



Whitepaper

## A Library Based Approach to Threading for Performance

David R. Mackay, Ph.D.

### Introduction

Libraries play an important role in threading software to run faster on Intel multi-core platforms.

Multi-core platforms are the default computer platforms sold throughout the world. On a multi-core platform there are multiple processors contained on a single chip. That is, each chip has multiple cores that are each a full processor. A decade ago only servers and high end workstations contained multiple processors in configurations called symmetric multi-processor (SMP) systems. A multi-core platform offers the same performance benefits a SMP system did years ago. (SMP systems with multiple multi-core chips are still available for high-end workstations and servers). Even laptops contain multi-core processors now. Each core or processor on a multi-core chip can independently complete computational tasks, improving platform performance and enhancing the end user experience.

Frequently it may require significant work for a development team to thread an application. Libraries which are already threaded provide a low development cost method to prepare software to benefit from the performance potential of multi-core platforms. This paper discusses the roles and benefits of libraries in preparing software for multi-core.

In order for software to take advantage of multiple cores, applications must be designed to parcel out work so that each core can do something. Threading is the most common way for a single application to assign tasks that can be run concurrently. Threads are part of the same process and thus are able to share data. Tasks are assigned to each thread, which the OS schedules to run on one of the cores of the processor. Enterprise software written for servers has long been threaded for performance and runs well on multi-core processors. Much of the software written for desktops has been threaded for convenience; that is tasks are assigned to different threads that are not computationally intensive (e.g. I/O). In this case, there is frequently little performance improvement by running these applications on multi-core platforms. If the tasks assigned to these threads are computationally intensive, they may each work independently and concurrently for some period and the overall performance of the application improves. Distributing the work

of the heavy computational tasks across multiple cores provides real benefit to the end user. The benefit is typically visible in one of two ways:

- the application may run faster and complete the work quicker,
- the other improvement might be an improved visual or audio experience, like higher resolution pictures and better rendering in game scenes, or the ability to have more objects simultaneously in motion in a game frame.

## **Libraries and Threading**

Libraries are great tools in software development. Whether they are developed in-house or licensed from third parties, they can play an important role in coding software as well as impacting performance and threading. Libraries are a great way to reuse developed code, which saves development and maintenance costs. By tuning a library for performance once, many applications may benefit. Once developers are familiar with the libraries, they can use them as they work on different projects. There are many things to consider when selecting a library to incorporate into a product. In this paper we limit discussion to factors related to threading.

## **Thread Safety**

A library can be thread safe, threaded for performance, or both thread safe and threaded for performance. We will explain what we mean by both of these terms. Let's take a brief look at each. If a library is thread-safe, multiple threads of an application can simultaneously call functions in the library. The most common problem with making libraries thread-safe is static and global variables. If a library or its functions contain static variables and multiple threads are reading and updating them, you may encounter undefined behaviors. The same is true for global variables, whether external or internal to the library. It may be possible to make a library thread-safe by adding mutexes or critical sections to the code accessing the static or global variables. While sometimes needed, for the sake of performance it is better to minimize the number of mutexes and critical sections in software. Whenever possible it is preferable to make a local copy of data and pass this into the library as a parameter. When each thread has its own local copy, it doesn't need to worry about data race conditions as it completes its tasks.

## **Threading libraries for Performance**

If a library is threaded it means that the library will decompose its tasks into smaller tasks and assign these to multiple threads. Again to do this safely, it might be necessary to use mutexes or critical sections. Typically a library will create its own set of threads for computation. It is possible though, for a library to use a thread pool shared with the main application, but this requires more careful consideration and design. When the library and main application use the same thread pool, there can be better efficiencies. We have seen cases where an ISV development team created threads for each library and ended up creating 80+ threads of which only 4 were active at any one time. A common thread pool can avoid this unnecessary allocation of resources.

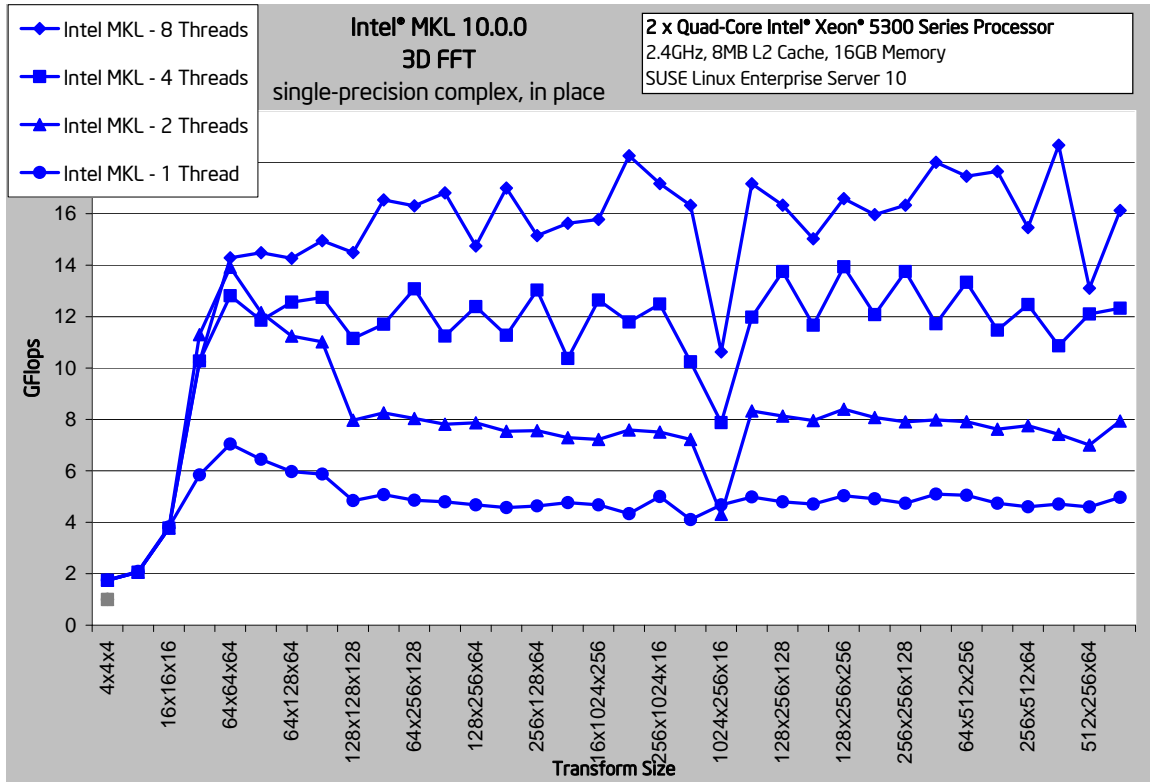
## Threading libraries and Thread-safety considerations

A library that is threaded may or may not be thread-safe. If it is threaded and thread-safe multiple threads may simultaneously call the library and the library may fork off additional threads, which will all run simultaneously. This is called nested threading. If it is threaded, but not thread-safe, only one thread should ever call the library at a time. If the library is not thread-safe and multiple threads call into it, the behavior may be undefined and results will not be deterministic. You should check with the library provider or library development team for documentation on its intended use in threaded environments. Intel® Thread Checker may help check the thread safety of a library.

The library or functions themselves may be thread-safe, but dependent on the pointers passed into them. For example the stringcopy command in the runtime library may be thread-safe, and one can make multiple calls to stringcopy from different threads satisfactorily, But if you make two calls to string copy from two different threads and pass in the same target address to both calls, the result will be nondeterministic, not because the function is not thread-safe, but because it was not called correctly.

## Examples of threaded libraries uses

The Intel® Math Kernel libraries (MKL) are threaded and thread-safe. Graph 1 below shows how well MKL scales as the number of threads increases from 1 to 8 threads. MKL incorporates several different APIs to the 3D FFT routine making it easy for developers to adopt. The graph below shows that it is already optimized and this will improve the performance of analytical code requiring a 3D FFT.



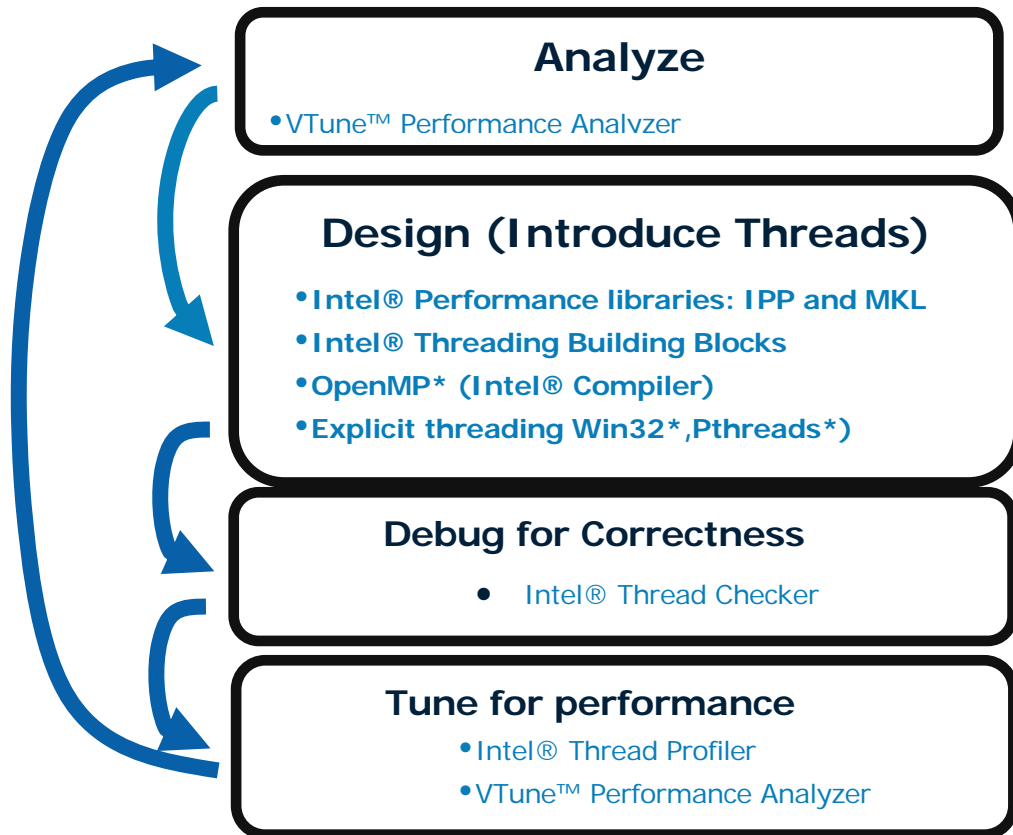
## Creating a threaded or thread-safe library

Those who choose to create their own threaded or thread-safe library in-house would do well to consider the steps we have outlined many times for developing threaded software as shown in Figure xx. There are 4 distinct phases:

- Analysis
- Selection and Implementation
- Correctness
- Tuning.

We will discuss each step briefly before jumping into a discussion about threads and libraries.

**Analysis** The first step is analysis. If you are beginning a new project the analysis might be a white board layout or UML definition. You will want to know the workflow and data flow. At this stage consider opportunities for parallelism: what tasks can be done in parallel, what tasks can be decomposed into smaller work chunks for multiple threads to operate on simultaneously. Plan the threading work into the software architecture from the very beginning. This is the best way to achieve good scalable code. If you are working with legacy code as is often the case it is important to begin with good understanding of its behavior. Even if you think you understand the behavior, it is valuable to verify that with data. We recommend VTune™ Performance Analyzer to understand performance bottlenecks and data collection to estimate potential gains from threading a particular library or region.



**Selection and Implementation:** After you have completed the analysis and have an idea of a region for which you want to improve performance, you need to select the best threading model. If you choose to make a library thread-safe, you may skip the selection of the threading model. If you choose to thread the library, there are several threading options. We at Intel highly recommend using a threading abstraction that allows the developer to identify parallel tasks and allows a runtime library to manage the threads. The two we recommend most highly are Intel® Threading Building Blocks and OpenMP\*. Intel® Threading Building Blocks, is a C++ template based approach to threading for performance. OpenMP\* is a pragma based approach to threading software (OpenMP\* is included as part of the Intel compilers and many other popular compilers as well). The last and most work intensive method is to use raw threads - Windows\* or POSIX\* threads. This last choice is the most work intensive and most prone to error. The references section of this document contains links to seminars that go into more depth about threading, debugging and tuning.

**Correctness:** Regardless of whether you chose to thread the library or to make it thread-safe, Intel® Thread Checker is a nearly essential product for checking for common threading errors such as race conditions. If the library is intended to be thread-safe, you will need to make a simple driver program that will have several threads call the library simultaneously. If you intend to thread the library you will want to use Thread Checker with the program using a workload that exercises that library. Intel®, Thread Checker will report the line numbers of the source code where the data race occurs. Intel® Thread Checker can also be used to check for thread safety of libraries you

obtain from other groups. Create a driver program where multiple calls to the library are made from different threads and run this through Intel® Thread Checker. When source is not available, it reports disassembly and unwinds the function call stack.

**Tuning:** After the application is running well, you will want to tune the threaded library, or the code calling the thread-safe library. Intel® Thread Profiler is a performance utility that helps developers identify work load imbalances and excessive synchronization calls. These are both items that can negatively impact performance. A little bit of investigation often yields good performance improvements.

## Summary

Multi-core platforms are increasing and the number of cores on these platforms is increasing rapidly too. Software that is threaded for performance will have an advantage over competitors' software that is not threaded for performance. Developing or licensing libraries that are threaded for performance or that are thread-safe and can be re-used across product lines is an efficient way to take advantage of multiple cores on modern platforms.

AutoCAD 2007 is only one such application already doing this and it is providing good performance for its customers. Developers should prepare their software for multi-core and libraries are a great way to do it. It can accelerate the adoption of threads in a software development cycle and deliver increased performance to your product. Intel Software Products help developers identify regions to thread, offer good threading abstraction models and help identify and eliminate data race conditions and performance bottlenecks. Thirty-day evaluations of all tools are available at <http://intel.com/software/products>.

## About the Author



David Mackay completed his Ph.D. from Stanford University in 1992. He joined Intel's Supercomputer Systems Division in 1992 where he worked on parallel programming and tuning. He managed workstation application tuning team before joining Intel's software developers division. David manages a team of consulting engineers responsible for performance and threading analysis tools as well support.

## [Related Links]

Kirkegaard, Knud J. et. al, "Methodology, Tools and Techniques to Parallelize Large-Scale Applications: A Case Study". Intel Technology Journal Volume 11, Issue 04 also available at (<http://www.intel.com/technology/itj/2007/v11i4/6-tools/1-abstract.htm>)

Threading Methodology: Principles and Practices, Version 2.0 ([http://cache-www.intel.com/cd/00/00/09/28/92869\\_92869.pdf](http://cache-www.intel.com/cd/00/00/09/28/92869_92869.pdf))