这个 if 语句块的作用是将每一个线程块中归约的结果从共享内存 s_y[0] 复制到全局内存 d_y[bid]。为了将不同线程块的部分和 s_y[0] 累加起来，存放到一个全局内存地址，我们尝试将上述代码改写如下：

```
if (tid == 0)
{
    d_y[0] += s_y[0];
}
```

遗憾的是，该语句不能实现我们的目的。该语句在每一个线程块的第 0 号线程都会被执行，但是它们执行的次序是不确定的。在每一个线程中，该语句其实可以分解为两个操作：首先，从 d_y[0] 中取数据并与 s_y[0] 相加；然后，将结果写入 d_y[0]。不管次序如何，只有当一个线程的“读-写”操作不被其他线程干扰时，才能得到正确的结果。如果一个线程还未将结果写入 d_y[0]，另一个线程就读取了 d_y[0]，那么这两个线程读取的 d_y[0] 就是一样的，这必将导致错误的结果。考虑 bid 为 0 和 1 的两个线程块中的 0 号线程，它们都将执行如下计算：

```
d_y[0] += s_y[0];
```

假如 bid 为 0 的线程先读取 d_y[0] 的值，然后计算 d_y[0] + s_y[0]，得到了一个数，但是当它还没有来得及将结果写入 d_y[0] 时，bid 为 1 的线程也读取了 d_y[0] 的值。无论是哪个线程先将计算结果写入 d_y[0]，d_y[0] 的值都不是正确的。要得到所有线程块中的 s_y[0] 的和，必须使用原子函数，其用法如下（见本章程序 reduce.cu）：

```
if (tid == 0)
{
    atomicAdd(&d_y[0], s_y[0]);
}
```

原子函数 atomicAdd(address, val) 的第一个参数是待累加变量的地址 address，第二个参数是累加的值 val。该函数的作用是先将地址 address 中的旧值 old 读出，计算 old + val，然后将计算的值存入地址 address。这些操作在一次原子事务（atomic transaction）中完成，不会被别的线程中的原子操作所干扰。原子函数不能保证各个线程的执行具有特定的次序，但是能够保证每个线程的操作一气呵成，不被其他线程干扰，所以能够保证得到正确的结果。注意：这里的 if 语句可保证每个线程块的数据 s_y[0] 只累加一次。去掉这条 if 语句将使得最终输出的结果放大 128 倍。另外，要注意的是，原子函数 atomicAdd() 的第一个参数是待累加变量的指针，所以可以将 &d_y[0] 写成 d_y。

最后要注意的是，在调用该版本的核函数之前，必须将 d_y[0] 的值初始化为 0。在本章程序 reduce.cu 中，我们用如 Listing 9.1 所示的“包装函数”对核函数