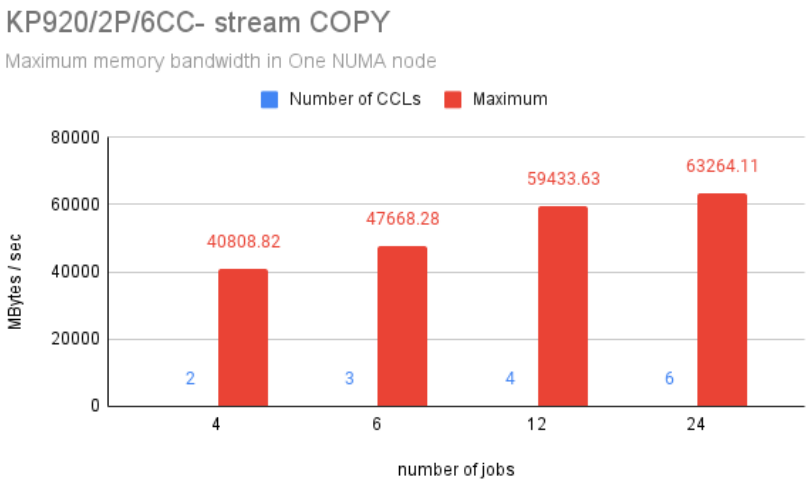
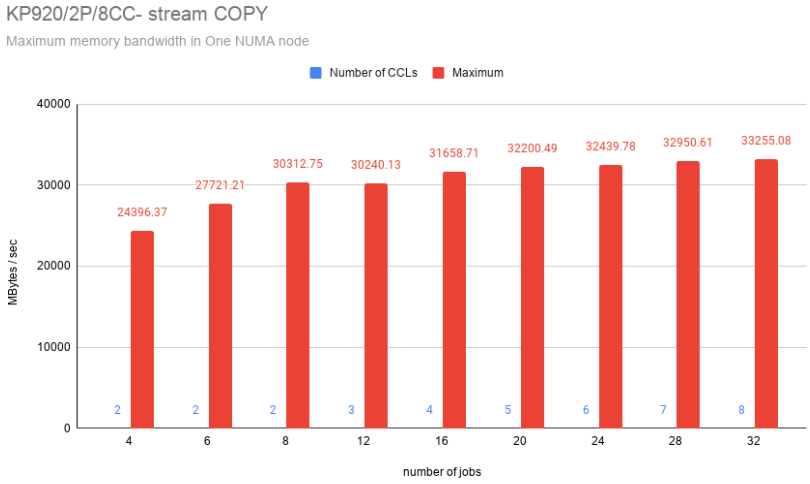
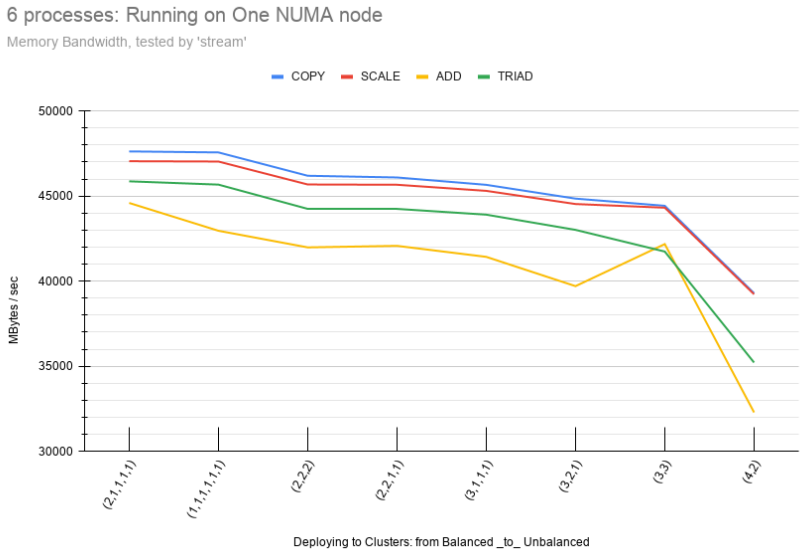
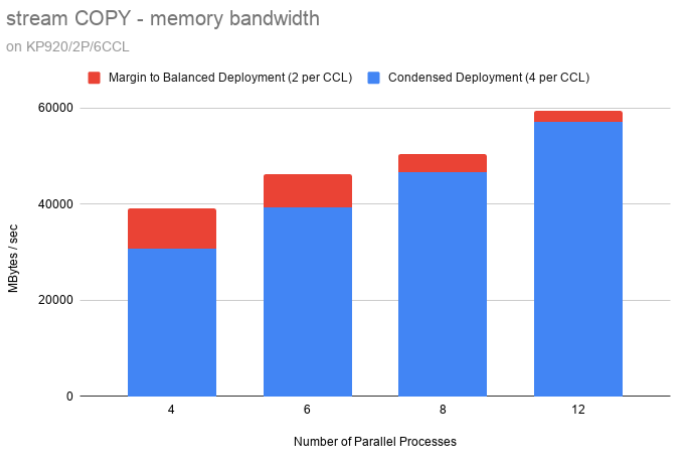
总结 2021/04/26:

Machine details: https://docs.google.com/document/d/1-eTN1eb6HogMyD-pbs-6AWb4mrFzZkkGWb1i7vGmdIU/edit?usp=sharing

KP920/2P/6CCL	KP920/2P/8CCL
CCL 内存带宽	
1. 在12个并行进程及以内,当每个CCL 两个进程(占用两个core)时, 测得的内存带宽达到最高. 把进程分布到更多CCL 不能显著增加内存带宽. 比如: 4进程, 分配到 2 CCL 内存带宽最高; 分配到 3CCL/4CCL, 内存带宽没有显著增加 (<4%). 2. 在16个并行进程及以上时,每个CCL都会至少有两个core被占用. 此时, 在保证一个进程一个core的前提下,把所有进程分配到4CCL, 5CCL,或者6CCL时, 所获得的总内存带宽没有本质区别. 3. 对于单进程实际获得的内存带宽, 1 NUMA 只跑 1 进程时, 单进程带宽最高. 随着进程数增多, 平均到每个进程的单进程带宽呈现增量下降趋势. 举例: (4进程内存总带宽 / 4) < 单进程带宽. 再举例: (12进程总带宽 / 2) < 6进程总带宽. 4. 均衡点大约出现在16进程/6CCL. 即, 并行进程总带宽 在 16个进程分布到6CCL (或5CCL) 时,达到最高. 此时可以认为单 NUMA 节点的内存带宽利用率达到了最大. 5. 多进程CCLs间的不均衡分配, 不如 均衡分配. 举例: 4进程分配到2CCL, (2:2)分配所得内存带宽 大于 (3,1)分配. 再举例: 6进程分配到	1. 在这台机器上看到的数据变化与 6CCL 机器非常不同。把多进程(从 4～ 32 进程)密集部署(每个CCL 跑 4个进程) 或者 稀松部署(每个CCL 只跑 1个进程), 测得的内存带宽基本一致。并且 stream 四种测试都是这个结果。 * 最大的差别也就是在 4 进程时, 不同部署方式的最大/最小带宽能相差 5～9%。 * 而并行 6～28进程, 不同部署方式的最大/最小带宽 相差小于 1%。可忽略不计。 2. 总带宽在 20个并行进程时达到接近最大值 (32GB/s), 之后并行进程数增加到24, 28, 32, 也并没有获得太多带宽增益。最大在32进程时 , 总带宽 (33GB/s) * 20进程/8CCL 这个并行度, 与 6CCL机器的 16进程 是大致对应的。都接近NUMA单节点满载的 三分之二。 3. 对于单进程实际获得的内存带宽, 1 NUMA 只跑 1 进程时, 单进程带宽最高. 这一点与 6CCL机器 相同。可以总结为 : 单进程带宽收益随着进程数增加而 递减。

3CCL, (2,2,2)分配所得内存带宽 大于 (4,1,1) 或者 (3,2,1)分配. 注(2021/04/12): 2. 16进程 4CCL/ 5CCL /6CCL, stream copy 带宽依次是: 64677, 65793, 65246. 最高/最低 = 1.017, 不足 %2 的差别，所以我说"没有本质区别"。 4. 如上的数据，最高，但是也是与 16进程/4CCL 差别不大。如果资源紧俏，我觉得部署到 4CCL 就够了。不过，或许需要指出的是，针对这个最大，我主要是指 STREAM copy/scale/triad 三种测试。 - 如果单点看 STREAM add 测试，最大值出现在 20进程/6CCL 时。 5. 这个我补充了测试。4进程，6进程，非均匀分配下，我都测过。不如(2:2:2)均匀分配。	异常数据点: 1. 并行8进程所获得内存总带宽 高于 并行12进程(某几个case下还高于20进程)。期待的趋势是进程数越高带宽越高。
1. 柱线1：横轴进程数，纵轴该进程数情况下，内核带宽峰值； 2. 柱线2:横轴进程数，纵轴均匀分布到几个CCL后，基本再扩散开已经没作用了(或者作用已经非常不明显了)，CCL数量拐点。 注:此处只统计 stream COPY  	

6 Jobs, running in One NUMA node, from Balanced deployment to Unbalanced deployment: 可以看出（2，2，2）好于（3，2，1）。以及总体上来说，放在 3CCL 好于 放在2CCL.  	
Memory bandwidth difference: 最差的情况（每CCL四个进程）与 均衡部署（每CCL两个进程）的差别：	



CCL 内存延迟

1. stream 四种测试所呈现的趋势是一致的。（COPY/SCALE/ADD/TRIAD）
2. 每 CCL 两个进程时，内存延迟最小。即：
   - 4进程分布在 2CCL，延迟最小。分布到 3CCL， 4CCL 意义不大。
   - 6进程分布在 3CCL，延迟最小。
   - 12进程分布在 4CCL/5CCL/6CCL，延迟都比较小。最差的是分布在3CCL，此时每个CCL都是满载4进程的。
3. 跑24进程（满载）测得的内存延迟 是 跑12进程（半载）的一倍。但两者的总内存带宽却是差不多的。
   - 结论：24进程造成了更多的拥堵，却没有获得更多的带宽。

1. 与左侧相同。
2. 内存延迟跟进程部署方式没有关系（稀松部署 vs. 密集部署 延迟相同）。
3. 与左侧一致。跑32进程（满载）测得的内存延迟 是 跑16进程（半载）的一倍 (15.4 -> 8.1 ns)。但两者的总内存带宽却是差不多的（33 -> 31 GB/s）。
   - 综合来看，在单 NUMA 节点上，半载时，能获得内存带宽和延迟的最佳平衡。（同意？）

跨NUMA 内存带宽

1. 彼此不存在内存重叠的并行进程,在 跨NUMA 运行时,所测得的 总内存带宽 与单NUMA 相比,呈等比例增长, 符合预期. 比如: 8进程 均衡分布在两个 NUMA 的 总内存带宽 = 4进程 均衡分布在一个 NUMA 的 总内存带宽 * 2. 再比如: 24进程 均衡分布在四个 NUMA 的 总内存带宽 = 6进程 均衡分布在一个 NUMA 的 总内存带宽 * 4. 2. 均匀部署 优于 不均匀部署. 举例: 8进程部署到 三个 NUMA, (3,3,2) 优于 (4,2,2) 3. 综合结果, 在均匀部署的前提下, 从 4进程 到 72进程, 分布到 1NUMA / 2NUMA / 3NUMA / 4NUMA, 都呈现出 总内存带宽 线性增长的趋势, 4NUMA 最高. 比如: 24进程, 从 1NUMA 到 4NUMA, 对应的内存带宽依次为 (63, 117, 149, 181).	1. 与左侧一致。 2. 参照 “CCL 内存带宽”， 部署方式在这台 8CCL 机器上不重要。 3. 与左侧一致。 异常点: 1. 16进程分布于 2 NUMA 和 3 NUMA 时，测得的总带宽大于20进程。期待的情况是 20进程总带宽更大。

跨NUMA 内存延迟

1. 当我们把 多进程分散到 多NUMA节点时，内存访问延迟按照预期，呈现了逐步下降趋势。 - 这个趋势在 12进程 和 24进程时，最为明显。举例:有12个进程时，放在一个NUMA上，内存延迟是 3ns;放到四个NUMA时，内存延迟就能下降到 1.5ns. 2. 4进程时，分布到1NUMA 和 4NUMA 区别并不大。虽然4NUMA更快，但是也仅有20%左右的缩短。	1. 趋势性比内存带宽的好，没有数据异常点。 2. 每个 NUMA 节点的进程数越多，那么其内存延迟越大。 3. 同样数量的进程，分散部署在 4 NUMA 的延迟，到 3 NUMA，到 2 NUMA，到 1 NUMA 的延迟，呈现递增趋势。

NUMA0 CPU 分别访问 NUMA0/1/2/3 的内存带宽 对比

1. 在这个测试里面，测试结果显示出：跨NUMA访问内存时，所获得的内存带宽变化规律，与实际计算类型/访问模式强相关。目前无法简单的总结。
比如：stream COPY/ SCALE/ ADD，这三个测试所呈现的规律基本一致。而 stream TRIAD 测试所呈现的趋势有时相反。具体见下面描述。

4. 计算和内存 位于不同package的 NUMA节点时（如NUMA0 和 NUMA2）
- 所有的四种 stream 测试都呈现出内存带宽下降的情形。但是，
- 但是，所测得的下降比例有很大不同。对于 stream COPY/ SCALE/ TRIAD，带宽下降比例在 30%左右。而 stream ADD 带宽下降比例高达 60%。

5. 多进程在计算节点在 NUMA0 上的分布形态（比如4进程部署于 2CCL，3CCL，或者4CCL），对最终结果影响不大。

6. 在多次测试中，发现同一测试用例，有数据不稳定的情况。选择暂时忽略这些异常数据点。比如 cpu（0，1，4，8）membind 1 ADD bandwidth. 这从另一个侧面反映出这个测试的动态随机性。

注（2021/04/19）：
针对第 3. 条，在多进程对比后，发现应该更正如下。
针对第 5. 条，更正为“跨numa访问内存的带宽，...和不跨NUMA访问本NUMA内存的CCL带宽特征相似”。
在改进的测试用例中，我兼顾从 1 job 到 16 jobs 不同的任务负载。期望整理出跟普适的规律。总结如下：

1. 对于并行 1 Job 和 4 jobs, stream COPY/ SCALE/ ADD 这三种任务的最大带宽，出现在内存绑定在相同 NUMA node 时。次之的，是内存绑定在同package的不同 NUMA node时（比如 计算在 NUMA0 --> 内存在 NUMA1）。
- 而对于运算更复杂些的 stream TRIAD，并行 1 job 和 4jobs 的最大内

1. 跟左侧相比，同package，有更多的情形 membind=1 的带宽 高于 membind=0：从 4 进程并行 到 16进程，所有四种 stream 测试（COPY/SCALE/ADD/TRIAD）无一例外。
- 甚至单进程时，stream ADD 和 stream TRIAD，也有这种现象。可重复。

2. 有很多情景，内存放在不同 package 时，得到的内存带宽高于 内存放在本地。membind=2，3 大于 membind=0 的情景。
- 包括：stream COPY/SCALE/ADD/SCALE：6，8，12，16进程。

3. 4 进程并行时，membind=2，3 跟 membind=0 相比，内存带宽差不多（稍微高/稍微低都有，但是差别不大）。本地内存并没有优势。

4. 单进程时，
- stream COPY/SCALE：membind 0 > 1 > 2 > 3. 符合预期。
- stream ADD/TRIAD：membind 1 > 0 > 2 > 3.（1 > 0) 是不符合预期的。

5. 虽然在 CCL 测试中（如上面”异常点“描述），8进程 stream SCALE 带宽高于 12进程的，但是
- 当把内存节点放到 membind=1，2，或3 时，测得的内存带宽又变回来了：8进程 < 12进程，符合预期。

存带宽，一律是出现在 内存绑定在同package的不同 NUMA node时（比如 计算在 NUMA0 --> 内存在 NUMA1 ）。 2. 当讨论并行 6 jobs ～ 16 jobs 时，更普遍的情况是无论哪种 stream 任务，都是 最大内存带宽 出现在 内存绑定在同package的不同 NUMA node时（比如 计算在 NUMA0 --> 内存在 NUMA1 ）。 - 有少数情况例外。比如 16进程 stream ADD 和 TRIAD 时，最大带宽在本地内存。 3. 计算和内存 位于不同package的 NUMA节点时（如NUMA0 和 NUMA2），内存带宽会普遍小于 计算和内存 位于同一package的 NUMA节点的情况。这符合预期，显示出同package的NUMA节点之间有更高的内存带宽。 4. 在 计算和内存 都位于 同一NUMA节点 时总结出来的内存带宽规律，在 计算和内存位于不同NUMA节点时，多数情况下也是适用的。比如： - 12进程运行在 3CCL 时内存带宽最小，6CCL 时内存带宽最大；当把内存绑定到不同NUMA节点时，这个规律依然成立。 - 再比如，4进程 stream COPY，分布到 2CCL，3CCL，4CCL 所获得的总带宽相差无几；当把内存节点绑定到不同NUMA时，这个规律依然成立。 5. 测试中，发现有些单点的异常情况（可重复）。这些在整理的Excel表格中标注为红色。目前不做进一步分析。比如， - 8进程分布在5CCL，内存绑定到同package相邻NUMA node的情况，所有四种stream测试都表现出了严重低于 4CCL 和 6CCL 的带宽结果。（expected，应该是 5CCL 与 4/6CCL 差不多高。） - stream TRIAD 测试中：4进程分布在2CCL，memnode = 3时，带宽严重^高于^ 4进程分布在 3CCL 和4CCL 时。 - 在 stream TRIAD 测试中，我发现了更多的 membind 异置 时的异常数据点。其中两个与 membind=2 有关。（同样的测试，和 membind =3 结果差异巨大）。	

NUMA0 CPU 分别访问 NUMA0/1/2/3 的 NUMA 节点间内存延迟

这里的描述是基于 4进程并行的 stream 测试。 1. 在四种 stream 测试中，结果不一致。有的跟 ACPI SLIT table 中保存的	1. 出现大多数情况下，membind=1 的 latency 最小。 - 各NUMA节点的 内存延迟 总体排名：membind=1 > 2 > 3 > 0

NUMA distance 相同，有的差别很大。这里的结论应该是要根据业务类型结合起来判断，是内存bounded，还是计算bounded。 2. 具体说，从 NUMA0 到 其余NUMA 节点（NUMA0，1，2，3），延迟分别是： - COPY/SCALE : 10, 11, 13, 14 （Note：体现单纯的 内存 bounded 的极限） - ADD : 10, 12, 22, 23 （Note：与 ACPI NUMA distance 一致） - TRIAD : 10, 8 , 12, 13 （Note：这个测试结果，为什么邻居节点 反倒比 本地节点（ 8 vs. 10) 还快？需要解释 ） 3. 如上，为什么 TRIAD 时 NUMA0 访问 NUMA1 比访问本地NUMA 还快？ 4. 前提：因为指定了 membind=0（或1，2，3），我认为没有发生数据迁移的情况。	这里有很大的问题，为什么本地节点的内存延迟最高？

NUMA interleave 的内存带宽，计算节点跟随

1. 背景：这个测试中的内存在哪几个NUMA节点间 interleave， cpu计算也被部署到同样的几个NUMA节点中。24进程并行。 2. 在所有cpu中依次 +2, +3, +4 间隔部署，所产生的效果与 按照每CCL中使用两个 core 的效果：相同，没有区别。都属于有效的均衡部署。这跟前述结论一致。 3. 趋势：总内存带宽在 interleave 到 3个NUMA时，达到最大。再往下，依次是：3 NUMA > 4 NUMA > 2 NUMA > 1 NUMA (其中 ADD 在 2 NUMA > 4 NUMA) - interleave，从 1NUMA 到 2NUMA，总内存带宽增涨 +50% 从 2NUMA 到 3NUMA，总内存带宽增涨 +15%，收效就不如跨 2NUMA 时明显了。 从 3NUMA 到 4NUMA，总内存带宽又回落到 跟 2NUMA 差不多的水平。 4. 同样的 24进程， interleave 比 不做interleave，总内存带宽要差。 - 2NUMA interleave, 损失 -17% - 3NUMA interleave, 损失 -25%	1. 背景：这个测试中的内存在哪几个NUMA节点间 interleave， cpu计算也被部署到同样的几个NUMA节点中。32进程并行。 2. 趋势：与左侧不同。interleave 到 越多NUMA节点，总内存带宽越大。4 NUMA > 3 NUMA > 2 NUMA > 1 NUMA. 单调上升。 - 分散到 4 NUMA时，内存带宽是 1 NUMA 的3.0倍。 总结：这台机器的 interleave 内存带宽收益，远高于左边的 6CCL 机器。（左侧的 6CCL 在最高时 interleave 的内存带宽 是 1NUMA 的 2.0倍）

- 4NUMA interleave, 损失 -40%	

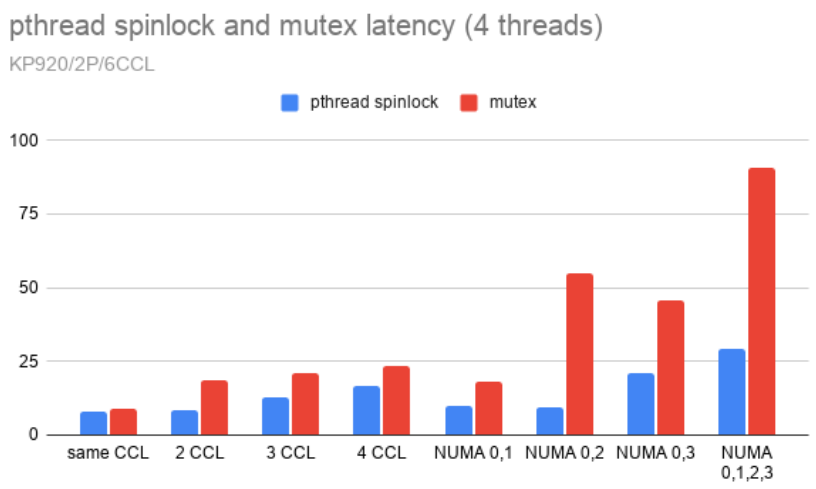
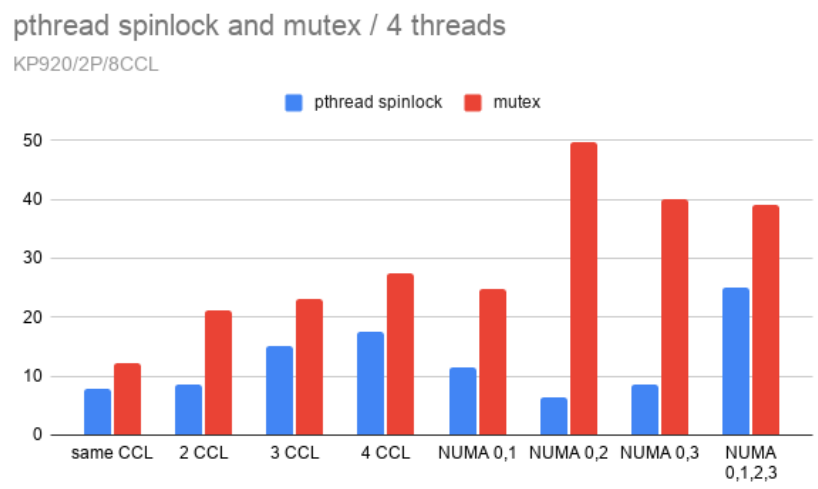
NUMA interleave 的内存延迟，计算节点跟随

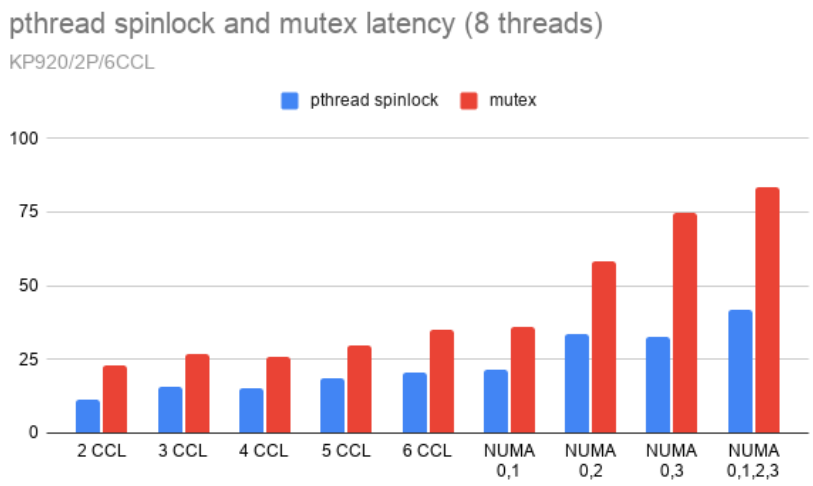
1. 这个测试中，采用了 24进程。这个进程个数对于 NUMA0 来说，是饱和的。所以，当 interleave 到 两NUMA 节点时，按照期待，会出现压力减轻，延迟变小的情况。 实际测试符合这个预期。并且，在 stream 四种测试结果所呈现的趋势，是彼此 一致 的。显示的都是如果使用 interleave 时，访问延迟 与 业务类型大概率是无关的。 2. 在 interleave 到 三个节点 NUMA（0，1，2）时，内存延迟最小。 3. 在 interleave 到 两节点 和 四节点，内存延迟差不多大。 接下来，采用 12进程，看看 interleave 是否仍然有利： 4. 呈现出与 24进程不同的趋势。 5. 对于 COPY/SCALE/TRIAD 三种测试，interleave 到 两个 / 三个 / 四个 NUMA 节点后，内存延迟都比只在一个 NUMA 节点时缩短了。并且 interleave到 2/3/4个NUMA节点的延迟都差不多大，4节点延迟最小。 6. 而对于 ADD 这个测试，看到的趋势有所不同。interleave 到 两个 NUMNA时延迟最小。4NUMA 和 1NUMA 延迟一样，最大。3NUMA 居中。	1. 这个测试中，采用了 32进程。这个进程个数对于 NUMA0 来说，是饱和的。 2. 趋势：与左侧不同。interleave 到 越多 NUMA 节点，内存延迟 就越低，单调下降。

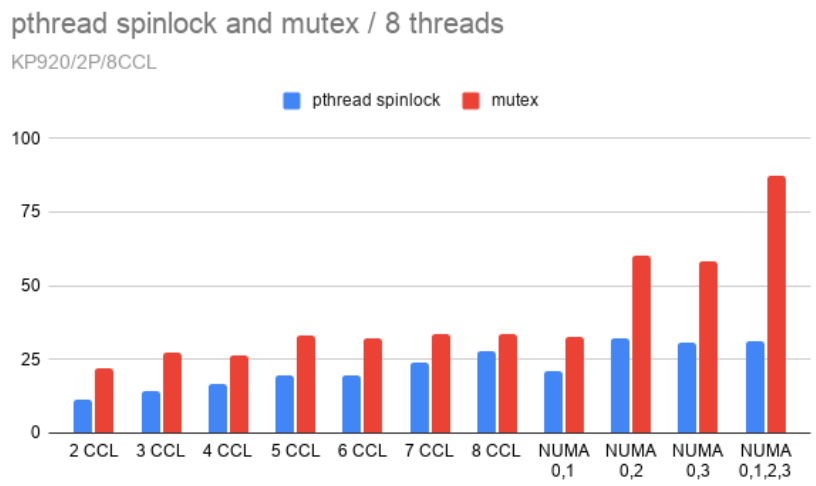
NUMA interleave 的内存带宽，计算节点固定 NUMA0

1. 背景：测试从 4进程 到 24进程 的情景。所有进程固定在 NUMA0. 尽可能的半满部署（2 jobs/CCL)。	1. 背景：测试从 4进程 到 32进程 的情景。所有进程固定在 NUMA0. 尽可能的半满部署（2 jobs/CCL)。

2. 所有情景中，interleave 到同package 的两个 NUMA节点（同package的，0 和 1）所测得的内存带宽都是大于只使用本地NUMA节点的。并行进程数大于8时更明显。 3. 然而，在 interleave 到 三个 NUMA节点 或者 四个 NUMA节点，内存带宽就比 两节点 小。总结：interleave 到 三 或 四 NUMA 得不偿失。	2. 所有情景中，interleave 到同package 的两个 NUMA节点（0 和 1）所测得的内存带宽都是大于只使用本地NUMA节点的。 3. 并行进程数小于等于8时，interleave 到 三节点（0，1，2）和 四节点（0，1，2，3）内存带宽跟　两节点（0，1）提升不大。高复杂性的任务　stream TRIAD 时 三/四节点 还会下降。 4. 并行进程数为 12 及以上时，interleave 到 的节点数量越多内存带宽越大：四节点（0，1，2，3）> 三节点（0，1，2）> 两节点（0，1）> 单节点（0）。
NUMA interleave 的内存延迟，计算节点固定 NUMA0	
1. 背景：同上。 2. 结论与内存带宽类似，interleave 到 两 NUMA节点时的 内存延迟最小。interleave 到 三或者四 NUMA节点时 内存延迟都比 两NUMA 大了。	1. 背景：同上。 2. 几乎所有测试结果都符合预期的趋势，单调增或减： - 进程数越多，内存延迟越大。 - interleave 到的节点数越多，内存延迟越小。 异常点： 3. 当 4 或 6 进程 interleave 到三节点（0，1，2）时，测试中有几处异常点，偏离上述2中的趋势。但差别并不大。
pipe延迟，CCL内/间, 跨NUMA	
1. 在相同 CCL 内，pipe延迟 最小。从 NUMA0 到 NUMA3，pipe延迟最大 总结： same CCL < cross CCL < NUMA 1 < NUMA 2 < NUMA3	1. 在相同 CCL 内，pipe延迟 最小。NUMA1，pipe延迟最大，不符合预期。 总结： same CCL < cross CCL < NUMA 2 < NUMA 3 < NUMA1
lock延迟，pthread spinlock/mutex， CCL， NUMA	

1. 背景：采用4线程互锁测试，(4/27) 增加了 8线程 的测试用例。 2. 在同一NUMA节点内，4线程分布在越多的 CCL， 则最终的 pthread spinlock 和 mutex 延迟越大。8线程也同样的规律。 3. 对于所有测试情形，都有 pthread spinlock 延迟 小于 mutex 延迟。 4. 当需要跨两个 NUMA 节点时，无论 4进程 还是 8进程，无论 pthread spinlock 还是 mutex，都是分布在 NUMA 0，1时延迟最小。(比分布在 NUMA 0，2 或者 NUMA 0，3 要小)。 5. 所有情景中，跨 四个 NUMA节点的线程间互锁，都是延迟最大的。符合预期。	1. 背景：采用 4线程 和 8线程 互锁测试 2. 在同一NUMA节点内，4 线程分布在越多的 CCL， 则最终的 pthread spinlock 和 mutex 延迟越大。8 线程也同样规律。 3. 8线程分布在两个 NUMA 节点时，NUMA 0，1 之间的 lock 延迟最短。pthread pinlock 和 mutex 同样规律。 4. 4线程分布在两个 NUMA 节点时，pthread pinlock 延迟在 NUMA 0，2 在之间最短； mutex 延迟在 NUMA 0，1之间最短。
	





代码交付

1.相关测试脚本，以及测试数据 https://gitee.com/docularxu/wayca-deployer/tree/working-kp920-6ccl-2p-benchmarking/（包含6CCL 和 8CCL）

- 6CCL: kp920.2P.6CCL/log.*
- 8CCL: kp920.2P.8CCL/log.*

2.测试工具-lmbench，原始代码来源于 https://jaist.dl.sourceforge.net/project/lmbench/development/lmbench-3.0-a9/lmbench-3.0-a9.tgz
有单点修改一个绑定CPU的bug，上传在 github:URL 和 OpenEuler-lmbench: URL.

- Build: cd src; make clean; make results;

3.lock-test，测试代码有修改，上传在 https://github.com/docularxu/benchmarks/tree/working-timing

- Build: cd lock; gcc -pthread lock_test.c -o lock_test;