Multithreading can provide better performance in situations where tasks can be divided into smaller sub-tasks that can be executed concurrently. Here are two programming examples where multithreading can provide better performance than a single-threaded solution:

1. Web servers: A web server needs to handle multiple requests from clients simultaneously. A single-threaded server can only handle one request at a time, which can cause delays and slow down the server's performance. On the other hand, a multithreaded server can handle multiple requests concurrently, improving response times and overall performance. Each thread can handle a separate client request, and the server can spawn multiple threads to handle multiple requests simultaneously.

2. Image processing: Image processing involves performing complex operations on a large amount of data. In a single-threaded solution, the entire image processing pipeline must be executed sequentially, which can be time-consuming. In a multithreaded solution, the processing pipeline can be divided into smaller sub-tasks, and each sub-task can be executed concurrently on a separate thread. For example, one thread can read the input image, another thread can apply a filter, and a third thread can write the output image. By parallelizing the processing pipeline, the overall processing time can be reduced, improving performance.

In both these examples, multithreading allows for concurrent execution of tasks, leading to better performance than a single-threaded solution. However, it's important to note that multithreading introduces its own set of challenges, such as race conditions and deadlocks, which must be carefully managed to ensure correct program behavior.