Virtual Manufacturing
A Vision for Virtual Paint Operations

Patrick Gaffney[1], Tim Hopkins[2]
[1] BSSI AS, Post Box 18, Nyborg, Heiane 4, 5879 Bergen, Norway
pat@bssi-tt.com,
WWW home page: http://www.bssi-tt.com
[2] Computing Laboratory, University of Kent
Canterbury, Kent, CT2 7NF, UK
t.r.hopkins@kent.ac.uk,
WWW home page: http://www.cs.kent.ac.uk/people/staff/trh/index.html

Abstract. The present paper deals less with Grid computing than with other aspects of this conference, namely numerical software. The goal of our work is to use mathematical modeling and computer simulation to avoid building non-production prototypes in manufacturing processes. Of particular interest is to eliminate prototypes associated with *painting processes* used in the automotive industry. Since painting processes are difficult to simulate accurately, the numerical software employed in the computational simulation is quite complex and requires substantial computing power to get a decent result within the timeframes of an industrial operational environment. On the other hand, the rewards for getting it right go straight to a company's bottom line: eliminating prototypes leads to tangible savings in terms of paint and materials, man power, energy costs, reduced warrantee costs, improved operational processes, and improved quality of the finished product. Getting it right therefore, is the subject of this paper.

1 Introduction

To most people painting a car conjures up pictures, from Discovery Channel and elsewhere, of robots spray painting cars on an assembly line. Although this is the way most vehicles end up by being painted it is by no means the whole story. For that, one needs to go back and look at the processes that prepare the metal of the car body so that it can become the final shiny product seen on the road.

Please use the following format when citing this chapter:

Gaffney, P., Hopkins, T., 2007, in IFIP International Federation for Information Processing, Volume 239, Grid-Based Problem Solving Environments, eds. Gaffney, P. W., Pool, J.C.T., (Boston: Springer), pp. 71-88.

That shiny product is the reflection of a sequence of processes that prepare the vehicle to sustain the effects from inclement weather - corrosion, while providing a base coating of the metal that enhances the quality of the finished vehicle.

The processes are well known to most people, although they may not connect them to being applied to cars on the road. Broadly speaking the processes consists of cleaning the metal of the vehicle body, preparing the metal to receive paint, applying a protective coating - electrocoating, rinsing, and baking. The protective coating is organic and the baking process is the final act that melts the resin polymer in the coating to provide the required corrosion protection.

It is crucial to get this sequence of painting processes right because errors at any stage have a detrimental impact on the bottom line of the automotive company, and some errors will become apparent to the unhappy owner. Errors will occur for a variety of reasons: maybe the cleaning was not effective enough, maybe some of the protective coating remains in puddles in the vehicle as it enters the oven, maybe the protective coating did not adequately cover all areas of the vehicle so that hard to reach areas do not have enough coating or maybe they do not have any. The list is almost endless, and each one of these items is significantly costly to fix.

In an attempt to understand the causes of errors and identify suitable remedial action, companies have traditionally experimented with physical prototypes of new vehicles by subjecting them to the same painting processes. However, experimentation and testing on physical prototypes is necessarily limited to the number of prototypes that can be built, not to mention the costs incurred in terms of manpower, materials, time, and the fact that the physical operating conditions, i.e. the paint tanks and cure ovens, must be used for the experiments, and therefore the normal manufacturing process must be interrupted, which in itself is an unacceptable cost to bear.

The *hope* of uncovering better ways of ensuring success in the application of the painting processes mentioned above is a vain one if all that is done is to use *physical* trial and error experimentation. On the other hand the *hope* becomes a reality if one can use computers to replace the *physical* nature of the experimentation. To do this requires that the painting processes must be simulated on a computer and this simulation requires detailed and accurate mathematical models of the underlying physics and chemistry of each painting process.

2 Virtual Manufacturing

Virtual manufacturing is the application of computer software to simulate a new product and the processes required to take that product from a design concept through to its actual manufacture using physical materials. Software simulation allows problems to be identified and remedied before the fabrication of the product begins.

Virtual manufacturing is not a new concept; it has been around for several decades, promoted, justifiably, by the use of finite element techniques to model complex geometries thereby expanding the range of products that are susceptible to this process.

The finite element method is a computational method to solve complex three-dimensional partial differential equations, especially those where it is important to know what happens in the *interior* of the solution domain.

Another technique involves recasting the partial differential equations as integral equations and then employing Boundary Integral Methods for their solution. For this recasting to be possible, a fundamental solution of the partial differential equations, a so-called Green's function, must exist. The set of industrial problems for which a Green's function exists is not as large as the set of problems that can be solved by the finite element method. However, for many industrial problems in engineering, it is possible to determine a Green's function and for these cases boundary integral methods are an alternative solution technique to the finite element method. Boundary integral methods are useful when it is important to know what happens on the *boundary* of a closed region as opposed to the *interior*. There are many industrial processes where the solution is only required on the boundary and for these cases, boundary integral methods are often the only practical approach because including the interior of the region makes a solution by the finite element method computationally intractable. Three particular processes where the solution is required on the boundary are:

- Electrochemical processes
- Transient heat flow in an oven
- Crack propagation

Thus, the boundary integral technique further extends the range of problems that can be addressed by virtual manufacturing. There are two main aspects of virtual manufacturing though: the simulation of the new product *and* the simulation of the processes. Finite element and boundary integral techniques extend the range of products that can be simulated but it still remains to simulate the processes themselves and the processes mentioned above are non-trivial.

Recent advances in the mathematical techniques underpinning the boundary integral method combined with advances in computer hardware and software mean that virtual manufacturing can be extended to simulating electrochemical processes especially those involved in the painting processes described in the previous section.

3 Characteristics of Painting Processes

Although painting processes are simple to describe, the resultant three-dimensional mathematical models of them are complex and range from potential theory, through fluid flow and computational fluid dynamics, to elasticity, to name but a few. Nevertheless, the overriding most important consideration of any simulation of these processes is the inclusion of *time*. Steady-state descriptions of these processes are completely inadequate as a basis for *predicting* what will happen in the physical process. Operators need to know what happens at any instant of a process and therefore accurate time-dependent solutions are a necessity for

employing computational simulation as a replacement for testing and experimentation using physical prototypes.

Computational simulations of painting processes are time dependent and for operational reasons a true time-stepping solution method must be employed. Solution domains involve complex geometries consisting of automobiles, paint tanks, cleaning tanks, and cure ovens. A *virtual* toolset that replaces *physical* trial-and-error testing thus requires incorporating all of these items and human experience into software that can be used easily by non-computer specialists, at both ends of the manufacturing chain: the upstream users at the design stage and the downstream users at the manufacturing stage. Only when this is done properly, and the software is validated, will users be convinced to switch to virtual tools as part of their everyday work environment.

4 Validation

The previous paragraphs help to characterize the nature of the software that industrial operational environments need. At the heart of the simulation software are methods and techniques that are traditionally described as *numerical software*. It is a long strenuous path to take that numerical software and form a system that commercial users will feel comfortable with. It is not only the look-and-feel of the software that is important it is the confidence that using the software will help them to understand better the physical manufacturing processes that they are really interested in. This means effectively that the users are confident of the results obtained from the simulation, in other words the software has been validated to produce accurate results that can be relied upon. This level of confidence is often not present in what many people regard as *numerical software*.

The testing regime, to which operational software of the functionality and calibre described above is subjected, is rigorous in the extreme. The process begins after the initial software has been written and takes the form of unit testing on geometries that exhibit some of the behavior expected in real automobiles and that can be verified by simple laboratory experiments conducted by the automobile company. Even these tests are more rigorous than those used to verify software published in ACM TOMS for example. As development proceeds, test geometries evolve in complexity until at a certain stage actual vehicle parts and bodies are used to stress test the software. This process is monitored closely by end users who actually reproduce the results by running the software themselves on their own machines and compare results with measurements taken of physical prototypes that have been stripped down and measured after painting.

These testing and validation procedures contribute to confidence building and numerical software developers should take heed and learn from them. For example, in addition to testing on increasingly complex models the process benefits from the active participation of an *independent champion* of the software, namely the automobile user. As development of the software proceeds it is in the interests of the automobile company that the software succeeds and therefore aberrations and errors that come to light from their testing are immediately brought to the attention of the

developers. In the course of developing the virtual toolset described in this paper, approximately 80% of the time has been spent on testing, verification, and debugging.

5 Virtual Paint Operations™ (VPO™)

VPO™ is the name of a family of software products and services developed by BSSI over the last 10 years according to the descriptions given in the previous sections, where the champion in this case is Ford Motor Company in North America. The software addresses the simulation of the painting processes: electrocoating, drainage, baking, and the elimination of air pockets. Each simulator in the toolset is a time stepping *predictor* that enables the operator to access the results of the simulation at any moment in time. Thus, for example, it is possible for an operator to monitor the oven baking process and see how the predicted solution behaves at every second the automobile is in the oven. In this way, the company can develop operating strategies that enable them to take account of changes in operations, for example different oven loads, without interrupting daily operations in the actual oven.

To address look-and-feel, the software is operated through a unified interface that provides access to all the simulators from one graphical user interface. The painting processes described previously are not independent of each other. For example, doing a poor job of electrocoating has an affect on the quality that can be expected from the cure oven. Consequently, the emphasis of the interface design is in the inter-operability of the simulators. Being able to simulate different what-if scenarios that *couple* baking, drainage, and electrocoating simulators - by passing data and results backwards and forwards between the different simulators - is an emphasis that stems from watching how users perform their daily tasks. The tasks are not independent and therefore as far as possible it should be possible to use the simulators in a similar way.

Operational environments, especially manufacturing processes that produce millions of items per year, have a strong emphasis on timeliness: operations and decisions need to be taken quickly and in a time that does not hamper the manufacturing process or delay the throughput of items (vehicles in this case). Consequently, it is very important that the simulators produce results quickly – not necessarily the *final* results but intermediate ones. For example, in a simulation of the electrocoating process of a vehicle the VPO™ EPD code can produce output at every $1/1000^{th}$ of a second and this output is available to the user for analysis as soon as it is produced. In this way, the user can take decisions without having to wait for the simulation to finish.

The single most effective way of handling timeliness, and the production of results rapidly, is to use parallel computing, and in today's environments clusters are the preferred architecture.

Of course all of the above – real-time visualization of results, coupling, inter-operability of simulators, and computational steering - assumes that information, in the form of intermediate results from the simulator, is readily available to the user,

which it is from the VPO™ System operating in a Grid-like environment, but it is not when the system is constrained to work in batch environments.

Of the painting processes discussed above, the one that is the most complex to simulate and computationally intensive is electrocoating.

6 Electrocoat Simulation – VPO™ EPD

The process to apply rust-preventing coatings to automobiles must ensure that all parts of the vehicle have an adequate minimum build of paint. Currently, the main method used to minimize salt spray induced corrosion involves application of an epoxy-based urethane coating using an electrodeposition technique called electrocoating. Car parts are suspended from a gantry, lowered, and transported through a tank filled with the urethane paint called electrocoat or e-coat. Each part is treated as a cathode and by applying a voltage to the anodes; the paint adheres to the car part.

Basic mathematical model

(a) The rate that paint is deposited on the cathode is a function of the local current density **j**:

$$\mathbf{d/dt(L_F(t)) = \beta * j}$$

$\mathbf{L_F}$ is the thickness of the paint and $\boldsymbol{\beta}$ is the current efficiency.

(b) The current density **j**, as a function of position over the cathode surface, is obtained from the *normal derivative* of the electric potential $\boldsymbol{\Phi}$, which is determined by solving the 3D Laplace equation $\nabla^2\boldsymbol{\Phi}=\mathbf{0}$, subject to boundary conditions.

(c) No boundary values are specified on the cathode but there is a time dependent relationship between surface potential and current density.

Basic numerical methods

Complicated car frame geometry and anode configurations are composed of thin metal and are thus modelled as an infinitely thin crack geometry. In order to treat thin parts, boundary integral equations for surface potential and surface flux are employed. By using the integral equation approach the normal derivative of the electric potential, see (b) above, is calculated directly, i.e., *without numerical differentiation*. The integral equations are approximated using a Galerkin method, which allows for a straightforward and mathematically correct analysis of the hypersingular flux equation essential for treating thin parts as a single surface. Accurate and efficient Galerkin singular integration algorithms based upon analytic integration have been developed by Gray [16]. Finally, a new proprietary time stepping algorithm is used to track the time evolution of the paint distribution (a).

The boundary integral equations are reduced to a finite system of linear equations by approximating the surface in terms of the elements defined by the nodal points of a computational grid, and then interpolating the surface potential and flux in terms of values at these nodes. This results in a matrix system

$$G[Y] = H[\Phi]$$

In this system, H and G are square matrices and [Φ] and [Y] are column vectors of the nodal values of potential and current. Taking into account the known boundary conditions, these linear equations must be solved simultaneously with the relationship between surface potential and current density (see (c) above).

The deployed commercial versions of the VPO™ EPD Simulator implement the above numerical methods in Fortran 90/95 adapted to parallel processing using the message passing interface, MPI.

Basic computational method

The main computational method is parallel processing, and in this paper the discussion focuses on the use the message passing interface (MPI) [3]. The computational problem decomposes into four main steps.

1. **Data distribution**: since each process needs access to the whole geometry the data files are read in and a copy stored on each process. We note here that one of the major users of this software is insisting on staying with MPI-1 [7]; we are developing using mpich2 [9] (but only using the MPI-1 subset). While MPI-2 [8] has facilities to allow each process to access the data files, we use the MPI-1 model where the master node reads the data and broadcasts it to all the other processes.

2. **Construction of the matrices**: The G and H matrices are both of order n where n is dependent on the total number of nodes defining both the car part (cathode) and the anodes and the tank.
   The elements of these arrays require the computation of a large number ($O(n^2)$) of two dimensional numerical integrals. The basic computational loop is of the form

   ```
   do i ∈ S
       do k ∈ T^k
           integral (element(i), element(j))
   ```

   where S is the set of all elements in the grid and T^k is a subset of these elements that is dependent on the type of integration taking place and the position of the element within the grid. Each element in S is involved in four different types of integral and the execution speed of these types differs by almost two orders of magnitude. All the integrals may be performed independently and each one only affects a maximum of six rows in each array. In addition, the updated rows are generally close together due to the way in which the grid generator assigns node numbers. Finally, the set of elements, T^k, only affects the column indices that are updated although these indices may take on a wide range and, for one type of integral, almost the whole row may be involved.

It therefore seemed sensible to assign contiguous blocks of whole rows to each process and to arrange the processes in a single column.

3. **Solution of the linear system**: This requires the factorization of G followed by the solution of n sets of linear equations where G is a general, dense matrix.
4. **Timestepping**: At this stage the main computational overhead is a sparse matrix / full vector multiply where the order of the sparse matrix is only dependent on the number of cathode (vehicle component) nodes. Generally this is substantially smaller than the order, n, of the original system. The sparsity level is not uniformly high but, given the size of the resultant systems and the large number of time steps required, taking account of sparsity is well worth the effort.

Timeline

Figure 1 shows an approximate timeline of the progression from initial development to full customer deployment. Almost from the beginning of development the testing and validation process began, getting progressively more complex as time passed.

Figure 1. Timeline showing development of VPO™ EPD

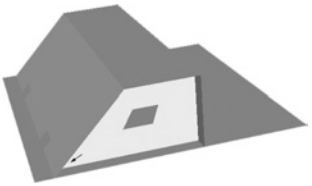
Testing and validation commenced with simple geometries because those are the ones that can be used for testing and experimenting in actual laboratory conditions, without impacting any manufacturing process. Simple laboratory experiments are the heart and key to the success of the VPO™ EPD simulator.

For a given electrocoat, the parameters used in simulating the electrocoating process can be determined, tested, and verified against laboratory experiments applied to simple geometries. These parameters are then incorporated into the mathematical model that is the core of the VPO™ EPD simulator, which can then be used in simulations using more complex geometries. When the supplied electrocoat changes, the laboratory experiments are repeated and the simulator model adjusted accordingly. This is a crucial aspect because it means that the model does not have to be retooled when it is applied to real vehicles, or when new vehicle models are produced, or when the conditions in the electrocoat tank change for any reason.

The following figures show some of the test geometries and their progression in complexity.

Mini-door

The geometry shown in the picture is extremely simple and is designed to reflect some of the characteristics found in real vehicle parts. The area indicated by the arrow is of particular interest to the thin-surface approach used in VPO™ EPD because at this junction a number of surfaces meet and have to be treated in a mathematically correct manner, proprietary techniques for which have been developed by BSSI.

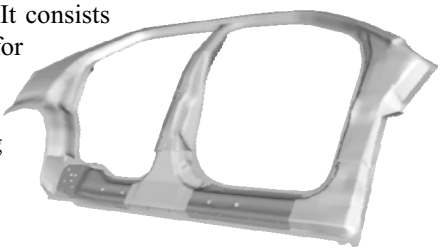


Rocker panel

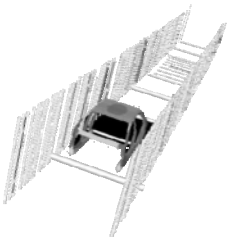
The rocker panel can be considered to be among the most problematical vehicle parts when it comes to electrocoating. It is a complicated structure with many interior parts that combine to form inaccessible areas where it is difficult for the e-coat paint to reach. If this happens then, the likelihood of corrosion problems in later life increases dramatically and this is the main reason why the use of VPO™ EPD is so important – to identify strategies that will avoid this situation.

Door Frame

A door-frame is the lion of vehicle parts. It consists of the rocker panel plus the supporting pillars for the vehicle's doors and wheels. It is a supremely majestic, complicated structure and the accurate modeling of electrocoating this vehicle part is a challenge, which when successful is immensely satisfying.



Larger part



The challenge to simulating a realistic operating environment not only comes from the size of the cathode – in this case a full vehicle frame – but also the tank and anodes in which the vehicle is transported. The tank shown in the picture is 35m long and contains 265 anodes, each one of which has to be meshed and placed in the correct position relative to the vehicle.

An indication of the relative sizes between the different geometries mentioned above can be seen in Table 1 where the number of nodes and elements required to model each one using VPO™ EPD are displayed.

Table 1. Example problem sizes for various car parts

	Nodes	Elements
Mini-door	2700	2758
Rocker panel	22906	21323
Door frame	34224	31163
Larger part	94156	86420

After running the gauntlet of complex geometry, and producing simulated parameter values to within limits set by the automotive company, the VPO™ EPD simulator can be applied in any vehicle program with confidence. Because of the underlying architecture employed, parallel processing using MPI, any vehicle geometry can be simulated within a time frame that is consistent with the automotive company's operational requirements.

Size and execution time are not the issues for discussion, rather the remainder of this paper addresses the challenges marked (**A**) and (**B**) in Figure 1.

7 (A) Implementation

The serial code mentioned in the timeline above began as a Fortran 77 prototype that was capable of running small geometries, such as flat plates and mini-doors. The code was developed on a 32-bit stand-alone PC using a commercial Fortran compiler [1].

A separate activity to write a commercial standard code began, from scratch, in 2003, and used Fortran 90 [2]. This move allowed a number of improvements, among which were:

1. The use of allocatable arrays and linked lists to ensure that memory requirements were kept to a problem dependent minimum.
2. The use of modules to encapsulate parts of the simulation, such as the simulation model. This allowed for easier changing of, and experimentation with, simulation models.
3. The execution speed decreased by approximately 25%.

The Fortran 90 serial code provided the basis for the commercial parallel code generated using MPI [3] together with associated numerical libraries such as ScaLAPACK [4].

Hardware

A requirement imposed by BSSI is that the parallel architecture used for simulations should, as far as possible, be composed from inexpensive, environmentally friendly, commodity hardware. In other words, if a cluster is to be used then consider one built from PCs that conform to this description. This requirement is not necessarily advocated by industry and in fact as later sections will show not adhering to the requirement imposes a hurdle to the vision for *virtual* paint operations that is the theme of this paper.

The initial implementation of the MPI version of the simulator used a 4-node, 32-bit architecture comprised of Intel Celeron 2.4 GHz machines, each with 2 GB of memory and interconnected with a gigabit network. This provided a great test environment, for a total cost of less than $3000, and enabled experimentation on small to medium sized geometries.

The major drawback with the 32-bit cluster was that it restricted the amount of memory available to 2 GB per processor. To accommodate larger memory requirements in a cost effective manner required the use of larger memory chips as well as an increase in the number of processors. This prompted a move to 64-bit architecture (to allow the addressing of larger amounts of memory) which in turn required a change of compiler technology from Lahey to PGI [15] since, at the time Lahey did not support full 64-bit addressing for individual arrays. The initial configuration for the 64-bit cluster was a 9-node cluster with each node comprising a 64-bit AMD Athlon 3800 processor, rated at 2.4 GHz, each node of which has 3 GB of memory and, as before, the nodes are interconnected by a gigabit network. In 2004, the total cost of this setup was less than $8,000 and has, until recently, been used to simulate the full range of vehicle geometries needed in operations at BSSI.

Source code

For ease of development, it was decided to have a single source code and to use fpp [5] to extract either the sequential or MPI versions of the software.

The decision to use fpp appeared to have several advantages, for example, sharing the majority of the source code between the two versions would mean that, in many cases, a single edit would correct or update both versions of the software. In practice the disadvantages far outweighed the advantages. As the code developed the use of macros increased and it became difficult to update the source with confidence, indeed it was often necessary to extract both versions to check that changes made were as intended. More importantly, it was not possible to use software tools, like the NagWare suite [6], on the master version. These problems contributed to a general downward drift in both the readability and testability of the code until they were finally separated.

MPI

The basic MPI framework is available from Argonne via the MPICH1 and MPICH2 implementations. Other implementations tend to be minor variations on the basic package and most of the time it is difficult to find out exactly what are the variations and what they might be useful for. Therefore, the starting point for our work was MPICH1 and then MPICH2.

Our experiences with MPI, starting out as complete beginners, have been mixed. The MPICH2 [9] implementation is freely available and is first class. The final product is efficient and has been error free. Our only minor gripe is that the installation guide listed a number of configuration options which appeared not to be either implemented or working. However the advice and help we received via email was excellent and we soon had a working installation.

We could have used far more assistance from support tools throughout the implementation and maintenance of the software. Debugging is an order of magnitude more difficult for a distributed code and a good profiler would have

enabled the early detection of inefficient parallel code and poor data distribution. As is to be expected a Google search for MPI support tools returns a large number of hits; what is required (and appears to be missing at the moment) is a definitive guide to which tools are (and are not) worth investing time and effort in.

We found the mpiP lightweight profiling library [10] very useful once we had learnt how to interpret the information it presented; we were impressed with both the documentation and the best-efforts support that were on offer. For a debugger TotalView [11] looks good but it is commercial (we were looking for open source) and thus Jumpshot [12] looks potentially useful.

There is also a problem with the migration to MPICH2. One of our industrial partners is staying with MPICH1; they have an efficient implementation and a large software base that still uses the earlier version. Despite the fact that MPICH2 contains all of MPICH1 they will neither upgrade nor make both versions available.

Compiler vendors have not, as yet, entered the specially tuned MPI installation market to any great degree. Few vendors offer pre-compiled versions of MPI itself and all that appears to be on offer are vanilla implementations with no special optimizations.

ScaLAPACK

In an MPI-based parallel environment, the *standard* package for solving dense linear systems using direct methods is ScaLAPACK, [4, 13], the parallel brother of LAPACK. Consequently, the ScaLAPACK routines *dgetrf* and *dgetrs* (see [4] for details) were used to implement step 3 in section 6 above. According to the ScaLAPACK manual [4] these routines for solving dense sets of linear equations work most effectively on a rectangular grid of processes with the data distributed using a block-cyclic distribution. Distributing by row slicing is likely to be inefficient and this is borne out by the execution times on a selection of test geometries. For example, the execution times obtained by solving the linear equations with n=7178 are given in Table 2.

Table 2. Execution times for ScaLAPACK routines *dgetrf* and *dgetrs* for solving GY = H with n = 7178

Grid	Time (seconds)
2 x 2	356
4 x 1	626
4 x 2	185
8 x 1	341

These timings clearly show the advantage of using a two dimensional grid even for a very small number of processes. The problem here is that a block cyclic distribution is sub-optimal for the parallel computation of the integral (step 2 in section 6 above). However, as the problem size grows the $O(n^3)$ complexity of the solution of the linear systems of equations dominates the computation and a move to a block cyclic distribution becomes far more appealing.

ScaLAPACK represents a monumental software effort; essentially it provides all the functionality of Lapack in distributed form with MPI forming one of the

available base level communication libraries. We suspect that the manual [4], while comprehensive, is now somewhat out-of-date as concerns example hardware and advice on how to tune the routines to deliver maximum throughput; a maintained web based manual would be an advantage here.

The downside of ScaLAPACK is that it is based on a number of layers that have remained unloved for the better part of a decade and the whole package requires considerable effort to install. The installation document is very old as are the Makefiles that are included (many for platforms that only now exist in museums). While some of the system dependent configuration settings can be obtained easily by running a number of small programs provided in the distribution, others are obscure. A mechanism that would allow successful installers to donate their makefiles and/or complete libraries would also be extremely helpful. Finally there is no easy way to extract and build a subset of the ScaLAPACK routines – for example, VPO™ EPD only wanted to call two top-level routines but the building of a call tree and the extraction of the routines in that tree was and is a non-trivial task.

The learning curve for ScaLAPACK is quite steep. Solving some problems was certainly not helped by the opacity of some of the error messages. For example, when using $P \times 1$ grid of processes to solve a general, square linear system of order n, it seems logical to set the size of the local blocks to $mb = \lceil n / P \rceil$ and nb to n (since each process will be holding all the columns). This results in an illegal value for nb error; which could only be resolved by examining the source code where it was discovered that nb needs to be equal to mb.

Finally, there are a number of support routines that act in rather peculiar ways; for example, the subroutine INFOG2L which *computes the starting local row and column indices and process row and column indices of a global element of a block-cyclically distributed matrix*. Put another way, if we wish to find the value of a_{ij} calling INFOG2L with the values i and j will return k, l, r and c such that element a_{ki} on the process in position (r,c) in the grid holds the required value. Now, it would appear safe to assume that, given i and j define a unique element, INFOG2L would return the same result on every process. However, this assumption is wrong because the routine requires the user to state which process is calling the routine so that it can return the wrong answer on all the processes except the one holding the element. It is not clear why this decision was made, especially as it is actually more efficient to compute the correct answer on all processes.

Such quirky behaviour is strange but the problem does not end there. At least one other implementer, IBM [14], has included this routine in their parallel ESSL library and they even provide an example which shows how the wrong answer is computed by all processes except one; consistency at all costs? We need to take care not to spread such peculiar behaviours and this requires some form of active maintenance.

Summary

For those of us who experienced the heyday of numerical software development in the 70s and 80s, it is depressing to discover that many of the hard earned lessons from that era appear to have been forgotten. As an example:

> *One of us (PG) was recently interviewed for over an hour by a very pleasant young man at Intel who turned out to be completely ignorant of the existence of NAG or IMSL, and who had never heard of the work on floating point arithmetic by Kahan and the*

IFIP Working Group on Numerical Software, and who, needless to say, was blissfully unaware of the extent to which that legacy of work is today present in the very good Intel Math Library.

In the world of cluster programming that level of ignorance appears to be very common and one feels cast adrift into a parallel world that seems to exist without any knowledge of, or influence from, the body of numerical software that existed before and after the BLAS; for this is the one piece of software that is peddled by the vendors as the general panacea for all one's numerical problems. When it comes to implementing high quality mathematical software for a parallel environment using MPI, the current situation is woeful. The present state of the open source MPI implementations certainly require some tuning and perhaps this is to be expected – if an out-of-the-box solution is required then some would say that commercial software should be used instead of open-source. This might very well be the case except at this stage it is very difficult to determine the differences between the open-source versions of MPI and those available from commercial vendors, except the price.

The exercise of constructing high quality mathematical software suitable and robust enough for commercial applications using open source / freely available software is possible but requires a great deal of effort and perseverance. The expenditure of this effort is tempered by the thought that many other users have, most likely, already trodden the same path and what a good idea it would be if such duplication of work could be avoided. It really is time that the developers of packages like ScaLAPACK. PBLAS, etc took a lead from the thriving open source community currently producing high quality software using the facilities provided by, for example, SourceForge. Here projects are kept alive and up-to-date either by the original authors or by other interested (and knowledgeable) parties rather than being allowed to languish unloved on the original authors web or ftp site. Sites like SourceForge provide a central, easy to find location where users can download the most recent version of the software, report bugs, donate Makefiles and ask questions; software improves by expanding the user community and gaining feedback.

It is possible with determination, and a strong and encouraging *champion* of the software, to produce quality industrial strength software. Real vehicle parts, including full frame automobiles and trucks, are now straightforward to simulate using the parallel version of VPO™ EPD in an elapsed time that makes experimentation with different designs practical. The end result is a software product that enables significant reduction in costs, enhances good environmental practices, and improves the quality of the end product, namely the vehicles on the road.

7 (B) Incorporating into production

This section describes the most interesting aspect of this work because the *value* of the work depends on how successful it is to engage the workforce in using the new *virtual* toolset. If we fail in this regard then, it doesn't really matter how clever we have been in solving complex simulation problems since end users will not use the tools, for one reason or another. And one lesson we have learned from many years of working with large corporations: it doesn't always follow that it is the fault of the software that is the reason why users do not use it, for they have other influences that

may be more powerful than the ones you thought of in designing the software. Those influences boil down to one word: people. People in all levels of the organization will have and will always want a say, justified or not, in the use of the software – and that is the problem that has to be addressed and solved if the software is to be a success. People come and go but the software must remain for the good of the company.

Skills - *engagement*

Complex manufacturing computational problems are rarely solved by the push of a button, and even if they were it would not be sensible to avoid human intervention at some point, if only to examine results and messages stemming from the computation.

Currently, industrial operational environments rely heavily on people with hands-on experience that it is crucial to utilize in the computer simulation of complex processes. This experience needs to be captured and used in order to augment decision making processes encoded in software.

For computer simulations to be more effective than physical prototyping, software must be adapted to the existing skill base rather than the other way around:

(a) Software should be designed to use the knowledge and experience of users, because the processes being simulated cannot normally be performed effectively without their input and guidance.

(b) Users should feel comfortable in using the software – as far as possible the graphical controls, labels, and functions of user interfaces should reflect terminology and processes that are familiar to users from their working experience in the plant.

(c) Users must have confidence in the software – this can only be gained over time and by comparing results from the virtual process with those from the actual events.

(d) Generating what-if scenarios and experimenting to see the effects of different hypotheses will produce substantial quantities of output that needs to be analyzed, interpreted, and distilled into summaries for management to make the correct decisions. The existing workforce may be equipped to do these tasks but their job will be easier and more effective if the software contains tools to mine the data and extract trends and patterns.

Only when all of these aspects are addressed successfully can the real benefits of virtual processing be obtained: a workforce that is *engaged* in the virtual process and that has the confidence to experiment with the software. Only then will they gain an understanding of what is possible and what will improve the end product and, in the process of achieving this, they will improve their own job satisfaction.

The timeline in section 5 shows that this process for VPO™ EPD began in 2003 and continues as the software evolves - *real* software continues to evolve as feedback and experience are gained from people.

Ken Kennedy

I think we will need very general API specifications that go beyond the current type matching in component integration systems. Additional useful information might be size of data arrays or files, shape, etc. Knowing this information might permit the scheduler to pick the most efficient service for a given invocation context. So the short answer is that good API specifications will be very important both in ensuring correctness and efficiency.

Mladen Vouk

Well defined, open and information rich APIs are extremely important in ensuring interoperability of workflows and understanding of the semantics behind different components and services. Without that we are heading towards another mixed-language programming problem that may create "stove-pipes" and isolation between communities instead of promoting the opposite.

Questioner: Keith Jackson

What role does semantic information play in a component architecture? What kinds of semantic information should a service expose?

Dennis Gannon

Semantic information plays an extremely important role in service composition. It can tell us if two services can truly be composed. Just because they are type compatible, it does not mean that the output of one service is really suitable for the input of another. Semantic information goes well beyond what you can express in standard type theories. For example, it can tell you about the dynamic range of a service, information about its time complexity based on input values, information about resource requirements for a given input.

Ken Kennedy

There are two kinds of semantic information: that which is of use to the developer and that which can be used by the compiler or program integration system to improve performance. From the compiler perspective (my specialty) we need to know when two sequences of component invocations are equivalent in a certain calling context so that we can optimize the program to that context. From that perspective, (asymptotic) running time of the underlying algorithm is important semantic information.

Mladen Vouk

I believe that without appropriate semantic information it will not be possible to move to large scale, diverse, hierarchical and verifiable, as well as fault-tolerant, workflow solutions. There are probably be at least two major categories of semantic information: a) canonical (e.g., that associated with supporting sciences such as mathematics, statistics, computer science, information technology, ...) related to such elements as data and data structures, operations on those structures, related events and behaviors, constraints and assumptions, re-usable functions, process models, etc., etc.;

and b) domain specific (which may cover a wide range of properties, from domain specific data structures and operations to performance, reliability and security requirements, to domain specific process models and dos and don'ts etc.). The trick is to have rich enough and flexible enough models and interface specification options to allow interfaces to automatically exchange, match, and assess mutual semantic and other interface information and provide self-verification and validation.

Questioner: Anne Trefethen

How do we get community agreement on the semantics?

Dennis Gannon

First find a common meta-language to describe the semantics of components. Second, we can work on basic interoperability. One can start with a set of domains where this is a concern, for example, biomedical apps. This is an area where three or four of the major workflow tools are in current use and many of the same services are used. One could convene a meeting to look at interoperability and semantics.

Ken Kennedy

It is very difficult to agree on a language in which to specify semantics, because the language needs to support the reasoning styles that are needed by particular tools. For example, in our work we wanted to be able to substitute a loop around a get of an out-of-core data array element with a get of a whole row or column. There is no way to do that without having deep compiler concepts like dependence built into the language. In other words, different systems will need different languages to support their activities.

I should note that my group is working with a developer who wants to replace much of the functionality of full MPI with a smaller efficient set of primitives that could be combined to provide all the functionality of the current MPI. In other words, they want a compiler that can replace MPI calls with provably equivalent calls to the lower-level library. (Here Gropp comments that this was not done in the original because it was so tightly integrated with the underlying platform). My comment in response: The platform would be an additional (silent) parameter to the component library.

Mladen Vouk

I think that a fairly broad meeting, relatively soon --- probably within a year --- that would involve both open source and closed source workflow communities, developers and stake holders is an essential step in bootstrapping workflow related interoperability, interfaces, protocols, etc.

Bill Gropp

One example was the process that started the MPI Forum. Ken Kennedy called a workshop of many of the developers of message passing systems to answer the question "is it possible to standardize message passing for scientific computing?". The result of this workshop was a commitment by many of those developers to consider a possible standard. Mladen's talk

listed many open source workflow system; perhaps a similar workshop would identify, if not one area of standardization, a much smaller number of topics, some of which may be able to develop standards or standard practices. It may also provide a way to develop standards for the interoperation of workflow tools, or standards for sharing representations of workflows.

PART 2

APPLICATION EXPERIENCE

R. Boisvert, Session Chair; B. Einarsson, Discussant

P. Gaffney & T. Hopkins: *Virtual Manufacturing - A Vision for Virtual Paint Operations*

D. Schissel: *Service-oriented Computation in Magnetic Fusion Research* (Abstract)

D. Walker: *Lessons Learned from the GECEM Project*

Panel Discussion

Virtual Manufacturing
A Vision for Virtual Paint Operations

Patrick Gaffney[1], Tim Hopkins[2]
[1] BSSI AS, Post Box 18, Nyborg, Heiane 4, 5879 Bergen, Norway
pat@bssi-tt.com,
WWW home page: http://www.bssi-tt.com
[2] Computing Laboratory, University of Kent
Canterbury, Kent, CT2 7NF, UK
t.r.hopkins@kent.ac.uk,
WWW home page: http://www.cs.kent.ac.uk/people/staff/trh/index.html

Abstract. The present paper deals less with Grid computing than with other aspects of this conference, namely numerical software. The goal of our work is to use mathematical modeling and computer simulation to avoid building non-production prototypes in manufacturing processes. Of particular interest is to eliminate prototypes associated with *painting processes* used in the automotive industry. Since painting processes are difficult to simulate accurately, the numerical software employed in the computational simulation is quite complex and requires substantial computing power to get a decent result within the timeframes of an industrial operational environment. On the other hand, the rewards for getting it right go straight to a company's bottom line: eliminating prototypes leads to tangible savings in terms of paint and materials, man power, energy costs, reduced warrantee costs, improved operational processes, and improved quality of the finished product. Getting it right therefore, is the subject of this paper.

1 Introduction

To most people painting a car conjures up pictures, from Discovery Channel and elsewhere, of robots spray painting cars on an assembly line. Although this is the way most vehicles end up by being painted it is by no means the whole story. For that, one needs to go back and look at the processes that prepare the metal of the car body so that it can become the final shiny product seen on the road.

Please use the following format when citing this chapter:

Gaffney, P., Hopkins, T., 2007, in IFIP International Federation for Information Processing, Volume 239, Grid-Based Problem Solving Environments, eds. Gaffney, P. W., Pool, J.C.T., (Boston: Springer), pp. 71-88.

That shiny product is the reflection of a sequence of processes that prepare the vehicle to sustain the effects from inclement weather - corrosion, while providing a base coating of the metal that enhances the quality of the finished vehicle.

The processes are well known to most people, although they may not connect them to being applied to cars on the road. Broadly speaking the processes consists of cleaning the metal of the vehicle body, preparing the metal to receive paint, applying a protective coating - electrocoating, rinsing, and baking. The protective coating is organic and the baking process is the final act that melts the resin polymer in the coating to provide the required corrosion protection.

It is crucial to get this sequence of painting processes right because errors at any stage have a detrimental impact on the bottom line of the automotive company, and some errors will become apparent to the unhappy owner. Errors will occur for a variety of reasons: maybe the cleaning was not effective enough, maybe some of the protective coating remains in puddles in the vehicle as it enters the oven, maybe the protective coating did not adequately cover all areas of the vehicle so that hard to reach areas do not have enough coating or maybe they do not have any. The list is almost endless, and each one of these items is significantly costly to fix.

In an attempt to understand the causes of errors and identify suitable remedial action, companies have traditionally experimented with physical prototypes of new vehicles by subjecting them to the same painting processes. However, experimentation and testing on physical prototypes is necessarily limited to the number of prototypes that can be built, not to mention the costs incurred in terms of manpower, materials, time, and the fact that the physical operating conditions, i.e. the paint tanks and cure ovens, must be used for the experiments, and therefore the normal manufacturing process must be interrupted, which in itself is an unacceptable cost to bear.

The *hope* of uncovering better ways of ensuring success in the application of the painting processes mentioned above is a vain one if all that is done is to use *physical* trial and error experimentation. On the other hand the *hope* becomes a reality if one can use computers to replace the *physical* nature of the experimentation. To do this requires that the painting processes must be simulated on a computer and this simulation requires detailed and accurate mathematical models of the underlying physics and chemistry of each painting process.

2 Virtual Manufacturing

Virtual manufacturing is the application of computer software to simulate a new product and the processes required to take that product from a design concept through to its actual manufacture using physical materials. Software simulation allows problems to be identified and remedied before the fabrication of the product begins.

Virtual manufacturing is not a new concept; it has been around for several decades, promoted, justifiably, by the use of finite element techniques to model complex geometries thereby expanding the range of products that are susceptible to this process.

The finite element method is a computational method to solve complex three-dimensional partial differential equations, especially those where it is important to know what happens in the *interior* of the solution domain.

Another technique involves recasting the partial differential equations as integral equations and then employing Boundary Integral Methods for their solution. For this recasting to be possible, a fundamental solution of the partial differential equations, a so-called Green's function, must exist. The set of industrial problems for which a Green's function exists is not as large as the set of problems that can be solved by the finite element method. However, for many industrial problems in engineering, it is possible to determine a Green's function and for these cases boundary integral methods are an alternative solution technique to the finite element method. Boundary integral methods are useful when it is important to know what happens on the *boundary* of a closed region as opposed to the *interior*. There are many industrial processes where the solution is only required on the boundary and for these cases, boundary integral methods are often the only practical approach because including the interior of the region makes a solution by the finite element method computationally intractable. Three particular processes where the solution is required on the boundary are:

- Electrochemical processes
- Transient heat flow in an oven
- Crack propagation

Thus, the boundary integral technique further extends the range of problems that can be addressed by virtual manufacturing. There are two main aspects of virtual manufacturing though: the simulation of the new product *and* the simulation of the processes. Finite element and boundary integral techniques extend the range of products that can be simulated but it still remains to simulate the processes themselves and the processes mentioned above are non-trivial.

Recent advances in the mathematical techniques underpinning the boundary integral method combined with advances in computer hardware and software mean that virtual manufacturing can be extended to simulating electrochemical processes especially those involved in the painting processes described in the previous section.

3 Characteristics of Painting Processes

Although painting processes are simple to describe, the resultant three-dimensional mathematical models of them are complex and range from potential theory, through fluid flow and computational fluid dynamics, to elasticity, to name but a few. Nevertheless, the overriding most important consideration of any simulation of these processes is the inclusion of *time*. Steady-state descriptions of these processes are completely inadequate as a basis for *predicting* what will happen in the physical process. Operators need to know what happens at any instant of a process and therefore accurate time-dependent solutions are a necessity for

employing computational simulation as a replacement for testing and experimentation using physical prototypes.

Computational simulations of painting processes are time dependent and for operational reasons a true time-stepping solution method must be employed. Solution domains involve complex geometries consisting of automobiles, paint tanks, cleaning tanks, and cure ovens. A *virtual* toolset that replaces *physical* trial-and-error testing thus requires incorporating all of these items and human experience into software that can be used easily by non-computer specialists, at both ends of the manufacturing chain: the upstream users at the design stage and the downstream users at the manufacturing stage. Only when this is done properly, and the software is validated, will users be convinced to switch to virtual tools as part of their everyday work environment.

4 Validation

The previous paragraphs help to characterize the nature of the software that industrial operational environments need. At the heart of the simulation software are methods and techniques that are traditionally described as *numerical software*. It is a long strenuous path to take that numerical software and form a system that commercial users will feel comfortable with. It is not only the look-and-feel of the software that is important it is the confidence that using the software will help them to understand better the physical manufacturing processes that they are really interested in. This means effectively that the users are confident of the results obtained from the simulation, in other words the software has been validated to produce accurate results that can be relied upon. This level of confidence is often not present in what many people regard as *numerical software*.

The testing regime, to which operational software of the functionality and calibre described above is subjected, is rigorous in the extreme. The process begins after the initial software has been written and takes the form of unit testing on geometries that exhibit some of the behavior expected in real automobiles and that can be verified by simple laboratory experiments conducted by the automobile company. Even these tests are more rigorous than those used to verify software published in ACM TOMS for example. As development proceeds, test geometries evolve in complexity until at a certain stage actual vehicle parts and bodies are used to stress test the software. This process is monitored closely by end users who actually reproduce the results by running the software themselves on their own machines and compare results with measurements taken of physical prototypes that have been stripped down and measured after painting.

These testing and validation procedures contribute to confidence building and numerical software developers should take heed and learn from them. For example, in addition to testing on increasingly complex models the process benefits from the active participation of an *independent champion* of the software, namely the automobile user. As development of the software proceeds it is in the interests of the automobile company that the software succeeds and therefore aberrations and errors that come to light from their testing are immediately brought to the attention of the

developers. In the course of developing the virtual toolset described in this paper, approximately 80% of the time has been spent on testing, verification, and debugging.

5 Virtual Paint Operations™ (VPO™)

VPO™ is the name of a family of software products and services developed by BSSI over the last 10 years according to the descriptions given in the previous sections, where the champion in this case is Ford Motor Company in North America. The software addresses the simulation of the painting processes: electrocoating, drainage, baking, and the elimination of air pockets. Each simulator in the toolset is a time stepping *predictor* that enables the operator to access the results of the simulation at any moment in time. Thus, for example, it is possible for an operator to monitor the oven baking process and see how the predicted solution behaves at every second the automobile is in the oven. In this way, the company can develop operating strategies that enable them to take account of changes in operations, for example different oven loads, without interrupting daily operations in the actual oven.

To address look-and-feel, the software is operated through a unified interface that provides access to all the simulators from one graphical user interface. The painting processes described previously are not independent of each other. For example, doing a poor job of electrocoating has an affect on the quality that can be expected from the cure oven. Consequently, the emphasis of the interface design is in the inter-operability of the simulators. Being able to simulate different what-if scenarios that *couple* baking, drainage, and electrocoating simulators - by passing data and results backwards and forwards between the different simulators - is an emphasis that stems from watching how users perform their daily tasks. The tasks are not independent and therefore as far as possible it should be possible to use the simulators in a similar way.

Operational environments, especially manufacturing processes that produce millions of items per year, have a strong emphasis on timeliness: operations and decisions need to be taken quickly and in a time that does not hamper the manufacturing process or delay the throughput of items (vehicles in this case). Consequently, it is very important that the simulators produce results quickly – not necessarily the *final* results but intermediate ones. For example, in a simulation of the electrocoating process of a vehicle the VPO™ EPD code can produce output at every $1/1000^{th}$ of a second and this output is available to the user for analysis as soon as it is produced. In this way, the user can take decisions without having to wait for the simulation to finish.

The single most effective way of handling timeliness, and the production of results rapidly, is to use parallel computing, and in today's environments clusters are the preferred architecture.

Of course all of the above – real-time visualization of results, coupling, inter-operability of simulators, and computational steering - assumes that information, in the form of intermediate results from the simulator, is readily available to the user,

which it is from the VPO™ System operating in a Grid-like environment, but it is not when the system is constrained to work in batch environments.

Of the painting processes discussed above, the one that is the most complex to simulate and computationally intensive is electrocoating.

6 Electrocoat Simulation – VPO™ EPD

The process to apply rust-preventing coatings to automobiles must ensure that all parts of the vehicle have an adequate minimum build of paint. Currently, the main method used to minimize salt spray induced corrosion involves application of an epoxy-based urethane coating using an electrodeposition technique called electrocoating. Car parts are suspended from a gantry, lowered, and transported through a tank filled with the urethane paint called electrocoat or e-coat. Each part is treated as a cathode and by applying a voltage to the anodes; the paint adheres to the car part.

Basic mathematical model

(a) The rate that paint is deposited on the cathode is a function of the local current density **j**:

$$\mathbf{d/dt(L_F(t)) = \beta * j}$$

$\mathbf{L_F}$ is the thickness of the paint and $\boldsymbol{\beta}$ is the current efficiency.

(b) The current density **j**, as a function of position over the cathode surface, is obtained from the *normal derivative* of the electric potential $\boldsymbol{\Phi}$, which is determined by solving the 3D Laplace equation $\nabla^2\boldsymbol{\Phi}=\mathbf{0}$, subject to boundary conditions.

(c) No boundary values are specified on the cathode but there is a time dependent relationship between surface potential and current density.

Basic numerical methods

Complicated car frame geometry and anode configurations are composed of thin metal and are thus modelled as an infinitely thin crack geometry. In order to treat thin parts, boundary integral equations for surface potential and surface flux are employed. By using the integral equation approach the normal derivative of the electric potential, see (b) above, is calculated directly, i.e., *without numerical differentiation*. The integral equations are approximated using a Galerkin method, which allows for a straightforward and mathematically correct analysis of the hypersingular flux equation essential for treating thin parts as a single surface. Accurate and efficient Galerkin singular integration algorithms based upon analytic integration have been developed by Gray [16]. Finally, a new proprietary time stepping algorithm is used to track the time evolution of the paint distribution (a).

The boundary integral equations are reduced to a finite system of linear equations by approximating the surface in terms of the elements defined by the nodal points of a computational grid, and then interpolating the surface potential and flux in terms of values at these nodes. This results in a matrix system

$$G[Y] = H[\Phi]$$

In this system, H and G are square matrices and [Φ] and [Y] are column vectors of the nodal values of potential and current. Taking into account the known boundary conditions, these linear equations must be solved simultaneously with the relationship between surface potential and current density (see (c) above).

The deployed commercial versions of the VPO™ EPD Simulator implement the above numerical methods in Fortran 90/95 adapted to parallel processing using the message passing interface, MPI.

Basic computational method

The main computational method is parallel processing, and in this paper the discussion focuses on the use the message passing interface (MPI) [3]. The computational problem decomposes into four main steps.

1. **Data distribution**: since each process needs access to the whole geometry the data files are read in and a copy stored on each process. We note here that one of the major users of this software is insisting on staying with MPI-1 [7]; we are developing using mpich2 [9] (but only using the MPI-1 subset). While MPI-2 [8] has facilities to allow each process to access the data files, we use the MPI-1 model where the master node reads the data and broadcasts it to all the other processes.

2. **Construction of the matrices**: The G and H matrices are both of order n where n is dependent on the total number of nodes defining both the car part (cathode) and the anodes and the tank.
   The elements of these arrays require the computation of a large number ($O(n^2)$) of two dimensional numerical integrals. The basic computational loop is of the form

   ```
   do i ∈ S
      do k ∈ T^k
         integral (element(i), element(j))
   ```

   where S is the set of all elements in the grid and T^k is a subset of these elements that is dependent on the type of integration taking place and the position of the element within the grid. Each element in S is involved in four different types of integral and the execution speed of these types differs by almost two orders of magnitude. All the integrals may be performed independently and each one only affects a maximum of six rows in each array. In addition, the updated rows are generally close together due to the way in which the grid generator assigns node numbers. Finally, the set of elements, T^k, only affects the column indices that are updated although these indices may take on a wide range and, for one type of integral, almost the whole row may be involved.

It therefore seemed sensible to assign contiguous blocks of whole rows to each process and to arrange the processes in a single column.

3. **Solution of the linear system**: This requires the factorization of G followed by the solution of n sets of linear equations where G is a general, dense matrix.
4. **Timestepping**: At this stage the main computational overhead is a sparse matrix / full vector multiply where the order of the sparse matrix is only dependent on the number of cathode (vehicle component) nodes. Generally this is substantially smaller than the order, n, of the original system. The sparsity level is not uniformly high but, given the size of the resultant systems and the large number of time steps required, taking account of sparsity is well worth the effort.

Timeline

Figure 1 shows an approximate timeline of the progression from initial development to full customer deployment. Almost from the beginning of development the testing and validation process began, getting progressively more complex as time passed.

Figure 1. Timeline showing development of VPO™ EPD

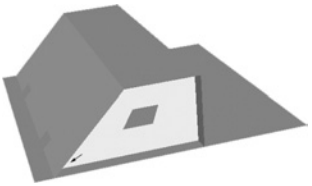
Testing and validation commenced with simple geometries because those are the ones that can be used for testing and experimenting in actual laboratory conditions, without impacting any manufacturing process. Simple laboratory experiments are the heart and key to the success of the VPO™ EPD simulator.

For a given electrocoat, the parameters used in simulating the electrocoating process can be determined, tested, and verified against laboratory experiments applied to simple geometries. These parameters are then incorporated into the mathematical model that is the core of the VPO™ EPD simulator, which can then be used in simulations using more complex geometries. When the supplied electrocoat changes, the laboratory experiments are repeated and the simulator model adjusted accordingly. This is a crucial aspect because it means that the model does not have to be retooled when it is applied to real vehicles, or when new vehicle models are produced, or when the conditions in the electrocoat tank change for any reason.

The following figures show some of the test geometries and their progression in complexity.

Mini-door

The geometry shown in the picture is extremely simple and is designed to reflect some of the characteristics found in real vehicle parts. The area indicated by the arrow is of particular interest to the thin-surface approach used in VPO™ EPD because at this junction a number of surfaces meet and have to be treated in a mathematically correct manner, proprietary techniques for which have been developed by BSSI.

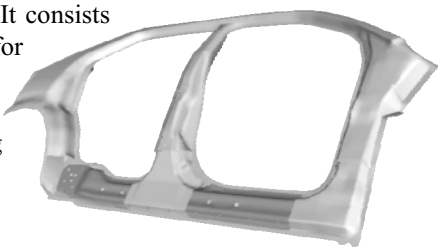


Rocker panel

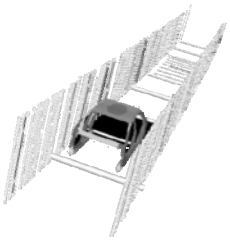
The rocker panel can be considered to be among the most problematical vehicle parts when it comes to electrocoating. It is a complicated structure with many interior parts that combine to form inaccessible areas where it is difficult for the e-coat paint to reach. If this happens then, the likelihood of corrosion problems in later life increases dramatically and this is the main reason why the use of VPO™ EPD is so important – to identify strategies that will avoid this situation.

Door Frame

A door-frame is the lion of vehicle parts. It consists of the rocker panel plus the supporting pillars for the vehicle's doors and wheels. It is a supremely majestic, complicated structure and the accurate modeling of electrocoating this vehicle part is a challenge, which when successful is immensely satisfying.



Larger part



The challenge to simulating a realistic operating environment not only comes from the size of the cathode – in this case a full vehicle frame – but also the tank and anodes in which the vehicle is transported. The tank shown in the picture is 35m long and contains 265 anodes, each one of which has to be meshed and placed in the correct position relative to the vehicle.

An indication of the relative sizes between the different geometries mentioned above can be seen in Table 1 where the number of nodes and elements required to model each one using VPO™ EPD are displayed.

Table 1. Example problem sizes for various car parts

	Nodes	Elements
Mini-door	2700	2758
Rocker panel	22906	21323
Door frame	34224	31163
Larger part	94156	86420

After running the gauntlet of complex geometry, and producing simulated parameter values to within limits set by the automotive company, the VPO™ EPD simulator can be applied in any vehicle program with confidence. Because of the underlying architecture employed, parallel processing using MPI, any vehicle geometry can be simulated within a time frame that is consistent with the automotive company's operational requirements.

Size and execution time are not the issues for discussion, rather the remainder of this paper addresses the challenges marked (**A**) and (**B**) in Figure 1.

7 (A) Implementation

The serial code mentioned in the timeline above began as a Fortran 77 prototype that was capable of running small geometries, such as flat plates and mini-doors. The code was developed on a 32-bit stand-alone PC using a commercial Fortran compiler [1].

A separate activity to write a commercial standard code began, from scratch, in 2003, and used Fortran 90 [2]. This move allowed a number of improvements, among which were:

1. The use of allocatable arrays and linked lists to ensure that memory requirements were kept to a problem dependent minimum.
2. The use of modules to encapsulate parts of the simulation, such as the simulation model. This allowed for easier changing of, and experimentation with, simulation models.
3. The execution speed decreased by approximately 25%.

The Fortran 90 serial code provided the basis for the commercial parallel code generated using MPI [3] together with associated numerical libraries such as ScaLAPACK [4].

Hardware

A requirement imposed by BSSI is that the parallel architecture used for simulations should, as far as possible, be composed from inexpensive, environmentally friendly, commodity hardware. In other words, if a cluster is to be used then consider one built from PCs that conform to this description. This requirement is not necessarily advocated by industry and in fact as later sections will show not adhering to the requirement imposes a hurdle to the vision for *virtual* paint operations that is the theme of this paper.

The initial implementation of the MPI version of the simulator used a 4-node, 32-bit architecture comprised of Intel Celeron 2.4 GHz machines, each with 2 GB of memory and interconnected with a gigabit network. This provided a great test environment, for a total cost of less than $3000, and enabled experimentation on small to medium sized geometries.

The major drawback with the 32-bit cluster was that it restricted the amount of memory available to 2 GB per processor. To accommodate larger memory requirements in a cost effective manner required the use of larger memory chips as well as an increase in the number of processors. This prompted a move to 64-bit architecture (to allow the addressing of larger amounts of memory) which in turn required a change of compiler technology from Lahey to PGI [15] since, at the time Lahey did not support full 64-bit addressing for individual arrays. The initial configuration for the 64-bit cluster was a 9-node cluster with each node comprising a 64-bit AMD Athlon 3800 processor, rated at 2.4 GHz, each node of which has 3 GB of memory and, as before, the nodes are interconnected by a gigabit network. In 2004, the total cost of this setup was less than $8,000 and has, until recently, been used to simulate the full range of vehicle geometries needed in operations at BSSI.

Source code

For ease of development, it was decided to have a single source code and to use fpp [5] to extract either the sequential or MPI versions of the software.

The decision to use fpp appeared to have several advantages, for example, sharing the majority of the source code between the two versions would mean that, in many cases, a single edit would correct or update both versions of the software. In practice the disadvantages far outweighed the advantages. As the code developed the use of macros increased and it became difficult to update the source with confidence, indeed it was often necessary to extract both versions to check that changes made were as intended. More importantly, it was not possible to use software tools, like the NagWare suite [6], on the master version. These problems contributed to a general downward drift in both the readability and testability of the code until they were finally separated.

MPI

The basic MPI framework is available from Argonne via the MPICH1 and MPICH2 implementations. Other implementations tend to be minor variations on the basic package and most of the time it is difficult to find out exactly what are the variations and what they might be useful for. Therefore, the starting point for our work was MPICH1 and then MPICH2.

Our experiences with MPI, starting out as complete beginners, have been mixed. The MPICH2 [9] implementation is freely available and is first class. The final product is efficient and has been error free. Our only minor gripe is that the installation guide listed a number of configuration options which appeared not to be either implemented or working. However the advice and help we received via email was excellent and we soon had a working installation.

We could have used far more assistance from support tools throughout the implementation and maintenance of the software. Debugging is an order of magnitude more difficult for a distributed code and a good profiler would have

enabled the early detection of inefficient parallel code and poor data distribution. As is to be expected a Google search for MPI support tools returns a large number of hits; what is required (and appears to be missing at the moment) is a definitive guide to which tools are (and are not) worth investing time and effort in.

We found the mpiP lightweight profiling library [10] very useful once we had learnt how to interpret the information it presented; we were impressed with both the documentation and the best-efforts support that were on offer. For a debugger TotalView [11] looks good but it is commercial (we were looking for open source) and thus Jumpshot [12] looks potentially useful.

There is also a problem with the migration to MPICH2. One of our industrial partners is staying with MPICH1; they have an efficient implementation and a large software base that still uses the earlier version. Despite the fact that MPICH2 contains all of MPICH1 they will neither upgrade nor make both versions available.

Compiler vendors have not, as yet, entered the specially tuned MPI installation market to any great degree. Few vendors offer pre-compiled versions of MPI itself and all that appears to be on offer are vanilla implementations with no special optimizations.

ScaLAPACK

In an MPI-based parallel environment, the *standard* package for solving dense linear systems using direct methods is ScaLAPACK, [4, 13], the parallel brother of LAPACK. Consequently, the ScaLAPACK routines *dgetrf* and *dgetrs* (see [4] for details) were used to implement step 3 in section 6 above. According to the ScaLAPACK manual [4] these routines for solving dense sets of linear equations work most effectively on a rectangular grid of processes with the data distributed using a block-cyclic distribution. Distributing by row slicing is likely to be inefficient and this is borne out by the execution times on a selection of test geometries. For example, the execution times obtained by solving the linear equations with n=7178 are given in Table 2.

Table 2. Execution times for ScaLAPACK routines *dgetrf* and *dgetrs* for solving GY = H with n = 7178

Grid	Time (seconds)
2 x 2	356
4 x 1	626
4 x 2	185
8 x 1	341

These timings clearly show the advantage of using a two dimensional grid even for a very small number of processes. The problem here is that a block cyclic distribution is sub-optimal for the parallel computation of the integral (step 2 in section 6 above). However, as the problem size grows the $O(n^3)$ complexity of the solution of the linear systems of equations dominates the computation and a move to a block cyclic distribution becomes far more appealing.

ScaLAPACK represents a monumental software effort; essentially it provides all the functionality of Lapack in distributed form with MPI forming one of the

available base level communication libraries. We suspect that the manual [4], while comprehensive, is now somewhat out-of-date as concerns example hardware and advice on how to tune the routines to deliver maximum throughput; a maintained web based manual would be an advantage here.

The downside of ScaLAPACK is that it is based on a number of layers that have remained unloved for the better part of a decade and the whole package requires considerable effort to install. The installation document is very old as are the Makefiles that are included (many for platforms that only now exist in museums). While some of the system dependent configuration settings can be obtained easily by running a number of small programs provided in the distribution, others are obscure. A mechanism that would allow successful installers to donate their makefiles and/or complete libraries would also be extremely helpful. Finally there is no easy way to extract and build a subset of the ScaLAPACK routines – for example, VPO™ EPD only wanted to call two top-level routines but the building of a call tree and the extraction of the routines in that tree was and is a non-trivial task.

The learning curve for ScaLAPACK is quite steep. Solving some problems was certainly not helped by the opacity of some of the error messages. For example, when using $P \times 1$ grid of processes to solve a general, square linear system of order n, it seems logical to set the size of the local blocks to $mb = \lceil n / P \rceil$ and nb to n (since each process will be holding all the columns). This results in an illegal value for nb error; which could only be resolved by examining the source code where it was discovered that nb needs to be equal to mb.

Finally, there are a number of support routines that act in rather peculiar ways; for example, the subroutine INFOG2L which *computes the starting local row and column indices and process row and column indices of a global element of a block-cyclically distributed matrix*. Put another way, if we wish to find the value of a_{ij} calling INFOG2L with the values i and j will return k, l, r and c such that element a_{ki} on the process in position *(r,c)* in the grid holds the required value. Now, it would appear safe to assume that, given i and j define a unique element, INFOG2L would return the same result on every process. However, this assumption is wrong because the routine requires the user to state which process is calling the routine so that it can return the wrong answer on all the processes except the one holding the element. It is not clear why this decision was made, especially as it is actually more efficient to compute the correct answer on all processes.

Such quirky behaviour is strange but the problem does not end there. At least one other implementer, IBM [14], has included this routine in their parallel ESSL library and they even provide an example which shows how the wrong answer is computed by all processes except one; consistency at all costs? We need to take care not to spread such peculiar behaviours and this requires some form of active maintenance.

Summary

For those of us who experienced the heyday of numerical software development in the 70s and 80s, it is depressing to discover that many of the hard earned lessons from that era appear to have been forgotten. As an example:

> *One of us (PG) was recently interviewed for over an hour by a very pleasant young man at Intel who turned out to be completely ignorant of the existence of NAG or IMSL, and who had never heard of the work on floating point arithmetic by Kahan and the*

IFIP Working Group on Numerical Software, and who, needless to say, was blissfully unaware of the extent to which that legacy of work is today present in the very good Intel Math Library.

In the world of cluster programming that level of ignorance appears to be very common and one feels cast adrift into a parallel world that seems to exist without any knowledge of, or influence from, the body of numerical software that existed before and after the BLAS; for this is the one piece of software that is peddled by the vendors as the general panacea for all one's numerical problems. When it comes to implementing high quality mathematical software for a parallel environment using MPI, the current situation is woeful. The present state of the open source MPI implementations certainly require some tuning and perhaps this is to be expected – if an out-of-the-box solution is required then some would say that commercial software should be used instead of open-source. This might very well be the case except at this stage it is very difficult to determine the differences between the open-source versions of MPI and those available from commercial vendors, except the price.

The exercise of constructing high quality mathematical software suitable and robust enough for commercial applications using open source / freely available software is possible but requires a great deal of effort and perseverance. The expenditure of this effort is tempered by the thought that many other users have, most likely, already trodden the same path and what a good idea it would be if such duplication of work could be avoided. It really is time that the developers of packages like ScaLAPACK. PBLAS, etc took a lead from the thriving open source community currently producing high quality software using the facilities provided by, for example, SourceForge. Here projects are kept alive and up-to-date either by the original authors or by other interested (and knowledgeable) parties rather than being allowed to languish unloved on the original authors web or ftp site. Sites like SourceForge provide a central, easy to find location where users can download the most recent version of the software, report bugs, donate Makefiles and ask questions; software improves by expanding the user community and gaining feedback.

It is possible with determination, and a strong and encouraging *champion* of the software, to produce quality industrial strength software. Real vehicle parts, including full frame automobiles and trucks, are now straightforward to simulate using the parallel version of VPO™ EPD in an elapsed time that makes experimentation with different designs practical. The end result is a software product that enables significant reduction in costs, enhances good environmental practices, and improves the quality of the end product, namely the vehicles on the road.

7 (B) Incorporating into production

This section describes the most interesting aspect of this work because the *value* of the work depends on how successful it is to engage the workforce in using the new *virtual* toolset. If we fail in this regard then, it doesn't really matter how clever we have been in solving complex simulation problems since end users will not use the tools, for one reason or another. And one lesson we have learned from many years of working with large corporations: it doesn't always follow that it is the fault of the software that is the reason why users do not use it, for they have other influences that

may be more powerful than the ones you thought of in designing the software. Those influences boil down to one word: people. People in all levels of the organization will have and will always want a say, justified or not, in the use of the software – and that is the problem that has to be addressed and solved if the software is to be a success. People come and go but the software must remain for the good of the company.

Skills - *engagement*

Complex manufacturing computational problems are rarely solved by the push of a button, and even if they were it would not be sensible to avoid human intervention at some point, if only to examine results and messages stemming from the computation.

Currently, industrial operational environments rely heavily on people with hands-on experience that it is crucial to utilize in the computer simulation of complex processes. This experience needs to be captured and used in order to augment decision making processes encoded in software.

For computer simulations to be more effective than physical prototyping, software must be adapted to the existing skill base rather than the other way around:

(a) Software should be designed to use the knowledge and experience of users, because the processes being simulated cannot normally be performed effectively without their input and guidance.

(b) Users should feel comfortable in using the software – as far as possible the graphical controls, labels, and functions of user interfaces should reflect terminology and processes that are familiar to users from their working experience in the plant.

(c) Users must have confidence in the software – this can only be gained over time and by comparing results from the virtual process with those from the actual events.

(d) Generating what-if scenarios and experimenting to see the effects of different hypotheses will produce substantial quantities of output that needs to be analyzed, interpreted, and distilled into summaries for management to make the correct decisions. The existing workforce may be equipped to do these tasks but their job will be easier and more effective if the software contains tools to mine the data and extract trends and patterns.

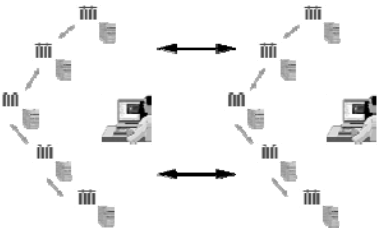
Only when all of these aspects are addressed successfully can the real benefits of virtual processing be obtained: a workforce that is *engaged* in the virtual process and that has the confidence to experiment with the software. Only then will they gain an understanding of what is possible and what will improve the end product and, in the process of achieving this, they will improve their own job satisfaction.

The timeline in section 5 shows that this process for VPO™ EPD began in 2003 and continues as the software evolves - *real* software continues to evolve as feedback and experience are gained from people.

The vision

It has been our experience that even when the workforce is engaged and eager to work with software tools that give them more insight; very often the wherewithal to use the software is not at their disposal because of computer policies that inhibit the true use of computer technology. In this regard, the freedom experienced by the Grid community is definitely lacking in industry – in our opinion to its detriment. Putting this to one side then, the vision that BSSI sees for using *virtual* toolsets in manufacturing processes, not just painting processes, is one where employees are empowered to run tools like VPO™ at their own workplace rather than through a centralized computing facility. The work conducted by BSSI over the last 10+ years shows that with the advances in computer technology and the consequent drop in prices, it is no longer necessary to use large computing facilities to obtain the benefits from virtual simulations. However, if some find objection to this remark then we can qualify it by saying that the simulation of painting processes described in this paper can definitely be accomplished by inexpensive, environmentally friendly, *personal* workstation clusters. All of the tools in VPO™ run on this type of cluster.

Personal clusters can have far reaching consequences for an industry fully committed to reducing costs by using computational simulation wherever appropriate. One obvious consequence is that a workforce equipped with *personal clusters* is much more likely to collaborate especially when the virtual tools are naturally interrelated as they are in VPO™. Successful collaboration is optimal for the company and therefore, the next stage of VPO™ that is already underway is the distributed environment shown in the picture where each personal cluster uses a combination of message passing and shared memory parallel processing.



Acknowledgments

The *champion* of VPO™ EPD, and the entire VPO™ toolset, is Ford Motor Company, and one individual in particular Jacob Braslaw. Both of them are to be commended for their foresight in starting the work over 10 years ago, which led to VPO™ EPD, and for their determination to see the work through to its present operational state with BSSI.

The experimentation with different workstation clusters and hardware would not have been possible without the willing participation of a hardware supplier with good business and service sense. BSSI is fortunate to have access to such a vendor in Datakjeden, [17], and especially their manager Stein Tore Hansen to whom we are gratefully thankful.

For help with their Fortran compiler above and beyond what normally one experiences, BSSI is very grateful to the support from PGI, [18], and their David Borer.

Finally, we would like to thank Brian Smith, for providing an early version of his Test Harness, [19], which we intend to integrate into our software maintenance cycle to improve and ease both our regression testing and debugging activities.

References

[1] Lahey Computer Systems, Lahey/Fujitsu Fortran 95 User's Guide, Linux Edition, Revision C, 2001.

[2] ISO/IEC, Information Technology -- Programming Languages -- Fortran - Part 1: Base Language (ISO/IEC 1539-1:2004), ISO/IEC Copyright Office, Geneva, 2004.

[3] W. Gropp, E. Lusk and A. Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface, Second Edition, The MIT Press, Cambridge Massachusetts, 1999.

[4] L. S. Blackford , J. Choi, A. Cleary, E. D'Azeuedo, J. Demmel, I. Dhillon, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker and R. C. Whaley, ScaLAPACK User's Guide, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[5] Numerical Algorithms Group Ltd., fpp: Fortran language pre-processor for the NAGWare f95 compiler, NAG Ltd, Oxford, UK, 2005.

[6] Numerical Algorithms Group Ltd., NAGWare Fortran Tools (Release 4.1), NAG Ltd, Oxford, UK, 2001.

[7] M. Snir, S. Otto, S. Huss-Lederman, D. Walker and J. Dongarra, MPI – The Complete Reference: Volume 1, The MPI Core, The MIT Press, Cambridge Massachusetts, 1998.

[8] W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir and M. Snir, MPI – The Complete Reference: Volume 2, The MPI-2 Extensions, The MIT Press, Cambridge Massachusetts, 1998.

[9] W. Gropp, E. Lusk, D. Ashton, P. Balaji, D. Buntinas, R. Butler, A. Chan, J. Krishna, G. Mercier, R. Ross, R. Thakur and B. Toonen, MPICH2 User's Guide: Version 1.0.3, Argonne National Laboratory, 2006.

[10] J. Vetter and C. Chambreau, mpiP: Lightweight, Scalable MPI Profiling, Version 3, http://mpip.sourceforge.net/, October, 2006.

[11] Etnus, TotalView Debugger, http://www.etnus.com/Documentation/index.php

[12] A. Chan, D. Ashton, R. Lusk and W. Gropp, Jumpshot-4 User's Guide, Argonne National Laboratory, 2005.

[13] Netlib, Scalapack Tools, http://www.netlib.org/scalapack/tools/infog2l.f

[14] IBM Corporation, Parallel Engineering and Scientific Subroutine Library for AIX: Guide and Reference, Version 3, Release 2, 2003.

[15] Portland Group, PGI User's Guide: Parallel Fortran, C and C++ for Scientists and Engineers, Oregon, US, 2006.

[16] L. J. Gray, J. M. Glaeser, T. Kaplan, *Direct Evaluation of Hypersingular Galerkin Surface Integrals*, SIAM Journal on Scientific Computing, volume 25, Number 5, pp. 1534-1556, (2004).

[17] http://www.datakjeden.no/default.asp?bID=1

[18] http://www.pgroup.com/

[19] Brian T. Smith, A Test Harness TH for Numerical applications and Libraries, these proceedings.

Q&A – Patrick Gaffney, Tim Hopkins

Questioner: Ian Reid

Are you using ACML for LAPACK / ScaLAPACK? If not, you should since it will almost certainly be the fastest for AMD processors. It is freely available from the AMD Website and is developed / supported by AMD & NAG.

Tim Hopkins:

Not at the moment. We looked at the download site several months ago and it appeared that Scalapack was no longer supported as it was listed only under their archive section. Given your comments we will look into this again.

Questioner: Boyana Norris

(a) Is the linear system sparse or dense?
(b) Why did you decide to use ScaLAPACK?

Pat Gaffney:

The linear systems are not sparse they are full matrices.

The first implementation of the system uses direct methods and therefore the MPI version is implemented using ScaLAPACK.

Another version of the system uses an acceleration technique, the Fast Spectral method that we have developed and that uses sparse techniques: the coefficient matrix is never assembled, and an iterative solver is required. This reduces the memory requirements significantly.

Questioner: Bill Applebe

Could commercial codes not have been used?

Tim Hopkins:

Certainly for the numerical computation there wasn't any software commercial or otherwise that implemented the methods we are using.

Pat Gaffney:

It is always possible to cobble together existing codes but that would not be as effective as the approach we have taken with VPO. Our customer tried for many years to use just such an approach and found it difficult to obtain necessary functional changes and support from the "one-size-fits-all" commercial code suppliers. For one thing, VPO has a uniform interface where all the VPO modules are accessible in one place. This was an important feature for the end customer and not one shared by disjoint systems.

However, by far the most important advantage of the VPO approach is that all of the modules: electrocoating, baking, drainage, voids, vibration, and future spray models, use the *same* computational grid, which is a definite advantage to the end-customer who already has a plethora of grids to contend with. Minimizing the number of grids by solving a range of problems using one grid is a distinct advantage and one offered by BSSI's VPO.

Questioner: Brian Smith

How do you test the software and how do you validate that the simulations are valid? What role does the customer play in the testing and validation process for the software?

Pat Gaffney:

The end customer has a rigorous sequence of procedures that must be followed before any software is accepted for operations. In the VPO work, testing is a continuous process involving both the end-customer and BSSI working closely together.

Comment: Ian Reid

On debuggers, you might like to look at DDT from Allinea as a cheaper alternative to Totalview.

Tim Hopkins:

Thanks for the information. Looking on their Web site they are also offering an optimization and profiling tool specifically for MPI applications. We will be evaluating these in the near future.

Questioner: Ron Boisvert

You painted a somewhat discouraging picture of the state of existing math software components that you are using to build your system. Given that such software is the underlying engine for technical applications, do you think that component-based grid systems and service oriented computing is realizable in the near term?

Tim Hopkins:

I'm sure that it is realizable in the near term in as much as people will be able to access the same low quality software they are currently familiar with via grid systems. What we are not hearing is how suppliers of grid systems intend to ensure that their users only have access to high quality, reliable and robust software. Perhaps this is a golden opportunity to impose quality on end users!