UNLOCKING THE POWER OF PARALLEL CODING TO ACCESS BETTER PERFORMANCE IN MULTI-CORE ENVIRONMENTS

ParaFormance™ explains how it provides easy access to the relevant technologies, skills and knowledge that delivers substantial performance improvements through parallel programming and provides additional benefits of saving energy in the process

ParaFormance™

UNLOCKING THE POWER OF PARALLEL CODING TO ACCESS BETTER PERFORMANCE IN MULTI-CORE ENVIRONMENTS

Gain fast and easy access to performance increases for your application from parallel programming that exploits todays multi-core processor technology without increasing energy consumption

Background

When data-intensive applications become CPU-bound, many development departments will recommend increasing processor power. Unfortunately, even where money and energy use is no object, there are physical limits on the speed that can be achieved by a single processor. Luckily, in many cases, the performance of an application can be increased using relatively cheap off-the-shelf multi-core hardware (where a processor has several individual CPU "cores"), and without impacting energy consumption. In this way, several processors can be used at the same time to speed up the application.

The majority of legacy data-intensive applications use a single thread of execution.

Rewriting the application by introducing threads, or using simple compiler switches, often fails to deliver the expected performance gains.

It is necessary to refactor the application so that it exploits parallelism. At this point in time, this is usually a difficult, time-consuming and expensive process that requires scarce specialist developer skills.

What is parallel coding?

Introducing parallelism into software can either be done automatically by a compiler, or else by introducing new code that is written by developers ("parallel coding"). Parallel coding is the process of taking existing code and modifying it to run in parallel. This is typically done by identifying in the code the places where many calculations or processor spurs occur. These calculations typically dominate a large part of the execution time, and are always executed in order one after other, despite how many cores the processor has. Parallel coding modifies these calculations so that they can be executed simultaneously on the different cores of the processor up to the maximum number of cores that are available. In the best case, performance increases are precisely in line with the number of cores, without significantly increasing the total energy usage, or hardware cost.

Compiler vs coding – is one better than the other?

While it is easy to simply change some compile-time settings to enable parallelism using a parallel-enabling compiler, the developer has no visibility or control of what is being done other than through setting flags when the code is compiled. Furthermore, using compiler flags means the programmer usually only has the option of one type of parallelism technique, and this may not be the most efficient method for the task to be performed, or take into account the underlying business logic of the application. In many cases, there may not be any performance improvement, and there could even be worse performance. Parallelizing the application code is normally the preferable option.

A typical large application may have hundreds or thousands of sites that could benefit from parallelism, and each of these could be parallelized using dozens of different techniques. Having identified the sections of code where there might be a performance gain from parallelism, the developer must then decide on the best method to improve performance, try these out and then revise their software. This may require changing parts of the application over which they have no knowledge or control. Worse, the software may no longer work properly. This is because a parallelized piece of code is non-deterministic, meaning that there is a chance that results of parts of any calculation (variables, arrays and pointers) may not be updated in the same order each time the routine is called. This is commonly known as a race condition. Dealing with all these issues requires significant programming skill, knowledge and expertise. These skills are now in high demand and short supply.

Why is it so challenging?

While basic parallel coding isn't always that difficult, it does need a different mindset. Moreover, it does become rapidly more challenging with code complexity, and there are numerous traps for the unwary to fall into. It is not unusual for carefully crafted parallel code to be slower than the original software, as well as containing very nasty bugs. To produce efficient and safe parallel code, the developer must be rigorous about understanding the full complexity of process scheduling and control, and may need to understand details across the entire application. This is hard to do on more than a small scale using the typical threading approaches that are widely used today.

Although it has a history that stretches back to the 1960s, parallel programing is still only taught in principle to undergraduates on some Computer Science courses. The majority of developers have to learn it as an additional practical skill once they reach employment. It has an increasingly important place in today's data hungry, energy and performance conscious application development arena. Although this has created the demand, the underlying skill and knowledge is in short supply.

What is needed is an easy-to-use tool-set that is accessible to all developers, that sits inside their normal IDE or other development environment, that integrates with the usual development workflow, and that allows them to quickly and easily access the full power of parallel programming.

This document gives an overview of the three-stage process that the ParaFormance tool-set follows, and that provides engineers with easy access to real performance improvements through parallel coding. ParaFormance is intuitive, built with non-specialized developers in mind.

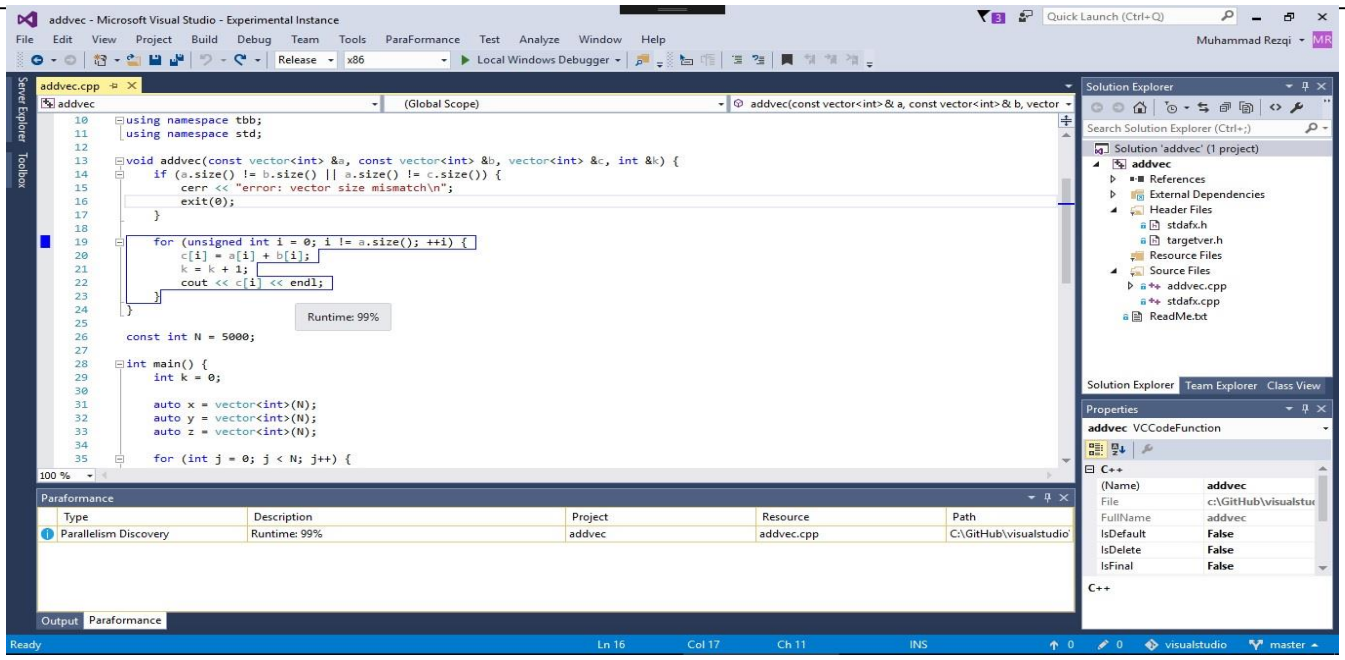

FIGURE 1 - IDENTIFICATION OF CODE THAT WOULD BENEFIT FROM PARALLEL CODING

1. Discovering parallelism opportunities.

In principle this stage of the process is similar to code review, but instead of looking for logic and syntax errors, it's about looking at the existing code to discover opportunities where functions and algorithms that would currently be processed sequentially could be refactored using parallel programming techniques. The benefit of using ParaFormance's tool-set is that it's done automatically and at much higher scan rates than when done by a human.

As an example, we have a customer producing software for railway infrastructure diagnostics who had a 3,500 line project that checks the condition of wheel and axle assemblies on moving trains in real-time. This needed to be sped up so that complete trains could be processed at normal track speeds. Utilizing their own team of five highly specialized developers and parallel coding experts took two full days to review the code. They found three opportunities that were considered suitable for parallel coding. Using the ParaFormance discovery technique on the same 3,500 lines of code, a single person discovered fifteen opportunities that were considered suitable for refactoring. This took just five minutes.

Performance's built-in intelligent heuristics and analytics ensure that it reports only the parts of the application code that will benefit from parallelization.

2. Refactor code to unlock parallel performance

Once the discovery step has been completed and any opportunities for parallelism have been highlighted, the code needs to be refactored to support parallel coding.

A number of different frameworks and standards can be employed for parallel coding. The choice of the most suitable depends on the purpose of the application, its overall requirements and the target execution environment. Selecting the right framework is imperative to obtaining the best possible performance increase. The choice of framework is based on the available memory, overheads, controls and support.

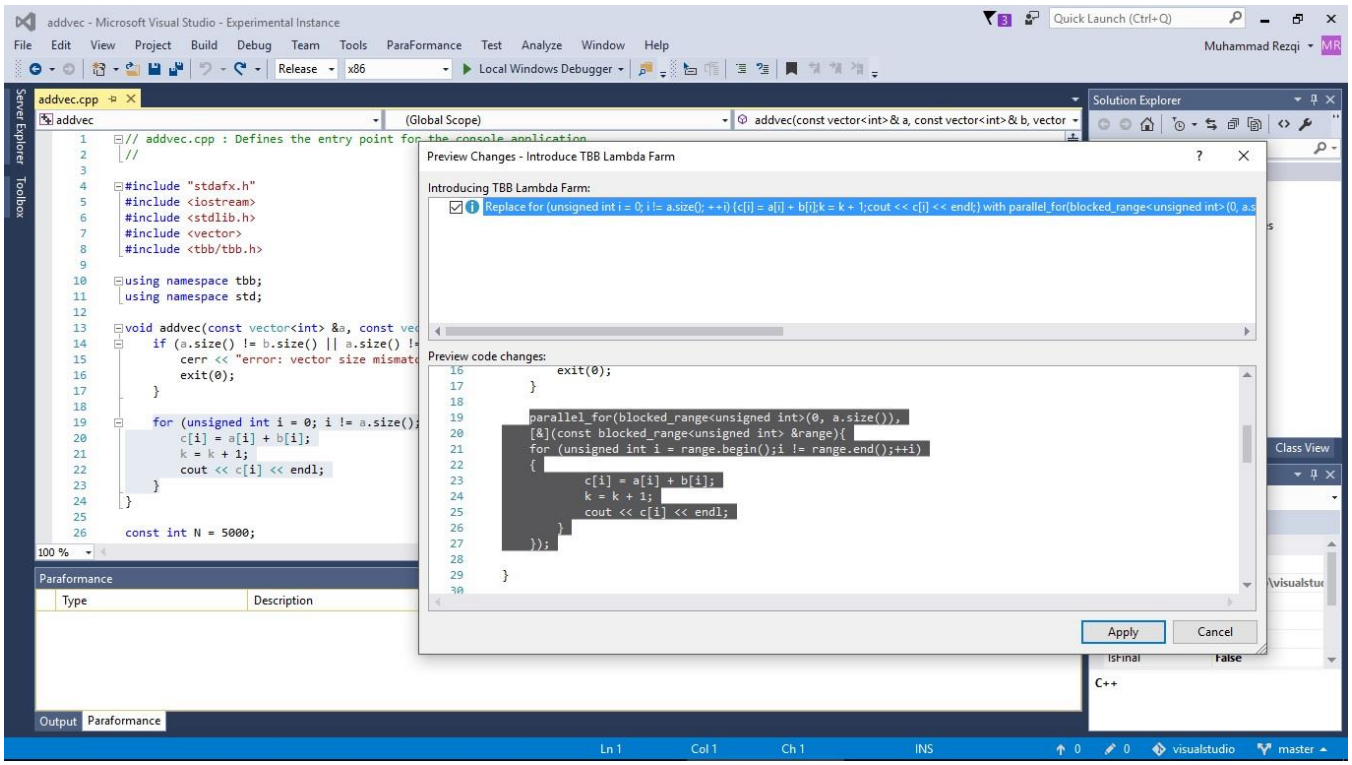

FIGURE 2 - PARAFORMANCE INTEGRATED IN THE IDE ENVIRONMENT SUGGESTING REFACTORING

To give the user the control to make the right choice ParaFormance provides support for three widely-used parallel frameworks: OpenMP, Intel TBB or PThreads. Based on which parallelization framework is selected, ParaFormance will suggest and insert parallel code to replace the existing code. Since the tool is an integral component of the normal IDE (Microsoft Visual Studio or Eclipse), developers can either accept the suggested solution, adapt it, or hand code their own solution.

The table below summarizes the key benefits and considerations of the different frameworks and identifies the best framework for some commonly encountered scenarios.

Open MP	Intel TBB	PThreads
+ Pragma based so parallel code is sequential with added parallelism via C pragmas + Good performance in some cases + Portable and has GPU support	+ Based on OOP and C++ constructs + More patterns than OpenMP including pipeline + Parallel code view embedded in the application for easy maintenance and extensibility	+ Works with C and C++ + Is the foundation for both OpenMP and TBB so has flexibly to support wide range of patterns and parallel algorithms + Portable across different architectures
- Limited pattern support (i.e. no pipelines) - More restricted than TBB, as requires parallel and sequential code to be kept - Pragmas can be very restrictive in most cases requiring significant rewriting of the code.	- Requires manual safety checking to remove race conditions - OOP approach can clutter code in some cases - Not portable and no GPU support	- Thread creation is very expensive. - No load balancing, deadlock control or control and communication, these have to be manually implemented - Patterns have to be manually implemented using thread creation. Very hard to get right.

3. Check thread safety and performance

While it may sound simple to find opportunities for performance improvement and then automatically refactor the existing code, it's imperative that any suggested code insertions consider all the following conditions:

- data dependencies
- variable access
- array access collisions
- memory/pointer collisions
- Object state
- IO/Streams
- STL containers

This ensures thread safety, which means any suggested refactoring has not introduced unpredictable behavior into the code, such as race conditions. Regardless of whether the parallelism has been inserted automatically or has been hand coded, if any race conditions or other important issues are detected, ParaFormance provides hints and suggestions on how to overcome them.

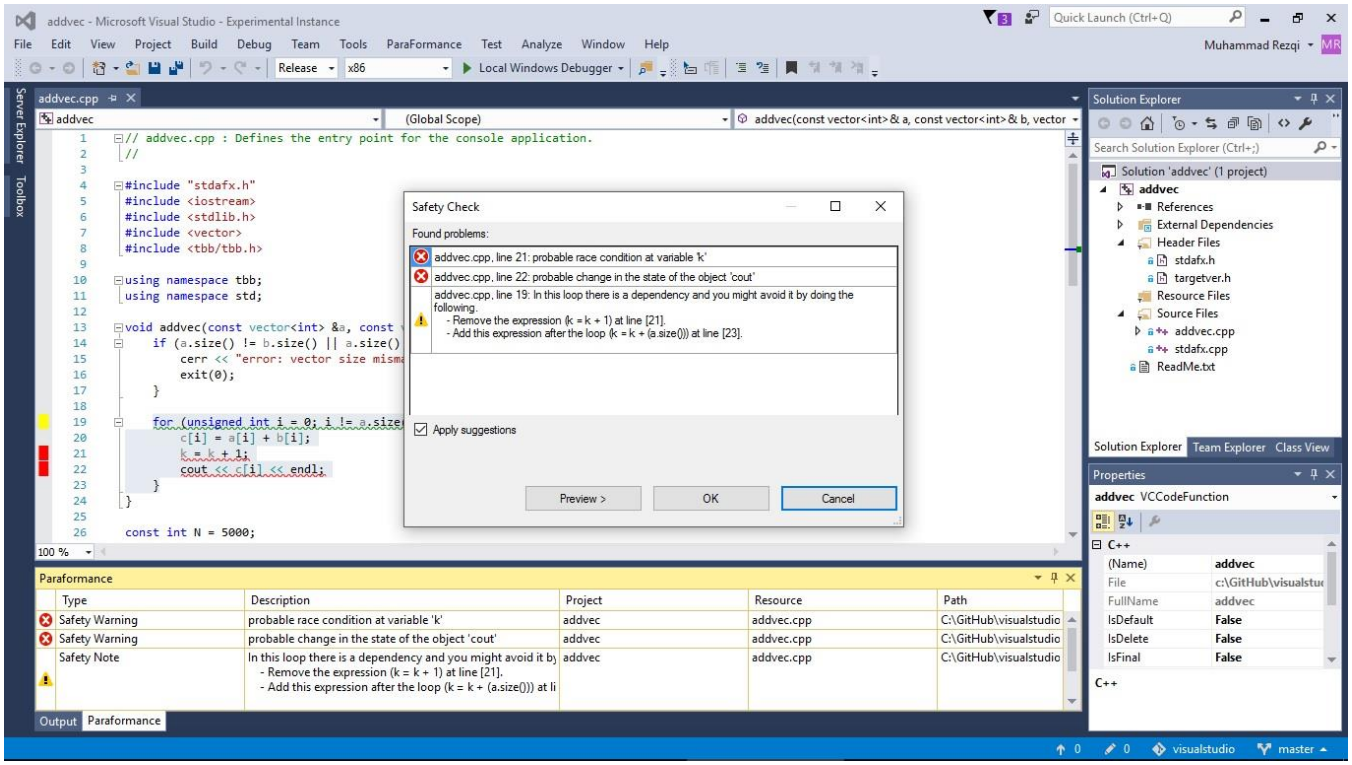

FIGURE 2 - SAFETY CHECKS BEING CARRIED OUT HIGHLIGHTING POTENTIAL SAFETY ISSUES

Since the refactored code only introduces parallel processing capability any existing unit testing, integration testing or regression testing will still be valid, and can ensure that the product meets its functional, code coverage and quality requirements.

The code is then built in the normal way and the performance increase can be checked.

Conclusion

Referring back to the earlier example of the 3,500-line railway infrastructure diagnostic application, having complete all the stages above, our client found that their application could gain a seven times performance increase. This was much better than their own, more laborious, manual implementation.

Using ParaFormance solved the problem quickly and effectively for them. From start to finish, going from a single-core legacy 3,500 line C++ application to one that is optimized and ready for multi-core, took about two to three hours, and that included the time to understand and install the ParaFormance tool.

This process would take normally take a specialist developer around one week of manual effort, and equate to between £2,000 and £3,000 of employment cost, based on typical salaries. Because hidden errors are identified and corrected, the application is safer, too, which will reduce ongoing maintenance and modification costs. The return on investment from using ParaFormance is thus very rapid and represents excellent value. Using the new ParaFormance approach, it is now quick and easy to access any possible performance improvement, and fully utilize multi-core processor technology for both legacy and new applications.

ParaFormance key features:

- **discovers** code that is suited to parallelization
- **checks** that code is thread safe and for code correctness
- **suggests** parallel code for insertion
- **integrates** with standard development environments
- **supports** popular parallel programming libraries (Intel TBB, OpenMP, PThreads and more)

	Support
Operating System:	Microsoft Windows 7, 8, 10, macOS, Linux
Programming Language:	C++
Compilers: supported:	GCC, Clang
IDE environments:	Microsoft Visual Studio, Eclipse
Development Hardware:	Any standard Machine
Target Hardware:	Shared Memory and multi/many core processors

For more information about ParaFormance and to request a free 30-day trial go to www.paraformance.com

ParaFormance™