Parallelization of a General-Purpose Reservoir Simulator

John E. Killough, University of Houston*
John A. Foster, Western Atlas Software*
James S. Nolen, Western Atlas Software**
John R. Wallis, Western Atlas Software**
Jason Xiao, Western Atlas Software*

Paper presented at the 5th European Conference on the Mathematics of Oil Recovery, Leoben, Austria, 3-6 Sept. 1996

ABSTRACT

Over the past few years advances in parallel computing have been significant. Large local memories, fast interprocessor communications, and high speed superscalar nodes have significantly improved the utility of this hardware for reservoir simulation. The following paper discusses a general-purpose reservoir simulator which has been ported to three parallel computers: the Cray T3D, the IBM SP2 and the Silicon Graphics Power Challenge. The unique feature of the model is its ability to run efficiently in either serial or parallel environments. Several test problems are presented ranging from IMPES black-oil with severe faulting to a fully-implicit six-component compositional simulation. Good to excellent parallel efficiencies are observed for all cases. Two parallel solvers form part of the simulator: a strip domain decomposition scheme and a technique based on composite grid decomposition. Both of these solvers show excellent parallel efficiency. A static load-balancing example is presented utilizing the flexibility of the local grid refinement data structure to achieve balance. Finally, prototype results for clusters of workstations show excellent parallel speedups, for a limited number of nodes, for both compositional and black-oil models.

INTRODUCTION

The motivation for parallel computing in reservoir simulation has always existed. From the earliest simulation models, computing resources have been severely taxed simply because the level of complexity desired by the engineer almost always exceeded the speed and memory of the hardware. The high-speed vector processors of the late seventies and early eighties led to orders of magnitude improvement in speed of computation and led to production models of several hundred thousand cells. The relief brought by these models unfortunately was short-lived. The desire for increased physics of compositional modeling and the introduction of geostatistically/structurally based

geological models led to increases in computational complexity even beyond the large scale models of the vector processors. Tens of millions of cells with complete reservoir parameters now became available for use by the engineer. Although upscaling provided a tool to dramatically reduce model sizes, the inherent assumptions of the upscaling technique left a strong desire by the engineer to incorporate all of the available data in studies. The only available solution to this problem became the subdivision of the model into small segments and the use of parallel computers.

The obvious attraction of parallel computing is the ability to reduce the elapsed time of a simulation by a factor of n for an n -processor computer. To accomplish this several factors are involved. Significant reduction in parallel efficiency can result from recursion in the linear equation solution, from the overhead of message-passing required for implicit Newton iterations, and from load imbalance due to heterogeneities in problem physics and reservoir characterization. To minimize these factors, the parallel simulator must maintain great flexibility and/or adaptability so the most efficient parallel data structures can be utilized for a given simulation or even a given timestep. Several authors have presented results on the parallelization of reservoir simulation models¹⁻⁵. The unique features of this work are the use of a general-purpose model and the application of local grid refinement for flexible parallel decomposition.

This article describes the techniques which have been used in a general-purpose, three-dimensional, compositional simulator to achieve high parallel efficiency. The base simulator is first described. This is followed by an explanation of the utilization of the local grid refinement data structure for parallelization. Next the two parallel linear equation solvers are presented. Finally, a set of test problems and results for both the linear equation solver and the full simulator are discussed.

* Now with Landmark Graphics Corporation

** Now with Geoquest Reservoir Technologies

DESCRIPTION OF THE SIMULATOR

The commercial reservoir simulator which formed the basis for this study, VIP, is a full-featured reservoir simulation software system for UNIX workstations⁶. The program is a multi-component, three dimensional, three phase reservoir simulator which contains a number of modules sharing a common compositional formulation. Both fully implicit and IMPES model formulations of the differential equations governing conservation and flow in the reservoir may be utilized including the relaxed volume-balance technique. A variety of matrix solution methods are provided for use with both formulations, including both direct and iterative procedures. In general, the most efficient solution method is a proprietary solver, referred to later in this work as the "serial solver", which uses residual constraints combined with preconditioned generalized minimum residual acceleration (GMRES) to substantially reduce both work and storage requirements⁷. The model contains as a significant component, a local grid refinement capability.

LOCAL GRID REFINEMENT

The vehicle for implementation of parallelism in the simulator was the existing code for simulation on locally refined grids. The methodology for local grid refinement became the basis for domain decomposition, both for models built with locally refined grids and for single grid models. Thus, it is useful to know the rules that govern the construction of locally refined grids in this code.

1. There is no limit on the number of levels of nested refinements.
2. Refinements may be either hexahedral corner point grids or cylindrical (2D or 3D) grids.
3. A cylindrical refinement cannot be further refined.
4. Any grid may be specified to use either IMPES or implicit finite difference approximations without regard to the implicitness of any other grid.

Intercellular flow in a locally refined grid is divided into two categories that require different methods of computation, intragrid flow and intergrid flow.

Intragrid flow refers to flow between two cells that are included in the standard finite difference stencil and additionally are contained within the same grid. Data required to compute flow between the two cells is directly addressed, and contributions to the residuals and to the Jacobian terms arising from the intercellular flow are computed simultaneously for both cells. The code executed is the same code used for single grid problems that contain no faults.

Intergrid flow refers to flow between two cells that touch each other across the boundary between two grids or between two cells that touch each other across a fault

within a single grid. In either case data for both cells is indirectly addressed. Flow between two cells within a single grid that connect across a fault is not distinguished from flow between two separate grids. For coding simplicity such connections are treated as grid to grid connections, where the two grids happen to be identical.

For grid to grid connections between two cells, contributions to the residuals and to the Jacobian terms arising from intercellular flow are computed separately for each cell. This means that for the sake of simplicity (and to reduce the amount of message passing in parallel operations) the flow rate and certain Jacobian contributions are unnecessarily computed twice. The penalty for this inefficiency is believed to be insignificant, as long as the number of boundary cells is small relative to the total number of cells.

When flow is computed between a cell in any given grid (Grid 1) and a cell in a neighboring grid (Grid 2), the data for Grid 2 is not addressed at its permanent storage location. Instead, data for all grid cells in Grid 2 that communicate with cells in Grid 1 are first packed into a buffer and then indirectly addressed within the buffer. This unnecessary movement of data is performed to achieve compatibility with parallel computing procedures and to simplify program logic. It may have the side benefit of reducing the number of cache misses, since the data used in the calculations will be stored in closer proximity.

PARALLELIZATION VIA LOCAL GRID REFINEMENT

As described above, domain decomposition is achieved by means of local grid refinement. The local refinement may be either real or simply a mechanism for decomposing the grid. Local grid refinement rules are relaxed, allowing grids of the same level to touch each other, a condition that is discouraged in single processor runs, since it can adversely affect the rate of convergence of the linear solver. With few restrictions any of the local grids may be assigned to any processor to achieve parallelization and load balance.

Each processor executes the same code, allocating storage and operating only on those grids that have been assigned to it. Intragrid flow calculations are performed with the same code used in single processor runs. With one exception intergrid flow calculations are also performed with the same code used in single processor runs. This exception involves the procedure used to pack the storage buffers.

If two communicating grids (Grids 1 and 2) lie on separate processors (Processors A and B), it is necessary to create duplicate storage buffers to perform intergrid flow calculations. In order to feed Processor A, a "send buffer" on Processor B is loaded with all data required to compute flow between Grids 1 and 2. This data is sent as a message from the send buffer on Processor B to a corresponding receive buffer on Processor A. The flow calculations are then performed on Processor A by

indirectly addressing data in the receive buffer. This procedure is reversed to compute intergrid flow on Processor B. (See Figure 1.) The duplicate flow calculations are executed simultaneously. If two grids lie on the same processor, send and receive buffers are used in the same fashion, except they have the same address, and no message is sent. Figure 1 depicts the case for a 2x2 decomposition on four processors.

For parallel domain decomposition the base grid is totally refined into subgrids so that all gridblocks in the base grid become inactive. Each grid is assigned to a processor so that only a portion of the model resides on a given processor. In this manner the simulator memory requirements become scaleable. That is, as the number of processors is increased, the grid assigned to a given processor becomes smaller.

For example, consider the simple case shown in Figure 2 in which the base grid consists of four blocks. Each of these blocks is then refined to the level desired for a given simulation. Grid 1 corresponds to the refinement of the upper left block in the base grid and is assigned to processor A. Similarly, each refinement of the base grid is assigned to a different (or the same) processor. At the end of the decomposition the complete base grid is refined and all refined blocks have been assigned to a processor. Multiple levels of grid refinements can be accommodated as well. If further refinement of a cell is required to a radial or Cartesian grid, the further refinement is assigned to a processor in the same manner as the level 1 refinements. To accommodate the case of existing datasets, 1x1 refinements are performed on the base grid to assign each cell to a subgrid and associated processor.

WELL MANAGEMENT

Since highly deviated and horizontal wells are so prevalent in modern oil field operations, it would be excessively restrictive to require that each individual well be contained entirely within a single parallel computing domain. Therefore, it must be assumed that messages will have to be passed between domains in order to gather all of the information necessary to allocate production and injection rates among multiple perforated intervals. Furthermore, use of production or injection constraints that involve groups of wells will, in general, lead to interdomain coupling of well rates.

Allocation of well flow rates requires such grid cell dependent data as pressures, saturations, and fluid composition. (This list is not all inclusive.) The base simulator was originally developed for high efficiency on vector processing computers⁶. In order to allow unit strides in addressing data in the well management code, all grid cell dependent data required for well calculations has always been moved into arrays dimensioned by the total number of perforated intervals at the beginning of each timestep or each nonlinear iteration, as appropriate. Thus, the well management code has never addressed data stored by grid cells.

At the beginning of each nonlinear iteration of a parallel simulation, each processor fills its perforation arrays to the extent possible with the grid cell data contained in its local memory. This data is then compressed into a send buffer and broadcast to all other processors in a single send per receiving processor. Upon receipt, the data is moved from the receive buffer to the perforation arrays by the receiving processor.

Upon conclusion of these operations each processor contains an identical copy of the fully loaded perforation arrays. This allows all of the wells subject to a group rate constraint to be allocated on the same processor, greatly simplifying the parallelization process. Shifting wells from one processor to another in order to achieve dynamic load balance also becomes easier when the data is available everywhere.

At of this writing no parallelization of the well management code has been attempted, so all processors simultaneously compute rates for all wells. The run times reported reflect this sequential step in the simulation process. Parallelization using the data flow described above is still possible; however, additional efficiency can be achieved by replacing the broadcast operations with more selective (but more complex) message passing.

THE PARALLEL LINEAR EQUATION SOLVERS

Two parallel linear equation solvers form the basis for the parallel reservoir simulator. These are both based on domain decomposition strategies using nested factorization rowsum constraints. Each solver provides capabilities for complexities such as faults, implicit bottomhole pressures, and horizontal wells.

The first solver involves a linear decomposition and is based on the work of Wallis⁸. The decomposition involves large domains and boundary planes which divide the grid along a single direction as shown in Figure 3. The approximate LU factorization approach used for preconditioning involves an L factor as follows:

$$L = \begin{pmatrix} A_1 & & & & & \\ & A_2 & & & & \\ & & A_3 & & & \\ & & & A_4 & & \\ Q_1 & R_2 & & & B_1 & \\ & Q_2 & R_3 & & & B_2 \\ & & Q_3 & R_4 & & B_3 \end{pmatrix}$$

The diagonals of the B submatrices are modified from the original coefficients to maintain rowsum constraints. This approximate factorization leads to independent solutions on each processor for the large A_i domains and the boundary plane matrices, B_i . The back substitution portion of the

preconditioning simply involves the updating of residuals from boundary solutions, and a second solve on the large A_i domains. To avoid the work associated with two large domain solutions, a unique two-way ordering is used such that the entire LU factorization involves only a single solution for the A_i matrices.

The second solver is based on a composite grid solution technique developed by Wallis and Nolen⁹. As described above, each grid is assigned to a processor. These may be the same or different processors. If all active grids lie on the same level, the composite grid solver performs all preconditioning solutions independently, i. e., block Jacobi. If grids are assigned to different levels, each grid of the same level is solved simultaneously with communication being performed after the entire solution of all grids on a level is completed. Factorization is performed so that rowsum constraints are also imposed. Figure 4 shows an example of a possible decomposition which can be solved with the composite grid technique (See Reference 10 for additional details.) First, the larger domains of level 3 are solved. Information from these solutions is then communicated across boundaries where necessary. Next the equations are solved for level 2, and finally, level 1 is solved. Each of the lower level solutions involve the use of rowsum constraints. Back substitutions are then performed in the reverse order. The flexibility of this solver allows the easy implementation of block red-black algorithms, or as a special case, the linear decomposition of the strip decomposition solver described above.

Figures 5 and 6 show results for both solution methods on a 95x95x35 (315,875 cells) hypothetical IMPES case. The efficiency of the solvers was compared to both a serial version of the same technique and to the proprietary "serial solver" of the original reservoir simulator used in this work. As shown in Figure 5, performance of the strip decomposition solver is superlinear for almost all cases ranging from 3-16 processors. Compared to the serial solver, superlinear performance is also observed. Superlinear performance was also observed for the composite grid solver compared to the serial version of the same algorithm. Compared to the "serial solver", the composite grid technique achieved about 75% of ideal speedup, i. e., a factor of twelve on sixteen processors. The speed advantage for the strip method results from the two-way ordering discussed above. Other studies have shown similar superlinear speedups on up to 71 processors and 4 million cells for the strip decomposition technique.

Because of its inherent computational advantage, the strip decomposition solver was used exclusively for the results of this work. As will be discussed later, the robustness of the solvers remains a continuing question. With the flexibility of the composite grid method, many approaches remain to be investigated. This will form the basis of future investigations.

PARALLEL COMPUTERS USED IN THIS STUDY

This study was conducted on the Cray T3D, the IBM SP2 and the Silicon Graphics Power Challenge. Both the IBM SP2 and the Cray T3D are parallel computers built with the MIMD (multiple instruction, multiple data) architecture and distributed memories. The number of processors for the SP2 can range to 512 while the Cray has a maximum of 1024 processors. For the SP2 with "Thin-2" nodes used in this study, memory can be expanded up to 512 Mbytes while the maximum for the T3D is 64 Mbytes. The SGI machine is a shared-memory device with maximum memory exceeding 2048 million bytes and up to sixteen processors. Peak single-processor performance for all machines is in the range of 200 double precision MFLOPS (Million Floating point Operations Per Second). Actual performance varies considerably for the individual processors. Typically for reservoir simulation, the SP2 processor delivers about 20-40 MFLOPS compared to about 10-20 MFLOPS for the T3D using optimized FORTRAN compilation. The SGI Power Challenge appears to deliver on the order of 20-30 MFLOPS for the problems tested in this study. The primary consideration for performance is strongly tied to memory access and cache misses. Since memory is relatively slow for all processors (perhaps as much as 8 to 1 compared to CPU cycle time), the ability for the compiler/system to anticipate memory references and provide data to the cache can uniquely determine processor performance, regardless of the raw speed of the processor itself. That is, the data must be available for the processor to use. Since the SP2 has a delivered performance of about 1.5-3 times that of the other computers, the overhead of message-passing becomes dramatically more important on the IBM machine.

Message-passing was utilized for parallelization on all machines. Although the SGI computer is shared memory, message-passing is provided which takes advantage of the shared-memory hardware. PVM (Parallel Virtual Machine) was used to first implement parallelization of the simulator. MPL (Message-Passing Language - IBM) and MPI (Message-Passing Interface) were later added as options for use by the parallel model. All results for the Cray T3D and SGI Power Challenge were produced using PVM. As discussed later, the use of the machine-specific version of PVM for these computers appears to have had little detrimental affect on processor performance. For the IBM SP2, on the other hand, PVM performance has been shown to be substantially less than the other available message-passing libraries - MPL and MPI. For this reason all results for the SP2 are reported using MPI. (MPI performance was found to be virtually identical to MPL.)

The interconnect topology for the distributed memory T3D and SP2 are somewhat different. The T3D utilizes a "toroidal" three-dimensional structure while the IBM contains an elaborate electronic switch to provide fast communications among processors. The latency for the T3D is about 6 microseconds compared to 40 microseconds for the SP2. Bi-directional data bandwidth is

approximately 40 Mbytes per second for the SP2 and greater than 100 Mbytes per second for the T3D.

There are generally three forms of message passing for a distributed memory system: synchronous, asynchronous, and global operations. Synchronous message passing requires that all messages be completed before computations can proceed. Asynchronous, on the other hand, allows some overlap of processing and communications. Global message passing operations distribute data from single processors to one processor or visa versa. Often some form of arithmetic function such as summation is performed as part of the global concatenation operation. These functions are particularly useful for evaluation of reduction functions such as dot products or for distribution of data from a single node to all nodes in the parallel partition. Often the global communications have been optimized for a particular architecture and are much more efficient than a simple set of send-receive operations for achieving the same results. Because of the greater raw processor speed and lower communications performance, efficiency of message-passing is of great importance on the SP2 to achieve good parallel performance. This implies that utilization of asynchronous message-passing must be emphasized to overlap compute work with communication overhead. Asynchronous communications were utilized wherever feasible in the reservoir simulator and especially in the solvers. Synchronous and global operations were also used for this study in cases where efficiency was not a major concern because of the length and number of messages involved.

TEST CASES

Five test cases are presented in the following sections which demonstrate the application of the parallel simulator in a combination of black-oil and compositional studies utilizing both the IMPES, and fully-implicit solution techniques. Two of the cases emphasize real-world heterogeneities including faults. Since the strip decomposition solver was used for all cases, decompositions were performed in only a single dimension, either x or y, for all cases.

Test Problem 1 - Small Black-Oil

The first test problem is based on the data from the SPE First Comparative solution problem. Instead of the original dimensions; however, the problem size was increased to 95x27x3 - 7695 cells. Fourteen wells were used instead of two. Figure 7 depicts the structure of the model and the well locations. This problem was used to test the simulator for all of the three parallel machines used in this study (T3D, SP2, and SGI). The model was decomposed along the x-direction as shown in the figure.

The flexibility of local grid refinement allowed decomposition of Test Problem 1 into 4, 8, and 16 domains with relative ease. First, 4, 8, or 16 large domains were defined with given physical gridblock sizes for grid level 0. These domains were then refined in all directions.

Refinements in the y- and z-directions were held constant at 27 fold and 3 fold respectively for all cases. For four domains, each large block in grid level 0 was refined by a factor of 24 with the last block having a factor of 23. For 8 and 16 domains the refinement was 12 and 6, respectively. Factors of 11 and 5 were used for the last domain in these cases, respectively. In this fashion, a decomposition was achieved which provided an almost perfectly load-balanced case for the strip decomposition parallel solver, i.e., one boundary yz-plane existed between equal large domains.

Test Problem 2 - Moderate-Size Black-Oil

The size of the second test problem was increased to 95x95x10 - 90250 cells. The domain decomposition used exactly the same approach as for Test Problem 1 except that the y- and z-direction refinements were increased to 95 and 10 respectively. Tests for up to sixteen processors were also used for Test Problem 2.

Test Problem 3 - Large, Faulted, Heterogeneous, Black-Oil

Test Problem 3 was developed by first utilizing a two-dimensional geostatistical mapping package for defining heterogeneities in forty geological horizons. An initial grid of 2.4 million cells formed the description. Although it would be desirable to simulate this entire dataset, upscaling and layer combining were used to reduce the eventual model to a 64x25x12 - 19200 cells. These cells were then refined by two in each dimension to increase the overall problem size to 153,600 cells (a moderately large model - at least by today's standards).

In addition to the heterogeneities in permeabilities and porosities provided through geostatistics, the model contained three large "scissors" faults with some sloping character as shown in Figure 8. Eighty-eight wells were spaced throughout the structure as shown in the figure with an equal number of injectors and producers. Completions were in all twenty-four layers, for all wells for a total of 2112 completions. Because of the heterogeneities, the use of completions in all layers of the model tests the implicit well bottomhole pressure option for rate specified wells - an integral part of the linear equation solver.

Typical PVT and relative permeability data for a middle eastern reservoir were also chosen for the problem.

Test Problem 4 - Real Compositional Example

The compositional simulation previously presented by Killough, et al.¹¹, is used to examine the problem of load balancing for a simulation of the real gas condensate field example. The original model was 22x29x5 or 3192 cells and 6 components. Sixteen wells were placed throughout the structure as shown in Figure 9. All wells were completed in all layers of the model. Since the largest dimension was in the y-direction, decomposition was performed linearly in y. That is, four domains were created as shown with initially 8, 7, 7, and 7 xz-planes each, respectively. Modifications were performed to this decomposition to achieve load balance.

Test Problem 5 - Refined Compositional Example

Test Problem 4 was refined by a factor of 3, 3, and 2 in the x-, y-, and z-directions, respectively, for an eighteen-fold increase in the total number of cells (57,420). Decomposition was maintained in the y-direction with each of the four domains containing 24, 21, 21, and 21 xz-planes. Load balancing was also investigated for this example through reallocating different numbers of planes to each of the domains as discussed below. Finally, a fully-implicit version of this six-component model was simulated to compare with the IMPES version and to demonstrate the robustness of the parallelization.

RESULTS

Test Problem 1

Simulations for Test Problem 1 were run on each of the targeted parallel platforms. Since it was not the purpose of this paper to compare the performance of different machines, the runs were made for different numbers of timesteps. Speedups were compared rather than raw elapsed times. Results for the Cray T3D are shown in Figure 10 for the cases of 4 to 32 processors. The speedups used for these comparisons were based on the original serial model using an unrefined grid. That is, a regular 95x27x3 grid with the same properties and well data was used for the comparison. These speedups then represent actual improvement in performance over current serial simulator capabilities. As shown in Figure 10 simulator performance for the T3D was superlinear for four and eight processors. As the number of processors increased to 16 and 32, the speedups showed some degradations due to an increase by a factor of three in the number of linear equation solution iterations. These encouraging results were not duplicated on the SGI computer as shown in Figure 11. Above four processors there appears to be a significant falloff in performance. Some of this can be attributed to linear solver degradation similar to the T3D results; however, the decline in performance for the case of 16 processors indicates that a significant bottleneck may exist. This may be the result of limited bandwidth to memory since on a shared memory computer, such as the Power Challenge, the bandwidth to memory does not increase linearly with the number of processors as is the case for both the T3D and SP2. Similar SGI performance was also noted for standalone tests of the strip decomposition linear equation solver. To some extent these results are preliminary; we are continuing to investigate the SGI computer to verify this performance. Results in Figure 12 for the SP2 show good speedups with a factor of twelve for the 16 processor case. Considering that the individual node on the SP2 is somewhat faster than that of the T3D, these results again show excellent performance.

Results for the remainder of the cases are presented only for the SP2 due to the availability of large-memory SP2 nodes for testing purposes.

Test Problem 2 Results

Test Problem 2 represented an increase in problem size of a factor of about 12 over Test Problem 1. Due to the strip decomposition, the amount of message passing also increased by a factor of 12 since the message sizes are proportional to the yz-plane dimension (95x10 versus 27x3). With the exception of the linear equation solution, computational work is proportional to the number of grid cells. For this reason it would be anticipated that performance would be similar to Test Problem 1. Figure 13 shows ideal speedups for this case in which the parallel performance is compared to the same code running on a single CPU. The solver performance is somewhat exaggerated due to the use of a block Jacobi method for the single-node case. The coefficient setup speedups (including well calculations) give the best indication of actual parallel performance: 3.8, 7.6, and 13.9 on 4, 8, and 16 processors respectively. Figure 14 shows real speedups for this problem. That is, the comparison is made with the case running on a single processor using a single grid and the best "serial solver". This comparison shows a slight decrease in overall simulator speedup due to the overhead of the grid refinement: 3.5, 7.1, and 12.1 for 4, 8, and 16 CPUs. Performance was excellent in spite of a 10 percent degradation in the number of linear and non-linear iterations as the number of processors increased.

These results provide a good comparison with the published performance of other parallel simulators. The 16 processor SP2 case showed performance of about 5.2 microseconds per gridblock-timestep. This compares with a published performance of about 2.7 microseconds per gridblock-timestep on a 128 node Thinking Machines' CM-5³.

Fully-implicit results versus the best serial performance for Test Problem 2 are shown in Figure 15. Overall simulator performance is similar to that for the IMPES case: 3.1, 6.4, and 11.1. The degradation is primarily due to solver performance. The fully-implicit strip decomposition solver uses a pressure predictor followed by a fully-coupled solution. As timesteps become larger, the matrix for the pressure equation becomes less diagonally dominant and hence solver performance degrades. This was noted in the single CPU case, but becomes more pronounced when the less-robust parallel solver is used.

Test Problem 3 Results

Test Problem 3 provides an example of a more realistic case than problem 1. Figure 16 shows results for the IMPES cases. Since local grid refinement was used to scaleup the original problem, the only possible comparison for speedups was the serial model using local grid refinement. Elapsed times for the serial well calculations were included in the coefficient setup times. Eleven processors were used as one of the cases simply to show that an even number of nodes was not required and that the model could easily adapt to an odd number of nodes. Results are similar to the second test problem: 4.95, 8.25, 10.38, and 13.36 for 4, 8, 11, and 16 processors. These speedups were achieved in spite of a two-fold increase in the

number of linear iterations in moving from four to sixteen domains. Coefficient setup speedups without the well calculations were 4.4, 8.2, 10.48, and 14.7. All cases had the same number of outer iterations since a single Newton iteration was used per timestep with a relaxed-volume formulation. For the sixteen processor case, performance was about 8.7 microseconds per gridblock-timestep. This was somewhat slower primarily due to an increase in the average number of linear iterations from 3 to 15 compared to Test Problem 2.

The fully-implicit results for Test Problem 1 were excellent for four processors but slowed somewhat for the eight processor case as shown in Figure 17. The degradation in performance for the larger number of processors was primarily due to the pressure predictor step of the linear solver as discussed above. The number of outer iterations also increased as a result of the solver degradation. Nonetheless, simulator performance was still respectable with a speedup of 6.39 overall compared to the serial case for 8 processors. It is likely that the use of other decompositions and the composite grid solver will substantially improve these results.

Test Problem 4 Results - Static Load Balancing

The realistic compositional case represented by test problem 4 has been shown in previous studies⁸ to exhibit significant load imbalance for the PVT calculations. The original grid was subdivided in the y-direction into four domains with similar numbers of xz-planes: 8,7,7,7. This resulted in a 200% variation in elapsed times for the PVT calculations between processors 1-3, and 4. Overall elapsed times varied by a factor of at least two. (See Figure 18 - TOTAL1 and PVT1.) With the linear decomposition used for this problem static load balancing can be easily accomplished. Three load balancing steps were taken in which the number of planes respectively for CPUs 1-4 were changed to 7,7,7,8 (TOTAL2 and PVT2); to 7,7,5,10 (TOTAL3 and PVT3); and finally to 7,7,6,9 (TOTAL4 and PVT4). This balancing resulted in a reduction in the elapsed times for the PVT calculations from 91 to 71 seconds or about 22%. Overall elapsed times decreased by only about 18% since other calculations in the simulation were now unequal among processors. Speedups were improved from 2.91 for the original case to 3.44 for the final balanced simulation.

This example demonstrates that linear decomposition can lead to a straight-forward approach to static load balancing. For other cases it is likely that dynamic techniques may be required.

Test Problem 5 Results - Fully-Implicit Compositional Simulation

The final test problem was used to demonstrate the robustness of the parallel simulator. First the problem size was increased by a factor of eighteen from test problem 4. Simulations showed that similar load balancing could be achieved with the larger model as depicted in Figure 19.

Variation in elapsed times for the PVT calculations was reduced from 50% to less than 10% for the balanced case.

To check the robustness of the fully-implicit formulation of the parallel model and linear equation solver, the 66x87x10, six-component model was run over the same time period as the IMPES run above. As shown in Figure 20, timesteps, outer iterations, and inner iterations were all reduced by a factor of five for this problem, indicating a significant increase in model stability.

Performance on Workstation Clusters

The IBM SP2 used for most of the results provided a unique feature to test performance of the model on a cluster of workstations. Simply by changing a UNIX environmental variable, the communications of the SP2 are transformed from the highspeed switch to ethernet speed. This results in an increase in latency to about 250 microseconds and a decrease in bandwidth to about 1 million bytes per second. These are comparable to an idealized cluster of workstations communicating over ethernet.

Figure 21 shows results for the large black-oil Test Problem 2. (SP2-IP refers to "internet protocol" or ethernet mode while SP2-US refers to "user space" or highspeed switch mode on the SP2.) For four processors, the results are excellent with speedups approaching ideal. As the number of processors are increased to eight and eleven, the performance degrades significantly probably due to saturation of the network and the significant latency with increased numbers of messages. Performance for these latter cases was actually worse than that for four processors.

Figure 22 show results for the large compositional Test Problem 5. For the overall simulation, elapsed times for the ethernet mode were nearly equal to the times for the highspeed switch indicating high parallel efficiency. These encouraging results will be further investigated on larger numbers of processors. Work station clusters offer the greatest promise for compositional simulations since message passing overhead will in general be small compared to computation times for this situation.

CONCLUSIONS

This work has presented a general-purpose parallel reservoir simulator which utilizes local grid refinement as a natural and flexible framework. The parallel model was developed with three forms of communication packages: PVM, MPI, and MPL. The message-passing scheme utilizing send-receive buffers for the simulator was described. Two parallel linear equation solvers were developed using strip and composite grid decompositions. Standalone results for the solvers showed excellent parallel speedups on an idealized basis and compared to an efficient serial solver for greater than four million nodes and up to seventy-one processors. Results for the overall simulator were good to excellent for a variety of problems ranging from small black-oil to large compositional cases for both IMPES and fully-implicit formulations. Finally, prototype simulations

for clusters of workstations produced excellent results for a limited number of nodes.

FUTURE WORK

The results of this study form the basis for future work on the parallel model. Clearly, a continued effort to improve the parallel linear equation solvers is required, especially for the fully-implicit formulations. The ultimate goal of the project is to directly simulate geological grids. This can only be achieved with memory scalability. Versions of both the initialization and simulation programs with reduced memory have been developed and will shortly be completely debugged. Initial results show at least an order of magnitude reduction in memory requirements for a sixteen processor case. The serial bottlenecks created by the well calculations are also being addressed. Theoretically, well-block information need only be passed for wells which penetrate grids on more than one processor. This technique is currently utilized in the solver and should substantially reduce communication overhead. For group and field constraints minimal length of messages is required, but there may be many messages so that latency becomes an issue. Serial overhead associated with facilities calculations may also have to be parallelized if these form a major portion of the CPU time for a simulation. Finally, static and dynamic load balancing for not only the phase behavior, but also well and local grid refinement calculations remains an issue to be addressed.

ACKNOWLEDGMENT

This work was supported in part by British Petroleum, Cray Research, IBM Corporation, Mobil, Oryx, Silicon Graphics, and the Texas Advanced Technology Program (Grant number 003652-247).

REFERENCES

1. van Daalen, D. T., Hoogerbrugge, P. J., Meijerink, J. A., and Zeestraten, P. J. A., "The Parallelization of BOSIM, Shell's Black/Volatile Oil Reservoir Simulator", *Proceedings of the First IMA/SPE European Conference on the Mathematics of Oil Recovery*, Oxford University Press, 1990.
2. Wheeler, J. A., and Smith, R. A., "Reservoir Simulation on a Hypercube", SPE 19804 presented at the 64th SPE Annual Conference and Exhibition, San Antonio, October, 1989.
3. Killough, J. E., and Bhogeswara Rao, "Simulation of Compositional Reservoir Phenomena on a Distributed Memory Parallel Computer", *Journal of Petroleum Technology*, November, 1991.
4. Rutledge, J.M., Jones, D. R., Chen, W. H., Chung, E. Y., "The Use of a Massively Parallel SIMD Computer for Reservoir Simulation", SPE 21213 presented at the eleventh SPE Symposium on Reservoir Simulation, Anaheim, 1991.
5. Gautam S. Shiralkar, R. Volz, R. Stephenson, M. Valle, and K. Hird, "Parallel Computing Alters Approaches, Raises Integration Challenges in Reservoir Modeling", *Oil and Gas Journal*, May 20, 1996, 48-56.
6. Nolen, J. S., and Stanat, P. L., "Reservoir Simulation on Vector Processing Computers", SPE 9644 presented at the SPE Middle East Oil Technical Conference, Manama, Bahrain, March, 1981.
7. Wallis, J. R., Kendall, R. P., and Little, T. E., "Constrained Residual Acceleration of Conjugate Residual Methods", SPE 13536 presented at the Eight SPE Reservoir Simulation Symposium, Dallas, Texas, February 10-13, 1985.
8. Wallis, J. R., Foster, J. A., and Kendall, R. P., "A New Parallel Iterative Linear Solution Method for Large-Scale Reservoir Simulation", SPE 21209 presented at the Eleventh SPE Symposium on Reservoir Simulation, Anaheim, California, February 17-20, 1991.
9. Wallis, J. R., and Nolen, J. S., "Efficient Linear Solution of Locally Refined Grids Using Algebraic Multilevel Approximate Factorizations", SPE 25239 presented at the Twelfth SPE Symposium on Reservoir Simulation, New Orleans, Louisiana, February 28-March 3, 1993.
10. Killough, J. E., and Wheeler, M. F., "Parallel Iterative Linear Equation Solvers: An Investigation of Domain Decomposition Algorithms for Reservoir Simulation", SPE 16021 presented at the 9th SPE Symposium on Reservoir Simulation, San Antonio, Texas, Feb. 1-4, 1987.

FIGURE 1: Communication Scheme

FIGURE 2: LGR Grid Decomposition

FIGURE 3: STRIP DECOMPOSITION

GRID LEVELS FOR COMPOSITE GRID DECOMPOSITION

FIGURE 4: LGR MULTILEVEL DECOMP.

SP2 - PARALLEL STRIP DECOMPOSITION
SOLVER 95x95x35 IMPES PROBLEM

FIGURE 5: Strip Decom. Results

SP2 - PARALLEL COMPOSITE GRID SOLVER
95x95x35 IMPES PROBLEM

FIGURE 6: Composite Grid Results

FIGURE 7: Grid and Structure for Test Problems 1 and 2

FIGURE 8: Grid and Structure for Test Problem 3

Figure 9: Grid and Structure for Test Problems 4 and 5

T3D - SPEEDUPS FOR 95X27X3 IMPES RUN

FIGURE 10: T3D RESULTS

SGI - SPEEDUPS FOR 95X27X3 IMPES RUN

FIGURE 11: SGI RESULTS

SP2 - SPEEDUPS FOR 95X27X3 IMPES RUN

FIGURE 12: SP2 RESULTS

SP2 - ACHIEVED SPEEDUPS FOR 95x95x10 IMPES RUN

FIGURE 13: SP2 IDEAL SPEEDUPS

SP2 - SPEEDUPS FOR 95x95x10 IMPES RUN

FIGURE 14: SP2 REAL SPEEDUPS

SP2 - SPEEDUPS FOR 95x95x10 IMPLICIT RUN

FIGURE 15: SP2 IMPLICIT CASE

SP2 - SPEEDUPS FOR 128x50x24 IMPES RUN

FIGURE 16: LARGE IMPES CASE

SP2 - SPEEDUPS FOR 128x50x24 IMPLICIT RUN

FIGURE 17: LARGE IMPLICIT CASE

SP2 - COMPOSITIONAL STATIC LOAD BALANCING

FIGURE 18: STATIC LOADBALANCING

SP2 - LARGE COMPOSITIONAL EXAMPLE
STATIC LOAD BALANCING

FIGURE 19: STATIC LOADBALANCING

SP2 - COMPARISON OF IMPLICIT AND IMPES
LARGE COMPOSITIONAL SIMULATIONS

FIGURE 20: IMPLICIT COMPOSITIONAL

SP2 - SPEEDUPS FOR 128x50x24 IMPES RUN
WORKSTATION CLUSTER VERSUS SP2

FIGURE 21: BLACK-OIL CLUSTER

SP2 - LARGE COMPOSITIONAL
WORKSTATION CLUSTER VERSUS SP2

FIGURE 22: COMPOSITIONAL CLUSTER