

# Parallel Processing with .NET

David Lifka, Lucia Walle, Veaceslav Zaloj, and John Zollweg  
Cornell Theory Center  
Ithaca, New York 14853

## Abstract

*Industry standard-based cluster computing has evolved as the best alternative for high-performance computing for both commercial and research use. A major reason for this popularity is the relatively recent availability of many parallel computing tools, libraries, and scheduling systems that were originally developed and used on proprietary Unix-based MPP and shared memory systems. In a way, industry-standard computing has finally caught up to the capabilities we've been accustomed to for years. So what about the future? From a user perspective the programming techniques have not evolved at all. The great price/performance of commodity computing comes from the volume markets that it leverages. Until users can access HPC resources from their laptops or PDAs without knowing it, HPC will never move from academia and research labs into the mainstream. Microsoft has recently introduced a new architecture called .NET[1] that holds the potential to provide seamless HPC access. .NET provides new tools and techniques to parallel computing that remove the difficult "plumbing" issues that face traditional HPC users. In this paper we explore the use of .NET for inherently parallel codes and compare performance and feature/functionality tradeoffs with traditional techniques.*

## 1 Introduction

There are many scientific and financial algorithms today that exhibit inherent parallelism. Applications in this category divide a problem up over numerous processes, each independently performing the same computation using different input data or parameters. A common programming approach to this problem is often referred to as the "master-worker" model. In the "master-worker" model, the "master" starts individual "workers," each performing the same task on a different set of data or parameters. These

workers are able to complete their computations asynchronously. It is the job of the master to manage the flow of data to workers and to collect the output and present it to the user. Workers are typically started on different processors on the same SMP machine or across machines in a network.

To date, most master-worker codes are written in some flavor of C or Fortran and use a message passing library such as MPI[2] or PVM[3] for interprocess communication between the master and workers. The main problem with this approach is that the programmer must understand all the intricacies of message passing, data management, and most difficult of all, distributed process management and security. Once the application has been written, typically some type of parallel job scheduler must also be mastered by the end user in order to request resources (compute nodes) to run the application.

.NET[1] removes these obstacles, letting the user/programmer leverage an object-oriented approach where each worker is an instantiated object and the master is actually a standard web interface or desktop application that requests these objects seamlessly over a network. A new language, C# [4][5], has been introduced to simplify programming .NET objects. It is the job of the .NET cluster architecture to find the best resource to instantiate the object and to ensure that the requesting user has the appropriate credentials to do so.

In this paper we compare the steps a typical user must take to develop a parallel master-worker application using a traditional C/MPI approach with several different .NET approaches. Our primary focus will be on performance, reliability, tool availability, and development issues.

## 2 Cluster Architectural Comparisons: Traditional HPC vs. .NET

The beauty of a typical .NET cluster configuration, such as Application Center 2000, is that it uses a subset of the industry standard components that are commonly used in HPC cluster configurations today. This means both systems can leverage the economy of scale and ever-increasing performance of the Intel architecture volume markets. The primary differences in these two systems are how they are used and administered.

### 2.1 A Traditional HPC Configuration

Today Beowulf clusters are the most common HPC systems. Although Beowulf clusters are often associated with the Linux operating system, their key feature is not the operating system that they run, but that they are built from commodity hardware. In fact, Beowulfs can be built with Microsoft Windows as their operating system [6]. Typically HPC Clusters are designed like Massively Parallel Processor, MPP, systems. MPP systems are considered to be “Scale-Out.” That is to say they are made up of two or more computers, or nodes, connected by a network. These nodes usually have one or more processors, local disk, memory, and a local operating system and software. The network that connects them can be as simple as an Ethernet or a more sophisticated high-bandwidth, low-latency solution like Myrinet [7] or Infiniband [8].

Users of these systems typically write C, C++ and Fortran codes that use MPI or PVM for interprocess communication. These users also tend to be fairly advanced programmers who understand the parallel aspects of the problem they solve, how to distribute pieces of the problem to different computers, and how to use the message passing library to synchronize data among parallel tasks.

Administrators of these systems must manage these compute nodes, ensuring an integrated, coordinated environment. They must assure that installation and maintenance of the operating system, application software, and hardware are consistent across all nodes and that the nodes are working correctly. Some type of batch scheduling system is commonly used to allocate resources to particular jobs, typically submitted to and maintained in a queue. Resource management and job scheduling is an entire area of research in itself. Trying to determine the most efficient or

fair way to allocate the nodes to the end users is a delicate procedure. These systems can be complex and require effort by users to have a working understanding of how to submit jobs and interact with the system.

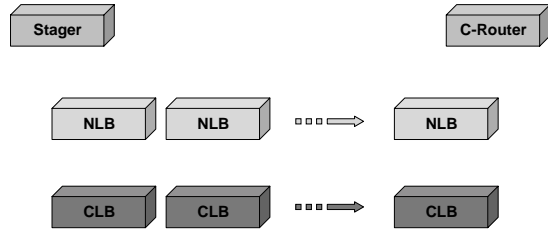
This method of computing is widely accepted in research and academic environments and has been put to use by businesses with sophisticated programming staffs. Its main problem is that it is not easy to use or support and therefore has not been widely adopted by businesses and general consumers. Computer science researchers have been struggling with these issues for years and improvements have been marginal.

### 2.2 A Microsoft Application Center 2000 Configuration

Like the Beowulf style systems, Application Center clusters scale by adding additional servers. Application Center is the scale-out .NET server platform from Microsoft. Presently, Application Center is additional .NET software that is installed on a Windows 2000 Advanced Server. When Microsoft .NET server is released, currently targeted for Q4 - 2002, the necessary software will be in it. *NOTE: Microsoft also provides a “scale-up” solution called Data Center for 8 or more processor SMP systems, but for the sake of comparison with Beowulf style systems we do not address scale-up in this paper.*

The Application Center 2000 .NET architecture has essentially two tiers. The first tier is Web Services. This provides remote access via HTTP, SOAP, and XML to computation and data services that are available on the local .NET cluster. The Web Services tier is also called the Network Load Balancing, NLB, tier. All machines in this tier appear as one virtual Web server accessed via a single URL. Application Center ensures that the load resulting from requests to this URL is balanced across the machines; and in the event of a machine failure, outstanding requests to that machine are rerouted to another on-the-fly. The second tier is the Component Load Balancing tier, CLB. This tier provides a place for computationally or data intensive components to be executed on behalf of a NLB tier request or directly from a Component Router, described later in this section.

Each of these tiers can have 12 machines, but it is possible to have multiple NLB and CLB layers in each tier to manage heavy loads. The only requirements are that all machines in a layer must match the other

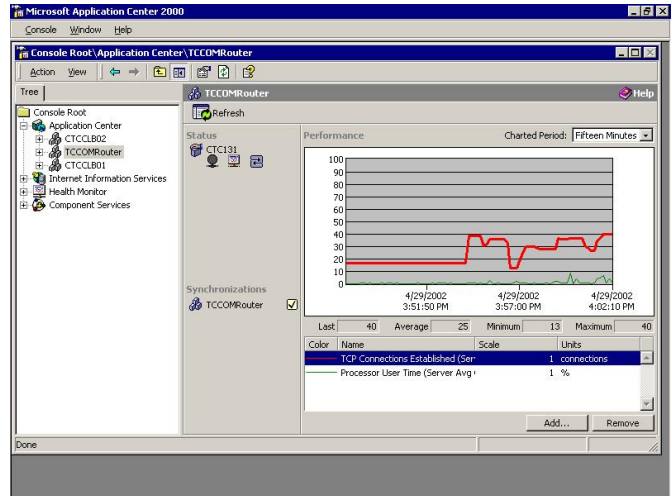


**Figure 1. .NET Architecture**

layers in that tier. For example, all NLB layers must have the same software as other NLB layers in that tier.

NLB and CLB tiers work together to provide a powerful and responsive resource. With the NLB tier, it is obviously important that the Web services are responsive. For this reason computationally or data intensive work should be pushed to the CLB tier. This allows the NLB tier to still have enough compute and network bandwidth to be responsive to the remote client. Figure 1 illustrates a basic Application Center Cluster configuration.

There are two additional machines that serve a special purpose in the Application Center Cluster. The first is the Component Router. This machine is aware of the component services available on the CLB tier and provides access to them for client codes that want to instantiate these services directly instead of through the NLB tier. Since the CLB tier is also load balanced by Application Center, when users instantiate objects via the Component Router, they will automatically get the best machine in the CLB tier for their request. All the end-user must know is the hostname of the Component Router instead of the details of each machine they are allocated, unlike parallel programming in a traditional HPC environment. The second special purpose machine is called the Stager Host. This machine emulates the CLB, NLB, and Component Router of the entire Application Center cluster. It provides a sand-box development environment for programmers so that



**Figure 2. Application Center Management Console**

their codes can be fully tested before deploying them to the actual NLB and CLB tiers. Once systems administrators determine that a given application is working correctly, they can use the Application Center management Console to automatically propagate the components to the appropriate tiers. Figure 2 shows a screen shot of the Application Center Management Console.

Some important features of the Application Center Architecture are that unlike traditional HPC clusters, it is possible to add machines to the cluster on demand without any negative impact on users of the system. Also the machines do not have to have the same hardware configuration. Yet another feature is that a systems administrator can take a machine out of the cluster temporarily to perform upgrades and installations without the users experiencing a systems outage.

From a user's perspective, leveraging .NET components can make computing much easier. The fact that computational or data service components can be reused by multiple applications via standard SOAP and XML means that the amount of code users have to generate themselves is greatly reduced and tends to be focused on the part of their science or business application they know best. In the event that users want to develop their own Web services or components for the CLB tier, Microsoft provides an incredibly rich development environment in Visual Studio .NET [9] that does common interface plumbing automatically.

### 3 Our Experiment

To compare the performance of the different computation options, we wrote a very simple application that computes prices of financial instruments (put and call options) using two algorithms: the Black-Scholes formula and a binomial tree algorithm. Calculations are performed for a range of current asset prices and exercise prices. In every case, the basic unit of computation returns an array of results for a range of exercise prices with other parameters fixed.

#### 3.1 Baseline Test

For reference, the calculations were performed using a master/worker code written in C that communicates using MPI function calls. The code that calculates the option prices was placed in a separate DLL. (For details on the DLL, see below.) This code was run as a batch job on a cluster of machines with the same characteristics (architecture and clock speed) as the Application Center machines. Wallclock times were measured in the master task for the loop in which calculations were assigned to the workers. The times for initializing the computation and for writing the results to files were excluded from this measurement.

Because the .NET approach involves a client on the user's desktop, the results of the computation must be sent over a network from the Application Center to the client. To make an estimate of this communication overhead, the runs of the MPI code were repeated with the master on a desktop workstation and the workers on a cluster in the machine room.

#### 3.2 .NET Approaches

In order to maximally exploit parallelism in an Applications Center cluster, the client side code should make asynchronous calls to the worker methods. The .NET Remoting namespace provides methods for making these asynchronous calls, for determining when they have completed, and for retrieving the results. For these tests, a single Windows Form application was written to instantiate the different COM+ objects used here and to send the necessary commands to perform the calculations. Figure 3 shows the form in use.

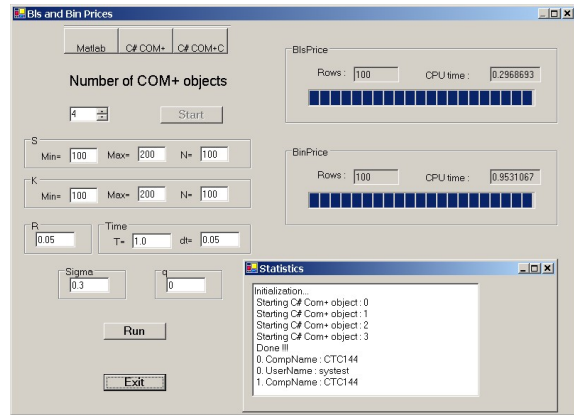


Figure 3. Windows Form for running COM+ objects on the Application Center cluster

#### 3.2.1 COM+ C# Wrapper and unmanaged C DLL

The simplest way to expose functions to external programs is to compile the code into a DLL. The "dll-export" option provides explicit interfaces to be called by any application. These interfaces are used locally; there is no way to directly call these remotely. The .NET runtime InteropServices namespace provides methods that marshal data between managed and unmanaged code. Classes that use these methods are commonly called .NET wrappers. If a .NET wrapper class inherits from the ServicedComponent .NET class it has the characteristics of a COM+ Object. Once the COM+ object is created it can be deployed into the Application Center along with the unmanaged C DLL. For these tests, the same C DLL was used for both the MPI runs and the runs with the COM+ C# wrapper.

This is the fastest approach to moving from the traditional MPI environment to Application Center. The C code requires no changes except to put it into a DLL with the appropriate exported interfaces, so development time is quite short. However, using the P/Invoke method in the C# wrapper code is not robust because the DLL is run in-process. If there is an error in the unmanaged DLL the .NET wrapper will crash leaving only an error message that the unmanaged DLL was the cause of the crash.

#### 3.2.2 COM+/C# DLL

In this case the C code must be rewritten in C#. In the program is a small to medium size application this

is not much trouble because much of the syntax is the same. A significant benefit of using this approach is that the entire application is in one .NET assembly (COM+ DLL). The COM+ object can be used either locally or remotely. By using an Application Center COM Router, the COM+ object can then be used throughout the Application Center cluster.

### 3.2.3 MATLAB Approach

MATLAB®[10] is a powerful computational engine with many modules that can be used to simulate scientific, engineering and financial models. MATLAB provides a few COM interfaces that make it usable as a computation server. Since .NET is capable of using COM objects it makes sense to write .NET wrappers around all of the MATLAB COM interfaces:

- string Execute
- string GetCharArray
- void GetFullMatrix
- void PutFullMatrix

The wrapper .NET class for these objects can be used as a new .NET computational object. The instances of this class will be capable of performing most MATLAB computations on local and/or distributed workstations where the MATLAB engine is installed.

This approach is very flexible because the developer writes his application entirely in the pure .NET environment (using VB.NET and/or C#) and need not be concerned with interoperability issues. The client code manages the MATLAB program as well as input and output data for the computation. They are exchanged with the MATLAB engine through these standard interfaces. On the server side, the computations are performed and temporary variables are kept in the address space of each MATLAB engine. These address spaces are totally separate from the client's address space. The user has the freedom to change the nature of his calculations without dealing with the system administrators.

However there are minuses in this approach due in part to the limited number of COM interfaces that MATLAB provides. Further, these interfaces exchange data for only two types (strings and doubles). This limitation requires that the user make an additional conversion of data on both the client and server sides to deal with other data types (*e.g.* integers, floats).

## 4 Performance Results

The results for the Black-Scholes option pricing algorithm are shown in Figure 4 and for the binary tree pricing algorithm in Figure 5. The numerical results for each approach were virtually the same, but the times required to obtain corresponding results varied widely, so the results have been presented using a logarithmic scale. Runs were done using 2, 4, 8, or 16 workers in addition to the master task.

For approaches in which the computation time was significantly greater than the communication time, the heights of the bars decrease linearly as the number of processes is doubled while keeping the problem size constant. The MATLAB results exhibit this behavior for all problem sizes. When the communication time is significantly greater than the computation time, the heights of the bars don't decrease much when additional tasks are used. This is especially true of the largest problem size for the three approaches where the computations were done in C or C# on a cluster and returned to the master task on a workstation.

The Black-Scholes algorithm is typical of a code that is computationally intensive because it requires evaluation of an error-function complement for each result. but it is not memory intensive. The binary tree algorithm is more memory intensive. Comparing the two approaches that use the C# COM+ object on Application Center, one can see that switching the computations from C to C# adds very little additional time for the Black-Scholes algorithm, but the increase in time for the binary tree algorithm is much more significant.

Comparing the two MPI experiments, one can see that the time required for communicating the data back to the master task is a significant fraction of the total time for the Black-Scholes algorithm, but is not for the binary tree algorithm. Comparing the C#-wrapped C DLL results with those for the MPI approach [all the computations were done using the same C DLL], one can see that much of the increase in time for the Applications Center approach is due to the communication time for the data back to the master task. The amount of additional overhead added by using the C# wrapper depends on problem size, but is independent of the algorithm. It is relatively insignificant for the binary tree algorithm because its computation time is much greater than the Black-Scholes algorithm. In the latter case, the additional overhead for using the C# wrapper is a

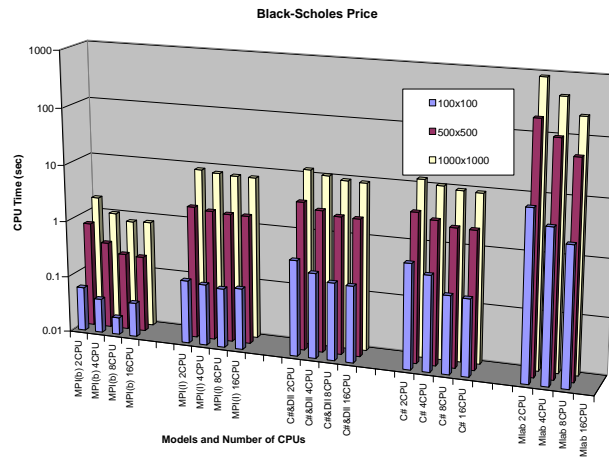


Figure 4. Results for experiments with the Black-Scholes pricing algorithm

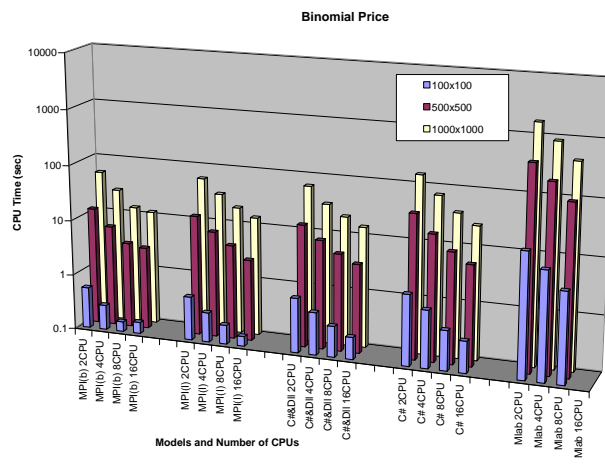


Figure 5. Results for experiments with the Binomial pricing algorithm

large fraction of the total, especially for small numbers of tasks.

The MATLAB calculations are very slow compared with all of the other approaches. This demonstrates that MATLAB is not a good vehicle for production parallel applications using Application Center clusters unless the numerically intensive part of the calculation can be put into a DLL. For users who wish to work in the MATLAB programming environment and want to gain the benefits of parallelism, the Cornell Multitask for MATLAB (CMTM) [11] running on a traditional cluster would probably give similar performance, but would allow communication between all processes. In that environment also, a user could put the numerically intensive part of the computation into a custom MEXfunction DLL and call it without having to get it installed into an Application Center.

## 5 Feature/Functionality Comparisons

All of the methods described above exploit different approaches to solving a parallel problem. The MPI approach is a traditional one that uses a simple C code. In order to use this approach, the MPI libraries need to be installed on each machine in the cluster. Calls to functions in the MPI libraries are made in the traditional way.

The C DLL can be exposed to a C# application using a P/Invoke call. A C# class can inherit COM+ properties and interfaces from the ServicedComponent class. The role of a C# wrapper is to assure a proper call from the master application on a remote machine to a function in a C DLL located in an Application Center cluster. With an appropriate C# wrapper available we can use the functionality of Application Center 2000 and COM+ services. This approach uses two DLL's: one is written in managed code (VB or C#) and the other is written in unmanaged code (C, C++ or FORTRAN).

A much cleaner approach is to rewrite the computational code in C# and incorporate it into the namespace of the C# class that inherits from ServicedComponent. This eliminates the need for P/Invoke calls to a local DLL. Only one managed DLL is required using this approach with Application Center 2000. Memory management and error handling in .NET applications is much improved compared to unmanaged code.

MATLAB functions will typically already be written in the MATLAB language. The master (client) application must send the MATLAB commands to perform computations as strings through the Execute COM interface. This style of programming is very generic, though tedious. Thus, a single .NET wrapper around the MATLAB ActiveX COM interfaces can be written once and reused for all MATLAB applications.

### 5.1 Strengths of Traditional HPC Methods

Application Center, with its independent applications that do not communicate directly with each other, is not designed for tightly coupled parallel applications that require communication between the worker tasks. However a new version of MPI/Pro[12], MPI/Pro.Net, may enable such applications to run efficiently on Applications Center clusters. However, because an Applications Center cluster does not guarantee resources equally to all tasks as dedicated access to nodes in a traditional cluster does, codes written for an Application Center cluster must use flexible algorithms that allow the workload to be adjusted between tasks according to conditions.

### 5.2 Strengths of .NET

As we have seen from the experiments described above, performance is typically slightly less with .NET, but the advantages of being able to control a computation from one's desktop through a custom form are often significant. Another advantage of .NET is the fact that your computation always begins as soon as you submit it. Although it may go slowly if there are many other applications competing for the Application Center resources, it will not sit in a queue doing nothing for hours, or perhaps days.

Application Center provides tools for monitoring resource usage, so it is easier for system administrators to learn whether their cluster is of an appropriate size and to add or remove resources from it as needed. Application Center also provides automatic load balancing, in the sense that new applications will be steered away from machines that are already busy. For managed code on the Application Center, it offers significant advantages in failure detection and error reporting. Codes that are appropriately structured even recover from node failures.

## 6 Conclusions

Application Center within the .NET framework provides an HPC environment for applications that require parallelism, fault-tolerance, and load balancing. The overhead associated with working in this environment is compensated by simplicity, ease of programming, immediacy of execution, and desktop integration.

## References

- 1 *Microsoft* <http://www.microsoft.com/net>
- 2 Gropp, W., Lusk, E., and Skjellum, A., *Using MPI - 2nd Edition*, MIT Press (1999)
- 3 *PVM* <http://www.csm.ornl.gov/pvm/>
- 4 Stiefel, M., Oberg, R. J., *Application Development Using C# and .NET*, Prentice Hall PTR, (2002)
- 5 Robinson, S., Harvey, B., Nagel, C., Greenvoss, Z., Cornes, O., Watson, K., Skinner, M., Glynn, J., Allen, S., *Professional C# (2nd Edition)*, Wrox Press Ltd. (2002)
- 6 Sterling, T., *Beowulf Cluster Computing with Windows*, MIT Press (2001)
- 7 *Myricom* <http://www.myricom.com>
- 8 *Infiniband* <http://www.infinibandta.org/home>
- 9 *Microsoft HPC* <http://www.microsoft.com/hpc>
- 10 *Math Works* <http://www.mathworks.com/>
- 11 *Cornell Multitask Toolbox for MATLAB* <http://www.tc.cornell.edu/CMTM>
- 12 *MPI/Pro* <http://www.mpi-softtech.com>