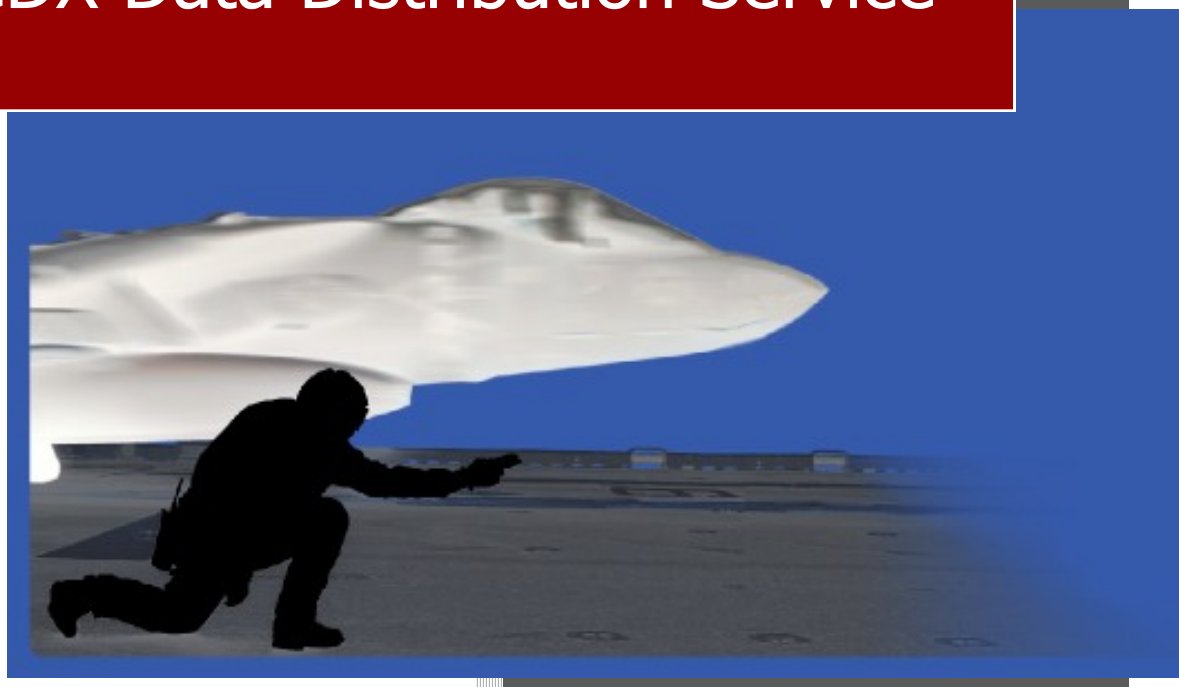
TWINOAKS COMPUTING INC.
PRACTICAL MIDDLEWARE EXPERTISE

White Paper

2009

Exploiting Multi-Core with CoreDX Data Distribution Service



Clark Tucker

Twin Oaks Computing, Inc

755 Maleta Ln, Suite 203
Castle Rock, CO 80108
720-733-7906
www.twinoakscomputing.com

10/19/2009

Contents

Introduction

It has been said that change is the only constant. In the computing world, hunger for more performance is a constant. The increase in multicore processors that squeeze more performance from less power is a constant. And with these constants come the need for applications that can deliver more performance on these advanced processor cores.

CoreDX DDS™ is designed to address this need. CoreDX DDS™ utilizes multicore hardware. Architected from the ground-up to exploit multicore resources, the CoreDX DDS™ middleware provides a clean interface to these parallel and distributed programming environments.

CoreDX DDS™ is a communications middleware. It is an implementation of the Data Distribution Service: a Publish-Subscribe data communications standard managed by the OMG. The DDS standard includes support for Type-Safe application defined data types; dynamic discovery of publishers, subscribers, and topics; and rich Quality of Service policy configuration.

With an internal threading model that is tuned to run on multiple cores and a flexi-

ble programming interface, CoreDX DDS™ offers distinct advantages to system architects. The CoreDX DDS™ API supports different data access models: polling, synchronous event notification, and asynchronous event notification. This flexibility, coupled with robust Quality of Service policy configuration, makes CoreDX DDS™ highly relevant to multicore applications.

This paper explores the application of CoreDX DDS to multicore applications, and presents some design patterns to help exploit those additional cores.

Multicore Challenges

With the growing prevalence of multicore computer systems, software architects are presented with many opportunities for performance improvements; however, parallel-programming environments raise significant complexity challenges. Programming languages and Operating Systems provide some tools to help reduce the complexity of parallel programming; but these fundamental tools are complex in themselves. Developers are adopting middleware, like CoreDX DDS™, because it offers a valuable abstraction layer that insulates them from the complexity of parallel and distributed programming.

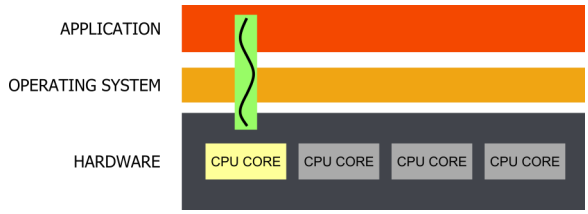


Figure 1 - Single Threaded Application with no Middleware. Additional cores are underutilized.

Previous Options

One approach is to maintain a single threaded application. This application will execute on a single core only, and will be limited by the capabilities of that single core. This approach is sufficient for some applications; however, the power and performance benefits of multicore hardware cannot be realized with these architectures. In the worst case, the extra cores are idle; or, somewhat better, are applied to operating system tasks.

The solution lies with parallel programming. Parallel programming environments have been available for years. There are

many approaches: multi-threading, multi-tasking, pipelining, and symmetric processing. It is possible to integrate these approaches into application software, and many systems have been developed with these techniques. However, these systems are often built around custom, proprietary implementations and are difficult to maintain or extend. For example, it may be necessary to modify the application code as the number of available cores increases.

Application software that supports parallel programming is inevitably complex and fragile, and the increased complexity is often an issue. This complexity arises from the need to address critical section protection, thread synchronization, library thread-safety, synchronous versus asynchronous event notification, processor affinity, cache management, and the debugging challenges presented by multi-threaded systems.

Does a multitasking approach help? In this approach, the data communication tasks are delegated to separate applications or tasks (processes). This may help distribute work across more cores, but the overhead and complexity of synchronization between applications is often greater and can induce significant latency. Therefore, multitasking is not an optimal approach.

Middleware components that are not architected to support multicore programming environments are of little help. These middleware components (see Figure 2) are restricted to a single core, and do not assist the developer with pipelining the data flow.

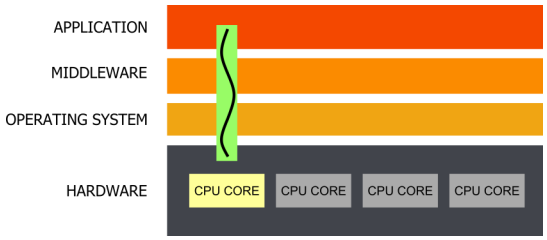


Figure 2 - Single Threaded Application with Single Threaded Middleware. Additional cores are underutilized.

Again, it is left to the application developer to develop multi-threaded application code if he wants to employ additional cores.

Does implementing a multi-threaded application on top of a single-threaded middleware layer help? Again, in many

cases, the answer is no. As shown in Figure 3, the internal synchronization of the single-threaded middleware layer will serialize the process, again resulting in underutilized cores.

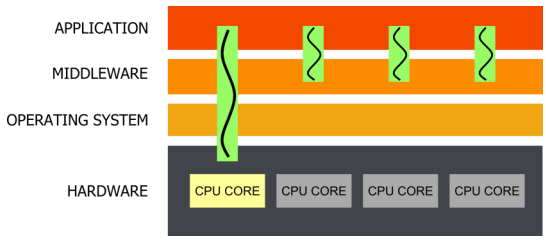


Figure 3 - Multi-Threaded Application with Single Threaded Middleware. Additional cores are still underutilized.

Even a multi-threaded application is serialized to a single core if the middleware technology is not designed to facilitate data pipelining. This is a situation that can be very frustrating to application engineers.

CoreDX DDS™ Solution

The CoreDX DDS middleware simplifies the task of putting those additional cores to work. The engineers at Twin Oaks Computing have worked with multi-threaded and parallel programming environments for over a decade. This expertise has been incorporated into the advanced data pipeline architecture in CoreDX DDS™.

By internally pipelining the flow of data, CoreDX DDS™ can employ multiple processing cores simultaneously. This automatic distribution of work across multiple cores is easy to exploit in application code because of the flexibility of the CoreDX DDS™ API. In particular, the following items are critical for successful multicore optimization:

- non-blocking data publication,
- (a)synchronous event notification,
- arbitrary data access, and
- loose-coupling

With these tools, developers can write application software with a single thread of control, and the CoreDX DDS™ middleware will distribute the communications work across multiple

CoreDX DDS™ extends the power of the Publish-Subscribe paradigm with Multicore support.

cores. This simplifies the application code while employing sophisticated parallel programming technologies.

Non-Blocking Data Publication

With non-blocking data publication, an application can present data to the CoreDX DDS™ middleware for publication, and then continue to perform additional processing. The application thread is not blocked while the middleware performs the distribution of the data. Further, this is an opportunity for the middleware to pipeline the data distribution tasks.

This type of architecture is often critical to data collection systems where the data is subject to bursts. An application is often required to handle data as it is presented from a device, and if the application is blocked while distributing the data to remote consumers, subsequent data samples might be missed.

Synchronous and Asynchronous Event Notification

Application software can be structured to receive event notifications (for example: data arrival) through multiple mechanisms. This flexibility supports both single and multiple threaded application code. The application developer is free to mix these models as necessary to achieve the desired architecture in the application code.

The combination of **WaitSets** and **Conditions** provides a simple mechanism to support single threaded access to middleware events. An application simply blocks on a **WaitSet** until the requested events have been detected.

Support for **Listener** callbacks allows the application developer to expand the threading model of the application and embodies an Event Driven Architecture (EDA). This architecture model is very flexible, and promotes Open Architecture principles.

With CoreDX DDS™, even Single-Threaded Applications can utilize multiple cores.

The unique data pipeline in CoreDX DDS™ makes it easy to leverage modern hardware.

Arbitrary Data Access

With CoreDX DDS™, applications are not required to access data samples in the order that they arrive. The application can choose to access data in a natural manner. There are numerous access patterns available to the developer, for example: ordered by time, grouped by key, only new data samples, and only old data samples. Further, the developer can request that the middleware maintain historical data, relieving the application code from this task.

Multi-Threaded Core

Finally, to complement all of these aspects, CoreDX DDS™ features a multi-threaded core. As shown in Figure 4, the CoreDX DDS™ middleware employs an advanced threading architecture to support data pipelining and event notification.

This technology directly addresses the need of system architects to fully harness multicore processing power. The result is full utilization of available cores, reduced data latency, and simplified application architecture.

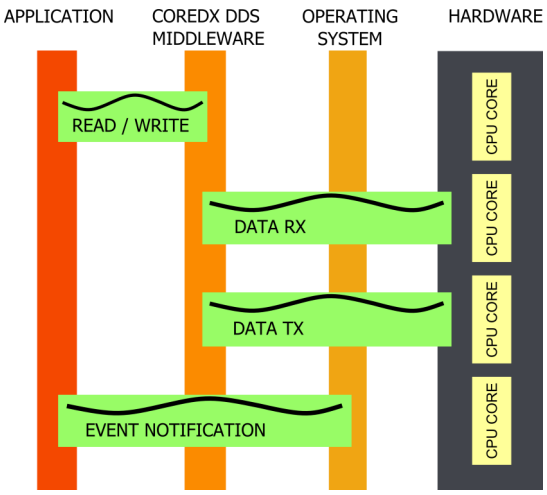


Figure 4 - Multi-Threaded Application with CoreDX DDS Multi-Threaded Middleware. All cores are utilized.

Application Implementation

The application engineer, with the help of CoreDX DDS™, can easily exploit multicore processors to full advantage. Because of the flexible API and advanced multi-threaded architecture of the CoreDX DDS™ middleware, applications can be single or multi-threaded and still utilize multiple cores.

The application can use synchronous or asynchronous event notification, or can poll for events as necessary. The flexible data read calls and the non-blocking write calls provide for flexible data handling patterns.

Finally, the dynamic discovery aspects of the DDS protocol promote Open Architecture goals. The application can be deployed in a dynamic environment, where data consumers and producers appear and disappear over time. The flow of data in these dynamic distributed systems is not predetermined, and can change on the fly. With robust Quality of Service policies, the application engineer has control over the critical aspects of communication, and can guarantee successful data exchange, even in these challenging dynamic environments.

All of these features of CoreDX DDS™ mean that the application engineer has choices in the design of the application architecture. An engineer with choices is a happy engineer.

Summary

In this paper we reviewed the opportunities presented by multicore hardware. We reviewed sub-optimal middleware architectures, and then presented the multicore capable CoreDX DDS™ middleware. This discussion highlighted the benefits of CoreDX DDS in a multicore environment, and identified how applications can easily realize these benefits.

As the demand for performance continues to grow, engineers can rely on CoreDX DDS™ to fully harness multicore processing power. With an easy and flexible programming model, CoreDX DDS™ makes these advanced processor resources accessible while simplifying application software implementation. More power with less work - CoreDX DDS™ is a real winner.

About Twin Oaks Computing

With corporate headquarters located in Castle Rock, Colorado, USA, Twin Oaks Computing is a company dedicated to developing and delivering quality software solutions. We leverage our technical experience and abilities to provide innovative and useful services in the domain of data communications. Founded in 2005, Twin Oaks Computing, Inc delivered the first version of CoreDX DDS in 2008. The next two years saw deliveries to over 100 customers around the world. We continue to provide world class support to these customers while ever expanding.

Contact

Twin Oaks Computing, Inc.

(720) 733-7906

+33 (0)9 62 23 72 20

755 Maleta Lane

Suite 203

Castle Rock, CO. 80108

www.twinoakscomputing.com