# SANDIA REPORT

SAND97–2551 • UC–705
Unlimited Release
Printed October 1997

# The Development and Performance of a Message-Passing Version of the PAGOSA Shock-Wave Physics Code

David R. Gardner, Courtenay T. Vaughan

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.



Sandia National Laboratories



SF2900Q(8-81)

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.



Available to DOE and DOE contractors from
Office of Scientific and Technical Information
PO Box 62
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from
National Technical Information Service
US Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A08
Microfiche copy: A01

SAND97-2551 Distribution
Unlimited Release Category UC-705
Printed October 1997

# The Development and Performance of a Message-Passing Version of the PAGOSA Shock-Wave Physics Code

David R. Gardner
Parallel Computational Sciences Department

Courtenay T. Vaughan
Parallel Computing Science Department
P.O. Box 5800
Sandia National Laboratories
Albuquerque, NM 87185-111

Prepared for
The Computational Mechanics and Material Modeling Technology Coordination Group of the Joint DoD/DOE Munitions Technology Development Program

## Abstract

A message-passing version of the PAGOSA shock-wave physics code has been developed at Sandia National Laboratories for multiple-instruction, multiple-data stream (MIMD) computers. PAGOSA is an explicit, Eulerian code for modeling the three-dimensional, high-speed hydrodynamic flow of fluids and the dynamic deformation of solids under high rates of strain. It was originally developed at Los Alamos National Laboratory for the single-instruction, multiple-data (SIMD) Connection Machine parallel computers. The performance of Sandia's message-passing version of PAGOSA has been measured on two MIMD machines, the nCUBE 2 and the Intel Paragon XP/S. No special efforts were made to optimize the code for either machine. The measured scaled speedup (computational time for a single computational node divided by the computational time per node for fixed computational load) and grind time (computational time per cell per time step) show that the MIMD PAGOSA code scales linearly with the number of computational nodes used on a variety of problems, including the simulation of shaped-charge jets perforating an oil well casing. Scaled parallel efficiencies for MIMD PAGOSA are greater than 0.70 when the available memory per node is filled (or nearly filled) on hundreds to a thousand or more computational nodes on these two machines, indicating that the code scales very well. Thus good parallel performance can be achieved for complex and realistic applications when they are first implemented on MIMD parallel computers.

## Acknowledgments

The PAGOSA code was developed at Los Alamos National Laboratory under the direction of Dr. J. W. Hopson. We thank Dr. Hopson for making the data-parallel version of PAGOSA available to us, and we thank Dr. D. B. Kothe of Los Alamos National Laboratory for providing technical advice concerning its structure. We thank the Advanced Computing Laboratory of Los Alamos National Laboratory, Los Alamos, NM 87545, for providing computing resources necessary to complete this work.

This work was supported under the Joint DoD/DOE Munitions Technology Development Program, and sponsored by the Office of Munitions of the Secretary of Defense.

We thank T. Mack Stallcup of Intel Corporation for providing technical information about the Paragon computer. We thank Martin W. Lewitt, formerly of nCUBE Corporation, for technical information about the nCUBE 2 computer.

UNIX® is a trademark of AT&T. CM-2, and CM-5 are trademarks of Thinking Machines Corporation. iPSC®, i860, and Paragon™ are trademarks of Intel Corporation.

This work was performed at Sandia National Laboratories supported by the U.S. Department of Energy under contract number DE-AC04-94AL85000. We thank the Massively Parallel Computing Research Laboratory at Sandia for providing computing resources necessary to complete this work.

# Table of Contents

# List of Figures

Intentionally Blank Page

# List of Tables

# The Development and Performance of a Message-Passing Version of the PAGOSA Shock-Wave Physics Code

## Introduction

An important class of shock-wave physics problems is characterized by large material deformations. These problems involve penetration, perforation, fragmentation, high-explosive initiation and detonation, and hypervelocity impact. These phenomena arise, for example, in armor/antiarmor research and development, the design of impact shielding for spacecraft, the modeling of lithotripsy for the disintegration of kidney stones using high-frequency sound waves, and hypervelocity impact problems. The most important of such problems are intrinsically three-dimensional and involve complex interactions of exotic materials, including alloys, ceramics and glasses, geological materials (*e.g.*, rock, sand, or soil), and energetic materials (*e.g.*, chemical high explosives).

Multidimensional computer codes with sophisticated material models are required to model this class of shock-wave physics problems realistically. The codes must model the multiphase (solid-liquid-vapor), strength, fracture, and high-explosive detonation properties of materials. Three-dimensional simulations may require millions of computational cells to adequately model the physical phenomena and the interactions of complex systems of components. Many scientists and engineers currently use Eulerian shock physics codes such as Sandia National Laboratories' CTH code [15][18] or Los Alamos National Laboratory's MESA [17] codes to model such problems [3].

CTH and MESA are serial codes which run on Cray vector supercomputers and on workstations. Owing to the expense of high-speed memory, vector supercomputers do not have enough memory to model problems which require more than a few million computational cells. Many problems of interest require tens of millions of cells. Even these inadequately resolved problems often require tens or hundreds of CPU hours to complete. Traditional serial vector supercomputers are too slow and have too little memory to calculate many important weapon safety problems, or to calculate complex design problems, such as the effects of materials selection and design parameters on the performance of modern armor.

Parallel shock physics codes running on current-generation massively parallel computers are beginning to provide the high resolution and short turnaround time required for these shock-wave physics problems. Several years ago, work at Sandia demonstrated that massively parallel computers running parallel versions of the CTH and MESA codes were highly competitive with serial vector supercomputers such as a Cray Y-MP [10][11] [12] [19]. Current-generation parallel computers, such as the Paragon XP/S, are demonstrating even better performance, both in terms of problem size and speed [8].

In this report we describe the development of a three-dimensional, multimaterial version of the PAGOSA shock-wave physics code for multiple-instruction, multiple-data (MIMD) parallel computers, and present the measured performance of the code on two different parallel computers, the nCUBE 2 and the Intel Paragon XP/S. The nCUBE 2 is a distributed-memory MIMD parallel computer with a hypercube communications topology. The Paragon XP/S is a distributed-memory MIMD parallel computer with a two-dimensional mesh communications topology.

The original data-parallel PAGOSA code was developed at Los Alamos National Laboratory in CM Fortran for the Connection Machine-2 (CM-2) and the Connection Machine-5 (CM-5)[16].

In earlier work we developed a three-dimensional, single-material, ideal-gas version of PAGOSA for MIMD parallel computers (specifically, the Cray Y-MP, the nCUBE 2, the Intel iPSC/860, and the CM-5) by translating it to Fortran 77 and adding routines for domain decomposition and interprocessor communication [9]. The excellent performance of this code [10][11] and the parallel CTH code (PCTH) [7][8][12][13][19] on a variety of MIMD parallel computers spurred the continuing development of parallel shock-wave physics codes, and, in particular, an MIMD version of the three-dimensional, multimaterial PAGOSA code.

By reporting the performance of a single applications code on a variety of parallel machines, we can provide an indication of the performance that may be attained on current-generation parallel computers. The performance results presented here are indicative of what one might achieve in first implementing a complex application code on a distributed-memory, message-passing parallel computer. We have made no attempt to optimize the code for any particular machine; all the machines are running essentially the same code. The performance results reported here are lower bounds for each architecture, because, given sufficient resources, most codes can be optimized for a given architecture.

The performance data we present in this report represent only one aspect of the performance of each computer. While the data address fundamental issues of computational speeds, other issues must also be considered when evaluating parallel computers, including the ease of sharing the machine among multiple users, the functionality of the operating system, the availability of graphical output devices and the ease of their use, and the machine acquisition and maintenance costs. Neither of the machines examined in this report should be accepted or rejected solely on the basis of the data presented in this report.

In the remainder of this report we discuss issues in parallel computing and measuring the performance of parallel computer codes. We then discuss the development of the MIMD PAGOSA code. We describe the test problems and the conditions used in our study and then present the performance results. Finally we compare the performance of MIMD PAGOSA on the two MIMD computers and present our conclusions.

## Issues in Parallel Computing

In this section we discuss some of the issues involved in parallel computing. We define data-parallel computing and message-passing computing, and general features of how these computing paradigms are implemented on the computers used in this study. Then we review what has been learned prior to this study about the performance of shock-wave physics codes on parallel computers.

Historically, two parallel computing architectures have been popular: the *Single-Instruction, Multiple-Data,* or SIMD, architecture, and the *Multiple-Instruction, Multiple-Data,* or MIMD, architecture. These architectures are visible to the user as a data-parallel programming paradigm or a message-passing paradigm, respectively. Often terms used to describe the architectures and the programming paradigms are used interchangeably.

In the SIMD architecture, a large number (usually thousands) of small processors perform operations on data under the control of a master processor. Each processor executes the same instruction simultaneously on the data to which it has access. SIMD computers may be thought of as large array processors. The SIMD approach is more general than it might first seem, because the result of the operation executed on the data in a processor may depend on the data itself through local flag variables. An example of an SIMD machine was the Connection Machine-2, manufactured by Thinking Machines Corporation. The SIMD programming paradigm focuses on and exploits parallelism in the data, but not in the instructions executed by the processors.

In the MIMD architecture, a large number (usually tens or hundreds) of more sophisticated processors (which we will refer to as *nodes*[1]) execute the same or different instructions on the data to which they have access. For distributed memory computers, the work performed by the nodes is coordinated via explicit passing of messages from one node to another. In shared memory computers, the work performed by the nodes may also be coordinated through the shared memory. Each node has its own operating system and its own copy of the instruction set. Although each node may be executing the same instruction at the same time, the more common mode of operation is for each node to execute instructions independently of the others, and then synchronize its execution with other nodes at various times via the passing of messages, such as the global determination of a time step. This latter mode of operation is called *loosely synchronous*. It is common for each node in an MIMD computer partition to be executing the same program in loosely synchronous mode, but more generally the nodes can be executing entirely different

1. Some MIMD processing elements are composed of more than one processor. For example, the computational processor in the Paragon XP/S is composed of an i860 XP RISC processor for computation and an additional i860 XP processor for communication. In most circumstances we will refer to a processing element in a parallel computer as a *processor* or a *node*. In those circumstances where it is important to distinguish between an a processing element and the processors which comprise it, we will refer to the former strictly as a node.

programs. For example, in an eight-node partition, four nodes might be devoted to performing an armor penetration simulation, while the other four might be devoted to forming graphical images of the simulation in parallel with the computation. Examples of MIMD computers are the nCUBE 2, manufactured by nCUBE Corporation, the Intel Paragon XP/S, and the Cray T3D. The message-passing programming paradigm exploits parallelism in both the data and the instructions executed by the nodes.

Both SIMD and MIMD computers often employ a front-end host to communicate information (instructions and data) to the parallel assembly of processors, and to collect data from the processors for transmission to the user. In the SIMD architecture, the host acts as a master processor to the slave processors. The host issues all the instructions, while the processors execute those instructions synchronously.

In the MIMD architecture, the nodes can function much more independently of the host, and the division of labor between the host and the nodes can vary widely. In particular, the work allocated to the front-end host can vary from minimal to a significant amount, depending on the application. Minimal work includes allocating and opening the requested assembly of nodes (*e.g.*, a hypercube in a computer with a hypercube communication topology), broadcasting the appropriate instruction set to each node, and closing and deallocating the assembly of nodes at the completion of the job. In addition, the host might broadcast the user input to the nodes or postprocess some of the data from the nodes for transmission to the user. In general, at least two distinct instruction sets are required, one for the host, and one (or more) for the nodes. In the case where the host performs more than the minimal work required, this programming model is termed the *host-node* model; such a model is explicitly required on the Cray Y-MP (when run as a distributed-memory message-passing computer).

On many MIMD computers it is possible to put processing that might be performed on the host on one or more of the nodes, allowing the host to perform only the minimal work required. On these machines the manufacturer may supply a generic host code which performs the minimal work and relieves the user of the burden of maintaining both a host code and a node code. The generic host code is called xnc on the nCUBE 2. A script called pexec that functions as a generic host code is commonly used on the Intel Paragon; pexec allocates a collection of nodes, loads and runs the application code(s), and finally deallocates the nodes. This programming model is termed the *node-only* model.

In general, SIMD computers are easier to program than MIMD computers because the interprocessor communication in SIMD computers is hidden from the programmer by the operating system and hardware. SIMD computers can often run applications that are strongly data-parallel faster than MIMD computers. In general, however, MIMD computers provide a more flexible computing environment, because different nodes can work on different subtasks simultaneously. Even on MIMD machines, much of the inter-node communication in MIMD computers may be hidden from the programmer by the

operating system and by the use of data-parallel languages such as Fortran 90. Overall, an MIMD computer is a more general-purpose computer than an SIMD computer.[2]

Earlier work at Sandia and at Los Alamos National Laboratory demonstrated that both SIMD and MIMD computers can be programmed effectively for shock wave physics calculations [12][16][19]. Our previous work with a single-material, ideal-gas version of PAGOSA demonstrated that MIMD computers consistently outperformed SIMD computers on large problems for machines of comparable cost [10][11]. Currently all major parallel computer manufacturers have adopted the MIMD architecture, which we interpret as evidence that the MIMD architecture is more suitable for general applications than the SIMD architecture.

## Parallel Code Performance Measurements

In this section we discuss issues in measuring the performance of codes on parallel computers. We consider computational rate, memory size, and scalability as three essential metrics of parallel computer performance.

The performance of parallel computers is commonly measured using several metrics. The peak theoretical speed is often cited. The results from the LINPACK benchmarks [6] or the NAS Parallel benchmarks [1] are more indicative of the performance which may be achieved by applications codes.

The LINPACK benchmark codes perform a factorization of a dense matrix A into a lower triangular matrix L and an upper triangular matrix U, such that A = LU. This factorization, called an *LU factorization*, is used in solving dense linear systems of equations of the form Ax = b. The benchmark uses standard LINPACK [5] routines in full-precision (64-bit) arithmetic in a Fortran 77 environment (there is also a set of benchmarks for the C programming language) to perform an LU factorization. The benchmark consists of several tests. The first is for a matrix of order 100 using a prescribed Fortran 77 program. The second test is for a matrix of order 1000 using any algorithm, but with a prescribed driver to set up the problem to be solved and to ensure consistent solution accuracy. The third test is to factor the matrix of largest possible order using any code on a parallel computer. The full LINPACK benchmark results for a computer consist of the time required to complete each test and also include the theoretical peak speed of the computer, the upper bound on machine performance. The most commonly cited LINPACK benchmark results provide an achievable upper bound for speed on problems involving the solution of dense linear systems by LU factorization; achieving the benchmark results often requires the use of special machine configurations and highly

---

2. The CM-5 does not fit neatly into the framework of the discussion in this paragraph because, while its underlying architecture is MIMD, the user can employ either a data-parallel or message-passing programming paradigm for applications. The CM-5 demonstrates that the programming paradigm can be independent of the underlying computer architecture.

optimized assembly code or other resources not normally available to the engineering analyst.

The NAS Parallel benchmark set is a collection of eight problems designed to indicate the performance of parallel supercomputers. They consist of five kernels, each emphasizing a particular type of numerical computation (*e.g.*, fast Fourier transforms), and three simulated computational fluid dynamics applications. The benchmarks are specified functionally, independent of the details of implementation, with specified operation counts. While the times required to complete the NAS Parallel benchmark tests provide a more realistic assessment of the performance of a parallel computer, the tests are still highly idealized compared to real applications codes.

Computational rate and problem size are distinct though related aspects of computer performance. Analysts usually want to perform simulations as quickly as possible; for example, when conducting a parameter study. If a problem is sufficiently large, a parallel computer may be able to run it faster than the fastest vector supercomputers even though the parallel computer has slower processors [10][11]. Vector supercomputers must use expensive high-speed memory to achieve high computational rates, and the cost of that memory places a practical limit on the memory size of the computer, and hence on the size of the simulations which can be performed with it. Distributed-memory parallel computers use slower, less expensive memory, and hence, for the same cost as a vector supercomputer, a parallel computer with much larger total memory can be acquired. Thus it is practical for a distributed-memory parallel computer to run much larger simulations than can be performed on existing vector supercomputers. From this point of view, the issue is not so much computational rate as memory size: if a computer does not have enough memory to perform the simulation, it does not matter how fast it is. Both computational rate and memory size should be considered when measuring the performance of a computer.

Elsewhere [8] we have demonstrated that while an MIMD parallel computer may provide a higher computational rate than a vector supercomputer, it must also have sufficient memory capacity to provide equal or greater resolution. A simulation with less resolution obtained more quickly may not be as useful to the analyst as a simulation with greater resolution. We have also demonstrated that when a parallel computer has enough memory, simulations of greater resolution can be obtained in less time than with a serial vector supercomputer.

In order to achieve a high execution rate on a parallel computer, both the parallel computer and the application code must also be scalable. By scalability of the computer, we mean that the time to send a zero-length message across the largest dimension of the computer increases no more than linearly with the number of nodes. By “largest dimension” we mean the maximum over all the nodes of the number of nodes a message must pass through in order to travel from one node to another while taking the shortest permissible path, including the sending and receiving nodes. For example, a hypercube

with $2^N$ nodes has dimension $N$. A `Paragon` with a mesh of 6 columns of 16 nodes has dimension 21, since messages are passed along a row and then down a column.

By *scalability* of the application code we mean that the execution speed of the code running a specific problem on a scalable parallel computer increases linearly with the number of nodes when the computational load per node is fixed [14]. Both the parallel computer (hardware and operating system) and the application code must be scalable in order to achieve high execution rates.

Three performance metrics are commonly used for application codes on parallel computers: fixed-size speedup, scaled speedup and parallel scaled efficiency. If the scaled speedup, or, equivalently, the parallel efficiency, varies linearly with the number of nodes, then the application code is scalable.

We first define the *speedup*, $S(P,N)$ to be the ratio of the time to solve a problem of size $N$ on one node, $T_1(N)$, to the time required to solve the same problem on $P$ nodes, $T_P(N)$:

$$S(P, N) = T_1(N) / T_P(N)$$

This defines a surface in three dimensions; an example is shown in Figure 1.

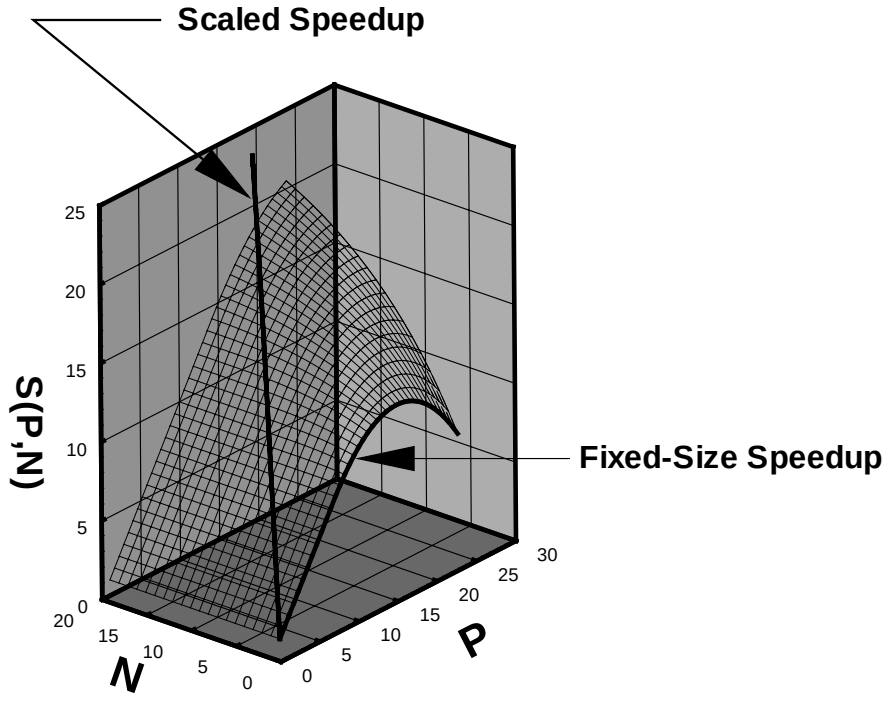



Figure 1. Illustration of the speedup surface, the fixed-size speedup curve, and the scaled speedup curve.

The *fixed-size speedup* $S_f$ is the ratio of the time required to solve a problem on a single node to the time required to solve the same problem on $P$ nodes, when the problem size $N$ is fixed. If the problem size is fixed, the locus of points on the speedup surface generated as the number of processors is varied is the fixed-size speedup curve. A fixed-size speedup curve is marked on the speedup surface (Figure 2). If we are interested in solving very large problems which will not fit on a single node (as is often the case), then fixed-size speedup is not a good measure of performance. However, engineers are often interested in solving a problem of fixed size as quickly as possible, and hence in the maximum of the fixed-size speedup curve. In this circumstance the fixed-size speedup is a useful measure of performance.

In contrast to the fixed-size speedup, the *scaled speedup* $S_s$ is the ratio of the time required to solve a problem of size $PN$ on a single node, $T_1(PN)$, to the time required to solve the problem of size $PN$ on $P$ nodes with a subproblem of size $N$ on each node, $T_P(PN)$, when the work per node is fixed [14]. Thus the problem size increases with the number of computational nodes. The scaled speedup can be calculated directly, as long as the problem of size $PN$ will fit on a single node, from

$$S_s(P) = S(P, PN) = \frac{T_1(PN)}{T_P(PN)}.$$

The locus of points on the speedup surface generated as the number of processors is varied and the problem size is increased in proportion to the number of processors is the scaled speedup curve. A typical scaled speedup curve is marked on the speedup surface (Figure 1). The projections of the fixed-size and scaled speedup curves on the $P$-$S$ plane are shown in Figure 2 to illustrate the difference between them.

When the problem of size $PN$ will no longer fit on a single node, $T_1(PN)$ must be estimated. One way to estimate the time $T_1(PN)$ is to extrapolate it from the behavior of $T_1(PN)$ on a single node as $PN$ increases [9]. For large problems, this may require extrapolation over several orders of magnitude, which introduces uncertainty into the validity of the resultant speedup. In this report we estimate the time $T_1(PN)$ by $PT_1(N)$. This represents the time required by a single node to perform the necessary calculations on each subdomain serially, assuming that no time is required to swap the subdomains in memory and assuming sufficient memory to hold all the subdomains. It is thus the shortest time that a single node could perform the same calculation as the parallel computer. Making this estimate is straightforward for an explicit code like PAGOSA; for codes with implicit components, however, one must ensure that the same computational work is done by the single node in processing all the subdomains as is done by the parallel computer. Here we calculate the scaled speedup $S_s(P)$ from the ratio of the product of the time required to solve the problem of size $N$ on a single node, $T_1(N)$ and the number of nodes, $P$, to the time taken to solve the problem of size $PN$ on $P$ nodes, $T_P(PN)$:

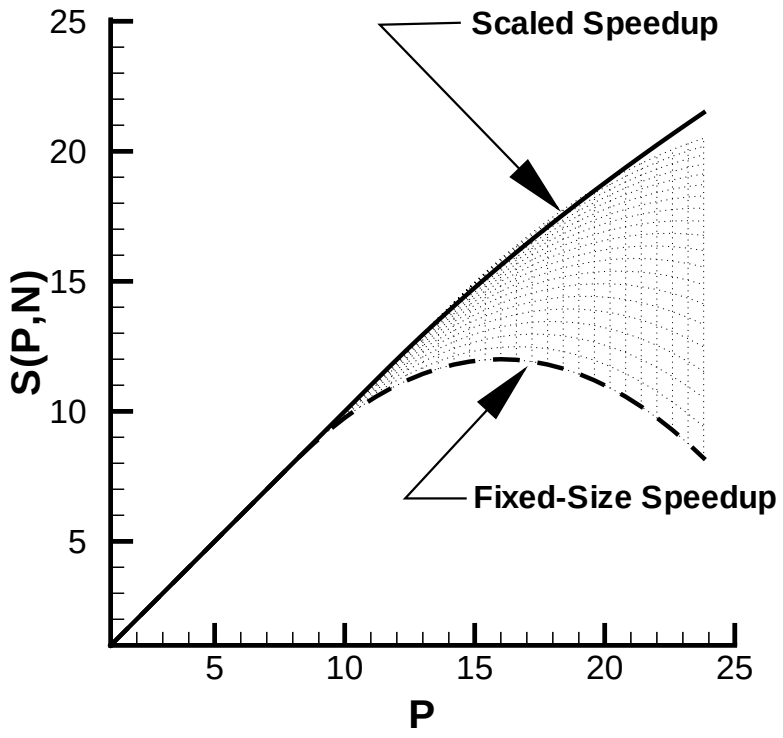



Figure 2. The fixed-size speedup and scaled speedup curves projected on the *P-S* plane.

For many scientific and engineering simulations (such as the test problems presented later and simulated with MIMD PAGOSA) the ratio $T_1(P)/T_N(PN)$ becomes constant when $P$ is sufficiently large, and $S_s(P)$ varies directly with $P$ [10][11], that is, the simulations are scalable.

The *parallel scaled efficiency* $\varepsilon$ is the scaled speedup divided by the number of computational nodes:

$$\varepsilon = S_s(P)/N = T_1(P)/T_N(PN).$$

The closer the parallel scaled efficiency is to one, the more efficient the parallel performance of the code is. The parallel scaled efficiency will always be less than one, owing to algorithmic, communication, or load-balancing overhead.

The *grind time* is a useful measure of the computational rate of a mesh-based, time-marching computer code, such as PAGOSA. The grind time, $t_{grind}$, is the execution time for the code calculating a given problem divided by the product of the number of time steps and the number of computational cells:

where $T_P(PN)$ is the execution time on $P$ nodes for a problem of $PN$ computational cells run for $n$ time steps. The grind time depends on the number of cells and the number of nodes, and also on the specific simulation. For a given code solving the same problem on different computers, the grind time indicates the performance of the code on that computer and is a useful metric for computer performance comparisons.

Of special interest in indicating parallel computer performance are the maximum problem size which can be run and the execution time (or equivalently, the grind time) on the maximum problem size. As discussed above, these are distinct though related measures of performance. These measures are both very problem-dependent, and so must be measured for a variety of problems for a given application code in order to adequately represent the performance of a given computer.

The relative performance of parallel computers can be assessed from measured values of scaled speedup, parallel scaled efficiency, grind time, maximum problem size and execution rate on the maximum problem size. The scaled speedup and parallel scaled efficiency both measure the scalability of a parallel code on a specific parallel computer. The grind time and maximum problem size data measure the absolute performance of the parallel code on the parallel computer.

## Development of the MIMD PAGOSA Code

In this section we describe the features of the PAGOSA 5.5 shock-wave physics code and the process of developing the MIMD PAGOSA 5.5 code from the SIMD version.

### Features of PAGOSA 5.5

PAGOSA is an explicit, three-dimensional, multimaterial shock wave physics code which has been developed at Los Alamos National Laboratory for the CM-2 and CM-5 massively parallel computers in CM Fortran, a variant of the Fortran 90 programming language [16]. PAGOSA is designed to model problems involving high-speed hydrodynamic flow and the dynamic deformation of solid materials. The core algorithms in PAGOSA were inherited from the MESA code [17], but some of the algorithms in PAGOSA have been rewritten to take advantage of the SIMD parallel architecture of the CM-2 and the data-parallel programming paradigm of the CM-5. MESA and PAGOSA were both developed specifically for three-dimensional armor/antiarmor simulations, although they are now used for a broad range of applications, and both codes include a variety of equations of state and material strength models.

The numerical algorithms used in PAGOSA solve the equations of conservation of mass, momentum and energy in an explicit, Eulerian finite difference formulation on a three-dimensional Cartesian mesh. A staggered mesh is used in which density and pressure are evaluated at the cell centers, and the velocities are evaluated at the cell vertices.

The solution at each time step is calculated in two phases, a Lagrangian phase and an advection phase. During the Lagrangian phase, the Lagrangian equations of motion are solved to obtain the values of the variables corresponding to a fluid element which has moved and distorted relative to the fixed Cartesian mesh, using a second-order accurate predictor-corrector scheme for the time integration.

During the advection phase, the updated variables at the original, fixed cell centers and vertices are calculated. The advection equations are solved using an operator-splitting scheme in which the advection operator is split into components along the three orthogonal mesh directions and the fluxes of mass, energy, momentum and stress through cell faces are calculated for each direction. Corrections for cross terms are not explicitly included, but approximate corrections are made implicitly by reversing the order of the advection directions in alternate timesteps. This tends to remove any directional bias introduced by the operator splitting. In each coordinate direction an upwind or donor-cell scheme is used to determine the fluxes of cell-centered quantities through the faces of a cell; a similar scheme is used for determining the momentum flux, but is based on a vertex-centered cell. A third-order accurate van Leer limiting scheme is used to correct the first-order accurate donor-cell fluxes. This makes it possible to maintain steep gradients of advected quantities without introducing non-physical oscillations.

Ideal gas, Mie-Grüneisen, polynomial, and Jones-Wilkins-Lee (JWL) equations of state, the von Mises elastic, perfectly plastic yield stress model, and a programmed burn model for high explosives have been implemented in the production version of PAGOSA [16].

## Development of MIMD PAGOSA 5.5 from SIMD PAGOSA 5.5

Conceptually, creating an MIMD version of PAGOSA from the SIMD version involves dividing the global computational domain into a collection of subdomains, with each subdomain assigned to a single node, and then adding the necessary communications between neighboring nodes to reproduce the global computational domain. When the global computational domain is divided into subdomains, each subdomain is surrounded by *ghost* cells, which are used for communicating results between neighboring nodes. This subdivision is illustrated for a two-dimensional domain in Figure 3. This process of dividing the global computational domain into subdomains, assigning a subdomain to a node, and implementing the necessary inter-node communications to reproduce the global domain is called *domain decomposition*.

More specifically, following the process we used in developing an MIMD version of the single-material, ideal-gas version of PAGOSA [9], we created an MIMD version of PAGOSA from the SIMD code (specifically, PAGOSA 5.5) by translating the CM Fortran to Fortran 77, adding routines to divide the global computational domain into subdomains and assign each to a node, and adding routines for inter-node communication.

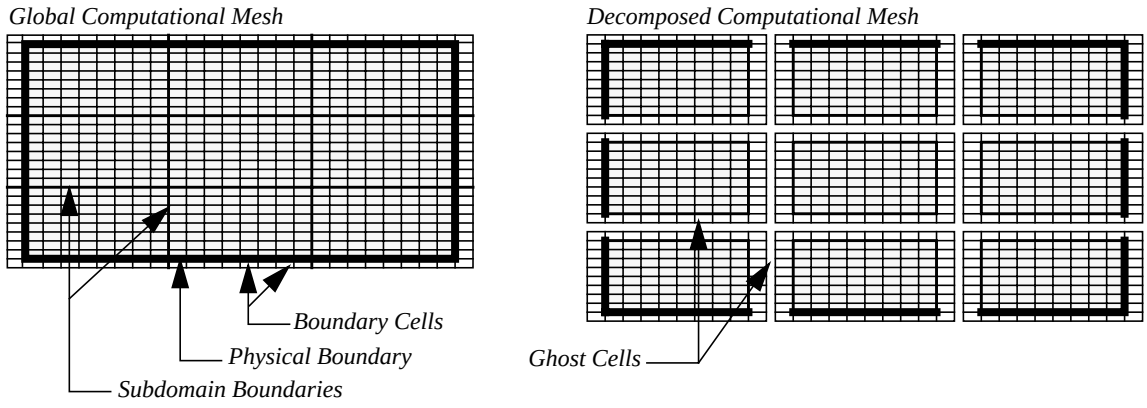



Figure 3. Illustration of the decomposition of the global computational domain into subdomains, in two spatial dimensions.

The translation from CM Fortran to Fortran 77 primarily involved replacing single-line matrix instructions, such as those for adding two vectors, by Fortran 77 DO loops; we also combined adjacent DO loops where appropriate. This process is illustrated with the program fragments shown in Figure 5. Each array statement in the CM Fortran (Figure 8, left) is equivalent to a set of DO loops in Fortran 77. In many cases, adjacent equivalent DO loops in the Fortran 77 translation can be combined, as illustrated in the right half of Figure 5.

```
CM Fortran:
   real,array(0:n1,0:n2,0:n3)::a,b
   .
   .
   .
   .
   b = a + cshift(a,1,1)
   c = c + cshift(a,1,1)
```

```
Fortran 77:
    real  a(0:n1, 0:n2, 0:n3),
    &     b(0:n1, 0:n2, 0:n3)
    .
    .
    .
    call shift_left(a)
    do 300 k = 0, n3
      do 200 j = 0, n2
        do 100 i = 0, n1-1
          b(i,j,k)=a(i,j,k)+a(i+1,j,k)
          c(i,j,k)=c(i,j,k)+a(i+1,j,k)
100     continue
        b(n1,j,k)=a(n1,j,k)+a(0,j,k)
        c(n1,j,k)=c(n1,j,k)+a(0,j,k)
200   continue
300 continue
```

Figure 4. Program fragments illustrating the translation from CM Fortran (left) to Fortran 77 (right). The DO loops implicit in the CM Fortran array addition are explicit in the Fortran 77 equivalent. In addition, the different communication overhead is evident for the simple index shift operations in CM Fortran and Fortran 77. Two communications, one for each cshift operation, are required in the CM Fortran code. The equivalent Fortran 77 requires only one, via the shift_left subroutine. The function of shift_left is illustrated in Figure 5.

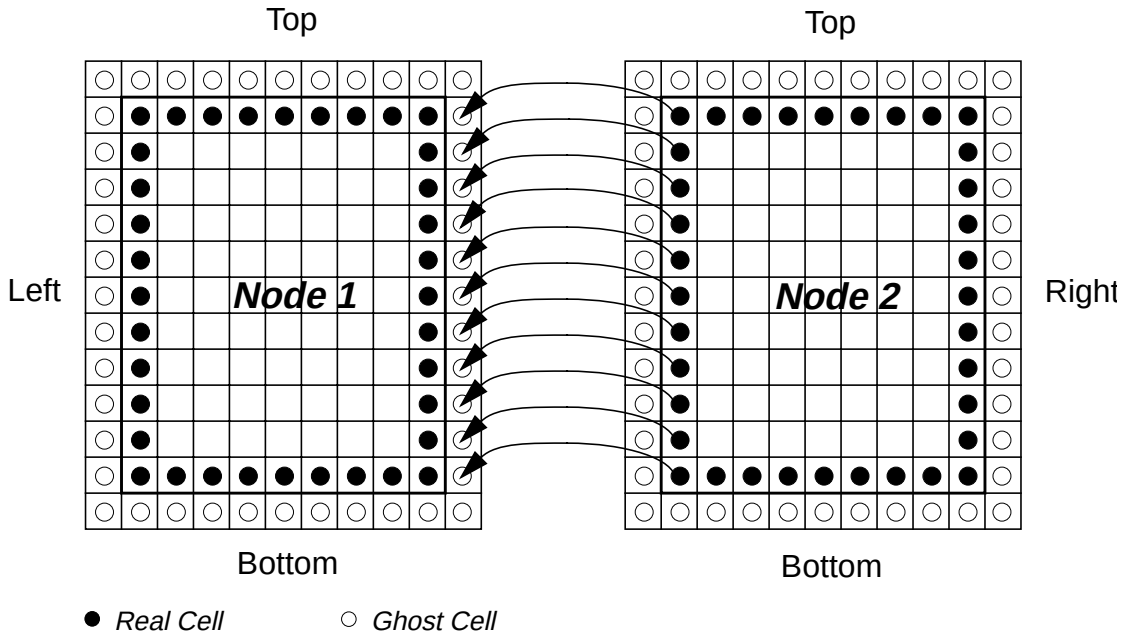



Figure 5. Illustration of shift_left local inter-node communication between two nodes for two-dimensional subdomains for a message-passing code. The elements on the left boundary of node 2 are communicated to the ghost cells on the right boundary of node 1.

The `cshift` function shown in Figure 5 performs a circular shift; for example, `cshift(a,1,1)` returns the array of values of the array `a` shifted by one in its first index. At the maximum value of the first index, the `cshift` function wraps around and returns the value of `a` at the minimum value of the first index (hence the name “circular shift”). This is made more clear by the equivalent `Fortran 77` code (Figure 4, right). Within a processor, `cshift` makes internal memory copies; between processors, `cshift` uses interprocessor communication to obtain the values required.

The equivalent `Fortran 77` code also illustrates the reduction in communication which is often possible in the message-passing code, compared to the data-parallel code. In the `CM Fortran` code (Figure 8, left), many internal memory copies are made and interprocessor communication is required for each call to the `cshift` function. In the equivalent `Fortran 77` code, no internal memory copies are required on the node, and only one communication, implemented with the function `shift_left`, is required. These reductions in memory copies and inter-node communication can significantly improve the performance of the message-passing code over the data-parallel code [11]. The function of the `shift_left` routine is illustrated in Figure 5 for two-dimensional subdomains. In this figure, node 2 sends the contents of its left-most cells to the right ghost cells on the node to its left, node 1.

## Features of MIMD PAGOSA 5.5

MIMD PAGOSA 5.5 contains all the features of the SIMD PAGOSA 5.5 code, except for the X Window graphics, tracer particles, and shadow regions for the high-explosive burn time calculations. (SIMD PAGOSA 5.5 does not contain fracture models).

Simulations from MIMD PAGOSA 5.5 can be visualized using the iso isosurface code, developed by Patricia J. Crossno at Sandia. iso can be run heterogeneously with MIMD PAGOSA, or used as a post-processor. iso runs on the Paragon and the nCUBE 2.

We have also developed an MIMD version of GEN, the generator, or problem-setup, code for PAGOSA. MIMD GEN runs on the same platforms as MIMD PAGOSA, and can be run either independently of MIMD PAGOSA (for verifying that a problem has been set up correctly) or called from within MIMD PAGOSA when running a simulation.

An input guide for MIMD PAGOSA 5.5 and MIMD GEN is included in Appendix A.

# The Test Simulations

The test simulations used to measure the performance of PAGOSA on the parallel computers and to demonstrate its capabilities were the oblique impact of a finned tungsten projectile on a stainless steel plate, the explosive welding of a copper tube to a steel plate, and the perforation of an oil-well casing by a shaped charge.

## The Finned Projectile Simulations

In the simulation of oblique impact of a finned tungsten projectile on a stainless steel plate, a finned tungsten projectile 0.9 cm long and 0.15 cm in diameter impacts a 0.15-cm thick stainless steel plate at an angle of $30^{o}$ from the normal and a speed of 1.0 km/s. We used linear Us/Up Mie-Grüneisen equations of state for the tungsten and the stainless steel. We ran this simulation with the hydrodynamic constitutive model to a time of 5 μs, when the projectile has clearly perforated the plate. The left frame in Figure 6 shows the initial configuration; the right frame shows the configuration at 5.0 μs. A typical input file for this simulation is given in Appendix B.

We also ran the same simulation with the elastic, perfectly plastic constitutive model for the tungsten and the steel to a time of 6.7 μs, when the penetrator has clearly penetrated the plate. The left frame in Figure 7 shows the initial configuration; the right frame shows the configuration at 6.7 μs. A typical input file for this simulation is given in Appendix B.

Figure 6. Simulation of a finned tungsten projectile obliquely impacting a stainless steel plate (hydrodynamic constitutive model).

$0\mu s$ $6.7\mu s$

Figure 7. Simulation of a finned tungsten projectile obliquely impacting a stainless steel plate (elastic, perfectly plastic constitutive model).

## The Explosive Welding Simulation

In the simulation of the explosive welding of a copper tube to a stainless steel plate [2], a cylindrical charge of the high explosive PBX-9501 is ignited inside a copper tube, and the resulting detonation welds the tube to a steel plate. The copper tube is 1.2 cm in inside diameter with a wall thickness of 0.4 cm. The tube is inserted through a $64^o$ bevelled hole in the plate and protrudes 3.0 cm beyond the surface of the plate. The internal diameter of the hole in the plate is 2.0 cm and the external hole opening is 4.0 cm. The PBX-9501 is in the form of a rod, 0.6 cm in outside diameter and of length 0.5 cm. The PBX-9501 is held

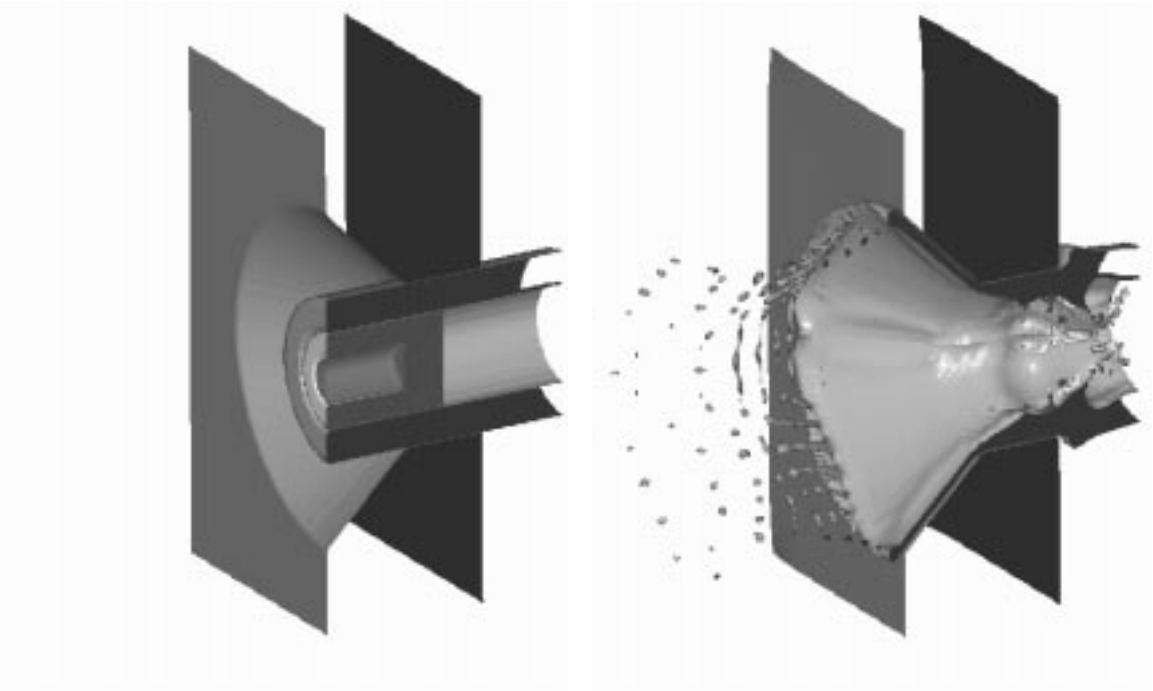


Figure 8. Simulation of the explosive welding of a copper tube to a stainless steel plate. The initial configuration is shown on the left; the simulation at 65 μs is shown on the right.

in a position coaxial with the tube by a foam cup, which fills the space between the PBX-9501 and the inner surface of the copper tube, and also projects 0.25 cm beyond the end of the high explosive. The PBX-9501 was detonated at the center of the internal end of the rod using a point detonator. We used a linear Us/Up Mie-Grüneisen equation of state and the elastic-perfectly plastic constitutive model for the copper, the 304 stainless steel, and the foam. The PBX-9501 detonation products were modeled with the Jones-Wilkins-Lee equation of state. The programmed burn model was used to model the detonation. For the performance measurements we ran this simulation a time of 15 μs, when significant deformation of the copper has occurred. The left frame in Figure 8 shows the initial configuration; the right frame shows the configuration at 65 μs, when the weld is essentially complete. A typical input file for this simulation is given in Appendix B.

## The Oil-Well Perforation Simulation

In the oil-well perforation simulation, the casing of an oil well is perforated with two small shaped-charge jets. Well bores are typically lined with steel pipe, or concrete casing, or both; the liner usually must be perforated with tiny high-explosive charges prior to pumping. Perforation allows production of oil from specific depths determined from logging data. The perforators are inserted into the well hole inside carrier tubes and then detonated when the tube has been lowered to the prescribed depth. They are designed to

make clean holes in the casing and to penetrate several inches outward into the surrounding oil-bearing strata.

In this simulation, two perforator charges are aimed horizontally in opposite directions inside a steel carrier tube that has been inserted in an oil well. The perforators are similar to a current industrial design, with a conical copper liner surrounded by high explosive and a steel case. The carrier tube is positioned flush against one side of the well casing. Each charge is point-detonated at the apex of the high explosive layer surrounding the conical copper liner. Energy release in the detonated explosive then causes the liners to converge and form shaped-charge jets that perforate the steel carrier tube and casing, and penetrate into the surrounding rock.

The inner diameter of the stainless steel well casing is 6.21 cm and the casing is 10.0 cm long and 0.77 cm thick. The casing is surrounded by rock, which is modeled as quartz. The carrier tube is stainless steel with a spherical cap. It has an inside diameter of 1.55 cm and a wall thickness of 0.44 cm. The space between the carrier tube and the well casing is filled with water. The stainless steel, the liner material, and the explosive charge casing were modeled with linear Us/Up Mie-Grüneisen equations of state and elastic, perfectly plastic constitutive models. The high-explosive, cyclotol, was modeled using the Jones-Wilkins-Lee equation of state with the hydrodynamic constitutive model, and was detonated using a programmed burn model. The water was modeled with a linear Us/Up Mie-Grüneisen equation of state and the hydrodynamic constitutive model.

The upper frame in Figure 9 shows the initial configuration; the lower frame shows the configuration at 140 μs, when the first perforation is complete. An input file for this simulation is given in Appendix B.

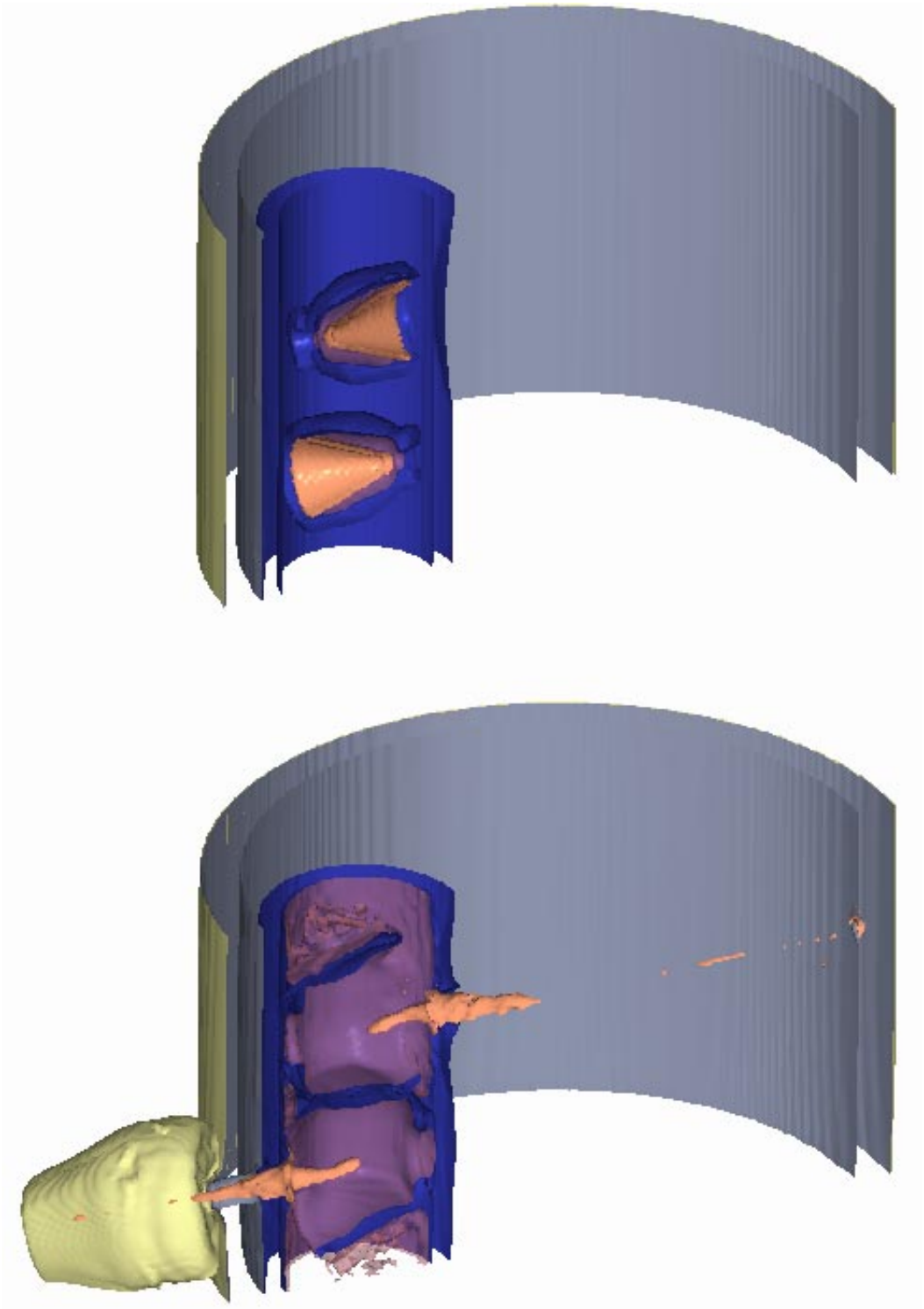


Figure 9. Simulation of the perforation of a steel oil-well casing by a shaped charge jet. The initial configuration is shown in the upper figure.The configuration at 140 μs is shown in the lower figure, when the lower shaped-charge jet has perforated the well casing and the upper shaped-charge jet has reached the far side of the casing. The water and air are not shown in these figures.

## The Test Conditions

The code was compiled on each machine at the highest level of optimization which still yielded correct answers (Table 1).

All simulations were conducted in full-precision (64-bit) arithmetic. The `-Knoieee` option used on the `Paragon` substitutes an in-line divide algorithm which produces results that differ from results generated by algorithms conforming to the IEEE 754 standard by no more than three units in the last place.

For the scaled speedup calculations for the message-passing version of PAGOSA, the problem size was increased by adding nodes in powers of two. A subdomain of fixed size, with the size depending on the machine, was placed on each node and the number of nodes was increased.

Only the main computational loop was timed, unless otherwise noted. I/O time was excluded, except for brief diagnostics which were similar for all versions of the code. Grind times reported in the tables in the following sections are averages over the full simulation in each case, unless otherwise noted. The repeatability of the grind time measurements was tested for each machine, and results are given in Table 2. On each machine the variation in the grind times was less than 0.2% over ten trials, and so the grind time results reported are for a single calculation.

**Table 1: Compiler Versions and Options for Compiling MIMD PAGOSA**

| Machine | Compiler Version | Compiler Options Used |
|---|---|---|
| nCUBE 2 | 2.2 | `-O2` |
| Paragon XP/S | R4.5 | `-Knoieee -O4 -Mframe -Mvect=recog,trans-form,cachesize:12288 -Mnostreamall -Mnoxp -Mnoperfmon -Mnodepchk -Mnostride0 -Mnodebug` |

**Table 2: Grind Time Repeatability***

| Machine | Grind Time Range (μs/cell/timestep) | Mean Grind Time (μs/cell/timestep) | Standard Deviation (μs/cell/timestep) |
|---|---|---|---|
| nCUBE 2 | 108.432—108.469 | 108.446 | 0.0123 |
| Paragon XP/S | 13.214—13.262 | 13.226 | 0.0171 |

* Results are for the hydrodynamic finned projectile simulation. Ten calculations were run on 64 nodes and for the largest subdomain which would fit on each node. Each calculation was run for the same number of time steps on each machine, and for at least 20 time steps.

## The Performance of MIMD PAGOSA

In this section we describe the nCUBE 2 and the Intel Paragon. We will present the performance of MIMD PAGOSA as measured by the scaled speedup, parallel scaled efficiency, and the grind time for four different simulations:

- The finned projectile simulation without material strength. This represents a minimal problem of two active materials (tungsten and stainless steel) with a simple but non-trivial equation of state which will exercise the interface tracker. For reference purposes, this simulation is designated fp1.
- The finned projectile simulation with material strength. This represents a minimal problem of two active materials (tungsten and stainless steel) with a simple but non-trivial equation of state and material strength which will exercise the interface tracker. This will indicate the computational expense of the material strength model. For reference purposes, this simulation is designated fp2.
- The explosive welding simulation. This represents a more typical simulation than the first two, and adds the detonation of the high explosive. For reference purposes, this simulation is designated ew.
- The oil-well perforation simulation. This represents a more complex simulation, with 10 materials of various types, including a high explosive. For reference purposes, this simulation is designated owp.

The intent of this study is to indicate the scalability of PAGOSA on the two MIMD parallel computers and the relative performance of the code on those machines. Both the MIMD computers ran the same code, compiled as described in the previous section, with no other computer-specific optimizations. We assume that given sufficient incentive MIMD PAGOSA could be optimized for either of the parallel computers used; our results provide a lower bound on the achievable performance of a real application code on these machines.

For each computer we present the scaled speedup, the parallel scaled efficiency, and the grind time for two cases:

- The largest subdomain per node, and
- The same subdomain size as used on the nCUBE 2.

Presenting results for the largest subdomain per node will indicate the best performance for each computer. Presenting the results for the same subdomain size on all the computers will allow comparisons between the machines to be made more easily. The nCUBE 2 has the smallest memory per node, and hence the smallest subdomain size.

## Message-Passing Performance on the nCUBE 2

In this section we present the performance of MIMD PAGOSA on the nCUBE 2, measured for the fp1, fp2, ew, and owp simulations. The compiler version and options are given in Table 1. We first briefly describe the nCUBE 2, and then present the performance results.

### *Description of the nCUBE 2*

The nCUBE 2 is a massively parallel, hypercube-topology, distributed-memory, multiple-instruction stream, multiple-data stream (MIMD) computer. Each node is capable of running one or more complete programs independently of the other nodes. All coordination or cooperation between nodes, *i.e.*, "parallel processing", is performed via explicit message passing calls. The maximum nCUBE 2 configuration is 8192 nodes with 16 megabytes of local memory each. The MPCRL's nCUBE 2 was acquired in 1989 and is configured with 1024 four-megabyte nodes.

The individual node is a single-chip VLSI implementation of nCUBE's proprietary instruction set architecture, integrating both communications and memory control. The remainder of a complete system consists mainly of memory chips, communications lines, power supplies and cooling. 1024 of the nodes fit into a 4-foot high 19-inch rack. Each node has 64-bit internal data paths, sixteen general registers, and 28 direct memory access (DMA) communication channels. While a logical address is 30 bits, the physical address is 26 bits, resulting in a maximum physical memory size of 64 megabytes per node. A 64-megabyte node is implemented on a "double-wide" module using 16-megabit chips. When double-wide modules are used the maximum configuration is reduced to 4096 nodes.

The instructions are complex (CISC not RISC). Up to three operands can be specified with addressing modes ranging from register direct to word offset plus stack pointer memory indirect. The node is rated at 7 million instructions per second, and 3.5 million single precision or 2.7 million double precision floating point operations per second. Typical performance is 4 to 7 mips or 1.5 to 2 megaFLOPs per node. The 1024-node system achieves 1.5 to 2 gigaFLOPs on applications that scale well. Eight-, 16-, 32- and 64-bit twos-complement and unsigned integer formats and 32- and 64-bit IEEE floating point formats are supported.

The 28 communication channels are paired to form 14 bi-directional links. Thirteen of the links may be connected to nearest neighbor nodes resulting in the maximum number of nodes of two raised to the 13th power, or 8192 nodes. The 14th channel may be connected to an I/O node, an identical node on a separate I/O board. "Wormhole routing" exploits a gray-code numbering of the nodes and the hypercube interconnection topology, to open direct links to distant nodes without using the memory or processing power of the intervening nodes. The farthest node can be only 13 "hops" away, so the 2-microsecond performance penalty for each hop is small relative to the 50- to 150-microsecond software

start-up overhead and the asymptotic 2.2 megabyte/second transfer rate in each direction for a total bandwidth of 4.4 megabytes/second in full duplex mode.

nCUBE supplies a cross-development environment supporting VME bus-based, shared-memory interfaces to either SUN or Silicon Graphics host workstations. Software Release 3.2 includes optimizing compilers for the Fortran 77, ANSI C and C++ languages. nCUBE compilers support the typical CISC processor optimizations: global register allocation, strength reduction, common sub-expression elimination, invariant code motion, *etc*. The compilers do not provide automatic parallelism; all parallelism must be explicitly specified by the programmer, whether at the lowest level of message passing library calls, or indirectly through calls to routines implementing a higher level paradigm or math libraries.

The nCX operating system and xnc generic host program combine to supply a UNIX® environment for programs. Operating system services which cannot be satisfied on the local node are converted to an exchange of messages to the I/O node or host resource which can supply the requested service.

While each node can be time-sliced among several user processes or jobs, a large system is usually shared among multiple users via a concept called *space sharing*. In space sharing, each user gets the exclusive use of a subset of the total available nodes called a *subcube*. A subcube is restricted by the hypercube architecture to consist of a power-of-two number of nodes. Single node subcubes are allowed.

### *Message-Passing Performance on the nCUBE 2*

The message-passing performance of MIMD PAGOSAon the nCUBE 2 is indicated by the grind time, scaled speedup, and parallel scaled efficiency data presented in Tables 3–6. This data is displayed graphically in Figures 8–12.The calculations were performed on an nCUBE 2 in the MPCRL. The repeatability of the grind time is shown by the data in Table 2.

**Table 3: Performance of MIMD PAGOSA on the nCUBE 2 for fp1**

| Number of Nodes | Problem Size on Node 0 $(nx \times ny \times nz)$ | Global Problem Size $(nx \times ny \times nz)$ | Scaled Speedup | Parallel Scaled Efficiency | Grind Time (μs/cell/ timestep) |
|---|---|---|---|---|---|
| 1 | $18 \times 12 \times 7$ | $18 \times 12 \times 7$ | 1.000 | 1.000 | 5754.819 |
| 2 | $9 \times 12 \times 14$ | $18 \times 12 \times 14$ | 1.844 | 0.922 | 3120.893 |
| 4 | $9 \times 12 \times 14$ | $18 \times 24 \times 14$ | 3.558 | 0.890 | 1617.461 |
| 8 | $9 \times 12 \times 14$ | $36 \times 24 \times 14$ | 6.909 | 0.864 | 832.892 |
| 16 | $9 \times 12 \times 14$ | $36 \times 24 \times 28$ | 13.43 | 0.839 | 428.501 |
| 32 | $9 \times 12 \times 14$ | $36 \times 48 \times 28$ | 26.23 | 0.820 | 219.438 |
| 64 | $9 \times 12 \times 14$ | $72 \times 48 \times 28$ | 52.04 | 0.813 | 110.586 |
| 128 | $9 \times 12 \times 14$ | $72 \times 48 \times 56$ | 102.4 | 0.800 | 56.213 |
| 256 | $9 \times 12 \times 14$ | $72 \times 96 \times 56$ | 201.3 | 0.786 | 28.594 |
| 512 | $9 \times 12 \times 14$ | $144 \times 96 \times 56$ | 397.7 | 0.777 | 14.470 |
| 1024 | $9 \times 12 \times 14$ | $144 \times 96 \times 112$ | 789.6 | 0.771 | 7.288 |

**Table 4: Performance of MIMD PAGOSA on the nCUBE 2 for fp2**

| Number of Nodes | Problem Size on Node 0 $(nx \times ny \times nz)$ | Global Problem Size $(nx \times ny \times nz)$ | Scaled Speedup | Parallel Scaled Efficiency | Grind Time (μs/cell/ timestep) |
|---|---|---|---|---|---|
| 1 | $15 \times 10 \times 6$ | $15 \times 10 \times 6$ | 1.000 | 1.000 | 9384.238 |
| 2 | $8 \times 10 \times 12$ | $15 \times 10 \times 12$ | 1.816 | 0.908 | 5168.621 |
| 4 | $8 \times 10 \times 12$ | $15 \times 20 \times 12$ | 3.469 | 0.867 | 2704.819 |
| 8 | $8 \times 10 \times 12$ | $30 \times 20 \times 12$ | 6.653 | 0.832 | 1410.551 |
| 16 | $8 \times 10 \times 12$ | $30 \times 20 \times 24$ | 12.82 | 0.801 | 731.922 |
| 32 | $8 \times 10 \times 12$ | $30 \times 40 \times 24$ | 24.82 | 0.776 | 378.141 |
| 64 | $8 \times 10 \times 12$ | $60 \times 40 \times 24$ | 49.17 | 0.768 | 190.858 |
| 128 | $8 \times 10 \times 12$ | $60 \times 40 \times 48$ | 95.99 | 0.750 | 97.758 |
| 256 | $8 \times 10 \times 12$ | $60 \times 80 \times 48$ | 185.3 | 0.724 | 50.656 |
| 512 | $8 \times 10 \times 12$ | $120 \times 80 \times 48$ | 365.2 | 0.713 | 25.694 |
| 1024 | $8 \times 10 \times 12$ | $120 \times 80 \times 96$ | 720.3 | 0.703 | 13.028 |

**Table 5: Performance of MIMD PAGOSA on the nCUBE 2 for ew**

| Number of Nodes | Problem Size on Node 0 $(nx \times ny \times nz)$ | Global Problem Size $(nx \times ny \times nz)$ | Scaled Speedup | Parallel Scaled Efficiency | Grind Time (μs/cell/ timestep) |
|---|---|---|---|---|---|
| 1 | $14 \times 7 \times 7$ | $14 \times 7 \times 7$ | 1.000 | 1.000 | 11475.076 |
| 2 | $7 \times 7 \times 14$ | $14 \times 7 \times 14$ | 1.778 | 0.889 | 6452.905 |
| 4 | $7 \times 7 \times 14$ | $14 \times 14 \times 14$ | 3.311 | 0.828 | 3466.073 |
| 8 | $7 \times 7 \times 14$ | $28 \times 14 \times 14$ | 6.170 | 0.771 | 1859.853 |
| 16 | $7 \times 7 \times 14$ | $28 \times 14 \times 28$ | 11.85 | 0.740 | 968.521 |
| 32 | $7 \times 7 \times 14$ | $28 \times 28 \times 28$ | 22.49 | 0.703 | 510.196 |
| 64 | $7 \times 7 \times 14$ | $56 \times 28 \times 28$ | 44.28 | 0.692 | 259.141 |
| 128 | $7 \times 7 \times 14$ | $56 \times 28 \times 56$ | 85.97 | 0.672 | 133.471 |
| 256 | $7 \times 7 \times 14$ | $56 \times 56 \times 56$ | 169.3 | 0.662 | 67.764 |
| 512 | $7 \times 7 \times 14$ | $112 \times 56 \times 56$ | 337.4 | 0.659 | 34.011 |
| 1024 | $7 \times 7 \times 14$ | $112 \times 56 \times 112$ | 659.6 | 0.644 | 17.396 |

**Table 6: Performance of MIMD PAGOSA on the nCUBE 2 for owp**

| Number of Nodes | Problem Size on Node 0 $(nx \times ny \times nz)$ | Global Problem Size $(nx \times ny \times nz)$ | Scaled Speedup | Parallel Scaled Efficiency | Grind Time (μs/cell/ timestep) |
|---|---|---|---|---|---|
| 1 | $15 \times 6 \times 6$ | $15 \times 6 \times 6$ | 1.000 | 1.000 | 18449.366 |
| 2 | $8 \times 12 \times 6$ | $15 \times 12 \times 6$ | 1.814 | 0.907 | 10168.498 |
| 4 | $8 \times 6 \times 12$ | $15 \times 12 \times 12$ | 3.373 | 0.843 | 5469.947 |
| 8 | $8 \times 6 \times 12$ | $30 \times 12 \times 12$ | 6.532 | 0.816 | 2824.440 |
| 16 | $8 \times 6 \times 12$ | $30 \times 24 \times 12$ | 12.44 | 0.778 | 1482.658 |
| 32 | $8 \times 6 \times 12$ | $30 \times 24 \times 24$ | 24.17 | 0.755 | 763.337 |
| 64 | $8 \times 6 \times 12$ | $60 \times 24 \times 24$ | 47.44 | 0.741 | 388.899 |
| 128 | $8 \times 6 \times 12$ | $60 \times 48 \times 24$ | 93.92 | 0.734 | 196.445 |
| 256 | $8 \times 6 \times 12$ | $60 \times 48 \times 48$ | 184.2 | 0.720 | 100.168 |
| 512 | $8 \times 6 \times 12$ | $120 \times 48 \times 48$ | 364.4 | 0.712 | 50.635 |
| 1024 | $8 \times 6 \times 12$ | $120 \times 96 \times 48$ | 728.9 | 0.712 | 25.312 |

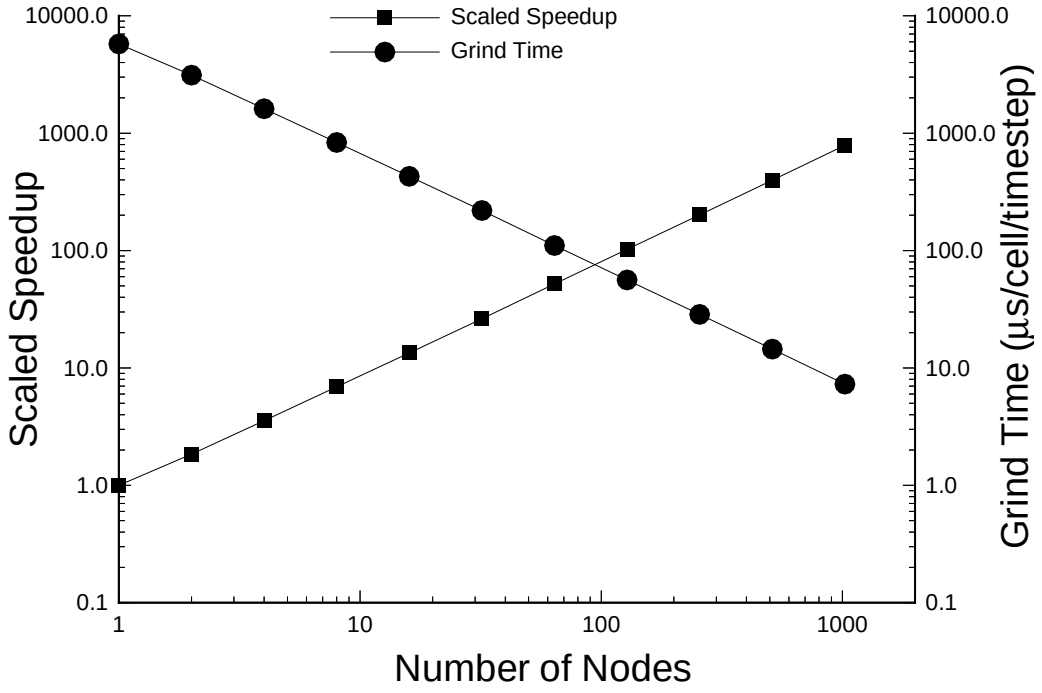



Figure 10. Scaled speedup and grind time for the fp1 simulation on the nCUBE 2.

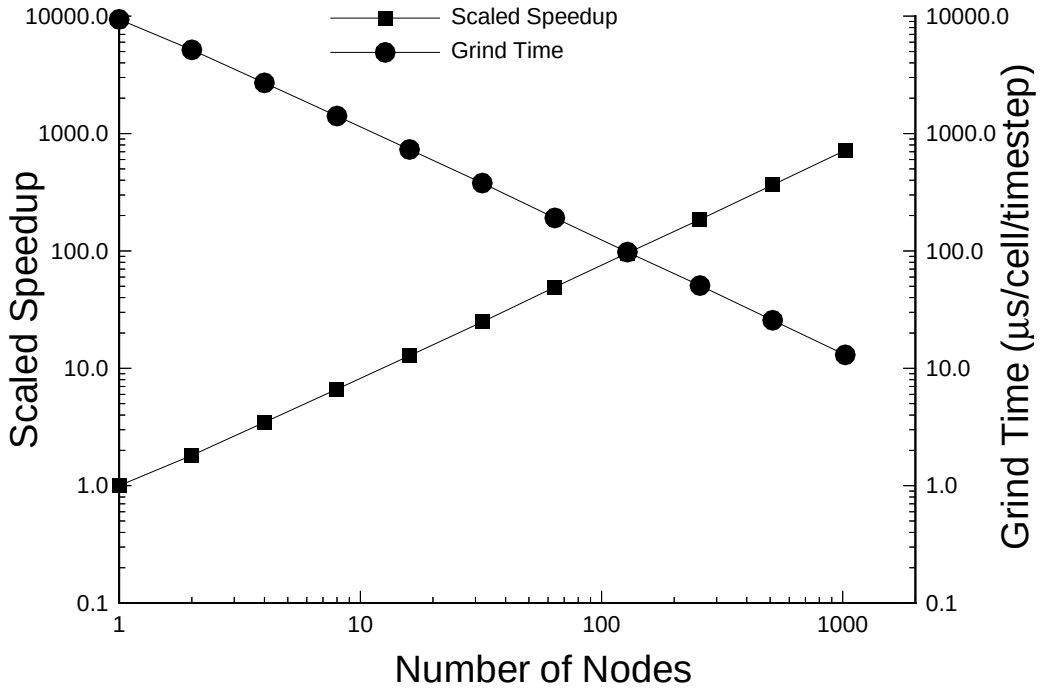



Figure 11. Scaled speedup and grind time for the fp2 simulation on the nCUBE 2.

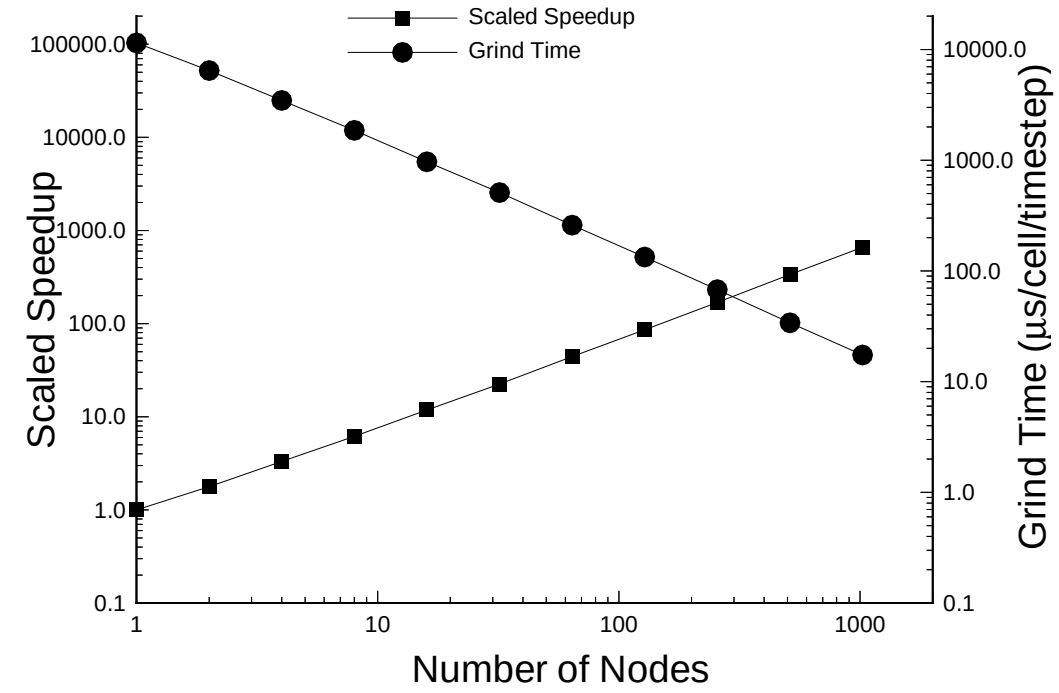



Figure 12. Scaled speedup and grind time for the ew simulation on the nCUBE 2.

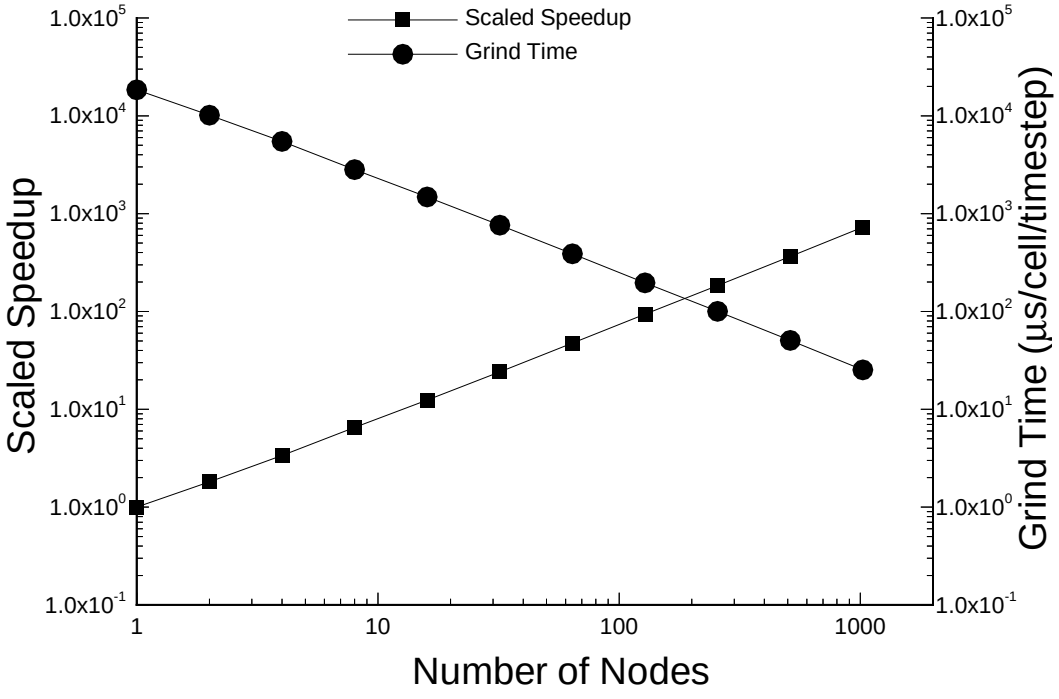



Figure 13. Scaled speedup and grind time for the owp simulation on the nCUBE 2.

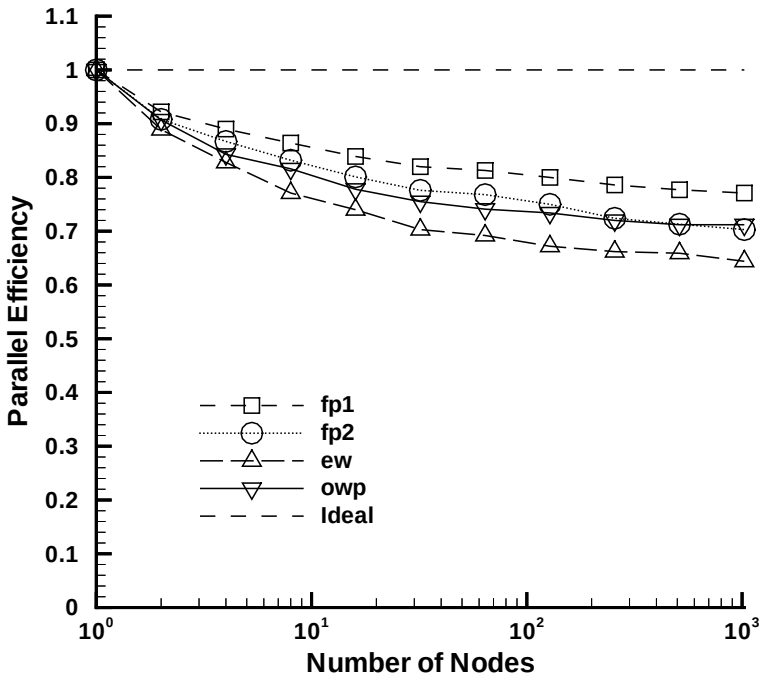



Figure 14. Scaled parallel efficiency as a function of the number of nodes on the nCUBE 2.

## Message-Passing Performance on the Intel Paragon

In this section we present the performance of MIMD PAGOSA on the Paragon, measured for the fp1, fp2, ew, and owp simulations. The compiler version and options are given in Table 1. We first briefly describe the Paragon, and then present the performance results.

### *Description of the Intel Paragon*

The Intel Paragon is a massively parallel computer which uses a two-dimensional mesh topology for communications. It is a Multiple-Instruction, Multiple-Data (MIMD) computer which can be used with either the Single-Program, Multiple-Data (SPMD) or Multiple-Program, Multiple-Data (MPMD) programming models. It uses explicit message passing for communications between nodes.

The Paragon at Sandia has 1824 compute nodes, 48 disk nodes, 11 service nodes, 3 HiPPI nodes, two Ethernet nodes, and one FDDI node. There are two sizes of compute nodes in the machine: 1312 nodes with 16 MB of memory and 512 nodes with 32 MB of memory. Each disk node contains 16 MB of memory and is connected to a five-disk, level-3 RAID which has 4.8 GB of formatted space. The 11 service nodes are 32 MB nodes which run the OSF-1/AD operating system (Open Software Foundation) with a single-system image. The network nodes are all 32-MB nodes, except for the FDDI node which only contains 16 MB of memory. There is a total of 36.5 GB of memory on the compute nodes and a total of 37.7 GB of memory in the machine.

The nodes in the Paragon are connected in a two-dimensional mesh. Each node is connected to the mesh via a mesh router chip which is on the back-plane of the Paragon, not on the node. The mesh router chips route the messages across the mesh back-planes without any intervention from the nodes, until the message arrives at the destination node. At this point, the message is delivered to the node via a network interface chip (NIC) on the node board. Messages are routed first in the X direction, then in the Y direction. Measured message latency, from user level to user level, on the Paragon is 17 microseconds for zero-length messages. The achieved bandwidth is 160 MB/s, which is over 90% of the theoretical bandwidth. The size of the mesh is 120x16 giving an aspect ratio of 7.5:1.

Each node in the Paragon contains two Intel i860-XP processors: one is used for computing, the other is used as a message co-processor. The processors operate at 50 MHz and have a performance of 75 MFLOPS using 64-bit arithmetic. It can run in dual instruction mode which simultaneously executes integer and floating-point instructions. In addition, the floating-point unit can execute dual operations using the adder and the multiplier units in parallel. There are on-board instruction and data caches on the i860 chip. Each is a four-way, set-associative cache of 16-byte lines for a total of 16 KB for instructions and 16 KB for data. Cache coherence is maintained using an MESI protocol.

The i860 also uses pipelines and has delayed branch instructions to avoid breaks in the pipes.

Each node also has a network interface chip which connects the node to the mesh router chip in the mesh. There are two DMA devices on each node which allow data to be fed onto the network with minimal support from a processor. While a DMA transfer is in progress from memory to the network, the processor can be doing other work. A node also has a 2-KB cache for sending messages and a 2-KB cache for receiving messages. These caches allow the DMA devices, or the mesh router chip, to initialize the sending or receiving of a message before actually putting data onto the network, or into memory.

The operating system shipped with the Paragon is OSF-1/AD from Open Software Foundation (OSF). It is based on the Mach microkernel architecture. It also contains system software from Locus Computing which is used to give a single-system image to the Paragon. This allows all OSF nodes to have uniform access to all resources such as disks and networks. It does not matter from which node an application makes a system request, each node has access to all of the resources of the machine. OSF uses the NX protocol for message passing. This is the same protocol used in the Intel iPSC/860, so iPSC/860 codes can be ported with little difficulty.

The Paragon at Sandia actually uses a heterogeneous operating system environment in which OSF and SUNMOS (Sandia/University of New Mexico Operating System) co-exist. OSF runs on the service nodes, disk nodes and network nodes, while SUNMOS runs on the compute nodes. SUNMOS is an operating system which was jointly developed by Sandia and UNM. It was designed as a single-tasking operating system which depends on OSF to provide services such as I/O (see [20]). The main task of SUNMOS is to run user processes, pass messages and provide an interface to OSF for I/O. SUNMOS provides the capability to use both processors for computing, as well as using one for computing and one as a message co-processor. This allows the user to use the second processor according to the needs of the application.

SUNMOS contains emulation libraries for both NX, the message passing library used on the Paragon, and Vertex, the message passing library used on the nCUBE 2. This allows applications which use either NX or Vertex to be easily ported to SUNMOS. In fact, some codes at Sandia use both the NX and Vertex message passing libraries. The implementation of the Vertex emulation is complete except for the functions which use pointers to buffers in the communications buffer. The NX emulation is not quite as complete. All message passing functions are implemented, except for hrecv(). However, all library calls which deal with processes are not implemented because SUNMOS is a single-tasking operating system.

The Paragon has cross compilers for both Sun and SGI development environments as well as native compilers on the Paragon. The languages supported include C, C++ and Fortran. C++ is provide by both Cfront (version 3.0) and a C++ compiler. The

compilers for the Paragon were developed by Portland Group Inc. There are also SUNMOS versions of the compilers for C, C++ and Fortran. These compilers use the Portland Group compilers to create object files and then link these files with the appropriate SUNMOS libraries. The nodes are shared among users via space sharing.

### *Message-Passing Performance on the Paragon*

The message-passing performance of MIMD PAGOSA on the Paragon is indicated by the grind time, scaled speedup, and parallel scaled efficiency data presented in Tables 7–10, and graphically in Figures 15–18. The calculations were performed on the 1824-node Paragon in the Massively Parallel Computing Research Laboratory at Sandia. Two subdomain sizes per node were used; the smaller is the same as the largest subdomain size per node on the nCUBE 2 and the latter is the largest that will fit on the 16-megabyte nodes of the Paragon. The repeatability of the grind time is shown by the data in Table 2**.**

**Table 7: Performance of MIMD PAGOSA on the Intel Paragon for fp1**

| Number of Nodes | Problem Size on Node 0 $(nx \times ny \times nz)$ | Global Problem Size $(nx \times ny \times nz)$ | Scaled Speedup | Parallel Scaled Efficiency | Grind Time ($\mu$s/cell/ timestep) |
|---|---|---|---|---|---|
| 1 | $18 \times 12 \times 7$ | $18 \times 12 \times 7$ | 1.000 | 1.000 | 903.748 |
| 2 | $9 \times 12 \times 14$ | $18 \times 12 \times 14$ | 1.790 | 0.895 | 505.017 |
| 4 | $9 \times 12 \times 14$ | $18 \times 24 \times 14$ | 3.463 | 0.866 | 261.001 |
| 8 | $9 \times 12 \times 14$ | $36 \times 24 \times 14$ | 6.668 | 0.834 | 135.536 |
| 16 | $9 \times 12 \times 14$ | $36 \times 24 \times 28$ | 12.99 | 0.812 | 69.551 |
| 32 | $9 \times 12 \times 14$ | $36 \times 48 \times 28$ | 25.16 | 0.786 | 35.917 |
| 64 | $9 \times 12 \times 14$ | $72 \times 48 \times 28$ | 49.83 | 0.778 | 18.138 |
| 128 | $9 \times 12 \times 14$ | $72 \times 48 \times 56$ | 96.66 | 0.755 | 9.350 |
| 256 | $9 \times 12 \times 14$ | $72 \times 96 \times 56$ | 191.4 | 0.748 | 4.723 |
| 512 | $9 \times 12 \times 14$ | $144 \times 96 \times 56$ | 377.2 | 0.737 | 2.396 |
| 1024 | $9 \times 12 \times 14$ | $144 \times 96 \times 112$ | 751.9 | 0.734 | 1.202 |
| 1 | $36 \times 24 \times 12$ | $36 \times 24 \times 12$ | 1.000 | 1.000 | 741.244 |
| 2 | $36 \times 24 \times 12$ | $72 \times 24 \times 12$ | 1.932 | 0.966 | 383.745 |
| 4 | $36 \times 24 \times 12$ | $72 \times 48 \times 12$ | 3.828 | 0.957 | 193.645 |
| 8 | $18 \times 24 \times 24$ | $72 \times 48 \times 24$ | 7.184 | 0.890 | 103.178 |
| 16 | $18 \times 24 \times 24$ | $144 \times 48 \times 24$ | 14.22 | 0.889 | 52.137 |
| 32 | $18 \times 24 \times 24$ | $144 \times 96 \times 24$ | 27.73 | 0.866 | 26.733 |
| 64 | $18 \times 24 \times 24$ | $144 \times 96 \times 48$ | 54.59 | 0.853 | 13.578 |
| 128 | $18 \times 24 \times 24$ | $144 \times 96 \times 96$ | 107.0 | 0.836 | 6.928 |
| 256 | $18 \times 24 \times 24$ | $144 \times 192 \times 96$ | 207.9 | 0.812 | 3.565 |
| 512 | $18 \times 24 \times 24$ | $288 \times 192 \times 96$ | 409.3 | 0.799 | 1.808 |
| 1024 | $18 \times 24 \times 24$ | $288 \times 192 \times 192$ | 814.1 | 0.795 | 0.909 |
| 1824 | $24 \times 25 \times 19$ | $438 \times 292 \times 146$ | 1353. | 0.742 | 0.548 |

**Table 8: Performance of MIMD PAGOSA on the Intel Paragon for fp2**

| Number of Nodes | Problem Size on Node 0 $(nx \times ny \times nz)$ | Global Problem Size $(nx \times ny \times nz)$ | Scaled Speedup | Parallel Scaled Efficiency | Grind Time ($\mu$s/cell/ timestep) |
|---|---|---|---|---|---|
| 1 | $15 \times 10 \times 6$ | $15 \times 10 \times 6$ | 1.000 | 1.000 | 1390.773 |
| 2 | $8 \times 10 \times 12$ | $15 \times 10 \times 12$ | 1.754 | 0.877 | 793.134 |
| 4 | $8 \times 10 \times 12$ | $15 \times 20 \times 12$ | 3.338 | 0.834 | 416.671 |
| 8 | $8 \times 10 \times 12$ | $30 \times 20 \times 12$ | 6.385 | 0.798 | 217.803 |
| 16 | $8 \times 10 \times 12$ | $30 \times 20 \times 24$ | 12.30 | 0.769 | 113.075 |
| 32 | $8 \times 10 \times 12$ | $30 \times 40 \times 24$ | 23.50 | 0.734 | 59.172 |
| 64 | $8 \times 10 \times 12$ | $60 \times 40 \times 24$ | 46.68 | 0.729 | 29.791 |
| 128 | $8 \times 10 \times 12$ | $60 \times 40 \times 48$ | 90.60 | 0.708 | 15.350 |
| 256 | $8 \times 10 \times 12$ | $60 \times 80 \times 48$ | 175.4 | 0.662 | 7.928 |
| 512 | $8 \times 10 \times 12$ | $120 \times 80 \times 48$ | 347.2 | 0.678 | 4.006 |
| 1024 | $8 \times 10 \times 12$ | $120 \times 80 \times 96$ | 684.4 | 0.668 | 2.032 |
| 1 | $30 \times 20 \times 10$ | $30 \times 20 \times 10$ | 1.000 | 1.000 | 1143.038 |
| 2 | $30 \times 20 \times 10$ | $60 \times 20 \times 10$ | 1.922 | 0.961 | 594.577 |
| 4 | $30 \times 20 \times 10$ | $60 \times 40 \times 10$ | 3.792 | 0.948 | 301.452 |
| 8 | $15 \times 20 \times 20$ | $60 \times 40 \times 20$ | 6.891 | 0.861 | 165.868 |
| 16 | $15 \times 20 \times 20$ | $120 \times 40 \times 20$ | 13.56 | 0.848 | 84.277 |
| 32 | $15 \times 20 \times 20$ | $120 \times 80 \times 20$ | 26.13 | 0.817 | 43.738 |
| 64 | $15 \times 20 \times 20$ | $120 \times 80 \times 40$ | 51.83 | 0.810 | 22.504 |
| 128 | $15 \times 20 \times 20$ | $240 \times 80 \times 40$ | 99.96 | 0.781 | 11.435 |
| 256 | $15 \times 20 \times 20$ | $240 \times 160 \times 40$ | 195.7 | 0.765 | 5.840 |
| 512 | $15 \times 20 \times 20$ | $240 \times 160 \times 80$ | 376.2 | 0.735 | 3.038 |
| 1024 | $15 \times 20 \times 20$ | $480 \times 160 \times 80$ | 739.5 | 0.722 | 1.544 |
| 1824 | $20 \times 21 \times 16$ | $366 \times 244 \times 122$ | 1077. | 0.591 | 1.060 |

**Table 9: Performance of MIMD PAGOSA on the Intel Paragon for ew**

| Number of Nodes | Problem Size on Node 0 ($nx \times ny \times nz$) | Global Problem Size ($nx \times ny \times nz$) | Scaled Speedup | Parallel Scaled Efficiency | Grind Time ($\mu$s/cell/ timestep) |
|---|---|---|---|---|---|
| 1 | $14 \times 7 \times 7$ | $14 \times 7 \times 7$ | 1.000 | 1.000 | 1763.937 |
| 2 | $7 \times 7 \times 14$ | $14 \times 7 \times 14$ | 1.704 | 0.852 | 1034.936 |
| 4 | $7 \times 7 \times 14$ | $14 \times 14 \times 14$ | 3.190 | 0.798 | 552.989 |
| 8 | $7 \times 7 \times 14$ | $28 \times 14 \times 14$ | 5.872 | 0.734 | 300.403 |
| 16 | $7 \times 7 \times 14$ | $28 \times 14 \times 28$ | 11.39 | 0.712 | 154.833 |
| 32 | $7 \times 7 \times 14$ | $28 \times 28 \times 28$ | 21.56 | 0.674 | 81.816 |
| 64 | $7 \times 7 \times 14$ | $56 \times 28 \times 28$ | 42.14 | 0.658 | 41.857 |
| 128 | $7 \times 7 \times 14$ | $56 \times 28 \times 56$ | 80.97 | 0.633 | 21.785 |
| 256 | $7 \times 7 \times 14$ | $56 \times 56 \times 56$ | 160.0 | 0.625 | 11.022 |
| 512 | $7 \times 7 \times 14$ | $112 \times 56 \times 56$ | 320.5 | 0.626 | 5.504 |
| 1024 | $7 \times 7 \times 14$ | $112 \times 56 \times 112$ | 628.0 | 0.613 | 2.809 |
| 1 | $28 \times 15 \times 15$ | $28 \times 15 \times 15$ | 1.000 | 1.000 | 1637.87 |
| 2 | $28 \times 15 \times 15$ | $56 \times 15 \times 15$ | 1.889 | 0.945 | 866.883 |
| 4 | $28 \times 15 \times 15$ | $56 \times 30 \times 15$ | 3.678 | 0.920 | 445.250 |
| 8 | $28 \times 15 \times 15$ | $56 \times 30 \times 30$ | 7.063 | 0.883 | 231.886 |
| 16 | $28 \times 15 \times 15$ | $112 \times 30 \times 30$ | 13.80 | 0.862 | 118.708 |
| 32 | $28 \times 15 \times 30$ | $112 \times 60 \times 30$ | 26.76 | 0.836 | 61.201 |
| 64 | $28 \times 15 \times 15$ | $112 \times 60 \times 60$ | 52.62 | 0.822 | 31.128 |
| 128 | $28 \times 15 \times 15$ | $224 \times 60 \times 60$ | 103.6 | 0.809 | 15.815 |
| 256 | $28 \times 15 \times 15$ | $224 \times 120 \times 60$ | 204.1 | 0.797 | 8.023 |
| 512 | $28 \times 15 \times 30$ | $224 \times 120 \times 120$ | 397.0 | 0.775 | 4.097 |
| 1024 | $28 \times 15 \times 15$ | $448 \times 120 \times 120$ | 796.7 | 0.778 | 2.056 |
| 1824 | $28 \times 15 \times 15$ | $360 \times 180 \times 180$ | 1476. | 0.818 | 1.1096* |

* This simulation was run to only 1 microsecond.

**Table 10: Performance of MIMD PAGOSA on the Intel Paragon for owp**

| Number of Nodes | Problem Size on Node 0 $(nx \times ny \times nz)$ | Global Problem Size $(nx \times ny \times nz)$ | Scaled Speedup | Parallel Scaled Efficiency | Grind Time ($\mu$s/cell/ timestep) |
|---|---|---|---|---|---|
| 1 | $15 \times 6 \times 6$ | $15 \times 6 \times 6$ | 1.000 | 1.000 | 3021.62 |
| 2 | $8 \times 12 \times 6$ | $15 \times 12 \times 6$ | 1.689 | 0.845 | 1788.75 |
| 4 | $8 \times 6 \times 12$ | $15 \times 12 \times 12$ | 3.268 | 0.817 | 924.620 |
| 8 | $8 \times 6 \times 12$ | $30 \times 12 \times 12$ | 6.352 | 0.794 | 475.816 |
| 16 | $8 \times 6 \times 12$ | $30 \times 24 \times 12$ | 11.99 | 0.749 | 252.102 |
| 32 | $8 \times 6 \times 12$ | $30 \times 24 \times 24$ | 23.61 | 0.738 | 127.990 |
| 64 | $8 \times 6 \times 12$ | $60 \times 24 \times 24$ | 46.52 | 0.727 | 64.954 |
| 128 | $8 \times 6 \times 12$ | $60 \times 48 \times 24$ | 92.58 | 0.723 | 32.636 |
| 256 | $8 \times 6 \times 12$ | $60 \times 48 \times 48$ | 180.2 | 0.704 | 16.771 |
| 512 | $8 \times 6 \times 12$ | $120 \times 48 \times 48$ | 357.4 | 0.698 | 8.456 |
| 1024 | $8 \times 6 \times 12$ | $120 \times 96 \times 48$ | 714.7 | 0.698 | 4.228 |
| 1 | $30 \times 12 \times 12$ | $30 \times 12 \times 12$ | 1.000 | 1.000 | 2299.889 |
| 2 | $30 \times 12 \times 12$ | $60 \times 12 \times 12$ | 1.951 | 0.975 | 1178.958 |
| 4 | $15 \times 24 \times 12$ | $60 \times 24 \times 12$ | 3.716 | 0.929 | 618.891 |
| 8 | $15 \times 12 \times 24$ | $60 \times 24 \times 24$ | 7.198 | 0.900 | 319.516 |
| 16 | $15 \times 12 \times 24$ | $120 \times 24 \times 24$ | 14.21 | 0.888 | 161.886 |
| 32 | $15 \times 12 \times 24$ | $120 \times 48 \times 24$ | 27.33 | 0.854 | 84.148 |
| 64 | $15 \times 12 \times 24$ | $120 \times 48 \times 48$ | 54.26 | 0.848 | 42.384 |
| 128 | $15 \times 12 \times 24$ | $240 \times 48 \times 48$ | 107.3 | 0.838 | 21.422 |
| 256 | $15 \times 12 \times 24$ | $240 \times 96 \times 48$ | 212.2 | 0.829 | 10.836 |
| 512 | $15 \times 12 \times 24$ | $240 \times 96 \times 96$ | 414.5 | 0.810 | 5.548 |
| 1024 | $15 \times 12 \times 24$ | $480 \times 96 \times 96$ | 808.1 | 0.789 | 2.846 |
| 1824 | $20 \times 13 \times 19$ | $380 \times 156 \times 152$ | 1499. | 0.822 | 1.534 |

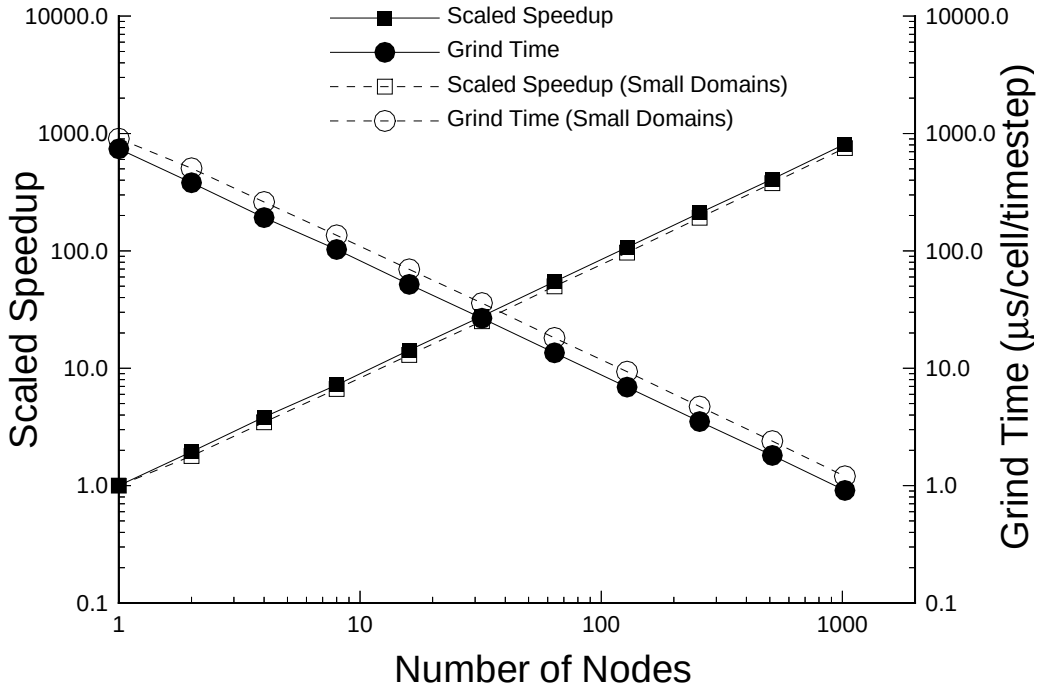



Figure 15. Scaled speedup and grind time for the fp1 simulation on the Paragon. The curves labeled “Small Domains” are for the subdomain size used on the nCUBE 2.

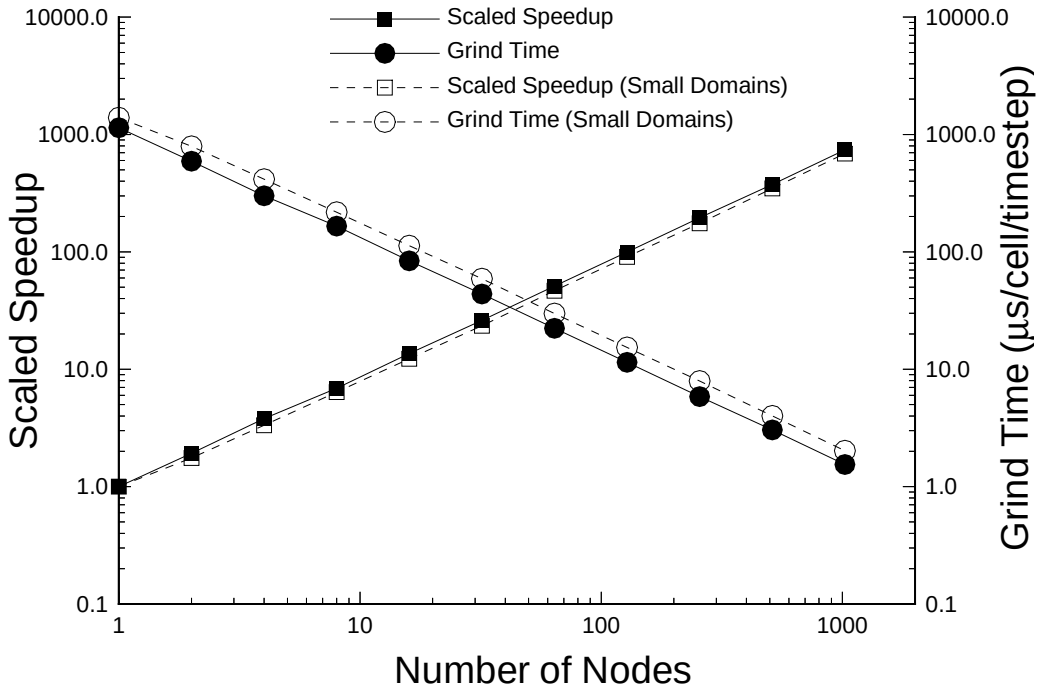



Figure 16. Scaled speedup and grind time for the fp2 simulation on the Paragon. The curves labeled “Small Domains” are for the subdomain size used on the nCUBE 2.

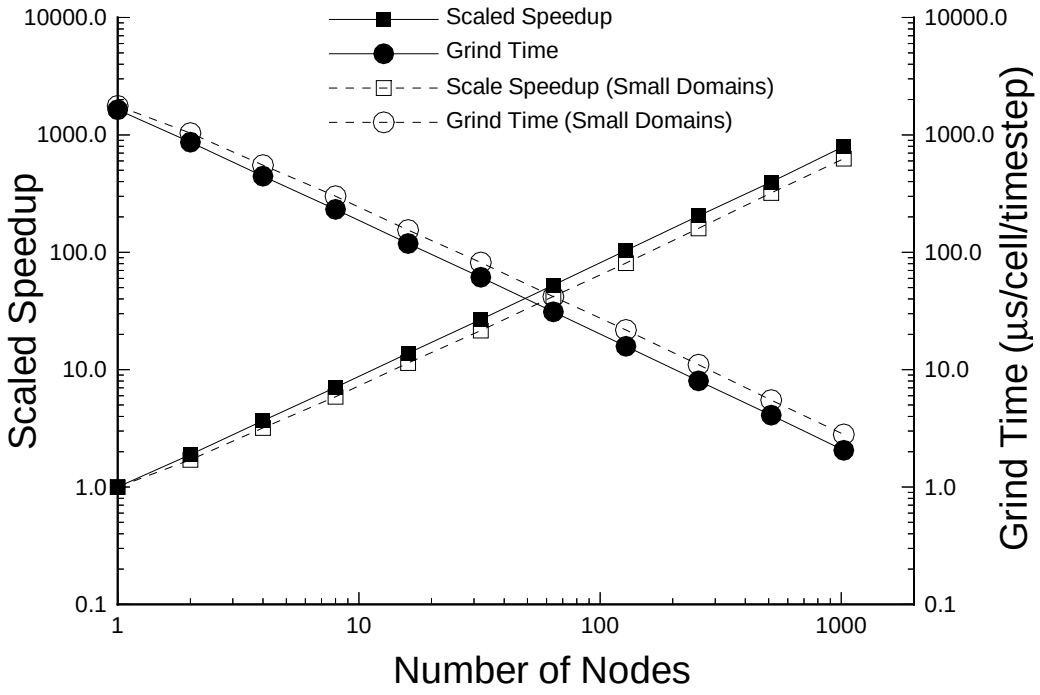



Figure 17. Scaled speedup and grind time for the ew simulation on the Paragon. The curves labeled “Small Domains” are for the subdomain size used on the nCUBE 2.

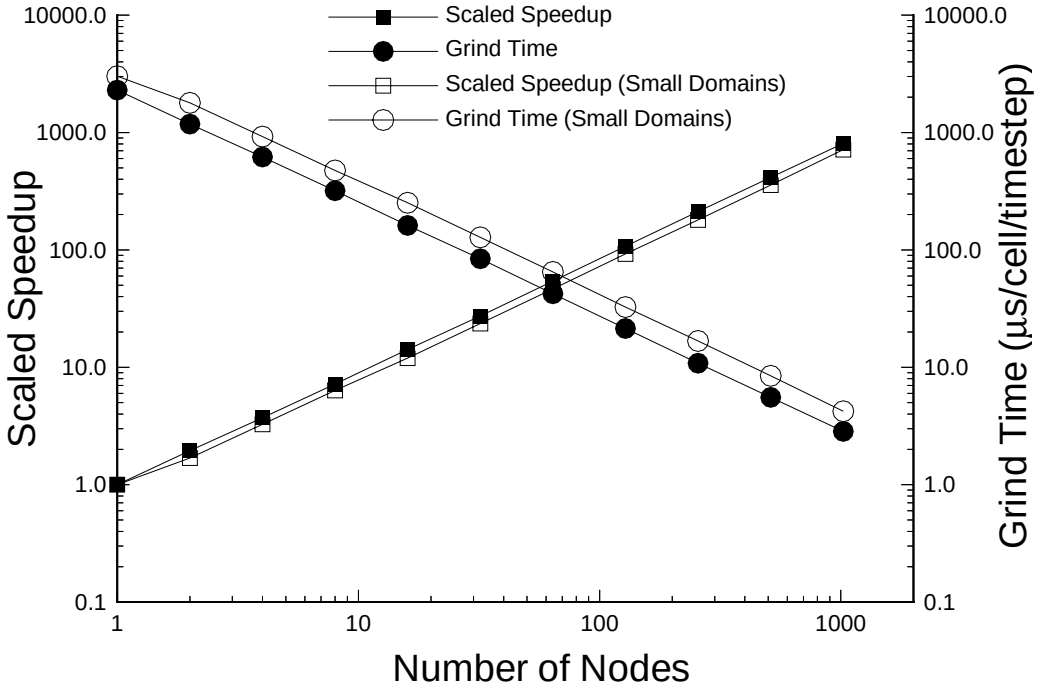



Figure 18. Scaled speedup and grind time for the owp simulation on the Paragon. The curves labeled “Small Domains” are for the subdomain size used on the nCUBE 2.

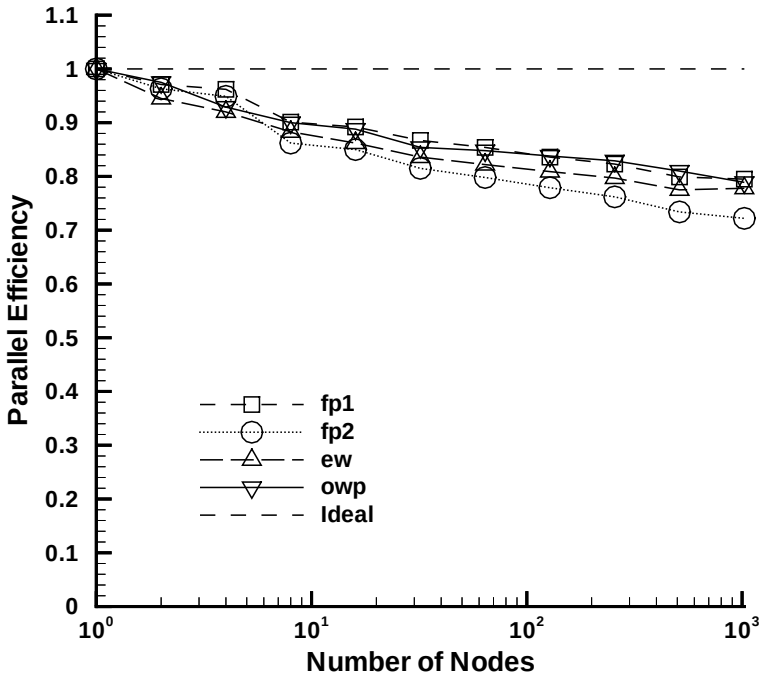



Figure 19. Scaled parallel efficiency on the Paragon.

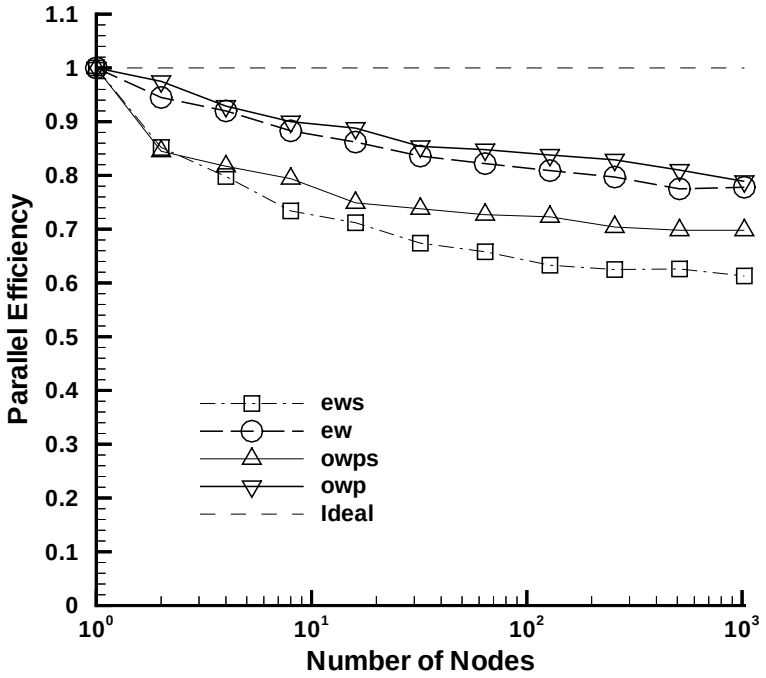



Figure 20. Effect of subdomain size on the scaled parallel efficiency on the Paragon for the ew and owp simulations. The points labeled “ews” and “owps” are for the ew and owp simulations, respectively, using the subdomain sizes used on the nCUBE 2

## Performance of MIMD PAGOSA on the nCUBE 2 and the Intel Paragon Message-Passing Computers

In this section we present a summary of the grind time, scaled speedup, and parallel scaled efficiency for MIMD PAGOSA running on the fp1, fp2, ew, and owp simulations for the same number of cells per node on each machine (Tables 11–14). We also present a summary of the grind time, scaled speedup, and parallel scaled efficiency for the largest problem solved on each computer (Table 15).

**Table 11: Performance of MIMD PAGOSA on the nCUBE 2 and the Intel Paragon for 18x12x7-Cell Subdomains for the Finned Penetrator Problem with No Material Strength (fp1)**

| Machine | Number of Nodes | Global Problem Size $(nx \times ny \times nz)$ | Grind Time ($\mu$s/cell/ timestep) | Scaled Speedup | Parallel Scaled Efficiency |
|---|---|---|---|---|---|
| nCUBE 2 | 1 | $18 \times 12 \times 7$ | 5755. | 1.000 | 1.000 |
| Paragon | 1 | $18 \times 12 \times 7$ | 903.7 | 1.000 | 1.000 |
| nCUBE 2 | 2 | $18 \times 12 \times 14$ | 3121. | 1.844 | 0.922 |
| Paragon | 2 | $18 \times 12 \times 14$ | 505.0 | 1.790 | 0.895 |
| nCUBE 2 | 4 | $18 \times 24 \times 14$ | 1617. | 3.558 | 0.890 |
| Paragon | 4 | $18 \times 24 \times 14$ | 261.0 | 3.463 | 0.866 |
| nCUBE 2 | 8 | $36 \times 24 \times 14$ | 832.9 | 6.909 | 0.864 |
| Paragon | 8 | $36 \times 24 \times 14$ | 135.5 | 6.668 | 0.834 |
| nCUBE 2 | 16 | $36 \times 24 \times 28$ | 428.5 | 13.43 | 0.839 |
| Paragon | 16 | $36 \times 24 \times 28$ | 69.55 | 12.99 | 0.812 |
| nCUBE 2 | 32 | $36 \times 48 \times 28$ | 219.4 | 26.23 | 0.820 |
| Paragon | 32 | $36 \times 48 \times 28$ | 35.92 | 25.16 | 0.786 |
| nCUBE 2 | 64 | $72 \times 48 \times 28$ | 110.6 | 52.04 | 0.813 |
| Paragon | 64 | $72 \times 48 \times 28$ | 18.14 | 49.83 | 0.778 |
| nCUBE 2 | 128 | $72 \times 48 \times 56$ | 56.21 | 102.4 | 0.800 |
| Paragon | 128 | $72 \times 48 \times 56$ | 9.350 | 96.66 | 0.755 |
| nCUBE 2 | 256 | $72 \times 96 \times 56$ | 28.59 | 201.3 | 0.786 |
| Paragon | 256 | $72 \times 96 \times 56$ | 4.723 | 191.4 | 0.748 |
| nCUBE 2 | 512 | $144 \times 96 \times 56$ | 14.47 | 397.7 | 0.777 |
| Paragon | 512 | $144 \times 96 \times 56$ | 2.396 | 377.2 | 0.737 |
| nCUBE 2 | 1024 | $144 \times 96 \times 112$ | 7.288 | 789.6 | 0.771 |
| Paragon | 1024 | $144 \times 96 \times 112$ | 1.202 | 751.9 | 0.734 |

**Table 12: Performance of MIMD PAGOSA on the nCUBE 2 and the Intel Paragon for 15x10x6-Cell Subdomains for Finned Penetrator Problem with Material Strength (fp2)**

| Machine | Number of Nodes | Global Problem Size $(nx \times ny \times nz)$ | Grind Time (μs/cell/ timestep) | Scaled Speedup | Parallel Scaled Efficiency |
|---|---|---|---|---|---|
| nCUBE 2 | 1 | $15 \times 10 \times 6$ | 9384. | 1.000 | 1.000 |
| Paragon | 1 | $15 \times 10 \times 6$ | 1391. | 1.000 | 1.000 |
| nCUBE 2 | 2 | $15 \times 10 \times 12$ | 5169. | 1.816 | 0.908 |
| Paragon | 2 | $15 \times 10 \times 12$ | 793.1 | 1.754 | 0.877 |
| nCUBE 2 | 4 | $15 \times 20 \times 12$ | 2705. | 3.469 | 0.867 |
| Paragon | 4 | $15 \times 20 \times 12$ | 416.7 | 3.338 | 0.834 |
| nCUBE 2 | 8 | $30 \times 20 \times 12$ | 1411. | 6.653 | 0.832 |
| Paragon | 8 | $30 \times 20 \times 12$ | 217.8 | 6.385 | 0.769 |
| nCUBE 2 | 16 | $30 \times 20 \times 24$ | 731.9 | 12.82 | 0.801 |
| Paragon | 16 | $30 \times 20 \times 24$ | 113.1 | 12.30 | 0.769 |
| nCUBE 2 | 32 | $30 \times 40 \times 24$ | 378.1 | 24.82 | 0.776 |
| Paragon | 32 | $30 \times 40 \times 24$ | 59.17 | 23.50 | 0.734 |
| nCUBE 2 | 64 | $60 \times 40 \times 24$ | 190.9 | 49.17 | 0.768 |
| Paragon | 64 | $60 \times 40 \times 24$ | 29.79 | 46.68 | 0.729 |
| nCUBE 2 | 128 | $60 \times 40 \times 48$ | 97.76 | 95.99 | 0.750 |
| Paragon | 128 | $60 \times 40 \times 48$ | 15.35 | 90.60 | 0.708 |
| nCUBE 2 | 256 | $60 \times 80 \times 48$ | 50.66 | 185.3 | 0.724 |
| Paragon | 256 | $60 \times 80 \times 48$ | 7.928 | 175.4 | 0.662 |
| nCUBE 2 | 512 | $120 \times 80 \times 48$ | 25.69 | 365.2 | 0.713 |
| Paragon | 512 | $120 \times 80 \times 48$ | 4.006 | 347.2 | 0.678 |
| nCUBE 2 | 1024 | $120 \times 80 \times 96$ | 13.03 | 720.3 | 0.703 |
| Paragon | 1024 | $120 \times 80 \times 96$ | 2.032 | 684.4 | 0.668 |

**Table 13: Performance of MIMD PAGOSA on the nCUBE 2 and the Intel Paragon for 14x7x7-Cell Subdomains for the Explosive Welding Problem (ew)**

| Machine | Number of Nodes | Global Problem Size $(nx \times ny \times nz)$ | Grind Time ($\mu$s/cell/ timestep) | Scaled Speedup | Parallel Scaled Efficiency |
|---|---|---|---|---|---|
| nCUBE 2 | 1 | $14 \times 7 \times 7$ | 11475. | 1.000 | 1.000 |
| Paragon | 1 | $14 \times 7 \times 7$ | 1764. | 1.000 | 1.000 |
| nCUBE 2 | 2 | $14 \times 7 \times 14$ | 6453. | 1.778 | 0.889 |
| Paragon | 2 | $14 \times 7 \times 14$ | 1035. | 1.704 | 0.852 |
| nCUBE 2 | 4 | $14 \times 14 \times 14$ | 3466. | 3.311 | 0.828 |
| Paragon | 4 | $14 \times 14 \times 14$ | 553.0 | 3.190 | 0.798 |
| nCUBE 2 | 8 | $28 \times 14 \times 14$ | 1860. | 6.170 | 0.771 |
| Paragon | 8 | $28 \times 14 \times 14$ | 300.4 | 5.872 | 0.734 |
| nCUBE 2 | 16 | $28 \times 14 \times 28$ | 968.5 | 11.85 | 0.740 |
| Paragon | 16 | $28 \times 14 \times 28$ | 154.8 | 11.39 | 0.712 |
| nCUBE 2 | 32 | $28 \times 28 \times 28$ | 510.2 | 22.49 | 0.703 |
| Paragon | 32 | $28 \times 28 \times 28$ | 81.82 | 21.56 | 0.674 |
| nCUBE 2 | 64 | $56 \times 28 \times 28$ | 259.1 | 44.28 | 0.692 |
| Paragon | 64 | $56 \times 28 \times 28$ | 41.86 | 42.14 | 0.658 |
| nCUBE 2 | 128 | $56 \times 28 \times 56$ | 133.5 | 85.97 | 0.672 |
| Paragon | 128 | $56 \times 28 \times 56$ | 21.78 | 80.97 | 0.633 |
| nCUBE 2 | 256 | $56 \times 56 \times 56$ | 67.76 | 169.3 | 0.662 |
| Paragon | 256 | $56 \times 56 \times 56$ | 11.02 | 160.0 | 0.625 |
| nCUBE 2 | 512 | $112 \times 56 \times 56$ | 34.01 | 337.4 | 0.659 |
| Paragon | 512 | $112 \times 56 \times 56$ | 5.504 | 320.5 | 0.626 |
| nCUBE 2 | 1024 | $112 \times 56 \times 112$ | 17.40 | 959.6 | 0.644 |
| Paragon | 1024 | $112 \times 56 \times 112$ | 2.809 | 628.0 | 0.613 |

**Table 14: Performance of MIMD PAGOSA on the nCUBE 2 and the Intel Paragon for 15x6x6-Cell Subdomains for the Oil-Well Perforation Problem (owp)**

| Machine | Number of Nodes | Global Problem Size $(nx \times ny \times nz)$ | Grind Time (μs/cell/ timestep) | Scaled Speedup | Parallel Scaled Efficiency |
|---|---|---|---|---|---|
| nCUBE 2 | 1 | $15 \times 6 \times 6$ | 18449.366 | 1.000 | 1.000 |
| Paragon | 1 | $15 \times 6 \times 6$ | 3021.62 | 1.000 | 1.000 |
| nCUBE 2 | 2 | $15 \times 12 \times 6$ | 10168.498 | 1.814 | 0.907 |
| Paragon | 2 | $15 \times 12 \times 6$ | 1788.75 | 1.689 | 0.845 |
| nCUBE 2 | 4 | $15 \times 12 \times 12$ | 5469.947 | 3.373 | 0.843 |
| Paragon | 4 | $15 \times 12 \times 12$ | 924.620 | 3.268 | 0.817 |
| nCUBE 2 | 8 | $30 \times 12 \times 12$ | 2824.440 | 6.532 | 0.816 |
| Paragon | 8 | $30 \times 12 \times 12$ | 475.816 | 6.350 | 0.794 |
| nCUBE 2 | 16 | $30 \times 24 \times 12$ | 1482.658 | 12.44 | 0.778 |
| Paragon | 16 | $30 \times 24 \times 12$ | 252.102 | 11.99 | 0.749 |
| nCUBE 2 | 32 | $30 \times 24 \times 24$ | 763.337 | 24.17 | 0.755 |
| Paragon | 32 | $30 \times 24 \times 24$ | 127.990 | 23.61 | 0.738 |
| nCUBE 2 | 64 | $60 \times 24 \times 24$ | 388.899 | 47.44 | 0.741 |
| Paragon | 64 | $60 \times 24 \times 24$ | 64.954 | 46.52 | 0.727 |
| nCUBE 2 | 128 | $60 \times 48 \times 24$ | 196.445 | 93.92 | 0.734 |
| Paragon | 128 | $60 \times 48 \times 24$ | 32.636 | 92.58 | 0.723 |
| nCUBE 2 | 256 | $60 \times 48 \times 48$ | 100.168 | 184.2 | 0.720 |
| Paragon | 256 | $60 \times 48 \times 48$ | 16.771 | 180.2 | 0.704 |
| nCUBE 2 | 512 | $120 \times 48 \times 48$ | 50.635 | 364.4 | 0.712 |
| Paragon | 512 | $120 \times 48 \times 48$ | 8.456 | 357.3 | 0.698 |
| nCUBE 2 | 1024 | $120 \times 96 \times 48$ | 25.312 | 728.9 | 0.712 |
| Paragon | 1024 | $120 \times 96 \times 48$ | 4.228 | 714.7 | 0.698 |

**Table 15: Summary of Maximum Simulation Sizes for the nCUBE 2 and the Paragon**

| Machine | Problem | Subdomain Size ($nx \times ny \times nz$) | Global Domain Size ($nx \times ny \times nz$) | No. of Nodes | Scaled Speed-up | Grind Time ($\mu$s/cell/ timestep) |
|---|---|---|---|---|---|---|
| nCUBE 2 | fp1* | $9 \times 12 \times 14$ | $144 \times 96 \times 112$ | 1024 | 789.6 | 7.3 |
| Paragon | fp1 | $24 \times 25 \times 19$ | $438 \times 292 \times 146$ | 1824 | 1353. | 0.55 |
| nCUBE 2 | fp2† | $8 \times 10 \times 12$ | $120 \times 80 \times 96$ | 1024 | 720.3 | 13.0 |
| Paragon | fp2 | $20 \times 21 \times 16$ | $366 \times 244 \times 122$ | 1824 | 1077. | 1.06 |
| nCUBE 2 | ew‡ | $7 \times 7 \times 14$ | $112 \times 56 \times 112$ | 1024 | 959.6 | 17.4 |
| Paragon | ew | $28 \times 15 \times 15$ | $360 \times 180 \times 180$ | 1824 | 1476. | 1.1 |
| nCUBE 2 | owp** | $8 \times 6 \times 12$ | $120 \times 96 \times 48$ | 1024 | 728.9 | 25.3 |
| Paragon | owp | $20 \times 13 \times 19$ | $380 \times 156 \times 152$ | 1824 | 1499. | 1.5 |

* Finned penetrator problem with no material strength.
† Finned penetrator problem with material strength.
‡ Explosive welding problem.
** Oil-well perforation problem.

## Summary and Conclusions

In this report we have presented the measured performance of the MIMD PAGOSA shock-wave physics code, developed at Sandia for MIMD parallel computers using a message-passing programming model, on the nCUBE 2 and the Intel Paragon. The original PAGOSA code was developed at Los Alamos National Laboratory for single-instruction, multiple data (SIMD) computers using a data-parallel programming model.

The performance of MIMD PAGOSA[1] on the various machines was measured in terms of the scaled speedup, the parallel scaled efficiency, and the grind time (execution time per computation cell per timestep). Scaled speedup tests (in which the problem size was increased in proportion to the number of computational nodes) were conducted for each parallel computer for four test problems: a finned projectile problem with no material strength (fp1, the minimal test problem), a finned projectile problem with material strength (fp2), the explosive welding of a copper tube to a steel plate (ew), and the perforation of an oil-well casing by shaped charge jets (owp). These represent increasingly complex and realistic engineering simulations. Summary tables (Tables 11–14) present the performance on the two machines on identical problem sizes. A summary table (Table 15) presents the largest calculation of each of the test problems on each machine.

The parallel scaled efficiencies were greater than 0.70 when the available memory per node was filled (or nearly filled). The linear variation of the scaled speedup and grind time with the number of computational nodes for each machine (Figures 8–12 for the nCUBE 2 and Figures 15–18 for the Paragon) demonstrate that MIMD PAGOSA is scalable on both the nCUBE 2 and the Paragon to large numbers of computational nodes (1024 and 1824, respectively) for a variety of problems, from simple problems to real-world problems. The scalability is maintained even if the memory of the computational node is not filled.

Not surprisingly, adding various material models, such as material strength and high-explosive burn, affects the code performance by increasing the memory required (for storing the model variables) and increasing the grind time (because the models must be evaluated). For example, for the nCUBE 2, adding the material strength models reduces the maximum number of cells per computational node by a factor of approximately 0.60 (from 1512 cells per node for fp1 to 960 cells per node for fp2) and increases the grind time by approximately a factor of 1.63 (one node) to 1.79 (1024 nodes). For the Paragon, adding the material strength models reduces the maximum number of cells per computational node by a factor of approximately 0.57 (from 10368 cells per node for fp1 to 6000 cells per node for fp2) and increases the grind time by approximately a factor of 1.54 (one node) to 1.93 (1824 nodes).

---

1. The performance data presented in this report represent only one aspect of the performance of each machine. While they address fundamental issues of computational and communication speeds, other issues must also be considered when evaluating computers, including the ease of sharing the machine among several users, the functionality of the operating system, the availability of graphical output devices and the ease of their use, the purchase price, and the maintenance costs. None of the machines examined in this report should be accepted or rejected based only on the data presented in this report. We have deliberately provided only enough interpretation to allow the reader to understand the results.

In Reference 10 we reported the performance of a simplified, single-material version of MIMD PAGOSA (smPAGOSA) on a variety of parallel computers and on various Cray vector supercomputers (viz., a Y-MP8E, a Y-MP8I and a Cray C90) for a three-dimensional, spherical blast wave problem (sbw). In Table 16 we compare the largest calculations with smPAGOSA on single processors of the Y-MP and C90 with similar calculations with MIMD PAGOSA on the nCUBE 2 and the Paragon. For the comparison, we list the calculations with MIMD PAGOSA which had both a similar number of cells, and a similar grind time. Since MIMD PAGOSA is a much more complex code than smPAGOSA (e.g., it reconstructs material interfaces), and since the fp1 problem involves three materials while the spherical blast wave involves only one, the comparison in the table favors smPAGOSA running on the Cray vector computers. The point of this comparison is that large shock-wave physics simulations can be run on MIMD supercomputers at speeds and with problem sizes equalling or exceeding those achievable with vector supercomputers. In particular, the last row of Table 16 illustrates that much larger problems can be run significantly faster on the MIMD supercomputers than on traditional vector supercomputers.

Thus large, real-world, shock-wave physics simulations can be performed on parallel computers with sizes and speeds that equal or greatly exceed those for a single processor of a Cray Y-MP, and scalable shock-wave physics codes can be developed to run on hundreds to a thousand or more computational nodes.

**Table 16: Comparisons of Calculations on the Cray Y-MP, Cray C90, nCUBE 2, and Intel Paragon**

| Machine | Code | Problem | No. of Nodes | Total Number of Cells | Grind Time (μs/cell/ timestep) |
|---|---|---|---|---|---|
| nCUBE 2 | MIMD PAGOSA | fp1 | 1024 | 1,548,288 | 7.3 |
| Cray Y-MP8I | smPAGOSA* | sbw† | 1 | 2,097,152 | 8.6 |
| Paragon | MIMD PAGOSA | fp1 | 256 | 2,654,208 | 3.6 |
| Cray Y-MP8I | smPAGOSA | sbw | 1 | 2,097,152 | 8.6 |
| Paragon | MIMD PAGOSA | fp1 | 128 | 1,327,104 | 6.9 |
| Paragon | MIMD PAGOSA | fp1 | 256 | 2,654,208 | 3.6 |
| Cray C90 | smPAGOSA | sbw | 1 | 4,194,304 | 3.1 |
| Paragon | MIMD PAGOSA | fp1 | 512 | 5,308,416 | 1.8 |
| Paragon | MIMD PAGOSA | fp1 | 1824 | 18,672,816 | 0.55 |

* Single-material version of PAGOSA.
† Three-dimensional, single-material, spherical blast wave problem.

# References


1. D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan and S. K. Weeratunga, The NAS Parallel Benchmarks, in *International. Journal of Supercomputer Applications,* vol. 5, pp. 63-73, 1991.

2. Welding of Tubular, Rod, and Special Assemblies, in *Explosive Welding, Forming, and Compaction*, T. Z. Blazynski, ed. Applied Science Publishers, London, 1983

3. D. J. Cagliostro, D. A. Mandell, L. A. Schwalbe, T. F. Adams and E. J. Chapyak, MESA 3-D Calculations of Armor Penetration by Projectiles with Combined Obliquity and Yaw, in *International Journal of Impact Engineering*, vol. 10, pp. 81–92, 1990.

4. D. D. Cline and J. A. Schutt, Three-Dimensional Advection on a Massively Parallel Hypercube. Fourth International Symposium on Computational Fluid Dynamics, University of California at Davis, September 9–12,1991.

5. J. J. Dongarra, J. Bunch, C. Moler and G. W. Stewart, *LINPACK User's Guide*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1979.

6. J. J. Dongarra, *Performance of Various Computers Using Standard Linear Equations Software,* Report CS-89-85. Computer Science Department, University of Tennessee, Knoxville, TN, March 1994.

7. H. E. Fang and A. C. Robinson, A. C., 1993, 3-D Massively Parallel Impact Simulations Using PCTH, in *Proceedings of the 1993 Summer Computer Simulation Conference*, held in Boston, MA, July 19–21, 1993. The Society for Computer Simulation, San Diego, CA, 1993.

8. H. E. Fang, C. T. Vaughan, and D. R. Gardner, Performance Issues for Engineering Analysis on MIMD Parallel Computers, in *International Mechanical Engineering Congress and Exposition '94*, held in Chicago, IL, November 6–11, 1994. ASME, Fairfield, NJ, 1994.

9. D. R. Gardner, D. D. Cline, C. T. Vaughan, *Implementation of a Single-Material Version of PAGOSA on MIMD Hypercubes*, SAND92-0640. Sandia National Laboratories, Albuquerque, NM, June 1992.

10. D. R. Gardner, D. D. Cline, C. T. Vaughan, *The Performance of PAGOSA on Several MIMD Massively Parallel Computers*. Sandia National Laboratories report SAND92-1880C. 1992 Nuclear Explosives Code Developers Conference held in Sunnyvale, CA, November 2–6, 1992.

11. D. R. Gardner, D. D. Cline, C. T. Vaughan, R. Krall and M. Lewitt, The Performance of PAGOSA on Several SIMD and MIMD Parallel Computers, SAND92-1452, Sandia National Laboratories, Albuquerque, NM, October 1992.

12. D. R. Gardner and H. E. Fang, 1992, “Three-Dimensional Shock Wave Simulations on Massively Parallel Supercomputers, in *Proceedings of the 1992 Summer Computer Simulation Conference*, held in Reno, NV, July 27–30, 1992. The Society for Computer Simulation, San Diego, CA, 1992.

13. D. R. Gardner and H. E. Fang, 1994, Three-dimensional Shock Wave Physics Simulations with PCTH on the Paragon Parallel Computer, in *Proceedings of the 1994 Simulation Multiconference, High Performance Computing Symposium 1994—Grand Challenges in Computer Simulation*, held in La Jolla, CA, April 11–13, 1994. The Society for Computer Simulation, San Diego, CA, 1994.

14. J. L. Gustafson, G. R. Montry and R. E. Benner, Development of Parallel Methods for a 1024-Processor Hypercube, in *SIAM Journal on Scientific and Statistical Computing*, vol. 9, pp. 609-638, 1988.

15. E. S. Hertel, Jr.; R. L. Bell, M. G. Elrick, A. V. Farnsworth, G. I. Kerley, J. M. McGlaun, S. V. Petney, S. A. Silling, P. A. Taylor and L. Yarrington, CTH: A Software Family for Multi-Dimensional Shock Physics Analysis, in *Proceedings, 19th International Symposium on Shock Waves*, held in Provence, France, 1993, Université de Provence, Provence, France, vol 1., 1993.

16. D. B. Kothe, J. R. Baumgardner, J. H. Cerutti, B. J. Daly, K. S. Holian, E. M. Kober, S. J. Mosso, J. W. Painter. R. D. Smith and M. D. Torrey, PAGOSA: A Massively Parallel, Multi-Material Hydrodynamics Model for Three-Dimensional High-Speed Flow and High-Rate Material Deformation, in *Proceedings of the 1993 SCS Simulation Multiconference*, held in Arlington, VA, March 29–April 1 1993. The Society for Computer Simulation, San Diego, CA, 1993.

17. K. S. Holian, D. A. Mandell, T. F. Adams, F. L. Addessio, J. R. Baumgardner and S. J. Mosso, MESA: A 3-D Computer Code for Armor/Anti-Armor Applications. Supercomputing World Conference, San Francisco, October 17–20, 1989.

18. J. M. McGlaun and S. L. Thompson, CTH: A Three-Dimensional Shock Wave Physics Code, *International Journal of Impact Engineering*, vol. 10, pp. 351-360, 1990.

19. A. C. Robinson, C. T. Vaughan, H. E. Fang, C. F. Diegert, C. F. and K.-S. Cho, Hydrocode Development on the nCUBE and the Connection Machine Hypercubes, in *Shock Compression of Condensed Matter 1991*, S. C. Schmidt, R. D. Dick, J. W. Forbes, D. G. Tasker, ed., Elsevier Science Publishers B.V., Amsterdam, The Netherlands, pp. 289-292, 1991.

20. S. R. Wheat, A. B. Maccabe, R. Riesen, D. W. van Dresser and T. M. Stallcup, PUMA: An Operating System for Massively Parallel Systems, in *Proceedings of the 27th Annual Hawaii International Conference on System Sciences*, held in Kihei, HI, January 4–7, 1994. IEEE Computer Society Press, Los Alamitos, CA, vol. 2, pp. 56-65, 1994.

# Appendices

## A MIMD PAGOSA 5.5 Input Guide

PAGOSA Input Summary
Version 5.5
26 Feb 1993 and July 1995

John Cerutti, X-3, MS F663, ph. (505) 667-0738
Jay Mosso, X-3, MS F663, ph. (505) 667-4276
Tom Bennion, X-4, MS F664, ph. (505) 667-1721
Doug Kothe, T-3, MS B216, ph. (505) 667-9089
Martin Torrey, T-3, MS B216, ph. (505) 667-0976
Ed Kober, T-14, MS B214, ph. (505) 667-5140
Los Alamos National Laboratory

David R. Gardner, 9221, MS 1111, ph. (505) 845-7875
Courtenay T. Vaughan, 9226, MS 1109, ph. (505) 845-7277
Sandia National Laboratories

This brief input summary has been provided for user reference in lieu of any other existing documentation. When a set of documentation exists, this abbreviated input summary will probably not be maintained nor available. Text which applies to only the SIMD version of PAGOSA 5.5 is printed in an *italic font*. Text which applies to only the MIMD version of PAGOSA 5.5 is printed in a **bold font**. Text which applies to both SIMD and MIMD PAGOSA 5.5 is printed in normal font.

The input for both the main code and GEN, the generator code, is the same. They do not use separate input files, but rather different parts of the same input file. For the more simple problems that will run without restarts, etc., this means that separate input files need not be maintained for each program. For more complicated runs, the user will want to have separate input files for each phase of the run, but the format, for now, is the same.

Some input examples are provided under the “examples” directory and others may be found with the test problems under the “tp” directory.

Both GEN and PAGOSA have sets of empty “user_mods” routines which may be programmed to perform special functions. For the generator, for example, this may be to initialize a space depending on density and energy distribution.

All of the user input, except for the first line, is by means of Fortran namelists. The first line is always the TITLE, which may be blank, or a descriptive title which will be used to label all output. It may be up to 100 characters long.

Each namelist block of variables is defined below with a short description. All of the namelists are not used in both GEN and the main code PAGOSA. The main code does not use the GEN or BODY namelists, and GEN does not use the TRACERS and DETS namelists. However, this is changing,

as for example, GEN takes on more functionality such as calculating programmed burn times. There is no order to the namelists and they can be given in any order since the file is logically rewound before each type of namelist read. The order given below is simply one recommended order. Many users like to have the MESH, OPTIONS and OUTPUTS given first for easy access, then the MATS, DETS, and TRACERS next with the GEN and BODY namelists last. Use the order which best accommodates the way you work. The MESH and MATS namelists are used to set the problem dimensions.

*To run SIMD GEN on the Connection Machine, add the PAGOSA path to your PATH or else access it explicitly, attach to the Connection Machine and type:*

*gen input-prefix*

*where "input-prefix" is the prefix of the input file (described below). To look at a previous GEN run, a second argument, an input dump file name on the Data Vault, will also be requested. More on this later too.*

*To run SIMD PAGOSA, do the same with the path, attach to the Connection Machine and type:*

*pagosa input-prefix restart-dump*

*where "input-prefix" is the prefix of the input file (described below) and "restart-dump" is the name of the restart Data Vault file. When starting from scratch with the GEN output, this file will usually be named "input-prefix.dump.0". When running SIMD GEN or the SIMD PAGOSA main code, if the input arguments are omitted, it will then ask for them interactively. When running non-interactively, this will cause the run to terminate.*

**To run MIMD GEN on the Intel Paragon running the SUNMOS operating system, add the PAGOSA path to your PATH or else access it explicitly and type:**

**yod -sz N gen input_prefix**

**where "N" is the number of nodes to be used and "input-prefix" is the prefix of the input file (described below).**

**To run MIMD PAGOSA on the Intel Paragon running the SUNMOS operating system, use the same path, and type:**

**yod -sz N pagosa input-prefix**

**where "N" is the number of nodes to be used and "input-prefix" is the prefix of the input file (described below). The restart dump number, if any, is specified in the input file (described below). When running MIMD GEN or MIMD PAGOSA, if the input arguments are omitted, the code will ask for them interactively. When running non-interactively, omitting the input arguments will cause the run to terminate.**

The input file must have the suffix ".inp" as the trailing part of the file name. The prefix part of the file name (before the ".inp") is also used to name all of the output files. **For MIMD PAGOSA, the graphics and restart output files may be named independently using options in the OUT-**

**PUTS and RESTARTS namelist blocks.** Thus, the input "prefix" part is all that is needed by GEN and the main program and the ".inp" suffix is optional when the user input file name is requested. *Both SIMD GEN and the SIMD main program also may request an input dump file name, a file on the Data Vault. This is required when restarting a run, or using SIMD GEN to view the volume fractions of a previous SIMD GEN or PAGOSA run.*

The input and output files now used or produced are described below. Most have the same prefix (obtained from the user input file) and the suffix is used for reference and differentiation. With the same prefix, the normal directory list commands will display them together.

GEN and PAGOSA Input and Output Files

| File Name | Description |
|---|---|
| stdin | Program run/prefix information. (Fortran unit *) |
| stdout | Program status information. (Fortran unit *) |
| stderr | System error status information. |
| prefix.inp | Standard code input. (Fortran unit 1) |
| prefix.out | Standard code list output. This includes regular cycle prints, mesh & material summaries, short and long edits, etc. (Fortran unit 2) |
| prefix.err | Code errors (also usually echoed in prefix.out). (Fortran unit 3) |
| prefix.aux | Auxiliary output, usually special user output. (Fortran unit 4, not available in GEN) |
| prefix.trc | Massless tracer particle output. (Fortran unit 7, not available in GEN) |
| *prefix.dump.0* | *SIMD GEN output dump file, used for starting PAGOSA. (Fortran unit 10: GEN write, and unit 9: PAGOSA read)* |
| **prefix.nnnn.0** | **MIMD GEN output dump file, used for starting PAGOSA. 'nnnn' is the node number. (Fortran unit 10: GEN write, and unit 9: PAGOSA read)** |
| prefix.dump.1 & prefix.dump.2 | Alternating restart dump files as requested by user. (Fortran unit 10) |
| **prefix.nnnn.1 & prefix.nnnn.2** | **Alternating restart dump files as requested by user. 'nnnn' is the node number. (Fortran unit 10)** |
| **prefix.nnnn.trm** | **Final restart dump file as requested by user. 'nnnn' is the node number. (Fortran unit 10)** |
| *prefix.gd* | *Graphics dumps as requested by the user. (Fortran unit 8: write and unit 11: read)* |

GEN and PAGOSA Input and Output Files (Continued)

| File Name | Description |
|---|---|
| **prefix.nnnn.init** | **Graphics dump initialization file. 'nnnn' is the node number.** |
| **prefix.nnnn.N** | **Graphics dump material file. 'nnnn' is the node number. N is the material number (0,1,…)** |

## MESH Namelist Variables
## (Mesh Specification)

The MESH namelist input is used to define the Eulerian (fixed) computational mesh. This is done by specifying mesh segments where each segment has a constant mesh interval or a geometric mesh interval defined by a constant expansion or contraction ratio. Simply specify the number of cells in each mesh segment, the coordinates of the mesh segments, and the ratio for each. A ratio of 1.0 (or 0.0) specifies a constant zone size (no geometric expansion or contraction). Use the utility tool "grid" to help find geometric ratios.

MESH Variables

| Variable | Default | Description |
|---|---|---|
| ncellx | - | Number of cells in each x-direction mesh segment |
| coordx | - | Coordinates of the x-direction mesh segment |
| ratiox | - | Geometric ratio of each x-direction mesh segment |
| ncelly | - | Number of cells in each y-direction mesh segment |
| coordy | - | Coordinates of the y-direction mesh segment |
| ratioy | - | Geometric ratio of each y-direction mesh segment |
| ncellz | - | Number of cells in each z-direction mesh segment |
| coordz | - | Coordinates of the z-direction mesh segment |
| ratioz | - | Geometric ratio of each z-direction mesh segment |

## OPTIONS Namelist Variables
## (Run-time Parameters)

The OPTIONS namelist input is used to specify the "run-time" parameters. These are the cutoffs, safety factors, etc.

OPTIONS Variables

| Variable | Default | Description |
|---|---|---|
| cutacc | 1.0e-9 | Acceleration cutoff |
| cutd | 1.0e-6 | Density cutoff (alias: cutrho) |
| cutvof | 1.0e-5 | Volume fraction cutoff |
| cutrecon | 0.0 | Recon volume fraction cutoff to make parallel |
| zeps | 1.0e-15 | Relative “zero” epsilon (alias: alittle) |
| safec | 0.75 | Safety factor for Courant timestep |
| safed | 0.25 | Safety factor for divergence timestep (alias: safediv) |
| safeu | 0.25 | Safety factor for u velocity timestep (aliases: safevel, safev, safew) |
| id_geom | 3 | geometry id (1: 2D-cart, 2: 2D-cyl, 3: 3D-cart) (alias: idgeom) |
| id_q | 1 | artificial viscosity id (alias: idartvis, idq)<br>1: Wilkins<br>2: Std div with l**2 calculated in the direction of pressure gradient<br>3: Std div with l**2 = diagonal<br>4: Std div with l**2 = dx**2<br>5: Std div with l**2 = dy**2<br>6: Std div with l**2 = dz**2<br>7: Std div with l*2=min(dx**2,dy**2,dz**2)<br>8: Std div with l**2 = max(dx**2,dy**2,dz**2)<br>9: Std div with l**2 = included in CQ1 and CQ2 by user |
| tensor_q | .false. | Enables a tensor form for the artificial viscosity instead of the standard scalar form. The id_q input variable also has meaning because it controls the form of the coefficient in front of the tensor. Dimensionless coefficients cq1 and cq2 are needed as well (the default values are usually acceptable). |
| cq1 | 0.2 | Dimensionless coefficient for the linear artificial viscosity term. (alias: art1) |

OPTIONS Variables (Continued)

| Variable | Default | Description |
|---|---|---|
| cq2 | 2.0 | Dimensionless coefficient for the quadratic artificial viscosity term. (alias: art2) |
| ibc | 6*0 | Boundary conditions<br>(Xmin, Xmax, Ymin, Ymax, Zmin, Zmax)<br>0:rigid/reflective<br>1:xmit-out<br>2:vacuum |
| t | 0.0 | Current time<br>(usually set by the code) |
| dth | 1.0e+10 | Current time step size<br>(usually set by the code) |
| istep | 0 | Current time step<br>(usually set by the code) |
| dtgrow | 1.05 | Timestep growth factor |
| dt0 | 1.0e+10 | Initial time step size<br>(alias: dtinit) |
| dtmin | 1.0e-6 | Minimum time step size |
| dtmax | 10.0 | Maximum time step size |
| clean | .false. | Enable clean option (uses clean_df values in $mats) |
| backup | .true. | Lagrangian phase autoback-up, else abort |
| fix_crossings | .false. | Detect and fix interface crossings |
| multidiv_type | ‘uniform’ | Set to ‘voidclose’ for void closure model |
| jciter | .false. | Set to .true. for strain rate iteration to occur for Johnson-Cook strength model |
| *mesh_velocity* | *(0.0,0.0,0.0)* | *Mesh translation velocity (constant, no spatial or temporal dependence)* |

OPTIONS Variables (Continued)

| Variable | Default | Description |
|---|---|---|
| *trc_move_mesh* | *0* | *Integer number of the tracer whose velocity is to be used for the mesh velocity. This velocity, unlike the mesh_velocity input variable, could vary in time according to the velocity sampled by the tracer. If trc_move_mesh and mesh_velocity are both nonzero, then the mesh moves according to the tracer number given by trc_move_mesh.* |
| *mesh_move_time* | *1.0e+19* | *Time to begin moving mesh* |
| *mesh_stop_time* | *0.0* | *Time to stop moving mesh* |

## OUTPUTS Namelist Variables (Output specifications)

The OUTPUTS namelist input is used to specify output times and delta times. It is also used to specify what will be output. Currently, there are printed outputs (long and short edits), *frame buffer plots (on the actual CM frame buffers or on your terminal using X-windows),* restart dumps, and a preliminary graphics dump capability. *Massless tracer particle outputs* are also provided, as well as a user programmable output capability.

OUTPUTS Variables

| Variable | Default | Description |
|---|---|---|
| t | 0.0 | Output times (alias: op_t) |
| dt | 0.0 | Delta output times (alias: op_dt) |

OUTPUTS Variables: Restart Dump Variables

| Variable | Default | Description |
|---|---|---|
| dump_freq | 0 | dump frequency<br>(0: none, -1 each cycle) |
| *dump_read_fms* | *.false.* | *read mode*<br>*(serial output: .false., FMS .true.)* |
| *dump_write_fms* | *.false.* | *write mode*<br>*(serial output: .false., FMS: .true.)* |

OUTPUTS Variables: Edit Variables

| Variable | Default | Description |
|---|---|---|
| long_freq | 0 | Long edit frequency (0: none) |
| short_freq | 0 | Short edit frequency (-1: each cycle) |
| ed_imax | nx | Maximum edit index in x-direction |
| ed_imin | 0 | Minimum edit index in x-direction |
| ed_jmax | ny | Maximum edit index in y-direction |
| ed_jmin | 0 | Minimum edit index in y-direction |
| ed_kmax | ny | Maximum edit index in z-direction |
| ed_kmin | 0 | Minimum edit index in z-direction |
| ed_xmax | X-max | Maximum edit coord in x-direction |
| ed_xmin | X-min | Minimum edit coord in x-direction |
| ed_ymax | Y-max | Maximum edit coord in y-direction |
| ed_ymin | Y-min | Minimum edit coord in y-direction |
| ed_zmax | Z-max | Maximum edit coord in z-direction |
| ed_zmin | Z-min | Minimum edit coord in z-direction |

*OUTPUTS Variables: EV Dump Variables (for tool GD_EV)*

| *Variable* | *Default* | *Description* |
|---|---|---|
| *ev_var* | – | *Variable names (vofm, d, dm, em, p, pm, c, cm, q)* |
| *ev_mat* | – | *material number for mixed cell variable, else 0 for mixed cell variables, "0" is not allowed* |

*OUTPUTS Variables: YNG dump Variables (for tool GD_YNG)*

| *Variable* | *Default* | *Description* |
|---|---|---|
| *yng_mat* | *None* | *Material number(s) for which interface polygons are to be computed and written to the EV dump.* |
| *yng_first* | *None* | *Number of the first graphics dump for which interface polygons and other EV variables (specified with ev_var, ev_mat) are to be computed and written to the EV dump.* |

*OUTPUTS Variables: YNG dump Variables (for tool GD_YNG) (Continued)*

| *Variable* | *Default* | *Description* |
|---|---|---|
| *yng_first* | *None* | *Number of the last graphics dump for which interface polygons and other EV variables (specified with ev_var, ev_mat) are to be computed and written to the EV dump.* |
| *yng_inc* | *None* | *Skip increment between the first and last graphics dump for which interface polygons and other EV variables (specified with ev_var, ev_mat) are to be computed and written to the EV dump.* |

*OUTPUTS Variables: Frame Buffer Plot Variables*

| *Variable* | *Default* | *Description* |
|---|---|---|
| *fb_color* | *255.0* | *Color maximum (min=0, max=255)* |
| *fb_contours* | *10* | *Number of contour* |
| *fb_freq* | *0* | *Frame buffer frequency (0: none, -1: each cycle)* |
| *fb_level* | *1* | *Plane level (min=1, max=direction-max-dim)* |
| *fb_mat* | *0* | *Material number (0 means average cell data)* |
| *fb_max* | *–* | *Variable maximum (0 means calculate maximum from data)* |
| *fb_min* | *–* | *Variable minimum (0 means calculate minimum from data)* |
| *fb_pixwin* | *512* | *Maximum number pixels in window* |
| *fb_plane* | *‘xz’* | *Plane and orientation (xy, yz, xz, yx, zy, or zx)* |
| *fb_smooth* | *0* | *Smoothing (0: none, >0: b-quad B-spline)* |
| *fb_var* | *‘d’* | *Plot variable names (d, e, p, c, q, dm, em, pm, rho, sie, prs, vof, etc.)* |
| *fb_width* | *0.5* | *Contour width (full width is 0.5 on either side)* |

*OUTPUTS Variables: X-Window Plot Variables*

| *Variable* | *Default* | *Description* |
|---|---|---|
| *X_freq* | *0* | *X-window frequency (0: none, -1: each cycle)* |
| *X_pixwin(id)* | *512* | *Maximum number of pixels to be used in initializing the size of X-window identifier. Window resizing (with the mouse) after initialization is permitted, and this action preempts X_pixwin.* |
| *X_xpix_border(id)* | *0.10* | *Percent of the horizontal width of X-window identifier devoted to margins outside the plot space. A nonzero value is especially recommended for vector plots.* |
| *X_ypix_border(id)* | *0.10* | *Percent of the vertical width of X-window identifier devoted to margins outside the plot space. A nonzero value is especially recommended for vector plots.* |
| *X_frame_plot(id)* | *.true.* | *Frame the image by outlining the computational domain.* |
| *X_show_interfaces(id)* | *.false.* | *Show material interfaces in X-window id. Interfaces are currently approximated as the VOF=1/2 level line. If input variable "X_interfaces" is not specified for X-window id, then all interfaces are drawn.* |
| *X_interfaces(n,id)* | *1:nmat-1* | *Material numbers of the interfaces to be drawn in X-window id. All interfaces (material numbers 1 to nmat-1) are drawn as a default. Example: X_interfaces(1,3) = 4,5,6, specifies interfaces for materials 4, 5, and 6 are to be drawn in X-window 3 (if X_show_interfaces(3) = .true.)* |

*OUTPUTS Variables: X-Window Plot Variables (Continued)*

| *Variable* | *Default* | *Description* |
|---|---|---|
| *X_plot_type(id)* | *'image'* | *Plot type in X-window id. Available options are 'image', 'contour', and 'vector'.* |
| *X_show_tracers(id)* | *.false.* | *If .true., plot any tracers (as points) presently in the plane of X-window id. The tracer points are plotted in X-window id regardless of its X_plot_type.* |
| *X_show_mesh(id)* | *.false.* | *If .true., plot the computational mesh in the plane of X-window id. The mesh lines are plotted in X-window id regardless of its X_plot_type.* |
| *X_color_map(id)* | *'rainbow'* | *Color map for X-window id. 'rainbow' spans from black to red, through the rainbow of colors; 'blue' spans black to blue, and 'grayscale' makes a gray-scale color map. Grayscale is recommended if black-and-white copies of the window are to be printed on a monochrome printer.* |
| *X_image_interp(id)* | *'bilinear'* | *Interpolation algorithm to be used for images plotted in X-window id. 'bilinear' uses a bilinear (first order) interpolation of computational cell data to nearby pixels, whereas 'ngp' use a "nearest grid point" (zeroth order) interpolation. Both techniques are useful: bilinear gives a smoother view of the data, making it easier for the eye to discern patterns in the data, whereas ngp gives the raw, rough data as the code sees it.* |

*OUTPUTS Variables: X-Window Plot Variables (Continued)*

| *Variable* | *Default* | *Description* |
|---|---|---|
| *X_color(id)* | *253.0* | *Maximum entry into the color table of X-window id (min=0, max=253). Positions 254 and 255 are reserved for background and foreground, respectively.* |
| *X_plane* | *'xy'* | *Coordinate plane and orientation of the plane for X-window id. Allowable values are xy, yz, xz, yx, zy, or zx.* |
| *X_level(id)* | *1* | *Plane level to be plotted in X-window id. The actual coordinate value associated with the plane will be displayed in the top window panel. (min=1, max=direction-max-dim)* |
| *X_var(id)* | *'d'* | *Plot variable name for X-window id. Allowable variables are the same as those used for the trc_var input variables, listed in detail in the ../lib/trc_data.FCM file. The only exception is if X_plot_type(id) is set to 'vector', in which case 'velocity' is the only acceptable variable name.* |
| *X_mat(id)* | *0* | *Material number for the variable to be plotted in X-window id. A 0 gives cell-averaged quantities.* |
| *X_max(id)* | *preset* | *Plot variable values in the plane of X-window id are scaled between 0 and 1 with this as its maximum. If X_max is not input, then the current maximum of the data is used.* |

*OUTPUTS Variables: X-Window Plot Variables (Continued)*

| *Variable* | *Default* | *Description* |
|---|---|---|
| *X_min(id)* | *preset* | *Plot variable values in the plane of X-window id are scaled between 0 and 1 with this as its minimum. If X_max is not input, then the current minimum of the data is used.* |
| *X_contour_values (n,id)* | *0.0* | *Values of contour lines to be drawn for the variable in X-window id. X_contours should then be set to the number of contour values specified in X_contour_values. (Example: X_contour_values(1,2) gives the value for contour #1 in X-window #2.)* |
| *X_contours(id)* | *10* | *Number of contour intervals to be used in contouring the variable in X-window id. The intervals are taken in even steps between the current variable minimum and maximum in the X-window plane. If quantities for X_contour_values have been supplied, and then X_contours should be equal to the number of contour values specified in X_contour_values. This input only applies if X_plot_type(id) is set to 'contour'.* |
| *X_vector_length* | *3*max(dx,dy,dz)* | *Length of vectors drawn on all vector plots. The default is 3 x the maximum mesh spacing in the problem.* |
| *X_arrowhead_fraction* | *0.10* | *Fraction of the vector length covered by the arrowhead in all vector plots.* |
| *X_arrowhead_angle* | *15.0* | *Arrowhead opening angle for the vector arrowheads on all vector plots.* |

*OUTPUTS Variables: X-Window Variable Aliases*

| *Variable* | *Aliases* |
|---|---|
| *X_plot_type (id)* | *'IMAGE', 'Image', 'image', 'img', 'VECTOR', 'Vector', 'vector', 'VECTORS', 'Vectors', 'vectors', 'CONTOUR', 'Contour', 'contour', 'CONTOURS', 'Contours', 'contours'* |
| *X_var(id) (for vectors)* | *'VELOCITY', 'Velocity', 'velocity', 'vlcty', 'vel', 'VEL', 'fluid velocity', 'FLUID VELOCITY'* |
| *X_color_map (id)* | *'BLUE', 'Blue', 'blue', 'RAINBOW', 'Rainbow', 'rainbow', 'rnbow', 'GRAYSCALE', 'Grayscale', 'grayscale', 'GREYSCALE', 'Greyscale', 'greyscale', 'gray', 'grey'* |
| *X_image_interp (id)* | *'ngp', 'NGP', 'first-order', 'first', 'same-cell', 'bilinear', 'BILINEAR', 'Bilinear', 'linear', 'LINEAR', 'Linear'* |

OUTPUTS Variables: Graphics Dump Variables

| Variable | Default | Description |
|---|---|---|
| gd_freq | 0 | Graphics dump frequency (0: none, -1: each cycle) (alias: dv_freq) |
| gd_var | – | Variable names (vofm,d,dm,em,p,pm,c,cm,q,u,v,w, bfm, bt, p_shk, etc.) |
| gd_mat | – | Material number for mixed cell variable, else 0 for mixed cell variables, "0" means all materials (note: some materials may not have bfm, p_shk, etc.) |
| *gd_read_fms* | *.false.* | *Read mode (serial output: .false., FMS: .true.)* |
| *gd_write_fms* | *.false.* | *Write mode (serial output: .false., FMS: .true.)* |
| **gd_iso** | **.false.** | **Flag to specify use of the iso code to create iso surfaces.** |
| **gd_numdisks** | **0** | **Number of disks to which to write graphics output files.** |
| **gd_diskpath** | **'.'** | **Root location on the disks for writing graphics output files.** |

OUTPUTS Variables: Graphics Dump Variables (Continued)

| Variable | Default | Description |
|---|---|---|
| **gd_disk_file** | **‘ ‘** | **Root name for the graphics output files, relative to gd_diskpath. A file number is appended to the file name by the code.** |
| **gd_write_file** | **‘ ‘** | **Output file name for the .polyh file used by the iso code.** |
| **gd_mat_thresh** | **0.5, 0.5** | **Threshold values used by iso. A value may be entered for the first n materials, and the default value will be used for the remaining nmat-n materials.** |
| **gd_frame_number** | **0** | **Starting value for the frame numbers.** |

*OUTPUTS Variables: KRKL Dump Variables*

| *Variable* | *Default* | *Description* |
|---|---|---|
| *krkl_plane* | – | *“xy”, “yx”, “xz”, “zx”, “yz”, or “zy”* |
| *krkl_level* | – | *Plane index (min=1, max=direction-max-dim)* |
| *krkl_first* | – | *First dump number (optional)* |
| *krkl_last* | – | *Last dump number (optional)* |
| *krkl_inc* | – | *Increment between dumps (optional)* |

OUTPUTS Variables: User Auxiliary Output Variables

| Variable | Default | Description |
|---|---|---|
| user_freq | 0 | User auxiliary frequency<br>(0: none, -1: each cycle) |

*OUTPUTS Variables: Tracer Output Control Variables*

| *Variable* | *Default* | *Description* |
|---|---|---|
| *tracer_freq* | *0* | *Tracer frequency*<br>*(0: none, -1: each cycle)* |
| *tracer_var* | – | *Variable names*<br>*(std 9: x, y, z, u, v, w, d, e, p)*<br>*(choices: c, q, vof, sxx, sxy, sxz, syy, syz, plst, plwk, else, bf, dq, p_shk, f_shk, etc.)*<br>*(alias: trc_var)* |
| *tracer_mat* | – | *Material number (alias trc_mat)* |

## RESTARTS Namelist and RAIDS Namelist Variables

**The RESTARTS and RAIDS namelists are used to write and read restart dumps for the MIMD version of PAGOSA. A separate RAIDS namelist block is required for each disk to which restart dumps will be written or from which they will be read.**

**RESTARTS Variables: Restart Dump Variables**

| Variable | Default | Description |
|---|---|---|
| restart | .false. | Flag to indicate whether the calculation will be restarted from a dump. |
| numraid | 0 | Number of disks to which the restart dumps will be written. Each disk must have its own RAIDS namelist block. |
| readfile | ‘ ‘ | The path used for the restart dump to be read. The full path has the form raidname/readfile.processor_number.ext. raidname and ext are defined below. processor_number is the logical number of the processor which will read the file. |
| ext | ‘ ‘ | The extension used to distinguish among restart dump sets. GEN and PAGOSA write dumps with extensions 1, 2, and trm. |
| writefile | ‘ ‘ | The path used for writing the restart dump. The full path has the form raidname/writefile.processor_number.ext. raidname is defined below. ext is defined above. processor_number is the logical number of the processor which wrote the file. A restart dump set must be read by the same number of processors which wrote the set. |

**RAIDS Variables: Restart Dump Variables**

| Variable | Default | Description |
|---|---|---|
| raidname | ‘ ‘ | The root path name for a disk to which restart dumps are written or from which they are read. A separate RAIDS block is required for each disk. On the Intel Paragon, the disks often have the root form /raid/io_N or /pfs/io_N, where N is the two-digit number of the disk (disk 1 is number 01, etc.). |

## MATS Namelist Variables
## (Materials Data)

The MATS namelist is used to specify all of the material properties (EOS, HE burn, Strength, Fracture and other data associated) for each material in the problem. Each material may be entered using a separate MATS namelist block without any indexing, or all the material data may be entered within a single MATS namelist block by using explicit indexing (subscripts). The former is recommended since it is then order-independent and easily changed as materials are added or deleted.

Mats Variables

| Variable | Default | Description |
|---|---|---|
| material | – | Material index (alias: mat) |
| matbak | – | Background material number |
| matname | – | Material name |
| priority | – | Material priority (aliases: pri, mpri) |
| d0 | – | Initial material density (alias: rho0) |
| d0s | – | Normal solid density (usually d0, alias: rho0s) |
| e0 | – | Initial specific internal energy (alias: sie) |
| pmin | – | Minimum pressure cutoff (simple spall scheme) (alias: prsmin) |
| clean_df | – | Density cutoff fraction for purging (aliases: cfrho, cleandf) |
| close | .false. | For void material, set to true for void closure |

Mats Variables: EOS (Equation-of-State) Data

| Variable | Default | Description |
|---|---|---|
| eosform | | EOS form |
| | ‘void’ | ‘vacuum’, ‘null’ |
| | ‘gas’ | ‘ideal’, ‘ideal gas’, ‘ideal-gas’, ‘perfect gas’ |
| | ‘poly’ | ‘polynomial’ |
| | ‘osb’ | ‘osborne, ‘mod-osb’, ‘mod-osborne’, ‘modified-osborne’ |
| | ‘usup’ | ‘us-up’, ‘us/up’, ‘gruneisen’, ‘mie-gruneisen’, ‘mie’ |
| | ‘jwl’ | |
| | ‘bkw’ | ‘bkw-gas’, ‘bkwgas’, ‘bkwfit’ |
| | ‘bkwhe’ | ‘bkw-he’, ‘he-bkw’, ‘hebkw’ |
| eoscon | | constants for each EOS form |
| | ‘void’ | (no constants) |
| | ‘gas’ | gamma |
| | ‘poly’ | a0, a1, a2c, a2e, a3, b0, b1, b2 |
| | ‘usup’ | c0, s, gamma0, gamma1, p0, d_max |
| | ‘osb’ | a1, a2c, a2e, b0, b1, b2c, b2e, c0, c1, c2c, c2e, eps0 |
| | ‘jwl’ | w, b1, c1, b2, c2 (c1 = d*r1, c2 = d*r2) |
| | ‘bkw’ | A, B, C, D, E, K, L, M, N, O, Q, R, S, T, U, Cv, Z, d_max, d_min |
| | ‘bkw-he’ | c, s, gamma0, gamma1, P0 dsmax, K, L, M, N, O, Cv(s), alpha, A, B, C, D, E, K, L, M, N, O, Q, R, S, T, U, Cv(g), Z, dgmax, dgmin, maxit, t_del |

Mats Variables: HE Burn Data

| Variable | Default | Description |
|---|---|---|
| burnform | | EOS form |
| | ‘none’ | ‘null’, ‘inert’ |
| | ‘prog’ | ‘progburn’, ‘program’, ‘time’ |
| | ‘dyna’ | ‘dynaburn’, ‘dynamic’ |
| | ‘cj’ | ‘cjburn, ‘c-j burn’, ‘volburn’ |
| | ‘ff’ | ‘ffburn’, ‘forest’, ‘ffb’, ‘ffpburn’ |
| | ‘msff’ | ‘msffburn’, ‘msforest’, ‘msffb’ |
| | ‘jtf’ | ‘jtfburn’, ‘tangburn’, ‘tang’, ‘model-t’ (not yet implemented) |
| burncon | | constants for each HE burn form |
| | ‘none’ | (no constants) |
| | ‘prog’ | detvel |
| | ‘dyna’ | matdyna, edyna, dedyna |
| | ‘cj’ | bf_min cut, bf_max cut, CJ density |
| | ‘ff’ | bf_min cut, bf_max cut, bp_min cut, bp_max cut, n (number of FF constants following), FF constants (“n” of them), Q lim |
| | ‘msff’ | bf_min cut, bf_max cut, bp_min cut, bp_max cut, n (number of MSFF constants following), MSFF constants (“n” of them), Q lim, p_shk min, p_shk max, q_shk min, dq min, reshock p |

MATS Namelist Variables (continued): Strength Data

A large number of additional arrays are allocated to calculate strength. It is now minimized by specifying strength only for those materials which require it. For example, if there are 10 materials specified for the whole problem but only two of then have strength, then only about 20% of the possible storage will be allocated. Also, only about 20% of the possible computer time will be spent in the strength routines compared to that if all 10 materials had strength. Users are encouraged to minimize the number of materials which will realistically require strength in order to minimize run time and space.

*There is a compiler option to compile the code with or without the strength routines and storage arrays. As such, for “hydro only” problems,*

*the code is much simpler and smaller if compiled without strength.* However, with recent code changes, the code compiled with strength but run without strength (hydro only), a minimum of storage is now dynamically allocated and the strength routines will never be called. It is no longer a significant time and space penalty to run a “hydro-only” problem with a version of the code compiled with the strength option (which is the default).

Mats Variables: Strength Data

| Variable | Default | Description |
|---|---|---|
| strform | | strength form |
| | ‘none’ | ‘null’, ‘hydro’ |
| | ‘ep’ | ‘e-p’, ‘epp’ |
| | ‘s-g’ | ‘sg’ |
| | ‘j-c’ | ‘jc’ |
| y0 | – | initial yield strength (alias: yld0) |
| ymax | – | maximum yield strength (alias: yldmax) |
| g0 | – | initial shear modulus (alias sm0) |
| gmax | – | maximum shear modulus (alias smmax) |
| eoscon | | constants for each str form |
| | ‘none’ | (no constants) |
| | ‘e-p’ | (no constants) |
| | ‘s-g’ | alpha, psi0, beta, delta, emelt, gama, gamap |
| | ‘j-c’ | bcap, ccap, an, am, emlt, eroom, gamap |

Mats Variables: Fracture Data

| Variable | Default | Description |
|---|---|---|
| fracform | | fracture form (not yet implemented) |
| | ‘none’ | ‘null’ |
| fracon | | constants for each fracture form (not yet implemented) |

## DETS Namelist Variables
## (Detonator Specifications)

The DETS namelist is used to specify the programmed burn detonation points. Only a simple scheme is now implemented which does not handle shadow regions.

DETS Variables

| Variable | Default | Description |
|---|---|---|
| type | 'point' | Detonator type: *'point'*, 'line', 'plane', cylindrical', 'spherical' (aliases: *'points'*, 'cyl', 'sph') |
| t | – | Detonation time(s) (alias: time) |
| xyz | – | X, Y, and Z detonator coordinate triple(s) |
| axis | – | 'x', 'y', or 'z' axis of line, axis normal to plane, or axis of cylinder |
| radius | — | cylinder or sphere radius |
| *shadow_fraction* | – | *Detonation-velocity reduction fraction for shadow region* |

## *TRACERS Namelist Variables*
## *(Massless Tracers Particles)*

*The TRACERS namelist input is used to specify massless tracer particles.*

*NOTE: Between restarts this data must be manually updated and reentered as the current tracer positions are not now written on the restart dumps. This will be eliminated soon and become automated.*

*TRACERS Variables*

| *Variable* | *Default* | *Description* |
|---|---|---|
| *xyz(i,np)* | *N/A* | *Tracer number (i = 0) and initial (x,y,z) coordinates (i=2,3,4) for tracer particle np (aliases: trc_xyz & tracer_xyz)* |
| *frame(i,np)* | *'Lagrangian' for i = 1,2,3* | *Reference frame for tracer particle np in direction i (i = 1 for the X-direction, 2 for Y, and 3 for Z) ('Eulerian' or 'Lagrangian') (aliases: trc_frame & tracer_frame)* |

## GEN Namelist Variables
## (Generator Specifications)

The GEN namelist input block is used to specify some basic input control information for the generator, GEN.

GEN Variables

| Variable | Default | Description |
|---|---|---|
| particles | 5 | Number of particles per cell and per coordinate direction used to calculate volume fractions statistically. |
| start_mode | 1 | 1 - new generate from input file<br>2 - view previous generate from a Data-Vault restart dump file |
| interactive | .true. | .false. - The generator will ask questions about various input options and ignores them from this input file. |
| restart_dump | .true. | .false. - Write a Data-Vault dump of the code variables for use in starting a calculation with PAGOSA. |
| *burn_times* | *.false.* | *.true. - Calculate and write a Data-Vault dump of the program burn times (only if there is program burn material in the problem)* |

## BODY Namelist Variables
## (Material Body Specifications)

The BODY namelist input is used to defined the material geometry, one body at time. In practice, it often takes several bodies of the same material to make-up the complete three-dimensional solid desired. As such, all bodies of the same material are added together (union). Previously defined bodies are also “subtracted” from subsequent body specifications. This is often very handy, but it is suggested that users try not to overlap bodies and have GEN help to find any overlaps. All surfaces defined within a body are also combined (intersection). Aliases for the variable names are given in parentheses.

BODY Variables

| Variable | Description | Allowable Input |
|---|---|---|
| material_number (mat) | material index | material number from EOS input |
| surface_name (surf) | | (first 3 characters used) |
| | Plane | plane’, ‘planar’, ‘pla’ |
| | Box | box’, ‘cube’, ‘cub’, parallelepiped’, ‘par’ |
| | Sphere | sphere’, ‘sph’ |
| | Ellipsoid | ellipsoid’, ‘ellipse’, ‘ell’ |
| | Cylinder | cylinder’, ‘cyl’ |
| | Cone | cone’, ‘conical’, ‘con’ |
| | Tabular | ‘tabular’, ‘tab’ |
| | Background | ‘background’, ‘bac’ |
| u0, v0, w0 | Initial velocity | Constant numerical values. |
| d0, e0 | Initial density & energy | The body is filled with the default density & energy of the material specified unless overridden here by specifying an initial d0 & e0. |
| fill | material location | |
| | To place the material inside the body or in the positive direction | inside’, ‘fill inside’, ‘insert’, ‘+’, ‘right’, ‘above’, ‘top’, ‘front’ |
| | To place the material outside the body or in the negative direction | outside’, ‘fill outside’, delete’, ‘-’, ‘left’, ‘below’, ‘bottom’, ‘back’ |

BODY Namelist Variables (continued)

For each surface there is an appropriate and necessary set of associated location and defining input parameters from the following set. Rotations are performed first (positive angles: counter-clockwise, negative angles: clockwise) and then translations. Rotations are performed about the body’s local origin unless a non-zero rotation point is specified.

BODY Variables: Geometry Parameters

| Variable | Description | Allowable Input |
|---|---|---|
| axis | orientation axis | ‘x’, ‘y’, or ‘z’<br>‘x-axis’, ‘y-axis’ or ‘z-axis’<br>‘1’, ‘2’, or ‘3’ |
| radius | body radius | Radius triple (may change soon)<br>(triple required for ellipsoid) |
| height | height of body | single value |
| xyz_length<br>(length) | length of sides | dx, dy and dz triple |
| xyz_translation<br>_pt<br>(trans,<br>xyz_trans) | new location of body | x, y, z triple |
| xyz_rotation_pt<br>(rot, xyz_rot) | body rotation point | x, y, z triple |
| xyz_rotation_an<br>gle<br>(angle,<br>rot_angle) | body rotation angle | Angle triple (degrees) |
| tabular_type<br>(tab_type) | type of tabular body | ‘rotation’ or ‘translation’<br><br>A rotated tabular body forms an n-sided collection of cylinders and cones if the axis of rotation is included as one side of the body. If the axis is not included, then the body has a toroidal shape.<br><br>A translated body is used to form an “extruded” shape; a “funny” cylinder with sides defined by the n-sided tabular entries instead of a circle. |
| rz_tabular_pt<br>(rz) | tabular body | Table of r,z points |
| rtheta-<br>tabular_pt<br>(rtheta) | tabular body | Table of r,theta points |
| file_name<br>(file) | tabular file name | File name (w/path spec)<br>(see TABULAR_DATA below) |

BODY Namelist Variables (continued): Surface Parameters

The parameters required for each surface are given below.

BODY Variables: Surface Parameters

| Surface | Parameters | Origin | Description |
|---|---|---|---|
| ‘plane’ | axis | 0,0 | Axis normal to plane |
| ‘background’ | N/A | N/A | This “surface” is used to define the remaining space not occupied. With “matbak” now required, this body type is optional. If present, it must be the last body specified. |
| ‘box’ | xyz_length | Center | Length of each of the three coordinate sides. |
| ‘sphere’ | radius | Center | (temporarily use 3*value) |
| ‘ellipsoid’ | radius<br>xyz_translation_pt<br>xyz_rotation_pt<br>xyz_rotation_angle | Center | Three radii required |
| ‘cylinder’ | radius<br><br><br>axis<br>height<br>xyz_translation_pt<br>xyz_rotation_pt<br>xyz_rotation_angle | Center of base | (temporarily use 3*value)<br><br><br>Axis of cylinder<br>Height of cylinder |
| ‘cone’ | radius<br><br><br>axis<br>height<br>xyz_translation_pt<br>xyz_rotation_pt<br>xyz_rotation_angle | Center of base | (temporarily use 3*value)<br><br><br>Axis of cone<br>Height of cone |

BODY Variables: Surface Parameters (Continued)

| Surface | Parameters | Origin | Description |
|---|---|---|---|
| 'tabular' | tabular_type | 0,0 | 'rotation' or 'translation' |
| | axis | | Axis of rotation or axis along which points are translated. |
| | rz-tabular_pt | | Table of r,z points making up an n-sided polygon in either the clockwise or counter-clockwise direction |
| | xyz_translation_pt<br>xyz_rotation_pt<br>xyz_rotation_angle | | |

## TABULAR_DATA Namelist Variables (Tabular Contour Specifications)

The TABULAR_DATA namelist input is used to read-in a contour table in either r,z or r,theta form. In the r,theta form, theta is measured from the normal to the z-axis and can take on positive or negative values in degrees. The positive values are counter-clockwise, while the negative values are clockwise. For example, 90 degrees is along the positive z-axis, -90 degrees is along the negative z-axis while 0 degrees is at the equator (perpendicular to the z-axis).

TABULAR_DATA Variables

| Variable | Description | Allowable Input |
|---|---|---|
| rz | r,z table of values | Constant coordinate values |
| rtheta | r,theta table of values | Constant coordinate values |

## SETVEL Namelist Variables for use in GEN (User Modified Initial Velocities)

GEN now includes two user modifiable routines called "setvel" and "setvel_inp". The first (in the file setvel.Fcm) calls the second (in the file setvel_inp.F). Right now, the sample code within each is to set a uniform radial inward or outward velocity. The inclusion of a non-zero

velocity in the SETVEL namelist input will set all materials in the problem to that radial velocity (negative is inward). As presently coded, it assumes the origin of the mesh is the origin of the radial velocity. If an additional table of numbers is provided, one for each material in the problem, then the radial velocity is set only for those materials flagged with a 1 (one) and not set for those flagged with a 0 (zero).

Code users are encouraged to modify these routines to suit their own special problem set-up requirements. A non-uniform density, energy, velocity, etc. distribution may be programmed in to facilitate a special problem.

As currently coded, a missing or empty SETVEL namelist causes no change to the velocity distribution as set-up by the other parts of the generator input.

SETVEL Variables

| Variable | Description | Allowable Input |
|---|---|---|
| vel | radial velocity | + (outward) or - (inward) real number<br>A zero or missing value currently has no effect on the velocity distribution. |
| mats | material table flags | A list of 0’s & 1’s with a one-to-one correspondence with the material numbers.<br>A “0” means do not impose the radial velocity on this corresponding material while a “1” (or other non-zero) means to apply the uniform radial velocity to this corresponding material.<br><br>Omitting the “mats” table causes the radial velocity to be imposed over the entire mesh for all materials. |

*CRAYLINK Namelist Variables for Use in GEN*

*GEN now includes the means to link from the MESA-3D generator the material volume fractions and programmed burn times. This namelist may be repeated as often as necessary to map over all of the desired regions of the mesh.*

*SETVEL Variables*

| *Variable* | *Description* | *Allowable Input* |
|---|---|---|
| *vf_lnk* | *vol-frac file name* | *(alias: vf_file)* |
| *bt_lnk* | *burn_time file name* | *(alias: bt_file)* |
| *lnk_imin*<br>*lnk_imax* | *x-direction indices* | *(defaults = 0,mx)*<br>*(aliases: imin & imax)* |
| *lnk_jmin,*<br>*lnk_jmax* | *y-direction indices* | *(defaults = 0,my)*<br>*(aliases: jmin & jmax)* |
| *lnk_kmin,*<br>*lnk_kmax* | *z-direction indices* | *(defaults = 0,mz)*<br>*(aliases: kmin & kmax)* |
| *i_offset,*<br>*j_offset,*<br>*k_offset* | *mapping offsets* | *(defaults = 0,0,0)*<br>*(aliases: ioffset, joffset & koffset)* |
| *u0, v0, w0* | *mesh velocities* | *(defaults = 0,0,0)* |

## *WINDOW*

*In addition to GEN and the PAGOSA main program, there is a preliminary windowing utility program which merges previous GEN or PAGOSA made restart dumps into a new restart dump. In this manner, different portions may be pieced together (or deleted) as a problem is run. Old portions may be deleted and new portions may be added. The mesh size may be expanded or shrunk as needed, but the mesh cells may not be "rezoned" or resized. As of now, portions may be "extracted" from previously made restart dump files and placed upon a whole new mesh. Also, two-dimensional portions may be "extruded" in the orthogonal directions to make a three-dimensional problem out of a two-dimensional problem.*

*EXTRACT Namelist Variables for use in WINDOW*

*EXTRACT Variables*

| *Variable* | *Description* | *Allowable Input* |
|---|---|---|
| *file* | *name of old restart dump* | *required, must not be blank* |
| *nx, ny, nz* | *size of old mesh* | *required* |
| *nmat* | *number of materials in old problem* | *Required* |

*EXTRACT Variables (Continued)*

| *Variable* | *Description* | *Allowable Input* |
|---|---|---|
| *imin, imax<br>jmin, jmax<br>kmin, kmax* | *mesh range of old problem from which to "extract"* | *(defaults = whole old mesh)<br>(less than or equal to the new "window" range)* |
| *istart,<br>jstart,<br>kstart* | *mapping offsets into the new "window" mesh* | *(defaults = 0, 0, 0)* |
| *mats_table* | *material mapping table* | *For each old material, a "zero" indicates that material is to be skipped (or dropped).*<br><br>*A non-zero "new" material number indicates that material is to be included, but as the "old n-th" material as given by this position (or index) within the table.* |
| *burnform* | *old HE burn form, if any* | *'prog', 'msff', etc. in the "old n-th" material position of this array as required to read the old restart dump.<br>(defaults: none)* |
| *fracform* | *old fracture form, if any* | *(not yet implemented)* |
| *strform* | *old strength form, if any* | *'ep', 'kospall', etc. in the "old n-th" material position of this array as required to read the old restart dump.<br>(defaults: none)* |
| *burn_times* | *old burn_time file name* | *Include only if needed for the next phase of the new problem.<br>(aliases: bt_file and prog_burn_times)* |

*EXTRUDE Namelist Variables for use in WINDOW*

*EXTRUDE Variables*

| *Variable* | *Description* | *Allowable Input* |
|---|---|---|
| *plane* | *name of 2D plane to extrude* | *required, must not be blank ‘xy’, ‘yz’, ‘xz’, ‘zx’, ‘yz’, ‘zy’* |
| *start_level* | *Existing level of the 2D plane* | *1 through mx, my, or mz (defaults to 1)* |
| *final_level* | *Level to which to extrude* | *may be above or below the start_level (defaults to mx, my, or mz)* |
| *imin, imax jmin, jmax kmin, kmax* | *Mesh range of new problem from which to “extrude”* | *(defaults = whole old mesh) (less than or equal to the new “window” range)* |

*WINDOW also has a set of empty “user_mods” routines which may be programmed to perform special functions required by the user. These may include changing a material throughout the mesh, etc.*

## Post-Processing Utility Programs

### GRID

This program is used to help calculate a mesh grid. It copied from and slightly improved upon the Mesa-2D program which does the same thing. Please refer to the Mesa-2D manuals at this time for more information, or else just try it!

### *GD (Graphics Dumps) Utilities*

*These utilities are for post-processing the graphics dumps. For input, they use an input file which may be identical to the original file used to run PAGOSA (however, rename it with a different prefix to avoid overwriting the same output files). For the most part, they are self explanatory, for there is not yet a detailed description of their workings.*

*GD Utilities*

| | |
|---|---|
| *GD_EV* | *Makes an EV (Eulerian Viewer) file for viewing data on the IRIS graphics system.* |
| *GD_KRKL* | *Makes a near faithful reproduction of a Mesa-2D dump for subsequent processing through KRKL for plots of two-dimensional slices.* |
| *GD_MERGE* | *Merges or extracts portions of graphics dump files.* |
| *GD_MOVIE* | *Shows on the frame buffer (or X-terminal) a “movie” like sequence of frames produced by GD_SLICE.* |

*GD Utilities (Continued)*

| | |
|---|---|
| *GD_SLICE* | *Extract two-dimensional slices from the graphics dumps and render them into frame buffer image using the frame buffer specifications in the OUTPUTS namelist block.* |
| **iso** | **Extract three-dimensional iso-surfaces of volume fraction from the graphics dumps and create a file of polyhedra which can be visualized using AVS modules.** |

*READ_TRACERS*

*This utility program reads the "prefix.trc" output by PAGOSA and extracts the data requested into a file of time and value pairs for plotting by Xvgr or other plotting programs. It asks for the prefix, the time range desired, the variable values wanted and some scaling information. Again, try it and see if it will extract the data you need.*

EXTRACT_SHORT

This utility extracts data from the short edits on the "prefix.out" file made by PAGOSA into a file(s) of time and value pairs. Again, this is for post-processing plots. The data which may be selected as of now by materials is volume, mass, density, total energy, internal energy and kinetic energy.

EXTRACT_LONG

This program extracts spatial data from the long edit on the "prefix.out" file. In addition to the prefix, it asked for the variables desired and the indices, or spatial range to be extracted.

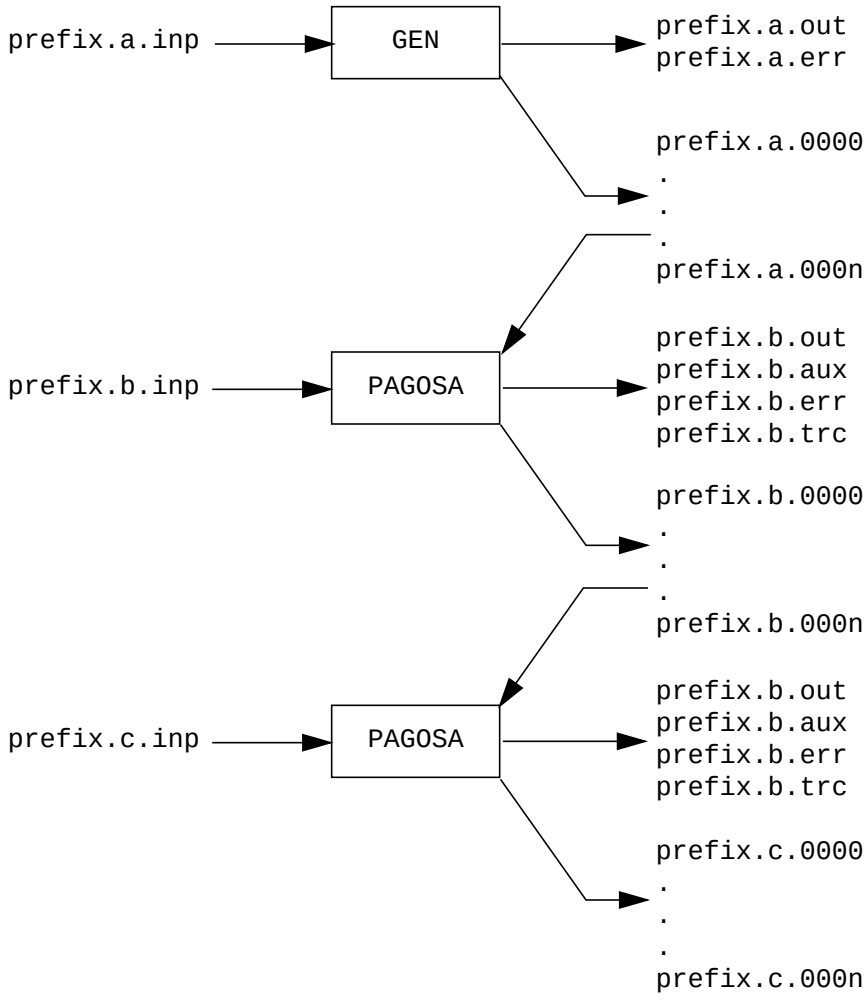
EXTRACT_RDR

This is a utility program to extract the integral of Rho*dr over time from the "prefix.aux" file written by the special "user_write" subroutine. The user must compile and run PAGOSA with this special version of the subroutine for this data to be calculated and written out on the auxiliary prefix.aux file.

Sample Sequence in Running GEN and PAGOSA

The following schematic is a sample sequence showing several phases in running GEN and then PAGOSA with a restart. Here there are separate input files for each run (“prefix.a.inp” for GEN, “prefix.b.inp” for the first PAGOSA, and “prefix.c.inp” for the PAGOSA restart), but they are derived from each other and are nearly identical. For example, the “b” version may have a special initial timestep and stopping time specified, while the “c” version may reset the initial time step large (which then has no effect), set a new stopping time, and update the tracer particle coordinates.

The names “prefix.a”,”prefix.b” and “prefix.c” are then used to name all of the output files with unique and non-overlapping names. The “prefix” part is usually a brief problem identifier (unclassified) which is also recommended to be a sub-file directory name. For example, “test1”, “test2”, etc. where “test” is the overall parent directory name. A naming scheme like this helps to keep things organized, logically together and non-overlapping



PAGOSA Directory Structure

The following directory structure currently applies to PAGOSA and all of its programs and utilities. Note the executable programs are stored in the /bin sub-directory while the binary libraries, sources and debugging intermediate source files are found in separate directories, one for each program.

The library (*.a file), sources (*.Fcm, *.F, *.f and *.fcm files) are found under each program directory in the format as shown for GEN and PAGOSA. The *.F files are the F77 sources *while the *.Fcm files are the CMF sources,* both used as input to the CPP preprocessor run under CC. The *.f and the *.fcm are the corresponding preprocessor output files which are actually compiled by *F77 and CMF respectively,* **or by if77**. They are also available for use with the debuggers.

The Makefiles are the compiler scripts for each program and are under constant change as we learn how to do things in a better way. This is par for UNIX. They (and the code storage structure) will also change when we start using the *CVS* or **SCCS** code maintenance system.

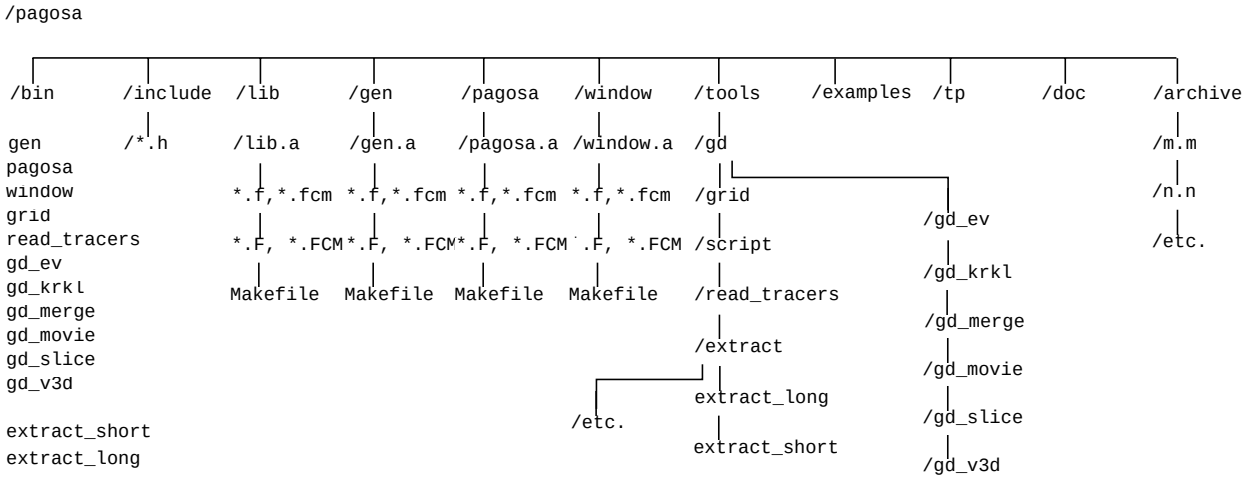
The /include directory is for the common blocks and variable definitions which are included within the programs by the preprocessor. Any changes to these may require recompiling all of the programs and libraries.

The /lib directory is for those subroutines which are used in common between GEN and PAGOSA or the other utility programs. As such, it is separated out and compiled once. Any changes to it require reloading (sometimes changing and re-compiling) all of the programs which use it. As of now, it is not real easy to recompile lib.a by itself without providing some compile switches to its Makefile. Compiling GEN or PAGOSA now automatically compiles and builds lib.a as needed.

The /examples directory is for some example input files to help users get started while the /tp directory is for test problems used for code validation and quality assurance (Q/A). The /archive directory is for each new code version by version number. The last (highest version number) is the version in the program directories which are compiled and stored in /bin.

The tools directory is the beginning of a set of utilities or tools for pre-post processing. The grid program assists in setting-up a variable mesh, or grid. *The read_tracers program is used to extract massless tracer particle data from the *.trc tracer output file, and the 'gd' series is used to process the graphics dumps. gd_ev is used to extract data from a graphics dump and make a set of EV input dump files for displaying on the Iris SGI. gd_krkl is used to make a MESA dump file for plotting by the KRKL plotting program. gd_merge is used to exact or merge together graphics dump files. gd_movie is used to display frame buffer plots (2D slices) made by the gd_slice program.*

## Abbreviated Directory Structure for PAGOSA



## B Test Problem Input Sets

In this appendix we present input sets for the two finned projectile problems, fp1 and fp2, for the explosive welding problem, ew, and for the oil-well perforation problem, owp.

### B1 Input Set for the Finned Projectile Problem with the Hydrodynamic Constitutive Model, fp1

```
Finned Projectile Obliquely Impacting a Steel Plate Hydrodynamically

 $mesh
       ncellx = 384,192,  coordx =  -1.0, 0.0, 0.5, ratiox = 1.0, 1.0,
       ncelly =  96, 96,  coordy =  -0.5, 0.0, 0.5, ratioy = 1.0, 1.0,
       ncellz =  96,      coordz =   0.0, 0.5, ratioz = 1.0,
 $end

 $mats
       mat     = 1,
       pri     = 2,
       matbak  = 1,
       matname = 'void',
       eosform = 'void',
 $end

 $mats
       matname = 'Tungsten Projectile',
       material = 2,
       priority = 1,
       d0 = 17.3, d0s = 17.3, e0 = 0.0, clean_df = 0.2, detvel = 0.0,
       eosform = 'us/up',
       pmin = -0.02,
       eoscon = 0.4, 1.295, 0.0, 1.43,
       strform = 'none',
       y0 = 0.0193, ymax = 0.0193, g0 = 1.28, gmax = 1.28,
       fracform = 'none',
 $end

 $mats
       mat = 3,
       pri = 3,
       matname = 'SS plate',
       d0 = 7.89, e0 = 0.0, clean_df = 0.2, pmin= -.06,
       eosform = 'us/up',
       eoscon = 0.4569, 1.49, 0.0, 2.17,
       strform = 'none'
       y0 = 0.010, g0 = 0.77,
       fracform = 'none',
 $end
```

```
$options

      dt0=0.010, dtmin=1.0d-5, dtmax=0.20,

      id_q=1, cq1=0.2, cq2=2.0,
      cutacc=1.0d-9, cutd=1.0d-5, cutvof=1.0d-5,

      dtgrow=1.2,
      safec=0.75, safed=0.25, safeu=0.25,

      ibc = 1,1, 1,1, 0,1,

$end

$outputs

      t=0.0, 5.00, dt=0.01,

      gd_freq=0,
      gd_var ='vofm',
      gd_mat =   0  ,

      short_freq=0,
      dump_freq=0,
$end

$gen
      particles = 5,
      startmode = 1,
      interactive     = .false.,
      restartdump     = .false.,
      burntimes       = .false.
$end

$body
      mat    =   1,
      surf   =   'plane',
      fill   =   'right',
      axis   =   'x',
      trans  =    0.150,  0.000,  0.000,
      rot    =    0.000,  0.000,  0.000,
      angle  =    0.000,  0.000,-30.000,
$end
```

```
$body
      mat   =    3,
      surf  =    'plane', 'plane',
      fill  =    'left' , 'right',
      axis  =    'x', 'x',
      trans =    0.150,  0.000,  0.000,
                 0.000,  0.000,  0.000,
      rot   =    0.000,  0.000,  0.000,
                 0.000,  0.000,  0.000,
      angle =    0.000,  0.000,-30.000,
                 0.000,  0.000,-30.000,
$end

$body
      mat    =   2,
      surf   =   'cone',
      fill   =   'inside',
      radius =   0.075,
      height =   0.200,
      axis   =   'x',
      trans  =  -0.200,  0.000,  0.000,
      rot    =   0.000,  0.000,  0.000,
      angle  =   0.000,  0.000,  0.000,
      u0     =   0.100,   v0 = 0.00,   w0 = 0.00,
$end

$body
      mat    =   2,
      surf   =   'cyl',
      fill   =   'inside',
      axis   =   'x',
      radius =   0.075,
      height =  -0.700,
      trans  =  -0.200,  0.000,  0.000,
      rot    =   0.000,  0.000,  0.000,
      angle  =   0.000,  0.000,  0.000,
      u0     =   0.100,   v0 = 0.00,   w0 = 0.00,
$end

$body
      mat    =   2,
      surf   =   'box',
      fill   =   'inside',
      axis   =   'x',
      length =   0.200,  0.300,  0.050,
      trans  =  -0.800,  0.000,  0.000,
      rot    =   0.000,  0.000,  0.000,
      angle  =   0.000,  0.000,  0.000,
      u0     =   0.100,   v0 = 0.00,   w0 = 0.00,
$end
```

```
$body
      mat    =   2,
      surf   =   'box',
      fill   =   'inside',
      axis   =   'x',
      length =   0.200,  0.050,  0.300,
      trans  =  -0.800,  0.000,  0.000,
      rot    =   0.000,  0.000,  0.000,
      angle  =   0.000,  0.000,  0.000,
      u0     =   0.100,   v0 = 0.00,   w0 = 0.00,
$end
```

## B2 Input Set for the Finned Projectile Problem with the Elastic, Perfectly Plastic Constitutive Model, fp2

```
Finned Projectile Obliquely Impacting a Steel Plate

 $mesh
       ncellx = 320,160,  coordx =  -1.0, 0.0, 0.5, ratiox = 1.0, 1.0,
       ncelly =  80, 80,  coordy =  -0.5, 0.0, 0.5, ratioy = 1.0, 1.0,
       ncellz =  80,      coordz =   0.0, 0.5, ratioz = 1.0,
 $end

 $mats
       mat = 1,
       pri = 3,
       matbak  = 1,
       matname = 'void',
       eosform = 'void',
 $end

 $mats
       matname = 'Tungsten Projectile',
       material = 2,
       priority = 1,
       d0 = 17.3, d0s = 17.3, e0 = 0.0, clean_df = 0.2, detvel = 0.0,
       eosform = 'us/up',
       pmin = -0.02,
       eoscon = 0.4, 1.295, 0.0, 1.43,
       strform = 'ep',
       y0 = 0.0193, ymax = 0.0193, g0 = 1.28, gmax = 1.28,
       fracform = 'none',
 $end

 $mats
       mat = 3,
       pri = 2,
       matname = 'SS plate',
       d0 = 7.89, e0 = 0.0, clean_df = 0.2, pmin= -.06,
       eosform = 'us/up',
       eoscon = 0.4569, 1.49, 0.0, 2.17,
       strform = 'ep'
       y0 = 0.010, g0 = 0.77,
       fracform = 'none',
 $end
```

```
$options

      dt0=0.010, dtmin=1.0d-6, dtmax=0.10,

      id_q=1, cq1=0.2, cq2=2.0,
      cutacc=1.0d-9, cutd=1.0d-5, cutvof=1.0d-5,

      dtgrow=1.2,
      safec=0.75, safed=0.25, safeu=0.25,

      ibc = 1,1, 1,1, 0,1,

      clean = .true.,

$end

$outputs

      t=0.0, 7.00, dt=0.01,

      gd_freq=0,
      gd_var ='vofm',
      gd_mat =   0  ,

      short_freq=0,
      dump_freq=0,
$end

$gen
      particles = 5,
      startmode = 1,
      interactive     = .false.,
      restartdump     = .false.,
      burntimes       = .false.
$end

$body
      mat    =   1,
      surf   =   'plane',
      fill   =   'right',
      axis   =   'x',
      trans  =    0.150,  0.000,  0.000,
      rot    =    0.000,  0.000,  0.000,
      angle  =    0.000,  0.000,-30.000,
$end
```

```
$body
      mat   =    3,
      surf  =    'plane', 'plane',
      fill  =    'left' , 'right',
      axis  =    'x', 'x',
      trans =    0.150,  0.000,  0.000,
                 0.000,  0.000,  0.000,
      rot   =    0.000,  0.000,  0.000,
                 0.000,  0.000,  0.000,
      angle =    0.000,  0.000,-30.000,
                 0.000,  0.000,-30.000,
$end

$body
      mat    =   2,
      surf   =   'cone',
      fill   =   'inside',
      radius =   0.075,
      height =   0.200,
      axis   =   'x',
      trans  =  -0.200,  0.000,  0.000,
      rot    =   0.000,  0.000,  0.000,
      angle  =   0.000,  0.000,  0.000,
      u0     =   0.100,   v0 = 0.00,   w0 = 0.00,
$end

$body
      mat    =   2,
      surf   =   'cyl',
      fill   =   'inside',
      axis   =   'x',
      radius =   0.075,
      height =  -0.700,
      trans  =  -0.200,  0.000,  0.000,
      rot    =   0.000,  0.000,  0.000,
      angle  =   0.000,  0.000,  0.000,
      u0     =   0.100,   v0 = 0.00,   w0 = 0.00,
$end

$body
      mat    =   2,
      surf   =   'box',
      fill   =   'inside',
      axis   =   'x',
      length =   0.200,  0.300,  0.050,
      trans  =  -0.800,  0.000,  0.000,
      rot    =   0.000,  0.000,  0.000,
      angle  =   0.000,  0.000,  0.000,
      u0     =   0.100,   v0 = 0.00,   w0 = 0.00,
$end
```

```
$body
      mat    =   2,
      surf   =   'box',
      fill   =   'inside',
      axis   =   'x',
      length =   0.200,  0.050,  0.300,
      trans  =  -0.800,  0.000,  0.000,
      rot    =   0.000,  0.000,  0.000,
      angle  =   0.000,  0.000,  0.000,
      u0     =   0.100,   v0 = 0.00,   w0 = 0.00,
$end
```

## B3 Input Set for the Explosive Welding Problem, ew

```
Explosive Welding of a Copper Tube to a Steel Plate

 $mesh
       ncellx = 240,240,  coordx =  -3.0, 0.0, 3.0, ratiox = 1.0, 1.0,
       ncelly = 120,      coordy =   0.0, 3.0,      ratioy = 1.0,
       ncellz = 120,      coordz =   0.0, 3.0,      ratioz = 1.0,
 $end

 $mats
       mat = 1,
       pri = 5,
       matbak = 1
       matname = 'void',
       eosform = 'void',
 $end

 $mats
       mat = 2,
       pri = 1,
       matname = 'copper',
       d0 = 8.93, e0 = 0.0, cleandf = 0.2, pmin = -.06,
       eosform = 'us/up',
       eoscon  = 0.394, 1.489, 0.0, 2.002,
       strform = 'ep',
       y0 = 0.003, g0 = 0.477,
       fracform = 'none'
 $end

 $mats
       mat = 3,
       pri = 2,
       matname = '304 SS plate',
       d0 = 7.89, e0 = 0.0, clean_df = 0.2, pmin= -.06,
       eosform  = 'us/up',
       eoscon   = 0.4569, 1.49, 0.0, 2.17,
       strform  = 'ep'
       y0 = 0.010, g0 = 0.77,
       fracform = 'none',
 $end
```

```
$mats
      mat = 4,
      pri = 3,
      matname = 'PBX-9501',
      d0 = ****,  e0 = ****, pmin = ****, detvel = ****,
      clean_df = 0.0,
      eosform = 'jwl',
      eoscon = ****, ****, ****, ****, ****,
      y0 = 0.0, ymax = 0.0, g0 = 0.0, gmax = 0.0,
      burnform = 'prog',
      strform  = 'none',
      fracform = 'none',
$end

$mats
      mat = 5,
      pri = 4,
      matname = 'foam',
      d0 = 0.32, e0 = 0.0, clean_df = 0.2, pmin = -0.001,
      eosform = 'foam-us/up',
      eoscon = 0.07, 1.13, 0.0, 1.70,
      strform = 'none',
      fracform = 'none',
$end

$dets
      time = 0.0,
      xyz =  1.0,  0.0,  0.0,
$end

$options

      dt0=0.005, dtmin=1.0d-5, dtmax=0.25,

      id_q=1, cq1=0.2, cq2=2.0,
      cutacc=1.0d-9, cutd=1.0d-5, cutvof=1.0d-5,

      dtgrow=1.2,
      safec=0.75, safed=0.25, safeu=0.25,

      ibc = 1,1, 0,1, 0,1,

      clean = .true.,

$end
```

```
$outputs

      t=0.0, 10.00, dt=1.00,

      gd_freq=0,
      gd_var ='vofm',
      gd_mat =   0  ,

      short_freq=0,
      dump_freq=0,
$end

$gen
      particles = 5,
      startmode = 1,
      interactive     = .false.,
      restartdump     = .false.,
      burntimes       = .false.
$end
$body
      mat    =   4,
      surf   =   'cyl',
      fill   =   'inside',
      axis   =   'x',
      radius =   0.300,
      height =   1.000,
      trans  =   0.000,  0.000,  0.000,
      rot    =   0.000,  0.000,  0.000,
      angle  =   0.000,  0.000,  0.000,
$end

$body
      mat    =   5,
      surf   =   'cyl',
      fill   =   'inside',
      axis   =   'x',
      radius =   0.600,
      height =   1.500,
      trans  =   0.000,  0.000,  0.000,
      rot    =   0.000,  0.000,  0.000,
      angle  =   0.000,  0.000,  0.000,
$end

$body
      mat    =   1,
      surf   =   'cyl',
      fill   =   'inside',
      axis   =   'x',
      radius =   0.600,
      height =   3.000,
      trans  =   0.000,  0.000,  0.000,
      rot    =   0.000,  0.000,  0.000,
      angle  =   0.000,  0.000,  0.000,
$end
```

```
$body
      mat    =   2,
      surf   =   'cyl'    , 'cyl'
      fill   =   'outside', 'inside'
      axis   =   'x'      , 'x',
      radius =   0.600,  0.600,  0.600,
                 1.000,  1.000,  1.000,
      height =   5.000,  5.000,
      trans  =   0.000,  0.000,  0.000,
                 0.000,  0.000,  0.000,
      rot    =   0.000,  0.000,  0.000,
                 0.000,  0.000,  0.000,
      angle  =   0.000,  0.000,  0.000,
                 0.000,  0.000,  0.000,
$end

$body
      mat    =   1,
      surf   =   'cone',
      fill   =   'inside',
      radius =   2.000,
      height =   2.000,
      axis   =   'x',
      trans  =   0.000,  0.000,  0.000,
      rot    =   0.000,  0.000,  0.000,
      angle  =   0.000,  0.000,  0.000,
$end

$body
      mat   =    3,
      surf  =    'plane', 'plane',
      fill  =    'left' , 'right',
      axis  =    'x', 'x',
      trans =    2.000,  0.000,  0.000,
                 0.000,  0.000,  0.000,
      rot   =    0.000,  0.000,  0.000,
                 0.000,  0.000,  0.000,
      angle =    0.000,  0.000,  0.000,
                 0.000,  0.000,  0.000,
$end
```

## B4 Input Set for the Oil-Well Perforation Problem, owp

```
Oil Well Perforator with Water

 $mesh
       ncellx = 96,144, coordx = -8.0, 0.0, 12.0, ratiox = 1.0, 1.0,
       ncelly = 96,     coordy =  0.0, 8.0,       ratioy = 1.0, 1.0,
       ncellz = 96, 96, coordz = -4.0, 0.0,  4.0, ratioz = 1.0, 1.0,
 $end

 $options
      cutacc = 1.0e-9, cutd = 1.0e-5, cutvof = 1.0e-4, cutrecon = 0.0,
      safec = 0.75, safed = 0.25, safeu = 0.25,
      id_geom = 3,
      id_q = 1, cq1 = 0.2, cq2 = 2.0,
      ibc = 1, 1, 0, 1, 1, 1,
      dtgrow = 1.1, dt0 = 0.02, dtmin = 0.0001, dtmax = 0.75,
      clean = .true., backup = .true., fix_crossings = .false.,
 $end

 $outputs
      t = 0.0, 30.0, dt = 0.1,
      long_freq = 0,
      short_freq = 0,
      dump_freq = 0,
      gd_freq = 0,
      gd_var = 'vofm',
      gd_mat = 0,
      user_freq = 0,
      tracer_freq = 0,
 $end

 $dets
      type = 'los_point',
      xyz = -2.2860, 0.0,  1.5,
            -0.3683, 0.0, -1.5,
      time = 0.0, 0.0,
 $end

 $mats
       matname = 'tubing air',
       material = 1,
       priority = 1,
       matbak   = 1,
       eosform = 'gas',
       d0 = 1.293e-03, e0 = 1.9335e-03, pmin = 0.0,
       eoscon = 1.4,
       strform = 'none',
       fracform = 'none',
 $end
```

```
$mats
      matname = 'liner',
      material = 2,
      priority = 2,
      eosform = 'usup',
      d0 = 8.93, d0s = 8.93, e0 = 0.0, clean_df = 0.1, pmin = -0.001,
      eoscon = 0.394, 1.489, 0.0, 2.17,
      strform = 'e-p',
      y0 = 0.001, ymax = 0.001, g0 = 0.477, gmax = 0.477,
      fracform = 'none',
$end

$mats
      matname = 'cyclotol',
      material = 3,
      priority = 3,
      eosform = 'jwl',
      burnform = 'program',
      d0 = ****,  e0 = ****, pmin = ****, detvel = ****,
      eoscon = ****, ****, ****, ****, ****,
      strform = 'none',
      y0 = 0.0, ymax = 0.0, g0 = 0.0, gmax = 0.0,
      strcon = 0.0,
$end

$mats
      matname = 'stainless steel',
      material = 4,
      priority = 4,
      eosform = 'usup',
      d0 = 7.896, e0 = 0.0, clean_df = 0.1, pmin = -0.003,
      eoscon = 0.4569, 1.49, 0.0, 2.17,
      strform = 'e-p',
      y0 = 0.010, ymax = 0.010, g0 = 0.81, gmax = 0.81,
      fracform = 'none',
$end

$mats
      matname = 'water',
      material = 5,
      priority = 5,
      eosform = 'usup',
      d0 = 1.0, e0 = 0.0, clean_df = 0.1, pmin = 0.0,
      eoscon = 0.18, 1.6, 0.0, 1.0,
      strform = 'none',
      fracform = 'none',
$end
```

```
$mats
      matname = 'quartz strata',
      material = 7,
      priority = 7,
      eosform = 'usup',
      d0 = 2.204, e0 = 0.0, pmin = 0.0,
      eoscon = 0.794, 1.695, 0.0, 0.9,
      strform = 'none',
      fracform = 'none',
$end

$gen
     particles    = 5,
     startmode    = 1,
     interactive  = .false.,
     restart_dump = .false.,
     burntimes    = .true.,
$end

$body
      mat    = 1,
      surf   = 'tabular',
      tab_type = 'rotation',
      axis   = 'x',
      fill   = 'inside',
      rz = 0.0000, -1.8070,
           0.1483, -1.8070,
           0.8725, -0.0000,
           0.0000, -0.0000,
      trans  = 0.0, 0.0, 1.5,
      rot    = 3*0.0,
      angle  = 3*0.0,
$end

$body
      mat    = 1,
      surf   = 'sphere',
      fill   = 'inside',
      radius = 3*0.16,
      trans  = -1.7471, 0.0, 1.5,
      rot    = 3*0.0,
      angle  = 3*0.0,
$end
```

```
$body
      mat    = 2,
      surf   = 'tabular',
      tab_type = 'rotation',
      axis   = 'x',
      fill   = 'inside',
      rz = 0.0000, -1.9472,
           0.2206, -1.9472,
           0.8725, -0.3734,
           0.8725, -0.2032,
           0.0000, -0.2032,
      trans  = 0.0, 0.0, 1.5,
      rot    = 3*0.0,
      angle  = 3*0.0,
$end

$body
      mat    = 2,
      surf   = 'sphere',
      fill   = 'inside',
      radius = 3*0.2388,
      trans  = -1.8558, 0.0, 1.5,
      rot    = 3*0.0,
      angle  = 3*0.0,
$end

$body
      mat    = 3,
      surf   = 'tabular',
      tab_type = 'rotation',
      axis   = 'x',
      fill   = 'inside',
      rz = 0.0000, -2.2860,
           0.1981, -2.2860,
           0.1981, -2.1336,
           0.3175, -2.1336,
           0.5156, -2.0345,
           0.8725, -0.8712,
           0.8725, -0.3734,
           0.0000, -0.3734,
      trans  = 0.0, 0.0, 1.5,
      rot    = 3*0.0,
      angle  = 3*0.0,
$end
```

```
$body
      mat    = 4,
      surf   = 'tabular',
      tab_type = 'rotation',
      axis   = 'x',
      fill   = 'inside',
      rz = 0.2794, -2.6543,
           0.1981, -2.2860,
           0.1981, -2.1336,
           0.8725, -0.2032,
           0.8725,  0.0,
           0.9266,  0.0,
           1.2065, -0.2774,
           1.2065, -1.1049,
           1.1532, -1.5164,
           0.8128, -2.1145,
           0.6985, -2.1717,
           0.6985, -2.6116,
      trans  = 0.0, 0.0, 1.5,
      rot    = 3*0.0,
      angle  = 3*0.0,
$end

$body
      mat    = 1,
      surf   = 'tabular',
      tab_type = 'rotation',
      axis   = 'x',
      fill   = 'inside',
      rz = 0.0000, 1.8070,
           0.1483, 1.8070,
           0.8725, 0.0000,
           0.0000, 0.0000,
      trans  = -2.6543, 0.0, -1.5,
      rot    = 3*0.0,
      angle  = 0.0, 0.0, 0.0,
$end

$body
      mat    = 1,
      surf   = 'sphere',
      fill   = 'inside',
      radius = 3*0.16,
      trans  = -0.9072, 0.0, -1.5,
      rot    =3*0.0,
      angle  = 0.0, 0.0, 0.0,
$end
```

```
$body
      mat    = 2,
      surf   = 'tabular',
      tab_type = 'rotation',
      axis   = 'x',
      fill   = 'inside',
      rz = 0.0000, 1.9472,
           0.2206, 1.9472,
           0.8725, 0.3734,
           0.8725, 0.2032,
           0.0000, 0.2032,
      trans  = -2.6543, 0.0, -1.5,
      rot    = 3*0.0,
      angle  = 0.0, 0.0, 0.0,
$end

$body
      mat    = 2,
      surf   = 'sphere',
      fill   = 'inside',
      radius = 3*0.2388,
      trans  = -0.7985, 0.0, -1.5,
      rot    = 3*0.0,
      angle  = 0.0, 0.0, 0.0,
$end

$body
      mat    = 4,
      surf   = 'tabular',
      tab_type = 'rotation',
      axis   = 'x',
      fill   = 'inside',
      rz = 0.2794, 2.6543,
           0.1981, 2.2860,
           0.1981, 2.1336,
           0.8725, 0.2032,
           0.8725,  0.0,
           0.9266,  0.0,
           1.2065, 0.2774,
           1.2065, 1.1049,
           1.1532, 1.5164,
           0.8128, 2.1145,
           0.6985, 2.1717,
           0.6985, 2.6116,
      trans  = -2.6543, 0.0, -1.5,
      rot    = 3*0.0,
      angle  = 0.0, 0.0, 0.0,
$end
```

```
$body
      mat    = 4,
      surf   = 'cyl', 'cyl', 'sphere' 'sphere'
      fill   = 'outside', 'inside', 'outside', 'outside',
      axis   = 'z', 'z',
      height = 10.0, 10.0,
      radius = 3*1.54686, 3*1.99136, 3*14.25238, 3*14.25238,
      trans  = -1.32715, 0.0, -5.0,
                          -1.32715, 0.0, -5.0,
                          14.75784, 0.0,  1.5,
                         -17.41214, 0.0, -1.5,
      rot    = 3*0.0, 3*0.0, 3*0.0, 3*0.0,
      angle  = 3*0.0, 3*0.0, 3*0.0, 3*0.0,
$end

$body
      mat    = 5,
      surf   = 'cyl', 'cyl'
      fill   = 'outside', 'inside',
      axis   = 'z', 'z',
      height = 10.0, 10.0,
      radius = 3*1.83261, 3*6.21284,
      trans  = -1.32715, 0.0, -5.0,
                2.89433, 0.0, -5.0,
      rot    = 3*0.0, 3*0.0,
      angle  = 0.0, 0.0, 0.0,
               0.0, 0.0, 0.0,
$end

$body
      mat    = 6,
      surf   = 'cyl', 'cyl'
      fill   = 'outside', 'inside',
      axis   = 'z', 'z',
      height = 10.0, 10.0,
      radius = 3*6.21284, 3*6.985,
      trans  = 2.89433, 0.0, -5.0,
                           2.89433, 0.0, -5.0,
      rot    = 3*0.0, 3*0.0,
      angle  = 0.0, 0.0, 0.0,
               0.0, 0.0, 0.0,
$end

$body
      mat    = 7,
      surf   = 'cyl'
      fill   = 'outside',
      axis   = 'z',
      height = 10.0,
      radius = 3*6.985,
      trans  = 2.89433, 0.0, -5.0,
      rot    = 3*0.0,
      angle  = 0.0, 0.0, 0.0,
$end
```

```
$body
    mat    = 1,
    surf   = 'background',
$end
```

# Distribution

1. External Distribution

14 Director
U.S. Army Research Laboratory
Aberdeen Proving Ground, MD
21005-5066

Attn.: AMSRL-WM-PD (K. A. Bannister)
Attn.: AMSRL-WM-TA (S. Bilyk)
Attn.: AMSRL-CI-A (H. J. Breaux)
Attn.: AMSRL-WM-PD (B. Burns)
Attn.: AMSRL-WM-PD (D. A. Hopkins)
Attn.: AMSRL-WT-TD (J. Huffington)
Attn.: AMSRL-WT-TC (K. D. Kimsey)
Attn.: AMSRL-CI-CA (Nisheeth Patel)
Attn.: AMSRL-WM-MF (A. M. Rajendran)
Attn.: AMSRL-WT-TD (M. Raftenberg)
Attn.: AMSRL-WT-NC (S. J. Schraml)
Attn.: AMSRL-WT-TD (S. Segletes)
Attn.: AMSRL-WT-TD (T. Wright)
Attn.: AMSRL-SC (W. H. Mermagen, Sr.)

3 Commander
U.S. Army Armament Research,
Development and Engineering Center
Picatinny Arsenal, NJ 07806-5001

Attn.: SMCAR-AEE-WW (E. L. Baker)
Attn: SMCAR-AET (W. Ebehara)
Attn.: SMCAR-AET-M (F. Witt)

3 Commander
U.S. Air Force Wright Laboratory
Eglin Air Force Base, FL 32549-6810

Attn.: MNMW (J. Foster, Jr.)
Attn.: MNMW (J. A. Collins)
Attn.: MNMW (M. Nixon)

1 Commander
U.S. Army Missile Command
Redstone Arsenal, AL 35898-5240

Attn: AMSMI-RD-ST-WF (D. Lovelace)

1 Director
U.S. Army Research Office
P. O. Box 12211
Research Triangle Park, NC 27709

Attn.: SLCRO (Dr. K. Iyer)

2 Director
U.S. Army Materials Technology
Laboratory
Arsenal Street
Watertown, MD 02172-0001

Attn.: SCLMT-MR (C. White)
Attn.: SCLMT-MR (Tony Chou)

1 Commander
Naval Weapons Center
China Lake, CA 93555-6001

Attn.: Code 3261 (T. J. Gill)

1 Commander
U.S. Army Belvoir Research,
Development, and Engineering Center
Fort Belvoir, VA 22060

Attn.: STRBE-NAA (S. Bishop)

2 Director
U.S. Naval Surface Warfare Center
10901 New Hampshire Avenue
Silver Spring, MD 20903-5000

Attn.: Code R12 (L. Hudson)
Attn.: Code R12 (P. Walter)

1 R.K. Garrett, Jr.
NSWC, Indian Head Division
Code 410G, Bldg ROB7
101 Strauss Avenue
Indian Head, MD 20640-5035

2 Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington, VA 22203-1714

Attn.: Lt. Col. Joseph Beno
Attn.: Maj. Robert W. Kocher

3 Special Defense Weapons Agency
HQ DNA/SPSD
6801 Telegraph Road
Alexandria, VA 22310-3398

Attn.: M. E. Giltrud
Attn.: J. Connell
Attn.: LTC Carlos Rubio

1 Director
U.S. Air Force Weapons Laboratory
Kirtland Air Force Base, NM 87185
Attn.: NTI (C. Mulligan)

1 Dr. Albert Holt
OUSD(A)/TWP/OM
Pentagon, Room 3B1060
Washington, DC 20301-3100

1 Warren Chernok
Defense Programs
US Department of Energy
1000 Independence Avenue, SW
Washington, DC 20585

1 D. B. Nelson, Executive Director
ER-7, GTN
Office of Energy Research
Scientific Computing Staff
US Department of Energy
Washington, DC 20585

1 Dr. William Happer, Director
Energy Research
US Department of Energy
1000 Independence Avenue, SW
Washington, DC 20585

1 James Decker
Energy Research
US Department of Energy
1000 Independence Avenue, SW
Washington, DC 20585

3 Office of Energy Research
Scientific Computing Staff
US Department of Energy
Washington, DC 20545
Attn.: ER-7, GTN (T. A. Kitchens)
Attn.: ER-7, GTN (D. Hitchcock)
Attn.: ER-7, GTN (F. Howes)

2 Institute for Advanced Technology
The University of Texas at Austin
4030-2 W. Braker Lane
Austin, TX 78759-5329
Attn.: Tom M. Kiehne
Attn.: Douglas D. Cline

21 Los Alamos National Laboratory
Mail Station 5000
P. O. Box 1663
Los Alamos, NM 87545
Attn.: F. Addessio, MS B216
Attn.: J. Cerutti, MS F663
Attn.: S. T. Bennion, MS F663
Attn.: G. E. Cort, MS G787
Attn.: P. Follansbee, MS F663
Attn.: K. Holian, MS B295
Attn.: J. W. Hopson, MS B216
Attn.: L. Hull, MS J960
Attn.: J. N. Johnson, MS F663
Attn.: D. B. Kothe, MS B216
Attn.: O. Lubek, MS B265
Attn.: D. A. Mandell, MS F663
Attn.: J. Moore, MS B265
Attn.: D. Rabern, MS F663
Attn.: J. G. Sanderson, MS J488
Attn.: M. Simmons, MS B265
Attn.: R. D. Smith, MS B216
Attn.: B. Spangenberg, MS F663
Attn.: L. Schwalbe, MS F663
Attn.: D. Tonks, MS B221
Attn.: H. Wassermann, MS B265

4 Lawrence Livermore National
Laboratory
P. O. Box 808
Livermore, CA 94550
Attn.: D. Baum, L-35
Attn.: R. Christiansen, L-35
Attn.: R. Couch, L-35
Attn.: D. Lassila, L-342

## 2. Internal Distribution

| | | | |
|---|---|---|---|
| 1 | MS | 0151 | G. Yonas, 9000 |
| 1 | | 0321 | W. J. Camp, 9200 |
| 1 | | 0318 | G. S. Davidson, 9215 |
| 1 | | 1111 | S. S. Dosanjh, 9221 |
| 30 | | 1111 | D. R. Gardner, 9221 |
| 12 | | 1111 | 9221 File |
| 1 | | 1110 | D. E. Womble, 9222 |
| 1 | | 1110 | D. S. Greenberg, 9223 |
| 1 | | 1109 | A. L. Hale, 9224 |
| 1 | | 1111 | G. S. Heffelfinger, 9225 |
| 1 | | 0441 | R. E. Leland, 9226 |
| 30 | | 1109 | C. T. Vaughan, 9226 |
| 1 | | 0819 | J. S. Peery, 9231 |
| 1 | | 0819 | E. Hertel, 9231 |
| 1 | | 0820 | P. Yarrington, 9232 |
| 1 | | 0439 | D. R. Martinez, 9234 |
| 1 | | 0833 | J. H. Biffle, 9103 |
| 1 | | 0825 | C. C. Wong, 1554 |
| 1 | | 0842 | C. M. Hart, 2500 |
| 1 | | 0861 | J. T. Hitchcock, 2521 |
| | | | |
| 1 | | 9018 | Central Technical Files, 8940-2 |
| 5 | | 0899 | Technical Library, 4414 |
| 2 | | 0619 | Review and Approval Desk, 12690<br>for DOE/OSTI |