Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Computational Physics 210 (2005) 325–341
JOURNAL OF COMPUTATIONAL PHYSICS
www.elsevier.com/locate/jcp

A new parallel strategy for two-dimensional incompressible flow simulations using pseudo-spectral methods

Z. Yin [a], Li Yuan [a], Tao Tang [a,b,*]

[a] *LSEC, Institute of Computational Mathematics, Academy of Mathematics and System Sciences, Chinese Academy of Sciences, P.O. Box 2719, Beijing 100080, PR China*
[b] *Department of Mathematics, Hong Kong Baptist University, Kowloon Tong, Hong Kong, PR China*

Received 3 November 2004; received in revised form 9 April 2005; accepted 18 April 2005
Available online 21 June 2005

Abstract

A novel parallel technique for Fourier–Galerkin pseudo-spectral methods with applications to two-dimensional Navier–Stokes equations and inviscid Boussinesq approximation equations is presented. It takes advantage of the programming structure of the phase-shift de-aliased scheme for pseudo-spectral codes, and combines the task-distribution strategy [Z. Yin, H.J.H. Clercx, D.C. Montgomery, An easily implemented task-based parallel scheme for the Fourier pseudo-spectral solver applied to 2D Navier–Stokes turbulence, Comput. Fluid 33 (2004) 509] and parallelized Fast Fourier Transform scheme. The performances of the resulting MPI Fortran90 codes with the new procedure on SGI 3800 are reported. For fixed resolution of the same problem, the peak speed of the new scheme can be twice as fast as the old parallel methods. The parallelized codes are used to solve some challenging numerical problems governed by the Navier–Stokes equations and the Boussinesq equations. Two interesting physical problems, namely, the double-valued ω–ψ structure in two-dimensional decaying turbulence and the collapse of the bubble cap in the Boussinesq simulation, are solved by using the proposed parallel algorithms.

Keywords: Parallel computing; Pseudo-spectral methods; Task distribution; Navier–Stokes equations; Boussinesq equations

1. Introduction

The pseudo-spectral method has been very popular in the research of highly accurate numerical simulations since the pioneer work of Orszag and Patterson [2,3]. For smooth solutions, the conver-

* Corresponding author.
E-mail addresses: yinzh@lsec.cc.ac.cn (Z. Yin), lyuan@lsec.cc.ac.cn (L. Yuan), ttang@math.hkbu.edu.hk (T. Tang).

0021-9991/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2005.04.010

gence order of the spectral methods is higher than any algebraic power of mesh size. For a comparable error on the uniform mesh, a much finer mesh is required for finite difference or finite element methods. This is one of the reasons that the spectral method has been widely used in spite of the prosperous development of adaptive grid methods. Some comprehensive overviews of the various applications of spectral methods in fluid dynamics can be found in [4,5]. With the fast development of supercomputers, more and more efforts have been devoted to parallelizing spectral methods. For challenging simulation problems such as turbulence research, many new interesting phenomena are discovered using large scale parallel computations (512^3–4096^3) (e.g. see [6,7], and a review paper [8]). The parallel schemes based on spectral methods have also been intensively investigated over the last decade – mainly for three-dimensional (3D) problems [9–16]. The main computation time for spectral methods is concentrated in the part of the Fast Fourier Transform (FFT). Under certain situations (especially in 3D simulations), it is possible to get high parallel efficiency by simply using a good parallel FFT subroutine in the code. In the following, we will denote this parallel FFT procedure as PFFT.

For 3D or higher dimensional FFT, the transpose-split method can provide very high parallel efficiency. Most commonly used 3D parallel schemes employ this kind of parallel FFT [9–13]. Iovieno et al. [14] combined the transpose-split method and de-aliased procedure in pseudo-spectral codes together to yield high parallel efficiency. Based on the three time-evolution equations of the 3D Navier–Stokes (NS) equations, Basu adopted a parallel scheme which can only be used on a 3-processor computer [15]. Ling et al. [16] try to parallel the 3D code by combining the 3-CPU method and the PFFT scheme, and a comparison showed that the combined scheme is always slower than PFFT.

In contrast to the world-wide efforts in 3D parallelization, rather limited efforts have been devoted to parallelizing the two-dimensional simulations [17,1]. In the meantime, people have been using higher and higher resolutions to investigate 2D turbulence: the resolutions adopted in [18] are 4096^2, and 8192^2 in [19]. It is quite common to treat the 2D parallel spectral code as a simplified version of the 3D codes, which are very efficient only in 3D cases (PFFT). As a result, the ratio of the communication time to the computation time in those 2D parallel codes is relatively large, and the parallel efficiency is much lower compared with the corresponding 3D codes [1]. As can been seen later in this paper, for simulations with resolutions of lower than 512^2, the speedup of the PFFT scheme saturates when more than 32 processors are used on SGI3800. The usage of more processors in those cases will only cause a lower speed and waste of computer resources.

To minimize the relatively long communication time, Yin et al. [1] propose a parallel task-distribution scheme (PTD) in the 2D pseudo-spectral NS code. Although this scheme is very easy to implement with a good parallel efficiency, it has the limitation that the code can only use 2, 4, and 6 processors to do the calculation. In this paper, we will parallel the 2D spectral codes by combining the PTD and PFFT schemes. The new strategy overcomes the shortcomings of the former schemes and shows a significant improvement in parallel efficiency. In the following, we will denote this combined strategy as PTF scheme (parallelization through task distribution and FFT).

The paper is organized as follows. In Section 2, the PTF scheme is applied to solve the 2D NS equation; the benchmark of the parallel code on SGI 3800 is presented. We also show several long-term numerical simulations with high resolutions, which reveal some interesting physical phenomena of 2D decaying turbulence [20,21]. The adoption of the PTF scheme saves about 20–40% of cpu time in those longtime runs (the largest number of time steps in those simulations is up to 1.6×10^6). In Section 3, the new scheme is applied to solve the 2D inviscid Boussinesq equations. A challenging numerical problem, which is studied previously in [22–24], is investigated with three kinds of resolutions (1024^2, 2048^2, and 4096^2). Some extended studies of the new scheme are presented in Section 4, and a summary of this work is given finally.

2. Parallel 2D pseudo-spectral code for the NS equations

The study of the 2D turbulence distinguishes itself from the 3D turbulence due to its unique phenomena such as inverse energy cascade and self-organization. In the past few decades, a particular kind of statistical mechanics [25,26] has been widely adopted to study the 2D freely decaying turbulence (see [20] and references therein).

As a powerful tool, direct numerical simulations (DNS) provide some useful theory check and inspire the deeper thoughts of the statistical theory (e.g. see [20,21]). Those simulations normally need to last as long as 100–1000 eddy turnover times before final states of the 2D decaying turbulence are reached. This means that the total calculations of this kind of 2D DNS, despite having a smaller array, are more or less the same as those of some short-term 3D DNS. For example, 20 time steps of a 2D DNS with the resolution of 1024^2 will take roughly the same CPU time as one time step of a 3D DNS on a grid of 256^3 on the same computer. On the other hand, because the time-evolution loop is impossible to be parallelized, the parallel procedure within one time step becomes essential to improve the performance of a 2D DNS code.

On modern supercomputers, the total CPU time involved in the 2D DNS is not as enormous as that of the 3D DNS if the grid points are the same in each dimension. The peak performance of the parallel code, which is indicated by the shortest wall clock time regardless of the number of CPUs used, becomes more important in 2D DNS. As will be seen later in this paper, the significant improvement of the peak speed is a big advantage of the PTF scheme.

2.1. The governing equations and pseudo-spectral methods

The 2D incompressible NS equations in term of vorticity and stream function are written as:

$$\frac{\partial \omega}{\partial t} + \mathbf{u} \cdot \nabla \omega = \nu \Delta \omega, \tag{1}$$

$$\Delta \psi = -\omega, \tag{2}$$

where $\mathbf{u} = (u,v)$ is the velocity, ν the kinematic viscosity, vorticity $\boldsymbol{\omega} = (0,0,\omega) = \nabla \times \mathbf{u}$, ψ stream function. The stream function is related to velocity by $u = \partial\psi/\partial y$ and $v = -\partial\psi/\partial x$. In the conservative form, Eq. (1) becomes

$$\frac{\partial \omega}{\partial t} + \nabla \cdot (\omega \mathbf{u}) = \nu \Delta \omega. \tag{3}$$

We adopted ABCN scheme to carry out the time integration, which discretizes the non-linear term $\nabla \cdot (\omega \mathbf{u})$ with a 2nd order Adams–Bashforth scheme and the dissipation term $\nu\Delta\omega$ with the Crank–Nicolson scheme. The particular time-stepping used is independent of the parallelization. More accurate schemes, such as 4th order Runge–Kutta, could also be employed.

When Eqs. (2) and (3) are solved by pseudo-spectral methods, the Fourier coefficients of ω are evaluated at each time step. The non-linear term $\nabla \cdot (\omega \mathbf{u})$ is obtained by being transferred back and from the physical space with FFTs; in the meantime, the de-aliasing procedure has to be adopted to remove all aliasing errors. There are two kinds of de-aliasing techniques available: padding-truncation and phase-shifts [4]. The FFT we used requires the dimension size to be the power of 2, which is faster than those FFTs that have prime factors other than 2. The costs of two de-aliasing techniques are the same as those for our FFTs. Therefore, we use the phase-shifts in this paper, which reserves more high wave numbers information than that for padding-truncation.

Table 1 shows the ten FFTs needed to solve Eqs. (2) and (3) in each time step [27]. The expression with a hat (e.g. $\hat{u}$, $\widehat{(U\Omega)}$) is the spectral space value of the corresponding physical value without hat (u, $(U\Omega)$, etc.).

Table 1
Ten FFTs needed to be calculated in each time step of NS spectral solver

	A	B	C	D
1	$\hat{u} \to u$	$\hat{v} \to v$	$\hat{U} \to U$	$\hat{V} \to V$
2	$\hat{\omega} \to \omega$		$\hat{\Omega} \to \Omega$	
3	$u\omega \to \widehat{(u\omega)}$	$v\omega \to \widehat{(v\omega)}$	$U\Omega \to \widehat{(U\Omega)}$	$V\Omega \to \widehat{(V\Omega)}$

The capital letters denote the phase-shift values of the corresponding lower case variables. For example, in a simulation with M^2 resolution

$$\Omega_{\mathbf{j}} = \sum_k \hat{\omega}_k e^{i\mathbf{k}\cdot(\mathbf{x}_{\mathbf{j}}+\delta)},$$

where $\mathbf{j}$ indicates the grid point, k is the wave number, $\mathbf{x} = (x,y)$, and $\delta = (\pi/M,\pi/M)$. Each arrow in Table 1 indicated one 2D FFT, and the four FFTs in the third row should be calculated after the six FFTs in the first two rows.

It is worth mentioning that the pseudo-spectral method needs to evaluate twelve FFTs for the non-conservative form (Eqs. (1) and (2)),[1] since the two FFTs in the 2nd row of Table 1 are replaced with four FFTs to transfer $\widehat{(\partial\omega/\partial x)}$, $\widehat{(\partial\omega/\partial y)}$, and their phase-shift counterparts to physical space. (Of course, the FFTs in the 3rd row need to be changed correspondingly although no more FFT is required.) Hence, it is a natural choice to employ the conservative formulation when we use PFFT scheme to parallel the 2D NS code. In the following subsection, we will give a brief discussion for the PFFT scheme. And we will also introduce some symbols and analyzing tools that will be used throughout the paper.

2.2. A brief discussion for the PFFT scheme

The PFFT scheme calculates the ten parallelized FFTs one after another according to the certain sequence mentioned in the previous section. It is observed that the research on parallel FFT is a fast developed field (e.g. see [28–31], or a relatively complete review in [32]). It is difficult to rank the available FFTs, since there are so many different versions of FFTs, different parallel schemes for the FFTs, and different parallel computers to implement them. In this paper, we will not devote our efforts to parallel FFT (simply use the most popular transpose-split [10,33]). Instead, we will try to find other methods to improve the parallel efficiency. Our parallel scheme will be even faster if a faster parallel FFT is adopted.

The total time (T_{sum}) for each processor is the sum of the computation time (T_{comp}) and the communication time (T_{comm}), and T_{comm} consists of two parts – transmission time (T_{sendrec}) and latency time (T_{delay})

$$T_{\text{sum}} = T_{\text{comp}} + T_{\text{comm}} = T_{\text{comp}} + T_{\text{sendrec}} + T_{\text{delay}}. \tag{4}$$

In the following, we make M^2 represent the resolution of the simulation, p the total number of processors, t_{sendrec} the time to transmit a word between processors, t_{delay} the latency time for a message passing, and t_c the per-element computation time in a single processor times a factor (the factor is 5 in the case of full complex FFT, and 5/2 for real-complex FFT) [4].

In the case of PFFT, the 2D NS equations need to calculate ten FFTs, which takes the major part of the computations

[1] The non-linear term in Eq. (1) and Fig. 3 of Yin et al. [1] is should be in conservative form ($\nabla\cdot(\omega\mathbf{u})$), because ten FFTs instead of twelve FFTs were considered in [1].

$$T_{\text{comp}} = \frac{10 t_{\text{FFT}}}{p} = \frac{10}{p}(M^2 \log_2(M^2)\, t_c), \tag{5}$$

where t_{FFT} is the time required by one processor to compute one FFT. For each FFT, one processor needs to send $(p-1)$ blocks of data to other $(p-1)$ processors, and receives the same amount of data from all other processors. The size of each block is M^2/p^2, so:

$$T_{\text{sendrec}} = 2 \times \left((p-1) \times \frac{M^2}{p^2} \times t_{\text{sendrec}} \right) \times 10, \tag{6}$$

$$T_{\text{delay}} = 2 \times ((p-1) \times t_{\text{delay}}) \times 10. \tag{7}$$

Hence, according to our analytical model, the total time for one time step on one processor is:

$$T_{\text{sum}} = (20p - 20)\frac{M^2}{p^2} t_{\text{sendrec}} + (20p - 20) t_{\text{delay}} + \frac{10}{p}(M^2 \log_2(M^2) t_c). \tag{8}$$

In fact, the values of t_c is smaller for larger CPU numbers because of the cache effect, while t_{sendrec} and t_{delay} are almost constant for a given parallel computer. Here, we are only trying to give an estimated analysis and treat t_{sendrec}, t_{delay}, and t_c as constants; accurate timing work will be the speedup plot resulting from the wall clock time (see, e.g. Fig. 1; the similar approach is also adopted in [13]).

For many parallel systems, t_{delay} is much larger than t_{sendrec} (sometimes a factor of 1000 [34,35]). So T_{delay} is a non-trivial part of T_{sum}, especially when M^2 is small. According to Eq. (8), if we use more processors in the simulation, T_{delay} will become larger while the rest part ($T_{\text{sendrec}} + T_{\text{comp}}$) will be smaller. Eventually, T_{delay} will become the dominating part of T_{sum}, which affects the parallel efficiency of the PFFT scheme. In the case of the 2D DNS, this phenomenon is seen in relatively small CPU numbers, because $T_{\text{sendrec}} + T_{\text{comp}}$ is not very large. In the 3D DNS with high resolutions (512^3 or higher), T_{delay} will take a very small portion in T_{sum} except for very large p. This is the main difference between the 2D and 3D problems.

Fig. 1 shows the speedup of PFFT on different resolutions on SGI3800 ($t_{\text{delay}} \simeq 5.6 \times 10^{-6}$ s; $t_{\text{sendrec}} \simeq 2.5 \times 10^{-8}$ s). The speedup factor is the wall clock time of serial run divided by that of the parallel run in the same resolution. For the resolution of 128^2, the run with eight processors has the top speed. If the CPU number is larger than eight, the speed of the code drops down, and the parallel code with 32 or 64 processors is even slower than the serial code. For resolutions of 256^2 and 512^2, 16 CPUs give the

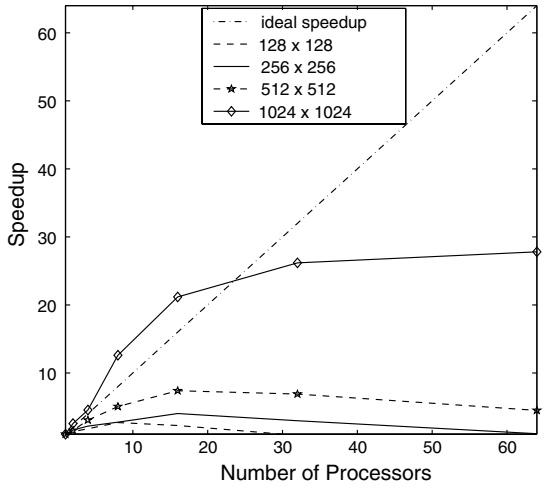

Fig. 1. The speedup of 2D Navier–Stokes code with PFFT scheme (or "1-n" scheme, see the discussion at the beginning of Section 2.3) on SGI3800.

best performance, while 32 and 64 CPUs will lead to worse performance than 16 CPUs. For 1024^2, the fastest run is the one with the maximum available CPUs. We can predict from the tendency of the curve (or Eq. (8)) that the speedup will also drop down for the 1024^2 run with 128 or more processors.

One may notice that the super-linear speedup for runs with 1024^2 resolution: in the case of 16 CPUs, a speedup of 21 is observed. The super-linear speedup is due to the so-called *cache effect*, which makes t_c smaller in parallel computing. The platform we choose here is SGI Origin 3800 with a relatively large cache size (8 MB). The behavior of the cache memory is very hard to predict, because modern computer architectures have very complex internal organizations [36,37]. Therefore, we do not take the cache effect into the consideration in the analyzing model.

To sum up, T_{delay} takes a large portion in the total communication time (T_{comm}) and seriously reduces the parallel efficiency for larger CPU numbers in the PFFT scheme. The issue of minimizing T_{delay} is the key to enhance the parallel efficiency.

2.3. A new parallel strategy – PTF

As mentioned earlier, there are ten FFTs needed to be evaluated at each time step. In PTF scheme, they can be divided into two, four, and six groups corresponding to the 2, 4, and 6 CPUs scheme in PTD [1]. In the following, we will call these three PTF schemes as "2-n", "4-n", and "6-n" scheme, respectively. (We also denote PFFT scheme as "1-n" scheme to unify notations). There are six FFTs in the 2-n scheme for each group, while three FFTs in the 4-n scheme. Note that there are twelve FFTs in the 2-n and 4-n schemes at each time step (there is only ten FFTs in serial and PFFT codes) because the two FFTs in the 2nd row of Table 1 are calculated twice to reduce the total communication time. The 6-n scheme will not be discussed in the rest of the paper, because we want to compare the parallel efficiency of the PTF scheme with the PFFT (1-n) scheme, and the number of CPUs involved should be powers of 2.

For any type of PTF scheme, one group will be called "master group", on which the time integration and data input and output (I/O) are carried out, and the other groups will be called "slave groups". When calculating FFT, data are exchanged only within each group. When it is necessary to transfer information between groups, the first node in master group will and only will communicate with the first nodes in the slave groups; likewise, the second nodes in different groups will communicate with each other, etc.

At the beginning of time loop, each node in the master group needs to send one block data (the size is $\frac{M^2}{p/2}$ for 2-n scheme, and $\frac{M^2}{p/4}$ for 4-n scheme) to the corresponding node in the slave group, and to receive the same amount of data at the end of time loop. Following the same procedure in the previous section, we can get the estimated cpu time of the PTF schemes:

$$\text{2-n}: \quad T_{\text{sum}} = (28p-48)\frac{M^2}{p^2}t_{\text{sendrec}} + (6p-10)t_{\text{delay}} + \frac{12}{p}(M^2\log_2(M^2)t_c), \tag{9}$$

$$\text{4-n}: \quad T_{\text{sum}} = (40p-96)\frac{M^2}{p^2}t_{\text{sendrec}} + \left(\frac{3}{2}p-2\right)t_{\text{delay}} + \frac{12}{p}(M^2\log_2(M^2)t_c). \tag{10}$$

As can be seen from Eqs. (8)–(10), T_{comp} in the 1-n, 2-n, and 4-n schemes are roughly the same (20% difference at most) for fixed M and p; T_{delay} in the 1-n scheme is about three times as large as that in the 2-n scheme, and about 13 times as large as that in 4-n scheme. In the meanwhile, T_{sendrec} in the 2-n or 4-n scheme is increased by relatively small factor (no more than 2) compared with 1-n scheme. Hence, T_{delay} in the 1-n scheme will occupy the largest partition of T_{sum} among the 1-n, 2-n, and 4-n schemes. However, even in the 2-n and 4-n schemes, T_{delay} still increases and the remaining parts of T_{sum} decrease for larger p; the bottleneck effect in parallelization still exits.

Fig. 2(a) shows the speedup plot for the 2-n scheme. For the resolution of 128^2, the peak speed appears at 16 CPUs run (compared with 8 CPUs run for the 1-n scheme). For the resolution of 256^2, the peak speed

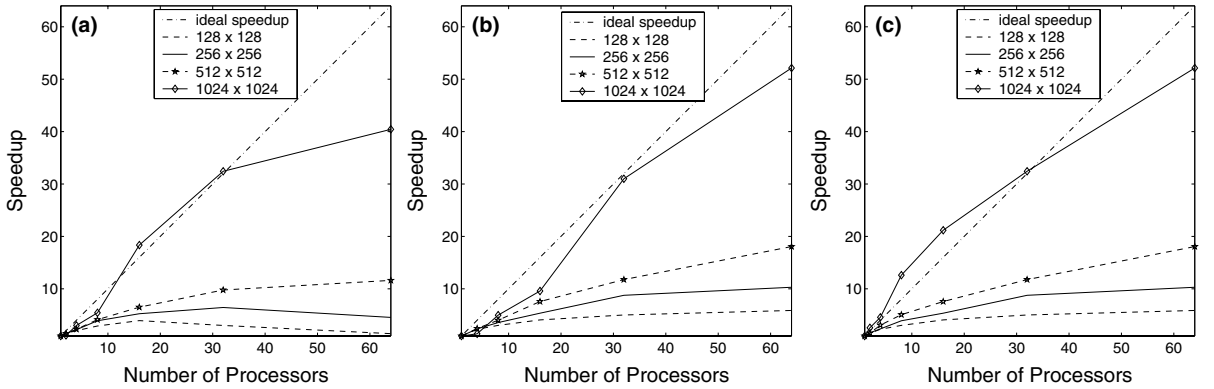

Fig. 2. The speedup of 2D Navier–Stokes code on SGI3800 with "2-n" scheme (a), "4-n" scheme (b), and "ideal" scheme. The values used in the "ideal" scheme are the top speedup among 1-n, 2-n, and 4-n schemes for the same resolutions and the same numbers of processors.

appears at 32 CPUs run (compared with 16 CPUs run for 1-n scheme). For 512^2 and 1024^2 in the 2-n scheme, and all the resolutions in the 4-n scheme (Fig. 2(b)), 64 CPUs run is the fastest. Because two extra FFTs are introduced and T_{sendrec} is slightly larger in 2-n and 4-n schemes, some runs in Fig. 2 are slower than the corresponding 1-n ones for certain resolutions and p. For example, there is no super-linear speedup observed for the 1024^2 curve in Fig. 2(b), although the 4-n scheme is twice as fast as the 1-n scheme when 64 processors are used.

In practice, the fastest scheme for fixed p and M^2 is always what we want to use to do longtime simulation. Fig. 2(c) is a combined figure which consists of the best performance point in Figs. 1 and 2(a) and (b). Table 2 indicates the corresponding schemes adopted to draw this "best speedup" plot. It should be emphasized that most points in the 1024^2 curve show super-linear speedup. The points when $p \leqslant 16$ come from the 1-n scheme, while the 2-n and 4-n schemes are faster for $p \geqslant 32$. For lower resolutions, the 1-n scheme works the best for small p, and the 2-n or 4-n scheme dominates the points on the corresponding curve in Fig. 2(c) gradually for larger p. On the first row of Table 2, which corresponds to the resolution of 128^2, the 4-n scheme dominates for $p \geqslant 4$.

The situations of $p = 2$ in the 2-n scheme and $p = 4$ in the 4-n scheme correspond to the PTD scheme. Although the benchmark results are slightly different from [1] due to different computers used, the conclusion is the same: PTD is an easily implemented and efficient parallel scheme for the 2D DNS for relatively small resolutions. The easy implementary property of PTD is inherited by PTF schemes: once the PFFT code is ready, it is very easy to change it into PTF codes. Hence, PTF is an attractive strategy, especially in 2D simulations.

Table 2
The corresponding schemes for the values in Fig. 2(c)

	1 CPU	2 CPUs	4 CPUs	8 CPUs	16 CPUs	32 CPUs	64 CPUs
128^2	1-n	2-n	4-n	4-n	4-n	4-n	4-n
256^2	1-n	1-n	4-n	2-n	4-n	4-n	4-n
512^2	1-n	1-n	1-n	1-n	4-n	4-n	4-n
1024^2	1-n	1-n	1-n	1-n	1-n	2-n	4-n

2.4. Numerical results for 2D decaying turbulence

In this subsection, we will use our PTF codes to investigate an interesting phenomenon in 2D decaying turbulence – the multi-valued ω–ψ structure.

People used to treat the functional relation $\omega = f(\psi)$ as an indication of the final state of 2D turbulence, but the simulation shown in Figs. 18 and 19 of Yin et al. [20] gives a counter example: the double-valued ω–ψ structure (see Yin [21] for a detailed discussion about this kind of structure). We will seek some other simulations to validate the generality of multi-valued structures, and there are mainly two choices to do this: (1) make changes in the initial conditions; (2) test different values of ν for certain kind of initial condition.

The parallel code we used before (PTD scheme [1]) has a very limited speedup because the maximum number of CPUs used is limited to six. Most simulations were carried out by changing the initial condition for relatively small ν with the resolution of 512^2 [21], which normally lasted from several days to a few weeks. For 1024^2 runs, the calculations for one time step are four times as large as those for 512^2, and the time step has to be smaller due to the CFL condition. A typical 2D decaying turbulence lasts from two months to half a year if we only use PTD schemes. With the PTF codes, it is now possible to carry out 1024^2 simulations to find the new double-valued ω–ψ structure.

We carried out two simulations starting from four equal-sized vortices patches, which are asymmetrically placed in a double periodic box (Fig. 3(a)). The first run adopted relatively low Reynolds number ($1/\nu = 4000$) with the resolution of 512^2. The time step is 0.0005. We used 32 CPUs in total (this is the maximum nodes available for longtime simulations in LSEC). According to Table 2, the 4-n scheme is the fastest approach under this situation. The simulation lasted for 17 h, and reached the quasi-steady state at $t \approx 200$ with a simple function relation of ω–ψ (Fig. 7(b) in [21]). The same run lasts for 29 h if we use the PFFT scheme (32 CPUs), and 82 h for the PTD scheme (4 CPUs).

The second run adopted a large Reynolds number ($1/\nu = 40{,}000$) with the resolution of 1024^2. We used the 2-n scheme code on 32 processors, and the quasi-steady state was reached at $t \approx 800$ after 16 days' calculation. In the late state of this simulation, the orientation of the flow pattern (Fig. 3(b)) is totally different from that obtained with the 512^2 run. Moreover, the double-valued ω–ψ structure is recovered (Fig. 3(c)), which fits the definition of the final state of the 2D turbulence given in [21]. Hence, it seems that further

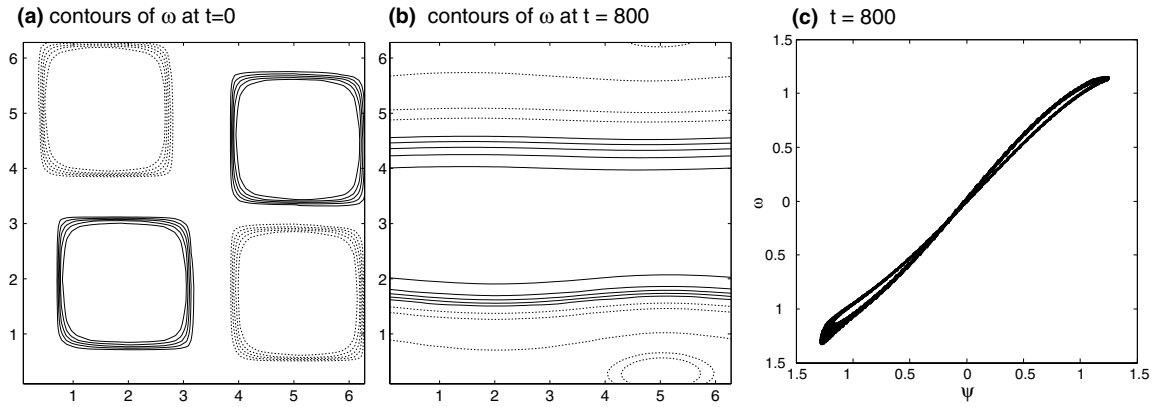

Fig. 3. Constant vorticity contours of the initial (a) and final (b) state in a Navier–Stokes simulation. Dashed contours represent positive vorticity and drawn contours represent negative vorticity. (c) is the ω–ψ scatter plot in the final state (with a double-valued structure).

computations do not lead to any new phenomenon. The same run lasts for 20 days if we use the PFFT scheme (32 nodes), and about one year for the PTD scheme (4 nodes).

It is interesting to see the appearance of multi-valued ω–ψ structures in high Reynolds number (40,000), while relatively low Reynolds number (4000) results in traditional functional relation. Although the physical mechanism of the multi-valued structure is still unclear, our simulations reveal some limitations of the statistical mechanics [21], and it is worthwhile to carry on further investigations in this direction. Of course, the PTF scheme may play an important role in these longtime computations because of its high parallel efficiency. With maximum 32 processors available for longtime simulation, the peak speed of PFT is only about 70% faster than that of PFFT for the resolution of 512^2, and 24% for 1024^2. However, for runs like what are shown in Figs. 3, 24% shortage still means 4 days' calculation on 32 nodes, which is definitely non-trivial.

3. Application of PTF scheme to 2D inviscid Boussinesq equations

It is very interesting to understand whether a finite time blow-up of the vorticity and temperature gradient can occur from a smooth initial condition in 2D inviscid Boussinesq convection. This has been studied recently by several groups, ending up with different conclusions: Pumir and Siggia [22] used the TVD scheme with the adaptive meshes (the maximum grid size is 256^2), which observed the blow-up process; E and Shu [23] did not obtain the blow-up phenomena by using the ENO scheme on a 512^2 grid and spectral methods on a 1500^2 grid; Ceniceros and Hou [24] also obtained blow-up free solutions with the adaptive mesh computation on a 512^2 grid. It is worth mentioning that the maximum mesh compression ratio in [24] is 8.83, which gives an effective resolution of 4600^2 on uniform mesh. Although these groups yielded different conclusions, they all tried to use the highest possible resolutions to investigate the problem. Hence, parallel computing is a natural choice.

The non-dimensionized 2D inviscid Boussinesq convection equations can be written in ω–ψ formulation:

$$\rho_t + \mathbf{u}\cdot\nabla\rho = 0, \tag{11}$$

$$\omega_t + \mathbf{u}\cdot\nabla\omega = -\rho_x, \tag{12}$$

$$\Delta\psi = -\omega. \tag{13}$$

Here, ρ is the density (we take the same denotation as [23] for an easy comparison) and the gravitational constant is normalized to be $\mathbf{g} = (0,-1)$. Again, ABCN scheme is adopted to carry out the time integration. Below we will discuss how to implement our parallel strategy for Eqs. (11)–(13). Numerical results on the finite time blow-up will be reported.

3.1. The application of the new strategy to Boussinesq equations

As shown in Table 3, there are 20 FFTs involved when the Fourier–Galerkin spectral methods are used to solve Eqs. (11)–(13). These FFTs can be divided into four independent groups, which are indicated by the different columns in Table 3. There is no communication within the different columns until all FFTs are finished. In each column, the 1st and the 2nd FFT need to be evaluated before the 4th one, while the 5th FFT should be calculated after the 1st and 3rd one are finished.

When the PTF scheme is used to parallelize the code, there are five options to divide the groups without too much extra data communication:

(1) `l-N` scheme – one group; each FFT is computed by all the CPUs involved, i.e., PFFT scheme. (Here, we use "N" instead of "n" to avoid conflicts with the parallel schemes for the NS equations).

Table 3
Twenty FFTs needed to be calculated in each time step of Boussinesq spectral solver

	A	B	C	D
1	$\hat{u} \to u$	$\hat{U} \to U$	$\hat{v} \to v$	$\hat{V} \to V$
2	$\hat{\omega}_x \to \omega_x$	$\hat{\Omega}_x \to \Omega_x$	$\hat{\omega}_y \to \omega_y$	$\hat{\Omega}_y \to \Omega_y$
3	$\hat{\rho}_x \to \rho_x$	$\hat{P}_x \to P_x$	$\hat{\rho}_y \to \rho_y$	$\hat{P}_y \to P_y$
4	$u\omega_x \to \widehat{(u\omega_x)}$	$U\Omega_x \to \widehat{(U\Omega_x)}$	$v\omega_y \to \widehat{(v\omega_y)}$	$V\Omega_y \to \widehat{(V\Omega_y)}$
5	$u\rho_x \to \widehat{(u\rho_x)}$	$UP_x \to \widehat{(UP_x)}$	$v\rho_y \to \widehat{(v\rho_y)}$	$VP_y \to \widehat{(VP_y)}$

(2) `2-N` scheme – two groups; column A & B in one group, and column C & D in the other group.
(3) `4-N` scheme – four groups, which belongs to different columns in Table 3, respectively.
(4) `8-N` scheme – eight groups (see column A–H of Table 4). Note that there are four extra FFTs introduced to save the communication time: the first row of FFTs in Table 3 are calculated twice in the first row of Table 4.
(5) `12-N` scheme – twelve groups; the first three rows of FFTs in Table 3 are calculated simultaneously in twelve groups. The results from the four groups calculating the first row of FFTs are transferred to the eight groups computing the 2nd and 3rd row, respectively, and the rest eight FFTs in the 4th and 5th rows are performed in those eight groups. (Like the 6-n scheme in the NS solver, the 12-N scheme will not be discussed here because the number of processors involved is not of the power of 2.)

Similar to Section 2, we can analyze the total computation time for different schemes:

$$\texttt{1-N}: \quad T_{\text{sum}} = (40p-40)\frac{M^2}{p^2}t_{\text{sendrec}} + (40p-40)t_{\text{delay}} + \frac{20}{p}(M^2\log_2(M^2)t_c), \tag{14}$$

$$\texttt{2-N}: \quad T_{\text{sum}} = (48p-80)\frac{M^2}{p^2}t_{\text{sendrec}} + (10p-16)t_{\text{delay}} + \frac{20}{p}(M^2\log_2(M^2)t_c), \tag{15}$$

$$\texttt{4-N}: \quad T_{\text{sum}} = (72p-160)\frac{M^2}{p^2}t_{\text{sendrec}} + \left(\frac{5}{2}p-2\right)t_{\text{delay}} + \frac{20}{p}(M^2\log_2(M^2)t_c), \tag{16}$$

$$\texttt{8-N}: \quad T_{\text{sum}} = (144p-384)\frac{M^2}{p^2}t_{\text{sendrec}} + \left(\frac{3}{4}p+6\right)t_{\text{delay}} + \frac{24}{p}(M^2\log_2(M^2)t_c). \tag{17}$$

For all the PTF discussed above (Eqs. (14)–(17)), it is clear that T_{delay} will take larger portion of T_{sum} when p becomes larger in those schemes. For fixed p, T_{delay} will take smaller portion of T_{sum} when more groups

Table 4
In 8-N scheme, the total twenty-four FFTs in each time step of Boussinesq spectral solver are divided into eight groups (A-H). There are four FFTs introduced in addition to the twenty FFTs in Table 3

	A	B	C	D
1	$\hat{u} \to u$	$\hat{U} \to U$	$\hat{v} \to v$	$\hat{V} \to V$
2	$\hat{\omega}_x \to \omega_x$	$\hat{\Omega}_x \to \Omega_x$	$\hat{\omega}_y \to \omega_y$	$\hat{\Omega}_y \to \Omega_y$
3	$u\omega_x \to \widehat{(u\omega_x)}$	$U\Omega_x \to \widehat{(U\Omega_x)}$	$v\omega_y \to \widehat{(v\omega_y)}$	$V\Omega_y \to \widehat{(V\Omega_y)}$
	E	F	G	H
1	$\hat{u} \to u$	$\hat{U} \to U$	$\hat{v} \to v$	$\hat{V} \to V$
2	$\hat{\rho}_x \to \rho_x$	$\hat{P}_x \to P_x$	$\hat{\rho}_y \to \rho_y$	$\hat{P}_y \to P_y$
3	$u\rho_x \to \widehat{(u\rho_x)}$	$UP_x \to \widehat{(UP_x)}$	$v\rho_y \to \widehat{(v\rho_y)}$	$VP_y \to \widehat{(VP_y)}$

are adopted in the PTF schemes (e.g. 4-N or 8-N scheme). Hence, the 4-N and 8-N schemes have some advantage over the 1-N and 2-N schemes when p is large. Some further discussions for Eqs. (14)–(17) will be continued in the following subsection together with the speedup plots (Fig. 4).

Like the 2D NS equations (Eqs. (2) and (3)), Eqs. (11) and (12) can also be written in the conservative form:

$$\rho_t + \nabla \cdot (\rho \mathbf{u}) = 0, \tag{18}$$

$$(\rho\omega)_t + \nabla \cdot (\rho\omega\mathbf{u}) = -\tfrac{1}{2}(\rho^2)_x. \tag{19}$$

When the Fourier–Galerkin spectral methods are used to solve the equations above, Eq. (18) presents some problems because the time evolution step is carried out in spectral space: to get the value of $\hat{\omega}$ in the next time step, $\widehat{(\rho\omega)}$ has to be transferred back to physical space so that the new value can be obtained by using the relation $\omega = (\rho\omega)/\rho$. Furthermore, there is an extra non-linear term $-\frac{1}{2}(\rho^2)_x$ in Eq. (19) which requires one extra FFT. The total number of FFTs in conservative form is the same as non-conservative form. Hence, unlike what we did for 2D NS equations in Section 2, the non-conservative equations are solved here.

3.2. Benchmarks and comparisons

In this section, we will show the speedup plots of the Boussinesq equations on SGI3800. Unlike what we did for the NS equations, we added a new resolution (64^2) to show the effectiveness of the new parallel scheme in the low resolution. The maximum processors used here are 32, which makes the resulting speedup plots (Figs. 4) more concise.

Fig. 4(a) shows the speedup plot for PFFT (or 1-N) scheme. The top speedup is obtained on 4 processors for the 64^2 grid, while 8 and 16 processors reach the top speed for the resolutions of 128^2 and 256^2, respectively. The speedup curves of 512^2 and 1024^2 indicate reasonably good parallel efficiency of PFFT scheme. The 512^2 runs reveal super linear speedup for 2, 4, 8, and 16 nodes, which is due to the cache effect. The 1024^2 runs only have super linear speedup in the cases of 8 and 16 nodes. The speedups of 32-node run for these two high resolutions are lower than 32 because T_{delay} begins to dominate the total computation time. We did not show the speedup plots for resolutions higher than 1024^2 because of the limit of the maximum local memory. However, since solving the equations with the highest possible resolution is the main task of our research, we will discuss the parallel efficiency on those high resolutions (2048^2 and 4096^2) in other ways later in the following section.

Fig. 4(b) shows the "best speedup" plot of PTF schemes. All the resolutions have their top speed in the 32 CPUs case. Moreover, almost all the points on the curves show super linear speedup for higher resolutions (512^2 and 1024^2). The peak speeds shown in Fig. 4(b) are increased by a factor of 27% (1024^2 resolution) to 171% (64^2 resolution) compared to the PFFT scheme. The lower resolutions get a higher factor because of the limitation of the total CPUs available. We can predict that the factor of 1024^2 resolution will be larger than 27% if the maximum number of CPUs used is larger than 32.

Table 5 provides the corresponding schemes to the points in Fig. 4(b). PFFT (or, 1-N) scheme never shows the best performance for resolutions of 64^2 and 128^2 for $p \geqslant 2$ (For $p = 1$, 1-N is the only choice, which is not necessary for comparison). For resolutions higher than 256^2, the 1-N scheme reaches the peak speed more frequently, especially for runs with fewer processors. As indicated in Table 5, the PTF schemes with more groups (e.g. 4-N or 8-N scheme) have better speedup for lower resolutions and larger number of CPUs, while the schemes divided into fewer groups (e.g. 1-N or 2-N scheme) work best for higher resolutions and relatively smaller number of CPUs.

In the real programming efforts, we have found that it is more convenient to fix the size of each group (instead of fixing p) because the size of main calculating arrays can be determined beforehand. The code will

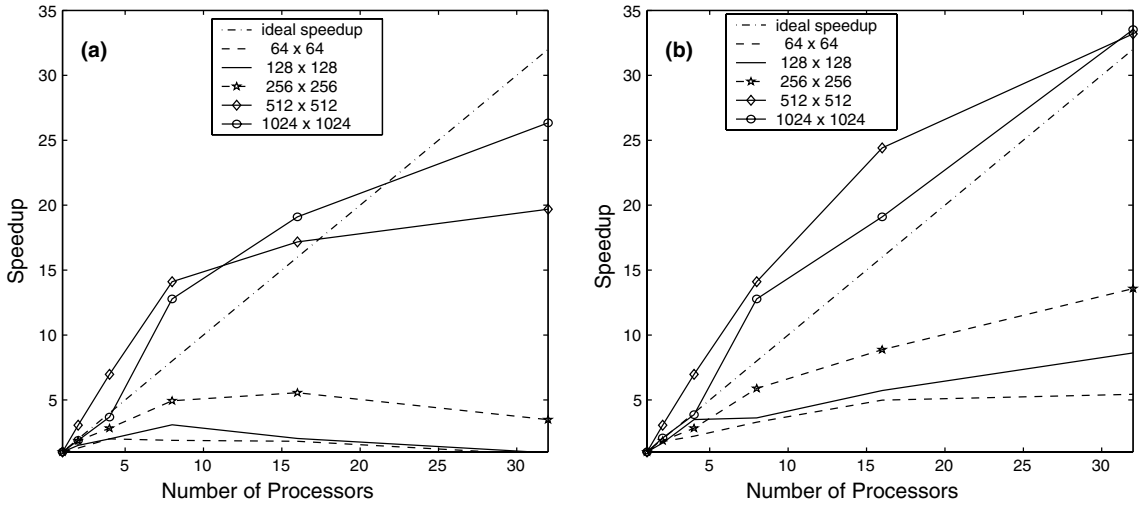

Fig. 4. The performance on SGI3800: (a) the speedup of 2D Boussinesq code with PFFT scheme (or "1-N" scheme); (b) the speedup of 2D Boussinesq code with "ideal" scheme. The values used on (b) are the top speedup among 1-N, 2-N, 4-N, and 8-N schemes for the same resolutions and the same numbers of processors.

Table 5
The corresponding schemes for the values in Fig. 4(b)

	1 CPU	2 CPUs	4 CPUs	8 CPUs	16 CPUs	32 CPUs
64^2	1-N	2-N	2-N	8-N	8-N	8-N
128^2	1-N	2-N	2-N	2-N	8-N	8-N
256^2	1-N	1-N	1-N	2-N	2-N	8-N
512^2	1-N	1-N	1-N	1-N	2-N	4-N
1024^2	1-N	2-N	2-N	1-N	1-N	2-N

get the number of groups (1, 2, 4, or 8) at the initial stage of real runs, and distribute the 20 FFTs into different groups. Thus, the programming effort concerning the PFT scheme is trivial if the PFFT code is already available.

Attention should be drawn to the performance of the 2-N scheme, which occupies eight of the total 30 places in Table 5. In the 1024^2 runs, the 2-N scheme is faster than the 1-N scheme on 2 and 4 nodes' simulations, which is counter to the conclusion we made in the above paragraph. Although most of the weird speedup behaviors in parallel computing can be attributed to cache effect, this one presents some difficulties because the more CPUs are used to calculate one FFT, the smaller t_c will be (due to *cache effect*). To explain this, we need to calculate the exact values of T_{comm} for the 1-N and 2-N schemes in our analytical models (Eqs. (14) and (15)):

$$\text{For the 1-N scheme,} \quad T_{\text{comm}} = \begin{cases} 10M^2 \times t_{\text{sendrec}} + 40 \times t_{\text{delay}} & \text{if } p = 2, \\ 7.5M^2 \times t_{\text{sendrec}} + 120 \times t_{\text{delay}} & \text{if } p = 4. \end{cases}$$

$$\text{For the 2-N scheme,} \quad T_{\text{comm}} = \begin{cases} 4M^2 \times t_{\text{sendrev}} + 4 \times t_{\text{delay}} & \text{if } p = 2, \\ 7M^2 \times t_{\text{sendrev}} + 24 \times t_{\text{delay}} & \text{if } p = 4. \end{cases}$$

It is clear that all parts of T_{comm} in the 2-N scheme are smaller than the 1-N scheme in the case of $p \leqslant 4$, although T_{sendrec} of the 2-N scheme is larger than that of the 1-N scheme for $p \geqslant 8$. Again, we assume that

t_c remains roughly the same for different schemes. The complete explanation has to take "cache effect" into consideration.

3.3. Numerical results for the Boussinesq convection

In this section, we will show some numerical simulations calculated by our parallel codes. To avoid the instability of the pseudo-spectral code [38], we adopted the same filter technique used in [23]. Also, we took the similar initial condition in [23] for easy comparison with former results:

$$\omega(x,y,0) = 0, \tag{20}$$

$$\rho(x,y,0) = 50\rho_1(x,y)\rho_2(x,y)[1-\rho_1(x,y)], \tag{21}$$

where

$$\rho_1(x,y) = \begin{cases} \exp\left(1 - \frac{\pi^2}{\pi^2 - y^2 - (x-\pi)^2}\right) & \text{if } y^2 + (x-\pi)^2 < \pi^2, \\ 0, & \text{otherwise,} \end{cases}$$

$$\rho_2(x,y) = \begin{cases} \exp\left(1 - \frac{(1.95\pi)^2}{(1.95\pi)^2 - (y-2\pi)^2}\right) & \text{if } |y-2\pi| < 1.95\pi, \\ 0, & \text{otherwise.} \end{cases}$$

The initial cap-like contour of ρ will develop into a rising bubble during the evolution, with the edge of the cap rolling up. At $t \simeq 3.16$, the density and vorticity contours develop into the shape of "two eyes." Fig. 5(a) and (b) shows the results with the resolution of 1024^2. For spectral methods with different resolutions (e.g. 1500^2 run of [23]), some good agreements have been observed. Arguments arise when the computations are carried on. At $t \simeq 3.7$, the smooth edge of the rolling eyes becomes unstable (see Fig. 5(c) and (d)). Whether this phenomenon is physically real or not is still an open question. It was argued that this collapse is due to the numerical effect [23,24].

As discussed above, the most straightforward way to investigate this problem is to increase the resolution of simulations. Therefore, we also adopted grids of 2048^2 and 4096^2 in the simulations. For the 1024^2 run, the time step used is 0.0008, which is determined by the CFL condition. It took the parallel 2-N scheme 20 h to reach $t = 3.7$ with 32 processors. It takes 25 h for PFFT schemes, and 28 days for the serial code. For the 2048^2 run, the time step used is 0.0004. The speed of the 1-N scheme and the 2-N scheme with 32 nodes are very close (a difference of 2%). We use the 1-N scheme because it is slightly faster. The code lasted for 13 days before it reached $t = 3.7$. For serial code, it may take one year provided the memory of the computer is large enough. For the 4096^2 run, the time step used is 0.0002. The 1-N scheme is the only choice because other schemes require larger computer memory: in principle, the memory sizes of the 2-N, 4-N, and 8-N schemes are 2, 4, and 8 times as large as that of the 1-N scheme. For this run, it is even a burdensome job for the parallel computation with 32 processors, which may last for four months in total. We actually use the interpolated data of the 2048^2 run at $t = 3.4$ as the initial condition. It is believed that the solution at $t = 3.4$ is smooth. The code lasted for three weeks before it reached $t = 3.7$.

Fig. 5(e) and (f) shows contour plots of the 4096^2 run at $t = 3.7$, on which the collapse is observed in the smooth edge. It is noticed that more rolling vortices are observed on the edge of the "eyes" in Fig. 5(e) than those in Fig. 5(c). There are some trivial differences if we make a detailed comparison within these three runs, but Fig. 5(e) and (f) clearly indicates that the collapsing process is not a non-physical phenomenon from the numerical artifacts. If we started the 4096^2 run from $t = 0$ using Eqs. (20) and (21) to get the initial data, the contour plots might be slightly different from what we obtained in Fig. 5(e) and (f). The main task here is to determine the possibility of the collapse phenomenon, the potential small difference in the flow pattern will not hurt the general conclusion. Some further details of the physical phenomenon will be discussed in another work. Here, we will try to focus our topic on the parallel solver.

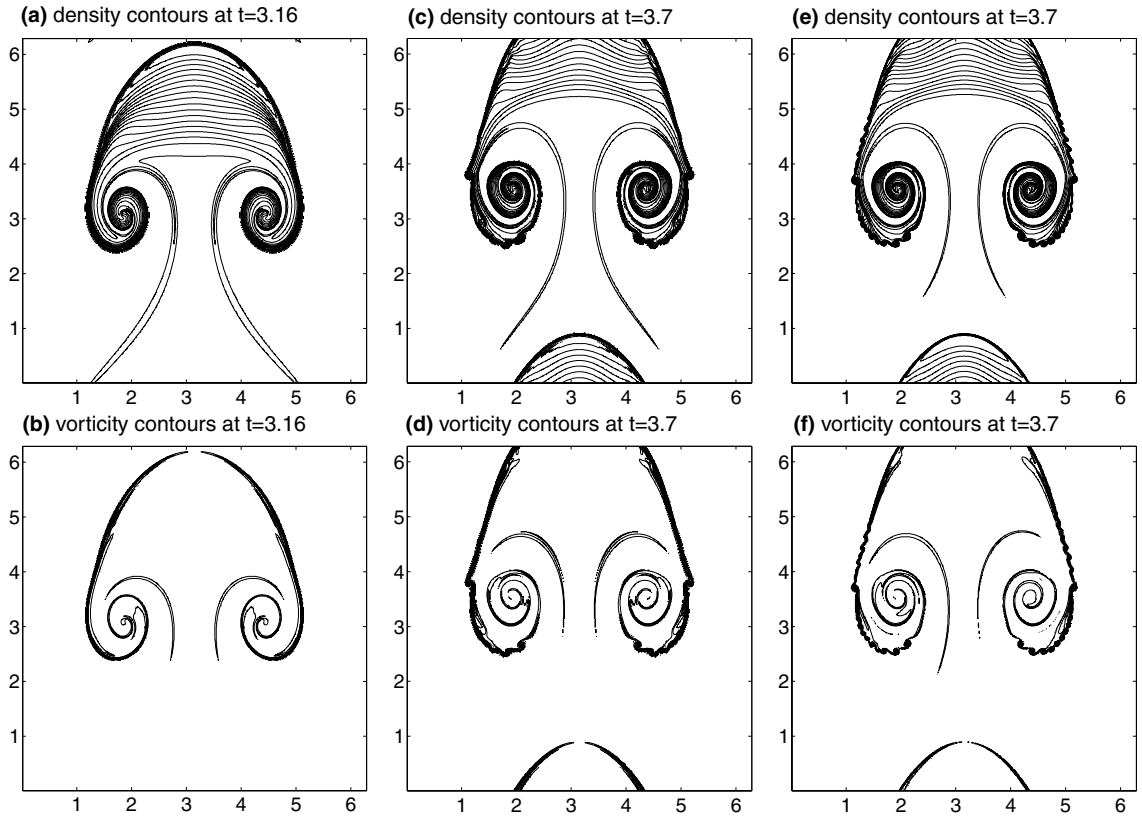

Fig. 5. (a)–(d) show the contour plots of density and vorticity in the resolution of 1024^2. (e) and (f) are the contour plots with the resolution of 4096^2.

Unlike the simulations in Section 2, most computations (two of the three) in this section use the traditional PFFT scheme mainly due to the limitation of the maximum CPU number. If more processors are available, the computer memory will present no problem to the PTF scheme. In this situation, the speed of PTF will also be faster than PFFT.

4. Some extended studies of PTF

- The platform of our research above is on SGI Origin3800 which has a large cache (8 M) and pretty fast interconnect (0.5 GB/s of data bandwidth). For those small cache and low interconnect computers, e.g. PC cluster with 0.5 M cache and 100-baseT (0.1 GB/s of data bandwidth), it will be difficult to obtain super-linear speedup. However, it is likely that PTF will have a greater advantage over PFFT because T_{comm} will have a larger fraction of T_{sum} on PC clusters than on SGI 3800.
- We use 64 processors to test our 2D Boussinesq code with the resolution of 4096^2 on Lenovo DeepComp 1800 (P4 Xeon 2 GHz – Myrinet; $t_{\text{delay}} \simeq 3.0 \times 10^{-6}$ s; $t_{\text{sendrec}} \simeq 1.3 \times 10^{-8}$ s). The 2-N scheme is 80% faster than PFFT, while 4-N is 20% slower than PFFT. This validates our prediction that PTF will continue to be fastest for large resolutions and large number of processors. It is worth mentioning that 4096^2 equals to 256^3, which means PTF might be faster than PFFT for intermediate resolution of 3D simulation on about 100 processors. This observation is different from the conclusion in [16].

From the speedup plots in lower resolutions (64^2–1024^2) of pseudo-spectral codes on DeepComp 1800, we can draw roughly the same conclusions as those in Section 2.3 and 3.2, which will not be discussed here to make the paper concise.

- PTF is especially suitable to deal with low resolutions with longtime evolution. The 1024^2 run in Section 2 lasts for 1.6×10^6 time steps. We are merely dealing with the pseudo-spectral method based on Fourier transform with uniform grid in this paper. In simulations of a Chebyshev expansion with non-uniform grid [39], time steps have to be very small to meet the CFL condition on smallest grid. For example, for a 256^2 resolution run with non-uniform grid in a most recently published paper [40], the time step is 2.0×10^{-4}, which is the same value as our 4096^2 run in this paper. As a result, the 256^2 simulation in [40] lasted for 5×10^6 time steps, and the traditional PFFT scheme will not have a good parallel efficiency. As a first try, we parallelized the code of Clercx [39] by using the PTD scheme, and obtained a speedup of 1.5 for the resolution of 128^2, and 2.5 for 256^2 with 4 processors on SGI 3800. In the future, the PTF scheme will play an important role in this kind of studies.
- It should be noticed that Foster has proposed a similar scheme based on a three FFTs model (see Chapter 4 of [34]), but the parallel efficiency of their scheme is rather limited because too few FFTs are involved: there is no choice but to divide the code into three main tasks. In a sense, the model of Foster is rather misleading because the third FFT should be calculated after the first two FFTs are finished. When dividing the available processors into three groups to calculate these three FFTs independently, the third group will be idling while the first two groups are busy, and the first two groups will be idling while the third group is busy. Thus, the idling time will occupy half of the total cpu time and spoil the efficiency of the parallel scheme (Foster actually measured the performance of their code by making the three groups of processors work simultaneously). In our PTF scheme, ten or twenty FFTs are considered. By skillfully arranging the FFTs based on their logic sequences in calculation (see Tables 1, 3 and 4), there are several choices to divide the groups of processors and there are no idling group throughout the calculation.
- It is well known that the speed of single processor increases much faster than the speed of memory access (five times faster in the last decade). Hence the latency of memory access in terms of processor clock cycle grows by a factor of 5 in the last 10 years [37]. For a fixed type of parallel scheme, the speedup curves (see Figs. 1, 2, and 4) for high resolution (e.g. 1024^2) in future machines may be similar to those of low resolutions in current computers (64^2 or 128^2). In the case of PFFT scheme applied on a 1024^2 resolution, it is possible that the speed of 64-nodes run is slower than that of the serial code in the future. On the other hand, the PTF scheme which has much less T_{delay} than that of the PFFT scheme will have greater advantage over the PFFT scheme in parallel computing. In other words, the PTF strategy is also a scheme for the future.

5. Conclusions

To sum up, the PTF scheme takes the advantages of the PTD and PFFT schemes, and can significantly increase peak speed of codes. The PTF scheme works best for low resolution run, or relatively large resolution with large number of CPUs. For very high resolutions, the PTF scheme is most likely slower than PFFT if not many processors are involved; PTF also needs more memory than that for PFFT, which presents some limitations on the scale of the highest resolution that one particular parallel computer can handle. These two disadvantages of PTF explain why similar schemes are not very popular in the 3D simulations, where the array sizes may be even larger than those for the 2D simulations with high resolutions. It is advisable to treat PFFT as a special case of PTF. Thus, the combined version of the parallel codes will always yield the best performance after a small amount of preparing work (e.g. making a

table similar to Tables 2 and 5 on the particular parallel machine to be used). In the meanwhile, the simple analytical model of the parallel scheme, which divides T_{sum} into three parts: T_{sendrec}, T_{delay}, and T_{comp}, is useful in explaining the performance of the parallel codes.

The parallel code of the NS equations helps us to find another example of double-valued ω–ψ structure; and the parallel Boussinesq code reveals the insight of an open question, which suggests that the collapse of the bubble cap may be a candidate for the finite time singularity formation. Both sets of investigation show that the parallel computing is useful in large scale 2D numerical simulations.

Acknowledgments

We thank Linbo Zhang, Zhongze Li, and Ying Bai for the support of using local parallel computers. Z.Y. also thanks Professor W.H. Matthaeus who supplied the original serial FORTRAN 77 Navier–Stokes pseudo-spectral codes. This work is supported by National Natural Science Foundation of China (G10476032). T.T. thanks the supports from International Research Team on Complex System, Chinese Academy of Sciences, and from Hong Kong Research Grant Council.

References

[1] Z. Yin, H.J.H. Clercx, D.C. Montgomery, An easily implemented task-based parallel scheme for the Fourier pseudospectral solver applied to 2D Navier–Stokes turbulence, Comput. Fluid 33 (2004) 509.
[2] S.A. Orszag, Accurate solution of the Orr–Sommerfeld stability equation, J. Fluid Mech. 50 (1971) 689.
[3] G.S. Patterson, S.A. Orszag, Spectral calculations of isotropic turbulence: efficient removal of aliasing interaction, Phys. Fluid 14 (1971) 2538.
[4] C. Canuto, M.Y. Hussaini, A. Quarteroni, T.A. Zang, Spectral Methods in Fluid Dynamics, Springer, New York, Berlin, 1987.
[5] J.P. Boyd, Chebyshev and Fourier Spectral Methods, second ed., Dover, Mineola/New York, 2001.
[6] S. Chen, G.D. Doolen, R.H. Kraichnan, Z. She, On statistical correlations between velocity increments and locally averaged dissipation in homogeneous turbulence, Phys. Fluid A 5 (1993) 458.
[7] Y. Kaneda, T. Ishihara, M. Yokokawa, K. Itakura, A. Uno, Energy dissipation rate and energy spectrum in high resolution direct numerical simulations of turbulence in a periodic box, Phys. Fluid 15 (2003) L21.
[8] P. Moin, K. Mahesh, Direct numerical simulation: a tool in turbulence research, Annu. Rev. Fluid Mech. 30 (1998) 539.
[9] R.B. Pelz, The parallel Fourier pseudospectral method, J. Comput. Phys. 92 (1991) 296.
[10] E. Jackson, Z. She, S.A. Orszag, A case study in parallel computing: I. Homogeneous turbulence on a hypercube, J. Sci. Comput. 6 (1991) 27.
[11] P.F. Fischer, A.T. Patera, Parallel simulation of viscous incompressible flows, Annu. Rev. Fluid Mech. 26 (1994) 483.
[12] M. Briscolini, A parallel implementation of a 3-D pseudospectral based code on the IBM 9076 scalable POWER parallel system, Parallel Comput. 21 (1995) 1849.
[13] P. Dmitruk, L.P. Wang, W.H. Matthaeus, R. Zhang, D. Seckel, Scalable parallel FFT for spectral simulations on a Beowulf cluster, Parallel Comput. 27 (2001) 1921.
[14] M. Iovieno, C. Cavazzoni, D. Tordella, A new technique for a parallel dealiased pseudospectral Navier–Stokes code, Comput. Phys. Commun. 141 (2001) 365.
[15] A.J. Basu, A parallel algorithm for spectral solution of the three-dimensional Navier–Stokes equations, Parallel Comput. 20 (1994) 1191.
[16] W. Ling, J. Liu, J.N. Chung, C.T. Crowe, Parallel algorithms for particles-turbulence two-way interaction direct numerical simulation, J. Parallel Distributed Comput. 62 (2002) 38.
[17] E. Fournier, S. Gauthier, F. Renaud, 2D pseudo-spectral parallel Navier–Stokes simulations of compressible Rayleigh–Taylor instability, Comput. Fluid 31 (2002) 569.
[18] A. Bracco, J.C. McWilliams, G. Murante, A. Provenzale, J.B. Weiss, Revisiting freely decaying two-dimensional turbulence at millennial resolution, Phys. Fluid 12 (2000) 2931.
[19] P. Dmitruk, D.C. Montgomery, Numerical study of the decay of enstrophy in a two-dimensional Navier–Stokes fluid in the limit of very small viscosities, Phys. Fluid 17 (2005) 035114.

[20] Z. Yin, D.C. Montgomery, H.J.H. Clercx, Alternative statistical–mechanical descriptions of decaying two-dimensional turbulence in terms of "patches" and "points", Phys. Fluid 15 (2003) 1937.
[21] Z. Yin, On final states of two-dimensional decaying turbulence, Phys. Fluid 16 (2004) 4623.
[22] A. Pumir, E.D. Siggia, Development of singular solutions to the axisymmetric Euler equations, Phys. Fluid A 4 (1992) 1472.
[23] W. E, C. Shu, Small-scale structures in Boussinesq convection, Phys. Fluid 6 (1994) 49.
[24] H.D. Ceniceros, T.Y. Hou, An efficient dynamically adaptive mesh for potentially singular solutions, J. Comput. Phys. 172 (2001) 609.
[25] D. Montgomery, G.R. Joyce, Statistical mechanics of negative temperature states, Phys. Fluid 17 (1974) 1139.
[26] D. Montgomery, W.H. Matthaeus, W.T. Stribling, D. Martinez, S. Oughton, Relaxation in two dimensions and the "sinh-poisson" equation, Phys. Fluid A 4 (1992) 3.
[27] S.A. Orszag, Numerical simulation of incompressible flows within simple boundaries. I. Galerkin (spectral) representation, Stud. Appl. Math. 50 (1971) 293.
[28] P.N. Swarztrauber, Multiprocessor FFTs, Parallel Comput. 5 (1987) 197.
[29] R.M. Chamberlain, Gray codes, Fast Fourier Transforms and hypercubes, Parallel Comput. 6 (1988) 225.
[30] R.B. Pelz, Parallel compact FFTs for real sequences, SIAM J. Sci. Comput. 14 (1993) 914.
[31] C. Mermer, D. Kim, Y. Kim, Efficient 2D FFT implementation on mediaprocessors, Parallel Comput. 29 (2003) 691.
[32] E. Chu, A. George, Inside the FFT Black Box, CRC Press, Boca Raton, London, New York, Washington, DC, 2000.
[33] C.Y. Chu, Comparison of two-dimensional FFT methods on the hypercubeProceedings of the Third Conference on Hypercube Concurrent Comput. Appl., Pasadena, CA, United States, 1988, vol. 2, ACM Press, New York, NY, USA, 1989, p. 1430.
[34] I. Foster, Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering, Addison Wesley, Pearson Education, Inc., 1995.
[35] D. Roose, R. Van Driessche, Parallel computers and parallel algorithms for CFD: an introduction, Special Course on Parallel Computing in CFD, AGARD R-807, NATO, 1995, p. 1.1.
[36] B.B. Fraguela, R. Doallo, J. Touriño, E.L. Zapata, A compiler tool to predict memory hierarchy performance of scientific codes, Parallel Comput. 30 (2004) 225 (and references therein).
[37] D.E. Culler, J.P. Singh, A. Gupta, Parallel Computer Architecture, Morgan Kaufmann Publishers, Inc., San Francisco, 1999.
[38] S. Ghosh, M. Hossain, W.H. Matthaeus, The application of spectral methods in simulating compressible fluid and magnetofluid turbulence, Comput. Phys. Commun. 74 (1993) 18.
[39] H.J.H. Clercx, A spectral solver for the Navier–Stokes equations in the velocity–vorticity formulation for flows with two nonperiodic directions, J. Comput. Phys. 137 (1997) 186.
[40] D. Molenaar, H.J.H. Clercx, G.J.F. van Heijst, Angular momentum of forced 2D turbulence in a square no-slip domain, Physica D 196 (2004) 329.