Grand Central Dispatch

A better way to do multicore.

Grand Central Dispatch (GCD) is a revolutionary approach to multicore computing. Woven throughout the fabric of Mac OS X version 10.6 Snow Leopard, GCD combines an easy-to-use programming model with highly efficient system services to radically simplify the code needed to make best use of multiple processors. The technologies in GCD improve the performance, efficiency, and responsiveness of Snow Leopard out of the box, and will deliver even greater benefits as more developers adopt them.

A Software Challenge

Historically, microprocessors gained speed by running at faster and faster clock speeds. Software ran faster as the clock speed increased without programmers having to do any additional work because the software was typically written to do things in consecutive sequence, an operation that could be sped up easily.

Processor clock speeds began to reach a limit because power consumption and heat became problematic, particularly for mobile systems. Because of these issues, CPU vendors shifted their focus from increasing clock speed to putting multiple processor cores into a single CPU, which would use less total energy per unit of computing power.

Although there is more computing power in a multicore processor, software no longer automatically becomes faster. Optimal multicore performance requires that operations be performed in parallel, so most applications need to be significantly rewritten to take full advantage of modern multicore systems.

The dominant model for concurrent programming—threads and locks—is too difficult to be worth the effort for most applications. To write a an efficient application for multicore using threads, you need to:

- Break each logical task down to a single thread
- Lock data that is in danger of being changed by two threads at once
- Build a thread manager to run only as many threads as there are available cores
- Hope that no other applications running on the system are using the processor cores

The problem is far more complex than simply breaking tasks into threads. Too many threads, improper locking of data, or resources "stolen" by other computing tasks can all affect the optimal performance of the software. The complexity of these issues has a well-deserved reputation for introducing bugs that are difficult to find and fix.

Power play
A little reduction in clock speed can go a long way in reducing power consumption. That's because the relationship between clock speed and power consumption isn't linear. The numbers vary with specific processor models and manufacturing processes, but reducing the clock speed of a processor by as little as 20 percent can cut the power consumption of the processor by nearly one-half. And you can add a second core to that processor at the reduced clock speed and nearly double the performance while seeing just a tiny increase in overall power consumption.

The Solution: Grand Central Dispatch

Grand Central Dispatch is a revolutionary, pervasive approach to multicore processing. GCD shifts the responsibility for managing threads and their execution from applications to the operating system. Mac OS X Snow Leopard provides APIs for GCD throughout the system, and uses a highly scalable and efficient runtime mechanism to process work from applications. As a result, programmers can write less code to deal with concurrent operations in their applications, and the system can perform more efficiently.

GCD gives developers a simple way to describe the different pieces of work that your applications need to do, and an easy way to describe how those pieces might be dependent upon one another. Units of work are described as *blocks* in your code, while *queues* are used to organize blocks based on how you believe they need to be executed. By using blocks and queues, you no longer need to worry about threads, thread managers, or locking data, making an application's code easier to understand. You can simply let the system manage the work queues and execute the blocks for optimal performance.

How much faster?
Placing a work item in a GCD queue is a lightweight operation. In fact, it requires only 15 instructions, which makes the blink of an eye seem like a long time. By comparison, setting up a thread and assigning work to it can require hundreds of instructions and take more than 50 times longer.

GCD has a multicore execution engine that reads the queues created by each application and assigns work from the queues to the threads it is managing. GCD manages threads based on the number of cores available and the demands being made at any given point in time by the applications on the system. Because the system is responsible for managing the threads used to execute blocks, the same application code runs efficiently on single-processor machines, large multiprocessor servers, and everything in between.

Without a pervasive approach such as GCD, even the best-written application cannot deliver the best possible performance on any given system, because that single application doesn't have full insight into everything else happening in the system. GCD understands the entire system and automatically maps the blocks of work a you write to individual threads and cores as appropriate. It does this through a combination of the following:

- Blocks, as extensions to C, C++, and Objective-C
- An efficient, scalable runtime engine
- A rich, low-level system API
- A convenient, high-level Cocoa API
- Sophisticated analysis and debugging tools

Only Mac OS X Snow Leopard provides such a seamlessly integrated and pervasive approach. This enables developers to radically simplify their multithreaded code while improving processor utilization, system responsiveness, code readability, and data consistency.

Benefits

Although Grand Central Dispatch was designed primarily to address the challenge of multicore computing, it also delivers a broad range of benefits to both developers and users, including the following:

Improved responsiveness—By making it both easy and efficient to move small chunks of work off the main thread, GCD helps developers make applications more responsive to user input. GCD applications also tend to require less code, CPU time, and memory than traditional mechanisms, enabling the system to run more efficiently.

Dynamic scaling—Optimizing the performance of a traditional multithreaded program typically requires that developers know the details of the hardware on which their applications are executing, and that no other program is running. While these assumptions may be appropriate for a high-performance computing environment, neither is true for desktop applications.

GCD enables you to structure your code in a way that identifies the available opportunities for parallelization, letting the system determine the optimal degree of parallelism with which to execute an application based on the current hardware configuration and the demands of other applications. This allows GCD-based applications—and the entire operating system—to efficiently scale from one to many processors without requiring any manual tuning.

Better processor utilization—GCD makes it very easy for both applications and frameworks to dispatch work they want to run separate from a program's main thread of execution, with the system deciding whether that means another thread or a different CPU. Even with existing applications, Snow Leopard frameworks are better able to distribute work across all available processors. As more applications explicitly adopt GCD, systems running Snow Leopard will have better information and smaller units of work, enabling even more efficient scheduling.

Cleaner code—Most importantly, GCD provides all these benefits while actually reducing the complexity of the code you have to write. Even experienced developers have found that adopting GCD dramatically improves the readability, maintainability, and correctness of their formerly multithreaded source code.

Programming Model in Depth

GCD provides powerful abstractions that enable developers to easily specify what they want done, while the system optimally schedules *how* it gets done.

Blocks

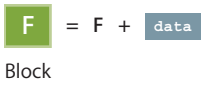
Blocks are a simple extension to C (as well as Objective-C and C++) that make it easy for you to define self-contained units of work. A block in code is denoted by a caret at the beginning of a function. For example, you could declare a block and assign it to x by writing:

```
x = ^{ printf("hello world\n"); }
```

This turns the variable `x` into a way of calling the function so

that calling `x( );` in the code would print the words *hello world*.

What's really powerful about blocks is that they enable you to wrap much more complex functions—as well as their arguments and data—in a way that can be easily passed around in a program, much as a variable can be easily referenced and passed.



Block

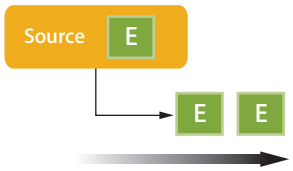
Queues

With Grand Central Dispatch, you schedule blocks for execution by placing them on various system- or user-queues. GCD then uses those queues to describe concurrency, serialization, and callbacks. Queues are lightweight user-space data structures, which generally makes them far more efficient than manually managing threads and locks.

Queue

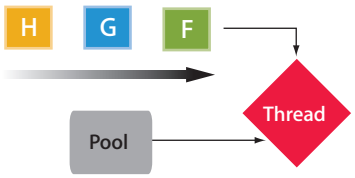
Event sources

In addition to scheduling blocks directly, you can associate a block and queue with an event source such as a timer, network socket, or file descriptor. Every time that source fires, a new copy of the block is added to the queue. This allows rapid response without the expense of polling or "parking a thread" on the event source.



Thread pools

Whenever a queue has blocks to run, GCD removes one block at a time and schedules it on the next available thread from the pool of threads that GCD manages. This saves the cost of creating a new thread for each request, dramatically reducing the latency associated with processing a block. Thread pools are automatically sized by the system to maximize the performance of the applications using GCD while minimizing the number of idle or competing threads.



Runtime Architecture in Depth

Most of the intelligence behind Grand Central Dispatch is provided by queues. As the centerpiece of both serialization and concurrency, queues need to be extremely efficient yet thread safe, so that they can be accessed quickly and safely from any thread.

To achieve this, blocks are added and removed from queues using atomic operations available on modern Intel processors, which are guaranteed to execute completely (without interruption) even in the presence of multiple cores. These are the same primitives used to implement locks; they are inherently safe and fast.

Global queues

GCD has a set of global concurrent queues available to each process. Each queue is associated with a pool of threads operating at a given priority (for example, high, default, or low). The queues are able to monitor resource demand across the entire operating system, not just a single process. This systemwide view is what allows GCD to automatically balance supply and demand for threads across multiple applications.

Whenever a developer enqueues a block on one of the global queues, GCD looks for any available threads at the appropriate priority. If so, it dequeues the block and assigns it to that thread. Conversely, whenever a thread finishes its work and becomes available, GCD checks to see whether there is a pending block on the associated concurrent queue, and dequeues that block based on a first-in/first-out (FIFO) policy. All of this happens as a side effect of queueing and completing work; GCD itself does not require a separate thread.

Private queues

In addition to using the shared concurrent queues, developers can create their own private serial queues within a given process to serialize access to shared data structures. These private queues are sometimes described as "islands of serialization in a sea of concurrency," because the default behavior of GCD is to schedule everything on the concurrent queues.

In fact, serial queues are scheduled using the global queues. Each serial queue has a target queue, which is initially set to the default priority concurrent queue. When a block is first added to an empty serial queue, the queue itself is added to the target queue. Because all operations are atomic, additional blocks can continue to be added to the serial queue.

When its turn comes, the serial queue is dequeued and executed using the same policy and mechanism as individual blocks added directly to the target queue. The only difference is that, in this case, execution (sometimes called "draining") means that each block in the serial queue is dequeued and executed one after the other.

Main queue

Every process has a unique, well-known main queue—always a serial queue—which is associated with the main thread of the program. This is particularly useful for Cocoa applications, because certain operations must always be scheduled on the main thread. The main queue is typically associated with CFRunLoop (for Core Foundation) or NSRunLoop (for Cocoa) on the main thread, both of which drain the main queue at the end of their work cycles.

Developer Tools

The Instruments application—part of the Xcode tools included with Snow Leopard—provides a strong set of capabilities for analyzing GCD usage and performance on multicore hardware.

The Dispatch instrument allows you to visualize your applications' use of GCD, seeing a real-time view and history of block and queue use as the applications run. In addition to this behavioral overview, Dispatch gives you a rich set of analyses, including:

- How many times a block has been executed
- The execution duration of a particular block instance
- The latency associated with a given queue (for example, a measure of how much work was there before each block)

The Dispatch instrument also provides stack backtraces showing both what the application was doing when the block was submitted and what the call stack looked like when it was actually invoked.

Conclusion

Multicore processors are changing the software landscape, requiring developers to reconsider how they write their applications to realize the benefits offered by these new computing powerhouses. Grand Central Dispatch is a new approach to building software for multicore systems, one in which the operating system takes responsibility for the kinds of thread management tasks that traditionally have been the job of application developers. Because it is built into Mac OS X at the most fundamental level, GCD can not only simplify how developers build their code to take advantage of multicore processors, but also deliver better performance and efficiency than traditional approaches such as threads. With GCD, Snow Leopard delivers a new foundation on which Apple and third-party developers can innovate to exploit the enormous power of both the hardware of today and the hardware of tomorrow.

For More Information

For more information about Mac OS X v10.6 Snow Leopard, visit www.apple.com/macosx.