Parallel Algorithms for

Modeling Flow in Permeable Media

ANNUAL REPORT

FEBRUARY 15, 1995 — FEBRUARY 14, 1996

Principal Investigators: G. A. Pope, K. Sepehrnoori, D. C. McKinney
Participants: M. Delshad, M. G. Edwards, L. W. Lake, M. T. Lim,
J. Liu, G. E. Speitel, Jr., T. Ucpinar, C. H. Wang, P. Wang

Center for Petroleum and Geosystems Engineering
The University of Texas at Austin
Austin, TX 78712

and

Principal Investigator: M. F. Wheeler
Participants: T. Arbogast, C. N. Dawson, P. T. Keenan,
H. Klee, D. W. Moore, L. F. Pavarino, M. Ramé

Computational and Applied Mathematics Department
Rice University
Houston, TX 77251-1892

March 1996

PREPARED FOR THE U.S. DEPARTMENT OF ENERGY
UNDER GRANT NUMBER DE-FG05-92ER25142

DISCLAIMER

TABLE OF CONTENTS

Summary 1

1.0 Introduction 2

2.0 Parallel Computing 3

2.1 CRAY T3D Architecture 6

2.2 CRAY T3E Architecture 7

2.3 Differences Between CRAY T3D and CRAY T3E 8

3.0 Description of the Model 11

4.0 Parallel UTCHEM 12

4.1 Program Initialization 13

4.2 Network Distribution 13

4.3 I/O Handling 14

4.4 Vectorization Issue 15

4.5 Implementation of Message-Passing Functions 16

5.0 Sample Data Description 17

6.0 Parallel Simulator Performance 17

87.0 Summary and Conclusions 21

References 23

Tables 26

Figures 27

SUMMARY

This report describes the application of distributed-memory parallel programming techniques to a compositional simulator called UTCHEM. The University of Texas Chemical Flooding reservoir simulator (UTCHEM) is a general-purpose vectorized chemical flooding simulator that models the transport of chemical species in three-dimensional, multiphase flow through permeable media. The parallel version of UTCHEM addresses solving large-scale problems by reducing the amount of time that is required to obtain the solution as well as providing a flexible and portable programming environment.

In this work, the original parallel version of UTCHEM was modified and ported to CRAY T3D and CRAY T3E, distributed-memory, multiprocessor computers using CRAY-PVM as the interprocessor communication library. Also, the data communication routines were modified such that the portability of the original code across different computer architectures was made possible.

Several simulations were performed in order to assess the performance of the code on the parallel computers. In this report, we present the performance results of the most time-consuming subroutines in the simulator as well as the performance results of the overall code.

The application of distributed-memory parallel programming on UTCHEM has proven to be an achievable method for decreasing the turn-around time of the simulator and enhancing the computational capabilities of the code for large-scale reservoir simulations.

1.0 INTRODUCTION

Many problems of interest in the petroleum industry call for massive computing power for the numerical simulation of oil recovery processes. The need for improved accuracy of the numerical schemes, however, leads to an inevitable overall refinement of the discretizations. Unfortunately, the conventional small-scale computers are quickly becoming inefficient for solving numerical modeling of these problems in terms of speed and capacity (Pashapour *et al.*, 1989). Although large-scale vector supercomputers, such as CRAY YMP and CRAY Y-MPC90, could meet the requirements, their costs are still unaffordable for many of the research institutions.

The growing availability of distributed parallel computing architectures provides a solution to the petroleum engineering problems at a reasonable cost. In addition to the low cost (compared to the high-end super computers) multiprocessor computers, many organizations have networks of workstations available for implementing large-scale parallel programs. Parallel programming in clusters of multiprocessor environments enables most of these companies to attack large-scale problems with little extra hardware cost.

The University of Texas Chemical Flooding reservoir simulator (UTCHEM) is a general-purpose chemical flooding simulator that models the transport of multicomponent chemical species in three-dimensional, multiphase flow through permeable media with variable temperature. The first parallel version of UTCHEM was implemented by researchers at Rice University (Ramé *et al.*, 1993; Ramé and Delshad, 1995). It relies upon the application of message-passing implementation of the parallel programming practices by using the domain decomposition approach. It was designed to use a large number of processors in the solution of a single problem by using the Single-Program Multiple-Data (SPMD) programming model. Later, we ported the code to CRAY T3D and T3E and IBM SP1 and SP2 multiprocessor environments and also to a heterogeneous

network of workstations using PVM and C-Linda. This report presents the results of simulations on CRAY T3D and T3E systems only. Although the code was implemented and tested on the other systems, those results will be the subject of our future reports. Descriptions of CRAY T3D and T3E as well as their performance comparisons are also summarized in this report.

2.0 PARALLEL COMPUTING

Parallel programming has emerged as an enabling technology in modern computing, driven by the ever-increasing demand for higher performance, lower cost, and sustained productivity in real-life computer applications. Concurrent events are taking place in today's high-performance computing because of the common practice of multiprogramming, multiprocessing, or multicomputing (Hwang, 1993).

From the earliest days of reservoir simulation, numerical models have continued to test the capacity of the existing computers. From both numerical and physical points of view, large numbers of gridblocks with more physical phenomena included are required to adequately model processes in reservoirs (Pashapour *et al.*, 1989; Killough, 1993).

Beginning in the mid-1970s the introduction of the large-scale vector super computers completely changed the traditional approach that had been taken toward the development of the numerical modeling of reservoir simulation. Although the vectorization approach dramatically increased the efficiency of the reservoir simulators, two important factors decreased its wide-scale applicability. First, the implementation of vectorization led to significant reorganization and recoding of existing numerical models. Second, the costs of the vector supercomputers were not affordable for the majority of research institutions.

The second approach for improved reservoir simulations was the utilization of shared-memory parallel supercomputers. Several publications in the literature have dealt

with the application of parallel computing to petroleum reservoir simulation in shared-memory parallel environments. Scott *et al.* (1987) investigated the parallelization of linear equation solvers and coefficient matrix routines. Chen *et al.* (1987) worked on compositional modeling using parallel processing on a CRAY X-MP. Barua and Horne (1989) implemented a solver for solution of a system of nonlinear equations in a black-oil simulator on an IBM-3090 parallel vector computer. Pashapour *et al.* (1989) worked on the original version of the UTCHEM simulator, addressing the vectorization and microtasking parallel processing techniques.

Another approach to the parallel processing is the application of distributed-memory computers. This approach employs the application of the Multiple Instruction Multiple Data (MIMD) parallel programming technique in which a number of independent processors work on the solution of a single problem. This advancement in reservoir simulation modeling, simulation on distributed-memory machines, has been performed by many researchers. Several authors, including Bhogeswara and Killough (1993) and Cowsar *et al.* (1991, 1992) worked on the parallel solution of linearized finite-difference equations. Mayer (1989) solved the Buckley-Leverett problem on a Thinking-Machines CM-2 machine and showed that the machine can handle large problems as well. Later, Rutledge *et al.* (1991) ported a three-phase three-dimensional black-oil simulator to a parallel architecture and successfully simulated a reservoir problem with two million gridblocks. Killough and Bhogeswara (1991) ported a commercial n-component, three-phase, equation-of-state simulator on an iPSC/860. Arbogast *et al.* (1994) developed a simulator in three spatial dimensions for two-phase groundwater flow and transport with biodegradation kinetics for massively parallel distributed-memory architectures. Ramé and Delshad (1995) ported a vectorized chemical flooding compositional simulator (UTCHEM) to a distributed-memory massively parallel computer. In spite of the

extensive research in this area, the issue of efficient utilization of distributed computing in the reservoir simulation area needs a lot of additional research.

The research that this report summarizes involves the application of distributed-memory computing practices on CRAY T3D and CRAY T3E. Each processor in the network exclusively owns its memory and computing power. The data and the amount of work are distributed equally among the processors. If, during the calculations, the processors need to share their data, they do it through message-passing methods, which will be discussed later.

The performance gain that can be obtained by improving the parallelization of the program can be calculated by using Amdahl's law (Hennessy and Patterson, 1990). Amdahl's law states that the performance improvement to be gained from using parallel execution is limited by the fraction of the time the parallel mode can be used. Assuming that we can execute the entire algorithm in parallel and use message passing in interprocessor communication, the amount of time, t_p, necessary to compute the entire algorithm in parallel is

$$t_p = \frac{t_s}{N} + t_{mp} \tag{1}$$

where t_s is the time that it takes to run the scalar version of the program, N is the number of processors, and t_{mp} is the time spent on message passing. Then the speed-up for the algorithm will be (Ramé and Delshad, 1995)

$$\text{Speed} - \text{up} = \frac{N}{1 + \frac{t_{mp} N}{t_s}} \tag{2}$$

Equation 2 indicates that the speed-up is theoretically limited by the amount of time spent on interprocessor communications, t_{mp}. The relative message passing overhead is t_{mp}/t_s. The maximum speed-up can be attained when this ratio goes to zero. This is a clear indication that for maximum efficiency, message-passing operations must be kept at

a minimum. Therefore, the interprocess communication scheme was modified in UTCHEM. Although the new interprocess communication scheme increased the complexity of the implementation algorithm, it drastically reduced the number of data exchange communications and thus improved the performance. Isolating the data exchange from the rest of the calculations enabled the implementation of a portable code across different computer architectures.

The simulator was also modified to use general interprocess communication packages, including PVM and C-Linda. The code was ported to CRAY T3D and T3E, IBM SP1 and SP2 architectures, and heterogeneous networks of workstations. Although we have completed the porting of the code to above mentioned platforms, here we only discuss the simulation results executed on CRAY T3D and T3E multiprocessor computers.

2.1 CRAY T3D Architecture

The CRAY T3D system is a Massively Parallel Processing (MPP) architecture that contains up to 2048 microprocessors, each accompanied by its own local memory (CRAY T3D Hardware Review, 1993). Each microprocessor of the CRAY T3D system is a DECchip 21064 (DEC Alpha EV4) from Digital Equipment Corporation capable of 150 MFLOPS peak performance. This reduced instruction set computing (RISC) microprocessor is cache-based, has pipelined functional units, issues up to six instructions per cycle, and supports IEEE standard 64-bit floating-point arithmetic. Each processor has its own local DRAM memory with a capacity of 64 Mbytes. Each processing element (PE) in the CRAY T3D system comprises the DEC Alpha microprocessor, local memory, and Cray Research-designed support logic. A CRAY T3D system node consists of two PEs sharing the Cray Research-designed switch and network support logic. The system is designed to support different styles of MPP programming, such as data parallel, work sharing, and message passing (Hennessy and Patterson, 1990).

The uniqueness of the CRAY T3D system is the result of its design implementation. With its current implementation, the CRAY T3D system connects to a host computer system that provides support for applications running on the T3D; that is, the T3D is not a self-sufficient system on which an application can be built and run. Instead, all applications written for the T3D are compiled on the host system and run on the T3D. The host system can be any Cray Research computer that has a special input-output (I/O) system. These systems include the CRAY Y-MP, CRAY Y-MP M90, and CRAY Y-MP C90 series computer systems.

2.2 CRAY T3E Architecture

The CRAY T3E system is an MPP architecture that contains up to 2048 microprocessors, each accompanied by its own local memory (CRAY T3E Hardware Review, 1996). CRAY T3E parallel systems use the DECchip 21164 (DEC Alpha EV5) from Digital Equipment Corporation, capable of 600 MFLOPS peak performance. In addition to the improvements on the microprocessor, Processing Elements (PEs) in the CRAY T3E system are connected by a high-bandwidth, low-latency bidirectional 3-D torus system interconnect network six times faster per PE than that used in the CRAY T3D system. It incorporates an adaptive message-routing mechanism to allow messages on the interconnect network to be rerouted around temporary "hot spots." Interprocessor data payload communication rates are 480 Mbytes per second in every direction through the torus; in a 512-PE CRAY T3E system, bisection bandwidth exceeds 122 Gbytes per second.

The CRAY T3E system performs I/O through multiple ports onto one or more scalable GigaRing channels. This dual-ring I/O channel, with data in the two rings traveling in opposite directions, delivers high I/O data bandwidth and enhances reliability. On the CRAY T3E system, each GigaRing channel has a maximum data payload bandwidth of 1 Gbyte/s. The largest CRAY T3E system configurations are capable of up

to 128 Gbytes/s of I/O bandwidth. Multiple channels can be configured to provide maximum disk throughput while maintaining the highest overall I/O bandwidth. Disk, tape, network, and system nodes can be configured in any number of combinations on multiple I/O channels.

2.3 Differences Between CRAY T3D and CRAY T3E

2.3.1 The operating system

To support the scalability of the CRAY T3E system, the operating system of CRAY T3D, UNICOS, was redesigned into UNICOS/mk. Unlike UNICOS in CRAY T3D, UNICOS/mk is distributed among the CRAY T3E system's PEs, not replicated on each. This distribution of operating system functions provides a global view of the computing environment - a single-system image - that allows administrators to manage a systemwide suite of resources as a single entity. As a result, systemwide resource management and utilization are greatly simplified compared to the previous versions of UNICOS.

UNICOS/mk is divided into "servers," which are distributed among the processors of a CRAY T3E system. Local servers process operating-system requests specific to each user PE. Global servers provide systemwide operating-system capabilities, such as process management and file allocation. In addition to the user PEs that run applications and commands, CRAY T3E systems include dedicated system PEs that run the global UNICOS/mk servers. Because global services are provided by system PEs and not replicated throughout the CRAY T3E system, UNICOS/mk efficiently scales, with full functionality, to service from tens to thousands of PEs with minimal overhead.

2.3.2 The I/O channels

The CRAY T3E system performs I/O through multiple ports onto one or more scalable GigaRing channels. This dual-ring I/O channel, with data in the two rings

traveling in opposite directions, delivers high I/O data bandwidth and enhances reliability. This multichannel architecture is different from what is implemented in CRAY T3D. In CRAY T3D, all processors in the system direct their I/O requests to a single dedicated I/O processor.

Regardless of size, all CRAY T3E systems may be configured with an I/O channel for every eight PEs (air-cooled) or 16 PEs (liquid-cooled). All I/O channels are accessible and controllable from all PEs. On the CRAY T3E system, each GigaRing channel has a maximum data payload bandwidth of 1 Gbyte/s and provides high-speed access to peripherals, networks, and other systems.

2.3.3 Torus interconnect network

The interconnect network provides the communication paths among the processing element nodes and the input/output gateways in CRAY T3D and T3E systems. The interconnect network forms a three-dimensional matrix of paths that connects the nodes in x, y, and z directions (Fig. 1), which enables the communication network to transfer data and control information between processing element nodes.

The efficiency of this network configuration is the result of two unique architecture design features. First, each communication link is composed of two different channels, that is, each PE node has a unidirectional send and a unidirectional receive channel as shown in Fig. 2.

These unidirectional communication links enable the two nodes to send and receive data and control signals simultaneously. In addition to this efficient communication scheme, communication links handle data send and receive operations through the usage of two separate buffers, which enables the system to transmit data more efficiently. In CRAY T3D each node consists of two microprocessors. As a result, each microprocessor needs to arbitrate for the interconnect network utilization with its pairing microprocessor.

CRAY T3E architecture eliminates this extra arbitration time by implementing only one microprocessor per node.

The second important feature of the interconnect network is its interleaving torus topology design (Fig. 3). A torus contains communication links that connect the smallest numbered node in a dimension directly to the largest numbered node in the same direction. This type of connection forms a ring where information can transfer from one node through all the nodes in the same dimension, and back to the original node.

In addition to the torus topology, the nodes in the interconnect network are interleaved, which means the physical placement of nodes is arranged so that the maximum wiring distance between nodes is minimized, resulting in faster communication rates.

This configuration of the torus network offers several advantages for network communications. One advantage is speed of information transfers. For example, in Fig. 3, Node 5 directly communicates with Node 0 instead of sending information through all of the nodes in one direction.

Another advantage of the interconnect networking scheme is the ability to avoid bad or busy communication links. For example, in Fig. 3, if Node 0 cannot transfer information directly to Node 1, it can still communicate with Node 1 by sending the information around the network through other nodes by using another direction.

In CRAY T3D the maximum data throughput on this interconnect network is 160 MB/sec whereas in CRAY T3E, this number was improved to 480 MB/sec.

2.3.4 Data cache

In CRAY T3E, there are two enhancements to the original data cache architecture of the CRAY T3D. The first change is the addition of a secondary cache (Scache). The secondary cache is a unified data and instruction cache. It is a 96 KB, three-way set-associative , write-allocate, write-back cache. The cache line size is 64 bytes. The Scache

latency is 8 cps or 26.6 ns. The Scache bandwidth is 4.8 GB/sec. The Scache (and therefore the Dcache as a proper subset) is hardware cache coherent with any remote references from other T3E nodes. It will allow two outstanding off-chip memory references without stalling. This means two cache misses to Scache can be taken without stalling the processor.

The second difference between the CRAY T3D and T3E is the existence of a third level of cache structure on the CRAY T3E. This is called the Stream Buffers. Stream Buffers amount to a tertiary level of cache that lies between the CPU and the local DRAM memory. There are six Stream Buffers, each with four 64-byte cache lines. The Stream Buffers are used by the hardware to transparently prefetch cache lines of data from DRAM asynchronously. Sustainable bandwidth to the Stream Buffers is approximately 600 MB/sec. The Stream Buffers are currently not cache-coherent with the on-chip caches, although there are some rather simple programming techniques that can be employed to make an application STREAMS-SAFE. A PVM or MPI program is automatically STREAMS-SAFE, which is ensured by the libraries automatically.

3.0 DESCRIPTION OF THE MODEL

UTCHEM is a multiphase, multicomponent, three-dimensional finite-difference simulator. It was originally developed to model surfactant enhanced oil recovery but modified for applications involving the use of surfactant for enhanced remediation of aquifers contaminated by nonaqueous-phase liquids (NAPLs). The balance equations in the parallel version are the mass-conservation equations and an overall balance that determines the pressure for up to three liquid phases. The vectorized version of UTCHEM, however, also solves an energy balance equation to determine the temperature and also has the capability of modeling the gas phase. The number of components is variable, depending on the application. When electrolytes, tracers, co-solvents, polymer, and other commonly needed components are included, the number of components may be

on the order of 12 or more. When the geochemical option is used, a large number of additional aqueous components and solid phases may be used. For the most recent description of UTCHEM, the readers are referred to Delshad *et al.* (1996).

The resulting flow equations are solved using a block-centered finite-difference scheme. The solution method is implicit in pressure and explicit in concentration (IMPES type). One- and two-point upstream and third-order spatial discretization are available as options in the code. To increase the stability and robustness of the second- and third-order methods, a flux limiter that is total-variation-diminishing (TVD) has been added (Liu, 1993). The third-order method gives the most accurate solution. A detailed description of UTCHEM formulation and numerical solution scheme is given elsewhere (Datta Gupta *et al.*, 1986; Saad, 1989; Liu *et al.*, 1994; Delshad *et al.*, 1996).

4.0 PARALLEL UTCHEM

Parallel processing is a practical means to speed up the turn-around time of large-scale computational problems. To achieve this goal, various parallel algorithms have been proposed in the past and tested in several parallel-programming environments (Azencott, 1992). The basic idea is to solve a large-scale engineering problem in a reasonable amount of time. The parallel version of the UTCHEM simulator addresses solving large-scale problems by reducing the amount of time that is required to obtain the solution as well as providing a flexible and a portable programming environment.

The first parallel version of UTCHEM was implemented by researchers at Rice University (Ramé *et al.*, 1993; Ramé and Delshad, 1995). It relies upon the application of message-passing implementation of the parallel programming practices. It was designed to use a large number of processors in the solution of a single problem by using the Single-Program Multiple-Data (SPMD) programming model.

The following sections explain the implementation of parallelization techniques in UTCHEM. It is not our intention to show how the problem is numerically solved but to show how the parallel implementation is performed in the parallel version of UTCHEM.

4.1 Program Initialization

Upon execution, the first copy of the program, generally referred as Node-0, checks for the initialization of the program and starts the other nodes if necessary[†]. As soon as the network initialization is complete, the parallel version of the code distributes the spatial gridblocks among the processors. Each processor receives data for several gridblocks of the reservoir for which the computations must be performed. This distribution is controlled by the Node-0, whose prime responsibility at this point is to ensure an even computational load distribution within the network.

4.2 Network Distribution

The two primary reasons for the implementation of a parallel program are to decrease the time required to solve the problem and to overcome the computer memory requirements of a large problem (Ouenes, 1993). As a solution to the second problem, the original array size implementation in UTCHEM was modified so that each processor in the network holds a subset of what is required in order to work on a subdomain of the reservoir. As a result, each processor in the network holds a data array just large enough to hold the subdomain plus a three-dimensional envelope around the subdomain, so that any data required from the neighboring nodes can be stored.

The gridblock distribution scheme assigns consecutively numbered blocks to the nodes of the virtual machine. Since the nearest nodes are frequently located in the same

[†] If the program is compiled for the PVM CRAY T3D or T3E environments, this step is unnecessary since the MPP system automatically starts all the nodes. On the other hand, for all C-Linda and other PVM versions, this step is essential.

computing node, this distribution scheme results in a reasonable locality of computation and a static load balance (Ramé and Delshad, 1995).

Each node receives a number of gridblocks according to the following formula:

$$N^{SUB}_{BL(x,y,z)} = \frac{N_{BL(x,y,z)}}{N_{CPU(x,y,z)}} + N_{neighbors} \tag{3}$$

where $N^{SUB}_{BL(x,y,z)}$ is the number of gridblocks in the subdomain, $N_{BL(x,y,z)}$ is the total number of gridblocks in the problem, $N_{CPU(x,y,z)}$ is the number of partitioning CPUs or nodes in each direction, and $N_{neighbors}$ is the number of neighbors of a subdomain in a given spatial direction.

An example of the subdomain storage (Fig. 4) can be demonstrated as follows. For a given reservoir simulation, assume that a hypothetical reservoir discretization scheme is 15x20x6 in x, y, and z directions, respectively. Furthermore, the available number of processors is 12. A suitable subdomain configuration for the given problem would be 3x4x1 nodes in x, y, and z dimensions. As a result, the subdomain array dimensions for each node will be 7x7x6. Figure 4 shows the relationship between the gridblock scheme for the problem and the required number of gridblocks within each subdomain. Remember that the parallelization of UTCHEM was implemented by a domain-decomposition scheme. During the course of the simulation, each process shares some of its data with the neighboring processes. The shaded areas represent the shared gridblocks between the neighboring subdomains.

Once the subdomain decomposition is completed, the computations are performed within each subdomain as a subset of the global problem. Since each subdomain requires some data from its neighbors, the processors share the required data by using explicit message-passing function calls.

4.3 I/O Handling

In parallel implementation of UTCHEM, the I/O is handled by a serial implementation through a designated node. During the initialization stage of the program, the designated processor, Node-0, is responsible for the data input and the distribution to the remaining nodes. Similarly, during the data output, Node-0 collects the output, by using simple message-passing function calls, from the other nodes in the network and writes them to the appropriate output files.

Using a designated process approach for I/O handling, however, creates two major problems. First, since Node-0 is the only processor that handles data I/O, it has to read the input data and distribute it through interprocess communication. This creates an undesirable situation where all the processing nodes, except Node-0, are kept idle during the data input process. Similarly, since the output data are sent through the message-passing algorithms, the system might be faced with an insufficient memory problem in case of an operation involving large amount of data output .

During a message-passing operation, PVM allocates a temporary memory buffer for each incoming message. Assuming a 64-node simulation run, 63 nodes send data to the Node-0. As a result, the message-passing environment creates 63 incoming message buffers, which may cause an insufficient memory problem. It is these authors' opinions that in the future versions of the code, this problem must be handled more efficiently.

4.4 Vectorization Issue

The original implementation of UTCHEM was specifically designed for the vector supercomputers (Pashapour, 1988), where the array components of the code were declared as long vectors with each entry referring to a physical location in the reservoir by means of a given gridblock numbering scheme. Unfortunately, this scheme is not particularly suitable for parallel computing (Booth and Misegades, 1986).

An implementation difficulty occurs during the optimization stage of do-loops in the vectorized codes. The parallelization of UTCHEM was implemented by a domain-decomposition scheme (Fig. 4). During the simulation, each process shares some of its data with the neighboring processes. This intersubdomain data sharing creates a huge computational overhead, since do-loops sweep over those padded regions where each newly calculated values are incorrect and must be replaced with the correct values known by the neighboring processes (Ramé and Delshad, 1995). Thus, a full new approach for the loops was implemented. Although the vectorized code was preserved, the coding complexity of the program was increased by allowing a single-step data communication resulting in less computational overhead.

4.5 Implementation of Message-Passing Functions

The message-passing algorithm was first implemented by Ramé *et al.* (1993) for Intel systems using NX library environment. In that version, the communications are handled synchronously, meaning that no computations are being performed during the interprocessor communications. The message-sending process waits until the destination process receives the data. This behavior was the default characteristic for the CM5 implementation of the next generation version of the program.

The PVM and C-Linda version of the code, however, do not rely on the synchronous message-passing approach, since this degrades the efficiency of the code where the network is composed of a cluster of workstations. The current implementation includes a nonblocking message-passing algorithm in which each processing node sends its data to a neighboring node and continues working on its subdomain. This will assure the maximum CPU usage within the network.

In addition to the asynchronous communication scheme, the code collects all of its communication-related functions into a single C-programming-language source-code file. This approach enables a straightforward implementation of a superset of original

communication functions for each new computer network. Using a compile time argument, it is possible to configure the same source code for different architectures without any modifications in the original FORTRAN source code.

5.0 SAMPLE DATA DESCRIPTION

The test cases were based on a polymer flooding pilot test in the Courtney sand of Chateaurenard field in France. The performance of this field was first simulated by Takagi *et al.* (1992) by using the sequential version of UTCHEM. Takagi used 25x25x3 gridblocks as the finest grid with gridblock sizes of 65 ft in length, 65 ft in width, and 3 ft in depth.

The reservoir description used in this example was a layered one, and therefore the discretization in both areal dimensions can easily be changed in order to test the code performance versus changing the surface-to-volume ratios in the subdomains. The configuration of the pilot was an inverted five-spot pattern with an average distance of 756 m between producing wells. The pilot operation consisted of multiple injection of a slug of 0.1 wt% hydrolyzed polyacrylamide tapered down to a concentration of 0.02 wt% in the final slug and followed by chase water injection.

To investigate large-sized problems in the parallel environment, the number of gridblocks was first increased from 25x25x3 to 50x100x6 while keeping the gridblock sizes the same. The simulations were executed for only 20 days where the original full simulation was 1540 days. The number of components for which the conservation equation is solved for is five in this example.

6.0 PARALLEL SIMULATOR PERFORMANCE

The above-mentioned pilot was discretized by using seven different numbers of gridblocks. The first case, 50x100x6 gridblocks, was chosen with the intent of comparing the efficiency of the CRAY T3D and T3E systems with that of other systems given in the

publication of Ramé and Delshad (1995). The later runs were conducted by increasing the number of gridblocks in the reservoir while keeping the gridblock sizes and the timestep constant. Although these new data sets do not represent the same reservoir given in Takagi *et al.* (1992), they were intended to keep the changes in the reservoir description to investigate the effects of number of gridblocks in the simulations.

The simulations were executed for 20 days because of the limited availability of computing time on parallel systems. Even in this limited amount of simulation time, the total initialization and data input took less than 1% of the total time in all cases. The results we present in this report can be linearly extrapolated for longer simulation times, since only a small percentage of the total time is consumed by initialization.

Figures 5 through 10 present the elapsed time for the most time-consuming subroutines as well as the total elapsed time for the entire simulation (less the output time) by using the Jacobi conjugate-gradient (JCG) as the linear solver. These results were obtained on a CRAY T3E system composed of 256 processors where each processor is equipped with 128 or 256 MB of memory. These results can be grouped into two categories: subdomain local calculations, where no interprocess communication is required, and stencil computations, where each subdomain requires data sharing with its neighboring subdomains.

The first group of routines involves the calculation of local physical properties such as *viscos* and *trap*. The block-property calculations involve the local properties. Therefore, no interprocess data exchange is necessary. Nevertheless, increasing the number of processors decreases the efficiency of these calculations because of the limited size of the processor cache. However, this decrease in the efficiency of the calculations remains low, as expected.

The second group, on the other hand, calculates the explicit timestepping of the mass-conservation equations (e.g. *coneq*) and the pressure-distribution calculations for the

entire system for each timestep (e.g., *solmat*). Performance degradation because of increasing number of processors in this group shows different characteristics depending on the number of gridblocks in the simulation. The general trend is that the performance of these calculations improves as the number of gridblocks increases. This is because of the fact that with the increasing number of gridblocks, each processor spends more time in calculations rather than message passing. This increases the efficiency of the total code (refer to Eq. 2). On the other hand, increasing the number of processors decreases the performance of each simulation since each processor needs to work on fewer and fewer gridblocks while the total number of communications increases. In the cases with small number of gridblocks this effect is so obvious that increasing number of processors does not improve the performance at all.

On CRAY T3E, all the sample cases showed an acceptable speed-up up to 32 processors. However, for the small-sized simulations, the performance started to degrade after 32 processors. Regardless of the number of gridblocks simulated, the above-mentioned two groups of subroutines are distinguishable. Subroutines using the local data (e.g., *viscos* and *trap*) continuously show a good scale-up regardless of the number of gridblocks, whereas the second group of subroutines (e.g., *solmat* and *coneq*) showed different characteristics for different number of gridblocks. Figures 11 through 16 show the speed-up for the simulations normalized to the minimum number of processors that can handle the memory requirements of each run. Since the subroutine *solmat* includes the JCG solver, which relies heavily on a dot product and matrix vector multiplication and takes a significant number of iterations to converge, of the given subroutines, solmat shows the worst speed-up among all subroutines. As expected, the speed-up gets closer to the ideal line as the problem size becomes larger (Fig. 16).

It is difficult to compare the speed-ups given in Figures 11 through 16 because each group of runs was normalized to a different number of processors depending on the

size of the problem. This was inevitable, since the increasing number of gridblocks requires a proportionally increasing memory size on each processor. Because of the memory limitations, we were forced to use an increasing number of processors just to be able to fit the code on each processor.

Figures 17 summarize the speed-ups observed when changing from one number of processors to another. For a relatively small number of processors, speed-up gets very close to the ideal speed-up as the number of gridblocks increases. This speed-up diminishes, as expected, as the number of processors increases. This does not always follow a certain trend. The reason for the small anomalies in the speed-up curves are due to the changing effects of the system data cache. Increasing the number of processors makes the system data cache more effective. However, if the number of gridblocks in each subdomain gets too small, the system starts consuming most of its time on interprocessor communications, which effectively diminishes the simulation performance. On the other hand, if the number of gridblocks in each subdomain is too large, the effectiveness of the data cache diminishes. These effects generate irregularities in the speed-up diagrams.

Because of the limited availability of system time, we were able to simulate only two of the sample cases on the CRAY T3D system. Figures 18 through 21 present the performance of each run conducted on CRAY T3D with up to 64 processors. Because of the limited memory size of each processor (64 MB per processor), we were not able to run one- and two-processor configurations for the 30,000 gridblocks case. Similarly, it was impossible to run the one-, two-, and four-processor cases for 60,000 gridblocks.

In the CRAY T3D, the general trend in timing of individual routines observed in the T3E was also observed. The only difference is that the performance gains were about 300 to 600 % better in the CRAY T3E. The main reasons for these were explained in

Section 2.3 of this report, namely secondary cache effects, changes in the Torus Network, better I/O channels, and the changes in the Operating System itself.

One important observation in the runs executed on the CRAY T3D is that some of the subroutines demonstrated a behavior called *Hyper-Linear Speed-ups*. The hyper-linear speed-up is the condition in which a given subroutine scales up more than twice when the number of processors is doubled. We see this behavior especially in the small number of processors and in routines that take very little time to execute. This can be explained by two factors. First, the timing routines involve a small randomness in their implementation. Second, on the CRAY T3D, each processor has a very small system cache. As the computational load of a particular subroutine increases, these effects begin to disappear.

In order to give a better perspective of distributed-memory machines, we outlined the performance of the distributed memory machines given in Ramé and Delshad (1995), the performance of CRAY T3D and CRAYT3E, as well as the results of the performance run on a vector computer CRAY YMP. Table 1 summarizes these results.

7.0 SUMMARY AND CONCLUSIONS

Our experience with the UTCHEM compositional simulator and distributed-memory massively parallel machines suggests that the implementation of a parallel program with high levels of portability is an achievable objective while taking advantage of different computer architectures. In doing this, using a standard message-passing library enabled us to incorporate the vendor-specific implementations of parallel communication packages, such as CRAY-PVM, without changing the program implementation. Using a system-specific parallel programming technique may increase the performance of the overall code. This, however, may require a complete modification of the existing code, increasing the implementation time, and also results in a nonportable code. Although our implementation was a portable code and did not include any system-specific instructions,

we have shown that a good degree of speed-up is still achievable by distributing a single problem among several processors.

Based on our experience in this work, we emphasize the importance of the interprocess communication overhead. Maintaining a high degree of parallel efficiency is closely related to the amount of data that each process uses and exchanges with the others. As the surface-to-volume ratio decreases in each subdomain (an increased number of gridblocks), the amount of data that each process needs to share with its neighbor decreases, resulting in better performance. In this regard, the amount of memory, RAM, in each processor plays an important role, since a larger subdomain problem can be loaded on each processor.

An asynchronous communication scheme enables the code to overlap communications and computations, resulting in a better performance. However, this approach involves a careful redesign of the existing implementation.

The improved algorithms and increased performance of the processors enables us to implement distributed-memory parallel programs in which the performance of the code can be better than that of specialized vector processors.

REFERENCES

Arbogast, T., C. Dawson, and M. F. Wheeler: "A Parallel Multiphase Numerical Model for Subsurface Contaminant Transport With Biodegradation Kinetics," *Computational Methods in Water Resources* (1994) **2**, pp. 1499-1507.

Azencott, R.: *Simulated Annealing: Parallelization Techniques*, John Wiley, New York, NY (1992).

Barua, J. and R. N. Horne: "Improving the Performance of Parallel (and Serial) Reservoir Simulators," paper SPE 18408 presented at the SPE Symposium on Reservoir Simulation, Houston, TX, Feb. 6-8, 1989.

Bhogeswara, R. and J. E. Killough: "Parallel Linear Solvers for Reservoir Simulation: A Generic Approach for Existing and Emerging Computing Architectures," paper SPE 25240 presented at the 12th SPE Symposium on Reservoir Simulation, New Orleans, LA, Feb. 28-Mar. 3, 1993.

Booth, M. and K. Misegades: "Microtasking: a New Way to Harness Multi Processors," Cray Channels, Summer 1986.

Chien, M. C. H., M. L. Wasserman, H. E. Yardumian, and E. Y. Chung: "The Use of Vectorization and Parallel Processing for Reservoir Simulation," paper SPE 16025 presented at the SPE Symposium on Reservoir Simulation, San Antonio, TX, Feb. 1-4, 1987.

Cowsar, L. and M. Wheeler: "Parallel Domain Decomposition Method for Mixed Finite Elements for Elliptic Partial Differential Equations," in *Fourth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, R. Glowinski *et al.* (ed.), SIAM, Philadelphia (1991).

Cowsar, L., A. Weiser, and M. Wheeler: "Parallel Multigrid Domain Decomposition Algorithms for Elliptic Equations," in *Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, D.E. Keyes *et al.* (ed.), SIAM, Philadelphia (1992).

Cray T3D Hardware Review, Cray Research Inc., Chippewa Falls, WI (1993).

Cray T3D Hardware Review, Cray Research Inc., Chippewa Falls, WI (1996).

Datta Gupta, A., G. A. Pope, K. Sepehrnoori, and R. L. Thrasher: "A Symmetric, Positive Definite Formulation of a Three-Dimensional Micellar/Polymer Simulator," *SPE Reservoir Eng.* (Nov. 1986), **1**(6), 622-632.

Hennessy, J. L. and D. A. Patterson: *Computer Architecture. A Quantitative Approach*, Morgan Kaufmann Publishers Inc., Palo Alto, CA(1990).

Hwang, K.: Advanced Computer Architecture: Parallelism, Scalability, Programmability. , McGraw-Hill, Inc., New York, NY (1993).

Killough, J. E. and M. F. Wheeler: "Parallel Iterative Linear Equation Solvers: An investigation of Domain Decomposition Algorithms for Reservoir Simulation," paper SPE 16021 presented at the SPE Symposium on Reservoir Simulation, San Antonio, TX, Feb. 1-4, 1987.

Killough, J.E ., and R. Bhogeswara: "Simulation of Compositional Reservoir Phenomena on a Distributed Memory Computer," *Journal of Petroleum Technology* Nov. 1991, pp. 1368-74.

Killough, J. E.: "Is Parallel Computing Ready for Reservoir Simulation? A Critical Analysis of the State of the Art," paper SPE 26634 presented at the SPE Annual Technical Conference, Houston, TX, Oct. 3-6, 1993.

Liu, J.: "High Resolution Methods for Enhanced Oil Recovery Simulation," Ph.D. dissertation, U. of Texas at Austin (Aug. 1993).

Liu, J., M. Delshad, G. A. Pope, and K. Sepehrnoori: "Application of Higher Order Flux-Limited Methods in Compositional Simulations," *J. Transp. in Porous Media* (1994), **16**, 1-29.

Mayer, D. F.: "Application of Reservoir Simulation Models to a new Parallel Computing System," paper SPE 19121 presented at the SPE Petroleum Computer Conference, San Antonio, TX, Jun. 26-28, 1989.

Nolen, J. S. and P. L. Stanat: "Reservoir Simulation on Vector Processing Computers," paper SPE 9644 presented at the SPE Middle East Oil Technical Conference, Manama, Bahrain, March 14-17, 1981.

Ouenes, A. and N. Saad: "A New Fast Parallel Simulated Annealing Algorithm for Reservoir Characterization," paper SPE 26419 presented at the SPE Annual Technical Conference and Exhibition, Houston, TX, Oct. 4-6, 1993.

Pashapour, A., G. A. Pope, K. Sepehrnoori, and G. Shiles: "Application of Vectorization and Microtasking for Reservoir Simulation," in *Parallel Super Computing: Methods, Algorithms, and Applications*, G.F. Carey (ed.), John Wiley and Sons, New Yory, NY (1989).

Ramé, M., L. Pavarino, A. Greeberg, K. Jordan, and M. Wheeler: "Parallel Chemical Flood Simulation: An Implementation of UTCHEM on Distributed Memory Processors," CAAM-TR93-23 (July 1993).

Ramé, M. and M. Delshad: "A Compositional Reservoir Simulator on Distributed Memory Parallel Computers," paper SPE 29103 presented at the SPE Symposium on Reservoir Simulation, San Antonio, TX, Feb. 12-15, 1995.

Rutledge, J. M., D. R. Jones, and W. H. Chen: "The Use of Massively Parallel SIMD Computer for Reservoir Simulation," paper SPE 21213 presented at the SPE Symposium on Reservoir Simulation, Anaheim, CA, Feb. 17-21, 1991.

Saad, N.: "Field Scale Simulation of Chemical Flooding," Ph.D. dissertation, U. of Texas at Austin (May 1989).

Scott, S. L., R. L. Wainwright, R. Raghavan, and H. Demuth: "Application of Parallel (MIMD) Computers to Reservoir Simulation," paper SPE 16020 presented at the SPE Symposium on Reservoir Simulation, San Antonio, TX, Feb. 1-4, 1987.

Takagi, S., G. A. Pope, K Sepehrnoori, A. G. Puts, and H. Bendakhlia: "Simulation of a Successful Polymer Flood in the Chateaurenard Field," paper SPE 24931 presented at the SPE 67th Annual Technical Conference, Washington, DC, Oct. 4-7, 1992.

Young, L. C.: "Equation of State Compositional Modeling on Vector Processors," paper SPE 16023 presented at the SPE Symposium on Reservoir Simulation, San Antonio, TX, Feb. 1-4, 1987.

Table 1. A comparison of the elapsed times for the 30,000-gridblock polymer flooding simulation on different computers.

Run	CRAY Y-MP	CM5	T3D	T3E	T3E	T3E
No. of Processors	1	64	64	16	32	64
Elapsed Time, sec	188	1920	365	165	99	66

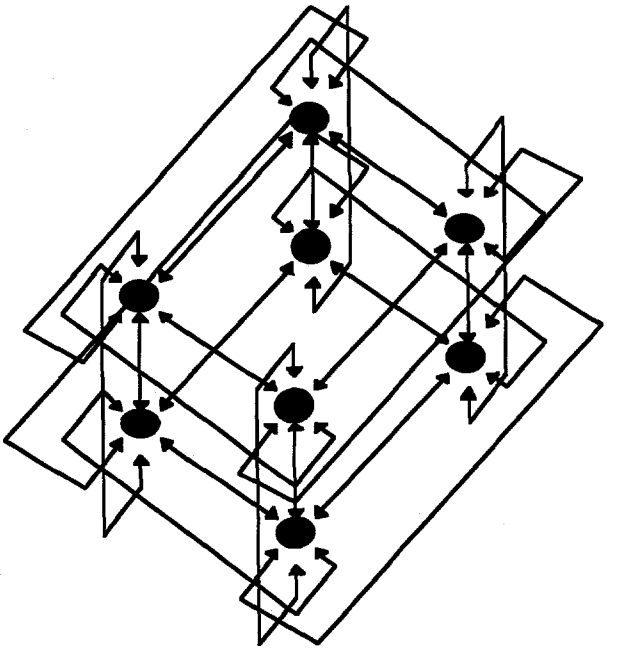

Figure 1. CRAY T3D and T3E interconnect network (CRAY T3D Hardware Review, 1993).

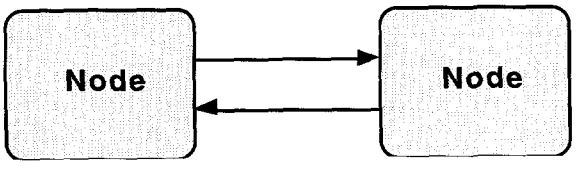

Figure 2. A communication link channel in T3D system.

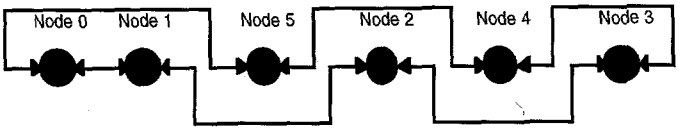

Figure 3. One-dimensional, interleaving Torus Interconnect Topology of T3D and T3E.

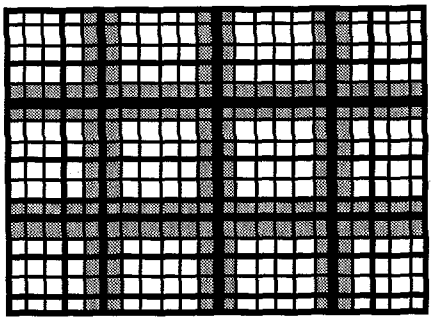

Figure 4. A sample reservoir gridblock scheme.

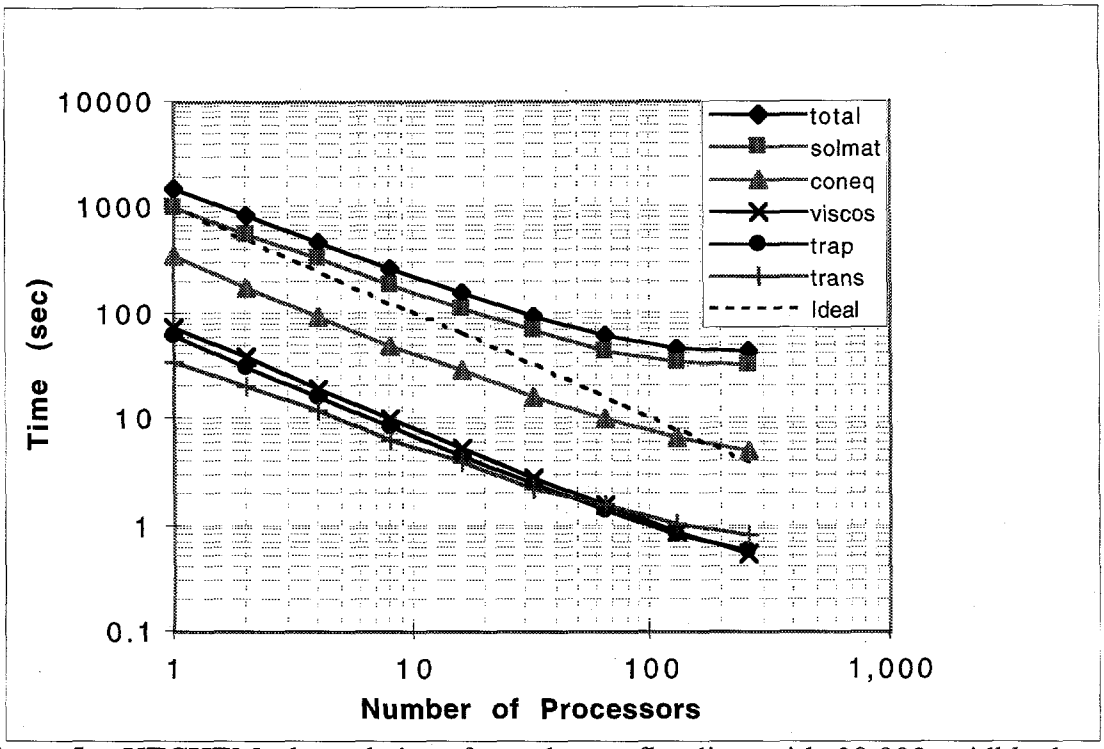

Figure 5. UTCHEM elapsed time for polymer flooding with 30,000 gridblocks on CRAY T3E.

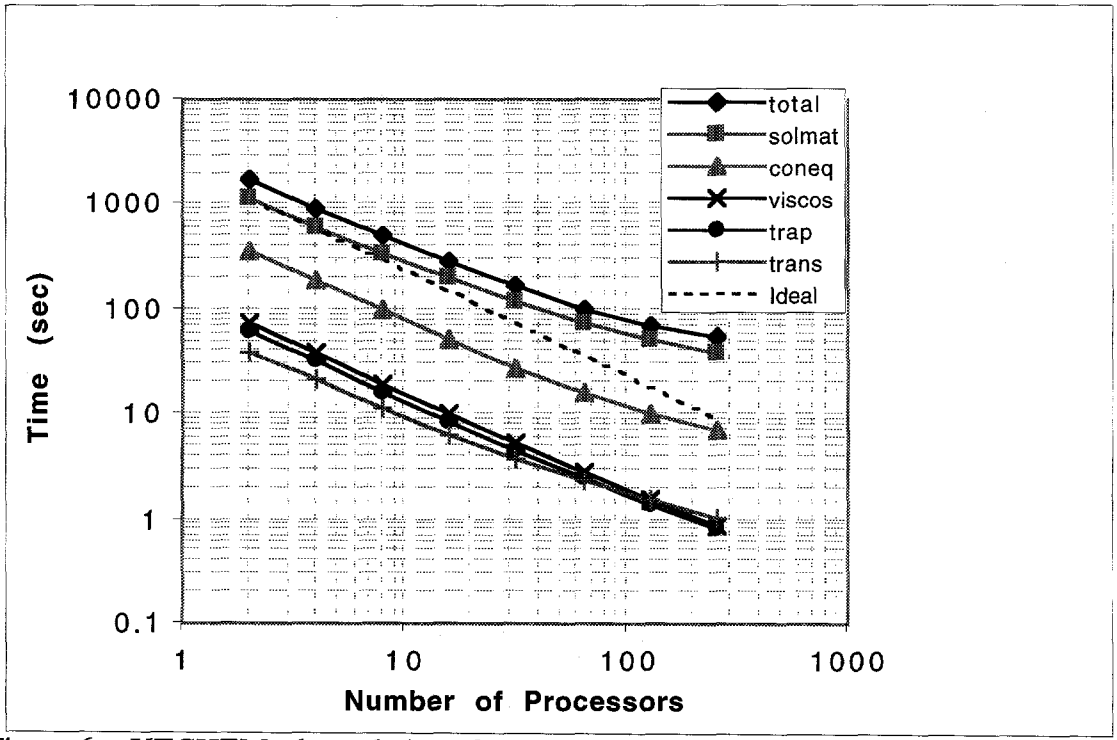

Figure 6. UTCHEM elapsed time for polymer flooding with 60,000 gridblocks on CRAY T3E.

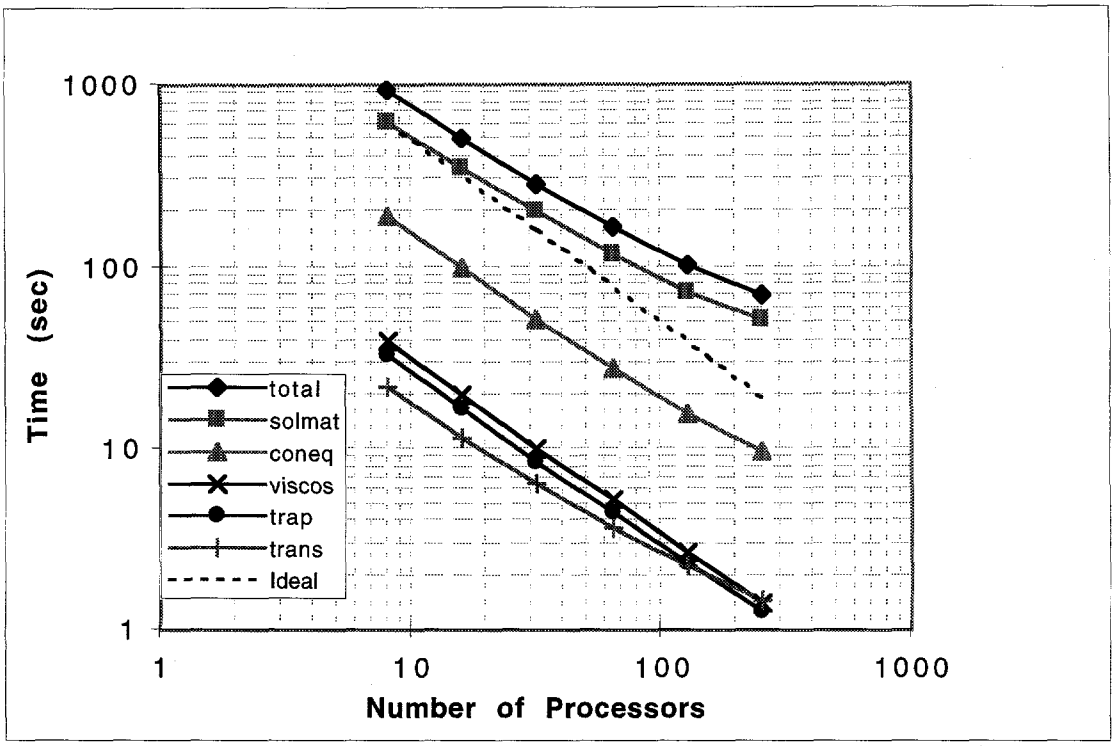

Figure 7. UTCHEM elapsed time for polymer flooding with 120,000 gridblocks on CRAY T3E.

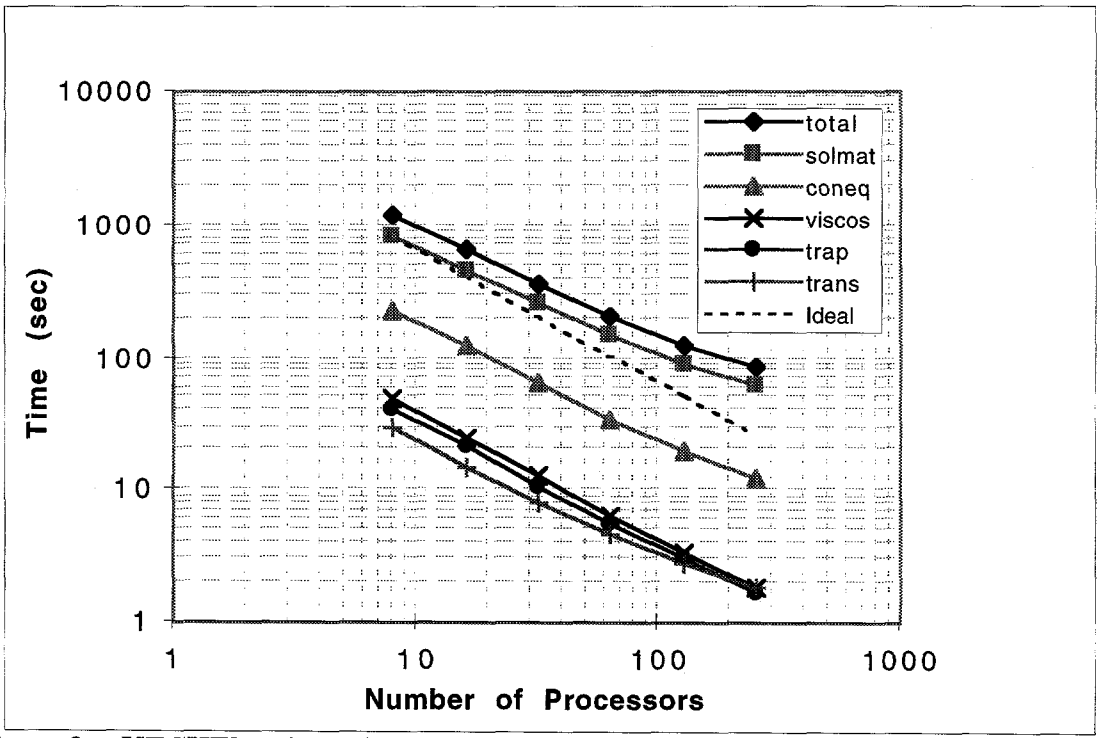

Figure 8. UTCHEM elapsed time for polymer flooding with 150,000 gridblocks on CRAY T3E.

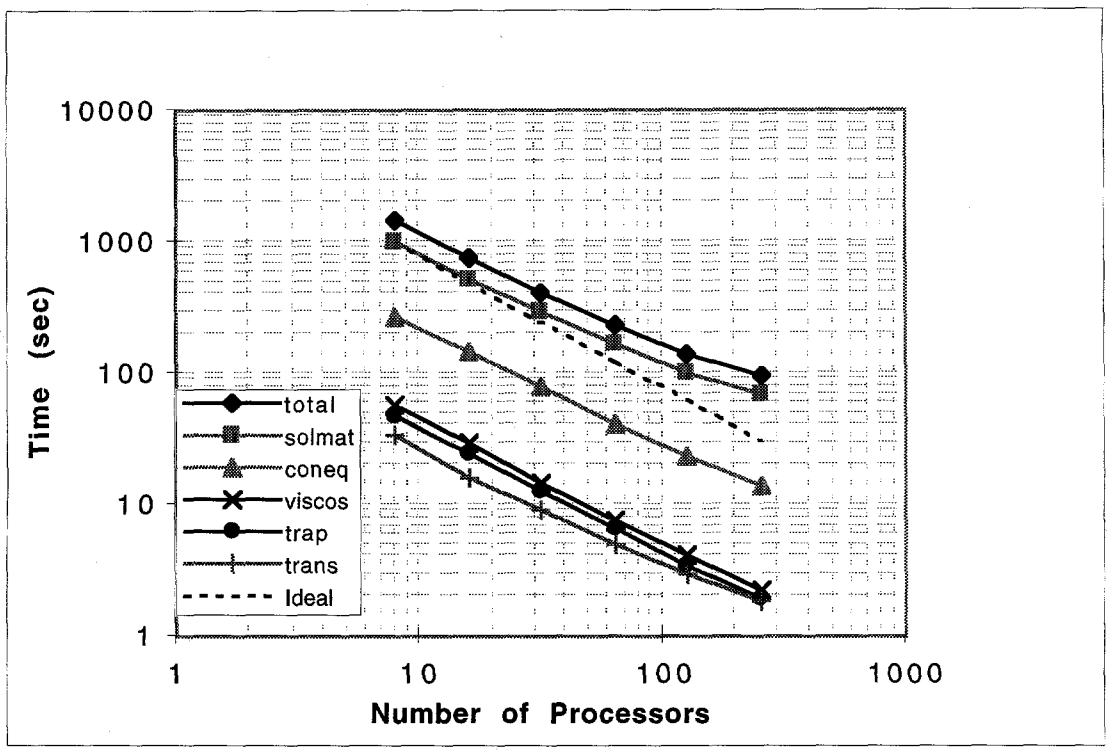

Figure 9. UTCHEM elapsed time for polymer flooding with 180,000 gridblocks on CRAY T3E.

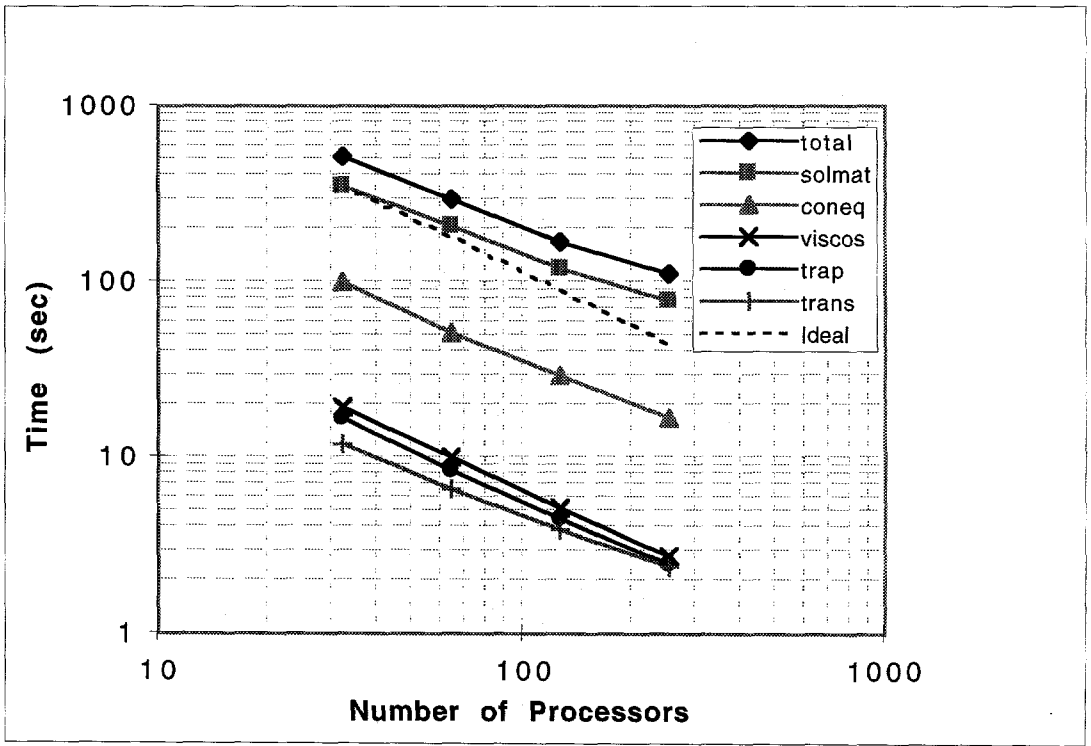

Figure 10. UTCHEM elapsed time for polymer flooding with 240,000 gridblocks on CRAY T3E.

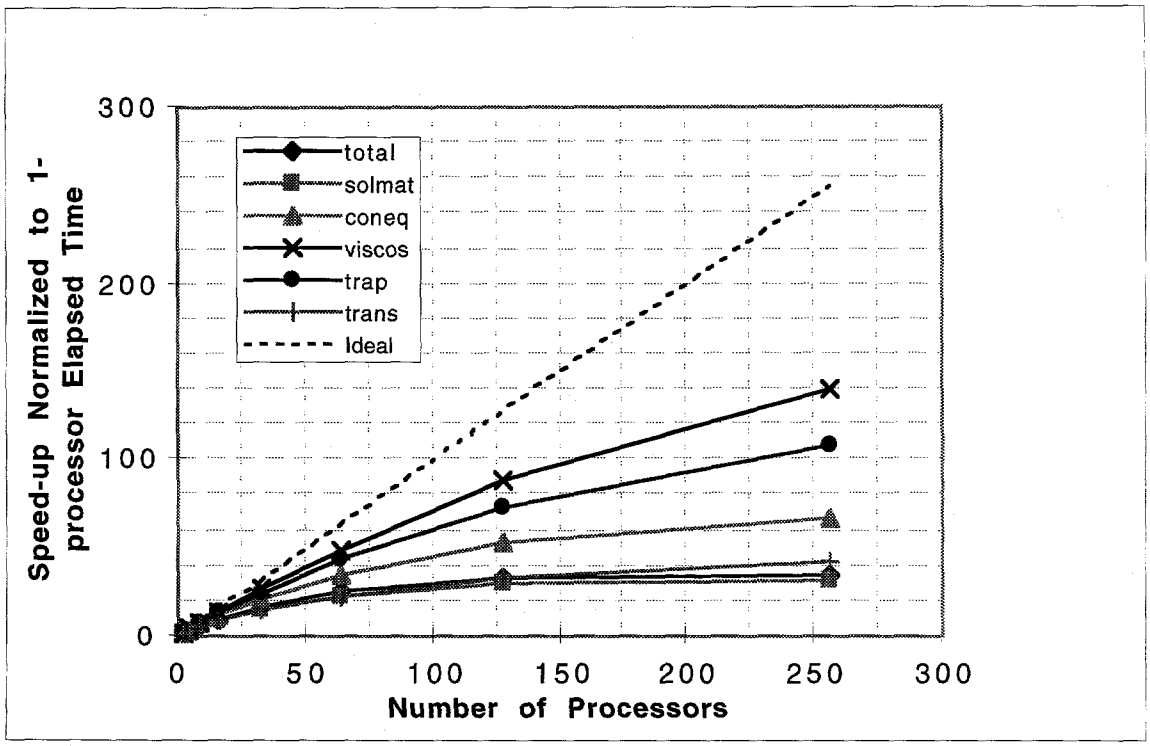

Figure 11. Speed-ups for polymer flooding with 30,000 gridblocks on CRAY T3E.

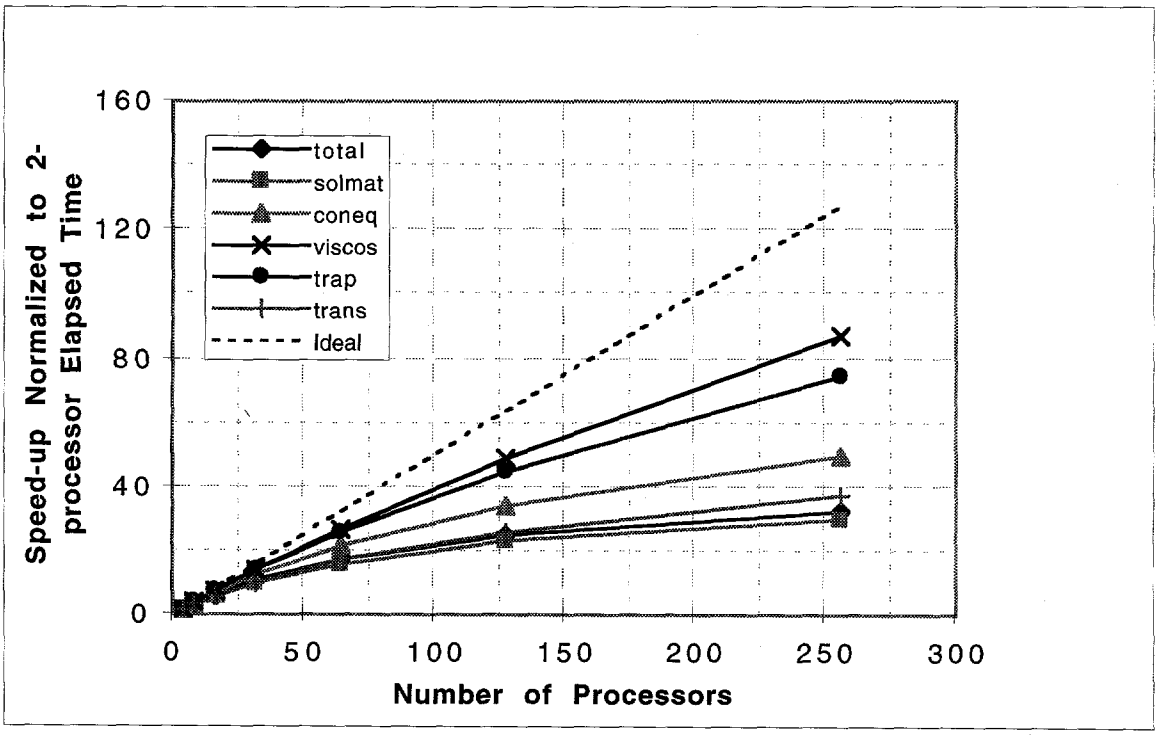

Figure 12. Speed-ups for polymer flooding with 60,000 gridblocks on CRAY T3E.

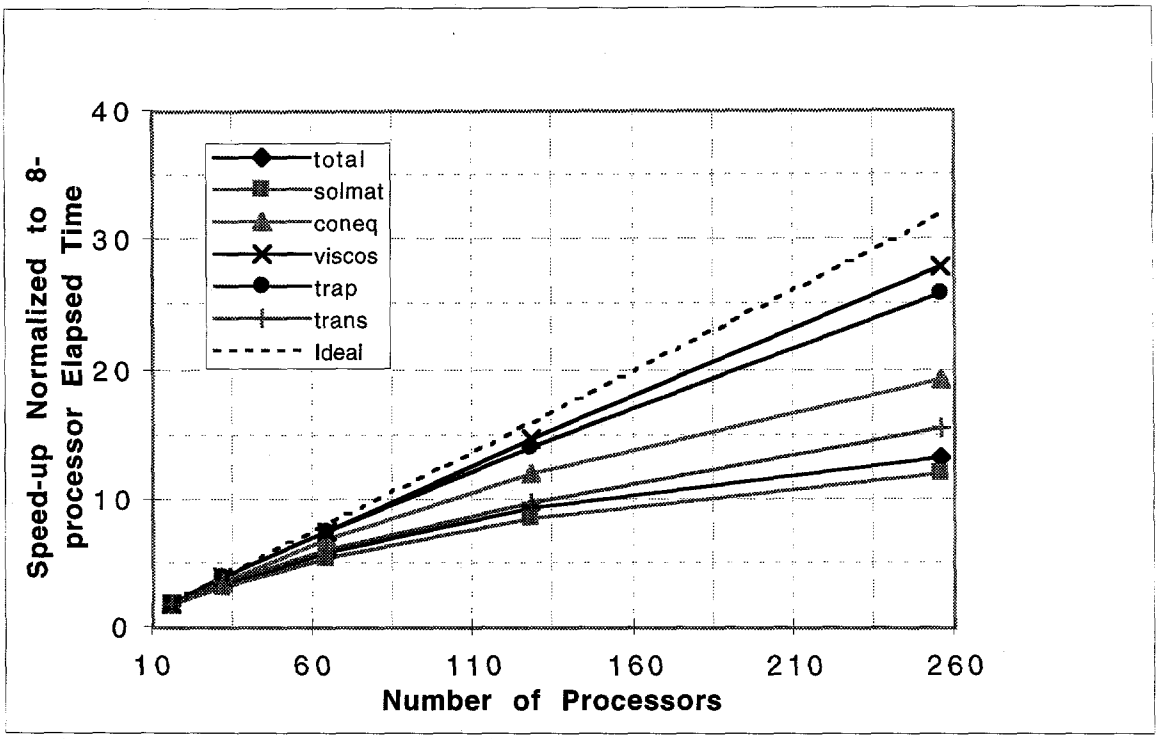

Figure 13. Speed-ups for polymer flooding with 120,000 gridblocks on CRAY T3E.

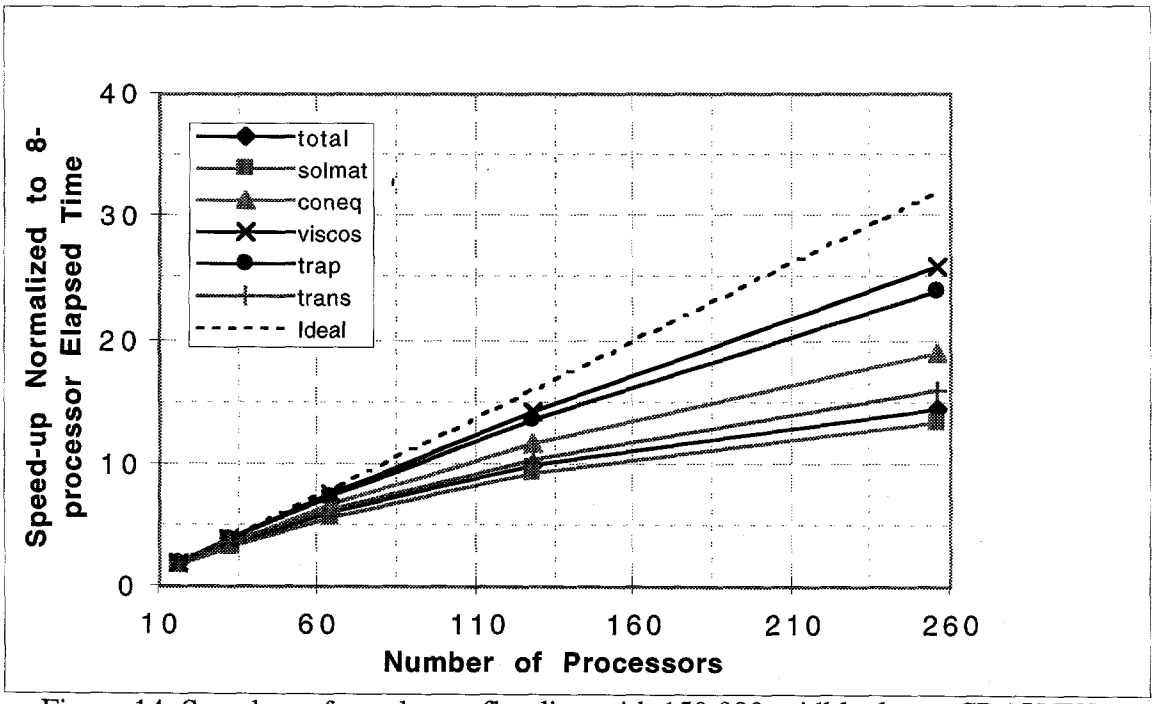

Figure 14. Speed-ups for polymer flooding with 150,000 gridblocks on CRAY T3E.

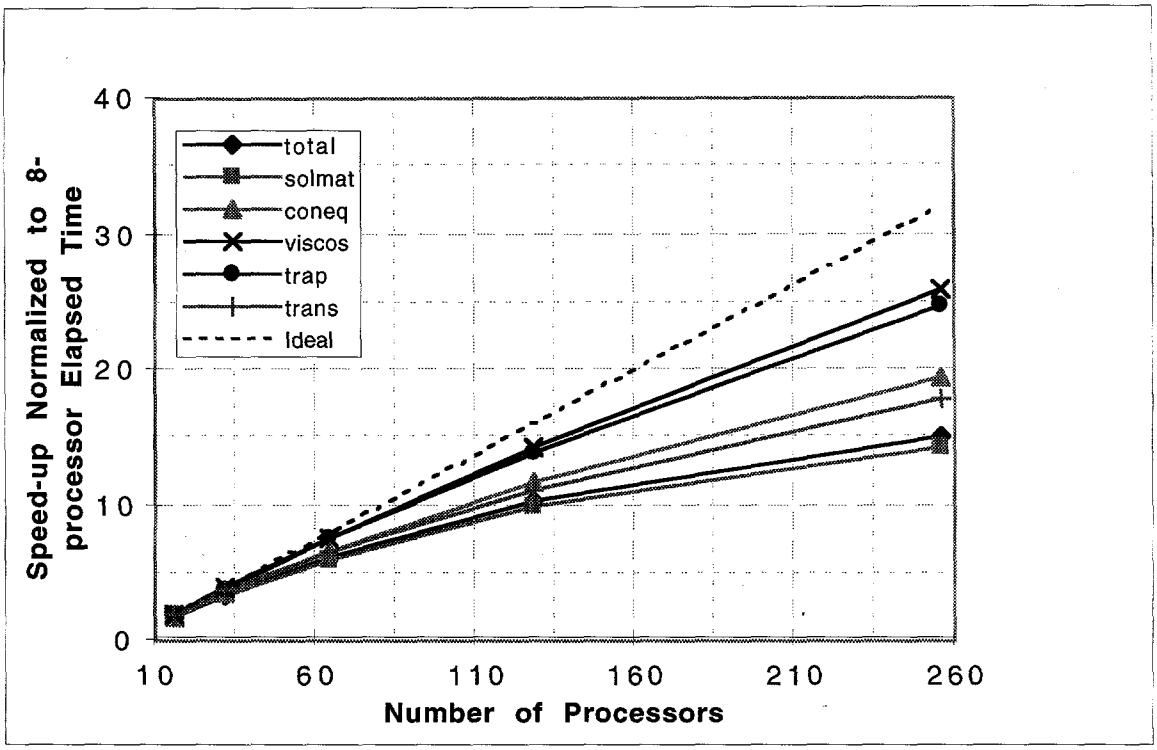

Figure 15. Speed-ups for polymer flooding with 180,000 gridblocks on CRAY T3E.

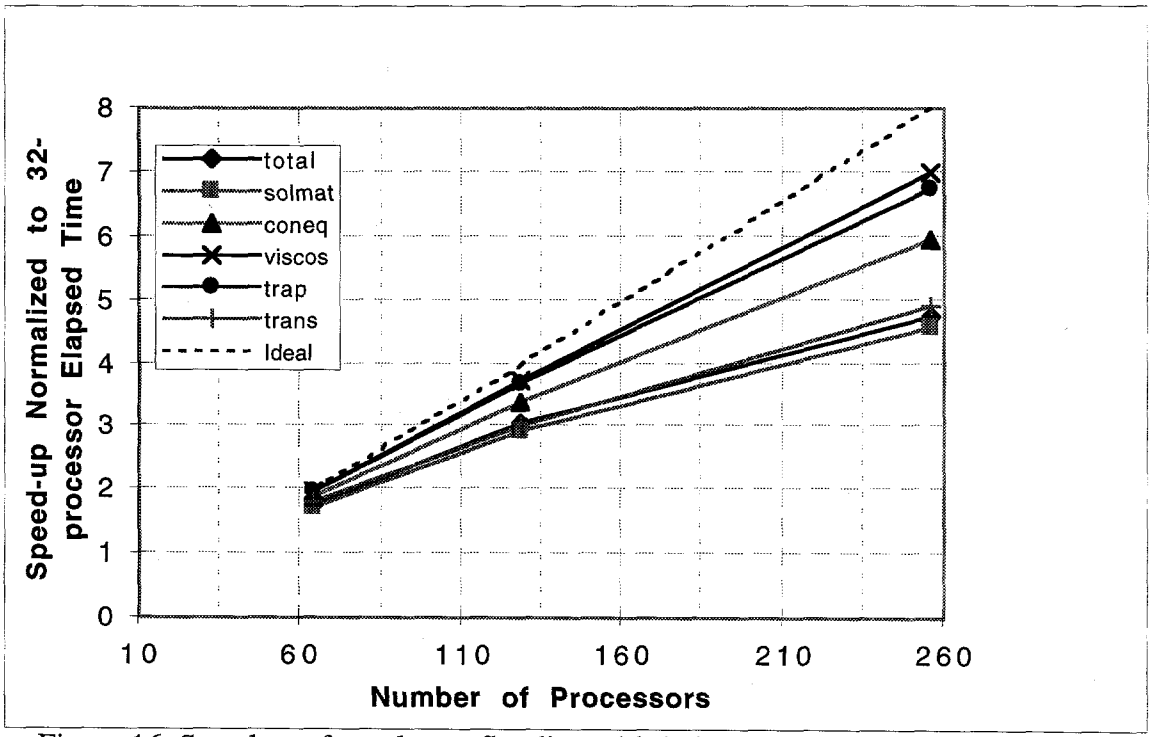

Figure 16. Speed-ups for polymer flooding with 240,000 gridblocks on CRAY T3E.

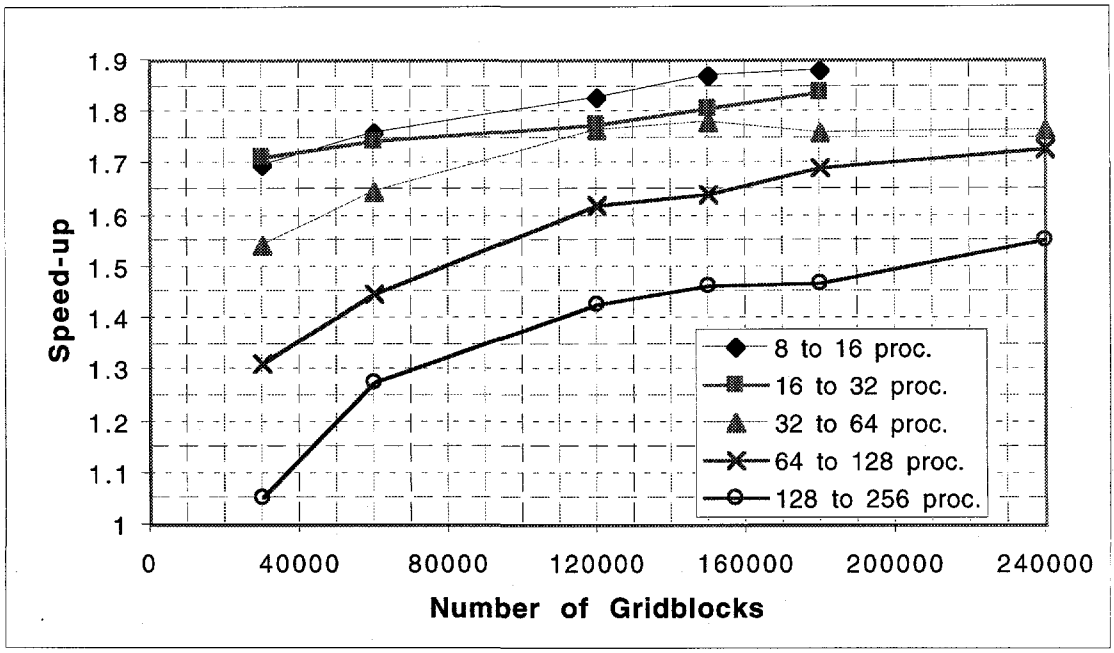

Figure 17. Comparison of speed-ups for different processors on CRAY T3E.

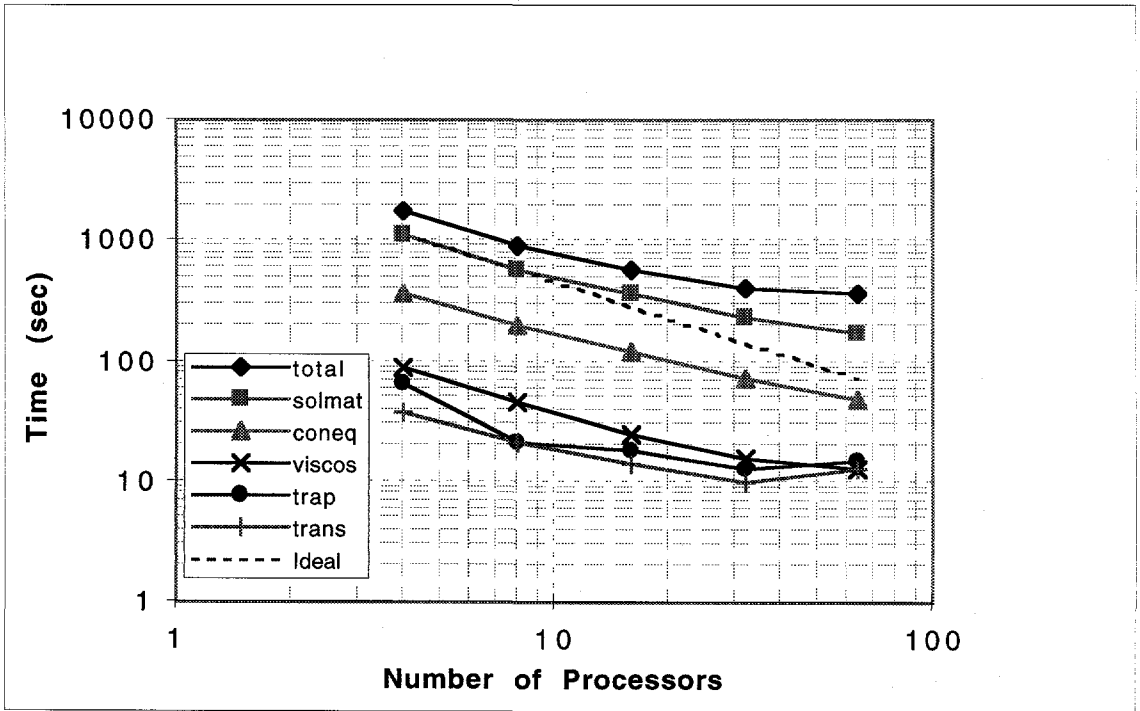

Figure 18. UTCHEM elapsed time for polymer flooding with 30,000 gridblocks on CRAY T3D.

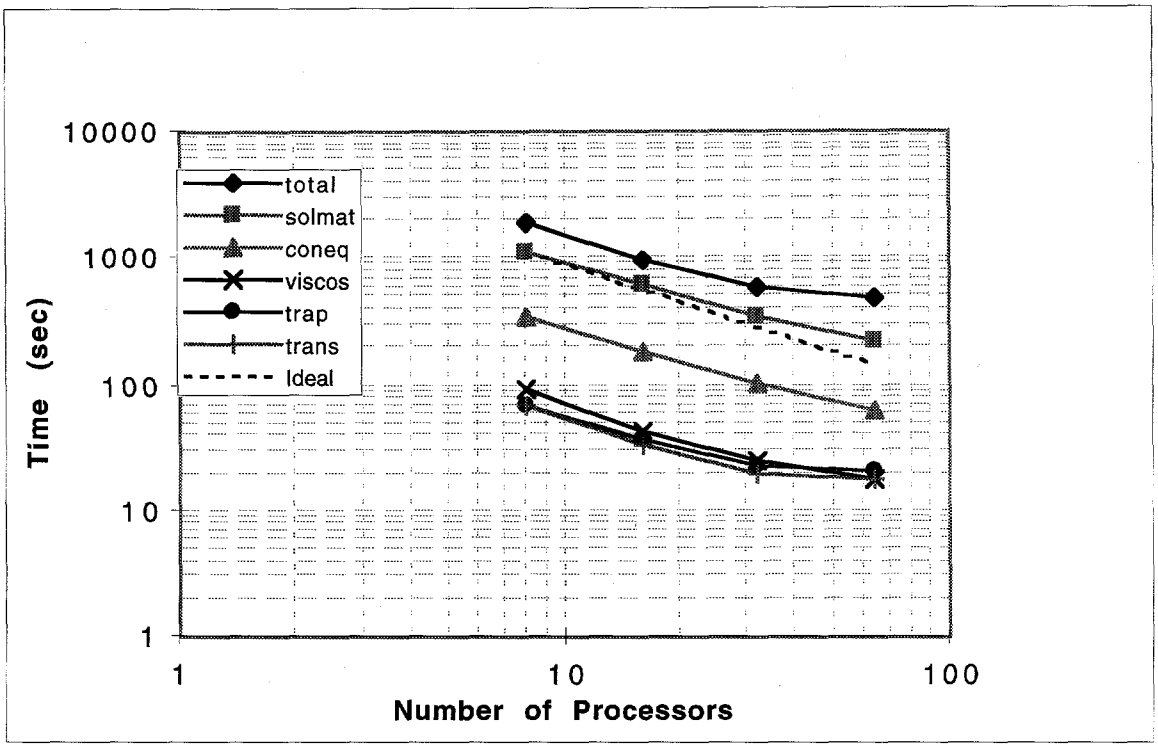

Figure 19. UTCHEM elapsed time for polymer flooding with 60,000 gridblocks on CRAY T3D.

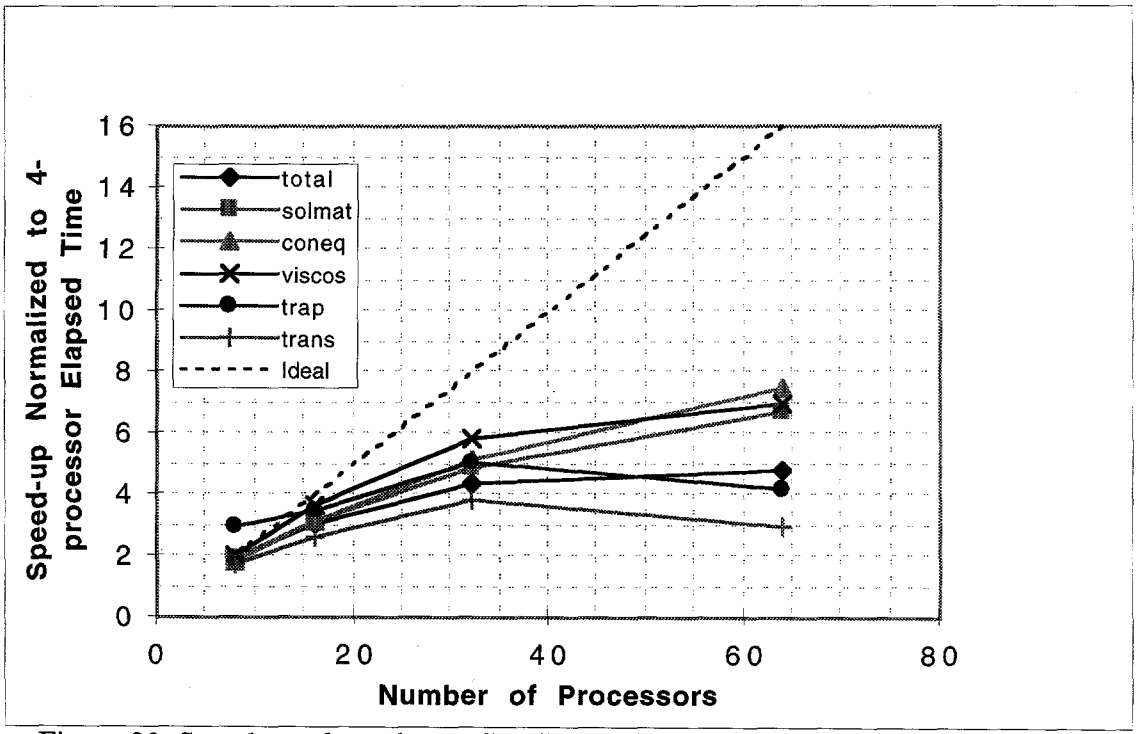

Figure 20. Speed-ups for polymer flooding with 30,000 gridblocks on CRAY T3D.

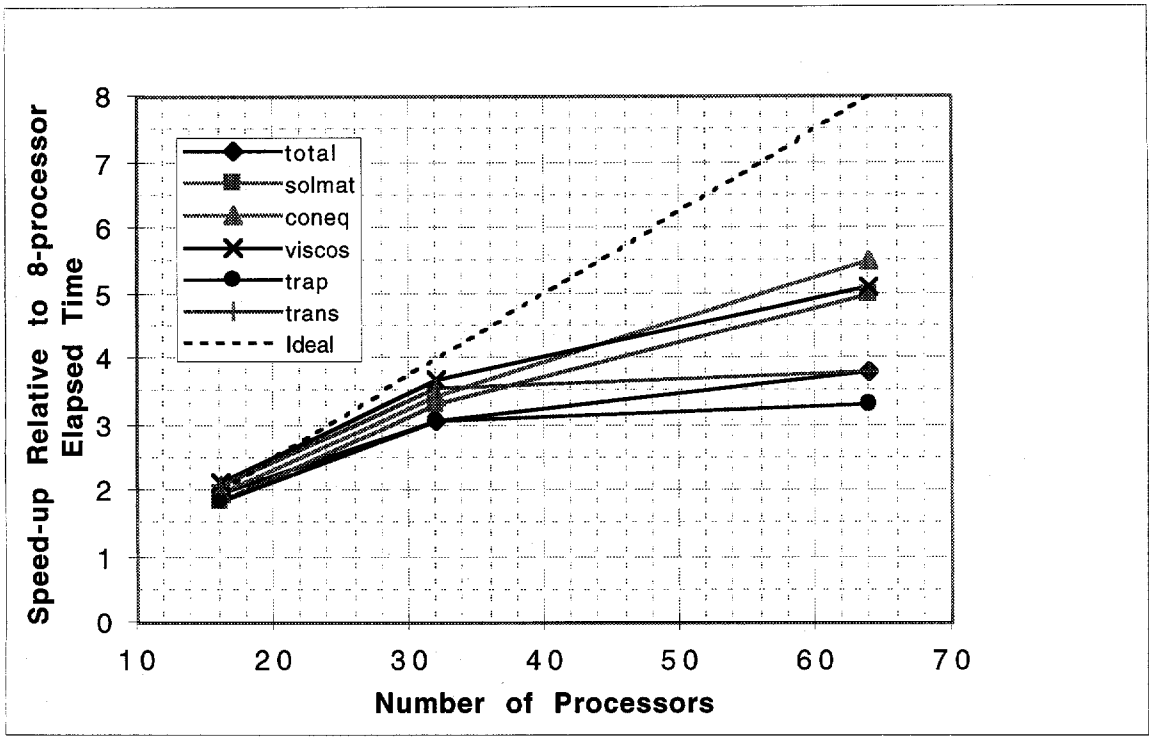

Figure 21. Speed-ups for polymer flooding with 60,000 gridblocks on CRAY T3D.