space for our array of spheres. The other change has been highlighted in the listing:

```
    HANDLE_ERROR( cudaMemcpyToSymbol( s, temp_s,
                                      sizeof(Sphere) * SPHERES ) );
```

We use this special version of `cudaMemcpy()` when we copy from host memory to constant memory on the GPU. The only differences between `cudaMemcpyToSymbol()` and `cudaMemcpy()` using `cudaMemcpyHostToDevice` are that `cudaMemcpyToSymbol()` copies to constant memory and `cudaMemcpy()` copies to global memory.

Outside the `__constant__` modifier and the two changes to `main()`, the versions with and without constant memory are identical.

6.2.4 PERFORMANCE WITH CONSTANT MEMORY

Declaring memory as `__constant__` constrains our usage to be read-only. In taking on this constraint, we expect to get something in return. As we previously mentioned, reading from constant memory can conserve memory bandwidth when compared to reading the same data from global memory. There are two reasons why reading from the 64KB of constant memory can save bandwidth over standard reads of global memory:

- A single read from constant memory can be broadcast to other "nearby" threads, effectively saving up to 15 reads.
- Constant memory is cached, so consecutive reads of the same address will not incur any additional memory traffic.

What do we mean by the word *nearby*? To answer this question, we will need to explain the concept of a *warp*. For those readers who are more familiar with *Star Trek* than with weaving, a warp in this context has nothing to do with the speed of travel through space. In the world of weaving, a warp refers to the group of *threads* being woven together into fabric. In the CUDA Architecture, a *warp* refers to a collection of 32 threads that are "woven together" and get executed in lockstep. At every line in your program, each thread in a warp executes the same instruction on different data.

When it comes to handling constant memory, NVIDIA hardware can broadcast a single memory read to each half-warp. A half-warp—not nearly as creatively named as a warp—is a group of 16 threads: half of a 32-thread warp. If every thread in a half-warp requests data from the same address in constant memory, your GPU will generate only a single read request and subsequently broadcast the data to every thread. If you are reading a lot of data from constant memory, you will generate only 1/16 (roughly 6 percent) of the memory traffic as you would when using global memory.

But the savings don't stop at a 94 percent reduction in bandwidth when reading constant memory! Because we have committed to leaving the memory unchanged, the hardware can aggressively cache the constant data on the GPU. So after the first read from an address in constant memory, other half-warps requesting the same address, and therefore hitting the constant cache, will generate no additional memory traffic.

In the case of our ray tracer, every thread in the launch reads the data corresponding to the first sphere so the thread can test its ray for intersection. After we modify our application to store the spheres in constant memory, the hardware needs to make only a single request for this data. After caching the data, every other thread avoids generating memory traffic as a result of one of the two constant memory benefits:

- It receives the data in a half-warp broadcast.
- It retrieves the data from the constant memory cache.

Unfortunately, there can potentially be a downside to performance when using constant memory. The half-warp broadcast feature is in actuality a double-edged sword. Although it can dramatically accelerate performance when all 16 threads are reading the same address, it actually slows performance to a crawl when all 16 threads read different addresses.

The trade-off to allowing the broadcast of a single read to 16 threads is that the 16 threads are allowed to place only a single read request at a time. For example, if all 16 threads in a half-warp need different data from constant memory, the 16 different reads get serialized, effectively taking 16 times the amount of time to place the request. If they were reading from conventional global memory, the request could be issued at the same time. In this case, reading from constant memory would probably be slower than using global memory.