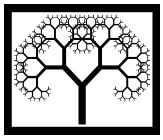
Paper 73



Proceedings of the Eighth International Conference on Engineering Computational Technology, B.H.V. Topping, (Editor), Civil-Comp Press, Stirlingshire, Scotland

Using WCF and TPL in Distributed and Parallel Finite Element Analysis

R.I. Mackie
Civil Engineering, School of Engineering, Physics and Mathematics
University of Dundee, United Kingdom

Abstract

This paper examines the use of .NET's Task Parallel Library (TPL) and Windows Communication Foundation (WCF) in finite element analysis. TPL is studied in terms of equation solvers. It is shown that there is no significant difference in execution speed compared with non-TPL code, though it does offer advantages in terms of software engineering. WCF based equation solvers are also examined, and a WCF approach to designing finite element software. Using WCF based solvers involves no loss of execution speed. WCF is an example of Service Oriented Architecture (SOA) and the possibilities for using this to increase flexibility of finite element based applications is discussed.

Keywords: component-oriented; object-oriented; parallel computing; distributed computing; server-oriented architecture; task parallel library.

1 Introduction

The world of computing has changed a great deal over the years. In the current environment most computers, both desktops and laptops, are multi-core. Furthermore, they are linked, both over internets and intranets. Parallel and distributed computing has been around for a long time with MPI being the most widely used technology. MPI and OpenMP were developed largely for scientific computing applications. The new possibilities bring with them new complexities, and full realisation of the potential requires tools that facilitate the handling of the complexities. Now that distributed and parallel computing are widely available general software technologies have been developed to facilitate their use.

Two technologies developed by Mircosoft are the Task Parallel Library (TPL) and Windows Communication Foundation (WCF). TPL deals with parallelism on a single machine, whereas WCF is designed for distributed computing. While both

these frameworks are developed for general computing, this paper will examine their use in scientific computing. Both the technologies are based on the .NET framework.

The author has previously examined the implementation of distributed computing using remote objects. Remote objects were part of the original .NET framework, and are still available. However WCF is seen as superseding remote objects, combining together various aspects of distributed computing.

The paper will look at the use of WCF and TPL in finite element software. In particular it will look at the implementation of equation solvers, finite element objects, and finite element models using a WCF approach, with appropriate use of TPL. The focus is on program design, using object and component oriented methods. The primary advantage of component oriented design is that it isolates complexity. Previous work has shown how this enhances program design for distributed computing. The current paper further emphasises the value of this isolation of complexity as it minimises the impact of the new technology on existing software.

Any programming technology needs to be assessed for its impact in the following areas:

- Speed of execution
- Program design (software engineering)
- Software capabilities

WCF and TPL will be assessed in each of these categories within the context of finite element analysis, though the emphasis will be on the latter two aspects.

2 Literature review

Parallel computing is not new, but what is relatively recent is that with virtually all computers having multiple cores, parallel computing is a mainstream concern. For while in recent years the number of transistors on a chip has continued to follow Moore's Law, the chip speed has not. Instead the number of cores per chip has multiplied [1]. Indeed, even some mobile phones are now dual core. In order to take full advantage of these capabilities it is necessary to design the software specifically to do so [2, 3]. Furthermore, all computers are linked together, whether it is via the internet, local networks or wireless computing. This change in architecture requires a change in the programming model if the full potential is to be realised. Therefore distributed and parallel computing is no longer the preserve of specialist machines, but is relevant to all computing.

Perhaps the most commonly used technology to achieve parallelism and use computer clusters in engineering computing is MPI [4]. Alongside this there is OpenMP [5] which is widely used as well. There are implementations of these for .NET and Java environments, but .NET and Java have their own means of implementing parallelism. A review and comparison of these approaches can be found elsewhere [6].

With version 4 of .NET the Task Parallel Library (TPL) was introduced. This was designed to make parallel programming easier. Qiu et al [7] have compared TPL with MPI, and also CCR (concurrency and co-ordination runtime) on a Windows clusters with 768 cores. They reported that MPI was better at low levels of parallelism, whereas the thread based TPL was better as the grain size decreased.

TPL, as its name implies, is task based. This means that its focus is on the tasks to be executed rather than the threads on which they are executed. There appears to be a convergence of thinking that parallel systems need to be task oriented. Ayguarde et al [8] report work on the integration of tasks into OpenMP, while Giacaman and Sinnen [9] present a system for Java called ParaTask (Parallel Task).

Just as parallel computing is by no means new, neither is distributed computing. For instance, Chanda and Baugh[10] used domain decomposition methods to implement distributed finite element analysis. Again as with parallel computing; distributed computing is now common place, with all computers, along with many other devices, being interconnected through the internet, and local networks. Indeed, most people's computing experience is of a distributed nature via the internet. As well as networks of computers are being an important area of distributed computing in engineering analysis, internet based computing is the object of increasing attention [11-14].

Previous work by the author has examined the use of remoting in .NET to implement distributed computing in finite element analysis [15,16]. The Windows Communication Framework (WCF) was introduced a few years ago. WCF is a Service-Oriented Architecture (SOA). There are several good books on WCF [17, 18]. It is relatively new, and has its roots more in general business computing. Hence, there has so far been little use of it in scientific or engineering computing. Exceptions have been in the area of distributed information systems. One example is work by Chang et al [19] who used WCF on distributed 3D-GIS . They cited the advantage of WCF being that it brought together several communication mechanisms, including .NET, DCOM, Message Queuing and Web Services. Chengping et al [20] have done work on the application WCF in the water industry, and Yang et al [21] have used it for borehole logging data obtained from geological investigations. Stopper and Gastermann [22, 23] have worked on the use of WCF in information systems in the manufacturing environment.

3 Task Parallel Library

Perhaps the most fundamental aspect of TPL is that, as its name implies, it is focused on tasks. Prior to TPL parallel execution was implemented using threads. Prior to TPL, at the most basic level one created threads and ran tasks on the threads. .NET introduced the threadpool. This has a pool of threads. Then, rather than new threads being repeatedly created and destroyed, a thread was taken from the pool when it was needed. However, the programmer still had to think about the thread

and the tasks to be executed. With TPL the programmer can focus almost entirely upon the task, with TPL taking care of most of the thread related aspects itself. This makes for better programming, and is the way that other environments are going as well [8, 9]. The tasks themselves can be either defined as object methods, or using lambda expressions (ie like anonymous methods).

The most basic form of parallelism is the use of loops. The code fragment below shows a simple example,

```
Parallel.ForEach(subDoms, subDom =>
{
   IDirSubDomSym sub = subDom as IDirSubDomSym;
   sub.Decompose();
   lock (lockDecompose)
   {
      knn.Assemble(sub.Knn, sub.GloLoc.ExtDofs);
      iSub++;
      DoSubDomCompleted(iSub);
   }
});
```

This simply instructs the program to carry out the decomposition for each sub-domain in a domain decomposition direct solver. The `Parallel.ForEach` construct tells the program to use a separate thread for each sub-domain. The loop will not complete until all the work for each sub-domain has been completed. The lock segment ensures that this part of the code is only executed by one thread at a time, thus ensuring data integrity.

Indexed for loops can be implemented in a similar manner, eg:

```
for (int i = 0; i < n - 1; i++)
{
   Parallel.For(i + 1, n, j =>
   {
      double q = A[j][i] / A[i][i];
      for (int k = i + 1; k < n; k++)
      {
         A[j][k] -= q * A[i]   [k];
      }
      x[j] -= q * x[i];
   });
}
```

It can be seen that the syntax is somewhat similar to OpenMP. The `for` loops can be used in a more sophisticated level, with cancellation tokens. It is also possible to use tasks in more sophisticated ways. Tasks can be executed in parallel, continuation tasks can be defined (where one task starts only when another one has finished), and child tasks can be created.

Cancellation tokens are used for stopping tasks. The client would create a cancellation token and this is passed to all the relevant tasks. If the client wishes to cancel the associated tasks then it sets the token. The tasks periodically check the token, and if it has been set to cancel they will then stop themselves in an elegant manner.

Within the context of equation solvers it is `foreach` loops that are the most common use of TPL. An important aspect of any parallel programming is the avoidance of data conflicts. While parallel frameworks can provide mechanisms for ensuring data consistency, they cannot handle the logic automatically. The programmer has to understand the data requirements and build in the logic him or herself. This is not the place to go into all the details, the interested reader is referred to [24], suffice to say that TPL offers much more sophisticated ways of implementing parallelism.

Within the context of equation solvers it is `foreach` loops that are the most common use of TPL. TPL uses a threadpool for each thread, so is not continually creating and destroying threads. An important aspect of any parallel programming is the avoidance of data conflicts. While parallel frameworks can provide mechanisms for ensuring data consistency, they cannot handle the logic automatically. The programmer has to understand the data requirements and build in the logic him or herself. A well as straightforward for loops, TPL can use chunking for loops where each iteration involves just a small amount of work. Cancellation tokens are very useful and the primary means for cancelling tasks.

Figure 1 gives some test results comparing calculations between a TPL based program and a non-TPL program (and with the WCF program referred to in the next section). Various methods are compared: direct solution (UDU), conjugate gradient without pre-conditioning (CG) and with pre-conditioning (CGPC), and Schur conjugate gradient (Schur CG). All the times have been normalised to 1 for the WCF time. The UDU runs were for a 98 816 degree of freedom problem, the others were all for 394, 240 degrees of freedom,

The non-TPL program was identical to the TPL one, except that delegates were used rather than tasks to parallelise the sub-domain aspects. As can be seen there is very little difference in speed. The advantage of TPL lies in the software engineering aspects, leading to simpler code.

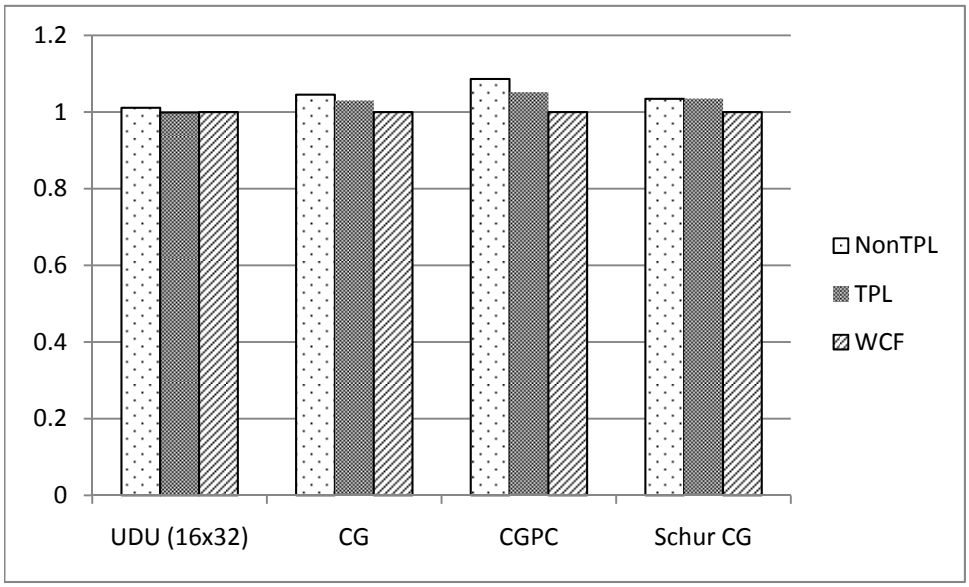


Figure 1 – Speed comparisons, normalised to WCF.

4 Windows Communication Framework

WCF is an example of service-oriented architecture (SOA). According to [17] the four tenets of SOA are:

- Service boundaries are explicit;
- Services are autonomous;
- Services share operational contracts and data schema, not type-specific metadata;
- Services are compatible based on policy.

What does this mean in practice? Well, it means that services are exposed purely as an interface. It should reveal nothing about implementation details. So clients should be able to use the service without being concerned about its location or implementation. The service should be independent of the client, or other services. The service should be entirely self-describing. So the service, as far as the client is concerned, is entirely described by the operations contract (which defines what the service can do) and the data contracts (which define the data required). All this is aimed at minimising coupling.

Now .NET also has remote objects, and these can be implemented and accessed through interfaces. This is very similar to WCF, so what is the difference? The key difference is that in remoting remote objects are treated just like local objects. This has advantages, but also drawbacks. The drawbacks include the fact that with remote objects issues such as lifecycle management, security and reliability become important. WCF is designed to handle these and other issues. The service can be located either locally, on the intranet or internet, and can be accessed by various means. Under WCF the client accesses the service in exactly the same way, regardless of where the service is.

The difference between WCF and remoting is that in remoting the client was essentially using the interface to manipulate an object. In WCF the client is accessing a service only. The service has full control over what happens the other side of the boundary. The interface is a contract of what the service will provide. Moreover, remoting was tightly tied in to .NET, and the remote objects could only be used by other .NET applications. While .NET is used to implement WCF services, they can be accessed by any other program or operating environments.

So why is a technology with its roots in business computing being considered for use in engineering software? Well, at the most basic level a distributed computing technology only needs to enable computation to be executed on another computer and to transfer and receive data This is essentially what MPI does. Remoting allows this to be done for .NET. WCF also allows this to be done, as do various Java technologies. However, now that we live in a distributed computing world the requirements become greater. As well as carrying out execution on remote computers there is the need for inter-operability. Then there is the need for security and reliability. WCF is a complete framework and takes account of these matters. Engineering design and analysis is an increasingly global and distributed exercise. So there are good reasons for considering the applicability of WCF.

This is not the place to go into all the details. Instead the focus will be on the aspects pertinent to the current context. In any situation there are two key aspects, Operation Contracts and Data Contracts. Operation Contracts define the messages that the service can handle, and Data Contracts are used to define more complex data types. A service must use only standard data types (doubles, strings, arrays etc) and Data Contract types. Ie the service must be completely self-defining.

4.1 Equation solvers

Consider an equation solver for sets of linear symmetric equations. Figure 2 shows a suitable arrangement of Operation Contracts for single domain solvers. `IWcfSolverSym` is the basic contract; `IWcfSolverSymDir` and `IWcfSolverSymIter` extend this a little for direct and iterative solvers respectively. These are interfaces, ie they say what a component can do, but nothing about how it will be done. `IWcfSolverSym` is the most important one.

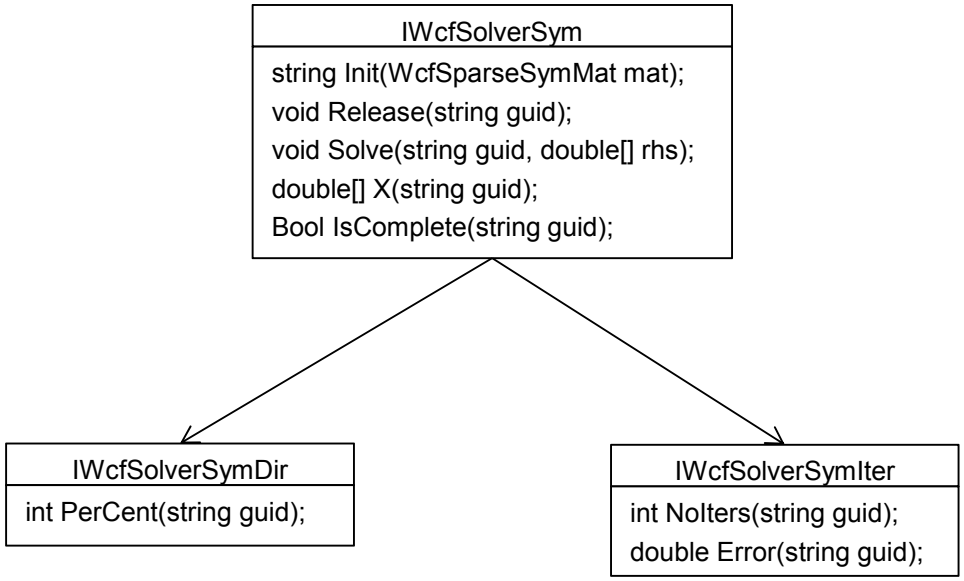


Figure 2- Interfaces for single domain solvers

In the code itself the interface declarations are decorated with attributes as follows:

```
[ServiceContract]
public interface IWcfSolverSym
{
   [OperationContract]
   string Init(WcfSparseSymMat mat);

   …
}
```

The [ServiceContract] attribute tells the compiler that the interface defines a service contract, and each method that is to be included in the ServiceContract is decorated with the attribute [OperationContract].

WcfSparseSymMat is a data type defined by the service that contains the data describing the equation matrix data. This is declared as a [DataContract]. DataContracts are used so that any client knows exactly what data the service requires, and is not dependent on any other data type. More is said about this later. Init is called by the client to initialise the service. This returns a string; this string is a guid (globally unique identifier) and is used to identify the equation set. It effectively acts as a handle (fulfilling the role of a pointer in a purely local program). So when the client later calls the service it uses this string to identify the equation set it is referring to. The client calls Release when it has finished with the service. The Solve method tells the service to solve the equations with the supplied right-hand-side. IsComplete is used to query whether the service has completed the solution, and X is used to obtain the solution.

For the two descendants. PerCent and Iter and Error give information on the progress of the solution.

A client could create a direct solver with the following code:

```
m_Factory = new ChannelFactory<IWcfSolverSymDir>("UDU Solver
  Piped");
m_WcfSolver = m_Factory.CreateChannel();
```

There is a two stage process. First a ChannelFactory is created. This factory will create objects that implement IWcfSolverSymDir. The text "UDU Solver Piped" identifies the service to be used. The client program has a configuration file, the text identifies the service to be used. In this case the service runs on the same computer, and is accessed by inter-process communication, the most efficient for accessing services running on the same computer. It could equally well be accessed by tcp or http. Then the second line actually creates the object. The object can actually be created every time it is needed, or it can be "kept alive" between calls.

Data is passed in the form WcfSparseSymMat. This is a Data Contract. These are defined by the service, as shown in Figure 3. WcfSparseSymMat itself used a further Data Contract, WcfSparseVector. These entities are defined as part of the service, so that the service is completely self-defined and any program using the service knows the format of the data required. So there is no coupling between the service and the client, such as both using a common library of data types.

The solution is carried out by the Solve method, which passes the right hand side. Once complete the solution can be obtained via the X method. The Solve method can be executed asynchronously so that it does not block the client program, and the X method is used to access the results once it has completed.

Now everything that has been done could have been done using remote objects, and some of the design features would have been maintained. For instance the client would have used an interface to the solver, and would have been logically, and possibly physically separate from the solver itself. The solver could have been

accessed via IPC, TCP or http. So what is the difference? The fundamental difference is that in remoting the remote object is treated just as though it is a local object. The second difference is that service could be accessed from a non .NET application. With regard to the former difference, this shows itself in the use of guid in the methods for the service.

Figure 3 shows the data contracts. As with service contracts, the definitions are decorated with attributes as follows:

```
 [DataContract]
public class WcfSparseSymMat
{
   [DataMember]
   public int Size {get; set;}
   …
}
```

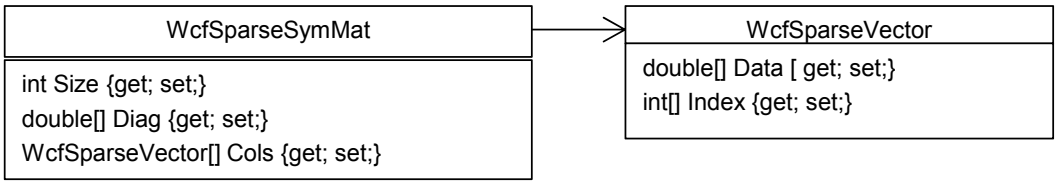


Figure 3 – Data Contracts

[`DataContract`] tells the compiler that the type is a data contract, and so it will be exposed as part of the service definition. Each data member that is to be exposed is decorated with [`DataMember`]. `WcfSparseSymMat` defines the coefficient matrix. It makes use of `WcfSparseVector`, which is another data contract. So the data required by the service consists only of basic types (int, double etc) and `DataContract` types.

Figure 4 shows the equivalent interfaces and data contracts for a domain decomposition solver. Only part of the interface for `IWcfSolverSubDomSym` is shown for the sake of brevity. There are several other methods related to the right-hand sides of the sub-domains etc. As with `IWcfSolverSym` there are various methods that add sub-domains, solve the equations, set the right-hand side data etc. Again, `guid` is used to identify a particular equation set, with `iSub` used to identify sub-domains within that equation set. `WcfSubDomSym` and the associated data contracts are used for passing the data.

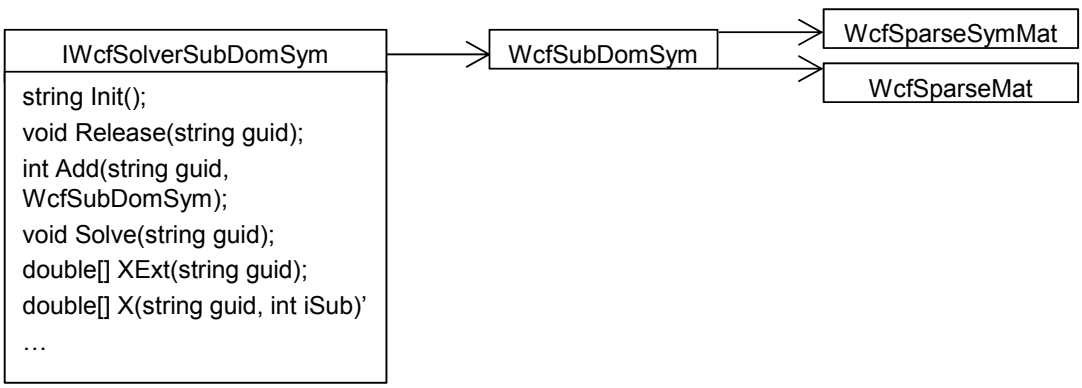


Figure 4 – Interface and Data Contracts for domain decomposition solvers

The client is totally unaware of how the service is provided. As before, it could be accessed via IPC, TPC or html, and could be on the local computer, intranet or internet. The domain decomposition solver itself could either be carrying out the solution on a single multi-core computer, or using several computers. This is of no concern to the client, but is entirely the responsibility of the service.

Now we will assess the WCF implementation in terms of the three criteria listed earlier in this paper. Figure 1 shows the speed comparisons using IPC (i.e. the solver service is running on the same computer). On the whole this is little different than the standard single program solvers. Indeed in some cases the WCF version is a little faster. This difference is relatively small, and more evidence would be needed to see if it was significant. It is possible that the fact the solver is now running in a separate process is leading to this small speed up, though this is only speculation at this stage. However, on IPC there is no loss of efficiency.

In terms of software engineering the client program itself does not contain any solver code at all, so there is complete separation of the client and solver code. Now this could have been achieved with remoting as well. The advantage of WCF is that it has better reliability, security and exception handling capabilities. For the case of the individual researcher or single research team these feature may not be of particular concern, but for commercial applications these things become more important.

Finally what about program capabilities? On a single computer it only needs to have one equation solver program. This would offer equation solver capabilities as a service, and a variety of programs could make use of this service. For very large scale problems a dedicated cluster of computers could be set up to offer the solving service.

4.2 Stiffness matrices

The next example will be that of element stiffness matrix calculation. This could be extended to other things like mass matrices, but for simplicity attention will be limited to the stiffness matrix calculation. The motivation for this is that an element has two distinct characteristics. One is its geometric aspects; the other is the structural features. One of the key difficulties in designing class libraries for finite elements is the interplay between these two aspects, and this can lead to quite complex class libraries. Due to space limitations, details will not be given here.

The system works well enough, but the drawback is that the client program has to supply quite a lot of data in order to use the service. There is also duplication, particularly with regard to iso-parametric calculations, these needing to be implemented on both the client and the service. There may be some advantages in that new elements could be implemented in a program via a service. The author is not currently convinced of the advantages of WCF in this context, but it does point to a useful application, examined in the next section.

4.3 Finite element model

Finite element programs commonly adopt the following model. The user creates a graphical model of the structure, with restraints, applied loads etc. In addition information to enable the program to create the finite element mesh are added. This can occur in various forms, such as specifying sub-divisions along lines, specifying mesh sizes at key points, using a background grid etc. So there is a structural model, and from this a finite element model is created. Most of the user interaction takes place via the structural model. It would there seem reasonable to investigate whether or not there is any advantage to be gained in treating the finite element model as a service.

Adopting this approach the client program consists of the structural/graphical model. The finite element model exists as a service. This is illustrated in Figure 5.

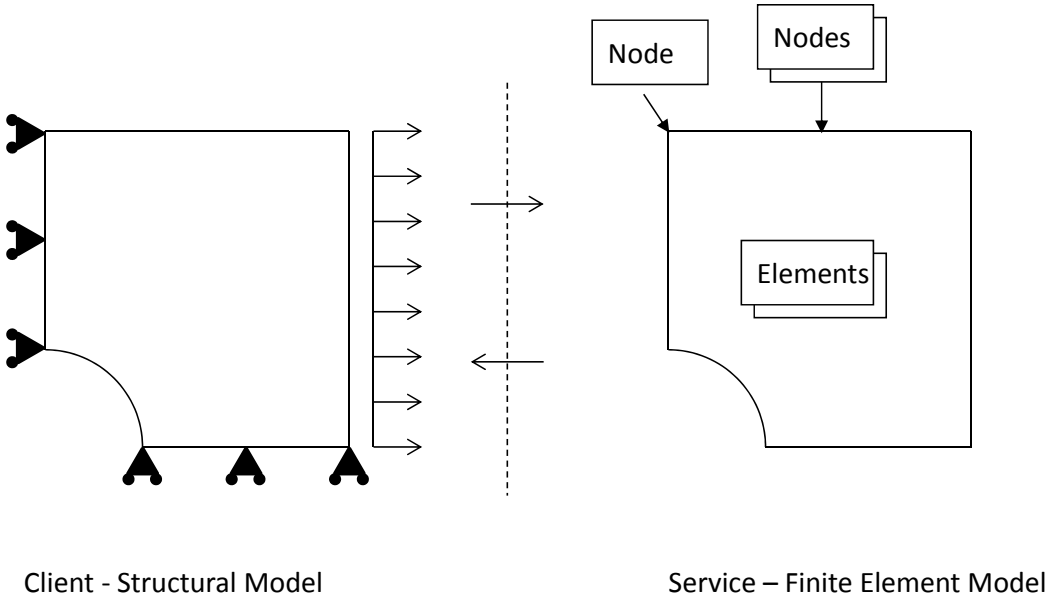


Figure 5 – Client-Service Architecture

The client consists of the structural model; this data is sent to the service, which then constructs the finite element model. The distributed finite element object model, described elsewhere [25, 16] is used, so nodes and elements are associated with key points, key lines, sub-structures etc. So if the client needs information associated with a particular line, for instance, then it is easily accessible from the model on the service.

What are the advantages, if any, of doing this? In terms of execution speed there is no immediate advantage, though equally there is little cost. The prime advantage comes in terms of software design. A key advantage of component oriented design is the isolation of areas of complexity. In a monolithic program there will be coupling between the structural and finite element model. In theory interfaces could be used to ensure logical separation, and while this can be achieved to some degree, it is

actually very difficult to avoid coupling the two. The use of a service for the finite element model better enforces or encourages the logical separation.

This leads on to other potential advantages. The finite element model could be on the same computer, or on a remote system. The remote system could also potentially be using a cluster of computers for solving very large problems. The client program would be completely oblivious to this. Maintenance and updating of the finite element service would be completely separate from the client program.

So there are advantages in terms of software engineering. This leads on to possible advantages in terms of capabilities. The typical finite element program is general purpose, allowing the user to create models of any complexity (within certain limits). However, suppose a team were interested in models of a certain type. As a simple example, consider a rectangular coupon with a hole in it, where the hole could have various dimensions and its location could also be varied.

A very simple program could be made that produced the structural model for this. This model would then be sent to the Finite Element Model service and analysed. Now the finite element model service might exist on the same computer, but more likely on a remote computer accessed via the internet. The client program would be very small, and could easily be implemented say on the web or even as an applet on a smart mobile phone!

So the use of SOA could lead to the development of a new class of applications and make the use of finite element analysis easier.

5 Conclusions

The paper has examined the use of the Task Parallel Library (TPL) and Windows Communication Framework (WCF) in the context of finite element analysis.

TPL is designed to make parallel programming easier, and adopts a task based approach. The code produced is similar in execution speed to a direct thread based approach, but offers more features to facilitate the programming.

A SOA approach was used to implement equation solvers. The code was similar in terms of execution speed to the direct TPL approach. The advantage was that equation solvers could be offered as a service by a powerful computer cluster. On a single computer, a single equation solver service could be used by a variety of programs that needed equation solution as part of their operation.

A SOA approach allows the logical and physical separation of the structural and finite element models in a finite element application. This leads to advantages in program design. Perhaps more importantly, it opens up the possibility of small applets being developed that make use of finite element analysis, the analysis being supplied by a remote service.

References

[1] C. Terboven. OpenMP in the Multicore Era. Invited talk at the 5th International Conference on Automatic Differentiation in August 2008, Bonn, Germany.

[2] D. Dig, "A Practical Tutorial on Refactoring for Parallelism", 26th IEEE International Conference on Software Maintenance in Timi□oara, Romania, September 12-18, 2010.

[3] A. Ebnenasir and R. Beik, "Developing Parallel Programs: A Design-Oriented Perspective", IWMSE 2009, Vancouver, Canada, May 18, 2009.

[4] Message Passing Interface, http://www-unix.mcs.anl.gov/mpi/

[5] OpenMP, http://www.openmp.org.

[6] R.I. Mackie, "High Performance Computing on Low Cost Computers: A Review of Parallel and Distributed Computing Methodologies for Finite Element Analysis", in B.H.V. Topping, J.M. Adams, F.J. Pallares, R Bru and M.L. Romero (Editors), Developments and Applications in Engineering Computational Technology, Saxe-Coburg Publications, Stirlingshire, UK, Chapter 12, pp 263-285, 2010. doi:10.4203/csets.26.12

[7] J. Qui, S Beason, S-H Bae, S Ekanayake and G Fox, "Performance of Windows Multicore Systems on Threading and MPI", 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, 2010.

[8] E Ayguade, N Copty, A Duran, J Hoeflinger, Y Lin, F Massaioli, X Teruel, P Unnikrishnan and G Zhang, "The Design of OpenMP Tasks", IEEE Transactions on Parallel and Distributed Systems, 20(3), 404-418, 2009.

[9] N Giacaman and O Sinnen, "Parallel Task for parallelizing object-oriented desktop applications", Parallel & Distributed Processing, Workshops, 2010 IEEE International Symposium, 2010, Atlanta Georgia, 19-23 April 2010.

[10] H.S. Chadha, J.W. Baugh, "Network-distributed finite element analysis" Advances in Engineering Software, 25, 267-280, 1996.

[11] H.M. Chen , Y.C. Lin, "Internet based Analytical Services using a Java based GUI", in Proceedings of the Tenth International Conference on Civil, Structural and Environmental Engineering Computing, Topping BHV, editor, Civil-Comp Press, Stirling, United Kingdom, paper 14, 2005.

[12] M. Nuggehally, Y.S. Liu, S.B. Chaudhari, P. Thampi, "An internet-based computing platform for the boundary element method", Advances in Engineering Software, 34, 261-269, 2003.

[13] J. Peng, K.H. Law, "Building finite element analysis programs in distributed services environment", Computers & Structures 82, 1813-1833, 2004.

[14] Z. Yang, J. Lu, A. Elgamal, "A Web-based platform for computer simulation of seismic ground response". Advances in Engineering Software 35, 249-259, 2004.

[15] R.I. Mackie, "Design and Deployment of Distributed Numerical Applications Using .NET and Component Oriented Programming", Advances in Engineering Software, 40, 665-674, 2009.

[16] R.I. Mackie, "Programming Distributed Finite Element Analysis: An Object

Oriented Approach", Saxe-Coburg, Stirling, ISBN 978-1-87.4672-31-9, 2007.
[17] J. Löwy, "Programming WCF (3rd ed)", O'Reilly, 2010.
[18] M.L. Bustamante, "Learning WCF: A Hands-on Guide", O'Reilly, 2007.
[19] C. Chang, Q. Hao and Z. Song, "Research on Distributed 3D-GIS Based on WCF", 2010 International Conference on Electrical and Control Engineering, Wuhan China, 25-27 June, 2010.
[20] Z. Chengping. N. Qian, L. Chuan, "Research and design of distributed sluice group joint dispatch system based on WCF and OPC technology", 2nd WRI World Congress on Software Engineering, WCSE 2010; Wuhan; 19 December 2010 through 20 December 2010
[21] G. Yang, J. Shen, W. Song, "The Design and Implementation of Distributed Logging Diagram System Based on WCF Technology", 2011 International Conference on Electric and Electronics, EEIC 2011; Nanchang; 20 June 2011 through 22 June 2011.
[22] M. Stopper and B. Gastermann, "Service-oriented communication concept based on WCF.NET for industrial applications", International MultiConference of Engineers and Computer Scientists 2010, IMECS 2010; Kowloon; 17 March 2010 through 19 March 2010.
[23] B. Gastermann and M. Stopper, "Agent-based services using WCF technology and RFID for autonomous control in continuous flow production", International MultiConference of Engineers and Computer Scientists 2011, IMECS 2011; Kowloon; 16 March 2011 through 18 March 2011.
[24] A. Freeman, "Pro. NET 4.0 Parallel Programming in C#", APress, 2010.
[25] R.I. Mackie, "Object oriented methods and finite element analysis", Saxe-Coburg, Stirling, ISBN 1-874-672-08-3, (2001)