Simulation of a batch chemical process using parallel computing with PVM and Speedup

F. G. Smith, III and R. A. Dimenna
Westinghouse Savannah River Co.
Aiken, SC 29808

Abstract

Speedup, a commercial software package for the dynamic modeling of chemical processes, has been coupled with the PVM software to allow a single process model to be distributed over several computers running in parallel. As an initial application, this coarse distribution technique was applied to a batch chemical plant containing 16 unit operations. Computation time for this problem was reduced by a factor of 4.7 using only three parallel processors in the UNIX computing environment. Better than linear acceleration was achieved from the significant reduction in computation required to reinitialize the smaller subprocesses at discontinuities in the solution. The process was physically divided at points that naturally separated the overall plant into distinct subprocesses. This facilitated the computation by minimizing the interconnection between the parallel units. Techniques were developed to make efficient material and energy transfers between the modeled subprocesses based on actual material transfers used in plant operations.

Keywords: parallel computing, coarse distribution, dynamic modeling, chemical process, batch processes.

1. Introduction

As computing costs have decreased and computers have become larger and faster, the dynamic modeling of chemical processes at greater levels of detail is an area of increasing interest. As opposed to more commonly used steady-state process models, a dynamic model can provide valuable information about process timings, transient behavior during startup and process upsets, and the coupling between unit operations. Inherently transient behavior such as reaction kinetics can be included in the model. These advantages are particularly apparent for batch processes that do not exhibit true steady-sate behavior in the first place. However, dynamic modeling clearly requires significantly greater computational effort than is needed to solve for a single steady-state operating condition. One method to reduce the time required to obtain a solution is to divide the problem into segments and solve the parts in parallel on multiple computer processors. This technique also mimics actual plant operations where many processes may be operating simultaneously. Parallel computing has been used to advantage in many areas of science and engineering and has increasingly been applied to industrial engineering problems. Overviews of industrial applications of parallel computing are given by Eldredge et al., (1997) and by Thole and Stuben (1999).

A considerable amount of work is in progress to develop better numerical methods to take full advantage of parallel computations. Large scale simulations of chemical processes is an application area where very large systems of coupled non-linear equations naturally occur. This is especially true when a large number of chemical species must be considered. Vegeais and Stadtherr (1990) describe a parallel computing strategy for chemical flowsheets. Mallya et al., (1997) and Paloschi (1996, 1998) focus their efforts on developing solvers to take advantage of a parallel architecture at the solver level. Our work has adopted the strategy of applying unmodified commercial software at the model development level, and as such, is more accessible to others wishing to use coarse distribution parallel computing techniques.

The Speedup software from ASPEN Technology was one of the first commercially available flowsheeting packages for the dynamic modeling of chemical processes (Pantelides, 1988). Speedup allows the user to readily develop unique models of unit operations using algebraic and first order differential equations that can then be combined into a system flowsheet and solved in a time dependent form. Since many of the unit operations at the Savannah River Site are unique, this framework has been particularly useful for flowsheet modeling needs at our location. ASPEN Technology has recently replaced Speedup with a new software package called Aspen Custom Modeler (ACM). ACM retains the equation based structure and solution methods used by Speedup while also providing a graphical user interface with convenient flowsheet building capabilities not available in Speedup. However, ACM is only supported on Windows based workstations and personal computers. Therefore, the work discussed here, which took advantage of the UNIX computing environment, is no longer supported by the commercially available software. Nevertheless, the general concepts and methods described in this paper focus on communication logic that can be implemented regardless of computing platform, and these will still be of interest to the engineering modeling community. The concepts are general and provide the basis for similar implementation with the newer ACM software. Work is now in progress at our site to adapt the methods described here to ACM in the Windows operating environment.

Parallel computing using Speedup has been previously reported in the literature (Paloschi, 1996 and Paloschi, 1998). In the previous work, the author replaced the Speedup nonlinear equation solver with his own coding that was designed to use parallel processors. In contrast, the work described in this report uses the commercial Speedup package without any modifications. We simply run a Speedup model on each of the parallel computers and exchange information between them using the PVM software (Geist et al. 1994). Each machine is then running a simulation that represents a part of the overall chemical process. Information exchange between the processors simulates the physical transfer of material and energy between the separate parts of the chemical plant , as well as process control signals. The previous work by Paloschi was also limited to obtaining a steady-state solution for a flowsheet. Paloschi and Zitney (1997) have reported on the parallel simulation of a chemical plant that is very similar in concept to the techniques described in this report. The authors describe using the Speedup flowsheet solver coupled with PVM for message passing to run both steady state and dynamic simulations of a chemical plant. The problem they apply the method to is a continuous chemical plant simulation with recycle. In the current work, we describe the dynamic

simulation of a batch chemical process. Modeling a batch process versus a continuous one leads to a fundamental difference in the calculation timing. Continuous simulation allows the selection of a time step that maximizes computational efficiency whereas, when simulating a batch operation, the time step is further constrained by the process communication logic. The work reported here on parallel simulation of a batch process produced greater than linear acceleration in computing time through the use of coarse discretization or coarsely noded parallel computing techniques.

2. Background

The chemical process described in this work models the treatment of radioactive waste at the Savannah River Site (SRS). The safe and efficient cleanup of radioactive waste is an area of extreme importance and national interest. Preliminary waste processing is first done in the site tank farm to separate the material into aqueous salt solutions and sludge suspensions. Sludge is pretreated by dissolving aluminum salts and washing out water soluble non-radioactive components to reduce the volume of highly radioactive waste. Plans are in progress to treat concentrated salt cake by first redissolving the material and precipitating insoluble radionuclides such as uranium and strontium. Cesium would then be removed from the remaining solution by a solvent extraction process now under development. However, at the time this work was performed, it was thought that a precipitation process would be used for cesium removal. Therefore, this paper is based on the older precipitation process. After concentration and washing, salt and sludge fractions containing most of the radioactivity will be sent to the Defense Waste Processing Facility (DWPF) for final treatment and vitrification. The DWPF plant has been in operation since 1996 vitrifying sludge waste alone pending the start of salt processing operations.

As described by Choi et al. (1991), an integrated steady-state model of the entire waste processing operation has existed for several years. This model was developed using the du Pont Company's proprietary process simulation software CPES (Chemical Process Evaluation System). This model has over 1700 process streams and 650 unit operations and tracks 183 chemical species. The model includes detailed waste processing chemistry and performs rigorous vapor-liquid equilibrium calculations. It has been used to plan waste removal operations and perform environmental permitting calculations for plant operations. Although most of the actual processing occurs in discrete batches, the CPES simulation treats the processes as equivalent continuous steady-state operations. Therefore, it provides only averaged information on system dynamic behavior and batch cycle behavior.

Dynamic modeling of various parts of the SRS High Level Waste (HLW) system has been undertaken in recent years. In particular, Gregory et al. (1994) describe the development of an integrated model of the entire HLW system including waste generation, storage in the tank farm, waste treatment, and final disposal. This model, originally written in Speedup, simulates tank farm operations and in-tank waste treatment processes in some detail. However, there is a large difference in the process time scales between DWPF operations and those in the tank farm. Waste treatment in the tank farm uses million gallon tanks and large volumes of material, whereas DWPF batches are on the order of several thousand gallons. A single batch of pre-treated waste will supply

many DWPF feed batches. Therefore, to achieve a reasonable simulation time, the large scale model treats DWPF operations as a simple continuous process to estimate the volume and composition of glass product and recycle streams.

To complement these process models, a detailed model of DWPF operations was developed by Smith (1998) with the objective of optimizing plant performance such that the environmental impacts from waste processing operations are minimized while the amount of waste treated is maximized. The model was used extensively for the evaluation of alternative methods of salt waste disposition.

The DWPF plant consists of three major processing areas: 1) Salt Processing Cell, 2) Chemical Processing Cell, where high level radioactive waste is chemically treated to produce a form suitable for vitrification, and 3) Melt Cell, where glass formation and canister pouring takes place. The DWPF model was written in Speedup and performs material and energy balances around 16 primary unit operations consisting of process vessels where liquid solutions are mixed, concentrated, separated, and stored. Aqueous and organic liquid phases and the vapor phase are considered in the process modeling. The liquid phase contains up to 42 chemical species, while nine species are tracked in the vapor phase. Table 1 lists the chemical species used in the model. Product glass composition and melt physical properties (density, liquidus temperature and viscosity) are predicted for given feed material compositions. In addition, the model accurately reflects batch operating sequences throughout the process and includes an algorithm to optimize the composition of the glass product by maximizing the amount of waste material included in the glass mixture. The model is used to predict chemical compositions in the process vessels and the vent system and to calculate material volumes throughout the entire process. A version of this model that uses simplified process chemistry and idealized vapor-liquid equilibrium calculations is currently operational and was used for the work described in this paper.

The full integrated model of the entire HLW system contained about 11,500 algebraic-differential equations and took over 12 hours of CPU time to simulate one year of operation on our fastest available workstation. To fully utilize the modeling, it is necessary to simulate the system behavior over several years of operation. Therefore, development of a parallel model to reduce the computational time was proposed. Since the integrated model is so complex, an initial effort with the smaller and more tractable DWPF model was chosen as a test problem with which to develop parallel computing methods.

3. Model Description

Figures 1-3 show schematic diagrams of the three processing areas in DWPF. These schematics represent the level of detail included in the dynamic process model. Labels enclosed in ovals denote feed streams to the process. Transfer streams between the parts of the process are labeled in hexagonal blocks. Unenclosed labels mark sinks where material leaves the process. Control valves with meters attached indicate transfer lines where a pre-specified volume of material flows through the line for each batch. The action of control valves that do not have meters attached is dictated by tank conditions.

Signals are sent to these control valves that will initiate flow when both the delivery and receiving tanks are in the correct state. That is, when the delivery tank is ready to transfer a finished batch and when the receiving tank is able to accommodate the additional volume, both tanks simultaneously signal the valve to open. Similarly, a closing signal to stop flow is sent when either the receiving tank is full or the tank sending material through the line has been emptied to its specified heel volume.

The following sections provide brief descriptions of the individual process areas. A more complete description of a slightly different version of the DWPF model has been published elsewhere (Smith, 1998).

3.1. Salt Cell

Figure 1 shows a schematic diagram of the DWPF Salt Cell as it is represented in the Speedup model. To simplify the schematic, initials are used to identify process units. The full process unit names and corresponding vessel volumes are listed in the table below the figure. Tank volumes are given in gallons, which corresponds with the usage in actual plant process measurements and control systems. The primary processing vessel in the DWPF Salt Cell is the Precipitate Reactor (PR) where precipitate from the tank farm is hydrolyzed using acid. The PR has a condenser/decanter unit that separates aqueous and organic phases boiled off during the hydrolysis reaction. The aqueous phase is returned to the PR and the organic byproduct collected in the Organic Waste Storage Tank (OWST). Off-gas vapors pass through a secondary condenser (not modeled) prior to venting. Additional tanks in the process feed raw materials and collect the liquid product.

To start the Salt Cell process, about 3500 gallons of precipitate from in-tank salt processing would be charged into the PR. The volume of each batch of precipitate added to the transfer tank can be varied and is specified through an input data file. Thereafter, the volume transferred into downstream tanks will depend upon either the amount of material that a receiving tank can accept or the amount of material a supply tank can discharge. The operating limits on tank volume for each vessel are defined within the model. Copper catalyst solution and formic acid are made up in their respective feed tanks. Raw materials are assumed to always be available. The Salt Cell process consists of two batch cycles in the PR. After each batch, the material remaining in the PR is transferred into the PRBT. When the PRBT has collected two batch volumes, it must be emptied before Salt Cell processing can continue. When the OWST is full, the contents are transferred to a waste disposal facility. The waste disposal process is modeled as an infinite sink able to accept organic waste product as required.

As noted in the introduction, work is currently in progress to replace the in-tank precipitation with a solvent extraction process. Raffinate liquor from the solvent extraction containing radioactive cesium will then be added directly to the DWPF Chemical Cell thereby eliminating the need for Salt Cell processing. Full operational details of this revised process have not been developed. Therefore, for demonstration purposes, the work described here has used the older process design.

3.2. Chemical Cell

Figure 2 shows a schematic diagram of the DWPF Chemical Cell model. The Chemical Cell contains the Slurry Receipt and Adjustment Tank (SRAT), where washed sludge from the tank farm is mixed with the aqueous product from the Salt Cell process and the Slurry Mix Evaporator (SME) where adjusted sludge is mixed with glass forming frit and concentrated. The SRAT and SME both have condensers and a common condensate collection tank (SMECT). Other tanks in the process feed raw materials and collect the condensate and liquid products. The process units shown in Fig. 2 indicate the level of detail included in the Speedup model.

After two Salt Cell batches have been completed, the aqueous product in the PRBT is transferred into the SRAT for further processing. At this point, the next Salt Cell batch can be started. If the SRAT is not available after two PR batches are finished, processing in the Salt Cell must wait until the contents of the PRBT have been transferred out before another batch can be started. As a part of the SME operations, an optimization calculation is performed to minimize the amount of frit (maximize the amount of waste) added to the slurry. The glass product must meet certain constraints on its composition to ensure that the material can be processed and that the resulting waste form will be acceptable. This poses a multivariable constrained optimization problem. Jantzen and Brown (1993) have described the appropriate glass property correlations and solution techniques. Basically, the composition of the slurry must be adjusted such that the liquidus temperature and viscosity of the molten glass fall within ranges that allow acceptable melter performance. Likewise, the final glass product must be homogeneous and meet performance criteria for resistance to leaching. These properties have been correlated with the chemical composition of the glass. Adding the minimum amount of glass frit to the slurry such that these compositional constraints are met optimizes the process by minimizing both material costs and the final volume of waste product. A simplified version of the original algorithm is implemented as Fortran procedure PCCS in the DWPF model. Batch sequencing techniques similar to those used to make material transfers were used to ensure that the optimization calculation was performed only one time for each SME batch and at the proper time in the batch cycle. The calculation must be performed after the SRAT batch has been transferred into the SME so that the composition of the SME heel is accounted for in the glass blending.

3.3. Melt Cell

The DWPF Melt Cell contains the Melter Feed Tank (MFT), glass melter, glass canister where the product is collected, the melter off-gas condenser, and a condensate collection tank (RCT). Slurry produced in the Chemical Cell is transferred into the MFT which serves as a holding tank to transition from the batch chemical processing into continuous melter feeding. Figure 3 shows a schematic diagram of the Melt Cell as it was modeled in Speedup with specific unit information provided in the accompanying table.

In the Melt Cell, slurry from the MFT is continuously fed to the joule heated glass melter at a rate of about one gpm. The melter is maintained at around 1050 C and pours the molten glass into three-meter tall stainless steel canisters that hold on the order of 1680

kg of glass. After filling, the glass canisters are sealed and stored on site pending the availability of a permanent storage facility. The model performs only material balance calculations around the melter to predict the composition of the glass product and melter off-gas. No energy balance calculations are performed around the melter. At the high melter temperature, volatile material in the feed stream, principally water, is driven off and the metals are converted into the oxide forms listed in Table 1, which then dissolve in the molten glass.

The DWPF Chemical Cell and the Melt Cell have been in actual operation for over five years. The Salt Cell is not currently operational, and at this time, the replacement of both this process and the in-tank precipitation step that precedes it are being investigated to mitigate the formation of flammable benzene vapors. Nevertheless, the integrated model described above can be used for preliminary attainability studies and the modeling can be modified easily to accommodate future process changes.

4. Parallel Computing Methods

4.1. Concept

The fundamental purpose of parallel computing methods is to employ multiple processors to evaluate independent pieces of a calculation simultaneously, and thereby reduce the computational time compared to a single computer evaluating the same pieces sequentially. Computational time savings are realized if the reduction in compute time is large compared to the additional overhead time incurred to distribute the problem to the multiple computers. Most modern parallel applications separate the calculations at the compiler level and distribute individual mathematical operations to the available processors. This approach generally leads to so-called, “massively parallel” applications and can employ hundreds of processors. At the other extreme is the approach we take in this paper, a “coarsely distributed” or “coarse node” parallelization. This method is based on the nature of the chemical processes being evaluated, and not on the numerical characteristics of the solution method.

A coarsely distributed parallel approach is a natural fit to the DWPF process described above. In fact, the parallel model was designed to mimic the actual DWPF operation as closely as possible. The several cells described, Salt, Chemical, and Melt cells, can each be operated in a batch mode with relatively infrequent pauses to either discharge material to a downstream cell or receive material from an upstream cell. Because of this, a computational model can be constructed to operate exactly the same way as the actual facility. For example, a model of the Chemical Cell can be initialized with a charge of material from the Salt Cell. It can then be run without further interaction from either of its neighboring cells until such time that its internal batch processing is completed. The criteria for completion of the batch are entirely self-contained in the Chemical Cell, so no intervention or information is required from the other cells. Once the batch is completed, the processing in the Chemical Cell halts and waits for the next appropriate operation, such as a discharge to the Melt Cell.

This type of behavior lends itself well to a Speedup model implementation in which the processes in each cell are developed into a model of that particular cell. The model for each cell is completely self-contained and requires only input and output information to run in conjunction with the other cell models. An external driver program was developed to coordinate the transfers of material to and from each of the cells.

The ability to realize a reduction in overall compute time depends on the same fundamental comparison as the massively parallel approach; viz., the reduction in time resulting from using multiple computers must be large compared to the overhead incurred in distributing the problem. In the case of this DWPF model, this criterion was satisfied easily, and for this reason, the coarsely distributed approach seemed a natural fit. The computational time for each of the cells, even when accelerated by reducing the size of each individual model, was much greater than the time needed to transfer flow stream information (volume and composition) at the relatively infrequent pauses of each cell model. In fact, even with a relatively old Ethernet™ network, the communication time was so small compared to compute time that it was essentially negligible. All of the time improvements were realized by reducing the Speedup calculation time, and this reduction was significant.

4.2. Implementation

Details of the file structure used on the parallel machines and information on how the combined Speedup and PVM parallel computing environment was created are provided in this section. For this initial effort, time was not available to optimize the system or investigate many of the variations in parallel configurations that could be tested. What is described here is a system that has worked well but could still potentially be improved.

The Parallel Virtual Machine (PVM Version 3.4, 1997) software was used to set up a small-scale parallel-computing environment on the SRS UNIX cluster. This parallel system consisted of first two and later three IBM RS/6000 workstations. The PVM software, freely available from several sources (Geist, et al., 1994), uses the parent-child construct to spawn multiple processes on machines in the parallel cluster. The software also provides methods for the jobs to exchange information over the cluster network. As a test, UNIX support personnel installed PVM, Speedup and Fortran first on a set of two machines in the cluster (pvma and pvmb) and later on three dedicated machines (pvmx, pvmy and pvmz). Message passing is improved if all of the machines use the same data architecture so that the need for message translation is eliminated. This was the case for our small test networks.

The following UNIX `.cshrc` file was created on each machine to set environmental parameters and path names that point to the location of the PVM and Speedup files:

```
------------------------.cshrc----------------------
setenv TERM vt100
setenv DISPLAY oxygen:0.0
setenv PVM_ARC AIX46K
setenv PVM_ROOT /usr/local/pvm
```

```
set path=($path $PVM_ROOT/lib)
set path$($path .)

source /Speedup/su556/libdisk/splogical.csh
-----------------------------------------------------
```

The file also identifies the local machine, in this case named oxygen, which was used to run the simulations and view the results. Each machine in the cluster must have a UNIX *.rhosts* file identifying the other machines in the PVM cluster and the user home directory on each machine. These files have the following structure:

on pvmx
```
---------.rhosts---------
pvmy fsmith
pvmz fsmith
-------------------------
```

In this file, *fgsmith* is the user home directory on both machines. Corresponding files were created on the other two machines in the cluster.

It is also desirable, but not absolutely necessary, to have a *Hostfile* on the machine that will be used as the parent machine where PVM is launched. This file has the structure:

on pvmx
```
----------Hostfile----------
pvmx ep=source
pvmy ep=source
pvmz ep=source
----------------------------
```

This file tells PVM which machines are available to use in the parallel computing environment and sets the directory (*source*) on each machine as the location where executable programs (ep) will be found. Therefore, each machine had a subdirectory named *source* created directly under the home directory and the executable programs were placed in these directories.

On the parent machine, pvmx, Speedup was run within the *source* directory. For example, the source file *chem_cell.speedup* was placed in the *source* directory and (assuming we are starting with a fresh Speedup directory) Speedup was run with the following command lines:

```
Speedup> new chem_cell
Speedup> store chem_cell
Speedup> comp(ile) dyn(amic) noex(ecute) run
Speedup> quit
```

This created an executable Speedup module in the directory. To link to the PVM Fortran and C libraries, the following *Speedup.include* file was also created in the *source* directory:

```
---------------Speedup.include--------------
```

```
-L/usr/local/pvm/lib/AIX46K –lfpvmz
-L/usr/local/pvm/lib/AIX46K –lpvmz
-------------------------------------------
```

The include file provides paths that direct the Speedup compiler to the PVM Fortran and C libraries. In addition, the Speedup procedures that call PVM subroutines need to use the include file *FPVMZ.H* by having the line:

```
include '../fpvmz.h'
```

in the subroutine. As a quirk of Speedup, the software converts all of its imbedded Fortran code, including that in any user written procedures, into uppercase. On the case sensitive UNIX machines this then required that the name of the include file also be uppercase. Therefore, the include file was copied from the PVM directories into the local directory where Speedup would be run and the file was renamed in uppercase.

Compiling and linking creates the executable Speedup module speedpdyn.exe in the subdirectory *scrdisk*. On each of the machines in the PVM cluster, these executables were copied into the *source* directory and assigned the more meaningful names Chem_Cell, Salt_Cell and Melt_Cell. For unknown reasons, PVM would not operate correctly unless the Speedup software was located in the home directory on the child machines. That is, on pvmy and pvmz, for example, it was necessary to run Speedup in the home directory and then store the executable in the *source* subdirectory. If the Speedup itself was located in the *source* directory the error message:

```
'could not determine problem size'
```

was generated within the PVM log file when PVM was executed. It was unclear whether this message was generated from PVM or Speedup and time was not available to resolve this issue. Since the system worked smoothly when Speedup was located in the child machine home directories, this directory structure was simply used during our test program. On the two child machines, the executables were placed in the *source* subdirectories. Once the Speedup executable files were created and located in the proper directories, PVM was started on the parent machine with the command line:

```
pvm Hostfile
```

The parent Speedup executable could then be run and the child processes automatically spawned from within Fortran procedures in the parent process.

Interfacing PVM with the flowsheet simulation package Speedup was one of the challenges of this project. Early in the development it was decided to internalize the PVM interface within the Speedup models. PVM uses its own library of subroutines to implement message passing between processes and specialized coding is required to make calls to the PVM subroutines. This coding was placed in Speedup procedures, which are user written Fortran subroutines that can be called from within the Speedup models. Since flowsheeting software commonly allows user written subroutines, this method should be of general applicability. The Speedup models pass information such as

tank volume, liquid density, and composition to the procedure along with a signal indicating when a material transfer is ready to take place. The procedure then used the PVM subroutines to transfer the data to the target process. Figure 4 shows a schematic diagram of the parallel model.

The coding from the Salt Cell model, which is the simplest of the three interfaces, can be used to illustrate the concepts involved. The objective is to transfer Precipitate Hydrolysis Aqueous (PHA) solution from the PRBT vessel into the SRAT. The PRBT is part of the Salt Cell model while the SRAT is part of the Chemical Cell model that is running on another computer. Within the Salt Cell model, a simple unit operation is used to represent the Chemical Cell, where the transfer from the PRBT will be received. The only purpose of this model is to call the Xfer_Chem procedure (see listing in appendix) with the correct information to make the transfer. Inputs to the procedure are the PRBT tank parameters (volume, density, concentration vector, and batch number), a signal that the PHA transfer is ready, and the current simulation time.

Examining procedure Xfer_Chem in the appendix, three types of calls are recognized through the Speedup provided flag `icall`. The procedure is called once before the dynamic calculations start with the flag `icall=1`. During this call, the process obtains a unique task identification number (tid) and finds the tid of the parent process using the PVM calls:

```
call pvmfmytid  (mytid)
call pvmfparent (mptid)
```

The Salt Cell model is programmed assuming that the Chemical Cell model will be the parent process. The choice of a parent process for the parallel network is arbitrary but influences the required code structure. The procedure is also called at the end of the dynamic simulation with the flag `icall=2`. This is used as a signal for the process to exit from the PVM system using:

```
call pvmfexit (info)
```

At all other times when the procedure is entered, while the dynamic simulation is running (flag `icall=0`), the procedure first checks that the simulation time has advanced since the last call. This prevents the procedure from performing the same instructions more than once while the Speedup solution is converging at each time step. If the simulation time has advanced (`time>tset`), the procedure checks that a transfer is not already in progress (`x_pha=.false.`) and that the signal that a transfer is ready to start has been set (`nsignl=1`). For the Salt Cell, a signal that a transfer is ready occurs when the PRBT has been filled with two batches of PHA from the PR. If both conditions are met, the procedure looks for a message from the Chemical Cell that it is ready to receive a batch of PHA. The Chemical Cell will send this message when its own batch cycle is at the point where PHA must be added to the SRAT vessel. The PVM calls used are:

```
call pvmfrecv   (mptid   , msgtype,  info)
call pvmfunpack (integer4,isignl,1,1,info)
call pvmfunpack (real8   ,time0 ,1,1,info)
```

The message is contained in the buffer `info` and consists of a signal flag (`isignl`) and the current Chemical Cell simulation time (`time0`). Once the Salt Cell has reached the point in its cycle where a PHA transfer needs to be made, it will wait to receive this message without performing any further calculations. This waiting period consumes minimal computer resources and is necessary to synchronize transfers between the two parts of the process.

When the message that a transfer can be made is received, the Salt Cell responds by sending the required information. The volume of PHA to be transferred (`pha_vol`), PHA density (`pha_rho`), composition vector (`pha_con`), and batch number (`nbatch`) are packed into the buffer `info` and sent to the Chemical Cell process using the PVM calls:

```
call pvmfinitsend (msgcode                ,info)
call pvmfpack     (real8   ,time   ,1   ,1,info)
call pvmfpack     (integer4,nbatch ,1   ,1,info)
call pvmfpack     (integer4,lpha   ,1   ,1,info)
call pvmfpack     (real8   ,pha_vol,1   ,1,info)
call pvmfpack     (real8   ,pha_rho,1   ,1,info)
call pvmfpack     (real8   ,pha_con,lpha,1,info)
call pvmsend      (mptid   ,msgtype       ,info)
```

The message also contains the current Salt Cell simulation time (`time`) and the length of the concentration vector (`lpha`).

After sending the message and writing the information to standard output for tracking purposes, the procedure compares the Salt Cell simulation time to the Chemical Cell simulation time. If the Chemical Cell is at a later time in the simulation, the Salt Cell uses the calculated difference (`delt`) to advance the calculation before the flow of material out of the PRBT is started. In this way, the simulation time between the two models is synchronized. Finally the procedure sets the transfer logical indicator `x_pha` to true to indicate that a transfer is in progress. Another transfer cannot start until this one is completed. On subsequent passes through the procedure, the subroutine will check to see if the flag `nsignl` has been set to a value of `0,` indicating that the transfer is complete. At this point, the logical signal `x_pha` is reset to false and the subroutine will again check for a Chemical Cell message that it is ready to receive a PHA transfer (`nsignl=1`). Similar logic is used in the other models and subroutines to effect transfers of information between the Chemical Cell and Melt Cell processes.

The procedure is relatively compact having a total of 150 lines of code including comments. Similar procedures were used in the other two models, which demonstrates that the creation of parallel computing systems using packages such as PVM does not require extensive coding.

5. Parallel Computing Results

When only two machines were available, two different versions of the DWPF Speedup model were tested. The first version split the Salt Cell from the rest of the model. This was a convenient initial test problem since there is only one stream connecting the Salt

Cell to the rest of the model (transfer of PHA from the PRBT to the SRAT). CPU timings comparing a run with the coupled model to one with the parallel calculation are shown in Table 2. Splitting the model in this way significantly improved the overall run time (acceleration factor of 1.64), even though the split was far from optimal. A more desirable split would balance the computational load between the processors more evenly. These initial results were encouraging, since even separating a small part of the model from the main calculation dramatically reduced the elapsed CPU time. As discussed below, this is likely the result of a significant reduction in the number of discontinuities and the associated reinitialization of the problem that must be handled by the individual processes.

With only one transfer stream between the parallel processes, the logic to test for transfers was clear. As described above, the Chemical Cell model was run up to the point where it was ready to receive a transfer of material from the PRBT to the SRAT. At this point the model sent a message to the Salt Cell model that it was ready to accept a transfer. In parallel, the Salt Cell model ran up to the point where the PRBT was ready to send a transfer to the SRAT. At this point the Salt Cell model tested whether the Chemical Cell had sent a transfer message. If the message had not been sent, the Salt Cell calculation would wait until it received the transfer message. In a similar fashion, the Chemical Cell model waited after sending a transfer message to the Salt Cell until the corresponding return message was received and the physical transfer could be made. As a part of the messages, each process sent the other the simulation time when it expected the transfer to occur. When a transfer was ready to be sent and received, each model compared the expected transfer times. The model that had not yet reached the greater of the two times then ran an additional delay time to catch up and synchronize the two calculations. This logic worked well with only one transfer stream.

As a second test problem, the DWPF model was split into a different configuration where the Melt Cell was separated from the rest of the model. This split required breaking two streams, one where material is transferred from the SMECT into the RCT, and the other from the SME to the MFT. As shown in Table 2, this split produced a slightly better division of the computational load between the nodes and gave an acceleration factor greater than two. Splitting the model in an even different fashion could perhaps improve the results further. However, the tested splits were ‘natural’ in the sense that the overall process was divided into its separately functioning parts.

With two transfer streams, it was necessary to modify the transfer message logic described above. Transfers between the SME and MFT could be scheduled exactly as described for those between the PRBT and the SRAT. However, the SMECT to RCT transfers were slightly different. Logic was required to transfer material from the SMECT to the RCT whenever at least 4000 gallons could be moved. This emulated actual operating procedure designed to avoid overfilling the tanks while minimizing the number of transfers. Since these transfers could occur while a SME batch was being processed, the Melt Cell model could no longer simply be allowed to run until it was ready to receive the next SME batch. If the Melt Cell model executed faster than the Chemical Cell model it could potentially miss some of the SMECT transfers. The transfer logic was therefore modified such that the Melt Cell would run until it was in a position to accept

either transfer from the Chemical Cell. Then, when the Chemical Cell signaled that either a SMECT or SME transfer was ready, the earliest transfer time was selected and whichever model had stopped prior to this time in the simulation was advanced to the transfer time and the transfer was executed. The models would then each continue running until another transfer could be made. In this way, each model could execute as fast as possible on its individual machine and the timing between the parts of the simulation only needed to be synchronized whenever material transfers occurred.

To further test the parallel-computing environment, UNIX support personnel created a three-node cluster of IBM RS/6000 workstations running PVM, Speedup and Fortran. Since different machines were used in this test network, execution timings are not directly comparable to the results shown in Table 2. The machines in the three-node cluster were all similar, having comparable CPU times for identical problems, but were somewhat older and slower than the original test machines. As a test problem, the DWPF model was now split into three parts (Chemical Cell, Salt Cell and Melt Cell) again following natural divisions within the process itself. Each separate model was compiled and executed on one of the PVM machines. This three-node cluster was successfully run with the execution timings shown in Table 3. Again, if the computational load were balanced more evenly, an even greater improvement in execution time could be achieved.

6. Comparisons of Parallel and Serial Computing Results

To establish that the serial and parallel versions of the model were equivalent, the time histories of the liquid volume in several of the tanks obtained from both simulation techniques were plotted together. The tanks involved in the inter-process transfers, the PRBT, SRAT, SME, SMECT, RCT and MFT, were selected for this comparison. Graphs of the tank volume as a function of time calculated by both the serial and parallel versions of the model for these six tanks are plotted in Figs. 5-10. The three transfer tanks, PRBT, SME and SMECT, were expected to behave identically in both versions of the model since the logic to initiate a transfer was identical. Any differences in liquid volume or timing would clearly indicate some discrepancy between the models. As shown in Figs. 5, 7 and 8, the three transfer tanks behaved identically in both versions of the model with the two sets of results overlaying exactly. Of more interest is the behavior of the three receiving tanks, SRAT, RCT and MFT. For these tanks to operate the same in both versions of the model the transfer logic must be correct and the time between the parallel parts of the model must be synchronized. As shown in Figs. 6, 9 and 10, the three receiving tanks behaved essentially identically in the two versions of the model. In most cases, the two results overlap so closely that the two separate curves can not be distinguished. Minor differences in the liquid volumes in the MFT and RCT can be seen on close examination of the results toward the end of the simulation. Implementing the inter-process transfers required modifications to the Speedup model logic which introduced minor variations in the calculation time step and made it difficult to perfectly reproduce the serial model behavior. However, the serial and parallel computations produce virtually identical results. Even the relatively complex and unpredictable behavior of the liquid volume in the RCT is reproduced in the parallel version of the model. This tank provided the strictest test that the logic in the parallel model implementing inter-tank transfers was correct.

7. Conclusions

As computing power becomes less expensive, the availability of multiple processors increases and the ability to use parallel computing methods becomes attractive. The work reported here establishes the feasibility and advantages of parallel computing with Speedup. Significant savings in computational time were realized by splitting a model of a batch chemical process into several parts and running the calculation on multiple computers in parallel. With three computers and a non-optimal splitting of the computational load we have achieved almost a factor of five in computational acceleration. Considering the timings in Table 3, it appears that adding another processor, splitting the Chemical Cell model into two equal pieces and shifting some of the Melt Cell calculations over to the Salt Cell would approach the goal of a load balanced parallel system. With a fourth processor, we would expect to achieve a further increase in the acceleration factor over the serial model computation. The greater than linear acceleration observed in our results is attributed to the isolation of model discontinuities to smaller pieces of the calculation and significant timesaving realized by avoiding reinitialization of the entire model.

With the batch model, a significant part of the timesaving realized from the parallel computation comes from eliminating the necessity to reinitialize the entire computation at every discontinuity. When modeling batch processes in particular, a large number of discontinuities typically occur during the solution. Discontinuities arise when process variables experience an abrupt change in state. For example, when flows start and stop, when state conditions such as a tank volumes or operating temperatures reach upper and lower limits, when timers used to control process sequencing start and stop all create discontinuities in the solution. At these points, the equation solver must stop and reinitialize the solution to account for the change in state variable before continuing with the solution. Restarting the solution at points of discontinuity is time consuming and can produce long run times. Significant savings in CPU time can be realized by reducing the number of discontinuities that must be handled. For example, with the Salt Cell, Chemical Cell, and Melt Cell all coupled, when a discontinuity occurs in one part of the model the entire solution must stop and reinitialize. By separating the models and running them in parallel, discontinuities are isolated more specifically and other parts of the model can continue to run when a discontinuity is encountered and reinitialization of the calculation is required in one segment.

The above arguments indicate that splitting a batch simulation into separate pieces running on individual processors will achieve better acceleration than using a parallel solver on the full problem. This result should pertain to the modeling of most batch process. The principal drawback to this scheme is the necessity of running separate copies of the underlying simulation software on each platform. Licensing multiple copies of commercial software is expensive whereas using a parallel solver embedded within one copy of the software can be more easily scaled to many processors and will be more cost effective. Nevertheless, often achieving an improvement in simulation run time by a factor of ten or even two can make a substantial difference in the usefulness of the model. The methods described in this report present a simple way to achieve this improvement in productivity.

Future work will focus on generalizing the concepts developed in this work. While an attempt was made to write relatively general algorithms to implement the information transfer between the processes some features of the scheme are specifically designed for this application. It is felt that with some additional effort, the methods developed here can be generalized to model material transfers in a parallel computing environment. Also the Speedup software used in this study is no longer commercially supported. Nevertheless, the techniques developed here are of general applicability that can be translated to other simulation packages and computer systems. At our site we are currently translating these methods to the Aspen Custom Modeler™ flowsheet simulation package running on a Microsoft Windows™ network of workstations.

8. Acknowledgements

The information contained in this paper was developed during the course of work performed under Contract No. DE-AC09-96SR18500 with the U. S. Department of Energy. The authors would like to express their thanks to the Savannah River Site UNIX support group for building the PVM test network used for this project.

9. References

Eldredge, M., T. J. R. Hughes, R. M. Ferencz, S. M. Rifai, A. Raefsky and B. Herndon, 1997, "High-performance parallel computing in industry," *Parallel Computing*, **23**, 1217-1233.

Geist, A., A. Beguelin, J. Dongarra, W. Jaing, R. Manchek and V. Sunderam, 1994, "PVM 3 User's Guide and Reference Manual," ORNL/TM-12187, Oak Ridge National Laboratory, Oak Ridge, TN.

Gregory, M. V., K. L. Shanahan and T. Hang, 1994, "A Dynamic Simulation Model of the Savannah River High Level Waste Tank Farm," Proceedings of the 1994 Simulation Multiconference, San Diego, CA.

Jantzen, C. M. and K. G. Brown, 1993, "Statistical Process Control of Glass Manufactured for Nuclear Waste Disposal," *American Ceramic Society Bulletin*, Vol. 72, (5): pp. 55-59.

Mallya, J. U., S. E. Zitney, S. Choudhary and M. A. Stadtherr, 1997, "A Parallel Block Frontal Solver For Large Scale Process Simulation: Reordering Effects," *Computers Chem. Engng.*, **21**, Suppl., S439-S444.

Paloschi, J. R. and S. E. Zitney, 1997, "Parallel Dynamic Simulation of Industrial Chemical Processes on Distributed-Memory Computers," AIChE National Meeting, Los Angeles, CA.

Paloschi, J. R., 1996, "Steady State Simulation Using MIMD Machines in the SPEEDUP Simulator," *Computers Chem. Engng.*, **20**, Suppl., S291-S296.

Paloschi, J. R., 1998, "Steps towards steady-state process simulation on MIMD machines: Implementation in the SPEEDUP simulator," *Computers Chem. Engng.*, **22** (9), 1189-1195.

Pantelides, C. C., 1988, "Speedup - Recent Advances in Process Simulation," *Computers Chem. Engng.*, **12** (7), 745-755.

Smith, F. G., 1998, "Modeling the Defense Waste Processing Facility with SPEEDUP," *Adv. In Envir. Res.*, **2** (3), 296-312.

Thole, C.-A. and K. Stuben, 1999, "Industrial simulation on parallel computers," *Parallel Computing*, **25**, 2015-2037.

Vegeais, J. A. and M. A. Stadtherr, 1990, "Vector processing strategies for chemical process flowsheeting," *AIChE J.*, **36**, 1687-1696.

Table 1. Chemical Species In DWPF Model

Liquid Phase Components					
Al_2O_3	B_2O_3	BO_3H_3	BaO	C_6H_6	CO_2H_2
CaO	$Ca_3P_2O_8$	Cr_2O_3	Cs_2O	$CsCO_2H$	CsTPB
CuN_2O_6	CuO	Fe_2O_3	H_2O	HNO_3	K_2O
KCO_2H	KTPB	Li_2O	MgO	MnO	NH_3
NH_4CO_2H	NH_4TPB	$NaCO_2H$	NaCl	NaF	$NaNO_2$
$NaNO_3$	NaOH	Na_2O	Na_2SO_4	$NaTi_2O_5H$	NaTPB
NiO	P_2O_5	SiO_2	TiO_2	U_3O_8	ZrO_2

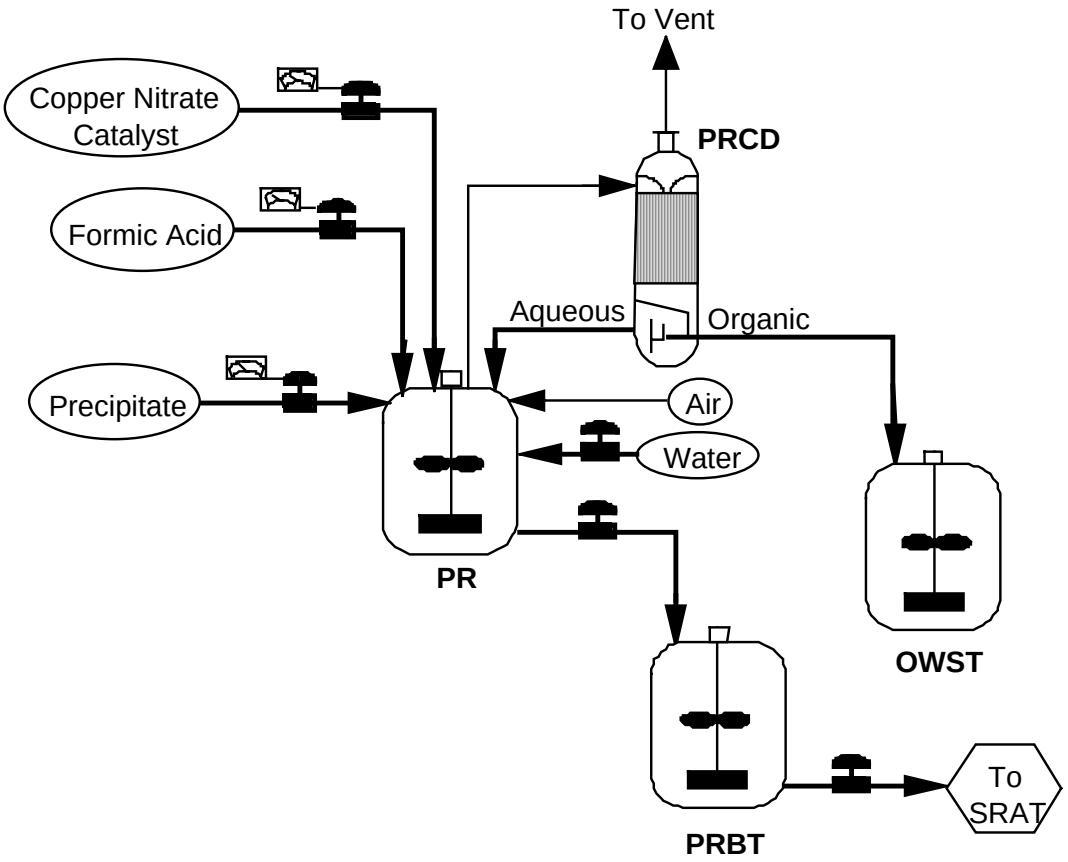
Vapor Phase Components					
C_6H_6	CO_2	CO_2H_2	H_2O	H_2	N_2
NH_3	NO_2	O_2			

Table 2. Two Node DWPF Model Timings in CPU Seconds.

Model	UNIX machine run times		Factor
	pvma	pvmb	
Full DWPF	807.9		1.00
Chem+Melt Salt Cell	493.7	60.0	1.64
Chem+Salt Melt Cell	362.8	82.8	2.23

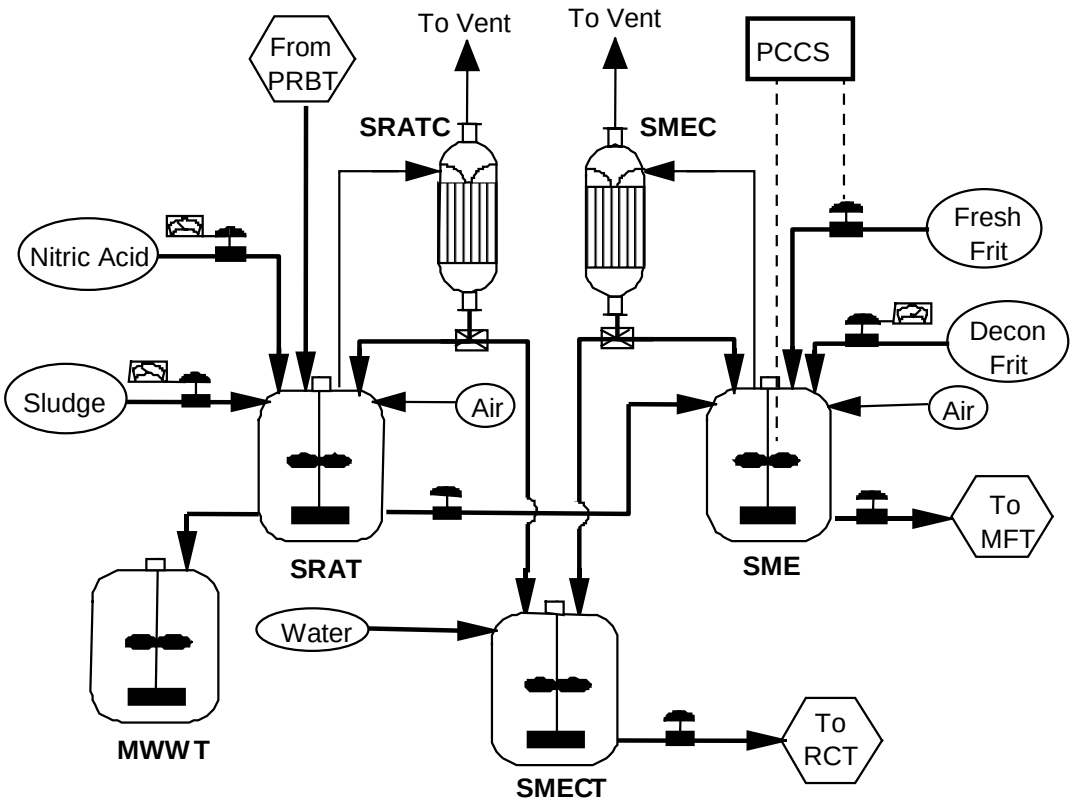
Table 3. Three Node DWPF Model Timings in CPU Seconds.

Model	UNIX machines run times			Factor
	pvmx	pvmy	pvmz	
Full DWPF	1362.4			1.00
Chem_Cell	289.6			4.70
Salt_Cell		96.3		
Melt_Cell			216.3	



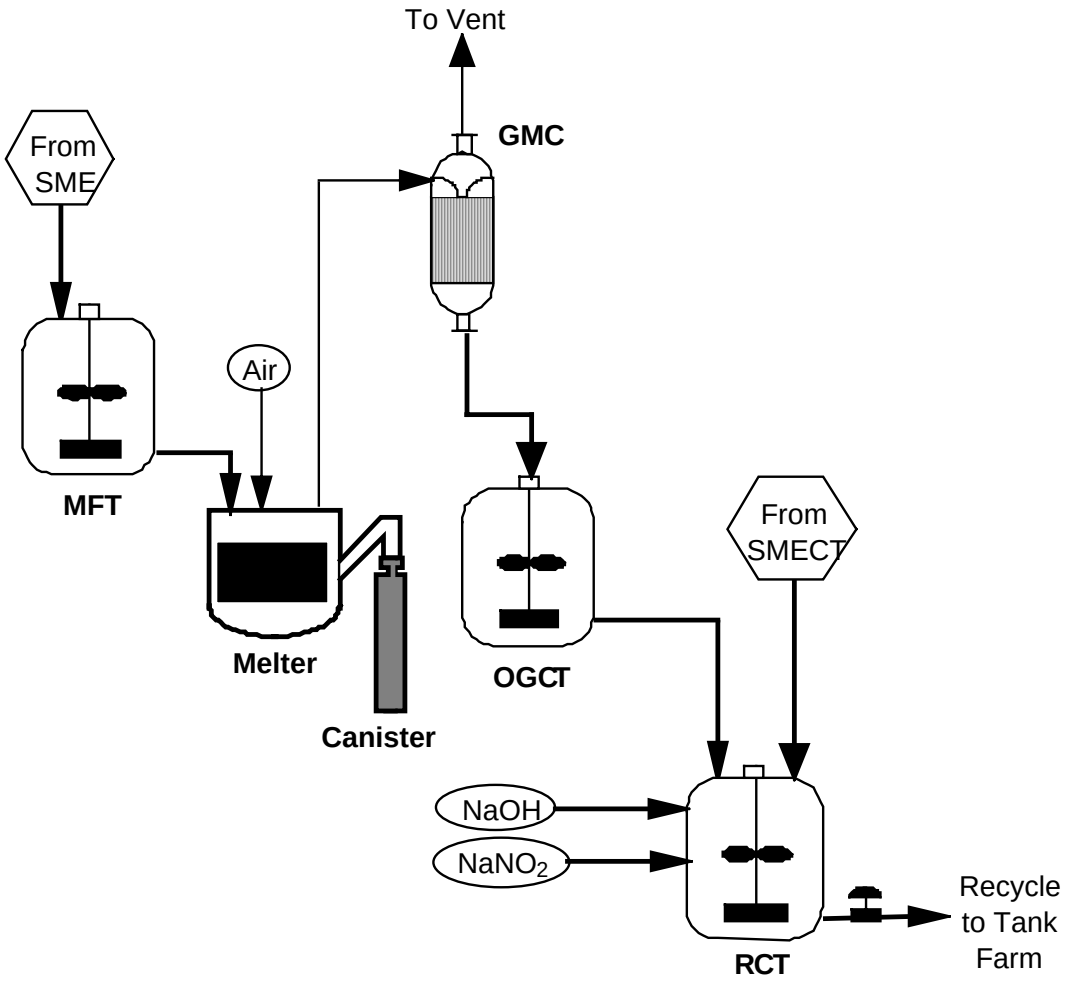
Unit ID	Full Name	Volume (gal)	Heel (gal)
PR	Precipitate Reactor	9000	1000
PRCD	Precipitate Reactor Condenser/Decanter	-	-
PRBT	Precipitate Reactor Bottoms Tank	9500	1500
OWST	Organic Waste Storage Tank	9000	1500

Figure 1. Model representation of DWPF Salt Cell.



Unit ID	Full Name	Volume (gal)	Heel (gal)
SRAT	Slurry Receipt and Adjustment Tank	12000	1500
SME	Slurry Mix Evaporator Tank	12000	1500
SRATC	SRAT Condenser	-	-
SMEC	SME Condenser	-	-
SMECT	SME Condensate Tank	9000	1500
MWWT	Mercury Water Wash Tank	1000	100

Figure 2. Model representation of DWPF Chemical Cell.



Unit ID	Unit Name	Volume (gal)	Heel (gal)
RCT	Receipt Collection Tank	9000	1500
OGCT	Off Gas Condensate Tank	9000	1500
GMC	Glass Melter Condenser	-	-
MFT	Melter Feed Tank	9000	1500
Melter	Glass Melter	1000	630
Canister	Glass Canister	180	0

Figure 3. Model representation of DWPF Melt Cell.

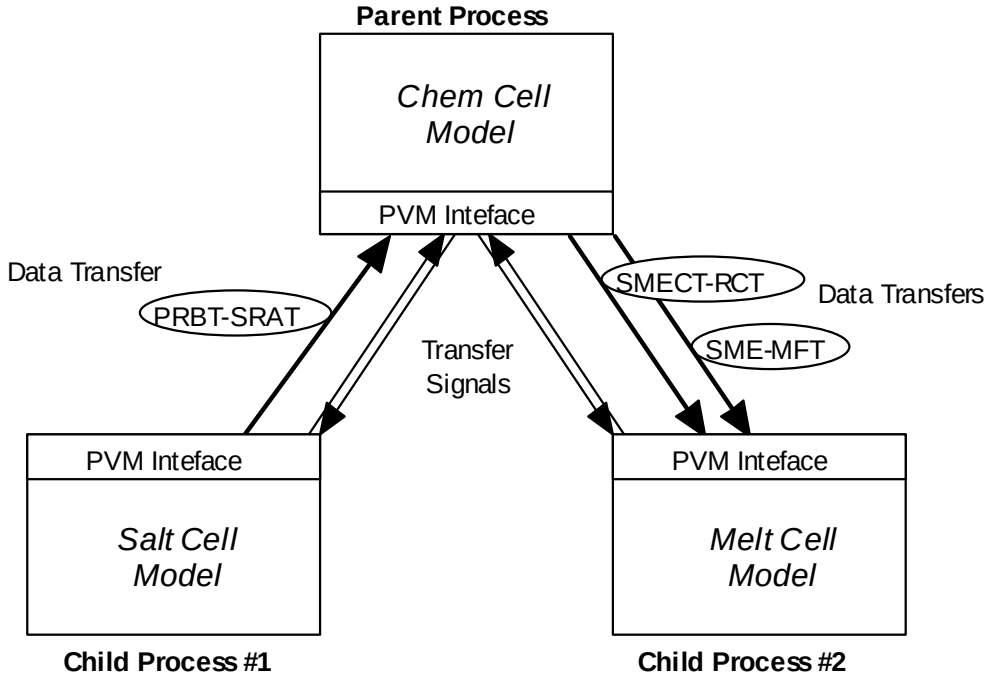


Figure 4. Parallel computing system used for DWPF model.

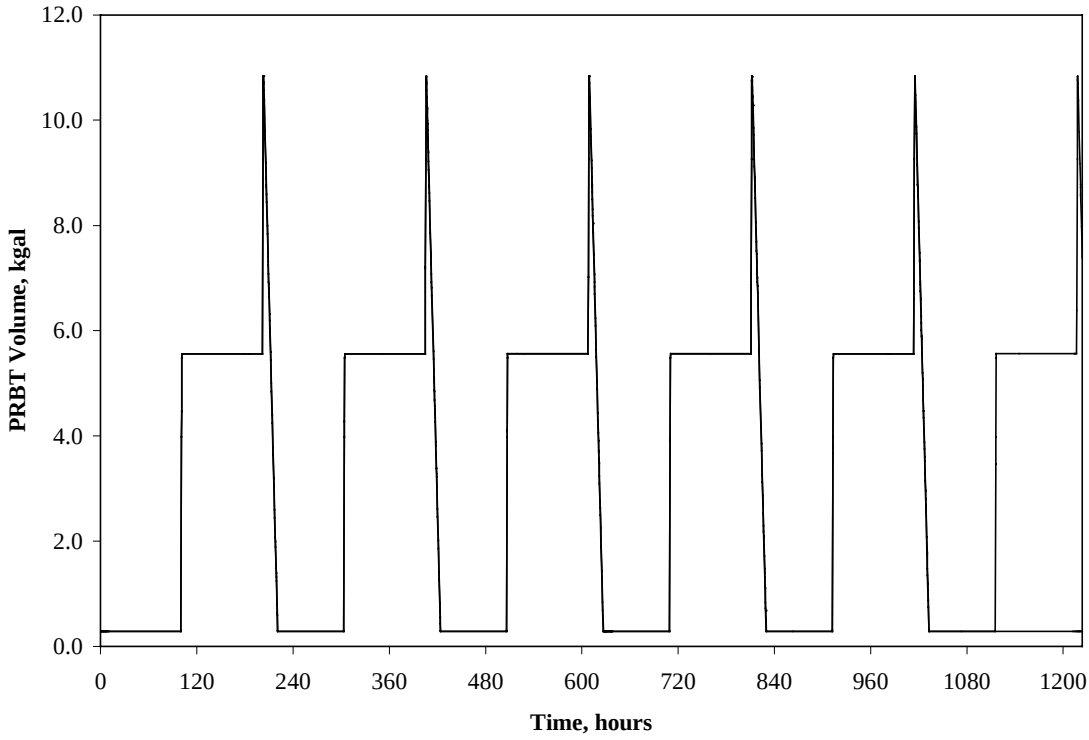


Figure 5. Time history of PRBT liquid volume calculated by serial and parallel models.

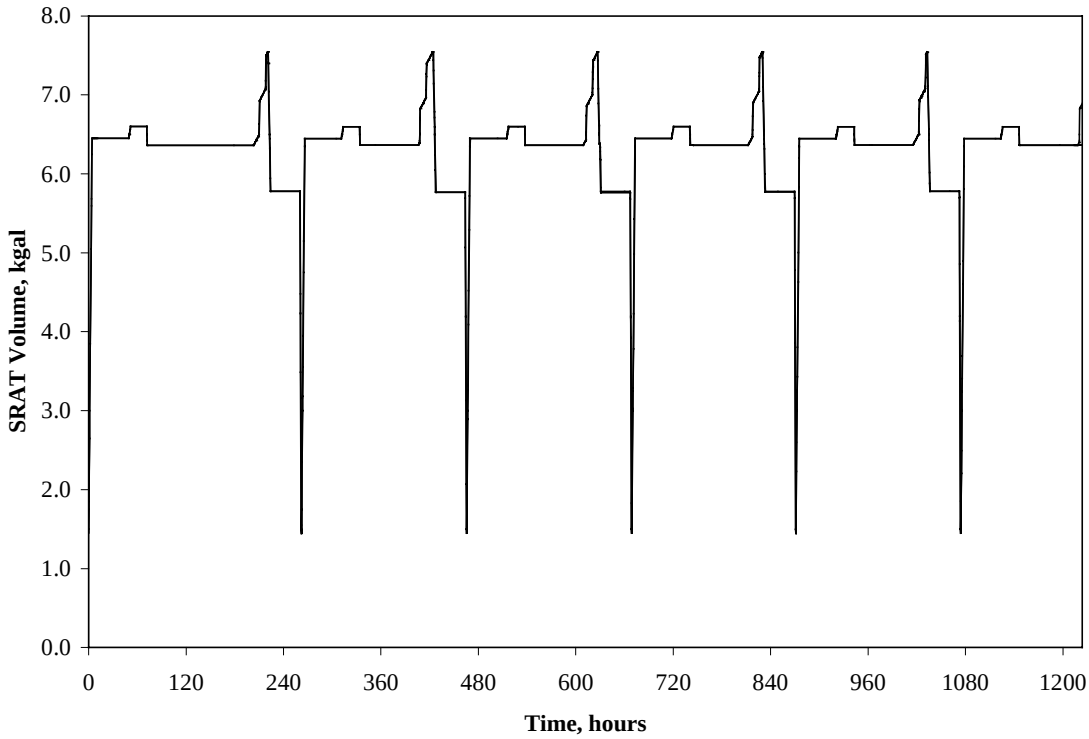


Figure 6. Time history of SRAT liquid volume calculated by serial and parallel models.

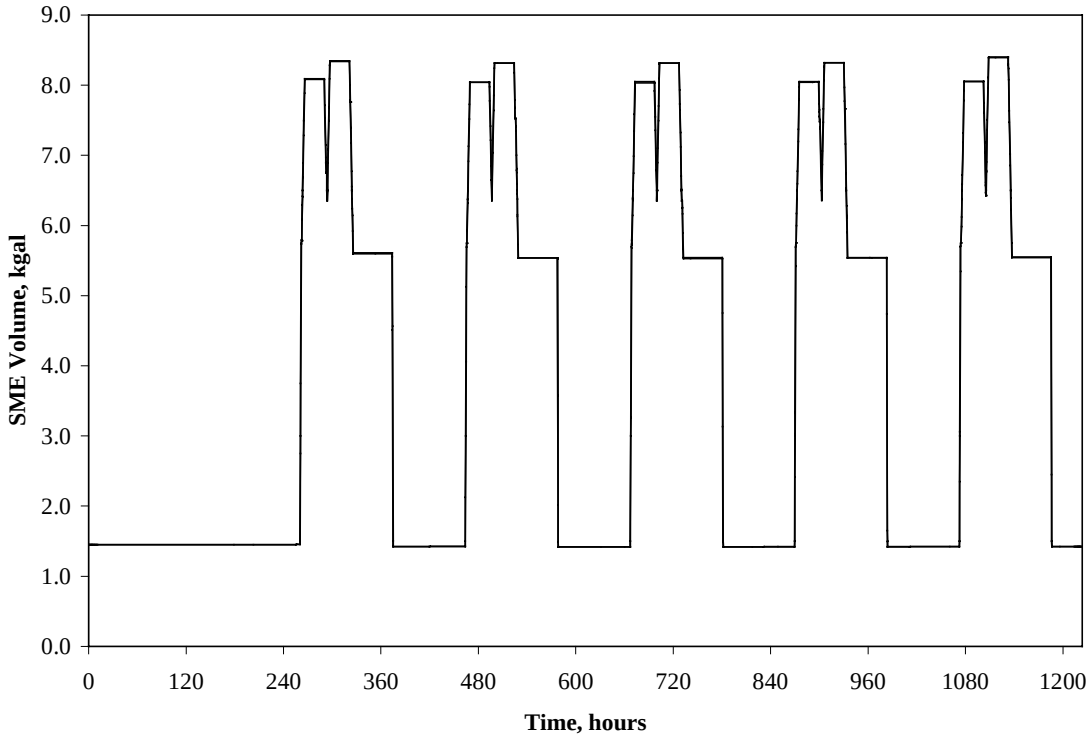


Figure 7. Time history of SME liquid volume calculated by serial and parallel models.

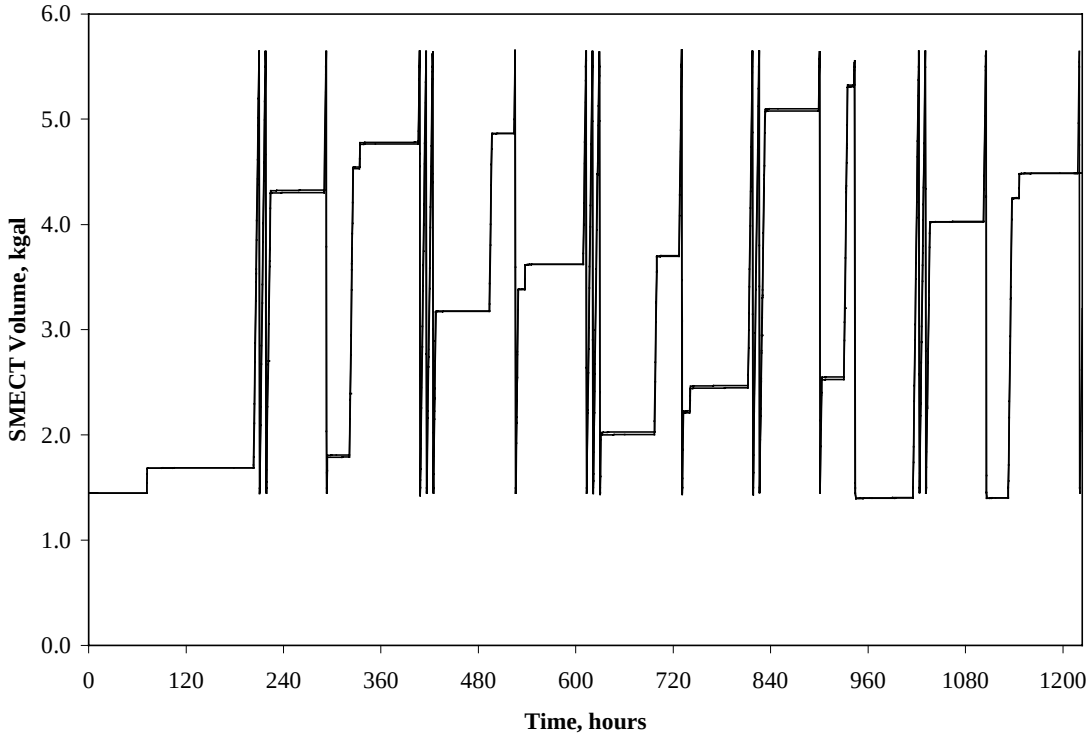


Figure 8. Time history of SMECT liquid volume calculated by serial and parallel models.

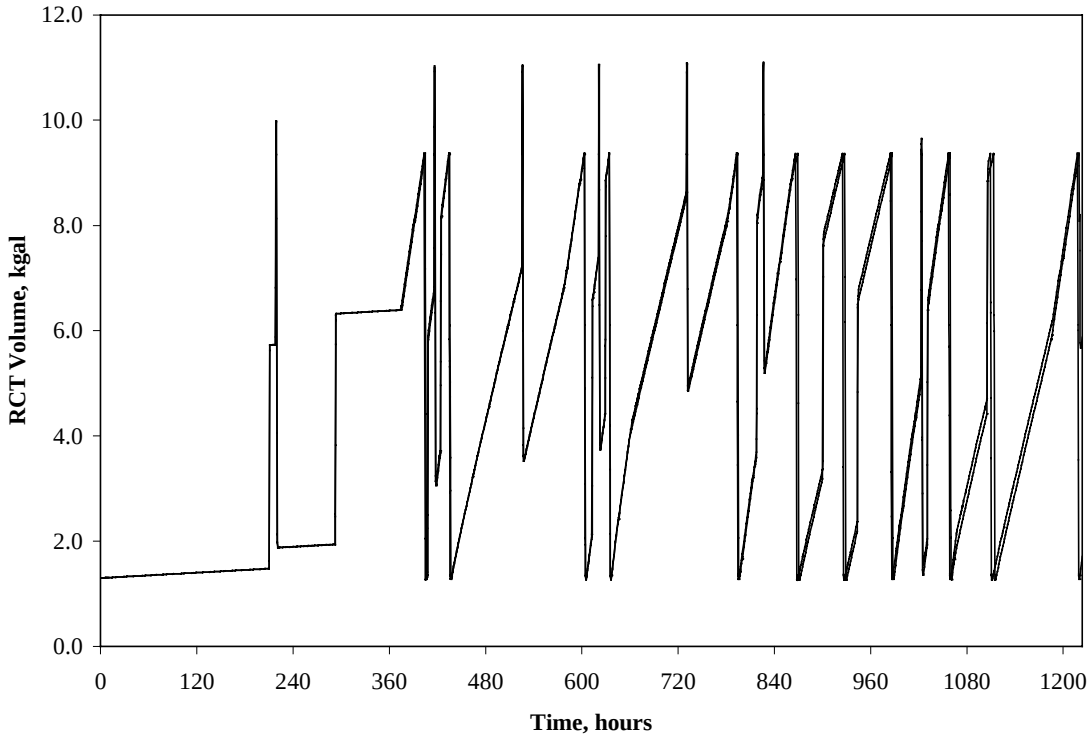


Figure 9. Time history of RCT liquid volume calculated by serial and parallel models.

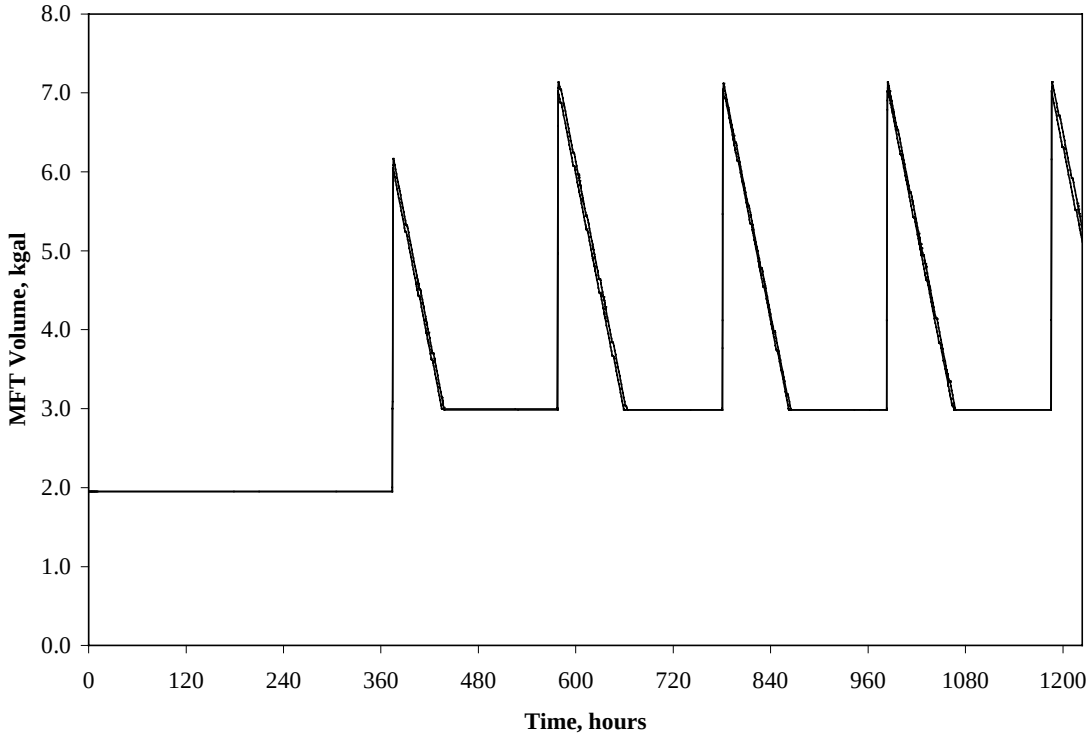


Figure 10. Time history of MFT liquid volume calculated by serial and parallel models.

Appendix: FORTRAN procedure to implement Salt Cell PVM-SPEEDUP interface.

```
PROCEDURE XFER_CHEM

     Input TIME, VOLUME, LIQDENSITY,
           MASSFRACTION(LIQCOMP),
           NOUNIT, NUMBER

    Output NUMBER, NOUNIT

PRECALL
POSTCALL

Code
c
c     subroutine to transfer information between
c     salt cell and chemical cell
c
      subroutine xfer_chem (time,pha_vol,pha_rho,
     &                      pha_con,lpha,
     &                      pha_sig,pha_bat,
     &                      pha_dt ,pha_mt,
     &                      ifail,itype,icall)
      implicit double precision (a-h,o-z)

      include '../fpvmz.h'

      dimension pha_con(lpha)

      character*5 tank
      integer tids(0:2)
      logical x_pha

      data delt   / 0.0    /
      data tset   / 0.001  /
      data isignl / 0      /
      data x_pha  / .true. /

      nbatch = nint(pha_bat)
      nsignl = nint(pha_sig)
c
c     no mesage encoding, assume that parallel
c     machines can all recognize native format
c
      msgcode = 1

      if (icall.eq.1) then
c
c     enroll this program in PVM as a child process
c     and get the parent tid
c
         call pvmfmytid  (mytid)
         call pvmfparent (mptid)

         write (*,'(/a,i6 )') ' salt cell ( child) tid = ', mytid
         write (*,'( a,i6/)') ' chem cell (parent) tid = ', mptid

      else if (icall.eq.2) then
c
c     leave PVM before exiting
c
         write (*,'(/a/)') ' salt cell process exiting pvm'
         call pvmfexit (info)
      else

         if (time.gt.tset) then
            if (.not.x_pha .and. nsignl.eq.1) then
c
c     wait for signal from chemical cell that srat is ready
c     for a transfer
```

```
c
               msgtype = 1
               call pvmfrecv   (mptid   ,msgtype   ,info)
               call pvmfunpack (integer4,isignl,1,1,info)
               call pvmfunpack (real8   ,time0 ,1,1,info)

               if (isignl.eq.-1) then
                  ifail = 4
                  write (*,'(a)') ' salt cell abort signal'
                  return
               end if
c
c     send a signal back to the chemical cell with the prbt
c     batch number, pha volume, density and composition
c
                  tank = ' prbt'
               msgtype = 2
               call transmit (tank,mptid,msgtype,msgcode,
     &                        nbatch,lpha,pha_rho,pha_vol,
     &                        pha_con,time,time0,delt)

               x_pha = .true.
            else if (x_pha .and. nsignl.eq.0) then
               x_pha = .false.
               isignl = 0
            end if
         end if
      end if

      tset   = time
      pha_dt = delt
      pha_mt = float(isignl)

      return
      end
c
c     subroutine to transmit pvm messages
c     containing transfer information
c
      subroutine transmit (tank,mptid,msgtype,msgcode,
     &                     nbatch,ncomp,density,volume,
     &                     conc,time,time0,delt)

      implicit double precision (a-h,o-z)

      include '../fpvmz.h'

      dimension conc(ncomp)

      character*5 tank

      call pvmfinitsend (msgcode                 ,info)
      call pvmfpack     (real8   ,time   ,1    ,1,info)
      call pvmfpack     (integer4,nbatch ,1    ,1,info)
      call pvmfpack     (integer4,ncomp  ,1    ,1,info)
      call pvmfpack     (real8   ,density,1    ,1,info)
      call pvmfpack     (real8   ,volume ,1    ,1,info)
      call pvmfpack     (real8   ,conc   ,ncomp,1,info)
      call pvmfsend     (mptid   ,msgtype        ,info)
c
c     write out transfer information
c
      write (*,'(  /,a,f12.5)')  '         time = ', time
      write (*,'(2x,2a,i2   )')  tank,  ' batch = ', nbatch
      write (*,'(    a,i2   )')  '   components = ', ncomp
      write (*,'(   2a,f12.5)')  tank,' density = ', density
      write (*,'(1x,2a,f12.5)')  tank, ' volume = ', volume
      do i=1,ncomp
         write (*,'(4x,2a,i2,a,f12.5)') tank,'(',i,') = ', conc(i)
      end do
c
```

```
c     check for time deficit between parts of process
c
      if (time0.gt.time) then
         delt = delt + time0 - time
      end if
      write (*,'(a,f12.5)') ' time deficit = ', delt

      return
      end
$Endcode
****
```