CITO Research
Tell Us a Question.

JULY 2010

The Multi-core Dilemma

white paper

Sponsored by

PERVASIVE®

Contents

CITO Research
Tell Us a Question.

Introduction

Buy a new system, boost performance. From laptops to servers, this idea has been axiomatic. In fact, exasperation with performance often drives hardware purchases.

But what about multi-core processors? These processors, long used on high-end servers, have now moved into the PC server and desktop system space. The problem is that corporate buyers at small to medium sized companies will expect the tried-and-true rule—buy new hardware, boost performance—to hold once again. However, the interaction between application performance and multi-core processors is far more complex, affecting numerous layers of the application and operating system stack. For some business applications, multi-core processors do not boost performance; worse, such processors actually slow down certain classes of applications.

CITO Research has become aware of this problem and has examined ways to address it. This white paper explains how multi-core processors negatively impact performance of certain business applications and what options are available to developers of such applications to mitigate this problem.

Topics covered in this white paper include:

- What are multi-core processors?
- Why do they require an architectural change?
- Why do some applications run slower on multi-core processors?
- What are the characteristics of applications that run slower on multi-core processors?
- What about applications that run faster on multi-core processors?
- What is the definition of a multithreaded and a parallel application?
- Why do multithreaded applications often not run faster on multi-core processors?
- What are the implications of multi-core processors for developers?
- What options are available for developers to help improve performance on multi-core processors?

The Context for This Explainer

Dan Woods, CTO and editor of CITO Research, began writing about the performance impact of multi-core processors and ways to accelerate processing of large amounts of data in his Forbes.com column in 2009. Pervasive is sponsoring this Explainer to raise awareness about the potential performance impact of multi-core processors. See the last section of this paper for Pervasive's analysis of this problem.

CITO Research
Tell Us a Question.

Developers and Hardware: A Historical Perspective

In the beginning, programmers created software using hardware-level instructions, writing directly in assembly code. The architecture of the processor was always in the programmer's mind. Operating systems and high-level languages essentially eased this burden. Developers could write their code in a language such as Basic, COBOL, C, Pascal, or Fortran, compile it, and the compiler would translate the code into machine-readable instructions.

Parallel programming remained a specialized skill set. While programmers who created operating systems needed to make their code parallel, application programmers usually did not need to parallelize most areas of their applications.

Fast forward to the present. The appetite for faster computing is insatiable. Chip developers such as AMD and Intel have pushed the limit on clock speeds (adding more speed would in essence heat chips so much that it would cook them). Adding multiple cores on a single chip became the easiest way to increase computing power. This leads directly to our central question: If increasing clock speeds always helped speed up software, does more cores have the same effect?

Multi-core's Complex Performance Impact

With a multi-core laptop, users find that they may be able to run more programs at once with less delay when switching between them. But for an individual program, a performance boost is far from certain. In order to boost the performance of a single application on multi-core systems, developers will have to write parallelized code that can take advantage of multiple cores simultaneously.

Despite the fact that many applications are multithreaded, the work with threads is often limited to completely separate areas. For example, user interfaces are often threaded to improve usability, but application logic typically consists of a series of serial, sequential operations that are not so easily parallelized. So the first requirement is that some sort of potential for parallelism must exist in the application for any possible performance improvement to occur from the use of multi-core systems. This can be tough to figure out for applications designed and created without any consideration for parallelism.

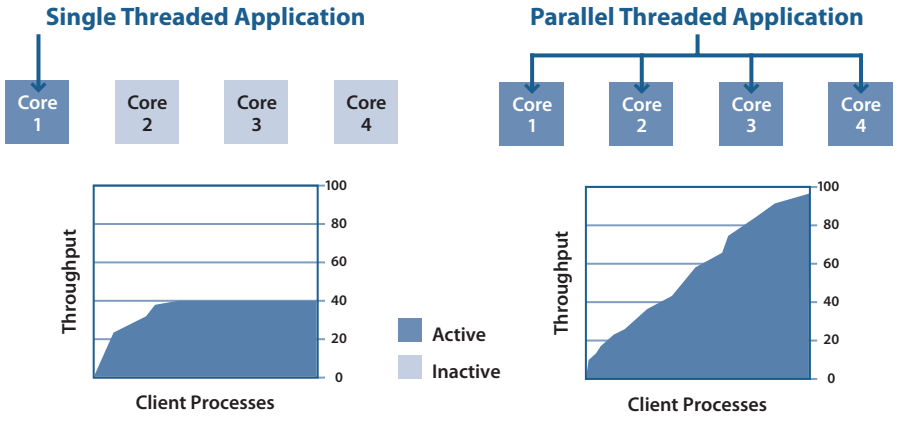

Figure 1: Single Threaded vs. Parallel Threaded Applications

The second barrier is the skill required. Effective programming for multi-core processors is tricky and presents even those developers experienced in multithreaded programming with new challenges.

Application developers are faced with two options:

- Rearchitect their code for concurrency (using toolkits and development environments designed for that purpose)
- Swap out components of the application stack that help address this issue, such as the underlying database

These options are not in fact mutually exclusive. To get maximum performance in a multi-core world, developers will ultimately have to rearchitect their applications to take advantage of these processors. Swapping out an element of the application stack, such as the underlying database, if this option is available, can boost performance and buy developers time to rewrite their applications to take advantage of multi-core hardware.

One of the challenges of analyzing the multi-core dilemma is that general advice is hard to come by. To understand how urgent the need is for refactoring any specific application (or any part thereof), we must understand several dimensions of the application, chip architecture, and operating system:

- Multi-core processors are not identical. The two major manufacturers of chips, Intel and AMD, have different approaches to the design of multi-core processors. The type of processor you select might be affected by the architecture of the multi-core chip. One application might run better on Intel; another might run better on AMD
- Operating system releases vary in their ability to handle concurrent threads (for example, the Windows Server 2008 task scheduler, both in Windows Vista and in Windows Server 2008, represents a significant improvement)
- Applications may run faster or slower on multi-core processors depending on the needs of the application

These dimensions of the problem must be kept in mind as we further lay out the details of the multi-core dilemma.

Where Multi-core Helps Performance

In certain situations, multi-core processing can dramatically boost performance. Think of multiple applications running at once. Simultaneously playing a video game and encoding your latest video project on a desktop system is a handy case where the impact of multi-core processors will be positive and immediate.

Some applications respond well to multi-core processing. If an application's work can naturally be broken up, run in parallel, and aggregated at the end of an operation, that application can benefit from multi-core processing. (Note, however, that the application must be refactored to run in this way, possibly making use of a toolkit that helps in parallelizing the work of the application.) Processor-intensive tasks like video and audio processing, scientific and financial modeling applications, and CAD rendering are examples of such applications. Most high performance and high throughput computing systems take this idea to the extreme and break up a problem such as rendering a computer animation into thousands of chunks that are spread over a huge farm of computers.

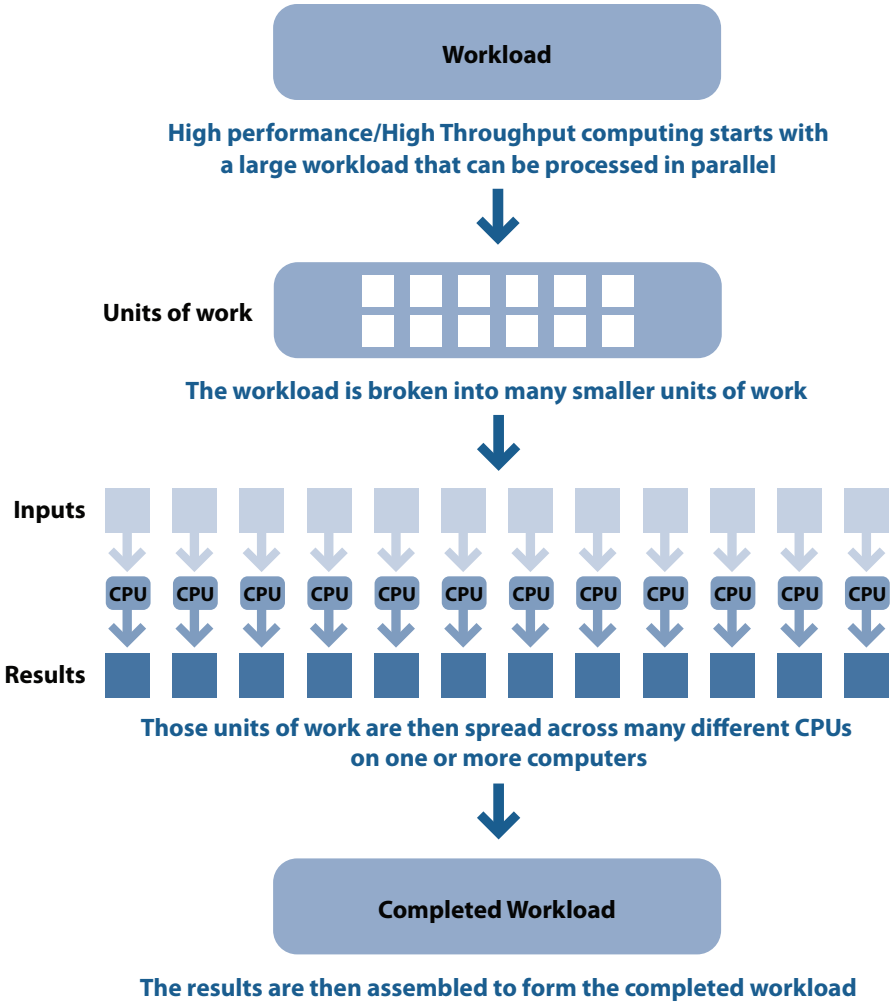

Figure 2: The Basic Structure of High Performance/High Throughput Computing

Where Multi-core May Hurt Performance

What is not clearly understood in the software development community is that multi-core processors can hurt performance when applications:

- Cannot break apart their operations well because much information must be held in memory at once
- Have many users doing the same thing at the same time, such as accessing a database

- Use a database and rely on the state of the data to come up with results
- Share data

In such cases, running on multi-core processors can actually slow down performance.

Examples of the first case, where it is difficult to break apart operations, include dynamic financial models where parameters can be changed by the user during runtime (to slice and dice the view of the model for example) while input is also coming in from real-time data feeds (such as stock or commodity prices).

Web shops (hopefully) have many simultaneous users. During the holiday shopping season, certain items are in high demand. Checking stock for availability, adding the items to the shopping cart, and confirming orders have obvious contention for specific resources that could lead to performance problems and customer dissatisfaction. For such applications, multi-core processing may be problematic.

Databases have many threads and many simultaneous processes whose interdependencies are complex, meaning that some processes must wait for others, making them challenging to parallelize. The state of the data must be carefully managed to ensure transactional integrity. Complex, multiuser applications often include a database. Such applications are vulnerable to a performance slowdown on multi-core processors.

Application Characteristics	Likely Performance on Multi-core without Modification to Application
Several desktop applications running simultaneously	Better performance
Applications that can be broken into multiple, independent processing steps required for completion (gaming, simulation, modeling, rendering). In situations such as these, the data is static and process can run without interruption or updating until complete	Performance is the same or better
Applications that use a database where transactional integrity must be maintained and have many users	Performance degradation
Applications that share data and have many users	Performance degradation

Table 1: Application Characteristics and Performance on Multi-core Processors

Since multi-user applications that share data are by and large business applications, the following is the likely impact:

- Desktop users see a positive impact from multi-core processors
- Small business users see a positive impact from multi-core processors when running productivity applications such as Microsoft Office
- Small business users see their multiclient applications used to run their businesses degrade significantly on multi-core processors

Small to medium businesses will in turn express dissatisfaction to their ISVs and demand help with the problem. An analysis of support calls at ISVs in the past two years has revealed a pattern. Applications that were running well have degraded performance after a multi-core hardware upgrade. In this way, a hardware upgrade on the customer side becomes a support problem on the ISV side.

The Multi-core Challenge

Now that we have discussed the big picture, we clearly understand how, for the first time, a major change in PC hardware is presenting the software industry with a problem. Not only do multi-core processors not make certain applications run faster; they slow some of them down. Development methodologies—as well as debugging and testing—have also changed in fundamental ways. We will now take a closer look at why this slowdown is happening and what can be done about it.

Multi-core also offers an opportunity. Tuned in the right way, software that would be slowed down can take advantage of multi-core processors and experience significant performance improvement. In some cases, swapping out portions of the application stack, such as the database, can address the issue with no immediate changes required from application developers. Developers should view this approach as a low risk way to buy time until tackling the larger task of rewriting applications. Figure 3 shows the three options.

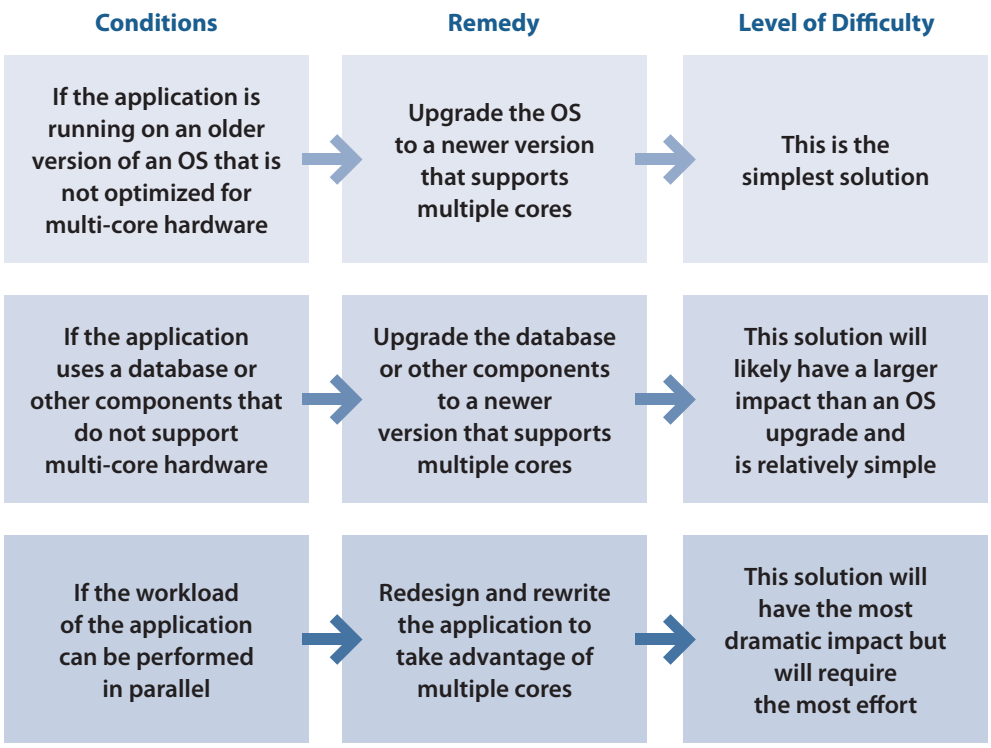

Figure 3: Decision Matrix for Optimizing an Application for Parallel Processing

Why Multithreaded Applications May Slow Down on Multi-core

It may seem counterintuitive at first that multithreaded applications would run slower on multi-core systems. The reasons for this are complex and warrant additional explanation.

In a multithreaded application that shares data, the synchronization between threads consumes a lot of system resources on a multi-core machine. In fact, shared data is the bane of parallel computing. If multiple threads access the same data, access must be synchronized. And when you synchronize access to the data, that portion of code cannot be executed by more than one thread and therefore it is not concurrent. This section of the code then becomes the single door that everyone must line up to go through.

Caching does not improve this problem; it exacerbates it. If multiple cores or processors have caches that point to the same data and one core modifies the data, the cached data on the other core is no longer valid, and the caches must be synchronized.

The accumulated overhead for all the synchronization of these operations is significant; ultimately it means that performance on multiple cores can be worse than on a single core machine that does not require such synchronization.

Where possible, each core should work on separate data; otherwise, there is overhead associated with synchronization and that overhead can slow down performance significantly.

Common Problems in Multithreaded Applications

Other difficulties facing programmers in this transition are age-old problems of parallel programming that manifest on multi-core systems, including:

- The "thundering herd"
- False sharing
- Memory contention

Let's examine these issues.

The Thundering Herd

The thundering herd problem occurs when a scheduler wakes up all the threads that could possibly handle an event (such as a web connection), finds out which one should proceed, and then puts all the others back to sleep when it is determined that it is not their turn to run.

Here's an analogy with a fire station. In a multithreaded scheduler, the fire alarm sounds when the processor has finished a task and is ready to do work. All the threads that are waiting are like fire fighters sleeping in their bunks. The process of waking up all the fire fighters to figure out who should get dressed and into the fire engine is far from efficient. The process of checking who should go next actually prevents work from being done and understandably degrades performance, particularly if there are many threads. The result is sometimes referred to as thrashing.

With multiple cores, the scheduler must deal with even more processors, so the thundering herd problem typically present in a multithreaded application becomes even bigger on multi-core systems.

False Sharing

One of the advantages of multithreaded programming is that multiple tasks can run simultaneously. In order for this to work as it is intended, however, these tasks need to have established prioritization of cache lines as they seek data from memory. Even if the pieces of data are different, the data's location in memory will affect which cache line is chosen. The two threads then begin to contend for the same line of memory, pulling the resource back and forth in a time-consuming struggle, even though from an application standpoint they are looking for different pieces of data. To take advantage of multi-core processing, applications must be written so that cache battles such as this do not occur.

Memory Contention

Programs that are written to take advantage of multiple cores must be aware of the simultaneous processes by which applications retrieve information from memory. Programs that are not multi-core optimized (which, let's face it, includes most programs) access information serially and thus can get into a situation called "memory contention," where the processors repeatedly check the cache to make sure that they have the latest data. The object of this checking procedure is to ensure that a thread on one processor does not execute on outdated data produced by another thread on another processor. While this procedure prevents errors, it slows down the machine considerably as the memory bus places new requests on pause to ensure correctness.

Adding more processors will not solve this problem—in fact, it will make it worse, as more processors place requests to the memory cache. The more caches that must be verified against each other, the more traffic there is. Even without caching, multiple threads attempting to access central memory will get stuck in contention with each other.

Multithreaded applications need to be refactored to break up locks into finer grained locks, spread out the load, and reduce contention among threads for shared resources such as memory.

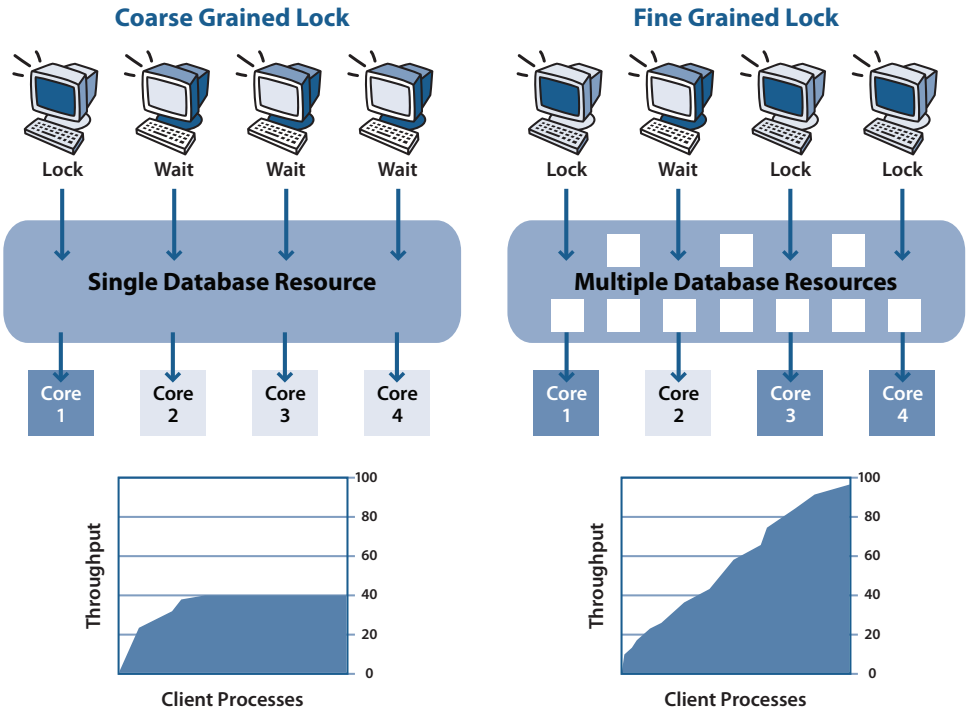

Figure 4: Moving from Coarse to Fine Grained Locks Increases Throughput

The Role of the OS

When contention for resources happens, the operating system handles the resolution of that contention. In the experience of the developers interviewed by CITO Research, operating systems also seem to handle thread contention slower on multi-core systems. In other words, the operating systems on multi-core systems take a longer time to resolve the contention points. This is true even for Windows Server 2008 and Windows Vista. On older Windows operating systems, the slowdown on multi-core systems is even greater.

This punctures the myth of the multi-core operating system. Installing an OS that is optimized for multi-core won't fix your problem if your applications are still asking that OS to perform tasks in a single-file fashion.

Think of your OS as a traffic policeman at a busy intersection. Each vehicle, or thread, wants to proceed immediately, but is looking to the traffic cop to tell it what to do. Imagine that, instead of making an independent assessment about which line of

traffic gets to move next, and waving the line on for a period of time, the traffic cop would instead have to ask each driver waiting behind each stop line whether or not they were ready to go before waving them through. You'd have a worse traffic jam than if there were no traffic cop at all. Essentially, this traffic jam, so to speak, is what happens when an OS gets a request from an application that does not incorporate instructions for multi-core processing. Not surprisingly, a traffic jam at the OS level is perceived by the user as an application performance problem.

A multi-core application provides instructions for the OS on how to manage shared resources and determine priority for access to those resources. This is what many ISVs are referring to when they say their applications feature a "lock-free implementation," meaning that information requests are organized in such a way that they do not compete for cache lines or access to central memory.

Options for Developers

Given all these problems, what is an ISV to do? Ultimately, applications will have to be rearchitected to perform at their highest level on multi-core machines. Toolkits are emerging to help facilitate this, such as Microsoft's Parallel FX library, which includes PLINQ (being incorporated in Visual Studio 2010), Intel Parallel Studio, and Open MPI. These toolkits provide developers with a parallel aware framework on which to rebuild their applications. New languages are also being developed, including Google Go.

The point is that this will take time—in some cases years. Each ISV will have to figure out how to address this challenge to avoid a negative impact on its customers.

In some cases, it is possible to swap out portions of the application stack, such as the database. This offers a simple way to address the multi-core dilemma and give yourself some time to work on making your application take advantage of powerful multi-core systems, which are doubtless here to stay. If you use Pervasive PSQL or another type of data storage that can be easily migrated to PSQL (like ISAM), moving to the multi-core optimized version of PSQL can boost application performance without recompiling or file conversions in most cases.

CITO Research
Tell Us a Question.

About Pervasive PSQL™ v11

Pervasive sponsored this CITO Research Explainer because the company feels that the market does not sufficiently understand this problem and because they have a solution for a certain class of ISVs.

Pervasive's response to the multi-core dilemma has been to focus on optimizing its database system, Pervasive PSQL v11, to take full advantage of multi-core systems.

The latest version of Pervasive PSQL has been designed to provide parallel threads performing similar activities, allowing the database to engage multiple cores during task execution. The result is that multiuser performance of the database engine increases in multi-core environments. Not only is multi-core operation significantly improved, **but database performance increases as cores are added**. In Pervasive's tests and at customer sites, performance gains of 50% over Pervasive PSQL v10 are typical, and improvements of 400% have been seen for some applications.

In brief, you should know the following about Pervasive PSQL v11 and how it helps to address the multi-core dilemma right away:

- PSQL v11 is truly a parallel implementation that makes use of multi-core processors and will accelerate most multiuser applications dramatically.
- Pervasive PSQL v11 includes several enhancements to the low-level synchronization mechanisms in the transactional interface. Multiple users can read the same cached file pages simultaneously and their operations can proceed on independent cores.
- Non-user activity such as checkpoints and log management can also use additional cores.
- Many bottlenecks in the database engine have been removed or diminished. For example, multiple users accessing independent files can proceed on independent cores. The database engine can also handle higher user loads with less overhead, resulting in steadier throughput.
- No recompile is required. Existing Pervasive PSQL applications will run on PSQL v11 without any changes.

Adopting PSQL will improve multiuser performance on multi-core hardware immediately, either preventing a slowdown or providing enough of a performance boost so that a parallel version of the application is not immediately necessary.

Other reasons for PSQL v11 adoption should also be mentioned:

- It is an extremely fast embedded database.
- It offers excellent ISAM performance and disk IO management.

- It is also quite possibly the ideal database for COBOL applications in need of a performance boost.
- It includes a 64-bit SQL engine as well as a 64-bit ODBC and ADO.NET provider.
- It is applicable for all applications based on Pervasive PSQL or Btrieve.

For developers whose applications use Pervasive PSQL as a database, upgrading to PSQL v11 can address the multi-core performance problem by simply swapping out the database layer. This option is also applicable for Btrieve-based applications as well as those written in COBOL and relying on ISAM databases (PSQL at its heart is a very fast ISAM). Upgrading is a great option for ISVs using any of the other database access methods that PSQL supports including ADO.NET, ODBC, JDBC, JCL, PDAC, OLE DB, and ActiveX.

Conclusion

For the first time in the history of computing, new hardware actually slows down applications. Some applications slow down a little, but multiuser applications that deal with shared resources and where state is very important (transactions, for example) can be slowed down significantly.

Parallelizing applications is difficult, time-consuming, and risky. It's hard to do the programming, it's hard to do the debugging, and it's hard to do the testing. Even if your team never misses a deadline on their upgrade cycles, addressing the many thorns on the multi-core rose is likely to change all that. When you have to manage not just conflict between threads, but conflict between threads on multiple cores, each of which has its own group of threads; the problem is amplified. If a multithreaded program has any weaknesses or inefficiencies, those problems will be magnified on multi-core systems. The operating system's management of resources is not adequate to enable most applications to benefit from multiple cores.

However, as we've seen, it is possible to change out portions of the application stack, such as the database. This offers a simple way to address the multi-core dilemma and give yourself some time to work on redesigning your application to take advantage of the powerful multi-core systems that are now de facto hardware upgrades. If you use Pervasive PSQL or use a data store that can be easily migrated to PSQL, moving to the next version of PSQL can boost application performance and help you avoid the multi-core slowdown.