预取指令和可在源代码内使用的内联函数（C 语言中的 _mm_prefetch 和 FORTRAN 中的 mm_prefetch）。

有关英特尔® 至强® 处理器和英特尔® 至强® CPU Max 系列处理器的整体架构和优化详情，请见英特尔软件开发人员手册、软件优化参考手册以及 GitHub。