实验七——并行实验

一、 实验目的

1) 探索糟糕并行代码的性能瓶颈

2) 使用 gem5 以及一个详细的缓存模型来发现性能变化规律

二、 实验步骤

步骤 I：真实硬件实验

1）在拥有至少 4 个核心（最好是 8 个或更多）的计算机上运行不同的并行算法来对数组求和。即，在真实硬件上使用 1、2、4、8、16 个线程运行 6 个不同的并行算法（不要超过硬件的最大线程数）。例如，如果只有 4 个核心，就不要运行 8 和 16 个线程。

步骤 II：gem5 模拟实验

1）实验配置：主板为 HW5X86Board，处理器为 HW5O3CPU，缓存层次结构为 HW5MESITwoLevelCacheHierarchy，内存为 HW5DDR4，时钟频率为 3GHz；

2）设置 16 个核心进行模拟，并把数组大小设置为 32768。

三、 实验结果

（一）步骤 I

线程数	naive	chunking	res-race	chunking-res-race	block-race	all-opt
1	0.8215	0.9352	0.9132	0.8990	0.9378	0.8911
2	1.3337	0.8225	1.1756	1.2244	0.6432	0.7905
4	1.3027	1.5064	1.7985	0.9995	0.7526	0.7878

上表中的数值表示程序运行时间（ms）（数组元素个数：32768）。

1）问题 1

（a）对于算法 1，增加线程数量是提高性能还是降低性能？使用数据支持你的答案。

当线程数量从 1 增加到 2 时，程序运行时间从 0.8215ms 增加到了 1.3337ms，性能下降；当线程数量继续增加到 4 时，运行时间变成了 1.3027ms，比线程数 2 性能略微提升，但仍低于线程数 1。

2）问题 2

（a）对于算法 6，增加线程数量是提高性能还是降低性能？使用数据支持你的答案。

当线程数量从 1 增加到 2 时，程序运行时间从 0.8911ms 降低到了 0.7905ms，性能提升；当线程数量继续增加到 4 时，运行时间变成了 0.7878ms，性能进一步提升。

（b）当使用 2、4、8、16 个线程时的加速比是多少（只回答不超过系统核心数的部分）。

线程数	naive	chunking	res-race	chunking-res-race	block-race	all-opt
2	0.616	1.137	0.777	0.734	1.458	1.127
4	0.631	0.621	0.508	0.899	1.246	1.131

上表中的数值表示各种算法相对于线程数 1 的加速比。

3）问题 3

（a）根据以上 6 个算法的结果数据回答什么是最重要的优化，是对数组分块、对 result 分配不同的访问地址、还是把 result 访问地址之间进行对齐？

通过实验数据可以发现，对数组分块或对 result 分配不同的访问地址后，程序的运行时间并没有得到充分改善，甚至在有些线程数下反而性能下降。而把 result 访问地址对齐后，程序的运行时间相对于 naive 算法得到了明显的改善。

因此，最重要的优化是对齐 result 访问地址。

（b）推测硬件实现是如何导致这个结果的。硬件的哪些特性导致这个优化最为重要？

由于硬件的缓存访问以块为单位，当对齐 result 访问地址后，可以确保每个线程频繁访问的数据位于不同的缓存块中。这样既避免了多个线程试图同时写入相同的内存位置所导致的竞争条件，也避免了多个线程访问相邻的内存位置时可能引起的缓存行失效，从而提高了缓存效率和内存访问的局部性。硬件的这些特性导致了对齐 result 访问地址这个优化最为重要。

（二）步骤 II

线程数	naive	chunking	res-race	chunking-res-race	block-race	all-opt
1	0.866	0.867	0.867	0.877	0.856	0.867
2	0.784	0.771	0.784	0.782	0.470	0.445
4	0.819	0.815	0.818	0.818	0.276	0.238
8	0.881	0.880	0.881	0.880	0.181	0.142
16	0.917	0.915	0.917	0.915	0.145	0.112

上表中的数值表示程序运行时间（ms）（数组元素个数：32768）。

4）问题 4

（a）使用 gem5 算法 1 和 6 在 16 个核心上的加速比是多少？

线程数	naive	all-opt	all-opt 相对于 naive 的加速比

1	相对于线程数 1 的加速比		0.999
2	1.105	1.948	1.762
4	1.057	3.643	3.441
8	0.983	6.106	6.204
16	0.944	7.741	8.188

（b）这与你在真实系统上看到的情况相比如何？

对于算法 1，当线程数为 2 或 4 时，在 gem5 上性能相较于线程数 1 有略微提升，但在真实系统上，性能却是下降的；对于算法 6，当线程数为 2 或 4 时，在 gem5 和真实系统上，性能都得到了提升，但 gem5 上的加速比明显高于真实系统。

5）问题 5

（a）哪种优化（对数组进行分块、使用不同 result 地址或在 result 地址之间放置填充）对命中率影响最大？

展示你用于做出这一判断的数据。讨论你比较的算法以及原因。

	naive		chunking		res-race		block-race	
线程 ID	hit	access	hit	access	hit	access	hit	access
1	25841	29994	31940	34135	27887	32043	29883	32044
2	25472	29656	31552	33799	27522	31706	29558	31702
3	25464	29647	31539	33790	27508	31691	29857	32038
4	25468	29652	31555	33805	27519	31703	29863	32048
5	25468	29651	31545	33795	27517	31700	29858	32033
6	25480	29665	31554	33807	27531	31718	29868	32050
7	25468	29650	31543	33793	27527	31709	29867	32048
8	25470	29654	31553	33803	27524	31708	29879	32059
9	25483	29666	31558	33810	27532	31715	29847	32026
10	25478	29662	31546	33797	27527	31711	29581	31724
11	25770	29987	31832	34113	27815	32029	29574	31719
12	25474	29655	31553	33803	27521	31703	29569	31708
13	25768	29983	31844	34129	27814	32032	29586	31728
14	25759	29972	31841	34125	27817	32033	29575	31715
15	25754	29970	31835	34119	27799	32015	29567	31705
16	25474	29653	31828	34107	27803	32014	29556	31694
命中率	85.92%		93.35%		86.83%		93.23%	

上表中的数值表示的是 16 个线程的 L1DCache 命中次数和总访

问次数，二者相除可得 L1Dcache 命中率。

根据上表数据可以发现，数组分块对命中率的影响最大。

6）问题 6

（a）哪种优化（对数组进行分块、使用不同 result 地址或在 result 地址之间放置填充）对读取共享的影响最大？

展示你用于做出这一判断的数据。讨论你比较的算法以及原因。

线程 ID	naive	chunking	res-race	block-race
1	1809	20	1809	1769
2	12	13	12	52
3	11	13	11	12
4	11	13	11	11
5	11	13	11	14
6	11	13	11	11
7	11	13	11	19
8	11	13	11	15
9	11	13	11	9
10	13	14	13	10
11	9	13	11	17
12	12	8	7	14
13	11	11	10	11
14	8	9	9	11
15	6	8	9	13
16	7	8	8	6

上表中的数值表示的是 board.cache_hierarchy.ruby_system.L1Cache_Controller.Fwd_GETS 的统计数据，即另一个缓存想要获取此缓存中处于共享状态的数据的次数。

根据上表数据可以发现，数组分块对读取共享的影响最大。

7）问题 7

（a）哪种优化（对数组进行分块、使用不同 result 地址或在 result 地址之间放置填充）对写入共享的影响最大？

展示你用于做出这一判断的数据。讨论你比较的算法以及原因。

线程 ID	naive	chunking	res-race	block-race
1	2006	1978	2009	13
2	2061	2056	2061	15
3	2061	2061	2061	7
4	2061	2061	2061	10
5	2061	2061	2061	17
6	2061	2061	2061	7
7	2061	2061	2061	19
8	2061	2061	2061	3
9	2061	2061	2061	8
10	2061	2061	2061	4
11	2052	2049	2050	13
12	2061	2051	2051	14
13	2050	2051	2050	2
14	2051	2051	2050	16
15	2051	2052	2050	14
16	2046	2045	2044	2

上表中的数值表示的是 board.cache_hierarchy.ruby_system.L1Cache_Controller.Fwd_GETX 的统计数据，即某个缓存必须回应另一个缓存请求将数据更改为可写状态的次数。

根据上表数据可以发现，在 result 地址之间放置填充对写入共享的影响最大。

8）问题 8

让我们回到之前的问题 3，“硬件的哪种特性导致这种优化最重要？”

（a）在我们看过的三个特征中，L1 命中率、读取共享或写入共享，哪一个对性能的影响最大？使用平均内存延迟（和整体性能）来回答这个问题。最后，你应该对哪些优化对命中率、读取共享性能和写入共享性能的影响最大有一个概念。

	naive	chunking	res-race	block-race
L1 命中率	85.92%	93.35%	86.83%	93.23%
读取共享	一般	最好	一般	一般

写入共享	一般	一般	一般	最好
平均内存延迟	34.357178	36.136696	33.467483	3.162650
运行时间	0.917	0.915	0.917	0.145

使用数组分块优化时，程序具有最优的 L1 命中率和读取共享，但相较于 naive 算法，其平均内存延迟反而更大，且程序运行时间提升有限。而在 result 地址之间放置填充后，程序具有最优的写入共享，相较于其它算法，其平均内存延迟不到原来的十分之一，程序运行时间也明显减少。

因此，写入共享对性能的影响最大。

（b）使用 gem5 模拟产生的结果数据，来回答什么硬件特性导致哪种优化最重要？

通过模拟数据可以发现，当写入共享的次数最少时，平均内存延迟最小，从而使得程序运行时间最短、性能最好。而对齐 result 地址后，每个线程访问的 result 数据位于不同的缓存块中，避免了竞争条件和缓存行失效的问题，从而使得写入共享的次数最少。

因此，写入共享的硬件特征导致对齐 result 地址的优化最重要。

9）问题 9

（a）将 crossbar 延迟分别设为 1 个周期和 25 个周期（除了你已经运行的 10 个周期）后运行模拟。随着缓存延迟的增加，不同优化的重要性如何变化？（你不必运行所有算法。你可能只需运行算法 1 和算法 6。）

	xbar_latency=1		xbar_latency=10		xbar_latency=25	
	naive	all-opt	naive	all-opt	naive	all-opt
平均 L1 命中率	85.91%	99.45%	85.92%	99.44%	86.31%	99.44%

读取共享	线程 1 为 1808，其余线程均值为 10.5	线程 1 为 22，其余线程均值为 12.7	线程 1 为 1809，其余线程均值为 10.3	线程 1 为 23，其余线程均值为 13.2	线程 1 为 1345，其余线程均值为 41.8	线程 1 为 21，其余线程均值为 13.5
写入共享	均值为 2053.7	均值为 11.6	均值为 2054.1	均值为 11.5	均值为 1939.4	均值为 10.8
平均内存延迟	11.367948	1.253588	34.357178	1.462239	68.935979	1.827219
运行时间	0.349	0.103	0.917	0.112	1.759	0.127

随着缓存延迟的增加，平均内存延迟也明显增加，而数组分块能够减少读取共享的次数，对齐 result 地址能够减少写入共享的次数，从而充分利用缓存层次结构、提升程序性能。因此，数组分块和对齐 result 地址这两种优化更加重要。

四、 实验心得体会

通过本次实验，我了解了不同的优化方法对于并行程序的不同影响，尤其是充分利用缓存层次结构对并行程序性能的巨大提升。此外，我也掌握了如何编写良好的并行代码以提升程序性能。