JAERI-Data/Code
96-014

等方性乱流コードの並列化

1996年3月

佐藤　滋*・横川三津夫・渡辺　正・蕪木英雄

日本原子力研究所
Japan Atomic Energy Research Institute

本レポートは，日本原子力研究所が不定期に公刊している研究報告書です。
入手の問合わせは，日本原子力研究所技術情報部情報資料課（〒319-11 茨城県那珂郡東海村）あて，お申し越しください。なお，このほかに財団法人原子力弘済会資料センター（〒319-11 茨城県那珂郡東海村日本原子力研究所内）で複写による実費頒布をおこなっております。

This report is issued irregularly.
Inquiries about availability of the reports should be addressed to Information Division, Department of Technical Information, Japan Atomic Energy Research Institute, Tokai-mura, Naka-gun, Ibaraki-ken 319-11, Japan.

編集兼発行　日本原子力研究所
印　　刷　㈱原子力資料サービス

JAERI－Data/Code 96－014

等方性乱流コードの並列化

日本原子力研究所計算科学技術推進センター
佐藤　滋*・横川三津夫・渡辺　正
蕪木　英雄

（1996年2月9日受理）

本論文では，3次元等方性乱流のシミュレーションを行なうために，分散メモリ型並列ベクトル計算機VPP500上でスペクトル法によるプログラムを作成し，実行時間や並列化効率，ロードバランス，スケーラビリティ等の性能の評価を行なった結果について述べる。計算にはプロセッサを16個まで使用し，フーリエモード数256 × 256 × 256までの解析を行なった。その結果，フーリエモード数を固定した場合にはプロセッサ数に対しては良いスケーラビリティが得られた。フーリエモード数が小さい領域では3次元FFTの計算を並列ベクトル計算機の理想的な計算時間と考えられる$N\log N$程度の時間で行なえるが，フーリエモード数が増えるとともに計算時間が増え，並列化効率は落ちることがわかった。

日本原子力研究所：〒153　東京都目黒区中目黒2－2－54
＊業務協力員，㈱日本総合研究所

JAERI－Data/Code 96－014

Parallelization of a Numerical Simulation Code for Isotropic Turbulence

Shigeru SATO*, Mitsuo YOKOKAWA, Tadashi WATANABE
and Hideo KABURAKI

Center for Promotion of Computational Science and Engineering
Japan Atomic Energy Research Institute
Nakameguro, Meguro-ku, Tokyo

(Received February 9, 1996)

A parallel pseudospectral code which solves the three-dimensional Navier-Stokes equation by direct numerical simulation is developed and execution time, parallelization effeciency, load balance and scalability are evaluated. A vector parallel supercomputer, Fujitsu VPP500 with up to 16 processors is used for this calculation for Fourier modes up to $256 \times 256 \times 256$ using 16 processors. Good scalability for number of processors is achieved when number of Fourier mode is fixed. For small Fourier modes, calculation time of the program is propotional to $N\log N$ which is ideal complexity of calculation for 3D-FFT on vector parallel processors. It is found that the calculation performance decreases as the increase of the Fourier modes.

Keywords: Navier-stokes Equation, Isotropic Turbulence, FFT, Spectral Method, Parallel Programming, VPP500

* On loan to Japan Research Institute, Limited

目　　次

Contents

1. 序論

近年コンピュータの発達とともに、ナビエ・ストークス方程式の直接数値シミュレーションが行なわれており、特に多くの計算時間とメモリを必要とする解析のひとつとして、高レイノルズ数の乱流現象のシミュレーションがある。

本報告では、３次元等方性乱流のシミュレーションをスペクトル法により行なうプログラムをFujitsu VPP500上で開発し、並列化の性能評価をした結果について述べる。VPP500は１プロセッサあたり約200メガバイトの主記憶と1.6 G flopsの演算性能を持つ分散メモリ型のベクトル並列計算機であり、今回の解析ではプロセッサ数を16個まで使用し、フーリエモード数 $256\times256\times256$ までの計算を行なった。

2. 計算手法

今回開発した計算コードには、基礎方程式として次に示す連続の式とナビエ・ストークス方程式を用いた。

$$\nabla\cdot u = 0 \quad \text{（連続の式）} \tag{1}$$

$$\frac{\partial u}{\partial t}+(u\cdot\nabla)u = -\frac{1}{\rho}\nabla p+\nu\nabla^2 u \quad \text{（Navier-Stokes 方程式）} \tag{2}$$

この式を渦度を用いて表し、速度、渦度にフーリエ展開を行ない、アインシュタインの規約を用いて表すと次式になる。ここで、ϵ_{jkl} は交代テンソルで、$\tilde{\boldsymbol{u}}_j(\boldsymbol{k})$、$\tilde{\boldsymbol{w}}_j(\boldsymbol{k})$ はそれぞれ速度 $\boldsymbol{u}_j(\boldsymbol{x})$ と渦度 $\boldsymbol{w}_j(\boldsymbol{x})$ のフーリエ係数であり、$\widetilde{\boldsymbol{u}_l\boldsymbol{u}_m}(\boldsymbol{k})$ は、$\boldsymbol{u}_l(\boldsymbol{x})\boldsymbol{u}_m(\boldsymbol{x})$ のフーリエ係数を表す。

$$\frac{d}{dt}\tilde{w}_j(k) = \epsilon_{jkl}k_k k_m\widetilde{u_l u_m}(k)-\nu k^2\tilde{w}_j(k) \quad , \quad j=1,2,3 \tag{3}$$

$$k_j\tilde{u}_j(k) = 0 \tag{4}$$

$$\tilde{w}_j(k) = -\epsilon_{jkl}k_k\tilde{u}_l(k) \tag{5}$$

$$u(x) = i\sum_{k_1=-\frac{N}{2}}^{\frac{N}{2}-1}\sum_{k_2=-\frac{N}{2}}^{\frac{N}{2}-1}\sum_{k_3=-\frac{N}{2}}^{\frac{N}{2}-1}\tilde{u}(k)\exp(ik\cdot x) \tag{6}$$

$$w(x) = \sum_{k_1=-\frac{N}{2}}^{\frac{N}{2}-1}\sum_{k_2=-\frac{N}{2}}^{\frac{N}{2}-1}\sum_{k_3=-\frac{N}{2}}^{\frac{N}{2}-1}\tilde{w}(k)\exp(ik\cdot x) \tag{7}$$

$$u_l(x)u_m(x) = \sum_{k_1=-\frac{N}{2}}^{\frac{N}{2}-1}\sum_{k_2=-\frac{N}{2}}^{\frac{N}{2}-1}\sum_{k_3=-\frac{N}{2}}^{\frac{N}{2}-1}\widetilde{u_l u_m}(k)\exp(ik\cdot x) \tag{8}$$

時間積分には次式の４段４次のルンゲ・クッタを用いた。

$$\tilde{w}_{old} = \tilde{w}$$

$$\tau = RHS(\tilde{w}),$$

1. 序論

近年コンピュータの発達とともに、ナビエ・ストークス方程式の直接数値シミュレーションが行なわれており、特に多くの計算時間とメモリを必要とする解析のひとつとして、高レイノルズ数の乱流現象のシミュレーションがある。

本報告では、３次元等方性乱流のシミュレーションをスペクトル法により行なうプログラムをFujitsu VPP500上で開発し、並列化の性能評価をした結果について述べる。VPP500は１プロセッサあたり約200メガバイトの主記憶と1.6 G flopsの演算性能を持つ分散メモリ型のベクトル並列計算機であり、今回の解析ではプロセッサ数を16個まで使用し、フーリエモード数 $256\times256\times256$ までの計算を行なった。

2. 計算手法

今回開発した計算コードには、基礎方程式として次に示す連続の式とナビエ・ストークス方程式を用いた。

$$\nabla\cdot u = 0 \quad \text{（連続の式）} \tag{1}$$

$$\frac{\partial u}{\partial t}+(u\cdot\nabla)u = -\frac{1}{\rho}\nabla p+\nu\nabla^2 u \quad \text{（Navier-Stokes 方程式）} \tag{2}$$

この式を渦度を用いて表し、速度、渦度にフーリエ展開を行ない、アインシュタインの規約を用いて表すと次式になる。ここで、ϵ_{jkl} は交代テンソルで、$\tilde{\boldsymbol{u}}_j(\boldsymbol{k})$、$\tilde{\boldsymbol{w}}_j(\boldsymbol{k})$ はそれぞれ速度 $\boldsymbol{u}_j(\boldsymbol{x})$ と渦度 $\boldsymbol{w}_j(\boldsymbol{x})$ のフーリエ係数であり、$\widetilde{u_l u_m}(\boldsymbol{k})$ は、$\boldsymbol{u}_l(\boldsymbol{x})\boldsymbol{u}_m(\boldsymbol{x})$ のフーリエ係数を表す。

$$\frac{d}{dt}\tilde{w}_j(k) = \epsilon_{jkl}k_k k_m\widetilde{u_l u_m}(k)-\nu k^2\tilde{w}_j(k) \quad , \quad j=1,2,3 \tag{3}$$

$$k_j\tilde{u}_j(k) = 0 \tag{4}$$

$$\tilde{w}_j(k) = -\epsilon_{jkl}k_k\tilde{u}_l(k) \tag{5}$$

$$u(x) = i\sum_{k_1=-\frac{N}{2}}^{\frac{N}{2}-1}\sum_{k_2=-\frac{N}{2}}^{\frac{N}{2}-1}\sum_{k_3=-\frac{N}{2}}^{\frac{N}{2}-1}\tilde{u}(k)\exp(ik\cdot x) \tag{6}$$

$$w(x) = \sum_{k_1=-\frac{N}{2}}^{\frac{N}{2}-1}\sum_{k_2=-\frac{N}{2}}^{\frac{N}{2}-1}\sum_{k_3=-\frac{N}{2}}^{\frac{N}{2}-1}\tilde{w}(k)\exp(ik\cdot x) \tag{7}$$

$$u_l(x)u_m(x) = \sum_{k_1=-\frac{N}{2}}^{\frac{N}{2}-1}\sum_{k_2=-\frac{N}{2}}^{\frac{N}{2}-1}\sum_{k_3=-\frac{N}{2}}^{\frac{N}{2}-1}\widetilde{u_l u_m}(k)\exp(ik\cdot x) \tag{8}$$

時間積分には次式の４段４次のルンゲ・クッタを用いた。

$$\tilde{w}_{old} = \tilde{w}$$

$$\tau = RHS(\tilde{w}),$$

$$
\begin{aligned}
& \tilde{w} = \tilde{w}_{old} + \frac{\Delta t}{6}\tau, \qquad w' = \tilde{w}_{old} + \frac{\Delta t}{2}\tau \\
\tau &= RHS(w'), \\
& \tilde{w} = \tilde{w} + \frac{\Delta t}{3}\tau, \qquad w' = \tilde{w}_{old} + \frac{\Delta t}{2}\tau \\
\tau &= RHS(w'), \\
& \tilde{w} = \tilde{w} + \frac{\Delta t}{3}\tau, \qquad w' = \tilde{w}_{old} + \Delta t\tau \\
\tau &= RHS(w'), \qquad \tilde{w} = \tilde{w} + \frac{\Delta t}{6}\tau
\end{aligned}
$$

ここで、$RHS(\tilde{w})$ は、(3) 式の右辺を表す。

3. 並列化について

計算領域は１辺が２πの３次元立方領域である。この各辺を上式のフーリエモード数で分割し、ひとつの次元に関してロードバランスが均等になるようにメッシュにおける計算を各プロセッサにわりあて、並列処理を行なう（Fig.3.1)。

ここでメッシュを割り当てる際に、メッシュ数がプロセッサ数で割り切れれば全プロセッサに均等に負荷が分散され、並列化の効率も良いものとなる。

本プログラムでは１次元 FFT の計算を x、y、z の各方向に繰り返して行なうことによって３次元の FFT を計算している。一般的に、フーリエ変換では他のメッシュのデータを必要とするため、FFT の計算に必要なデータが他のプロセッサ上に存在する場合には、プロセッサ間でデータの通信を行なう必要がある。並列計算における多次元の FFT の計算手法には、良く知られたものとしてそれぞれの軸方向にプロセッサ間の通信を行ないながら１次元の FFT を行なうやり方（distributed FFT）と、１次元の FFT を１つのプロセッサ内だけで計算し、各々の軸について計算する場合には、データを並び変えて FFT 計算ルーチンに渡してやるという手法（transpose FFT）がある。本プログラムでは後者の transpose FFT を用いた。この方法では、データの並び変えの際にプロセッサ間でデータの通信が行なわれることになる。

ナビエ・ストークス方程式の非線形項のフーリエ係数は、一旦実空間に逆変換して積を計算し、もう一度フーリエ空間に変換することによって求める。このとき高速フーリエ変換を用いることができる。この方法は擬スペクトル法と呼ばれているが、この際に aliasing error が生じるので、３／２則で error を除いた。これは、フーリエモード数を N としたときに $M = 2/3N$ であらわされる有効モード数 M よりも大きいモードに対して (9)〜(11) 式のように 0 を代入してフーリエ変換を行なう。

$$x_j = 2\pi j/N \tag{9}$$

$$u_j = \sum_{k=-N/2}^{N/2-1} \hat{u}_k e^{ikx_j} \tag{10}$$

$$
\begin{aligned}
& \tilde{w} = \tilde{w}_{old} + \frac{\Delta t}{6}\tau, \qquad w' = \tilde{w}_{old} + \frac{\Delta t}{2}\tau \\
\tau &= RHS(w'), \\
& \tilde{w} = \tilde{w} + \frac{\Delta t}{3}\tau, \qquad w' = \tilde{w}_{old} + \frac{\Delta t}{2}\tau \\
\tau &= RHS(w'), \\
& \tilde{w} = \tilde{w} + \frac{\Delta t}{3}\tau, \qquad w' = \tilde{w}_{old} + \Delta t\tau \\
\tau &= RHS(w'), \qquad \tilde{w} = \tilde{w} + \frac{\Delta t}{6}\tau
\end{aligned}
$$

ここで、$RHS(\tilde{w})$ は、(3) 式の右辺を表す。

3. 並列化について

計算領域は１辺が２πの３次元立方領域である。この各辺を上式のフーリエモード数で分割し、ひとつの次元に関してロードバランスが均等になるようにメッシュにおける計算を各プロセッサにわりあて、並列処理を行なう（Fig.3.1)。

ここでメッシュを割り当てる際に、メッシュ数がプロセッサ数で割り切れれば全プロセッサに均等に負荷が分散され、並列化の効率も良いものとなる。

本プログラムでは１次元 FFT の計算を x、y、z の各方向に繰り返して行なうことによって３次元の FFT を計算している。一般的に、フーリエ変換では他のメッシュのデータを必要とするため、FFT の計算に必要なデータが他のプロセッサ上に存在する場合には、プロセッサ間でデータの通信を行なう必要がある。並列計算における多次元の FFT の計算手法には、良く知られたものとしてそれぞれの軸方向にプロセッサ間の通信を行ないながら１次元の FFT を行なうやり方（distributed FFT）と、１次元の FFT を１つのプロセッサ内だけで計算し、各々の軸について計算する場合には、データを並び変えて FFT 計算ルーチンに渡してやるという手法（transpose FFT）がある。本プログラムでは後者の transpose FFT を用いた。この方法では、データの並び変えの際にプロセッサ間でデータの通信が行なわれることになる。

ナビエ・ストークス方程式の非線形項のフーリエ係数は、一旦実空間に逆変換して積を計算し、もう一度フーリエ空間に変換することによって求める。このとき高速フーリエ変換を用いることができる。この方法は擬スペクトル法と呼ばれているが、この際に aliasing error が生じるので、３／２則で error を除いた。これは、フーリエモード数を N としたときに $M = 2/3N$ であらわされる有効モード数 M よりも大きいモードに対して (9)～(11) 式のように 0 を代入してフーリエ変換を行なう。

$$x_j = 2\pi j/N \tag{9}$$

$$u_j = \sum_{k=-N/2}^{N/2-1} \hat{u}_k e^{ikx_j} \tag{10}$$

$$\hat{u}_k = \begin{cases} \tilde{u}_k & -M/2 \le k \le M/2 \\ 0 & otherwise \end{cases} \tag{11}$$

次に、計算時間の測定方法について述べる。本コードは２つのプログラムから構成されている。最初のプログラムでは初期速度場を計算してファイルに出力し、次のプログラムで初期速度ファイルを読み込み、ルンゲクッタ法で時間ステップを進めながらナビエ・ストークス方程式を擬スペクトル法で解く。計算時間の大部分は後者のプログラムの時間ステップの計算の中の高速フーリエ変換である。他にファイルから読み込む時間と並列計算のための準備時間がかかるが、以下の計算結果の解析ではフーリエ変換に着目しているため、後者のプログラムについてファイルからデータを読み込む時間及び並列計算のための準備時間（オーバーヘッド）にかかった時間を除いた時間を VPP-Fortran の gettod 関数を用いて測定し、計算時間の評価に用いる。プログラムのコンパイルは、システムの標準値である $-Oe$ オプション（最適化レベル e）を用いて行なった。なお、計算時間を測定するのに clock ではなくて gettod を用いたのは、プロセッサ間のデータの通信にかかった時間も考慮するためである。

4. 計算結果の解析

本プログラムを用いて、計算を行ない、フーリエモード数に対する計算時間及び、プロセッサ数に対する計算時間と並列化効率を測定した。また、等方性乱流のシミュレーションを行ない、結果が Kolgomorov の$-5/3$ 乗則に従っていることを確認した。

4.1 フーリエモード数に対する計算時間の変化

一般に、３次元フーリエ変換では計算量が $O(N^4)$ になるが、高速フーリエ変換を用いると計算量を $O(N^3 \log N)$ に減らすことができる。また、これをベクトル化した場合 N が $O(10)$ 程度の領域では３次元のうちのひとつの次元について同時にベクトル処理できると考えられ、この計算量は $O(N^2 \log N)$ になると考えられる。

さらに、ひとつの次元について P 個のプロセッサで並列化したとすると、この計算量は

$$O(\frac{N^2}{P} \log N) \tag{12}$$

となるが、プロセッサ数 P がフーリエモード数 N に比べて大きければ、ひとつの次元に対しての計算が同時に処理できると考えられるため、計算時間は $O(N \log N)$ に比例するものと予想できる。

そこで、スカラシリアル、ベクトルシリアル、ベクトルパラレルの３通りの計算を、フーリエモード数を $2^4, 2^5, 2^6, 2^7$ について計算し、計算時間を測定した。但し、ベクトルパラレル計算は、プロセッサを 16 個用いて行なった。

スカラシリアル、ベクトルシリアル、ベクトルパラレルの３通りの計算を、フーリエモード数を 16 から 256 の間で変化させて行なった結果を Table 4.1.1 に、また、それらをプロットした結果を Fig.4.1.1 に示す。ここで、ベクトルパラレル計算は、プロセッサ数 16 個で行なった。

$$\hat{u}_k = \begin{cases} \tilde{u}_k & -M/2 \leq k \leq M/2 \\ 0 & otherwise \end{cases} \tag{11}$$

次に、計算時間の測定方法について述べる。本コードは２つのプログラムから構成されている。最初のプログラムでは初期速度場を計算してファイルに出力し、次のプログラムで初期速度ファイルを読み込み、ルンゲクッタ法で時間ステップを進めながらナビエ・ストークス方程式を擬スペクトル法で解く。計算時間の大部分は後者のプログラムの時間ステップの計算の中の高速フーリエ変換である。他にファイルから読み込む時間と並列計算のための準備時間がかかるが、以下の計算結果の解析ではフーリエ変換に着目しているため、後者のプログラムについてファイルからデータを読み込む時間及び並列計算のための準備時間（オーバーヘッド）にかかった時間を除いた時間を VPP-Fortran の gettod 関数を用いて測定し、計算時間の評価に用いる。プログラムのコンパイルは、システムの標準値である $-Oe$ オプション（最適化レベル e）を用いて行なった。なお、計算時間を測定するのに clock ではなくて gettod を用いたのは、プロセッサ間のデータの通信にかかった時間も考慮するためである。

4. 計算結果の解析

本プログラムを用いて、計算を行ない、フーリエモード数に対する計算時間及び、プロセッサ数に対する計算時間と並列化効率を測定した。また、等方性乱流のシミュレーションを行ない、結果が Kolgomorov の$-5/3$ 乗則に従っていることを確認した。

4.1 フーリエモード数に対する計算時間の変化

一般に、３次元フーリエ変換では計算量が $O(N^4)$ になるが、高速フーリエ変換を用いると計算量を $O(N^3 \log N)$ に減らすことができる。また、これをベクトル化した場合 N が $O(10)$ 程度の領域では３次元のうちのひとつの次元について同時にベクトル処理できると考えられ、この計算量は $O(N^2 \log N)$ になると考えられる。

さらに、ひとつの次元について P 個のプロセッサで並列化したとすると、この計算量は

$$O(\frac{N^2}{P} \log N) \tag{12}$$

となるが、プロセッサ数 P がフーリエモード数 N に比べて大きければ、ひとつの次元に対しての計算が同時に処理できると考えられるため、計算時間は $O(N \log N)$ に比例するものと予想できる。

そこで、スカラシリアル、ベクトルシリアル、ベクトルパラレルの３通りの計算を、フーリエモード数を $2^4, 2^5, 2^6, 2^7$ について計算し、計算時間を測定した。但し、ベクトルパラレル計算は、プロセッサを 16 個用いて行なった。

スカラシリアル、ベクトルシリアル、ベクトルパラレルの３通りの計算を、フーリエモード数を 16 から 256 の間で変化させて行なった結果を Table 4.1.1 に、また、それらをプロットした結果を Fig.4.1.1 に示す。ここで、ベクトルパラレル計算は、プロセッサ数 16 個で行なった。

図より、スカラシリアル、ベクトルシリアルの計算時間は、予想通りそれぞれほぼ $O(N^3 \log N)$、$O(N^2 \log N)$ に従っている。

また、ベクトルパラレルの実際の計算時間は、モード数が小さい時には $O(N \log N)$ に近い傾きになっているが、モード数が増えるに従って勾配は $O(N^2 \log N)$ から $O(N^3 \log N)$ に近付くことがわかる。これは、実際の予想（ $O(N \log N)$～$O(N^2 \log N)$ 程度の傾き）より大きい。そこで、プログラムをより詳細に調べた。

本プログラムで計算時間の大部分が FFT であるが、このなかでもっとも時間がかかる部分は下に示したような do loop の部分である。

```
*vdir novector
*vocl loop,scalar
      do  400  istage = 1, nl1, 2
          ....
*vocl loop,vector
        do  410  k = kst,ken
*vocl loop,novrec
          do  410  j = 1, n2
            j2  =  2*j
            j1  =  j2 - 1
            do  410  i = 1, m
              cr(i,j1,k)  =  fr(i,j,k)  +  fr(i,j+n2,k)
              ci(i,j1,k)  =  fi(i,j,k)  +  fi(i,j+n2,k)
              temp1       =  fr(i,j,k)  -  fr(i,j+n2,k)
              temp2       =  fi(i,j,k)  -  fi(i,j+n2,k)
              cr(i,j2,k)  =  temp1*cn(j+jbase)  +  temp2*sn(j+jbase)
              ci(i,j2,k)  =  temp2*cn(j+jbase)  -  temp1*sn(j+jbase)
 410    continue
        ...
 400  continue
```

ベクトルパラレル計算により、本プログラムのこの部分についてのみの計算時間を測定し、フーリエモード数に対してプロットしたものが、Fig.4.1.2 である。図に示すようにモード数が大きくなるにつれて計算時間は $N^3 \log N$の傾きに近くなり、プログラム全体の計算時間のグラフとほぼ傾向を示している。この部分の計算ではプロセッサに対する計算の割り当てが 3番めの次元（変数 k ）について行なわれているが、プロセッサ間の通信は行なわれていない。したがって、プログラム全体の計算時間が $N^3 \log N$の傾きに近付くのは、メッシュサイズが増えたことによるプロセッサ間のデータ通信量の増大のせいではなく、他に起因するものだと言える。

さらに下に示すように、計算のこの部分だけからなる別プログラムを作成して、ループ回数や各プロセッサへの計算の割り当てかた、配列の大きさなどを上と同じものにしてこの do loop にかかる計算時間を測定した。

```
      parameter (mpe=16)
      parameter (loopc=1000)
c     parameter (ml=4, mk=10)
c     parameter (ml=5, mk=20)
c     parameter (ml=6, mk=42)
      parameter (ml=7, mk=84)
c     parameter (ml=8, mk=170)
```

```
      parameter (m=2**ml, mp=m+2, m2=m/2, mk2=mk/2, ml1=ml-1 )
c
      real*8 ttt(10), sss(10)
      real*8 xxx1, xxx2, xxx
      real*8 yyy1, yyy2, yyy
c
!xocl processor p(mpe)
!xocl index partition ipnk=(p,index=1:mk,part=band)

c
      DIMENSION  FR(mp,m,mk),FI(mp,m,mk),CR(mp,m,mk),CI(mp,m,mk)
!xocl local  FR(:,:,/ipnk),FI(:,:,/ipnk),CR(:,:,/ipnk),CI(:,:,/ipnk)
      DIMENSION  CN(m2*ml1),  SN(m2*ml1)
c
c
      write(6,*) ' number of processors=',mpe
      write(6,*) ' number of mode=',mk
c
      sss(1)=0.0
      sss(2)=0.0
      xxx = 0.0
      yyy = 0.0
c
      icount=0
c
c---------------------------
c     call gettod(ttt(1))
c---------------------------
!xocl parallel region
!xocl spread do
      do 88 iiii=1,mpe
 88    idpe=iiii
!xocl end spread

c-----------------------------------------------------------------------/
      nnn=mk
      if(mod(nnn,mpe).eq.0) then
         mmm = nnn/mpe
      else
         mmm = (nnn/mpe)+1
      endif

      kst=1
      ken=kst+mmm-1
      do 11 ipe=2,idpe
         kst=kst+mmm
         ken=kst+mmm-1
         if(ken.gt.nnn)then
            ken=nnn
         end if
 11   continue
c-----------------------------------------------------------------------/
      nnn=N
      if(mod(nnn,mpe).eq.0) then
         mmm = nnn/mpe
      else
         mmm = (nnn/mpe)+1
      endif

      nst=1
      nen=nst+mmm-1
      do 12 ipe=2,idpe
         nst=nst+mmm
         nen=nst+mmm-1
         if(nen.gt.nnn)then
            nen=nnn
         end if
```

```
 12   continue

!xocl spread do
      do 89 iiii=1,mpe
 89   write(6,*)'PE',iiii,' K:',kst,ken,' N:',nst,nen
!xocl end spread

c
      do 50 k=kst, ken
      do 50 j=1, m
      do 50 i=1, mp
           fr(i,j,k)=0.0
           fi(i,j,k)=0.0
   50 continue
c
      do 60 i=1, m2*ml1
         sn(i)=0.0
         cn(i)=0.0
   60 continue
c
   10 continue
      icount=icount+1
c--------------------------
      call clock(xxx1,2,2)
      call gettod(ttt(1))
c--------------------------
c
*VDIR NOVECTOR
*vocl loop,scalar
      DO  400  ISTAGE = 1, mL1, 2
        JBASE  =  m2*(ISTAGE-1)
*vocl loop,vector
        DO  410  K = kst,ken
*VOCL LOOP,NOVREC
          DO  410  J = 1, m2
            J2  =  2*J
            J1  =  J2 - 1
            DO  410  I = 1, m
              CR(I,J1,K)  =  FR(I,J,K)  +  FR(I,J+m2,K)
              CI(I,J1,K)  =  FI(I,J,K)  +  FI(I,J+m2,K)
              TEMP1       =  FR(I,J,K)  -  FR(I,J+m2,K)
              TEMP2       =  FI(I,J,K)  -  FI(I,J+m2,K)
              CR(I,J2,K)  =  TEMP1*CN(J+JBASE)  +  TEMP2*SN(J+JBASE)
              CI(I,J2,K)  =  TEMP2*CN(J+JBASE)  -  TEMP1*SN(J+JBASE)
 410    CONTINUE
c
c--------------------------
      call clock(xxx2,2,2)
      call gettod(ttt(2))
c--------------------------
      sss(1)=sss(1)+ttt(2)-ttt(1)
      xxx = xxx + xxx2-xxx1
c
  400 continue
c
  500 continue
      if( icount.lt.loopc ) goto 10
!xocl end parallel
c
      write(6,*) ' loopc=',loopc
      write(6,*)
      write(6,*) ' in  parallel regin1 tod_time = ',sss(1)
      write(6,*) ' in  parallel regin1 cpu_time = ',xxx
c
      stop
      end
```

各モード数に対する計算時間を Fig.4.1.3 に示す。この図は、プログラム全体を実行した中でこの do loop にかかった計算時間を測定したものとは明らかに異なり、N が小さい領域では $O(N \log N)$ よりも傾きが小さく、N が 100 程度の領域で当初のベクトルパラレル計算にかかる計算時間の予想に近い $O(N \log N)$ になっている。

以上より、この部分にかかる計算時間は、プログラム全体のメモリ管理、プロセス管理などとも関係するため、この部分だけの計算時間を部分的に測定してもパフォーマンスを正しく評価することはできないと思われる。

さらに、この勾配の変化は、この他にベクトル長やプロセッサ数、及び様々なコンパイラの最適化オプションなどによっても変わると思われるが、今後詳細に検討したい。

4.2 プロセッサ数に対する計算時間の変化

次に、プロセッサ数を 1 個から 16 個まで変化させた場合の計算時間の変化を、モード数 64^3 と 128^3 の 2 ケースについて調べた。結果を Table 4.2.1 及び Fig.4.2.1 に示す。

この図より、モード数 128 の場合の実行時間はほぼ 1／（プロセッサ数）になっていて、プロセッサ個数についてのスケーラビリティは保たれていることがわかる。

並列化効率のグラフを Fig.4.2.2 に示す。ここで、並列化効率は (13) 式で定義する。

$$\text{並列化効率} = \frac{T_1}{P \times T_P} \tag{13}$$

P ＝ 使用プロセッサ数

T_1 ＝ 1プロセッサ使用時の実行時間

T_p ＝ Pプロセッサ使用時の実行時間

なお、モード数が 128 の場合の計算は、1 プロセッサでは計算に必要なメモリが確保できないため、2 プロセッサ以上で計算を行ない、2 プロセッサ使用時の並列化効率を 1.0 としてグラフを作成した。

図のモード数が 64（有効モード数＝ 42）のケースでプロセッサ数が 1 、 2 個の場合にくらべて 4 、 8 、 16 個の場合の並列化効率が落ちているのは、有効モード数をプロセッサに均等に割り当てることができなかったためと思われる。モード数が 128（有効モード数＝ 84）の場合のプロセッサ数 8 、 16 についても同様である。

全体的な傾向として、プロセッサ数が増えるに従って並列化効率は下がっていることがわかる。

比較のため、単純な do loop を各プロセッサに割り当てて、ロードバランスを変化させ、並列化効率に与える影響を測定してみた。この計算はローカル配列の同一プロセッサ内のデータの和の計算を行なっており、プロセッサ間の通信はしていないことになる。

```
      do 100 k=kst, ken
      do 100 j=1, nn
      do 100 i=1, mm
```

```
          a1(i,j,k)=b1(i,j,k)+c1(i,j,k)
100 continue
```

この場合の計算時間を Table 4.2.2 に、また並列化効率のグラフを、ロードバランスから直接計算した並列化効率の値とともに Fig.4.2.3 に示す。単純な、通信を伴わない計算の場合にはこのようにロードバランスが直接並列化効率に反映された値と非常に近い結果になる。しかし、実際のコード内でこのような計算が行なわれる場合には前述のモード数対計算時間の解析で述べたのと同じく、この部分だけの計算時間の評価では全体のパフォーマンスは評価できないものと思われ、これについても正しく評価するにはさらに測定方法を検討する必要がある。

4.3 等方性乱流シミュレーション

この並列化コードを用いて、初期速度場としてガウス分布でエネルギースペクトル分布が $k^4 \exp(-2k^2)$ に比例する速度場を与え、低スペクトル領域でエネルギーを与え続けて定常状態になるまで計算した。結果を Fig.4.3.1 に示す。

計算に用いたパラメータは、動粘度 $= 0.005[m^2/s]$、$\Delta\, t = 0.005[sec]$ である。図より、計算結果はほぼ Kolgomorov の $-5/3$ 乗則に従っていることがわかる。

5. まとめ

本報告では、等方性乱流シミュレーションコードを並列ベクトル計算機 VPP500 上で評価した結果について示した。3次元の高速フーリエ変換にかかる計算時間は、ベクトルシリアル計算の場合 $N^2 \log N$ に比例するが、ベクトルパラレル計算の場合、フーリエモード数が小さい時には $N \log N$ で、モード数が大きくなるにつれ $N^3 \log N$ に近付くことが分かった。

ここで計算時間の勾配が $N^3 \log N$に近くなるのは、プロセッサ間の通信時間の増大によるものではなく、計算機システム全体でのメモリやプロセス管理のメカニズム等によるものではないかと考えられる。これを正確にモデル化するためには、VPP500 システム内部のより詳しいハードウェア、ソフトウェア等の状況を考慮に入れてモデルを作成する必要がある。

```
          a1(i,j,k)=b1(i,j,k)+c1(i,j,k)
100 continue
```

この場合の計算時間を Table 4.2.2 に、また並列化効率のグラフを、ロードバランスから直接計算した並列化効率の値とともに Fig.4.2.3 に示す。単純な、通信を伴わない計算の場合にはこのようにロードバランスが直接並列化効率に反映された値と非常に近い結果になる。しかし、実際のコード内でこのような計算が行なわれる場合には前述のモード数対計算時間の解析で述べたのと同じく、この部分だけの計算時間の評価では全体のパフォーマンスは評価できないものと思われ、これについても正しく評価するにはさらに測定方法を検討する必要がある。

4.3 等方性乱流シミュレーション

この並列化コードを用いて、初期速度場としてガウス分布でエネルギースペクトル分布が $k^4 \exp(-2k^2)$ に比例する速度場を与え、低スペクトル領域でエネルギーを与え続けて定常状態になるまで計算した。結果を Fig.4.3.1 に示す。

計算に用いたパラメータは、動粘度 $= 0.005[m^2/s]$、$\Delta\, t = 0.005[sec]$ である。図より、計算結果はほぼ Kolgomorov の $-5/3$ 乗則に従っていることがわかる。

5. まとめ

本報告では、等方性乱流シミュレーションコードを並列ベクトル計算機 VPP500 上で評価した結果について示した。3次元の高速フーリエ変換にかかる計算時間は、ベクトルシリアル計算の場合 $N^2 \log N$ に比例するが、ベクトルパラレル計算の場合、フーリエモード数が小さい時には $N \log N$ で、モード数が大きくなるにつれ $N^3 \log N$ に近付くことが分かった。

ここで計算時間の勾配が $N^3 \log N$に近くなるのは、プロセッサ間の通信時間の増大によるものではなく、計算機システム全体でのメモリやプロセス管理のメカニズム等によるものではないかと考えられる。これを正確にモデル化するためには、VPP500 システム内部のより詳しいハードウェア、ソフトウェア等の状況を考慮に入れてモデルを作成する必要がある。

参考文献

[1] R.B. Pelz : Journal of Computational Physics, 92, 296 (1993)

[2] 横川三津夫, 蕪木英雄：ハイパフォーマンスコンピューティング, 46-7, 37, (1993).

[3] S. A. Orszag : Studies in Applied Mathematics, L-4, (1971).

[4] S. A. Orszag : Phys. REev. Lett, 28-2,76, (1972).

[5] K. Yamamoto : Proc. Parallel CFD '94, ed by A., Ecer et al, Elsevier, Science, (1994).

Table 4.1.1 Fourier mode number vs. execution time (sec).

mode	scalor-serial	vector-serial	vector-parallel (PE=16)
2^4	7.9601	2.9087	0.9094
2^5	84.1470	13.2804	1.8733
2^6	1630.21	67.3152	5.5997
2^7	—	—	38.2906
2^8	—	—	269.6676

Table 4.2.1 Processor number and execution time (sec).

mode=64^3, mode=128^3

PE number	mode=64^3	mode=128^3
1	67.3556	—
2	35.0434	271.1112
4	18.7385	136.6654
8	10.2951	71.2271
16	5.6072	38.6500

Table 4.2.2 Processor number and execution time (sec) for test program.

mode=64^3, mode=128^3

PE number	mode=64^3	mode=128^3
PE=1	1058777.0	2108919.6
PE=2	529214.1	1059169.1
PE=4	291635.6	529319.4
PE=8	152305.2	282256.6
PE=16	79020.1	152172.7

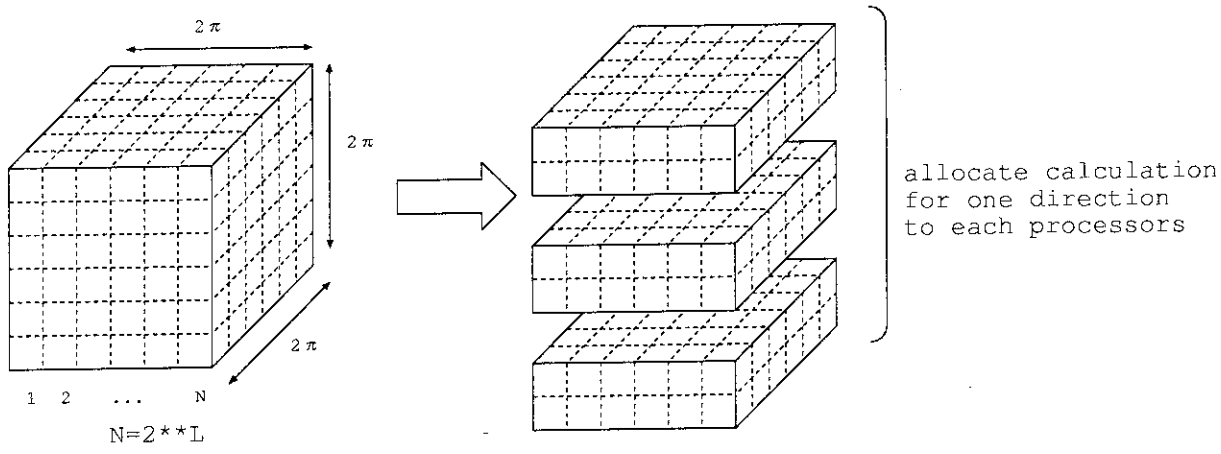

Fig.3.1 Allocation of meshes to processors

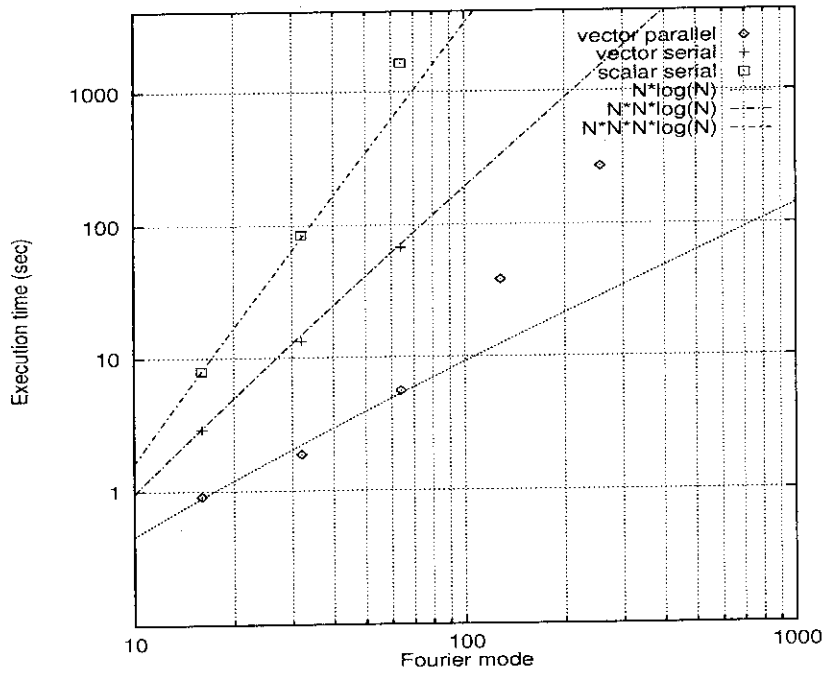

Fig.4.1.1 Execution time vs. Fourier mode using 16 processors

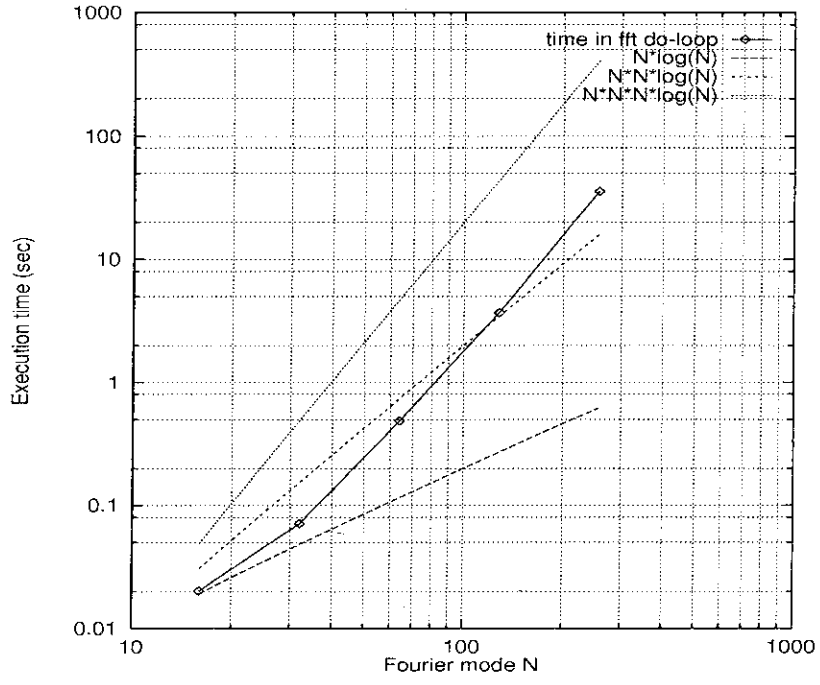

Fig.4.1.2 Execution time in FFT do loop vs. Fourier mode using 16 processors

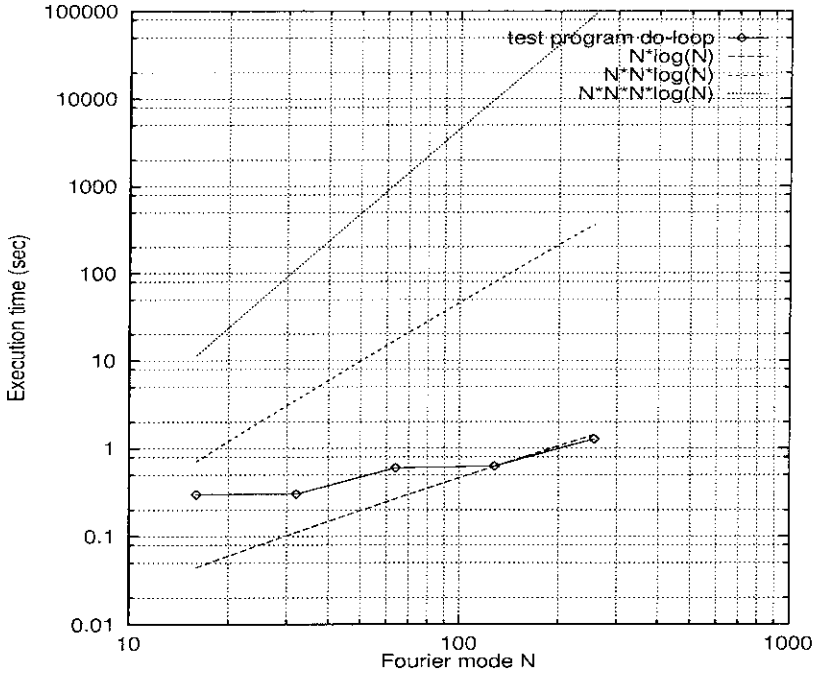

Fig.4.1.3 Execution time in model program vs. Fourier mode using 16 processors

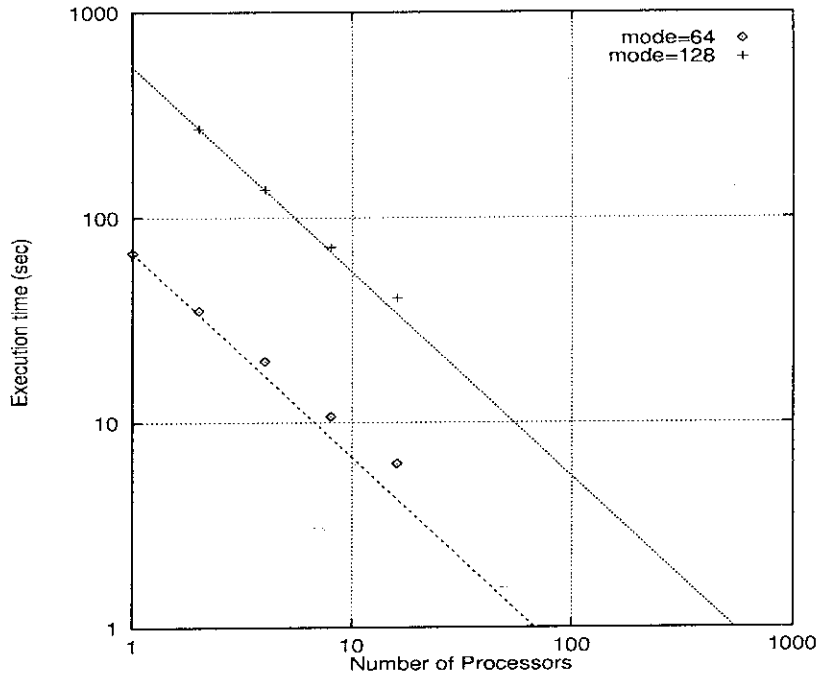

Fig.4.2.1 Processor number vs. Execution time

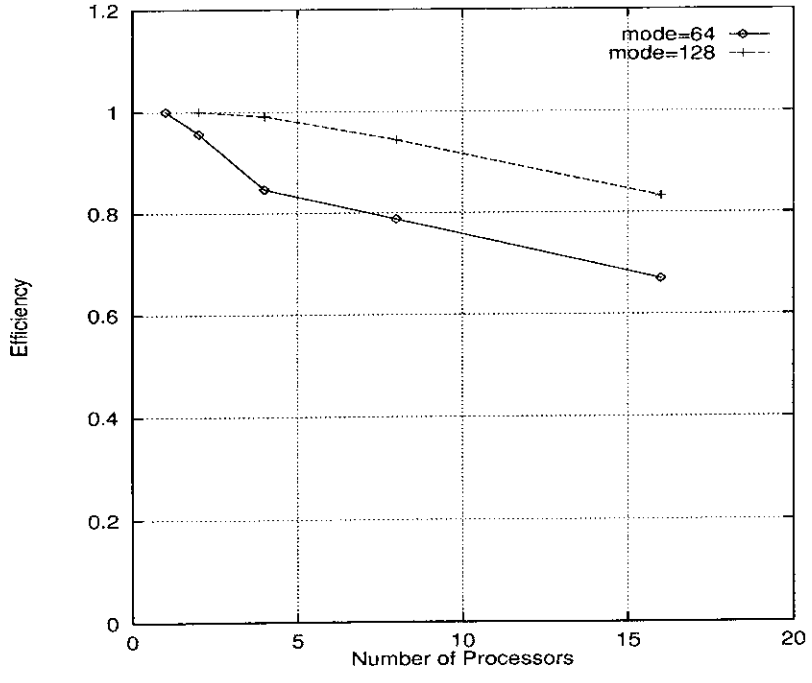

Fig.4.2.2 Processor number vs. Effeciency

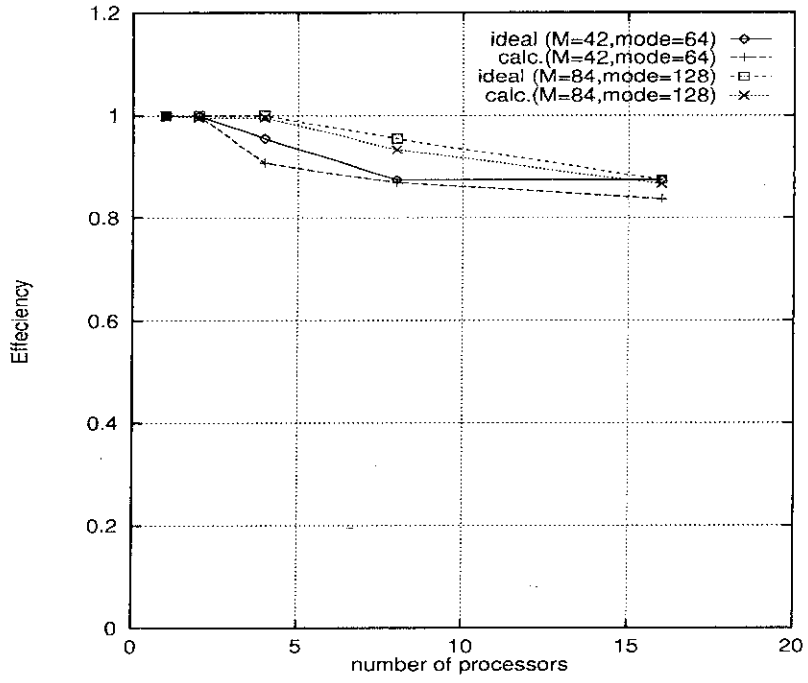

Fig.4.2.3 Effeciency for simple calculation

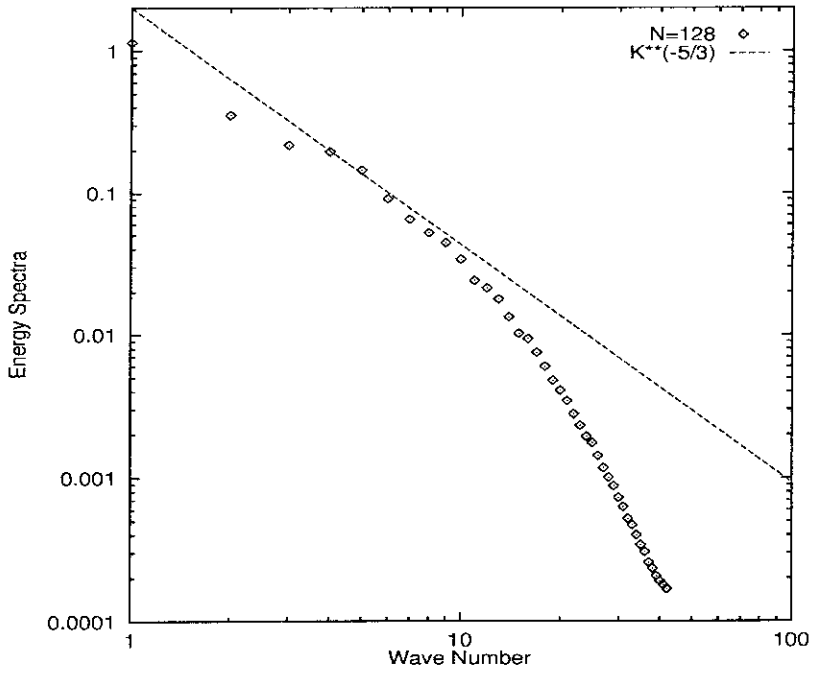

Fig.4.3.1 Energy spectrum

付録1　プログラム説明

今回の解析で用いたプログラムは、初期速度場を作成してファイルに出力するプログラム setfl0 と、この初期速度場のファイルを読み込んでナビエ・ストークス方程式を FFT で解いて時間発展をルンゲ・クッタ法により計算するプログラム trans5 の２つのプログラムから構成される。以下にそれぞれのプログラムのサブルーチン構成及びサブルーチンの簡単な説明を示す。

付録1.1　サブルーチン説明

付録1.1.1　初期速度場設定プログラム

```
setfl0 --- predin
       +- mkcnst
       +- vdft3i
       +- setfld --- makvel --- setvel
                  |          +- tospc1 --- vft3f1
                  |          +- tospc2 --- vft3f2
                  |          |          +- gtksp3
                  |          +- uninit
                  |          +- ranfld --- nrdist --- time
                  +- makvor
                  +- putfld
```

predin: 計算に必要なパラメータ（エネルギー係数、フーリエモード等）の入力
makvel: 入力指定によりいろいろな初期速度場を作成する
makvor: 渦度を速度から計算する
putfld: 計算した初期渦度場をファイルに出力する

付録 1.1.2　　FFT 及び時間発展計算プログラム

```
trans5 --- datain
       +- getfld
       +- mkcnst
       +- vdft3i
       +- mklist
       |   ........................... do  1000  loop1 = loopst, looped ....
       +- printl                                                            .
       |   ........................... do  2000  loop2 = 1, loopi ....      .
       +- calcfn --- om2pv0                                                 .      .
       |          +- omg2pv --- putksp                                      .      .
       |          |           +- vft3b1                                     .      .
       |          +- vft3b2                                                 .      .
       |          +- convl1 --- vft3f1                                      .      .
       |          +- convl2 --- vft3f2                                      .      .
       |                     +- gtksp2                                      .      .
       +- addvar                                                            .      .
       |   ..................................................................      .
       +- statis --- stat01 --- putksp                                             .
       |          |           +- vft3b1                                           .
       |          |           +- vft3b2                                           .
       |          +- omg2uv                                                       .
       |          +- stat02 --- putksp                                           .
       |          |           +- vft3b1                                           .
       |          |           +- vft3b2                                           .
       |          +- stat03                                                       .
       |   ..........................................................................
       +- putfld
```

datain: 計算に必要なパラメータ（動粘度、タイムステップ等）の入力
getfld: 初期渦度場のファイルからの読み込み
calcfn: FFT によりフーリエ係数の計算
omg2pv: スペクトル空間の渦度を物理空間の速度に逆変換
convl1, convl2 : 速度の非線形項のフーリエ係数を計算
addvar: ルンゲ・クッタ法で時間積分を計算
statis: 最大渦エネルギーや、エネルギースペクトルを計算して出力
putfld: 計算結果をリスタート計算のためにファイルに出力

付録 1.2　　入力データ説明

付録 1.2.1　　初期速度場設定プログラム setfl0 の入力データ

setfl0 では、下に示す入力データの他に、使用プロセッサ数とフーリエモード数（2の冪乗で指定)、有効モード数の指定をプログラム中の parameter 文により行なう。

フーリエモード 2^7、有効モード数 84 の場合は、setfl0.f の

```
PARAMETER  ( ML=7, MK=84 )
```

及び インクルードファイル params の

```
PARAMETER  ( NL=7, NK=84 )
```

の部分で、またプロセッサ数 4 の場合はインクルードファイル PROC の

```
parameter (mpe=4 )
```

の部分で指定する。これらは入力ファイルのデータのフーリエモード数と矛盾してはならない。

- データフォーマット

  レコード 1 ： nsize, kind, e0, header

  フォーマット ： 2i5,f10.0,5x,a32

  nsize ： フーリエモード数

  kind ： 初期速度場選択オプションなお、現在使用できるオプションは kind=4 のエネルギースペクトルが $k^4 \exp^{-2k^2}$ の等方性乱流のみである。

  e0 ： エネルギー係数

  header ： 初期速度場出力ファイルヘッダ名

- 入力データ例

  以下の入力データでは、フーリエモード数 128、初期速度場はエネルギースペクトルが $k^4 \exp^{-2k^2}$ の等方性乱流、エネルギー係数 0.1 で、出力ファイルを./result/というディレクトリ内に 00000 という名前で作成する。

```
128    4        0.1      ./result/
```

付録 1.2.2　FFT 及び時間発展計算プログラム trans5 の入力データ

trans5 では、下に示す入力データの他に、使用プロセッサ数とフーリエモード数（２の冪乗で指定）、3/2 則で用いられる有効モード数の指定をプログラム中の parameter 文により行なう。

フーリエモード数 2^7、有効モード数 84 の場合は、trans5.f の

```
PARAMETER  ( ML=7, MK=84 )
```

及び インクルードファイル params の

```
PARAMETER  ( NL=7, NK=84 )
```

の部分で、また使用プロセッサ数 4 の場合はインクルードファイル PROC の

```
parameter (mpe=4 )
```

の部分で指定する。これらは入力ファイルデータや、setfl0 で初期乱流を計算するのに用いたフーリエモード数と矛盾してはならない。

- データフォーマット

  レコード 1 ： nsize, dt, rnu, loopi, loopst, looped, lopinc, lopfin, maxcpu, maxrgn

  フォーマット ： i5,2f10.0,7i5

  レコード 2 ： header

  フォーマット ： a32

  nsize ： フーリエモード数

  dt ： 時間ステップ Δ t

  rnu ： 動粘度

  loopi ： 内側の時間ステップループの繰り返し回数

  loopst ： 外側の時間ステップループの初期値

  looped ： 外側の時間ステップループの最終値

  lopinc , lopfin , maxcpu , maxrgn ： 未使用

  header ： 初期速度場入-出力ファイルヘッダ名

- 入力データ例

  以下の入力データでは、フーリエモード数 128、動粘度 0.005、時間ステップ 0.005 で、500 回×5 回 = 2500 回の時間ステップの計算を行なう場合の入力データ初期速度場は./result/というディレクトリの 00000 というファイルから読み込まれ、2500 ステップの計算を行なった結果が./result/00005 というファイルに出力される。途中の計算結果（エネルギースペクトル等）は 500 ステップ毎にリスト出力される。

```
  128      0.005      0.005  500     1     5     1     1     1    10    20
./result/
```

付録2　　ロードバランスと並列化効果についての考察

ロードバランスと並列化効率の関係を調べるため、スカラ変数の和を計算するプログラムと、ベクトル変数の和を計算するプログラムを作成し、両者の並列化効率を比較した。

ベクトル変数の和

```
00000025          c--------------------------
00000026                call clock(xxx1,2,2)
00000027                call gettod(ttt(2))
00000028          c--------------------------
00000029          c
00000030          !xocl spread do /ipn
00000031       s2       do 100 j=1,ll
00000032       v2         do 100 i=1,i1
00000033       v2             x1 = x1 + 1
00000034       v2   100 continue
00000035          !xocl end spread
00000036          c
00000037          c--------------------------
00000038                call clock(xxx2,2,2)
00000039                call gettod(ttt(3))
00000040          c--------------------------
```

スカラ変数の和

```
00000025          c--------------------------
00000026                call clock(xxx1,2,2)
00000027                call gettod(ttt(2))
00000028          c--------------------------
00000029          c
00000030          !xocl spread do /ipn
00000031       s2       do 100 j=1,ll
00000032       v2         do 100 i=1,i1
00000033       v2             a1(j) = a1(j) + 1
00000034       v2   100 continue
00000035          !xocl end spread
00000036          c
00000037          c--------------------------
00000038                call clock(xxx2,2,2)
00000039                call gettod(ttt(3))
00000040          c--------------------------
```

用いたパラメータは ll=50 であり、i1 を変えて実行時間を調べた。

表 a.1 スカラ変数の和の実行時間（単位：マイクロ秒）及び並列化効率

	PE=1 実行時間	PE=1 並列化効率	PE=4 実行時間	PE=4 並列化効率
(1) i1=1×10^7	10086.0	1.0	2318.5	1.088
(2) i1=9×10^7	88351.5	1.0	19664.1	1.123
理論値	—	1.0	—	0.9615

同じ計算を、後日ふたたび繰り返した結果が、下表である。

表 a.2 スカラ変数の和の実行時間（単位：マイクロ秒）及び並列化効率

	PE=1 実行時間	PE=1 並列化効率	PE=4 実行時間	PE=4 並列化効率
(1) i1=1×10^7	7464.43	1.0	2388.31	0.781
(2) i1=9×10^7	66234.0	1.0	19704.8	0.840
(3) i1=100×10^7	734038.5	1.0	215363.5	0.850
理論値	—	1.0	—	0.9615

表 a.3 ベクトル変数の和の実行時間（単位：マイクロ秒）及び並列化効率

	PE=1 実行時間	PE=1 並列化効率	PE=4 実行時間	PE=4 並列化効率
(1) i1=1×10^7	54065.1	1.0	14180.6	0.953
(2) i1=9×10^7	484465.8	1.0	126240.4	0.959
理論値	—	1.0	—	0.9615

次に、ベクトル変数の和について、同様に実行時間及び並列化効率を測定した。
後日ふたたび繰り返した結果が、下表である。

表 a.4 ベクトル変数の和の実行時間（単位：マイクロ秒）及び並列化効率

	PE=1 実行時間	PE=1 並列化効率	PE=4 実行時間	PE=4 並列化効率
(1) i1=1×10^7	54136.3	1.0	14253.2	0.950
(2) i1=9×10^7	434338.6	1.0	126603.1	0.956
(3) i1=100×10^7	5379577.2	1.0	1398870.4	0.961
理論値	—	1.0	—	0.9615

この並列化効率の理論値は、ループ数 (=50) を 4 個の PE に割り当てた場合に 13,13,13,11 という割り当てられ方をするため、これに基づいて

$$\text{理論値} = \frac{50}{13 \times 4} = 0.9615$$

により計算した。

このように、ベクトル変数についての和を計算する際には、並列化効率はロードバランスが直接反映されたものになるが、スカラ変数をもちいた場合には PE のロードバランスは並列化効率には影響を与えないように見える。

さらに、ベクトル変数を用いた計算では、何度同じ計算を行なっても並列化効率はが安定しているが、スカラ変数の方は実行するたびに並列化効率が大きく変わることがわかる。これは gettod で経過時間を測っても、 clock で cpu 時間を測ってもどちらもほぼ同じ傾向を示した。

この理由としては、プログラムが実行された際のシステム全体の混み具合や、計算へのシステム時間やユーザ時間の割り当てられかたに原因があると考えられるが、詳しいことはまだ解明できていない。

いずれにしろ、並列化効率や実行速度等の評価を行なう場合には、このようにシステムの状況により計算時間が左右されるという要素を考慮に入れる必要がある。