合并写缓冲（Write Combining, WC）是一种优化访存性能的策略，主要用于提高写操作的效率，减少总线和内存带宽压力。以下是其相关概述：

1. 产生背景

随着处理器性能的提升，访存瓶颈（Memory Wall）成为限制系统整体性能的重要因素。写操作，尤其是小规模的分散写，会导致总线和存储器的高开销，因此需要一种机制来优化写操作，从而减少访存延迟，提高存储带宽利用率。

2. 由来

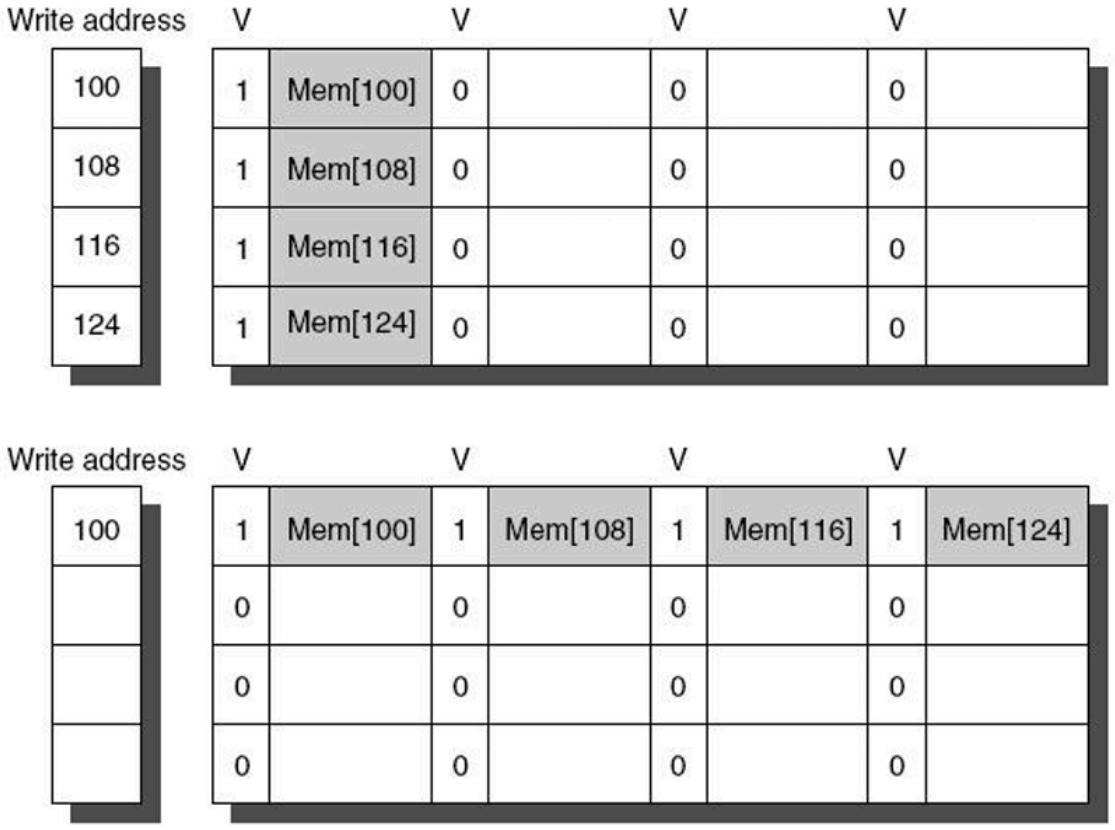
写合并策略最初是为了解决缓存系统中的写失效问题，特别是在处理连续地址的写入请求时，将多个小的写请求合并成一个大的写操作，从而减少访存次数。

3. 工作原理

写合并策略是指，在向写缓冲器写入地址和数据时，如果写缓冲器中存在被修改过的块，就检查其地址，看看本次写入数据的地址是否与写缓冲器内的某个有效块地址匹配，如果匹配，就把新数据与该块 合并，称为“合并写”

写合并应用存在两种情况，分别是写穿透 cache 与写回 cache。这两种 cache应用写合并策略时的优化方式也时不同的。

- **写穿透缓存（Write-Through Cache）**：在写穿透的情况下，连续的对同一个 cache 块中不同字的写失效，则一个cache block 的写失效还未处理返回时，就又发生失效，将这些写失效合并为一个写到内存，多个字写入比单字写入更快，更符合缓存替换行为的特点。这种情况下的优化，是对写穿透过程中使用的写缓冲区的优化。还有简单的情况则是在正常处理写请求时，在写请求发送后，在写缓冲区中进行合并，这样在向下一级发送请求时，可以进行一个缓存块的连续写，加快写请求的处理。在大多数的访存路径上，写操作的完成是需要通过写缓冲区的，读与写操作实际上都与写缓冲区有关，在使用写缓冲区时，读操作实际上可能会在写缓冲区中读取最新消息。
- **写回缓存（Write-Back Cache）**：而对于写回缓存来说，写失效发生时，会产生写分配操作，即为了处理写失效，首先将缺失的块从下一级存储取回该级缓存，再将写数据的内容覆盖该块数据。如多个写失效所写内容可以拼接成一个完整的 cache 块，那么就可以不用再从下一级存储中读取，而直接将合并的写失效的数据块作为新块，放入缓存中。如此可以减少缺失代价，减少访存总线带宽压力。
- 使用写合并策略前后的缓冲区状态



4. 优缺点

优点

- **减少访存总线带宽压力**：合并多个小写入请求，提高内存控制器的写入效率。
- **降低缓存缺失代价**：特别是对于写回缓存策略，减少从内存加载数据的需求，提高缓存命中率。
- **提高访存吞吐量**：适用于访存密集型应用，如多媒体处理、大规模数据计算等。

缺点

- **可能增加写延迟**：如果合并策略不当，可能导致等待写缓冲区填满后才执行写入，增加延迟。
- **一致性管理复杂**：在多核或多处理器系统中，需要额外的机制保证写合并操作不会违反缓存一致性协议。
- **通常不适用于 I/O 地址空间**：I/O 设备通常要求严格的访问顺序，且 I/O 地址空间多为不可缓存（Uncacheable，UC）区域。如果写合并被应用到 I/O 设备，可能导致访问顺序错乱，影响设备的正确性。例如，某些硬件寄存器需要按照特定顺序写入，否则可能出现未定义行为。因此，除非 I/O 设备明确支持写合并（如显存、某些高速 DMA 设备），否则通常应禁用 WC 以确保正确性。

5. 影响合并写缓冲（Write Combining, WC）的因素

合并写缓冲（WC）的性能和效果受到多个关键因素的影响，包括缓冲区数量、数据访问模式、缓冲区管理策略、内存一致性、处理器架构和存储介质特性等。

---

1. 缓冲区数量

缓冲区的数量直接影响写合并的成功率和整体效率：

- **缓冲区越多，合并成功率越高**，减少访存次数，提高总线带宽利用率。
- **缓冲区过少可能导致溢出（Buffer Overflow）**，写入请求不能被合并，降低优化效果。

优化方法：

- **增加缓冲区深度（Entries）**，提升可存储的写入请求数量，提高合并成功率。
- **结合写回缓存（Write-Back Cache）**，减少不必要的主存访问，提高 WC 效果。
- **动态调整缓冲区大小**，自适应不同应用场景，以平衡吞吐量与延迟。

---

2. 数据访问模式

数据写入的方式决定了 WC 的合并成功率：

- **顺序写入（Sequential Write）**：相邻地址的数据更容易被合并，提高写吞吐量。
- **非连续写入（Random Write）**：由于地址不连续，合并失败的概率更高，可能导致更多的主存写入。

优化方法：

- **采用数据对齐（Alignment）**，减少跨行写入，提高合并成功率。
- **调整软件访问模式**，尽量使用批量写入（Batching），避免小块随机写入。

---

3. 缓冲区管理策略

写缓冲区的管理方式会影响 WC 的性能，主要包括：

- **FIFO（先进先出）**：先写入的请求先被处理，适用于简单的写合并情况。
- **LRU（最近最少使用）**：减少缓冲区数据被过早写入，提高合并的可能性。
- **Tag Matching（地址匹配）**：判断新写入请求是否可以与现有缓冲区中的数据合并，提高合并效率。

优化方法：

- **提高缓冲区命中率**，如根据数据访问模式调整替换策略。

---

4. 内存一致性和同步机制

在多核或多处理器系统中，写合并必须与缓存一致性协议（如 MESI、MOESI）协调，否则可能导致数据不一致。影响因素包括：

- **内存屏障（Memory Barrier）**：如 `mfence`、`sfence` 等，确保写操作按预期顺序执行。
- **写入排序（Write Ordering）**：某些架构默认支持乱序写入，可能导致数据被重新排列，影响 WC 的正确性。

优化方法：

- **确保关键数据写入时使用适当的同步机制**，避免合并带来的顺序问题。

6. 未来发展

未来，写合并策略可能会结合人工智能优化，如通过机器学习模型预测访存行为，进一步提升写合并的效率。

有关缓冲区数量对合并写缓冲的影响的实验验证

缓冲区数量是有限的，这意味着，在一个循环中，你不应该同时写超过缓冲区数量的不同的内存位置，否则你将不能享受到合并写（write combining）的好处。

该代码对不超过缓冲区数量的不同内存位置进行写操作，runCaseOne()是同时对6个不同内存位置进行写操作，runCaseTwo()是分成两次循环分别对3个不同内存位置进行写操作，由于此时写入的不同内存位置数量小于缓存区数量，所以runCaseOne()和runCaseTwo()都享受到了合并写的好处，它们都是在缓冲区被写满后再一起写回内存中，减少访存延迟，但runCaseOne()是同时进行写操作，runCaseTwo()是分开进行写操作，所以runCaseTwo()所需的时间会更长一些

```
public final class WriteCombining2 {

    private static final int ITERATIONS = Integer.MAX_VALUE;
    private static final int ITEMS = 1 << 24;
    private static final int MASK = ITEMS - 1;

    private static final byte[] arrayA = new byte[ITEMS];
    private static final byte[] arrayB = new byte[ITEMS];
    private static final byte[] arrayC = new byte[ITEMS];
    private static final byte[] arrayD = new byte[ITEMS];
    private static final byte[] arrayE = new byte[ITEMS];
    private static final byte[] arrayF = new byte[ITEMS];

    public static void main(final String[] args) {
        for (int i = 1; i <= 3; i++) {
            System.out.println(i + " WriteCombining duration (ns) = " +
runCaseOne());
            System.out.println(i + " WriteCombining duration (ns) = " +
runCaseTwo());
        }
    }

    public static long runCaseOne() {// write to all arrays in one loop
        long start = System.nanoTime();
        int i = ITERATIONS;

        while (--i != 0) {
            int slot = i & MASK;
            byte b = (byte) i;
            arrayA[slot] = b;
            arrayB[slot] = b;
            arrayC[slot] = b;
```

```
            arrayD[slot] = b;
            arrayE[slot] = b;
            arrayF[slot] = b;
        }
        return System.nanoTime() - start;
    }

    public static long runCaseTwo() {// write to all arrays in two loops
        long start = System.nanoTime();
        int i = ITERATIONS;
        while (--i != 0) {
            int slot = i & MASK;
            byte b = (byte) i;
            arrayA[slot] = b;
            arrayB[slot] = b;
            arrayC[slot] = b;
        }
        i = ITERATIONS;
        while (--i != 0) {
            int slot = i & MASK;
            byte b = (byte) i;
            arrayD[slot] = b;
            arrayE[slot] = b;
            arrayF[slot] = b;
        }
        return System.nanoTime() - start;
    }
}
```

```
1 WriteCombining duration (ns) = 5108826200
1 WriteCombining duration (ns) = 7663301300
2 WriteCombining duration (ns) = 4621979600
2 WriteCombining duration (ns) = 7135555100
3 WriteCombining duration (ns) = 4969624800
3 WriteCombining duration (ns) = 7617525200
```

该代码对超过缓冲区数量的不同内存位置进行写操作，runCaseOne()是同时对12个不同内存位置进行写操作，runCaseTwo()是分成两次循环分别对6个不同内存位置进行写操作，由于runCaseOne()此时写入的不同内存位置数量大于缓存区数量，所以，每次循环都需要访问内存，将缓冲区清空后，缓冲区才有位置容纳其它要写入的数组，这样就导致runCaseOne()没有享受到合并写的好处。可以看到runCaseOne()测得的时间远高于runCaseTwo()。

```
public final class WriteCombining1 {

    private static final int ITERATIONS = Integer.MAX_VALUE;
    private static final int ITEMS = 1 << 24;
    private static final int MASK = ITEMS - 1;

    private static final byte[] arrayA = new byte[ITEMS];
    private static final byte[] arrayB = new byte[ITEMS];
    private static final byte[] arrayC = new byte[ITEMS];
    private static final byte[] arrayD = new byte[ITEMS];
    private static final byte[] arrayE = new byte[ITEMS];
```

```
    private static final byte[] arrayF = new byte[ITEMS];
    private static final byte[] arrayG = new byte[ITEMS];
    private static final byte[] arrayH = new byte[ITEMS];
    private static final byte[] arrayI = new byte[ITEMS];
    private static final byte[] arrayJ = new byte[ITEMS];
    private static final byte[] arrayK = new byte[ITEMS];
    private static final byte[] arrayL = new byte[ITEMS];

    public static void main(final String[] args) {
        for (int i = 1; i <= 3; i++) {
            System.out.println(i + " WriteCombining duration (ns) = " +
runCaseOne());
            System.out.println(i + " WriteCombining duration (ns) = " +
runCaseTwo());
        }
    }

    public static long runCaseOne() {// write to all arrays in one loop
        long start = System.nanoTime();
        int i = ITERATIONS;

        while (--i != 0) {
            int slot = i & MASK;
            byte b = (byte) i;
            arrayA[slot] = b;
            arrayB[slot] = b;
            arrayC[slot] = b;
            arrayD[slot] = b;
            arrayE[slot] = b;
            arrayF[slot] = b;
            arrayG[slot] = b;
            arrayH[slot] = b;
            arrayI[slot] = b;
            arrayJ[slot] = b;
            arrayK[slot] = b;
            arrayL[slot] = b;
        }
        return System.nanoTime() - start;
    }

    public static long runCaseTwo() {// write to all arrays in two loops
        long start = System.nanoTime();
        int i = ITERATIONS;
        while (--i != 0) {
            int slot = i & MASK;
            byte b = (byte) i;
            arrayA[slot] = b;
            arrayB[slot] = b;
            arrayC[slot] = b;
            arrayD[slot] = b;
            arrayE[slot] = b;
            arrayF[slot] = b;
        }
        i = ITERATIONS;
        while (--i != 0) {
            int slot = i & MASK;
```

```
            byte b = (byte) i;
            arrayG[slot] = b;
            arrayH[slot] = b;
            arrayI[slot] = b;
            arrayJ[slot] = b;
            arrayK[slot] = b;
            arrayL[slot] = b;
        }
        return System.nanoTime() - start;
    }
}
```

```
1 WriteCombining duration (ns) = 84363019400
1 WriteCombining duration (ns) = 10061744000
2 WriteCombining duration (ns) = 84618938300
2 WriteCombining duration (ns) = 9021899100
3 WriteCombining duration (ns) = 81645469700
3 WriteCombining duration (ns) = 10040959900
```