Parallel Chemical Flood Simulation: An Implementation of UTCHEM on Distributed Memory Processors

M. Ramé L. Pavarino A. Greenberg
K. Jordan M. Wheeler

May 1993

TR93-23

TR93-23

Parallel Chemical Flood Simulation: An Implementation of UTCHEM on Distributed Memory Processors

M. Ramé†, L. Pavarino†, A. Greenberg*, K. Jordan* and M. Wheeler†

†Department of Computational and Applied Mathematics,
Rice University, P.O. Box 1852, Houston, TX 77251
*Thinking Machines Corporation, 245 First St.
Cambridge, Massachusetts 02142-1264

July 18, 1993

ABSTRACT

This work describes experiences porting the UTCHEM chemical flood simulator from a serial environment to the nodal environment of distributed memory massively parallel computers. Two conversion strategies have been explored. The first approach required the least amount of effort. The serial version of the code was modified to execute independently on each processor. This version permits many different problems to be evaluated simultaneously but does not confer any computational advantage. In the second version, the program was altered, using domain decomposition, to distribute the data, and message passing communication, to couple the node computations. This allowed for execution of a single problem across all of the computing nodes. This investigation shows that the message passing version of the code speeds up well, as more computational nodes are employed, for problem sizes with a small surface to volume ratio. A more efficient linear solver is implemented in this work, which promises both good parallel efficiency and robustness for large-scale simulation problems.

1.- INTRODUCTION

Problems of current importance in oil recovery and underground contaminant clean-up studies involve transport in heterogeneous porous media. The numerical modeling of these phenomena requires large scale simulation that may only be accomplished, at a reasonable cost, on distributed memory parallel computers.

Improved oil recovery techniques target oil production from fields depleted by secondary recovery to nearly the residual oil saturation, leaving behind up to 70% of the total oil in place. The rather high cost of mobilizing the oil in place justifies its fine tuning for an acceptable profit. Consequently, a renewed interest has developed in improved-accuracy numerical schemes for reservoir simulation, i.e., fine-grid and locally-refined-grid methods, as well as more physically sound algorithms that minimize adverse effects of numerical dispersion.

Data acquisition of porosity and permeability from the underground is extremely costly and, as a result, only a very coarse reservoir description is possible by measurement alone.

However, new techniques of geostastistics permit the generation of stochastic set of material and transport porous media properties to an arbitrary degree of refinement. The statistical parameters of the generated set are *conditioned* to agree with those of the originally measured set (refs). Clearly, the stochastic data set (or geostatistical realization) is not unique since one is only trying to match a finite number of statistical moments of the original sample. This feature is used in the field of conditional simulation, where a statistically significant number of simulations can be run, using different realizations. If one is trying to predict future recovery patterns for a field, these multiple simulations provide upper and lower bounds for the macroscopic performance parameters of the reservoir, such as injection/production pressure and flow rate requirements. If the simulation is used as a history-matching tool, then one can select, from the multiple runs, the realization of physical properties for which the production history for the given field best matches the actual recovery data. Once this is done, one can use the simulation with some confidence to predict future field performance.

Technologies for underground contaminant clean-up are rapidly developing. Large plumes have been identified, for which the remediation process may require several decades, at a significant cost. Reduction of treatment time even by a few percent can represent huge savings and a shorter exposure to a contaminant threat. Accurate large-scale simulations, that can capture processes over a range of length scales, can greatly aid optimization of clean-up strategies. In this respect, simulation can also help the development and testing of new clean-up techniques.

As a consequence of the above discussion, a fertile ground for attempting to run large-scale simulations is rapidly growing. However, improving the accuracy of the numerical schemes leads inevitably to an overall refinement of the discretization grids. Therefore, the computing times, as well as computing memory requirements to run such simulations are too large to be handled by conventional computers. High performance vector computers of Cray-type can sometimes handle these problems, though at a prohibitive cost. Over the last ten years, the rapid development of distributed memory parallel computers has appeared to offered the required high performance at a moderate cost for large-scale reservoir simulation[1].

Finally, production policy constraints may sometimes demand that one run rather coarse simulations to minimize the turn-around times. It has been shown[2] that coarse discretization schemes can approximate the solution to the original flow and transport problem only if additional dispersion terms are added to the governing equations used for the coarse discretization. This phenomenon can be explained by remembering that the equations that describe the momentum, heat and mass transport in porous media are *volume-averaged* versions of the corresponding continuum-mechanics governing equations[3]. This level of averaging introduces dispersive terms into the porous media model equations. The numerical discretization can be viewed as a second level of averaging, in that one has to assign physical properties to each node in the discrete space, which must contain some information (weighted average) about these properties in some neighborhood of the given grid node. The way this weighted average should be carried out and the appropriate way to correlate the new dispersion coefficients to the parameters of the discrete-approximation space and of the physics of the problem is a matter current research. Therefore, fine grid simulations are the necessary tool to test the different coefficient correlation and weighting hypotheses.

This work is concerned with flow situations where, because of the prevailing hydrodynamic regimes, the convection terms dominate in the model equations. Concentration

or saturation shocks form and interact as the flow progresses. The miscible displacement of a fluid by another, is modeled by a species balance for the solute, given by

$$\phi \frac{\partial c}{\partial t} + \underline{\nabla}\cdot(c\underline{u}) = \underline{\nabla}\cdot\left[\underline{\underline{D}}\cdot\underline{\nabla}c\right] \qquad (1)$$

and the hydrodynamic pressure equation is given by

$$\underline{\nabla}\cdot\left[\frac{\underline{\underline{K}}}{\mu(c)}\cdot\underline{\nabla}\ p\right] = 0, \qquad (2)$$

where $\underline{u}$ is the Darcy velocity, c is the solute concentration, μ is the mixture viscosity, $\underline{\underline{D}}$ is the dispersivity tensor, $\underline{\underline{K}}$ is the permeability tensor and ϕ is the porosity. The pressure equation represents an incompressible flow for which the Darcy velocity field is divergence free, i.e., $\underline{\nabla}\cdot\underline{u} = 0$. These equations are coupled through the dependence of the mixture viscosity on the concentration and the dependence of the superficial velocity on the pressure given by Darcy's law.

The solution of the elliptic problem (equation (2)) rapidly dominates the total CPU time as the grid is refined. Thus, a section of this work addresses the development of robust linear solvers to tackle the systems of equations that arise from the discretization of equation (2) over a three-dimensional space. The condition number κ of the resulting linear system is poor owing to two competing reasons: Fourier analysis of the discrete equations shows that κ grows as $1/h^2$, where h is the size of the spatial discretization; also, the entries of the permeability tensor are strongly dependent on position, sometimes exhibiting permeability contrasts of 3 to 5 orders of magnitude over short length scales.

This paper is organized in the following form. Section 2 gives an overview of the two families of parallel computing hardware, i.e., distributed and shared memory, and a description of the particular systems tested in this work, i.e., the Intel iPSC/860 and Touchstone Delta, and the Thinking Machines Connection Machine 5. Section 3 describes UTCHEM, the chemical flood simulator used for this investigation, the physical phenomena modeled in it and the strategy used in converting the code for distributed memory parallel computers. In section 4, a summary of parallel run performances is given for some sample cases, as well as a discussion on observed problems and possible solutions to them. Section 5 presents a domain-decomposition-type linear solver developed as a result of the experience from the parallel performance runs, i.e., the poor conditioning of the existing solver in the original simulator for large problems. Section 6 shows parallel numerical experiments for this solver. Finally, section 7 gives some conclusions and expected future developments.

2.- PARALLEL COMPUTING

Parallel (or distributed) computing refers to the partition of a large problem into smaller pieces so that a number of processors can concurrently effect the computations. The concept behind this approach is that concurrent computing can substantially reduce the computing time as compared to the time necessary to run the same application on a single processor of similar

capabilities to one of the members of the concurrent processor set.

Two families of parallel computers exist presently. Those in which data are accessible to all processors directly are called shared memory parallel processors. These machines typically have a few (4 to 16) rather powerful processors on the network, which are directly connected to each one of several memory banks. Examples of this kind are the CRAY Y-M/P and the IBM 3090VF supercomputers. The cost of building these systems is understandably high, since they require a large number of interconnections between the various processors and all of the memory banks.

As an alternative, the idea of distributed memory parallel processors started about ten years ago. These computers have an array of processors (typically less powerful than those in the above family), each of them with its own computing memory. One can imagine these systems as a cluster of computers linked by a local communication network. Data are therefore distributed across the processors and a particular section of the data is directly accessible to the processor on whose memory that section of data sits. If required by a given step of the computation, all other processors in the array have to issue a request for this data to be sent to them from the owner processor. This idea will be further explained in the next section, in connection with the actual parallelization of the simulator. For now, though, one can think of the parallel simulation as spending a fraction t_{CPU} of the total elapsed time on actual computations and a fraction t_{mp} on message-passing operations to share the necessary data. The computing t_{CPU} time corresponds to the total execution time of the serial algorithm. In parallel the total execution time is given by

$$t_{par} = \frac{t_{CPU}}{N_{CPU}} + t_{mp},$$

where N_{CPU} is the number of processors in the array. In the above equation, it is assumed that the entire algorithm can be executed in parallel. The speed-up for the algorithm is therefore given by $\frac{t_{CPU}}{t_{par}}$, i.e.,

$$\text{Speed-up} = \frac{N_{CPU}}{1 + \frac{t_{mp}\, N_{CPU}}{t_{CPU}}}.$$

The above equation is known as Amdhal's law and shows that the theoretical acceleration by parallel computing is limited by the message-passing overhead time, t_{mp}. When this time approaches zero, the speed-up approaches N_{CPU}.

The relative message-passing overhead of an algorithm is given by the ratio $\frac{t_{mp}}{t_{CPU}}$, which can be estimated from the parallel speed-ups and the number of processors. The maximum attainable speed-up tends to $\frac{t_{CPU}}{t_{mp}}$ for large N_{CPU}, irrespective of the actual value of N_{CPU}, which shows that it is crucial to minimize the message-passing overhead of a parallel algorithm if one is to obtain significant performance.

The UTCHEM simulator was ported to three distributed memory parallel processors, i.e., the Intel iPSC/860 (hypercube) and Touchstone Delta and the Thinking Machines Connection Machine 5.

Hypercube iPSC/860 Architecture

An iPSC/860 hypercube is a parallel processor configurable up to 128 computing nodes. The compute processor is RISC-based Intel i860 chip with a peak computing rate of 20 MFlops in double precision and with 8 to 16 MBytes of memory. Compiler limitations only allow the user to see about 2-4 Mflops for common applications. A compute node can have up to 6 nearest neighbor direct connections . The system's name comes from the fact that these are *cubes* of dimension greater than 3. The hypercube connects with the outside world through a front end of PC-386 type. There are parallel storage devices on the hypercube, as well as serial storage devices on the front end. Node level languages are Fortran 77 and C. These languages include NX message-passing library extensions.

Touchstone Delta Architecture

The Touchstone Delta computer is a prototype (one of a kind) massively parallel processor that can be configured up to 516 computing nodes. The compute nodes are improved i860 chips, with a peak rating of 60 MFlops in double precision. Compiler peculiarities limit the expected performance to around 5-7 Mflops. Each compute node has 16 Mbytes of memory, and is connected to a message routing chip (MRC). The MRC's are components of a two-dimensional mesh-type communication network, so that each CPU on the Delta has only four nearest neighbors. The system supports node level code in Fortran 77 and C, with the message-passing extensions, using the NX library as for the hypercube. The communication software makes no attempt of optimizing the routing and messages proceed always in one direction of the mesh first and, then, in the perpendicular direction.

Connection Machine Architecture

A Connection Machine CM5 is a massively parallel, distributed memory computer. It may be configured with tens to thousands of processing nodes. Each node consists of a SPARC scalar chip set, four vector pipelines, 32 MBytes of memory, and a network interface. Nodes communicate with each other and with a variety of I/O devices via a point to point data routing network (DR) and a multifunction broadcast, combining, reduction network (CN). The CM5 may be programmed in the message passing style using the CMMD message passing library. This supports node level code written in Fortran 77 or C. A CM5 may also be programmed in the data parallel style using either CM Fortran, a Fortran 90 variant, or C*, a parallel extension of the C language. The data parallel languages may be used to construct programs that are global in extent (they manage the resources of all processors in the system) or they may be used to create node level programs which communicate using the message passing library. Currently, the vector pipelines may be accessed either by writing data parallel code or by writing assembly level routines embedded in message passing programs. Node level programs written in Fortran 77 will not be able to take advantage of the vector performance of the CM5 without the addition of such assembly level routines. A well structured Fortran 77 program may be converted to CM Fortran, however by using the CMax conversion tool. This is provided as a standard component of the CM5 software suite.

3.- THE CHEMICAL FLOOD SIMULATOR: UTCHEM

The serial simulator

UTCHEM is a Fortran 77 chemical flood simulation program developed by the Petroleum Engineering Department at the University of Texas at Austin[4]. The simulator computes the transport of several petroleum related chemical species in a three-dimensional

multiphase flow through permeable media. Typical flow constituents are: surfactants, polymers, electrolytes, water and oil. These interact in brine oil, and microemulsion phases which are subject to gravity, viscosity, capillary and dispersive forces. Various physical phenomena are modeled by the program. These include dispersion, adsorption, interfacial tension, relative permeability, capillary trapping of residual phase constituents, phase behavior, viscosity, capillary pressure, inaccessible pore volume, and permeability reduction by the polymer[5].

Mathematically, the simulation is comprised of n differential equations; one for pressure and (n -1) mass balance equations (where n is the number of components in the system, which can be as high as 11). Each equation is discretely represented using a second order finite difference approximation for the spatial derivatives, and a forward difference approximation for the time derivatives. See reference 3 for a detailed description of the equations and the discrete approximations. Boundary conditions for the model specify a 'no flow' condition imposed by the nonpermeable physical boundary of the reservoir, inflow and outflow conditions provided by a distribution of injection and production wells, respectively.

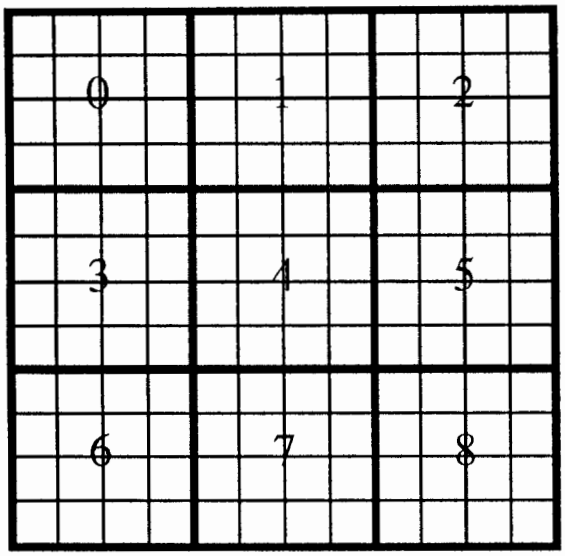
The Cartesian numerical discretization is of mixed-finite-element type (block centered finite differences)[4]. Solution of the system is performed in the IMPES style. First the pressure equation is solved implicitly in terms of saturation dependent terms. The system of linear equations arising from the discretization of the pressure equation is solved by the Jacobi-preconditioned conjugate gradient method. Then the mass conservation equations are solved explicitly for the total concentrations.

The code is optimized for vector computers so that arrays that refer to the three-dimensional space are declared as linear arrays. By this scheme, one can operate on 'long vectors' over the entire reservoir in a single loop construct. The special treatment of boundary grid blocks is achieved by performing additional specific computations on these grid blocks. An earlier version of UTCHEM performed at over 1 Gflop on a single processor of a CRAY Y-M/P (1 Gflop = 1 billion of floating-point operations per second).

The parallel simulator

Systems with as flexible a computing model as those described above offer several different strategies for porting the UTCHEM code. We have identified two options that do not require completely rewriting the code. In the first, an independent program is executed on each node. In the second, a problem is distributed across all the nodes of the machine in a message passing model.

Our first approach required modest modification to the serial version of the code to allow it to execute independently on each node of the CM5. In this instance, the code required only a few small changes to ensure that I/O was handled properly within the parallel message passing environment. No message passing calls or data mapping changes were introduced. When executed, this code model runs a copy of the program with each node computing a different problem. This port required only a few hours to complete. We note that while the port was quickly done, there was no computational speed up introduced. The ability to run many simulations in parallel can find use in conditional simulation, where many simulations can be run concurrently with different geostatistical realizations of the properties as input. This use of the parallel port has not been pursued at this point.



(a)

1 40	2 41	3 42	4 43	5 44	6 45
7 52	8 53	9 54	10 55	11 56	12 57
13 64	14 65	15 66	16 67	17 68	18 69
19 76	20 77	21 78	22 79	23 80	24 81
25 88	26 89	27 90	28 91	29 92	30 93
31 100	32 101	33 102	34 103	35 104	36 105

(b)

Figure 1: Local and global numbering schemes for the parallel code. (a) Mapping of a two-dimensional problem onto an array of nine processors (numbered 0 through 8). Thick lines indicate interprocessor boundaries. (b) Local and global grid block numbers appear on upper right corner and lower left corner of each grid block, respectively, for the process loaded on processor 4.

Our second port, using message passing, was designed to use all of the computing nodes in the simulation of a single problem by the single-program-multiple-data (SPMD) programming model. Upon execution, this version of the code distributes the spatial grid blocks across all of the processors. Each processor will generally receive many blocks. Array declarations were changed to allocate only the required memory to work on the subdomain of the reservoir contained in each processor. However, since some arrays are involved in a stencil calculation (arising from the discretization of the model equations) all of the arrays were declared to allocate enough memory for a subdomain plus a three-dimensional padding envelope around the subdomain so that data from neighboring processors can easily be brought into a given processor. By this strategy, the long-vector style of the original serial code can be easily preserved and, therefore, the parallel code can take advantage of machines with a node-level vectorizing Fortran 77 compiler.

The grid block distribution scheme assigns consecutively numbered blocks (for some numbering of the blocks in the system) to the nodes of the machine. This results in reasonable locality of computation (nearest neighbors are frequently located on the same computing node), and reasonable load balance is achieved (blocks are assigned so that nodes receive either $\frac{N_{BL\{x\ y\ z\}}}{N_{CPU\{x\ y\ z\}}}$ or $\frac{N_{BL\{x\ y\ z\}}}{N_{CPU\{x\ y\ z\}}} + 1$ grid blocks, where $N_{BL\{x\ y\ z\}}$ is the global number of grid blocks in the x-, y- or z-directions, respectively, and $N_{CPU\{x\ y\ z\}}$ is the number of processors into which the problem is decomposed in the x-, y- or z-direction, respectively). Figure 1a shows a two-dimensional example of the grid block distribution strategy for nine processors. Figure 1b shows the size of the array declared on processor 4 (processors are numbered sequentially from 0 through 8), where both global and local numbering schemes can be easily

identified. Computations on the blocks are then performed in parallel, where each processor performs the same computations on a subset of the global problem (SPMD model) in approximate synchronization. Data between neighboring processors are communicated at specified synchronization points by explicit message passing calls.

In this port, the I/O was handled serially through a designated node in the processor array. This amounts to all processors sending their partial output set to the collecting processor and, subsequently, the collecting processor writing to disk. In principle this could be handled by declaring one global array (big enough to hold one entry per grid block of the global problem). However, because of the way the I/O was handled in the original serial code and since the aim of this work was to not change any features of the simulator from the users standpoint (e.g., the structure of the output files), a global array big enough to hold 25 times the global number of grid blocks had to be declared. Since the memory per processor of distributed memory machines is relatively small and the SPMD programming model is used, i.e., each processor loads the same code on its memory, this imposes severe limitations on the maximum size of the problem that could be run. This poor I/O handling can be easily improved and work is being done to minimize or hopefully eliminate this adverse effect.

The message passing was first implemented for the Intel systems using the NX library environment. Ten routines were added to the original 30 in the process of parallelizing the code. The communication is done synchronously in this implementation, meaning that no computation is being performed while the processors exchange data. General code modification were also necessary since not all grid blocks on the boundary of a subproblem is a boundary grid block for the global problem.

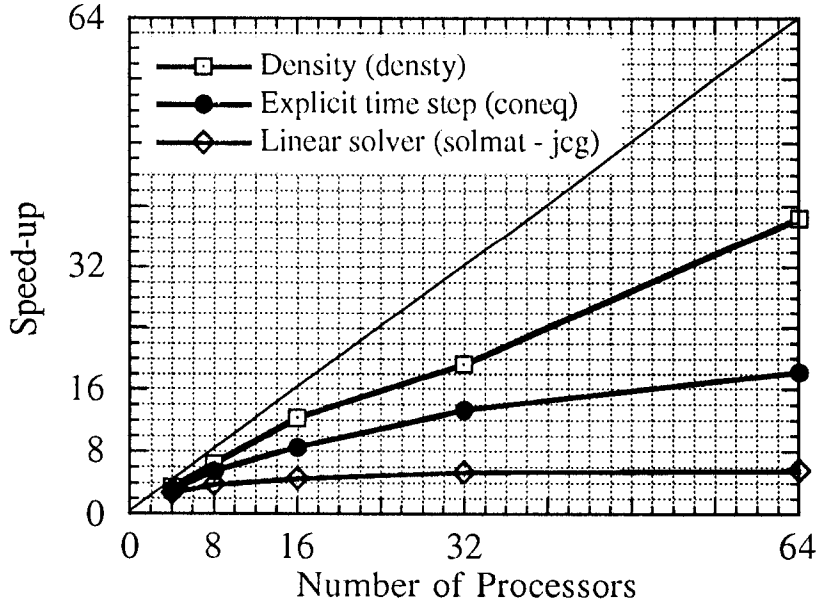


Figure 2: Sample parallel speed-ups for UTCHEM on the Touchstone Delta system. Plotted tasks are: 1. density evaluation at each grid block, 2. the explicit time stepping of the concentration equations and 3. the assembly of the pressure matrix coefficient and right hand side vector and the system's solution by JCG.

Since CMMD, the CM5 message passing environment, is a superset of the functions found in the Intel NX environment, converting the code to run on the CM5 required only that we provide emulation functions for the Intel primitives and small modifications to account for differences in I/O behavior. This version of UTCHEM used only 18 communications related functions. Creating emulations for these NX calls required little effort.

4.- PARALLEL CODE PERFORMANCE

The performance of the code ports is evaluated by running the serial code and each of the code ports against the same test input. The input describes a small test case that contains a reservoir of 33x33x2 grid blocks, where each cell is 22.7x22.7x0.5 feet. Our test case executes a two well simulation comprised of 11 components and three tracers. There is one injection well and one production well. They are located at opposing x-y vertices. Each run represents a 40 day simulation with 20 times steps per day.

A serial version of the code, which was run on a Sparcstation 10, took 24.75 minutes to execute. The first parallel port, where each node runs a copy of the program, took 57.7 minutes to execute on the CM5. This difference in performance can be accounted for by noting that the Sparcstation 10 uses a superscalar version of the SPARC chip (launches more instructions per cycle), has memory subsystem which is faster than the system on the CM5 node (each load and store is faster), executes a different cache memory policy (more efficient use of cache and faster cache response on loads), and runs at a higher clock frequency. The various processor differences can easily account for the factor of two difference in speed between a serial workstation and a CM5 node.

Performance of the message-passing port was evaluated on configurations of 1, 2, 4, 8, 16, 32 and 64 processors for both Intel systems and of 1, 2, 4, 8, 16, 32, 64 and 128 processors for the CM5. Figures 2 and 3 show sample speed-up results for some of the major tasks, on both the Delta and CM5 systems. The parallel performances are similar on both

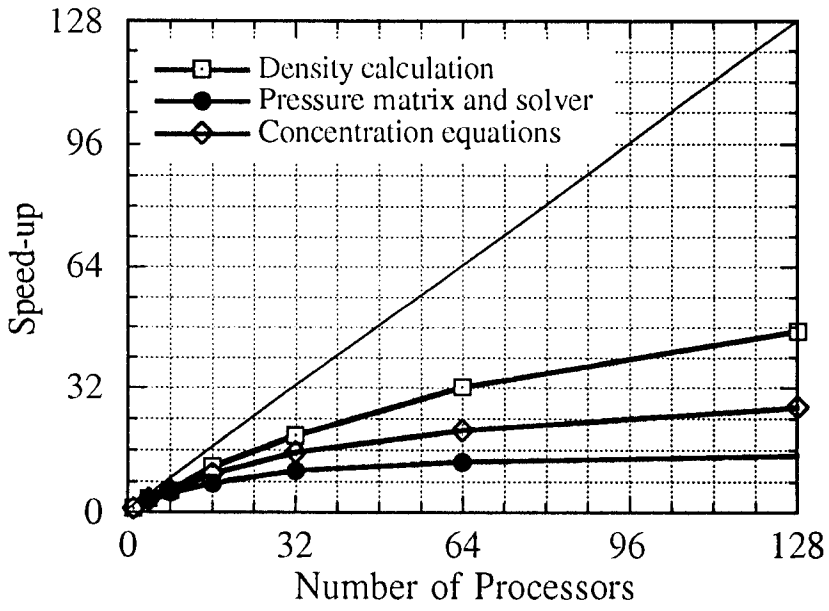


Figure 3: Sample parallel speed-ups for UTCHEM on the CM5 system. Plotted tasks are same as on figure 2.

systems and they are not great on either on beyond about 32 processors. Similar behavior was found on the iPSC/860 (not shown here). In particular, even the density calculation routine, which does not involve any movement of data, shows an overhead of 100% at 64 processors. The reason for this lies in the long vector style of the code: every do loop construct operates not only in the interior of the local problems but also on the surrounding padded region. Also, as the number of nodes increases, the ratio of grid cells to boundary cells decreases until the communication cost begins to dominate the computation cost. This is evidenced by the flattening of the speed up curve in Figures 2 and 3. Increasing the problem size should decrease this adverse effect and restricting all computations to the interior of the local arrays should eliminate this source of overhead completely.

The explicit time stepping of the mass conservation equations, involves some message-passing from the stencil computations and shows a more severe degradation that the density routine, because of the combined effect of communication and padding-computation overhead. This latter can be eliminated as explained above and the former could be minimized by implementing asynchronous communications, i.e., message-passing and computations are overlapped in time.

5.- DOMAIN DECOMPOSITION LINEAR SOLVER

As noted in the previous section, the parallel performance of the existing solver in the original UTCHEM is less than acceptable. Moreover, once the code is well tuned to handle larger simulations, the anticipated performance of such solver will even be worse. The range of the pressure matrix coefficients, for realistic problems, runs from 3 to 6 orders of magnitude. In such a case, a Jacobi conjugate gradient solver like that currently used in the code will either take many iterations to converge or will not converge at all.

This section introduces a domain decomposition solver based on additive Schwarz preconditioning[7,8] of the conjugate gradient algorithm. The additive Schwarz preconditioner is formed by projecting the global equations onto subspaces represented by the local linear systems to be solved on overlapping subdomains. A subdomain is the collection of grid cells resident on a given computing node. Since the parallel port of UTCHEM already declared extended arrays, this appears to be a natural choice of preconditioner for this problem.

The necessary steps to form the preconditioner are as follows. The basic conjugate gradient method iteratively solves the linear system $\mathbf{Ap} = \mathbf{b}$ where $\mathbf{A}$ is the pressure matrix, $\mathbf{p}$ is the unknown (pressure) vector, and $\mathbf{b}$ is the right hand side boundary value vector. At each iteration n, the preconditioned residual vector is computed from the solution to $\mathbf{Mz}^n = \mathbf{r}^n$, where $\mathbf{r}^n$ is the current residual, i.e., $\mathbf{r}^n = \mathbf{b} - \mathbf{Ap}^n$, $\mathbf{z}^n$ is the preconditioned residual and $\mathbf{M}$ is the preconditioning matrix. In the framework of additive Schwarz methods, the matrix $\mathbf{M}$ is never assembled since one can compute the action of $\mathbf{M}^{-1}$ on $\mathbf{r}^n$ by the following steps:

1. Project the residual and the coefficient matrix of the global (entire domain) problem onto the extended subdomain (includes overlapping boundary grid blocks) on each subdomain. Solve $\mathbf{A}_i\mathbf{z}^n_i = \mathbf{r}^n_i$ on each extended subdomain (i.e., for i ranging from 1 to the number of subdomains or processors) using homogeneous Dirichlet boundary conditions. Where the extended subdomain boundary matches the global domain boundary, the original boundary conditions are used.

2. After all subdomains solutions have been computed, add the partial results to assemble the global preconditioned residual.

3. If there are large numbers of subdomains, use an additional coarse grid solver to allow for information on the global residual $\mathbf{z}^n$ to spread globally across subdomains. Here, the coarse problem is a finite-element discretization of the global domain where each subdomain is a single element. Boundary conditions for the coarse problem are the same as those for the original problem, i.e., no-flow Neumann conditions. The coarse solution is then added to the subdomain solutions by interpolation of the result.

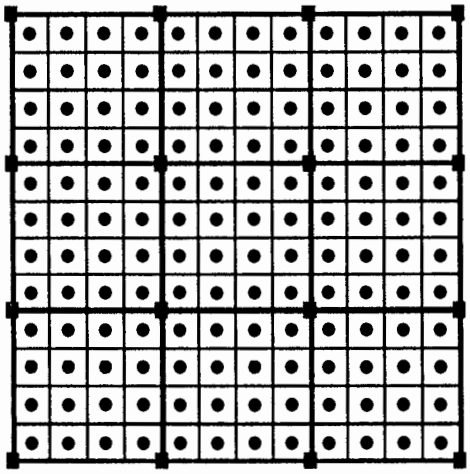


Figure 4: Location of the fine and coarse degrees of freedom for the additive Schwarz preconditioner. Squares denote coarse-grid unknowns and circles denote fine grid unknowns.

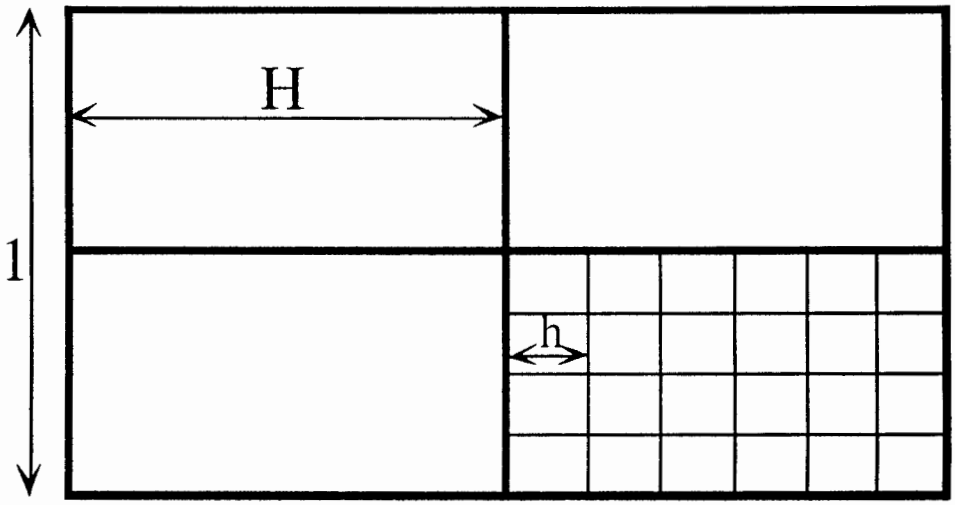


Figure 5: Length scales of interest in the convergence analysis of domain decomposition solvers. Global domain is O(1), subdomain sizes are O(H) and discretization elements (or grid blocks) are O(h).

The coarse grid problem is formulated based on a piecewise linear approximation to the discrete problem where each element in three dimensions has eight degrees of freedom, i.e., one at each of the vertices of the normalized cube. Figure 4 shows the location of the coarse and fine grid unknowns (squares and circles, respectively) for a two-dimensional problem with nine subdomains. Notice that the Galerkin type degrees of freedom of the coarse-grid problem are shifted by h/2 with respect to the nearest degrees of freedom of the mixed-finite-element fine discretization.

Consider a global domain of characteristic dimension O(1), where subdomains are O(H) and the underlying discretization is O(h) (see figure 5). It has been shown[8], for a parabolic or an elliptic problem with constant coefficients that the condition number of the additive-Schwarz-preconditioned system is bounded by

$$\kappa \leq C\left[\, 1 + \frac{H}{\delta} \,\right],$$

where κ is the condition number of the additive-Schwarz-preconditioned system, δ is the extent of overlap and C is a constant, independent of H, h and δ. This expression suggests that, for constant subdomain size, the condition number should reach a constant value. Numerical experiments were conducted solving elliptic pressure problems (equation 2) for subdomains of constant size (5x5x5 grid blocks), δ = h and increasing numbers of subdomains. The overlap for these experiments was a single layer of grid blocks, thus giving H/h = 7. Figure 6 shows that, when the coarse grid algorithm is turned on, the condition number appears to reach a constant level for large numbers of subdomains. The straight line on the same plot shows the growth of the condition number when the coarse space is eliminated. This linear growth with

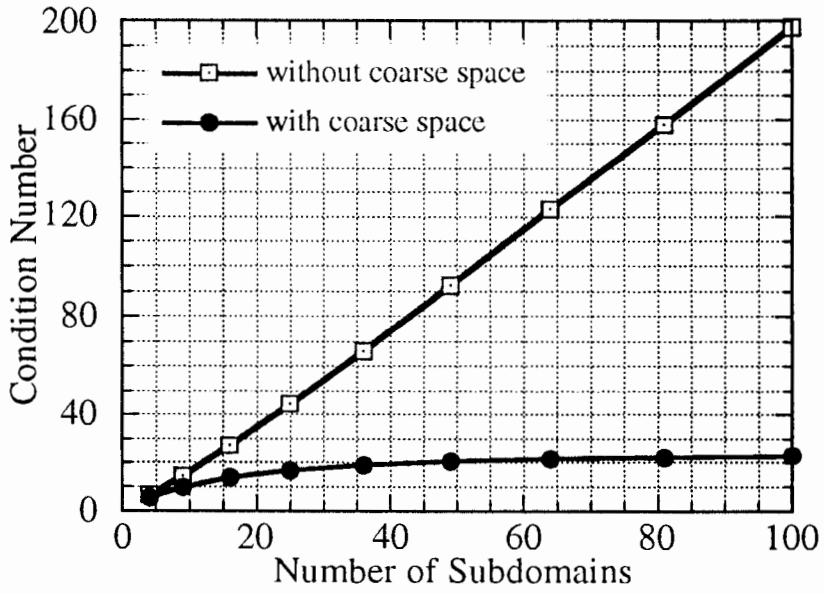


Figure 6: Asymptotic behavior of the condition number for the additive Schwarz preconditioned system, both with and without the action of a coarse space, for increasing numbers of subdomains. The subdomain problem size is 5x5x5 in all cases.

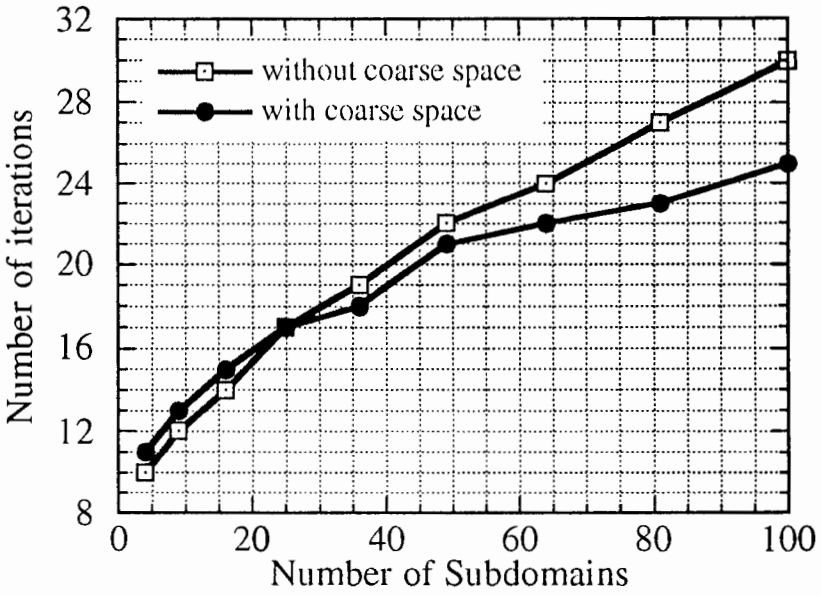


Figure 7: Required number of iterations for convergence of the cases in figure 6.

the number of subdomains has been proven[9] and is due to the fact that information must travel across the subdomain (processor) mesh which becomes linearly harder as the number of subdomains grows.

Figure 7 shows the equivalent plot for the number of iterations taken with and without a coarse-space preconditioner. The number of iterations of one and the other do not differ as much as suggested by the difference in condition numbers of the systems. This is probably due to the fact that the problem of constant coefficients is inherently simple. For parallel experiments with Galerkin problems, see reference 10.

6.- PARALLEL SOLVER PERFORMANCE

From the algorithm description in the previous section, one can see the following parallel features of the preconditioner construction. The projections necessary to set up the local subdomain problems and their solution are totally independent and can therefore be run in parallel with minimal overhead.

The construction of the coarse grid matrix is also a parallel operation to a large extent. The local stiffness matrix and local vector of right hand sides are set up independently by each processor and the local contributions are then accumulated at once to form the global stiffness matrix and the global vector of right hand sides. In the present work, all processors solve the entire coarse grid problem simultaneously (only serial step) and, subsequently, each processor adds its own contribution to the total preconditioner by interpolation within each subdomain.

The parallel solver can be directly hooked up to the parallel UTCHEM simulator. However, for flexibility purposes, the tests shown here run the solver detached from the simulator. The local problems as well as the (serial) coarse-grid problem were solved by an orthomin(K) method with an incomplete LU factorization (iLU(0), i.e., zero degree of

additional infill is generated by the factorization) as its preconditioner[11,12]. This method is of conjugate residual type, i.e., based on the Gram-Schmidt normalization. The number K of previous search directions (eigenvectors) of the orthomin iteration was fixed to five to decrease the required storage of this method. This number of search directions has been reported to maintain the robustness of the method[11].

Figure 8 shows the parallel timings for the problems used in figure 6. The runs shown were made on the Delta system. For an algorithm with no overhead, one should expect an execution time independent of the number of processors, since the subdomain problems have a constant size as the number of processors increases. However, the timings increase with the number of subdomains. This is due to the effect of the coarse problem being solved serially and also to the (relatively smaller) communication overhead of the outer conjugate gradient iteration. One could discriminate between these to sources of overhead by performing a more detailed timing study, which was not pursued at this point.

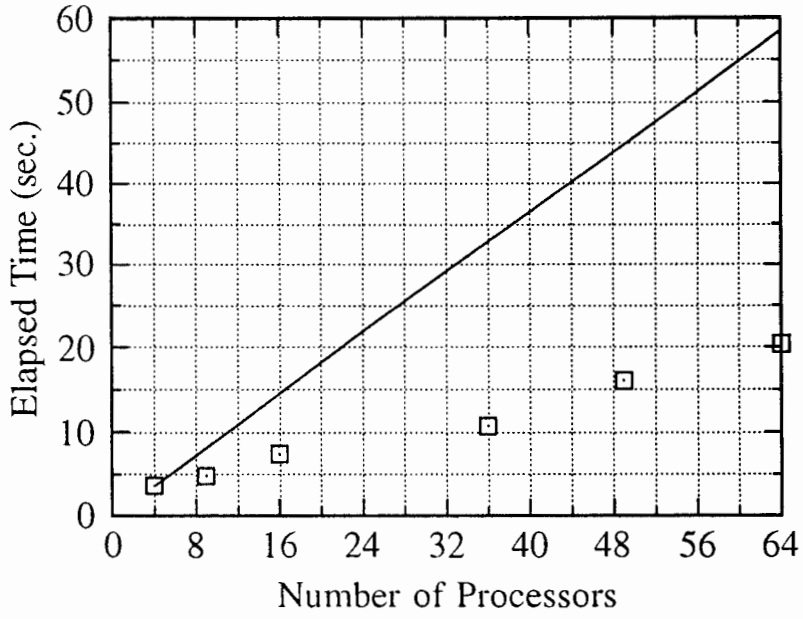


Figure 8: Timings for the parallel additive Schwarz method. Subdomain sizes are constant at 5x5x5 grid blocks. The global problem size is given by 125 multiplied by the number of subdomains.

More importantly, the plot shows the advantage of running in parallel. The straight line indicates the time the algorithm would take if it was proportional to the size of the global problem. Since all the algorithm components have an operation count which is proportional to the number of unknowns, the latter is the approximate timing plot one would get when running serially for the same range of problem sizes. The savings from running in parallel are about 60%. This is not outstanding but acceptable, nevertheless, considering that the subdomain sizes are rather small. Increasing the size of the subdomain problems, thus increasing the global problem size or refining the overall discretization, will show a gain with respect to serial computing time in the neighborhood of one order of magnitude. More detailed experiments with this domain decompostion solver will be made avialable in a separate technical report in the near future.

7.- CONCLUSIONS

Our experience with the UTCHEM code and the distributed memory parallel environments suggests that the conversion from a strictly serial implementation to one which uses the full resources of the machine is an achievable, although nontrivial, task. A degree of speedup can be attained by distributing a single problem across all of the nodes of the machine. For computationally intensive codes, however, we believe that performance gains will generally be realized by using the vector units (like those available on the CM5) to solve large subdomains on each node (thus proportionally reducing the communications burden). Use of the CM5 vector facilities virtually demands that the code be written in or translated to CM Fortran (or C*). The availability of a tool like the CMax Converter makes the translation of large existing codes possible but still not straightforward. It is nonetheless an easier task than rewriting a large code form scratch.

More parallel efficiency can still be extracted from the code by restricting computations to the subdomain interiors, and, whenever possible, by use of asynchronous communications. By this approach, communications and computations can be partially overlapped in real time. Both these improvements involve considerable rewriting of the code.

Domain decomposition methods appear as a very efficient parallel approach to the solution of large linear systems. The results of the parallel additive Schwarz method show that one can tackle large problems with a conceptually simple solver. More tests are still needed to check the robustness of the method in the presence of physical property heterogeneities and/or anisotropies.

ACKNOWLEDGMENTS

The financial support from the State of Texas, the U.S. Department of Energy and participating companies of the Flow in Porous Media Parallel Computation Project at Rice University is acknowledged. Special appreciation goes to Adam Greenberg from Thinking Machines Corp. and Kirk Jordan from Kendall Square for unlimited help and guidance in this research. We also like to thank Thinking Machines Corp. for providing the computing resources for this work.

References

[1]J. Killough and R. Bhogheswara, Simulation of Compositional Reservoir Phenomena on a Distributed Memory Parallel Computer, Jour. Pet. Tech., Nov. 1991, pp. 1368-1374.

[2]M. Espedal, P Langlo, O. Seavareid, E. Gislefoss and R. Hansen, Heterogeneous Reservoir Models: Local Refinement and Effective Parameters, in Proceedings of Eleventh Symposium on Reservoir Simulation, Anaheim, CA, Feb 17-20, 1991.

[3]J. Slattery, Flow of Viscoelastic Fluids through Porous Media, AIChE Journal, v.13, n.6, November 1967, pp. 1066-1071.

[4]N. Saad, Field Scale Simulation of Chemical Flooding, PhD Dissertation, Department of Petroleum Engineering, The University of Texas at Austin, 1989.

[5]G. Pope, L. Lake, K. Sepehrnoori, Modeling and Scale-up of Chemical Flooding-Third Annual and Final Report for the Period Oct 1987 - Sept 1988, Prepared for the US Department of Energy under Contract no. DE-AC19-85BC10846, Bartlesville, OK, March 1990.

[6]A. Datta-Gupta, G. Pope, K. Sepehrnoori, and R. Thrasher, A Symmetric, Positive Definite Formulation of a Three-Dimensional Micellar/Polymer Simulator, SPE Reservoir Engineering, November 1986, pp. 622-632.

[7]M. Dryja and O. Widlund, Additive Schwarz Methods for Elliptic Finite Element Problems in Three Dimensions, in Proceedings of Fifth Conference on Domain Decomposition Methods for Partial Differential Equations, Edited by T. Chan, D. Keyes, G. Meurant, J. Scroggs and R. Voigt, SIAM, Philadelphia, 1992.

[8]M. Dryja and O. Widlund, Domain Decomposition Algorithms with Small Overlap, Technical Report 606, Department of Computer Science, Courant Institute, May 1992. (To appear: SIAM J. Sci. Stat. Comput.)

[9]O. Widlund, Iterative Substructuring Methods: Algorithms and Theory for Elliptic Problems in the Plane, in proceedings of First International Symposium on Domain Decomposition Methods for Partial Differential Equations, SIAM, Philadelphia, 1988.

[10]W. Gropp and B. Smith, Experiences with Domain Decomposition in Three Dimensions: Overlapping Schwarz Methods, Technical Report, Mathematics and Computer Science Division, Argonne National Laboratory, 1992. (To appear in proceedings of Sixth International Symposium on Domain Decomposition Methods)

[11]J. Wallis, Vectorization of Preconditioned Generalized Conjugate Residual Methods, Mathematical and Computational Methods in Seismic, Exploration and Reservoir Modeling, SIAM, Philadelphia, 1987, pp. 250-251.

[12]P. Vinsome, Orthomin, An Iterative Method for Solving Sparse Banded Sets of Simultaneous Linear Equations, in Fourth Symposium on Numerical Simulation of Reservoir Performance, Los Angeles, 1976 (SPE paper 5729).