Scalability of Parallel Spatial Direct Numerical Simulations on Intel Hypercube and IBM SP1 and SP2

Ronald D. Joslin (Corresponding Author)
Flow Modeling and Control Branch
NASA Langley Research Center
Mail Stop 170
Hampton, VA 23681
(804) 864-2234
Fax: (804) 864-7897

Ulf R. Hanebutte
Reactor Analysis Division
Argonne National Laboratory
Argonne, IL 60439

and

Mohammad Zubair
IBM Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

HEADLINE: Scalability of Parallel Spatial DNS

Scalability of Parallel Spatial Direct Numerical Simulations on Intel Hypercube and IBM SP1 and SP2

Ronald D. Joslin,[1] **Ulf R. Hanebutte,** [2] **and Mohammad Zubair**[3]

Abstract

The implementation and performance of a parallel spatial direct numerical simulation (PSDNS) approach on the Intel iPSC/860 hypercube and IBM SP1 and SP2 parallel computers is documented. Spatially evolving disturbances associated with the laminar-to-turbulent transition in boundary-layer flows are computed with the PSDNS code. The feasibility of using the PSDNS to perform transition studies on these computers is examined. The results indicate that PSDNS approach can effectively be parallelized on a distributed-memory parallel machine by remapping the distributed data structure during the course of the calculation. Scalability information is provided to estimate computational costs to match the actual costs relative to changes in the number of grid points. By increasing the number of processors, slower than linear speedups are achieved with optimized (machine-dependent library) routines. This slower than linear speedup results because the computational cost is dominated by FFT routine, which yields less than ideal speedups. By using appropriate compile options and optimized library routines on the SP1, the serial code achieves 52–56 Mflops on a single node of the SP1 (45 percent of theoretical peak performance). The actual performance of the PSDNS code on the SP1 is evaluated with a "real world" simulation that consists of 1.7 million grid points. One time step of this simulation is calculated on eight nodes of the SP1 in the same time as required by a Cray Y/MP supercomputer. For the same simulation, 32-nodes of the SP1 and SP2 are required to reach the performance of a Cray C-90. A 32 node SP1 (SP2) configuration is 2.9 (4.6) times faster than a Cray Y/MP for this simulation, while the hypercube is roughly 2 times slower than the Y/MP for this application.

KEY WORDS: Spatial direct numerical simulations; incompressible viscous flows; spectral methods; finite differences; parallel computing.

[1] Flow Modeling and Control Branch, NASA Langley Research Center, Hampton, VA 23681

[2] Reactor Analysis Division, Argonne National Laboratory, Argonne, IL 60439

[3] IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598

1. INTRODUCTION

The state of the three-dimensional boundary-layer flow on the wings and fuselage of an aircraft determines the viscous drag portion of the total drag of the aircraft. This viscous drag, which is flow-state dependent, can amount to 40 or 50 percent of the total drag. (See Bushnell et al., 1977.) Any decrease in the viscous drag can lead to reduced fuel expenditures. This fuel savings can translate directly into reduced operating costs for the industry. The flow field on a wing can be in a laminar, turbulent, or transitional (an intermediate state that indicates transition from a laminar to a turbulent flow) state. Because a laminar flow state yields less viscous drag than a turbulent flow state, laminar flow on the wings is preferable and results in a net fuel savings. Today, turbulent flows engulf most of the wing area of commercial aircraft. Clearly, any aircraft manufacturer that successfully designs an aircraft with "laminar flow wings" (i.e., wings covered primarily by laminar flow) will have an enormous advantage.

As yet, the transition from laminar to turbulent flow is not completely understood. The first reasonably comprehensive method for predicting transition was derived from stability theory, which is the e^N method by Smith and Gamberoni (1956) and Van Ingen (1956). Although the e^N method is widely used to predict transition in a broad class of flows, it does have some limitations: a quasi-parallel boundary layer is assumed; no amplitude information about the ingested disturbance in the boundary layer is taken into account; and the method is semiempirical, which requires some foreknowledge of the flow in transition. The true physical problem involves the ingestion of disturbances that interact in a nonlinear manner in the later stages of transition and are imbedded in a growing boundary layer. Consequently, a method that accounts for nonparallel flow and nonlinear interactions is necessary to predict transition.

Recently, Herbert and Bertolotti (1987) have devised a nonlinear, nonparallel computational method that is based on the parabolized stability equations (PSE). With some success,

Malik and Li (1992) have extended the PSE approach to compute crossflow disturbances in swept Hiemenz flow. Validation of this new approach for a broad class of flows will continue throughout this decade. Before the development of this theory, the only approach to solve the nonparallel, nonlinear boundary-layer transition problem was by direct numerical simulation (DNS). To date, most studies with DNS have been limited to the temporal formulation, in which a spatially periodic computational domain travels with the disturbance and the temporal evolution of the disturbance is computed. This method has enabled the extension of the simulations into the later stages of transition (Zang and Hussaini, 1987, 1990; and Laurien and Kleiser, 1989) and has provided a database of qualitative information that unfortunately lacks the physically realistic spatial representation. Spatial DNS computes spatially evolving disturbances and can provide needed quantitative information about transition. Progress in spatial DNS has been made by, among others, Danabasoglu et al. (1990, 1991) for channel flows; Fasel (1976), Spalart (1989), Fasel et al. (1990), Rai and Moin (1991a, 1991b), Bestek et al. (1992), and Joslin et al. (1992, 1993) for boundary-layer flows; and Joslin and Streett (1994) for swept-wing flows. For a more complete list of accomplishments in transition prediction with DNS, refer to the reviews by Kleiser and Zang (1991) and Reed (1994).

Enormous speed and memory requirements are necessary for spatial DNS because of the large domains and intensive computations that are involved. Machines that can process large amounts of data at faster speeds are in ever increasing demand. Two possibilities exist for achieving high computational speeds: technological advancements and parallel computations. Technological advancements alone will not provide the desired computational speed because certain intrinsic physical limitations are being reached. An important limitation is the cycle speed, which is governed by the propagation speed of the signal in the given media. For example, the Cray 1 (delivered in 1976) had a cycle time of 12.5 nsec, and the Cray 2 (delivered in 1987) had a cycle time of 4.1 nsec. Although 11 years elapsed, an improvement

of only a factor of 3 in processor speed has been achieved. Parallel computation is a more attractive alternate approach because the cost and size of computer components decrease by an order of magnitude compared with Cray-type supercomputers, with only an incremental decrease in component speed. The real advantage to parallel computing is the increase in computational speed that occurs as the number of processors are increased.

A large body of literature covers the treatment of numerical algorithms for vector and parallel computers. (See Ortega and Voigt, 1988.) In most cases, the numerical treatments have focused on simplified problems. In other cases, algorithms for computationally intensive kernels in isolation from the entire application have been studied. These studies are all necessary; however, an efficient scheme for a kernel does not necessarily result in the efficient implementation of the whole scientific application. Typically, an entire application consists of a number of kernels with different data-distribution requirements, which necessitates data movement between two kernels. This movement can considerably degrade the performance of the entire application on a parallel machine.

The exploitation of parallelism for real-world computations becomes an even greater challenge in the absence of tools that transform sequential codes into parallel codes. A number of issues must be considered in the implementation of an entire application on a parallel computer. Some of these issues include:

Data Mapping. The data distribution among the various processors of a parallel machine is a key factor in the efficiency of a parallel implementation. For many scientific applications, the optimal data distribution is not obvious and requires experimentation.

Communication Requirement. The choice of the algorithm and the data distributions determines the communication requirements of an application. Two factors are treated separately: the communication volume and the frequency of communication. Frequent interprocessor communication is not desirable on parallel machines because of the high communication-setup overheads. On such machines, large messages and infrequent commu-

nication are preferable.

Problem Granularity. For a given number of processors, a problem granularity exists below which parallelization is not effective. The problem granularity depends on the hardware and software characteristics of the parallel machine.

Single Node Performance. The efficient implementation of a code on a single node is important because any improvement in the computational performance on a single processor will have a multiplicative effect on the overall parallel performance.

In summary, the whole scientific application needs to be implemented carefully to obtain desirable performance on parallel machines.

Scientific applications can be clearly categorized according to their suitability for parallel implementation; however, many scientific and engineering applications fall into grey areas where the suitability of parallelization for the application is not clear. In most cases, the application must be implemented and tested to determine its suitability for parallel computations. These applications include a variety of diverse numerical approaches. Some examples of these applications are: Fischer et al. (198l), Henderson and Karniadakis (1991), and Fisher and Patera (1994) who discussed the use of spectral element methods to characterize the unsteady Navier-Stokes equations; Otto (1993), who studied chemical reactions in a computational fluid dynamics code; Jackson et al. (1991), who studied incompressible turbulence with a temporal DNS code; and Eidson and Erlebacher (1995), who used a fully balanced tridiagonal solver (all three directions) in a temporal DNS code to study compressible, isotropic turbulence. Fisher and Patera (1994) summarize current work in the area of parallel simulation of viscous incompressible flows. Although they discuss a variety of numerical techniques which have been used on parallel architectures, including the first parallel computation of a viscous incompressible flow by Moin and Kim (1982) on a 64-processor ILLIAC IV, three-dimensional (3D) spatial direct numerical simulation (DNS) algorithms are not discussed in the review.

A serial spatial DNS code described by Joslin et al. (1992, 1993), which was used to study laminar-to-turbulent transition flow problems on Cray vector machines, is used to determine its suitability for parallelization on distributed-memory parallel machines. The present paper documents the PSDNS implementation and performance studies conducted by Joslin and Zubair (1993) on the Intel hypercube, the follow-up work by Hanebutte et al. (1994) on the IBM SP1, and current IBM SP2 results.

2. GOVERNING EQUATIONS

To compute the disturbance development, the incompressible Navier-Stokes equations are solved. The streamwise direction is x, the wall-normal direction is y, and the spanwise direction is z. A sketch of the computational domain is shown in Figure 1. Instantaneous velocities $\underline{\tilde{u}} = (\tilde{u}, \tilde{v}, \tilde{w})$ and pressure $\tilde{p}$ are decomposed into the base components $\underline{U} = (U, V, W)$ and P and the disturbance components $\underline{u} = (u, v, w)$ and p so that

$$\underline{\tilde{u}}(\underline{x}, t) = \underline{U}(\underline{x}) + \underline{u}(\underline{x}, t) \quad \text{and} \quad \tilde{p}(\underline{x}, t) = P(\underline{x}) + p(\underline{x}, t) \tag{2.1}$$

where $\underline{x} = (x, y, z)$ and t is time.

The base flow is generally the steady-state solution of the Navier-Stokes equations. For simplicity, this study will use the Blasius similarity profiles for the base flow of a flat-plate transition problem.

To determine the disturbance component of the instantaneous velocities and the pressure, substitute equation (1) into the Navier-Stokes equations and subtract the base-flow equations. The resulting unsteady, nonlinear disturbance equations are

$$\frac{\partial \underline{u}}{\partial t} + (\underline{u} \cdot \nabla)\underline{u} + (\underline{U} \cdot \nabla)\underline{u} + (\underline{u} \cdot \nabla)\underline{U} = -\nabla p + \frac{1}{R_{\delta_o^*}} \nabla^2 \underline{u} \qquad \nabla \cdot \underline{u} = 0. \tag{2.2}$$

Boundary conditions at the wall and in the far field are

$$\underline{u} = 0 \quad \text{at} \quad y = 0 \qquad \text{and} \qquad \underline{u} \to 0 \quad \text{as} \quad y \to \infty \tag{2.3}$$

The equations have been nondimensionalized with respect to the free-stream velocity U_∞, the kinematic viscosity ν, and some length scale at the inflow (e.g., displacement thickness δ_o^*). A Reynolds number can then be defined as $R_{\delta_o^*} = U_\infty \delta_o^* / \nu$.

3. NUMERICAL TECHNIQUES

The serial algorithms contained in the spatial direct numerical simulation code are described in detail by Joslin et al. (1992, 1993). Because the focus of this paper is on the parallelization of the code, the serial routines are only listed.

In the streamwise direction (x-direction), fourth-order central finite differences are used for the pressure equation. At boundary and near-boundary nodes, fourth-order differences are used. For the first and second derivatives in the momentum equations, sixth-order compact differences by Lele (1992) are used. At the boundary and near-boundary nodes, explicit fifth-order finite differences are used. The compact differences lead to tridiagonal systems (TRIDIAG), and the central finite differences lead to a pentadiagonal system (PENTADIAG); both of these systems can be solved efficiently by an LU-decomposition with the appropriate backward and forward substitutions.

In the wall-normal direction (y-direction), a Chebyshev series is used to approximate the disturbance at the Gauss-Lobatto collocation points. Because this series and its associated spectral operators are defined on [-1, 1] and the physical problem of interest has either a semi-infinite $[0, \infty)$ or a truncated domain $[0, y_{\max}]$, an algebraic-mapping transformation is employed. The Chebyshev series operators lead to matrix-matrix multiplications (MATMAT) to determine derivatives.

In the spanwise direction (z-direction), periodicity is assumed, which allows for Fourier series representations. With the Fourier series, spectral accuracy is obtained in the spanwise direction, and fast Fourier transforms (FFT) may be used for fast computation of derivatives. For more details on the spectral methods used here, refer to Canuto et al. (1988).

For time marching, a time-splitting procedure was used with implicit Crank-Nicolson

differencing for normal diffusion terms; an explicit three-stage Runge-Kutta method by Williamson (1980) was used for the remaining terms. This time-stepping procedure was used successfully by Streett and Hussaini (1991) for Taylor-Couette flow simulations and is desribed by Joslin et al. (1992) for the flat-plate boundary-layer problem. In the time-splitting procedure, the pressure is omitted from the momentum equations for the fractional Runge-Kutta stage and time is advanced from to the intermediate disturbance velocities, a Poisson pressure equation is solved for the full-stage pressure, and the full-stage velocities are obtained with the pressure inclusion.

Disturbances are introduced into the boundary layer by forcing at the inflow boundary; however, the swept-wing problem of Joslin and Streett (1994) used surface suction and blowing to introduce disturbances. At the outflow, the buffer-domain technique of Streett and Macaraeg (1989) is used.

4. PARALLEL IMPLEMENTATION OF THE SPATIAL DNS

In this section, the various data-distribution options available for implementation in the three-dimensional DNS code on a parallel machine are discussed, and the data distribution used in this application is outlined.

4.1 Data Mapping

The DNS code consists of a number of computationally intensive kernels. Dependent upon the data mapping, some of these kernels are executed locally on a single processor, and the rest are executed globally across the processors. The kernels that are executed locally do not require communication between processors; kernels that are executed globally require communication. The major computationally intensive kernels are the matrix-matrix multiplication, the FFT, the tridiagonal solver, and the pentadiagonal solver. The operation counts that correspond to the kernels are illustrated in Table 1; these operation counts are for a one-time iteration of the DNS code. Of these major kernels, the matrix-matrix multiplication is the most computationally intensive kernel. Hereafter, n_p denotes the number of

processors on a parallel machine, and n_x, n_y, and n_z denote the number of data items (i.e., grid points) in the streamwise, wall-normal, and spanwise directions.

For the three-dimensional problem, three major data mappings exist:

x-mapping. The three-dimensional data are partitioned into n_x two-dimensional planes of $n_y n_z$ data items each. The first n_x/n_p planes are mapped to processor P_0, the next n_x/n_p planes are mapped to processor P_1, and so on.

y-mapping. The three-dimensional data are partitioned into n_y two-dimensional planes of $n_x n_z$ data items each. The first n_y/n_p planes are mapped to processor P_0, the next n_y/n_p planes are mapped to processor P_1, and so on.

z-mapping. The three-dimensional data are partitioned into n_z two-dimensional planes of $n_x n_y$ data items each. The first n_z/n_p planes are mapped to processor P_0, the next n_z/n_p planes are mapped to processor P_1, and so on. An example of this mapping is shown in Figure 2 for $n_p = 4$.

As stated earlier, the data mapping determines whether a particular kernel is to be executed across all processors or executed locally on a single processor. Table 2 shows the executions of the major kernels for the three data mappings. For the x-mapping, a great deal of communication is clearly required, which is undesirable. Both the y- and z-mapping are more desirable than the x-mapping because most of the kernels are executed locally. Because Table 1 indicates that the matrix-matrix multiply has a higher operation count than the FFT, the z-mapping should be more efficient than the y-mapping. Hence, our implementation of the PSDNS code is based on the z-mapping.

Because the data are distributed among the n_p processors in block form with a z-mapping, a global re-mapping (shown in Figure 3) allows the utilization of optimized serial FFT library routines simultaneously on each node.

4.2 FFT Implementation and Routine Specifics

The FFT kernel computes $n_x n_y$ sequences of discrete Fourier transforms of size n_z. The z-mapping distributes sequences across all processors of the machine. One way to compute the discrete Fourier transform of these sequences is as follows: first, the transpose is taken of the three-dimensional data (the resulting distribution is an x-mapping), the one-dimensional FFT algorithm is then executed on sequences of length n_z on each processor, the results are multiplied by a coefficient array, the inverse Fourier transform of the data is then computed, and, finally, the results are transposed back to the original data distribution. In this scheme, all global data movement occurs in the transpose step. To keep communication overheads to a minimum, the transpose operation must be implemented efficiently. The transpose operation is a complete exchange operation; every node has an equivalent amount of data to exchange with every other node. The *xor* algorithm is used to implement this exchange procedure because it is the optimal procedure on the Intel iPSC/860. The *xor* algorithm (Bokhari, 1991), which is illustrated in Table 3, schedules various exchanges at particular nodes to avoid link contention. In this scheme at the ith step, a node j sends data to node iXj.

5. PSDNS VALIDATION

The evolution of a Tollmien-Schlichting wave in the three-dimensional flow is used to validate the PSDNS approach. A disturbance with an arbitrarily small initial amplitude $u_o = 0.0001\%$ of the freestream velocity is introduced into the boundary layer by a forcing at the inflow for the PSDNS. The inflow disturbance profiles are obtained with linear stability theory (LST). The parallel-flow assumption is used for comparison with LST. Calculations are made with an inflow Reynolds number $R_{\delta_o^*} = 900$ and frequency $\omega = 0.0774$. The PSDNS was computed on a grid of 200 uniformly spaced streamwise nodes (60 nodes per disturbance wavelength), 61 wall-normal collocation points, and 8 spanwise nodes. The outflow boundary is $121\delta_o^*$ from the inflow boundary and the far-field (or free-stream) boundary is $75\delta_o^*$ from

the wall. For the time-marching scheme, the disturbance period is divided into 320 time steps.

Figure 4 shows the streamwise evolution of the computed streamwise (u) and wall-normal (v) velocity components of PSDNS compared with LST. Very good agreement in amplitude and phase are found at every spanwise location in the physical domain. (Note, that the buffer-domain region is nonphysical and that the DNS and LST results are not expected to agree.) This simple test case demonstrates that the PSDNS code is validated with the theoretical solutions.

6. PSDNS ON INTEL IPSC/860 HYPERCUBE

Although direct simulations of transition involve large computational grids and many thousands of time steps per simulation, the performance of the PSDNS code can be sufficiently examined for a single time step on smaller grids. The cost and feasibility of a full-scale simulation can be estimated by using scaling information.

The range of parameters is limited to the capability of the machine. The Intel iPSC/860 hypercube at NASA Langley Research Center has 32 processors, each with 8 megabytes of memory. Because the single precision is limited to 32-bit words and because simulations of transition require the computations of small-scale phenomena, all performance test cases are double-precision (64-bit words) computations.

6.1 Performance of PSDNS on Hypercube

The first sequence of performance simulations is computed on a grid of 64 streamwise points and 41 wall-normal points. Figure 5 shows both the computational cost (total cost minus communication) and the communication cost for each processor in CPU sec for a variation in the spanwise grid and the number of processors. For a given computational grid, a decrease in both the computational and communication cost is achieved by increasing the number of processors. For this parallel implementation, the communication cost does not exceed 6 percent of the total cost for all grids and variations in the number of processors that

were considered. Joslin and Zubair (1993) showed the relative cost of the major numerical techniques and the speedup of each technique with the number of processors. As expected from Table 1, the computational cost breakdown indicates that the majority of the time was spent on matrix-matrix multiply operations, and a negligible change occurs in the cost contributions from each numerical technique with an increase in the number of processors. The speedup of each numerical technique as the number of processors increases indicates a nearly ideal linear speedup for the matrix-matrix multiply and the tridiagonal solver. Because matrix-matrix multiplies account for nearly 80 percent of the total computational cost and because the speedup for the matrix-matrix multiplies are nearly ideal, the total speedup approaches a nearly linear rate. For example, the theoretically ideal speedup rate is 4 with an increase from 8 to 32 processors and the speedup of 3.4 was realized in the total computational cost. Concentrating on matrix-matrix multiply, a machine-dependent BLAS routine (Dongarra et al., 1990) was introduced and the performance measurements were repeated. Figure 6 shows the computational and communication costs with a spanwise grid refinement and an increase in the number of processors. In comparison with Figure 5, the trend of reduced cost as the number of processors is increased remains the same; however, the relative communication cost has become significant. Although the quantitative cost of communication is the same as before the new matrix-matrix multiply routine was introduced, the communication now equals 20 to 30 percent of the total cost because the new matrix-matrix multiply routine has reduced the total computational cost by a factor of 4 to 5 in comparison with the original code implementation. The results in Figures 5 and 6 clearly demonstrate the significant advantages of using machine dependent library routines (when available) over user supplied routines.

The relative balance in work load between respective processors is an important element in documenting the PSDNS performance. Figure 7 shows the computational and communication cost for each stage of the three-stage Runge-Kutta time step for each processor of an

8-processor simulation on a spanwise grid of $n_z = 8$. In this figure, the AIMS performance software shows the work load for each processor in a separate display area. The lines that connect the processor areas indicate global communication; the shaded areas indicate the computational work; the blank spaces between shaded areas (horizontal direction only) indicate idle times. The results show that all processors are load balanced, with the exception of the first node (P_0). Because of the influence-matrix pressure solver that is used on the first node only, additional work is always required on this node. The idle time amounts to about 15 to 20 percent of the total cost for a single time-step advancement. For the present combination of numerical techniques and parallel implementation, this is the best load balance that can be expected.

Figure 8 shows the relative cost of the numerical techniques and the speedup of each technique with the number of processors. Unlike the initial implementation, where matrix-matrix multiply accounted for 80 percent of the total computational cost, the use of the optimized (or BLAS) matrix-matrix multiply routine leads to more balanced relative workloads among the major numerical techniques. (Hereafter, all computations and results are with the BLAS matrix-matrix multiply routine.) The communication and FFT have comparable relative cost and now dominate over the other major numerical techniques. The matrix-matrix multiply and the tridiagonal solver both have nearly ideal speedups; however, the total computational speedup is only about half of the ideal rate. This decrease in efficiency occurs because the FFT routine is the dominant numerical technique and the FFT rate of speedup is not ideal.

The present multiprocessor implementation of the PSDNS can be further evaluated by varying the number of $x - y$ planes on each processor. For example, the use of eight spanwise grid points on an eight-processor simulation indicates that each processor performs computations on a single $x-y$ plane; eight spanwise grid points on a four processor simulation indicates that each processor performs computations on two $x-y$ planes. Figure 9 shows the

cost for a single $x-y$ plane on each processor compared with the cost of four $x-y$ planes on each processor. Because FFT and global communication costs increase, cost increases are incurred in both cases with increased number of processors. Although the FFT technique and the global communication are both affected by changes in the number of processors for a fixed number of $x-y$ planes on each processor, the costs are constant for all other techniques. For four $x-y$ planes on each processor, note the initial drop in the computational cost from two to four processors.

In the remainder of this section, the streamwise, wall-normal, and spanwise grids will be refined to determine scaling costs. In Figure 10, the computational and communication cost with number of processors and spanwise grid refinement for the case where the streamwise grid is refined from $n_x = 64$ to $n_x = 128$. Consistent with the coarse-grid results shown in Figure 6, decreases in both the computational and communication costs are achieved by increasing the number of processors, and the communication cost accounts for 10 to 30 percent of the total computational cost.

Figure 11 shows the major numerical technique costs and the increase in computational cost associated with a spanwise grid refinement on a fixed number of processors: eight. The numerical techniques and communication cost are balanced; however, communication becomes very significant as the spanwise grid is refined. Because the FFT routine is the dominant numerical technique and because the FFT slowdown rate is small, the total rate of slowdown closely follows the FFT rate. This result is advantageous; it indicates that the cost increases are less than the refinement factor of spanwise grid refinement. If the spanwise grid is refined by a factor of four, then only a factor-of-three increase in the total computational cost results.

Finally, Figure 12 shows that streamwise grid refinements lead to nearly linear theoretical increases in cost; Figure 13 shows that wall-normal grid refinements also lead to nearly ideal linear increases in computational cost. Although the results in Figure 13 show that

the matrix-matrix multiplies scale like n_y^2, the FFT and communication dominate the cost leading to the total cost scaling like the FFT rate. These scaling results can be used to estimate the cost of simulations which require fine grids (i.e., large number of grid points).

6.2 Large-Scale Simulation on Hypercube

The present performance data for the PSDNS suggest that insufficient core memory is a limitation of the hypercube. The largest grid that fits on a single node has 128 streamwise, 41 wall-normal, and 4 spanwise points (21,000 total grid points). An attempt to perform computations with 8 spanwise planes of this same $x-y$ grid failed because of insufficient memory. Because the code requires about 160 bytes per grid point, the 21,000-point grid used about 3.4 megabytes of memory; the failed grid required 6.8 megabytes (plus operating system). The largest grid that could potentially be used for the present PSDNS code has less than 42,000 grid points on each processor. To determine if simulations of transition can be undertaken, the grid requirements must be specified to estimate the computational cost requirements and then compared with the core-memory limitations. The grid resolution is highly dependent on the physical problem and the numerical techniques. To estimate the feasibility of using PSDNS on the hypercube, a sample transition problem computed on a single processor of a Cray-2 supercomputer is used for comparison.

In a recent study, Joslin and Streett (1994) computed the linear and nonlinear evolution of a crossflow vortex packet on a swept wing with spatial DNS on the Cray-2. The cost of this computation amounted to approximately 125 CPU hrs with a single processor. The grid contained 901 chordwise, 61 wall-normal, and 32 spanwise points (1.76 million total points) which required 36.6 megabytes of core memory. The unsteady computation required 9500 time steps in the time-marching scheme to reach the nonlinear inflectional velocity profile stage, which occurs just prior to the laminar-to-turbulent transition.

Each $x-y$ plane of the Joslin and Streett (1994) study contained 55,000 grid points (8.8 megabytes of memory), which is beyond the capability of the present hypercube (8 megabytes

per processor). In this case, the feasibility of using PSDNS has been easily determined by examining the memory limitation alone. However, if 16 or 32 megabytes of memory per processor were available transition studies could potentially be conducted on the hypercube. Then the feasibility of using this parallel computer would rest on the computational cost of such a simulation.

The temporal cost can be determined based on the previous performance results of a single time step. From data in Figures 12 and 13, the rates at which computational cost increases with streamwise and wall-normal grid refinements can be roughly estimated at 1.95 and 2.15, respectively. The performance results indicate that a single time step on a grid of 64 streamwise, 41 wall-normal, and 32 spanwise points distributed on 32 processors will cost 3.7 sec for each processor. With the scaling rates, the computation by Joslin and Streett (1994) performed on the hypercube is estimated to cost 77 sec for each processor per time step. This results in a total cost of 206 hr for each processor to achieve the nonlinear inflectional velocity profile state described by Joslin and Streett (1994). With the dedicated use of a 32-processor hypercube with 16 megabytes of memory per processor, a simulation could be completed in approximately 9 days, which is nearly twice the cost of using a Cray-2 supercomputer. This comparison is a rough estimate of the total computational cost required for the simulations because only small grids can be used. On a grid with 64 streamwise, 41 wall-normal, and 32 spanwise points, the computational cost of 3.0 sec resulted on a single processor of a Cray-Y/MP; the performance was 189 megaflops. For the same grid, the computational cost on the hypercube resulted in 3.7 sec for each processor and roughly 153 megaflops.

In summary, simulations can apparently be performed on the hypercube, provided that each processor has at least 16 megabytes of memory. Similar to using supercomputers, the hypercube would require a number of days to complete a single simulation. To decrease the memory and computer-cost requirements, two basic alternatives can be explored. The

first alternative is to increase the number of processors working on a given grid; the second alternative is to reduce the computational grid size for a given simulation. However, by decreasing the size of the grid, the PSDNS will not resolve the smaller scales (subgrid), which will degrade the results. To capture these small scales with an appropriate model, the PSDNS approach becomes a large-eddy simulation (LES) code, or PSLES. As discussed and demonstrated by Piomelli et al. (1990), LES can reduce the computational grid and cost by an order of magnitude in comparison with DNS. If a subgrid-scale mode could accurately capture the physics in the boundary-layer flow, then a PSLES on a computational grid of 256 streamwise, 41 wall-normal, and 32 spanwise points, distributed on 32 processors, could potentially be computed. The cost for a single time step would be 14 sec for each processor. For the swept-wing problem described by Joslin and Streett (1994), the total computational cost on this LES grid would amount to 37.5 hr for each processor, or 1.5 days. Although the PSLES performance scalings are slightly underestimated because additional costs are involved with this model, PSLES seems plausible, and its use on parallel computers will be explored in the near future.

7. PSDNS ON IBM SP1

The IBM SP1 (Gropp et al., 1994) scalable parallel computer utilized in the presented performance study consists of 128 processing nodes. Each node is essentially an IBM RS/6000 model 370 workstation with a clock rate of 62.5 MHz. The local memory is 128 Mb, and the processor data and instruction cache is 32 Kb each. The individual nodes are connected by a multistage network that consists of high-performance switches (50μsec latency, 8.5 Mb bandwidth); each switch can support up to 16 nodes. The peak performance obtained with one multiplication and one addition on 64-bit floating point numbers per clock cycle is 125 Mflops for each processing node. However, in practice, a FORTRAN code delivers 15 to 75 Mflops. The access to Argonne's SP1 is controlled by a scheduler, which ensures that the requested node partition is operated in a dedicated mode. Thus, only the user application

and some necessary Unix demons are executed on the assigned processor partition.

Because the PSDNS code is based on the message-passing paradigm with explicit data distribution, good portability among a broad class of parallel computers is expected. The code was ported to the SP1 with only minor changes. To perform local FFT's in the spanwise direction n_z, the data must be remapped; an x-mapping allows the utilization of optimized serial FFT library routines (IBM Guide and Reference, 1992) in the z direction. The hypercube implementation of the PSDNS code relies on the *xor* algorithm for the global data exchange; the IBM implementation makes use of a global index routine provided by the AIX-parallel environment (IBM Parallel Programming Reference, 1993). As demonstrated in Figures 5 and 6, a significant performance gain can be achieved by utilizing a machine-specific basic linear algebra subprogram (BLAS) level 3 routine for the matrix-matrix multiplication. Because this routine is also available on the IBM as part of the ESSL library (IBM Guide and Reference 1992), the advantage can also be taken in the SP1 and SP2 implementations. The performance of the application code is further improved through appropriate selection of the compiler options. As a result, the run time of the serial code can be reduced by a factor of 2.3 compared with a compilation without any options. To demonstrate the significant impact of the library routines and the power of the SP1, consider the test problem above where the Cray Y/MP achieved 189 Mflops and a single node of the hypercube realized only 5 Mflops. For the same problem, a single node of the SP1 delivers 52.5 Mflops for the double-precision (i.e., 64-bit) computation.

The performance of the simulation code for a wide range of problem sizes with number of processors is documented. Further, a scaling analysis is presented in which each of the three problem dimensions are scaled individually and the number of processors is kept constant. The performance study is concluded with a discussion of a real-world large-scale simulation. Recall that thousands of time steps are required for a single simulation and that performance figures for only one time step of the PSDNS code are presented here.

7.1 Performance of PSDNS on SP1

Performance data are collected for the serial code on a single node of the SP1 and for the parallel code on up to 64 processing nodes. The chosen problem dimensions are representative of actual simulations that are currently performed on Cray-class supercomputers. To determine scaling information, the wall-normal dimension is fixed at $n_y = 41$ grid points, the streamwise grid is varied to include $n_x = 64$, 128, and 256, and the spanwise direction is varied from $n_z = 8$ to 128. Thus, experimental performance data can be obtained for problems that range from as small as 64 streamwise, 41 wall-normal, and 8 spanwise grid points (20,992 grid points) to a problem that is 64 times larger and contains 256 streamwise, 41 wall-normal, and 128 spanwise grid points (1.3 million grid points).

Similar to the hypercube measurements, the PSDNS code is instrumented with a set of timers to record separate performance data for different parts of the computation (the total and four dominating algorithmic kernels) and the communication. These measurements are wall-clock time. In addition, by including the idle time that results from the necessary synchronization points of the code in the time data, processor-independent performance figures can be obtained. Processor idle time is discussed below in conjunction with the large simulation for which the small serial fraction of the PSDNS code is experimentally determined.

Figures 14-16 show the computational and communication cost with variation in the streamwise and spanwise grids. The excellent scaling of the code on the SP1 are clearly evident; however, large communication costs relative to the computation costs are incurred because of the unbalanced architecture of the current SP1 (ie., network performance lags behind compute performance of processing nodes) on the one hand and the algorithmic communication penalty on the other hand. The communication penalty must be incurred in order to utilize highly optimized serial FFT routines in the spanwise direction. The good scaling of the communication cost with respect to the number of processors is noteworthy

because the communication that occurs in the PSDNS code involves a complete exchange, which represents a stringent test to the communication network.

The speedup of a parallel code for fixed-size problems is an important performance metric. For the PSDNS code, Figures 17-19 show the speedup for variations in the streamwise and spanwise grids with number of processors. The results demonstrate that the performance of the algorithm can be improved by either increasing the number of spanwise grid points or increasing the number of streamwise grid points. However, the code is less sensitive to changes in the size of the streamwise dimension than it is to changes in the number of spanwise grid points. For all test cases, the parallel efficiency of the PSDNS code stays above 40 percent, even when 64 processing nodes are utilized.

A theoretical speedup metric can be obtained by ignoring all communication costs. These metrics are also given in Figures 17-19. For large problems with 64 and 128 spanwise grid points, a superlinear speedup is observed. The superlinear theoretical speedup for the large problems is not a surprise. The good scalability of the algorithm, combined with the better memory access of the local portion of the distributed data structure is the most evident explanation. For a discussion of superlinear speedup, the reader is referred to Sun et al. (1994).

To further examine the performance of the simulation algorithm, the cost of the major numerical techniques and communication with number of processors is shown in Figures 20 and 21 for $n_x = 64$ and $n_x = 256$. The cost of each kernel relative to the computational cost shows that both the FFT and the matrix-matrix multiply each require roughly 30 percent of the total computing time on the SP1. When the number of processors is small, the cost for the FFT routine is higher than for the matrix-matrix multiply. The tridiagonal and pentadiagonal systems remain nearly constant at about 10 and 5 percent of the total computational cost, respectively. The cost for communication is relatively high, and for a large number of processors the communication costs reach levels as high as 80 to 90 percent

of the computational cost. The slight drop in the communicational costs between 32 to 64 processors in Figure 21 can be attributed to the fact that idle time is included in the overall computational cost.

The itemized speedup curves in Figures 20 and 21 show the following: a superlinear speedup for the FFT kernel; an ideal linear speedup for the tridiagonal solver; a nearly ideal speedup for the matrix-matrix multiply; and a moderate speedup for the pentadiagonal solver. The overall speedup of the computational fraction of the algorithm is close to that of the matrix-matrix multiply (the kernel that dominates the algorithm).

7.2 Complete Data Exchange

The performance of the complete data exchange portion of the algorithm deserves a closer analysis. The startup latency and bandwidth are the two most important quantities for evaluating a communication network. However, to analyze an application code, obtaining only an experimental bandwidth that includes the startup costs is sufficient. Before a meaningful discussion of the bandwidth achieved per processor can be given, both the actual message volume and the message size must be determined. The regular and throughout the simulation fixed communication pattern results in deterministic values for these quantities. The data-exchange routine is called 51 times during one time step of the algorithm. Thus, the message volume during one time step of the double-precision code (i.e., 8 bytes per data item) is given by $51 \times 8 \times n_x n_y (n_p - 1)\frac{n_z}{n_p}$. For example, the message volume is given in Figure 22 for $n_z = 32$, 64, and 128 spanwise grid points and up to $n_p = 32$ processing nodes. The message volume quickly approaches its asymptotic value of $51 \times 8 \times n_x n_y n_z$. Also shown in Figure 22, the message size of each individual message drops rapidly with the number of processors. The formula for determining the message size is $8 \times n_x n_y n_z / n_p^2$. To obtain the experimental bandwidth, the message volume must be divided by one-half of the required wall-clock time. The factor of one-half is used because the data must be sent as well as received. The experimental bandwidth achieved per processor is shown in Figure 23

as a function of the message size, which is given on a logarithmic scale. For large messages, the experimental bandwidth is close to the maximum value of the network bandwidth (8.5 Mb/sec). However, the startup latency reduces the observed bandwidth for smaller message sizes. The startup latency prevents the communication from being ideally scalable with the number of processors. A fixed-size problem distributed among a larger set of processors necessitates the exchange of more messages of shorter message size. Hence, network contention does not slow down the data exchange.

7.3 Scaling Analysis of the PSDNS Code

The scalability study is summarized in Figures 24-26. The test case with 64 streamwise, 41 wall-normal, and 32 spanwise grid points is used as the pivot point for this study, which is carried out on 16 processors. Figure 24 depicts the computational costs and slowdown for streamwise grid refinement. The FFT, the matrix-matrix multiply, and the communication cost are slightly superlinear; the overall computation time doubles for 128 grid points and quadruples for 256 points. Figure 25 gives the computational costs and slowdown for the individual kernels with a wall-normal grid refinement. The normalized count for the matrix-matrix multiply is $O(n_y^2)$ (Table 1), which can readily be seen in the slowdown curve for this major kernel. If n_y is doubled, an execution time that is four times larger results for the matrix-matrix multiply. The pentadiagonal solver also shows a slowdown that is more than linear; the FFT, the tridiagonal solver, and the communication costs scale linearly. The scaling of the overall computational cost follows the dominating kernel (matrix-matrix multiply); thus, it exhibits a slowdown of 2.8 when the wall-normal dimension is doubled. Finally, Figure 26 shows the costs and slowdown resulting from a spanwise grid refinement. A nearly linear scaling of the overall execution time (computation plus communication time) is observed for the range of spanwise dimensions (32 to 128 grid points). A closer look at the individual slowdown confirms the normalized count of $O(\log_2 n_z)$ (Table 1) for the FFT kernel. Although all other kernels scale linearly, the FFT kernel causes the computational

cost to follow the logarithmic increase of the FFT. We can expect that the overall time will increase at a rate that is greater than linear as the number of spanwise grid points is increased.

7.4 Memory Requirement of the PSDNS Code

The memory requirement of the PDSNS code limited the applications available for study on the hypercube; however, the SP1, with 128 Mb of core memory on each node, allows these much larger simulations. The largest grid that can be calculated on a single node of the SP1 contains 671 744 grid points (e.g, 128 streamwise, 41 wall-normal, and 128 spanwise points). The executable code for this calculation requires 110 Mb of core memory. A rough estimate for the memory requirement for the serial code then can be given as 170 bytes per grid point. However, because the memory requirement is a function of both the problem size and the number of processors, no simple relationship between executable code size and problem size can be given. In Figure 27, the memory sizes of the executable code for the large test case are presented. The reduction in the executable code size caused by the distributed data structure is clearly visible. The total memory requirement (i.e., the size of the parallel executable code times the number of processors) is larger than the memory needed by the serial code (due to the additional overhead that results from the data remapping routine).

7.5 Large-Scale Simulation on SP1

The swept-wing flow study by Joslin and Streett (1994) was used to determine the feasibility of using the hypercube for transition simulations. For a comparable SP1 feasibility estimate, a grid with 896 streamwise, 61 wall-normal, and 32 spanwise grid points was used. (In this parallel implementation, the streamwise grid size must be an integer factor of the number of processors.) Thus, the computational grid contains over 1.7 million grid points. The Cray Y/MP performs one time step of this simulation in 54 seconds and delivers 240 Mflops. Therefore, the computational expense of one time step is 12,960 Mflop. The large core memory of the SP1 allows a problem of the same size to be computed on as few as eight

processing nodes. The computational costs of the PSDNS algorithm with various number of SP1 processors are presented in Figure 28. The dashed line gives the total time required by the algorithm to perform one time step. If we compare these SP1 timings with the times required by a single node of the Cray Y/MP and Cray C-90 (marked with solid squares in the same plot) we see that the PSDNS code is highly competitive with these serial supercomputer performances for as few as 8 and 32 processing nodes of the SP1, respectively.

The actual measured execution time, which includes communication and idle time, is given in Table 4 for $n_p = 8$, 16, and 32 nodes of the SP1. An idealized execution time can be obtained by subtracting those times that each processing node spends idle or in communication. Because the serial part of the algorithm is performed only on the first node, two idealized times must be recorded; one value for the first node and another for the remaining processing nodes. The performance of the PSDNS code (in Mflops), based on the actual time and the idealized time, is given in Table 4. The idealized performance of 55 Mflops per processor is noteworthy. Recall that even though the peak performance of a single node is 128 Mflops, 15 to 75 Mflops are generally observed for actual applications. The last column of Table 4 shows the memory requirements of the executable code; these numbers show that the code is far from reaching the local memory limit of 128 Mb.

A performance summary for the communication part of the algorithm is given in Table 5. The presented values for the message volume and the message size are calculated with the formula presented in section 7.2. The measured experimental bandwidth agrees well with the values shown in Figure 23 for the large test case.

By using the idealized execution times for node 1 and for nodes 2–n_p in Table 4 (the idealized execution time excludes all idle time and communication costs), one can determine experimentally the serial and parallel fractions of the PSDNS algorithm. The difference between the two execution times is the time spent in the serial part of the parallel algorithm. If we multiply the execution time of node 2 by the number of processors, we obtain the

execution time of the parallel portion. In this context, total execution time is equal to the time spent by all processing nodes combined. If we normalize the time spent in the serial and parallel portions of the algorithm with the total execution time we obtain the serial fraction s and the parallel fraction p, respectively. Surprisingly, the serial fraction is only 1.4 percent, and the parallel fraction is 98.6 percent of the total. Amdahl's law (Amdahl, 1967) provides a theoretical speedup that is derived from these two quantities:

$$S_p = \frac{1}{s + \frac{p}{n_p}} \tag{7.1}$$

For 8, 16, and 32 processing nodes, the theoretical speedup S_p of the PSDNS code is 7.29, 13.22, and 22.32, respectively. In the limit of $n_p \to \infty$, the speedup asymptotically reaches the value $1/s$. Even though the parallel granularity of the PSDNS code is restricted for this problem to 32 processing nodes, the theoretical maximum speedup is 71.

8. PERFORMANCE OF PSDNS ON IBM SP2

Although the next-generation parallel computer from IBM, called the SP2 (IBM Press Release, 1994; Saini, 1994), is identical to the SP1 architecture, its node performance has more than doubled and the communication network bandwidth has increased fourfold. For the SP2, the increase in communication bandwidth relative to the computing performance should provide a better balanced system, which should further improve the performance results of PSDNS.

The IBM SP2 [Ref. NAS] located at NASA Langley Research Center for which performance measures are presented contains 48 processing nodes (IBM RS/6000 Model 590). Each of these nodes has at least 128 Mb of main memory and 2 Gb of disk space. The nodes are based on a POWER2 multi-chip RISC processor containing two integer and two floating-point computation units. The clock rate is 66.7 MHz and the data cache is 256 kb. Each floating-point unit can finish two 64-bit operations (one multiplication and one addition) per clock cycle. This results in a theoretical peak performance of 267 MFlops. The nodes of the

SP2 are inter-connected by a high-performance switch (TB switch level 2 at NASA Langley is approximately 45 μsec latency, 34 Mb/sec bandwidth)

The large-application simulation for which the SP1 performance is given in Figure 28, has been repeated on the 48 nodes SP2 at NASA Langley Research Center. The computational costs of the PDNS simulation with variation in the number of SP2 processors is compared in Figure 29 with the Cray Y/MP and Cray C-90 timings. A preprocessing step prior to the compilation of the FORTRAN code (invoked by the IBM xlf compiler as option -P), which has been instrumental in achieving high computational performance on the nodes of the SP1 has not been made available on the SP2 at NASA Langley at the time of this report. Therefore, the performance numbers presented in Figure 29 do not demonstrate the full potential of the SP2 hardware, rather the results reflect the limitations due to the current software support. In spite of the software limitation, the improvement in communication performance on the SP2 is clearly visible in Figure 29. While the time spent for computation could not be noticeably reduced on the SP2 compared to the SP1 (above mentioned software limitations), the communication improved nearly 5 fold. This decrease in communication cost has made the SP2 highly competitive with the Cray C-90 in terms of compute-power per dollar. One can expected that future software releases will take advantage of the POWER2 RISC architecture and thus the SP2 will deliver higher compute performance as well.

9. CONCLUDING REMARKS

The performance of a recently implemented parallel spatial direct numerical simulation (PSDNS) approach on the Intel iPSC/860 hypercube and IBM SP1 and SP2 is documented. The PSDNS results are in good agreement with linear stability theory for a small-amplitude test case, which serves as the initial validation of the code on the parallel computer.

The results for the hypercube show that the work is well balanced between the processors (except the first node, which will have approximately 15 to 20 percent larger work load because of the numerical techniques employed). Furthermore, a speedup with a factor of

4 to 5 was obtained by using machine-dependent libraries rather than standard Fortran routines. The feasibility study of using the PSDNS on the hypercube for transitional flows indicates that such simulations could be conducted provided each processor had a minimum of 16 megabytes of memory; however, the computational cost is approximately twice that of the Cray-2 supercomputer. Furthermore, the use of a subgrid-scale model to compute large-eddy simulations (PSLES) would reduce the computational cost by an order of magnitude compared with PSDNS. Large-eddy simulations could readily be used to study transition on the hypercube at a reasonable computational cost.

The performance study of the PSDNS code on the larger and more powerful IBM SP1 and SP2 distributed memory machines yields results which emphasize the advantages of parallel machines. The scalability information obtained by independently varying the number of grid points in each of the three problem dimensions confirms the theoretical scaling analysis. It is demonstrated that only eight nodes of the SP1 are needed to perform such a large-application simulation in the same amount of time as required by a Cray Y/MP. Furthermore, the utilization of 32 processing nodes on the SP1 reduces the execution time to roughly one-third (to nearly one-fifth on the SP2 with 32 nodes). Both the parallel efficiency of the PSDNS code (above 40 percent for all performed calculations) on the SP1 and the high serial performance of 52–56 Mflops on a single SP1 node (45 percent of theoretical peak performance) contribute to this success. While computations with 32 SP1 processors become highly competitive to the advanced Cray C-90, a 32 SP2 processor computation exceeds the C-90 performance.

ACKNOWLEDGMENTS

This research was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-19480 while the second and third authors were in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681. Also, the authors gratefully acknowledge

use of the Argonne National Laboratory High-Performance Computing Research Facility (HPCRF). The HPCRF is funded principally by the U.S. Department of Energy Office of Scientific Computing.

REFERENCES

Amdahl, G. (1967). Validity of the single-processor approach to achieving large scale computing capabilities. *Proceedings of the AFIPS Conference*, 483–485.

Bestek, H., Thumm, A., and Fasel, H. F. (1992). Numerical investigation of later stages of transition in transonic boundary layers. In *First European Forum on Laminar Flow Technology*, March 16-18, 1992. Hamburg, Germany.

Bokhari, S.H. (1991). Complete Exchange on the iPSC/860. *NASA CR 187498.*

Bushnell, D. M., Hefner, J. N., and Ash, R. L. (1977). Effect of compliant wall motion on turbulent boundary layers, *Phys. Fluids*, **20**(10), s31-48.

Canuto, C., Hussaini, M. Y., Quarteroni, A., and Zang, T. A. (1988). *Spectral Methods in Fluid Dynamics.* Springer-Verlag, New York.

Danabasoglu, G., Biringen, S., and Streett, C. L. (1990). Numerical simulation of spatially-evolving instability control in plane channel flow. *AIAA Paper No. 90-1530.*

Danabasoglu, G., Biringen, S., and Streett, C. L. (1991). Spatial simulation of instability control by periodic suction blowing. *Phys. Fluids A*, **3**(9), 2138-2147.

Dongarra, J.J., DuCroz, J., Hammarling, S., and Duff, I. (1990). A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, **16**(1), 18–28.

Eidson, T. M., and Erlebacher, G. (1995). Implementation of a fully-balanced periodic tridiagonal solver on a parallel distributed memory architecture, (to appear) *Concurrency: Practice and Experience.*

Fasel, H. F. (1976). Investigation of the stability of boundary layers by a finite-difference model of the Navier-Stokes equations. *J. Fluid Mech.*, **78**, 355-383.

Fasel, H. F., Rist, U., and Konzelmann, U. (1990). Numerical investigation of the three-dimensional development in boundary-layer transition. *AIAA J.*, **28**(1), 29-37.

Fischer, P. F., Ho, L.-W., Karniadakis, G. E., Ronquist, E. M., and Patera, A. T. (1988). Recent advances in parallel spectral element simulation of unsteady incompressible flows. *Computers and Structures*, **30**(1-2), 217-231.

Fischer, P.F., and Patera, A.T. (1994). Parallel simulation of viscous incompressible flows. *Annu. Rev. Fluid. Mech.*, **26**, 483–527.

Gropp, W., Lusk, E., and Pieper, S.C. (1994). Users Guide for the Argonne National Laboratory IBM SP1. Argonne National Laboratory.

Hanebutte, U.R., Joslin, R.D. and Zubair, M., (1994) Scalability of a parallel spatial direct numerical simulation Code on the IBM SP1 parallel supercomputer. *NASA CR 194975.*

Henderson, R., and Karniadakis, G. E. (1991). Hybrid spectral-element-low-order methods for incompressible flows, *J. Sci. Comp.*, **6**(2), 79-99.

Herbert, Th., and Bertolotti, F. P. (1987). Stability analysis of nonparallel boundary layers. *Bull. Am. Phys. Soc.*, **32**, 2079.

IBM Parallel Programming Reference (1993). AIX Parallel Environment, Release 1.0. SH26-7228-00.

IBM Guide and Reference (1992). Engineering and Scientific Subroutine Library, Version 2. SH23-0526-00.

IBM Press Release (1994). Cornell Theory Center First To Receive IBM's Newest High Performance Power Parallel System. Obtained via World Wide Web, April 5, 1994.

Jackson, E., She, Z.-S., and Orszag, S. A. (1991). A case study in parallel computing: I. Homogeneous turbulence on a hypercube, *J. Sci. Comp.*, **6**(1), 27-45.

Joslin, R. D., Streett, C. L., and Chang, C.-L. (1992). Validation of three-dimensional incompressible spatial direct numerical simulation code–a comparison with linear stability and parabolic stability equations theories for boundary-layer transition on a flat plate. *NASA TP 3205.*

Joslin, R. D., Streett, C. L., and Chang, C.-L. (1993). Spatial DNS of boundary-layer transition mechanisms: Validation of PSE theory. *Theoret. Comput. Fluid Dynamics*, **4**(6), 271-288.

Joslin, R. D. and Streett, C. L. (1994). The role of stationary crossflow vortices in boundary-layer transition on swept wings. *Phys. Fluids*, **6**(10), 3442-53.

Joslin, R.D., and Zubair, M. (1993). Parallel spatial direct numerical simulations on the Intel iPSC/860 hypercube. *NASA CR 191513.*

Kleiser, L., and Zang, T. A. (1991). Numerical simulation of transition in wall-bounded shear flows. *Ann. Rev. Fluid Mech.*, **23**, 495-537.

Laurien, E., and Kleiser, L. (1989). Numerical simulation of boundary-layer transition and transition control. *J. Fluid Mech.*, **199**, 403-440.

Lele, S. K. (1992). Compact finite difference schemes with spectral-like resolution. *J. Comput. Phys.*, **103**, 16-42.

Lynch, R. E., Rice, J. R., and Thomas, D. H. (1964). Direct solution of partial difference equations by tensor product methods. *Num. Math.*, **6**, 185-199.

Malik, M. R. and Li, F., Three-dimensional boundary layer stability and transition. *Aerotech '92*, Paper No. 921991, 1992.

Moin, P., and Kim, J. (1982). Numerical investigation of turbulent channel flow. *J. Fluid Mech.* **118**, 341–377.

NAS: Numerical Aerodynamic Simulation (NAS) NASA Ames Research Center, Overview of the NAS SP2. Obtained via the World Wide Web, last updated Dec. 14, 1994.

Ortega, J. M. and Voigt, R. G. (1988). A bibliography on parallel and vector numerical algorithms. *ICASE Interim Report 6.*

Otto, J. C. (1993). Parallel execution of a three-dimensional, chemically reacting, Navier-Stokes code on distributed-memory machines. *AIAA Paper 93-3307-CP.*

Piomelli, U., Zang, T. A., Speziale, C. G., and Hussaini, M. Y. (1990), On the large-eddy simulation of transitional wall-bounded flows. *Phys. Fluids A*, **2**(2), 257-265.

Rai, M. M., and Moin, P. (1991a). Direct numerical simulation of transition and turbulence in a spatially-evolving boundary layer. *AIAA Paper No. 91-1607.*

Rai, M. M., and Moin, P. (1991b). Direct numerical simulation of turbulent flow using finite-difference schemes. *J. Comput. Phys.*, **96**, 15-53.

Reed, H. L. (1994). Direct numerical simulation of Transition: The spatial approach. In *Progress in Transition Modelling*, AGARD-R-793, 6.1-6.46.

Saini, S. (1994). The IBM SP2: hardware, software, porting, and optimization overview. Numerical Aerodynamics Simulation Program, NASA Ames Research Center, NAS User Seminar, July 27, 1994.

Smith, A. M. O., and Gamberoni, N. (1956). Transition, pressure gradients, and stability theory. *Douglas Aircraft Company Report No. ES-26388.*

Spalart, P. R. (1989). Direct numerical study of leading-edge contamination. In *Fluid Dynamics of Three-Dimensional Turbulent Shear Flows and Transition*, AGARD-CP-438, 5.1-5.13.

Streett, C. L., and Macaraeg, M. G. (1989). Spectral multi-domain for large-scale fluid dynamic simulations. *Int. J. Appl. Numer. Math.*, **6**, 123-140.

Streett, C. L., and Hussaini, M. Y. (1991). A numerical simulation of the appearance of chaos in finite-length Taylor-Couette flow. *Appl. Numer. Math.*, **7**, 41-71.

Sun, X.-H., and Zhu, J. (1994). Shared virtual memory and generalized speedup. *NASA CR 191592.*

Van Ingen, J. L. (1956). A suggested semi-empirical method for the calculation of the boundary-layer transition region. *University of Delft Report VTH-74*, Department of Aerospace Engineering, Delft, The Netherlands.

Williamson, J. H. (1980). Low-storage Runge-Kutta schemes. *J. Comput. Phys.*, **35**(1), 48-56.

Zang, T. A., and Hussaini, M. Y. (1987). Numerical simulation of nonlinear interactions in channel and boundary-layer transition. *Nonlinear Wave Interactions in Fluids*, **87**, 131-145.

Zang, T. A., and Hussaini, M. Y. (1990). Multiple paths to subharmonic laminar breakdown in a boundary layer. *Phys. Rev. Lett.*, **64**, 641-644.

Figure 1. Computational domain of boundary-layer transition problem.

Figure 2. Computational domain.

Figure 3. Global remapping results in local FFT's in spanwise direction.

Figure 4. Amplitude growth with downstream distance for a Tollmien-Schlichting wave with initial amplitude $u_o = 0.0001\%$, Reynolds number $R_{\delta_o^*} = 900$, and frequency $\omega = 0.0774$.

Figure 5. Computational and communication cost with number of processors for initial implementation of PSDNS, where $n_x = 64$ and $n_y = 41$.

Figure 6. Computational and communication cost with number of processors for optimized PSDNS, where $n_x = 64$ and $n_y = 41$.

Figure 7. Computational-cost breakdown for each processor for initial implementation of PSDNS, where $n_x = 64$, $n_y = 41$, and $n_z = 8$.

Figure 8. Computational-cost breakdown and speedup with number of processors for PSDNS, where $n_x = 64$, $n_y = 41$, and $n_z = 64$.

Figure 9. Computational-cost breakdown for one (left) and four (right) $x - y$ planes per processor with number of processors for PSDNS, where $n_x = 64$ and $n_y = 41$.

Figure 10. Computational and communication cost with number of processors for PSDNS, where $n_x = 128$ and $n_y = 41$.

Figure 11. Computational-cost breakdown and speedup with spanwise grid for PSDNS, where $n_x = 128$, $n_y = 41$, and $n_p = 8$.

Figure 12. Computational-cost breakdown and slowdown with streamwise grid refinement for PSDNS, where $n_y = 41$, $n_z = 32$, and $n_p = 16$.

Figure 13. Computational-cost breakdown and slowdown with wall-normal grid refinement for PSDNS, where $n_x = 64$, $n_z = 32$, and $n_p = 16$.

Figure 14. Computational and communication cost with number of processors for PSDNS, where $n_x = 64$ and $n_y = 41$.

Figure 15. Computational and communication cost with number of processors for PSDNS, where $n_x = 128$ and $n_y = 41$.

Figure 16. Computational and communication cost with number of processors for PSDNS, where $n_x = 256$ and $n_y = 41$.

Figure 17. Computational speedup with spanwise grid for PSDNS, where $n_x = 64$ and $n_y = 41$.

Figure 18. Computational speedup with spanwise grid for PSDNS, where $n_x = 128$ and $n_y = 41$.

Figure 19. Computational speedup with spanwise grid for PSDNS, where $n_x = 256$ and $n_y = 41$.

Figure 20. Computational-cost breakdown and speedup with number of processors for PSDNS, where $n_x = 64$, $n_y = 41$, and $n_z = 64$.

Figure 21. Computational-cost breakdown and speedup with number of processors for PSDNS, where $n_x = 256$, $n_y = 41$, and $n_z = 64$.

Figure 22. Message volume and size for one timestep for PSDNS, where $n_x = 64$ and $n_y = 41$.

Figure 23. Communication bandwidth per processor as function of message size, where $n_x = 64$ and $n_y = 41$.

Figure 24. Computational-cost breakdown and slowdown with streamwise grid refinement for PSDNS, where $n_y = 41$, $n_z = 32$, and $n_p = 16$.

Figure 25. Computational-cost breakdown and slowdown with wall-normal grid refinement for PSDNS, where $n_x = 64$, $n_z = 32$, and $n_p = 16$.

Figure 26. Computational-cost breakdown and slowdown with spanwise grid refinement for PSDNS, where $n_x = 64$, $n_y = 41$, and $n_p = 16$.

Figure 27. Memory size of executable with spanwise grid for PSDNS, where $n_x = 256$ and $n_y = 41$.

Figure 28. Computational and communication cost with number of processors for PSDNS on IBM SP1, where $n_x = 896$, $n_y = 61$, $n_z = 32$.

Figure 29. Computational and communication cost with number of processors for PSDNS on IBM SP2, where $n_x = 896$, $n_y = 61$, $n_z = 32$.

Table 1. Operation Counts for the Major Kernels

Kernel	Operation count (oc)	Normalization count $= \text{oc}/n_x n_y n_z$
MAT-MAT	$O(n_x n_y^3 n_z)$	$O(n_y^2)$
FFT	$O(n_x n_y n_z log_2 n_z)$	$O(\log_2 n_z)$
TRIDIAG	$O(n_x n_y n_z)$	$O(1)$
PENTADIAG	$O(n_x n_y n_z)$	$O(1)$

Table 2. Major Kernel Executions With Different Data Mappings

Kernel	x-mapping	y-mapping	z-mapping
MAT-MAT	global	global	local
FFT	local	local	global
TRIDIAG	global	local	local
PENTADIAG	global	local	local

Table 3. Illustration of the *xor* Scheme for Complete Exchanges

node steps	0	1	2	3	4	5	6	7
1	1	0	3	2	5	4	7	6
2	2	3	0	1	6	7	4	5
3	3	2	1	0	7	6	5	4
4	4	5	6	7	0	1	2	3
5	5	4	7	6	1	0	3	2
6	6	7	4	5	2	3	0	1
7	7	6	5	4	3	2	1	0

Table 4. Performance of Application Simulation on 8, 16, and 32 Nodes of SP1

no. of proc. n_p	Actual sec	Idealized sec		Actual MFLOPS		Idealized MFLOPS		Exec. Mbytes
		1	2-n_p	/proc.	Total	/proc.	Total	
8	53.75	32.4	29.1	30	241	55	440	79
16	29.75	17.7	14.5	27	436	55	880	60
32	18.75	10.2	7.1	22	691	56	1760	50

Table 5. Total Data Exchange for Single Iteration Step of Large Application on SP1

no. of proc. n_p	Comm, sec	Message		Bandwidth	
		Vol. Mb	Size Mb	Total Mb/sec	/proc. Mb/sec
8	19.0	595	0.208	63	7.9
16	11.0	638	0.052	116	7.3
32	7.0	659	0.013	176	5.5

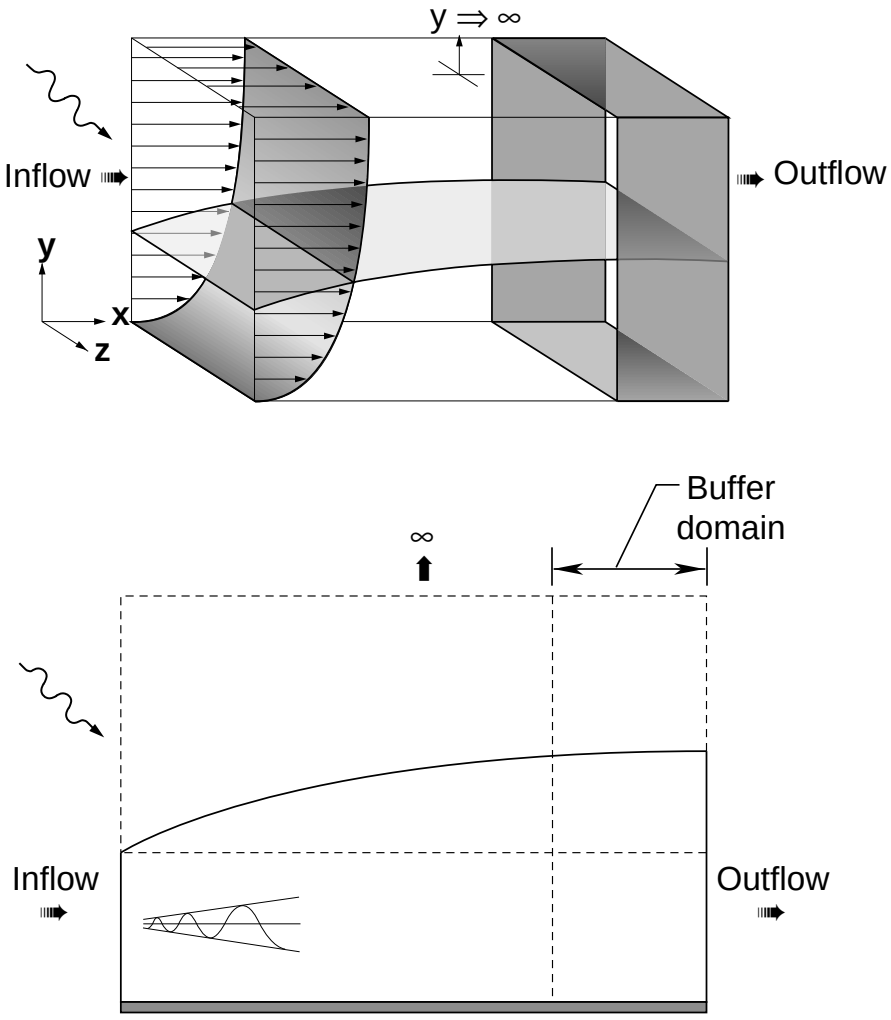
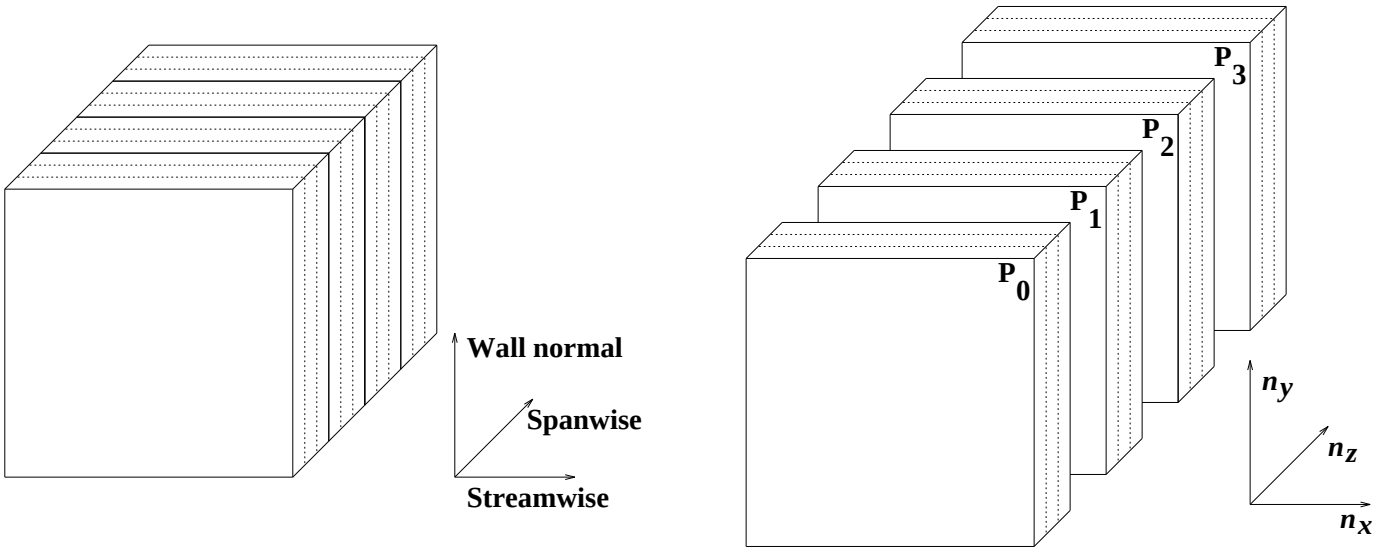


Figure 1. Computational domain of boundary-layer transition problem.



Entire domain. Mapped onto four processors.

Figure 2. Computational domain.

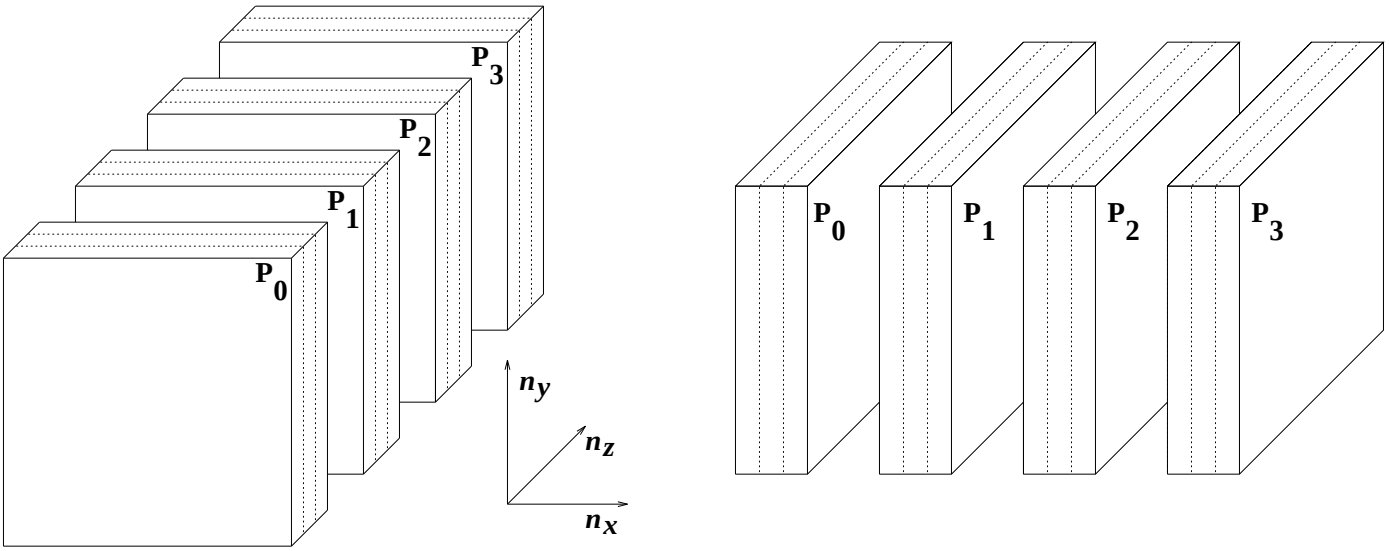


Figure 3. Global remapping results in local FFT's in spanwise direction.

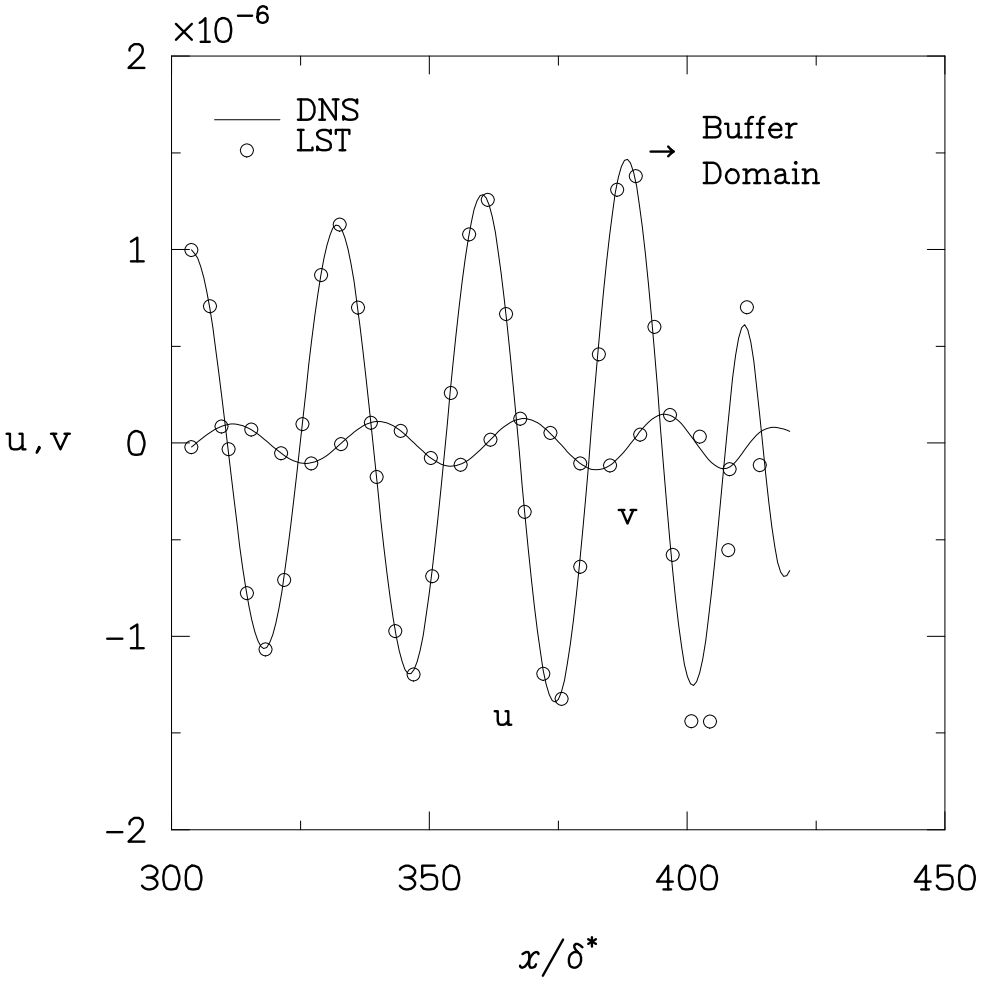


Figure 4. Amplitude growth with downstream distance for a Tollmien-Schlichting wave with initial amplitude $u_o = 0.0001\%$, Reynolds number $R_{\delta_o^*} = 900$, and frequency $\omega = 0.0774$.

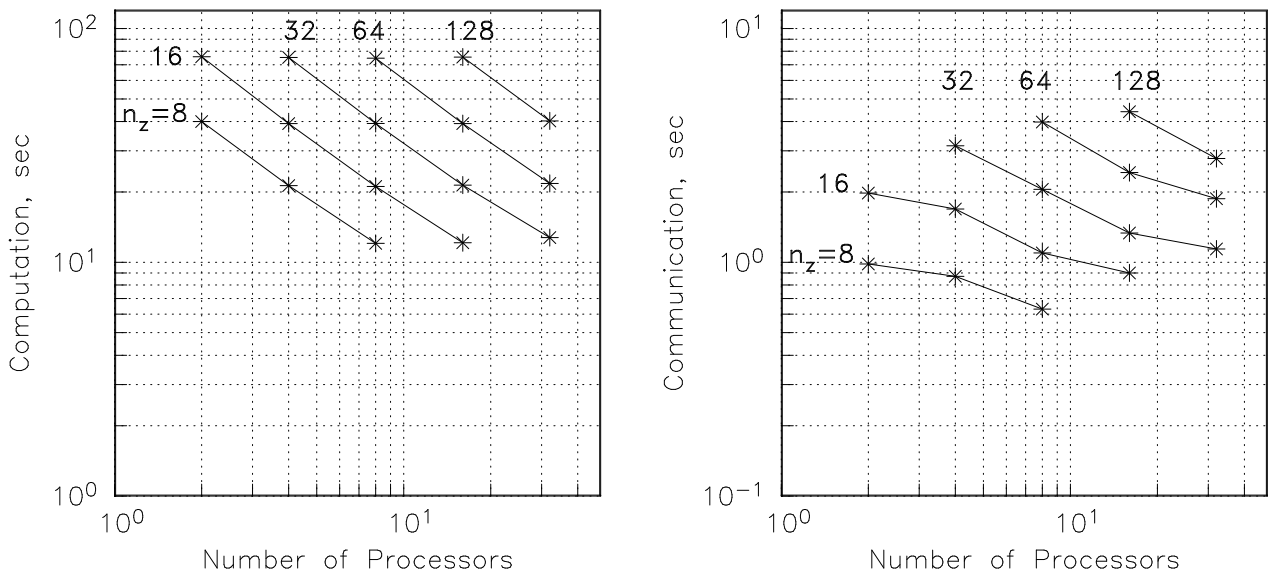


Figure 5. Computational and communication cost with number of processors for initial implementation of PSDNS, where $n_x = 64$ and $n_y = 41$.

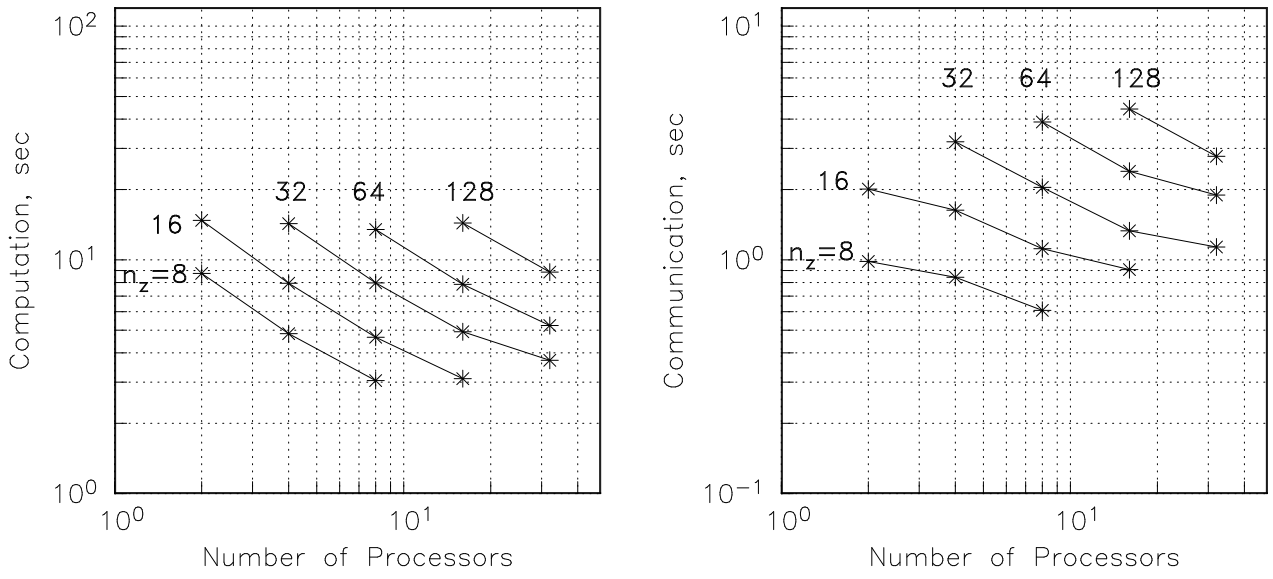


Figure 6. Computational and communication cost with number of processors for optimized PSDNS, where $n_x = 64$ and $n_y = 41$.

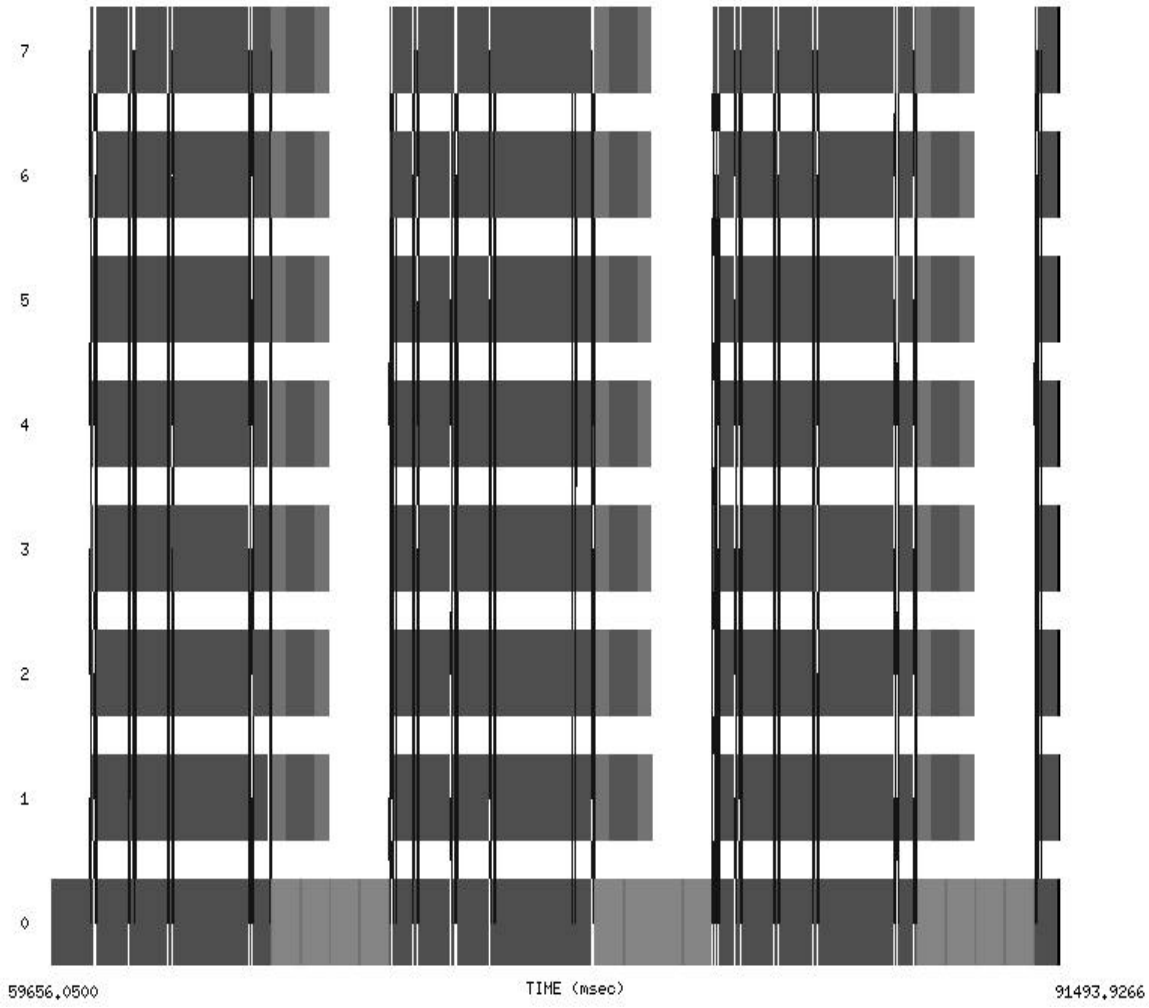


Figure 7. Computational-cost breakdown for each processor for initial implementation of PSDNS, where $n_x = 64$, $n_y = 41$, and $n_z = 8$.

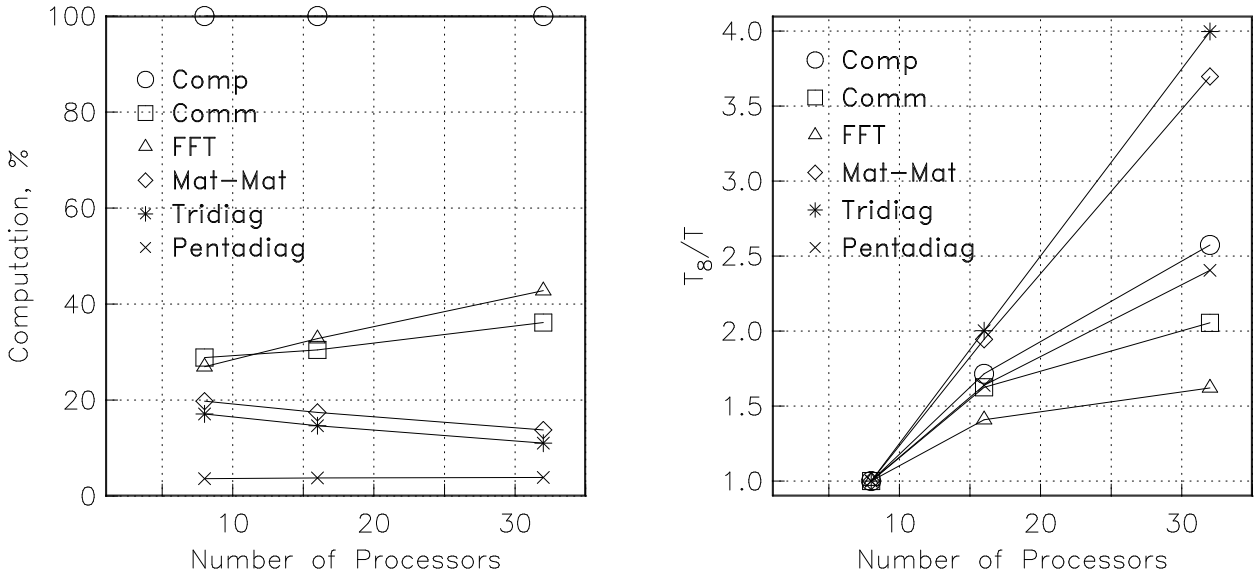


Figure 8. Computational-cost breakdown and speedup with number of processors for PSDNS, where $n_x = 64$, $n_y = 41$, and $n_z = 64$.

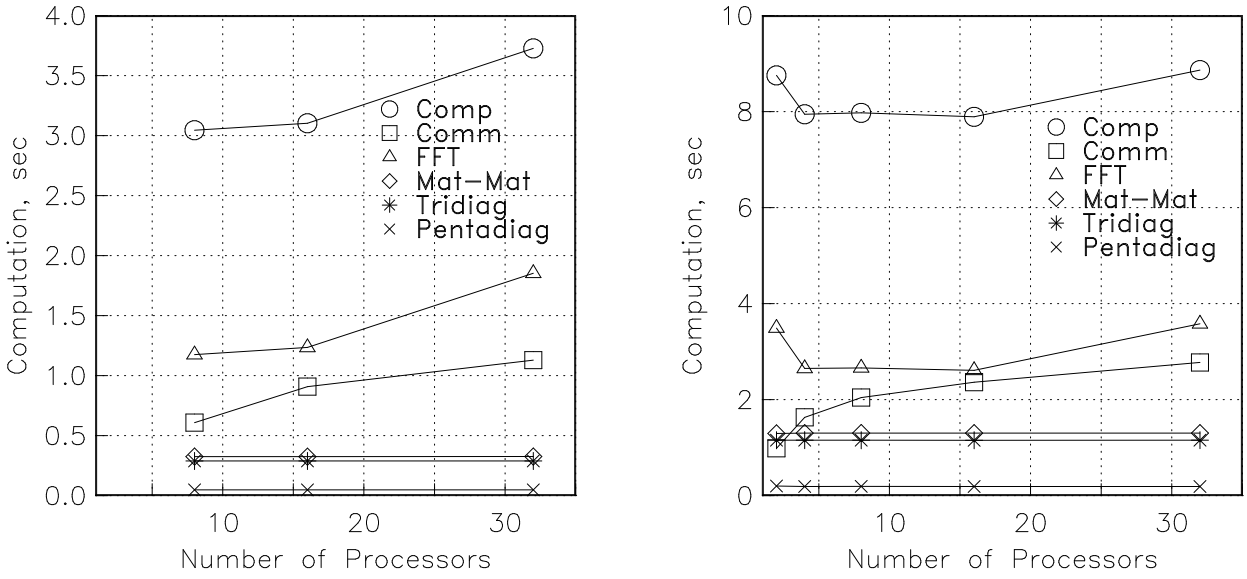


Figure 9. Computational-cost breakdown for one (left) and four (right) $x - y$ planes per processor with number of processors for PSDNS, where $n_x = 64$ and $n_y = 41$.

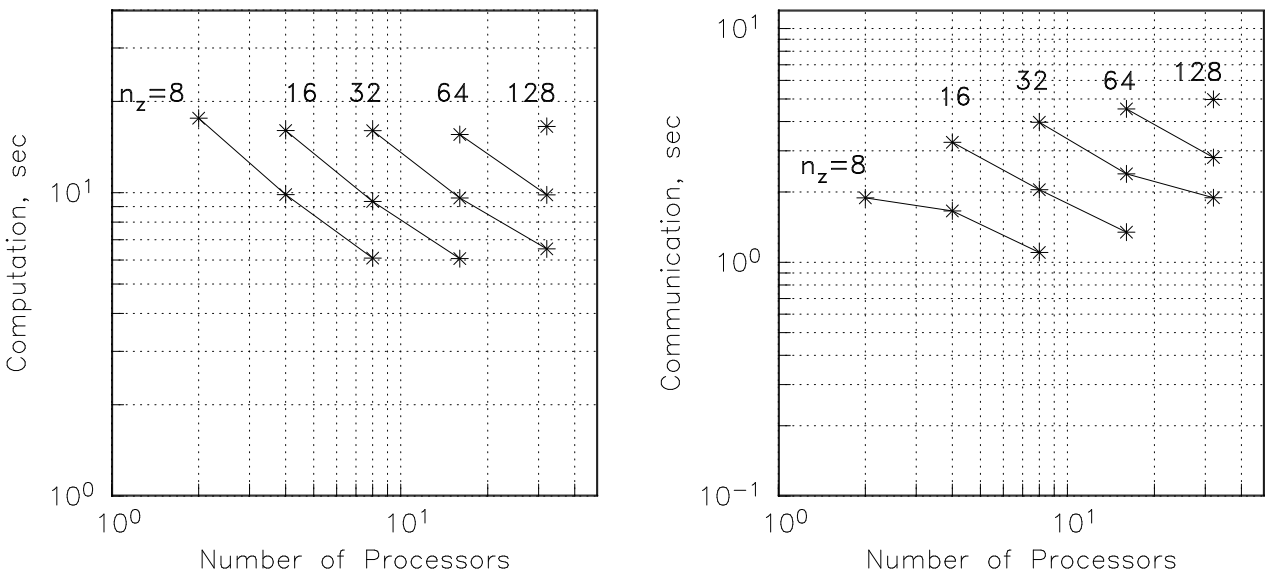


Figure 10. Computational and communication cost with number of processors for PSDNS, where $n_x = 128$ and $n_y = 41$.

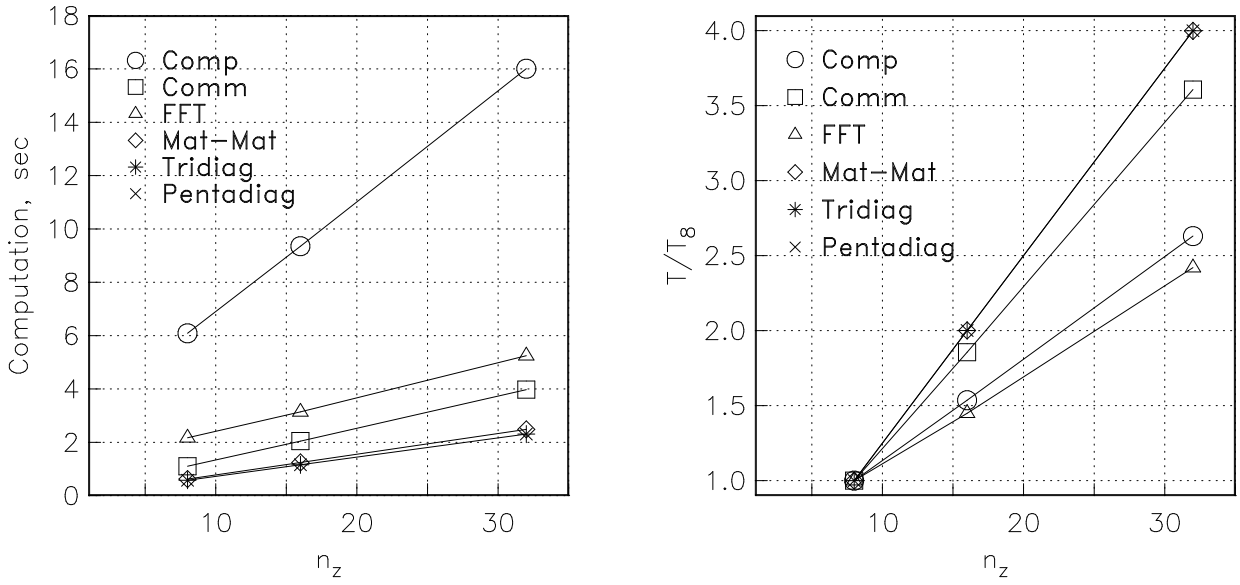


Figure 11. Computational-cost breakdown and speedup with spanwise grid for PSDNS, where $n_x = 128$, $n_y = 41$, and $n_p = 8$.

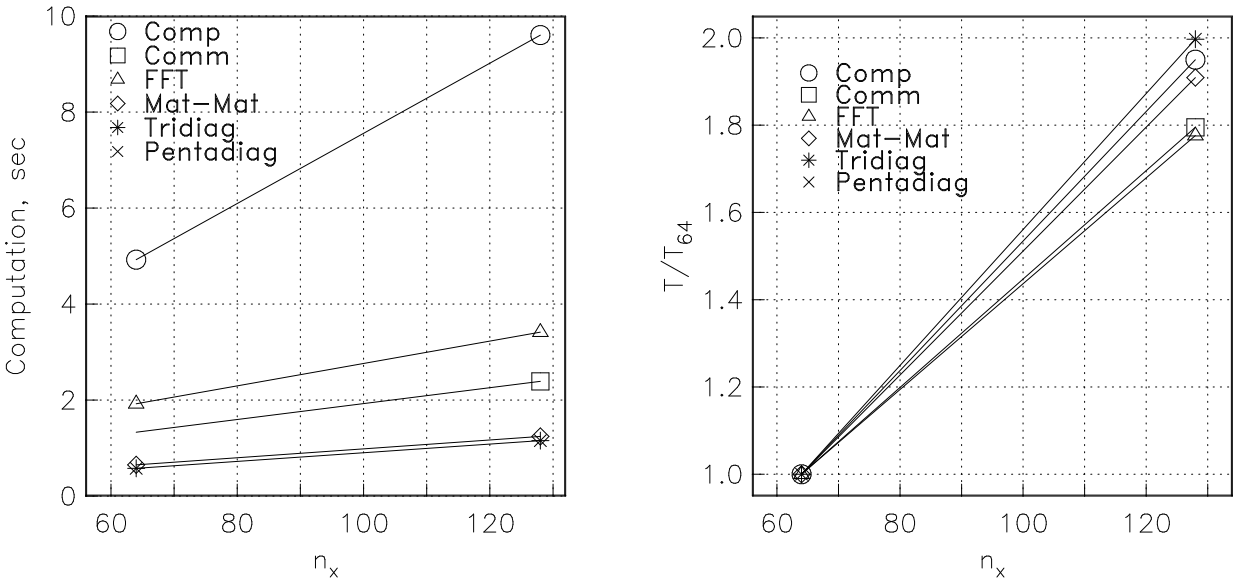


Figure 12. Computational-cost breakdown and slowdown with streamwise grid refinement for PSDNS, where $n_y = 41$, $n_z = 32$, and $n_p = 16$.

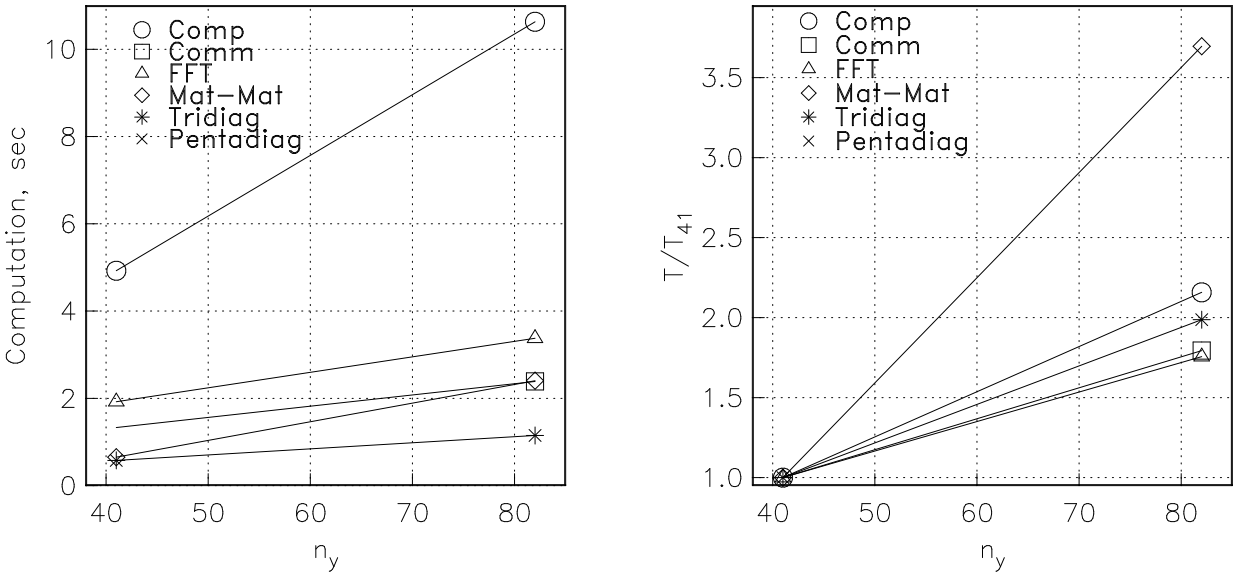


Figure 13. Computational-cost breakdown and slowdown with wall-normal grid refinement for PSDNS, where $n_x = 64$, $n_z = 32$, and $n_p = 16$.

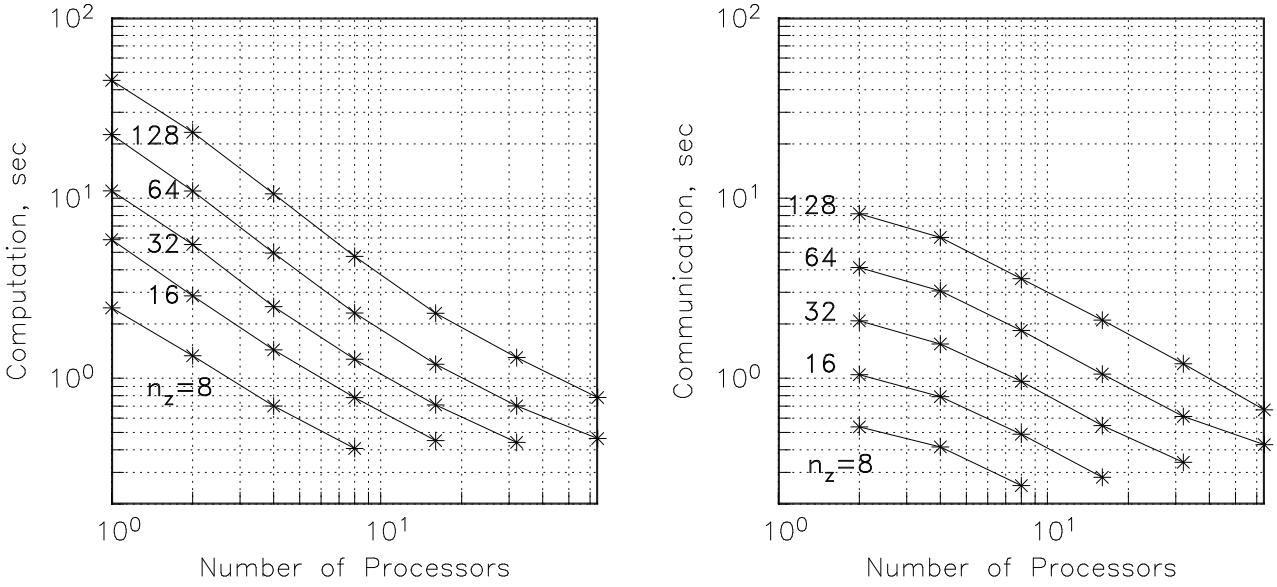


Figure 14. Computational and communication cost with number of processors for PSDNS, where $n_x = 64$ and $n_y = 41$.

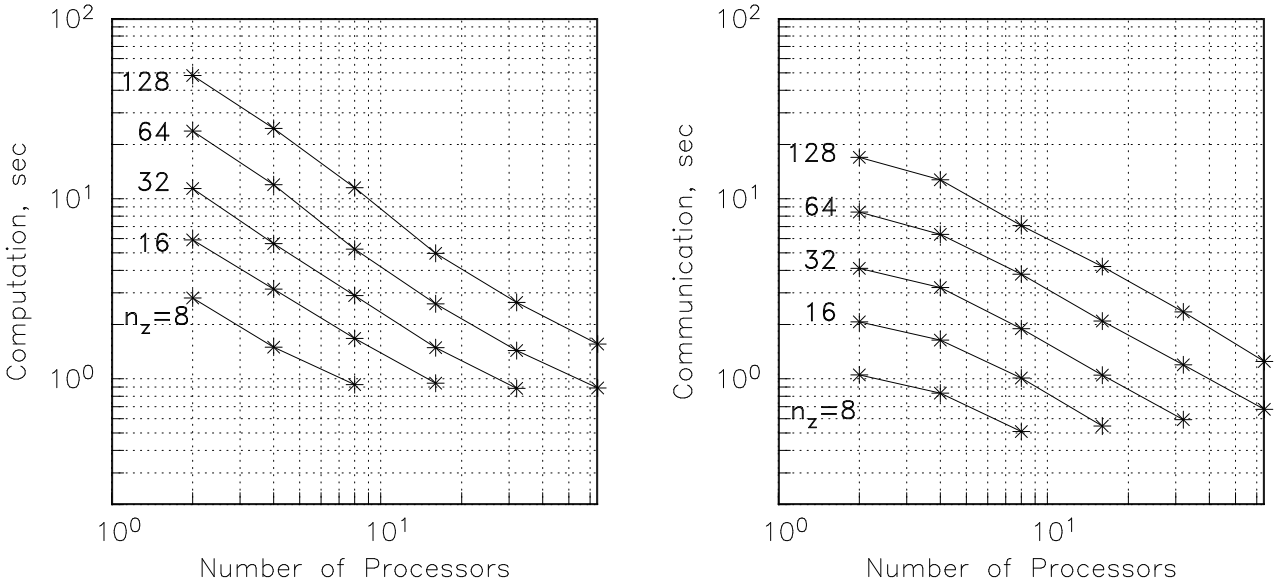


Figure 15. Computational and communication cost with number of processors for PSDNS, where $n_x = 128$ and $n_y = 41$.

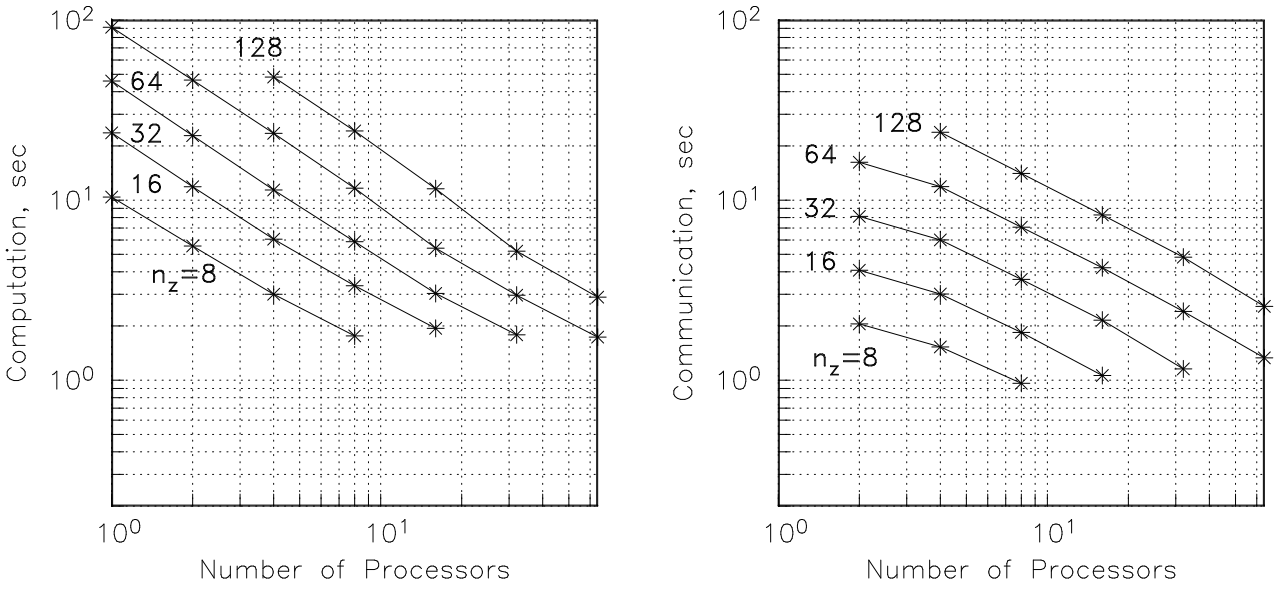


Figure 16. Computational and communication cost with number of processors for PSDNS, where $n_x = 256$ and $n_y = 41$.

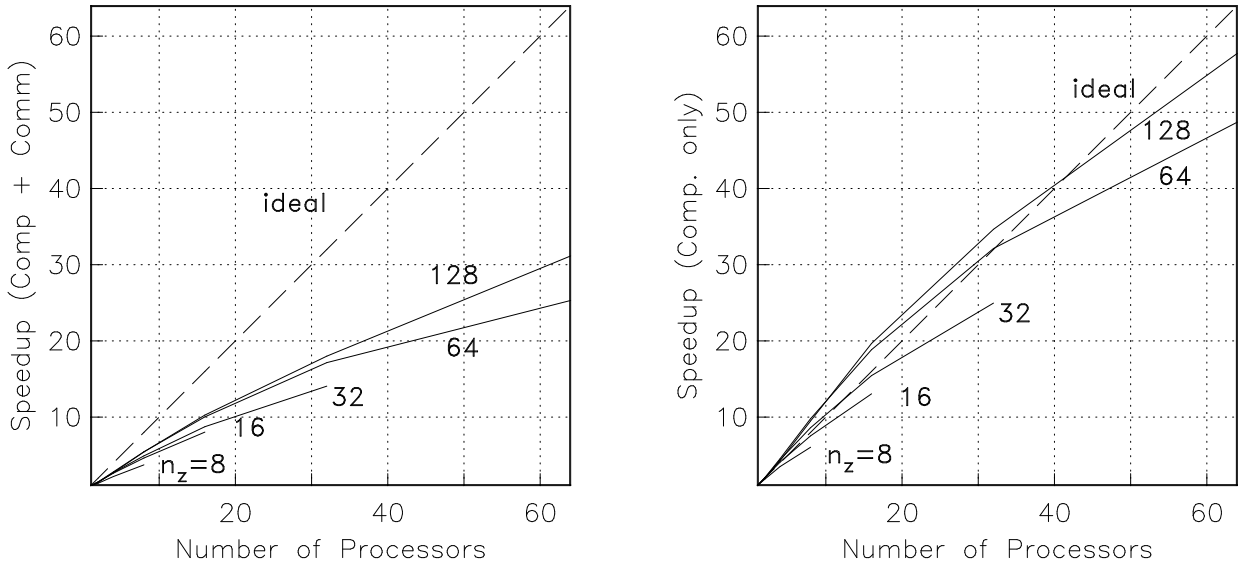


Figure 17. Computational speedup with spanwise grid for PSDNS, where $n_x = 64$ and $n_y = 41$.

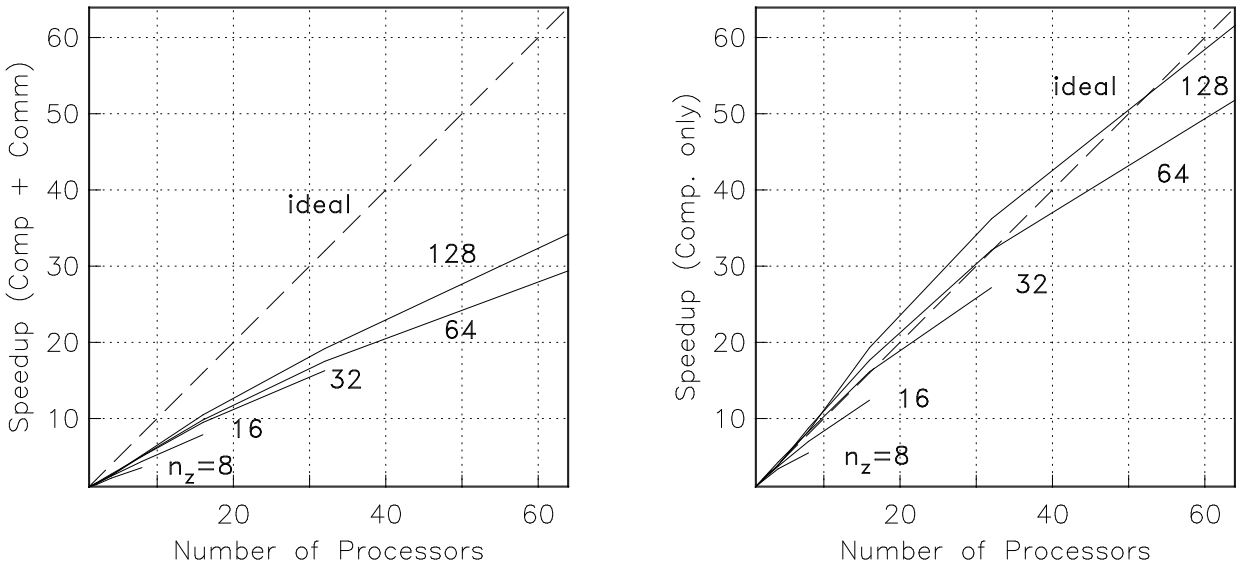


Figure 18. Computational speedup with spanwise grid for PSDNS, where $n_x = 128$ and $n_y = 41$.

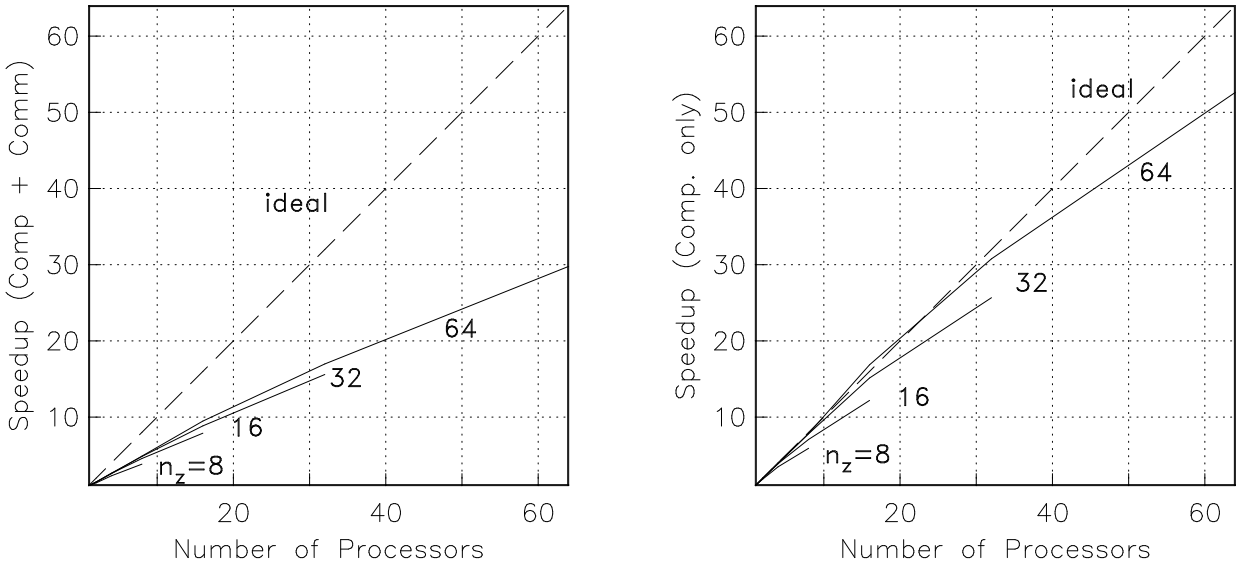


Figure 19. Computational speedup with spanwise grid for PSDNS, where $n_x = 256$ and $n_y = 41$.

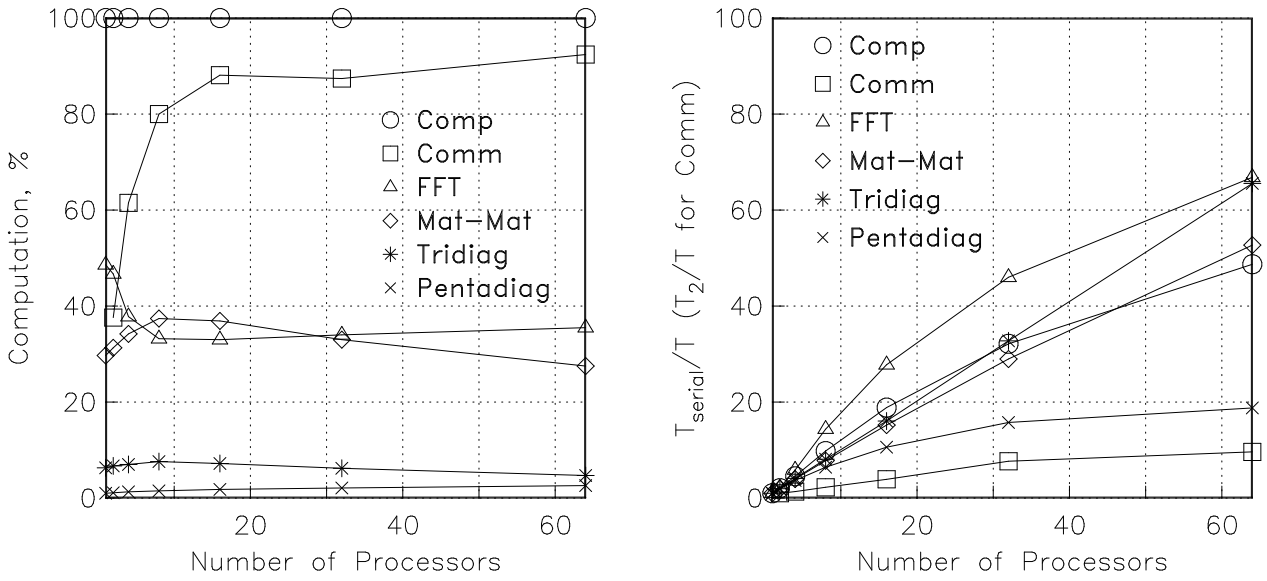


Figure 20. Computational-cost breakdown and speedup with number of processors for PS-DNS, where $n_x = 64$, $n_y = 41$, and $n_z = 64$.

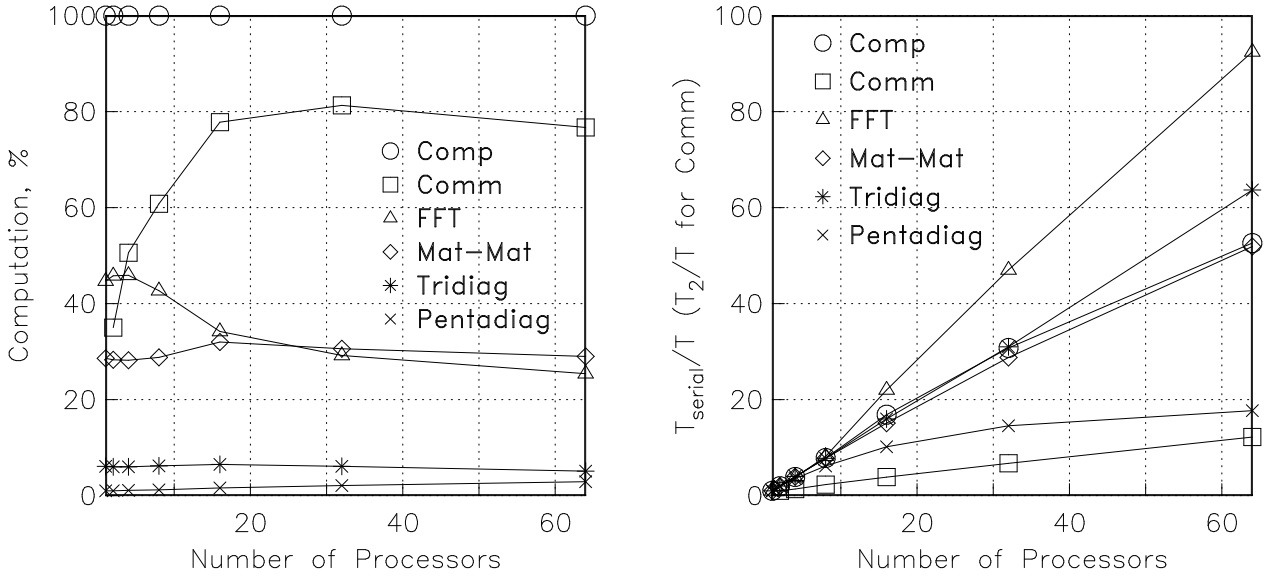


Figure 21. Computational-cost breakdown and speedup with number of processors for PS-DNS, where $n_x = 256$, $n_y = 41$, and $n_z = 64$.

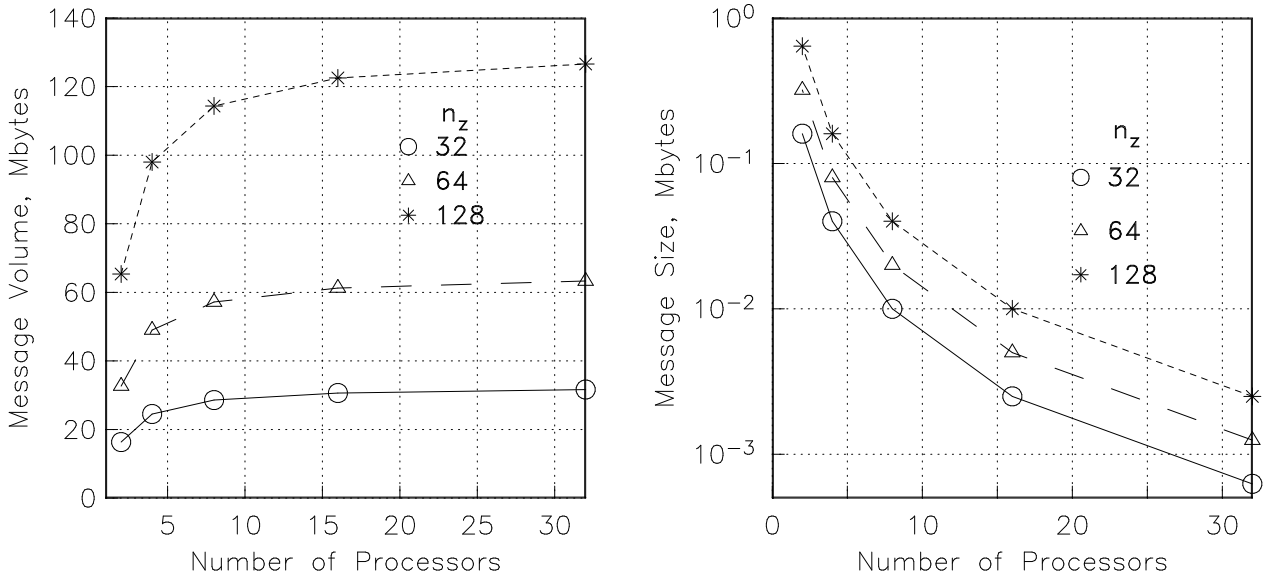


Figure 22. Message volume and size for one timestep for PSDNS, where $n_x = 64$ and $n_y = 41$.

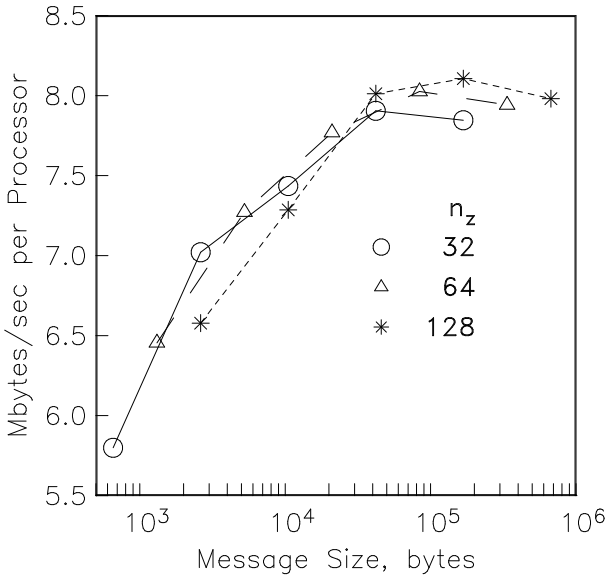


Figure 23. Communication bandwidth per processor as function of message size, where $n_x = 64$ and $n_y = 41$.

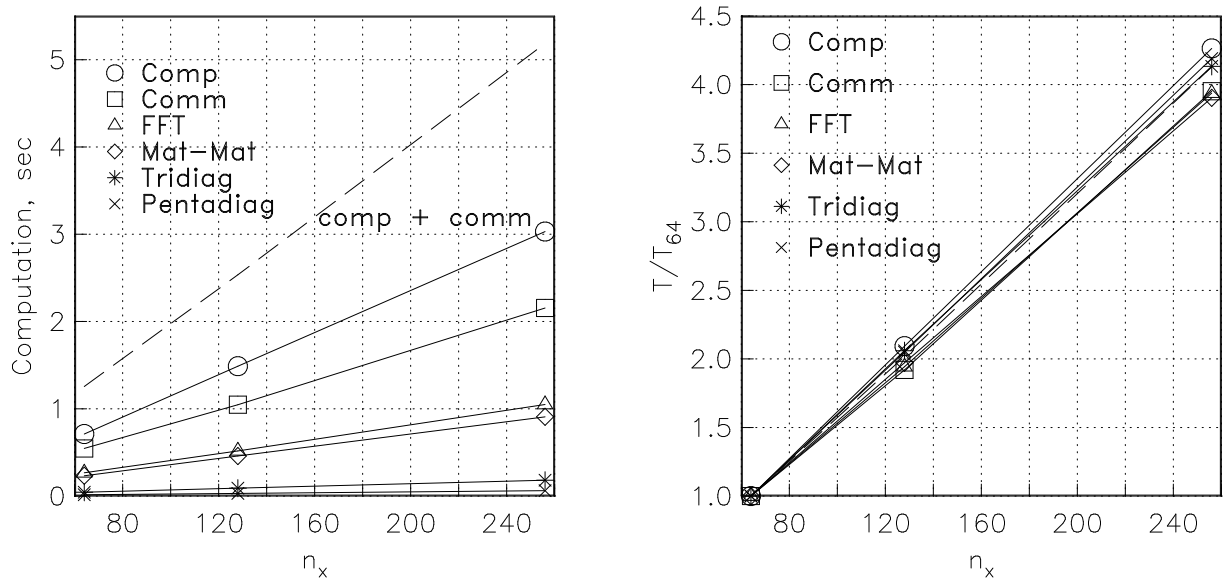


Figure 24. Computational-cost breakdown and slowdown with streamwise grid refinement for PSDNS, where $n_y = 41$, $n_z = 32$, and $n_p = 16$.

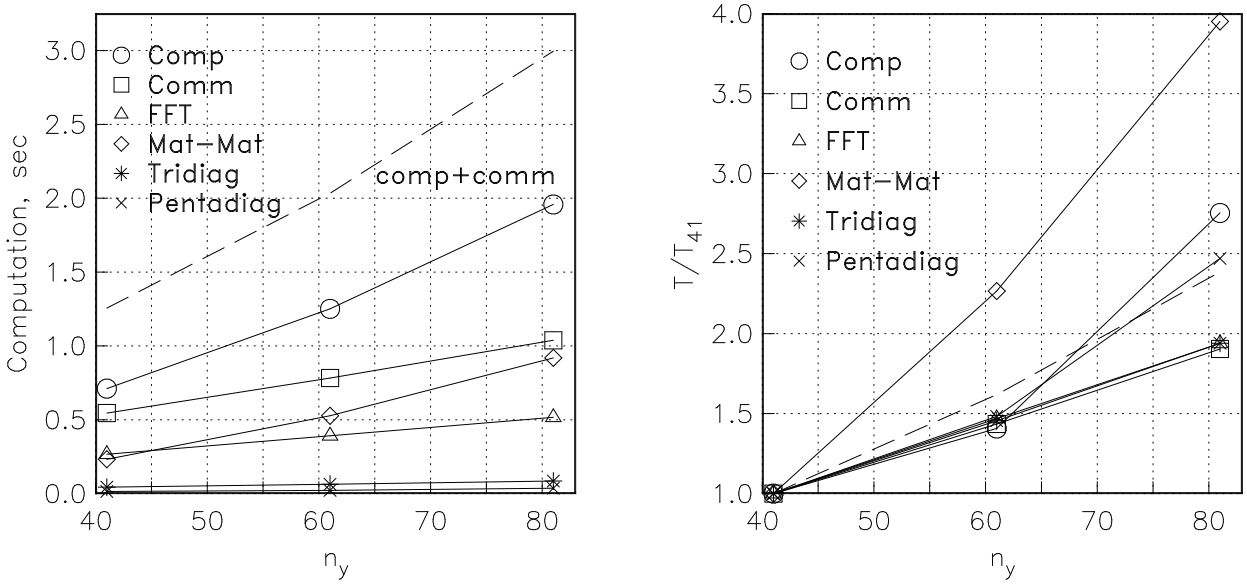


Figure 25. Computational-cost breakdown and slowdown with wall-normal grid refinement for PSDNS, where $n_x = 64$, $n_z = 32$, and $n_p = 16$.

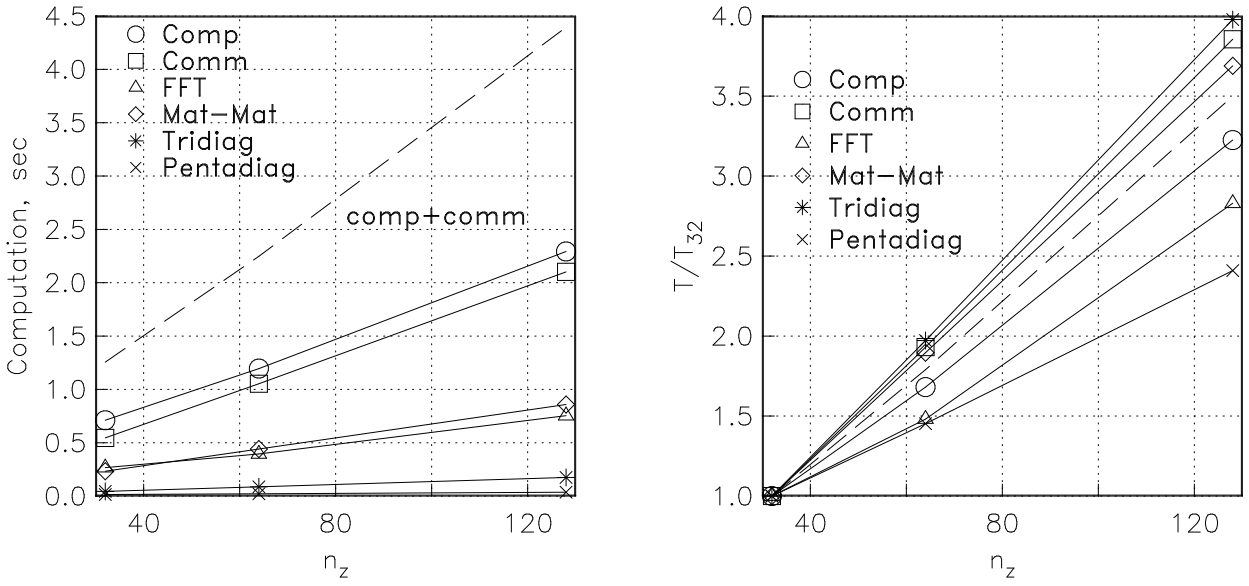


Figure 26. Computational-cost breakdown and slowdown with spanwise grid refinement for PSDNS, where $n_x = 64$, $n_y = 41$, and $n_p = 16$.

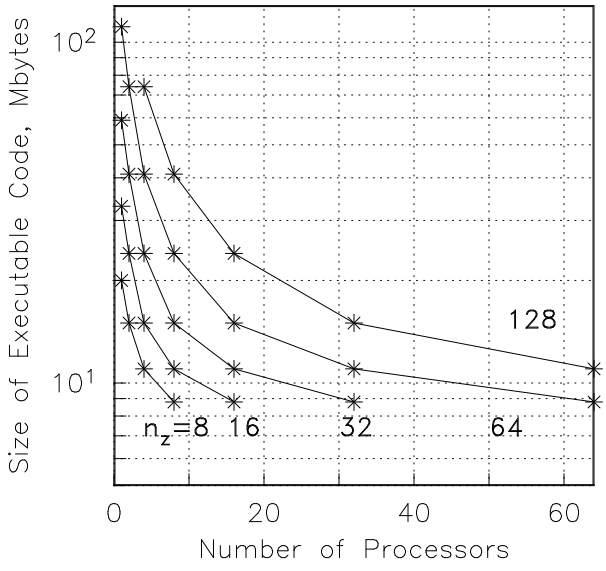


Figure 27. Memory size of executable with spanwise grid for PSDNS, where $n_x = 256$ and $n_y = 41$.

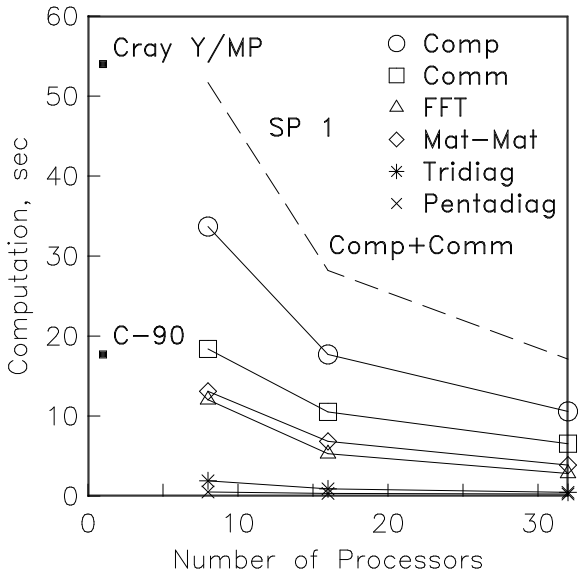


Figure 28. Computational and communication cost with number of processors for PSDNS on IBM SP1, where $n_x = 896$, $n_y = 61$, $n_z = 32$.

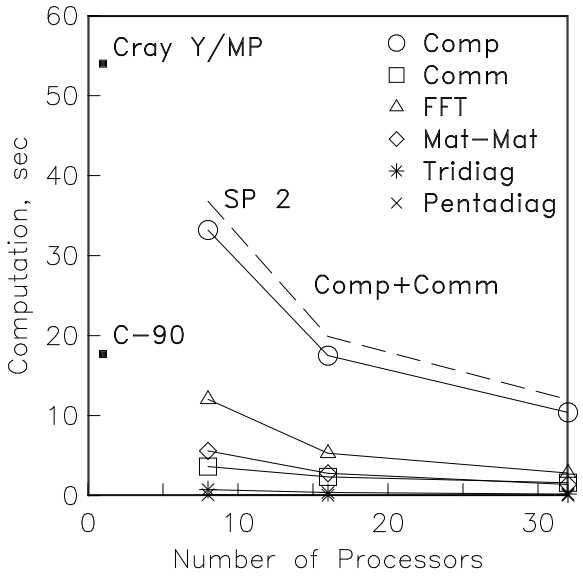


Figure 29. Computational and communication cost with number of processors for PSDNS on IBM SP2, where $n_x = 896$, $n_y = 61$, $n_z = 32$.