SPH 熱対流プログラムの説明書

菖蒲迫 健介 (九州大学) [*1]
(2023/06/27)

■ はじめに

プログラムの中身がブラックボックスでも良い人は，1 節のみご覧ください．プログラムを書き換えたい人やコードを理解したい人は，2 節も続けてお読みください．

■ 改訂

- 2022 年 12 月：プログラムの大枠が完成
- 2023 年 06 月：解析プログラムを自動化

目次

[*1] 九州大学大学院理学府 地球惑星科学専攻 地球内部ダイナミクス研究室 博士 1 年
E-mail: shobuzako.kensuke.242@s.kyushu-u.ac.jp

1 プログラムの概要

1.1 基本的なこと

■ 実装内容

◦ 計算系

- 弱圧縮性 SPH 法 (weakly-compressible SPH method; Monaghan, 1994) をベースとした 1 成分系 2 次元熱対流プログラム
- 下壁は固定壁，上壁は固定壁もしくは自由表面，横壁は固定壁もしくは周期境界を選択可能
- 任意の状態方程式 ($p = p(\rho, T)$) を入力することで，ブシネスク系以外の計算も可能
- 弱圧縮性近似 [e.g., Chorin, 1967; Rempel, 2005] による陽解法計算
- 慣性変化法 [Takeyama et al., 2017] によりプラントル数が大きい計算も可能
- OpenMP によるノード内並列計算を実装

◦ SPH のテクニカルな部分

- 粒子再配列操作として，PS 法 [e.g., Xu et al., 2009; Huang et al., 2018] と OPS 法 [Khayyer et al., 2017] を使用可能
- カーネル関数として，5 次スプライン関数と Wendland kernel C^4, C^6(Wendland, 1995) を標準実装 (追加実装可)
- 高精度微分演算モデルは実装なし
- 圧力勾配項には和モデル [浅井, 2022]，粘性項には Morris formula[Morris et al., 1997] を使用
- 壁計算には仮想マーカー [Marrone et al., 2011; Asai et al., 2013] を使用
- 重力密度分離法 [菖蒲迫修論, 2022] により SPH 密度と浮力に関する密度を分離

■ プログラムの在処 (Github)

https://github.com/shobuzako-kensuke/GDS-VIM-WCSPH-convection

■ 必要な環境

- OpenMP が使用可能な gfortran
- Python(numpy, matplotlib 等の基本的なライブラリーも使用) の標準環境
- Make ファイルの実行環境

■ その他

- 計算には fortran90 を使用し，解析には Python を使用

1.2 計算プログラムの使い方 (program_src ディレクトリ)

■ 手順

基本的な手順は以下である.

計算プログラムを動かす手順

(1) program_src/input.f90 で計算したい系を選択[a]
(2) ターミナルを開き，program_src まで移動して，`make` コマンドを実行
(3) `[Message] Makefile has been made.` と表示されたら OK
(4) 同じディレクトリに作成された `start_calculation.exe` を実行
(5) `[Message] Your calculation has been completed !!`と出力されれば，終了

[a] 付属の Excel ファイルで緩和パラメーター ζ, ξ を計算.

■ 計算パラメーターの入力 (input.f90 ファイル)

input.f90 では以下のようなパラメーターを入力する (表 1.1，図 1.1)．なお，緩和パラメーターや係数の説明は菖蒲迫修論 (2022) を参考にされたい[*2].

表 1.1: input.f90 で入力するパラメーター

引数名	説明	備考
`file_name`	ファイル名	アルファベット＋数字 3 つで指定
`RSST_VIM`	VIM の有無	0：RSST のみ，1：RSST + VIM
`num_threads`	スレッド数	OpenMP のスレッド数
`total_step`	計算ステップ数	–
`write_step`	書き出し間隔	指定ステップ数毎に書き出し
`coe_step`	タイムステップ係数 ϕ_t	CFL 条件等の係数 ($\Delta t = \phi_t \frac{\Delta x}{c}$)
`Ra_0`	系全体のレイリー数	次元付きの量を暗に指定
`num`	1 辺の粒子数	–
`zeta`	ζ	RSST の緩和パラメーター
`xi`	ξ	VIM の緩和パラメーター
`num_MP_side`	1 辺のマッピング粒子数	デフォルトは 50
`kernel_switch`	カーネル関数の選択	0：5 次スプライン，1：C^4，2：C^6
`coe_smooth`	ϕ_h	smoothing length の係数 ($h = \phi_h \Delta x$)
`wall_v`	滑り条件	0：no-slip，1：free-slip

RSST 系には空気対流 ($Pr = 0.71$)，RSST-VIM 系にはマントル対流 ($Pr \approx \infty$) を実装してある.

[*2] https://github.com/shobuzako-kensuke/Material-Storage/tree/main/01_gakui_ronbun

```
!=============================================================================================!
!                           thermal convection using GDS-WCSPH                                !
!---------------------------------------------------------------------------------------------!
!                           Copyright by Kensuke Shobuzako (2022)                             !
!=============================================================================================!

module INPUT

    implicit none

    !>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>!
    !                                    INPUT below                                            !
    !>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>!
    !================================== system setting ===================================!
    ! NAME
    character(99), parameter :: file_name = 'R100'       ! file I.D.
    ! RSST_VIM system
    integer, parameter :: RSST_VIM        = 0            ! 0(air), 1(mantle)
    ! the number of threads(cores)
    integer, parameter :: num_threads     = 8            ! the number of threads
    ! STEPS
    integer, parameter :: total_step      = 500          ! total steps
    integer, parameter :: write_step      = 10           ! writing step
    real(8), parameter :: coe_step        = 0.3d0        ! coefficient of time step
    ! SYSTEM INFO
    real(8), parameter :: Ra_0            = 1.0d+4       ! Ra_0 number
    integer, parameter :: num             = 50           ! the number of particles in an edge
    ! RELAXATION PARAMETER
    real(8), parameter :: zeta            = 10800.d0     ! relaxation parameter (RSST)
    real(8), parameter :: xi              = 1.0d0        ! relaxation parameter (VIM)
    ! the number of analysis point (default:50)
    integer, parameter :: num_MP_side     = 100          ! the number of MP in an edge
    !================================== kernel function ==================================!
    integer, parameter :: kernel_switch   = 2            ! 0(Quintic), 1(WenC4), 2(WenC6)
    real(8), parameter :: coe_smooth      = 2.3d0        ! coefficient of smoothing length
    !================================ boudary condition ==================================!
    integer, parameter :: wall_v          = 0            ! 0(no-slip), 1(free-slip)
    !>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>!
    !                                    INPUT end                                              !
    !>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>!

```

図 1.1: input.f90 の入力画面.

1.3　解析ファイルの使い方 (analysis_src ディレクトリ)

analysis_src/input.py で動かしたい解析プログラムを選択し，パラメーターを入力する (図 1.2). 図は fig ディレクトリに保存される．input.py の説明は以下の通り．

- 全部で 3 つの解析プログラムを用意 (「1」で起動)
  (1) `do_plot_movie_material`：静止画を出力
  (2) `do_plot_analysis`：物理量の分布，時間変化やヌッセルト数を描画
  (3) `do_plot_initial_state`：初期条件を出力

- `num_save_file`：解析したいファイル番号を指定
  【注意】初期条件も考慮に入れること

- `plot_〇〇〇`：静止画のタイプを指定 (「1」で起動)

- `plot_phy`：静止画で描く物理量を指定 (デフォルトは温度の「6」)

```
#>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>#
#                           INPUT below                               #
#>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>#

# if exe, put 1
do_plot_movie_material = 1
do_plot_analysis       = 1
do_plot_initial_state  = 1

# input save file number you want to analyse
# see in ./../save_file/V_rms_save*.dat
# [note] put (the number - 1)
num_save_file = 50

#======================
# for movie materials
#======================
# movie switch (if exe, put 1)
plot_without_wall     = 1
plot_with_wall        = 0
plot_all_without_wall = 0
plot_all_with_wall    = 0
plot_PS_shift         = 0

# select physical quantity scattered in movie_materials
# (2,3,4,5,6,7,8,9) = (u,v,rho_S, pre, tem, eta, rho_sm, rho_G)
plot_phy = 6

#<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<#
#                           INPUT end                                 #
#<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<#
```

図 1.2: input.py の入力画面.

2 プログラムの中身

プログラムの中身を知りたい人は本節も読まれたい．

2.1 全体の流れ (program_src/main.f90)

プログラムのアルゴリズムは以下である．main.f90 内で使用するモジュールを表 2.1 に示した．

(1) 計算システムの確定 (粒子数/カーネル関数/空気対流 or マントル対流など)
(2) inner 粒子，wall 粒子，virtual marker 粒子，mapping 粒子 (解析粒子) の設定
(3) inner 粒子の初期条件を入力
(4) バックグラウンドセルの設定 (近傍 9 つのセル番号を確定)
(5) 近傍粒子探索 (NNPS[*3]) の実行
(6) virtual marker を用いて，初期の壁情報を確定
(7) 初期条件の書き出し
(8) 時間ループの開始
(9) 連続の式，運動方程式，温度の時間発展式の右辺を計算
(10) SPH 密度/速度/温度を更新 (時間積分には 2 次のアダムス・バッシュフォース法を使用)
(11) 位置を更新 (もしも，壁を越えていたら完全弾性衝突で跳ね返す)
(12) NNPS の実行 (所属セル番号の更新)
(13) 人工的な状態方程式および本当の状態方程式から，圧力と重力密度を決定
(14)
- PS 法を使う場合 … 壁情報を更新し，PS 法を実行．さらに，NNPS の再実行
- PS 法を使わない場合 … スキップ (手動による切り替えが必要)

(15) 壁情報を更新
(16) 書き出しを行う場合は，ここで実行
(17) 時間ループの終了

2.2 主な使用配列

基本的に glb_arg.f90 内で配列を定義し，initial_setting.f90 で割り付けを行うことにしている．粒子配列の大きな分類は以下である．

- `SP`：inner 粒子と wall 粒子
- `VM`：vertual marker 粒子
- `MP`：mapping 粒子

例えば，`SP_rho_S(:)` は inner 粒子と wall 粒子の SPH 密度を格納する配列である．二次元目の引数があるものは，x, y 方向の意味である．その他の主な配列の説明は，表 2.2 の通りである．

[*3] Nearest Neighbor Particle Searching.

表 2.1: モジュールの説明 (全て f90 ファイル)

名前	説明
INPUT	input ファイル
GLB_SETTING	計算で用いるパラメーターの決定
GLB_ARG	割り付け前の配列を定義
cal_EOS	人工的な状態方程式と本当の状態方程式を記載
INITIAL_SETTING	配列の割り付けと初期条件の貼り付け
NNPS	近傍粒子検索に関するサブルーチンを収納
cal_KERNEL	カーネル関数とその微分に関するサブルーチンを収納
cal_WALL	vertual marker 粒子および wall 粒子の計算
set_time_step	実効的なタイムステップを決定
cal_main	連続の式/運動方程式/温度の時間発展式/時間積分等を実行
PS_scheme	PS 法を実行
cal_mapping	mapping 粒子の計算
analysis	今のところ $V_{\rm rms}$ を計算
WRITE_OUT	出力 (書き出し) に関するサブルーチンを収納

表 2.2: 配列の説明

名前	1 次元目の引数	2 次元目の引数	説明
SP_kind(:)	粒子属性	-	0(inner), 1(bot), 2(top), 3(right), 4(left)
NNPS_arg(:,:)	粒子の ID	所属セル番号	バックグラウンドセル配列
NNPS_9(:,:)	近傍セル番号	所属セル番号	近傍セル番号の記憶配列
ID_VM_WL(:)	VM の ID	-	各 VM 粒子に対する wall 粒子番号を記憶
NNPS_arg_VM(:,:)	VM の ID	所属セル番号	VM 専用のバックグラウンドセル配列

2.3 その他のプログラム

以下のような便利なプログラムを用意した.

- delete.py：取得データ，描画した図などを再帰的に削除する (ダウンロード時の状態に戻す)
- program_src/main_time_stamp.f90：各モジュールの計算時間を測定[*4]

[*4] 使用するにはプログラムの書き換えが必要.

謝辞

本ノートは，九州大学 大学院理学府 地球惑星科学専攻 地球内部ダイナミクス研究室が主催する「初期地球勉強会」の議論をもとに作成しています．勉強会では，指導教員の吉田 茂生准教授をはじめ，川田 佳史さん (@国立研究法人 海洋研究開発機構 海洋機能利用部門 海底資源センター) と，中島 涼輔さん (@本学理学研究院) らに有意義なご意見を賜りました．また，本ノートの作成にあたっては，川田 佳史さんの LaTeX テンプレートを使用させて頂きました．深く感謝申し上げます．

参考文献

■ 論文

[1] Asai, M., Fujimoto, K., Tanabe, S., Beppu, M. (2013) Slip and No-Slip Boundary Treatment for Particle Simulation Model with Incompatible Step-Shaped Boundaries by Using a Virtual Maker, *Transactions of the Japan Society for Computational Engineering and Science*, **2013**

[2] Chorin, A. J. (1967) A numerical method for solving incompressible viscous flow problems, *Journal of Computational Physics*, **2**, 1, 12–26

[3] Hosono, N., Saitoh, T. R., Makino, J. (2013) Density-Independent Smoothed Particle Hydrodynamics for a Non-Ideal Equation of State, *Publications of the Astronomical Society of Japan*, **65**, 5, 108

[4] Huang, C., Zhang, D. H., Shi, Y. X., Si, Y. L., Huang, B. (2018) Coupled finite particle method with a modified particle shifting technology, *International Journal for Numerical Methods in Engineering*, **113**, 2, 179–207

[5] Khayyer, A., Gotoh, H., Shimizu, Y. (2017) Comparative study on accuracy and conservation properties of two particle regularization schemes and proposal of an optimized particle shifting scheme in ISPH context, *Journal of Computational Physics*, **332**, 236–256

[6] Marrone, S., Antuono, M., Colagrossi, A., Colicchio, G., Le Touzé, D., Graziani, G., (2011) δ-SPH model for simulating violent impact flows, *Computer Methods in Applied Mechanics and Engineering*, **200**, 13–16, 1526–1542

[7] Monaghan, J.J. (1994) Simulating Free Surface Flows with SPH, *Journal of Computational Physics*, **110**, 2, 399–406

[8] Morris, J. P., Fox, P. J., Zhu, Y. (1997) Modeling Low Reynolds Number Incompressible Flows Using SPH, *Journal of Computational Physics*, **136**, 214–226

[9] Rempel, M. (2005) Solar Differential Rotation and Meridional Flow: The Role of a Subadiabatic Tachocline for the Taylor - Proudman Balance, *The Astrophysical Journal*, **622**, 2, 1320–1332

[10] Saitoh, T. R., Makino, J. (2013) A Density-Independent Formulation of Smoothed Particle Hydrodynamics, *The Astrophysical Journal*, **768**, 1, 44

[11] Takeyama, K., Saitoh, T. R., Makino, J. (2017) Variable inertia method: A novel numer-

ical method for mantle convection simulation, *New Astronomy*, **50**, 82–103

[12] Wendland, H. (1995) Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree, *Advances in Computational Mathematics*, **4**, 1, 389–396

[13] Xu, R., Stansby, P., Laurence, D. (2009) Accuracy and stability in incompressible SPH (ISPH) based on the projection method and a new approach, *Journal of computational Physics*, **228**, 18, 6703–6725

■ 本

[14] Turcotte, D., Schubert, G. 原著, 安藤亮輔, 岩森光, 沖野郷子, 片山郁夫, 加納靖之, 川田佳史, 木下正高, 坂口有人, 田中愛幸, 中西正男, 西山竜一, 山野誠, 吉田晶樹 訳 (2020) 『ジオダイナミクス』, 共立出版

[15] 浅井 光輝 (2022) 明解粒子法：SPH,MPS,DEM の理論と実践, 丸善出版

[16] 岩下 武史, 片桐 孝洋, 高橋 大介 (2015) スパコンを知る：その基礎から最新の動向まで, 東京大学出版会

[17] 牛島 省 (2006) OpenMP による並列プログラミングと数値計算法, 第 5 刷, 丸善出版

[18] 牛島 省 (2007) 数値計算のための Fortran90/95 プログラミング入門, 森北出版

[19] 木田重雄, 柳瀬真一郎 (1999) 『乱流力学』, 朝倉書店

[20] 後藤 仁志 (2018) 粒子法：連続体・混相流・粒状体のための計算科学, 森北出版

■ ネット資料

[21] 片桐 孝洋 (2019) 第 1 回 プログラムの高速化の基礎, 名古屋大学情報基盤センター
https://www.r-ccs.riken.jp/wp/wp-content/uploads/2020/09/katagiri190411.pdf

[22] 片桐 孝洋 (2019) 第 2 回 並列処理と MPI の基礎, 名古屋大学情報基盤センター
https://www.cc.u-tokyo.ac.jp/events/lectures/X01/shiryou-2.pdf

[23] 片桐 孝洋 (2019) 第 3 回 OpenMP の基礎, 名古屋大学情報基盤センター
https://www.r-ccs.riken.jp/wp/wp-content/uploads/2020/09/katagiri190425.pdf

[24] スーパーコンピュータ超入門講習会, 九州大学情報基盤研究開発センター
https://www.cc.kyushu-u.ac.jp/scp/doc/users/lecture/2019/superintro-2019-1.pd

付録 A　プログラムの並列効率

A.1　並列効率の測り方

並列効率を表現する最も一般的な指標は，**スピードアップ** (speed up) と呼ばれるものである．

$$S \equiv \frac{T_s(n)}{T_p(n)} \tag{A.1}$$

ここで，S はスピードアップ，$T_s(n)$ は逐次計算に要する計算時間，$T_p(n)$ は並列計算に要する計算時間である．大きさが n の問題を p 個のプロセッサで解くことを考える．この問題において，並列化が可能な部分にかかる計算時間を $t_p(n)$ とし，並列化が不可能な部分にかかる計算時間を $t_s(n)$ とする．この時，逐次計算を行う場合では

$$T_s(n) = t_s(n) + t_p(n) \tag{A.2}$$

である．一方，並列計算を行う場合は，理想的には

$$T_p(n) = t_s(n) + \frac{t_p(n)}{p} \tag{A.3}$$

であるが，実際には通信や並列化のための新たな**オーバーヘッド** $t_o(n,p)$ が発生するので

$$T_p(n) = t_s(n) + \frac{t_p(n)}{p} + t_o(n,p) \tag{A.4}$$

と期待される．これらを用いると，スピードアップ S は

$$S = \frac{t_s(n) + t_p(n)}{t_s(n) + t_p(n)/p + t_o(n,p)} \tag{A.5}$$

と表現される．n を固定した問題においては p の増加とともに p 倍のスピードアップが望まれるが，同時にオーバーヘッド $t_o(n,p)$ も増大するため，実際の並列計算では，ある p を超え始めると S が減少する傾向にある．

ここで，**並列効率** ε はスピードアップをスレッド数で割ったものとして定義される．

$$\varepsilon \equiv \frac{S}{p} \leq \frac{t_s(n) + t_p(n)}{pt_s(n) + t_p(n) + pt_o(n,p)} \tag{A.6}$$

最も理想的な状況はプログラム全体が並列化可能かつ $t_o(n,p) = 0$ の場合なので

$$\varepsilon = \frac{S}{p} = \frac{t_p(n)}{t_p(n)/p} \times \frac{1}{p} = 1 \tag{A.7}$$

となる．これはスレッド数と並列効率のプロットにおいて，傾き 1 の直線で表される状況が最大の並列効率であることを示す．通常は $t_s(n) \neq 0$ で $0 \leq t_o(n,p)$ なので，$0 \leq \varepsilon \leq 1$ となる．(A.6) 式から明らかであるが，逐次計算領域が大きく ($t_s(n)$ 大)，オーバーヘッドが大きい ($t_o(n,p)$ 大) システムでは効率が出ない．

■ アムダールの法則

全体の計算時間に対する逐次計算部分の割合を $f(n) = t_s(n)/(t_s(n) + t_p(n))$ とおく．この時，(A.5) 式において $0 \leq t_o(n,p)$ であることを考慮すると

$$S(n,p) \leq \frac{t_s(n) + t_p(n)}{t_s(n) + t_p(n)/p + t_o(n,p)} \leq \frac{t_s(n) + t_p(n)}{t_s(n) + t_p(n)/p} = \frac{1}{f(n) + (1 - f(n))/p} \tag{A.8}$$

という関係を得る．この式は**アムダールの法則** (Amdahl's law) と呼ばれる．この式は，p をどれだけ増やしたとしてもスピードアップの上限が $1/f(n)$ で制約されることを示しており，並列化に際しては逐次演算を少なくすること ($f(n)$ を小さくすること) が重要であると解釈される．

並列効率が 1 を超える状況がまれに生じることもある．これは各スレッドが扱う問題のサイズ小さくなると，計算機の一時参照メモリ (キャッシュメモリ等) に置かれた変数の使用率が高まることなどが原因と考えられる．この状況は，スレッド数と効率が傾き 1 の直線よりも上側にくるような状況であり，スーパーリニアスピードアップ (superlinear speedup) と呼ばれる．従って，配列サイズは小さい方が良いとされる．

A.2 自作プログラムの並列効率について

菖蒲迫の弱圧縮性 SPH 熱対流プログラムの並列効率を調べてみた．

■ 条件

- 使用マシン：Intel corei7–9700(3.00GHz) で最大 8 スレッド並列が可能
- 100 ステップ分の cpu time で比較 (3 回の平均値を採用)

■ 結果

- 結果を図 A.1 に示した (図の読み方はキャプションを参照)
- 粒子数が少ないと効率が出ない (図 (a),(b)) → 恐らくオーバーヘッド割合が大きい
- 100 から 200，400 への増加は 4 倍ずつ比例しているので，期待通り (図 (c),(d))[*5]
- 50×50 はスレッド数の増加と共に配列効率が鈍るので，これで規格化すると，効率が上がっているように見える (図 (c),(d))

■ 考察

- 50×50 程度だと効率が出ないが，100×100 以上だとそこそこ出る (並列効率 75% を達成)
- 逐次計算 (スレッド数 1) で 100×100 が 4 倍以下なので，粒子数に非依存な計算の負荷がそれなりに大きいかもしれない

[*5] 本当はタイムステップが小さくなった分だけ別にコストが増えるが，今回は 100 ステップ縛りなので関係なし．

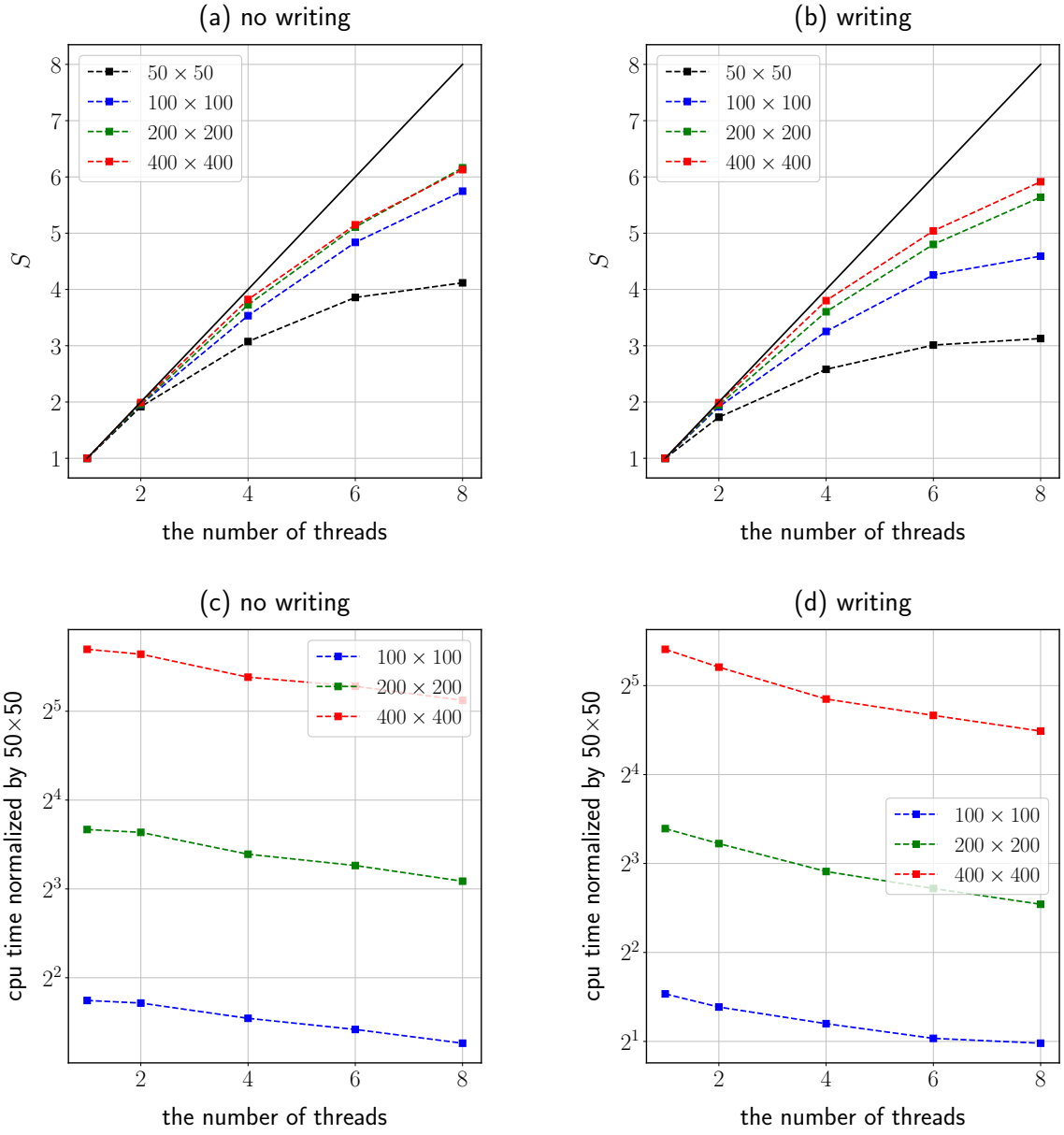

図 A.1: 並列効率について．横軸はスレッド数．(a) 書き出しを行わない場合のスピードアップの結果．(b) 毎ステップ書き出しを行った場合のスピードアップの結果．(c) 書き出しを行わない場合の cpu time．ただし，50 × 50 の結果で規格化した．(d) 毎ステップ書き出しを行った場合の cpu time．

A.3 並列効率の計算プログラム

```
#=============================================================================================#
#                           CPU time of thermal convection program                            #
#---------------------------------------------------------------------------------------------#
#                           Copyright by Kensuke Shobuzako (2023)                             #
#=============================================================================================#

#====================#
# charm
#====================#
import numpy as np
import matplotlib.pyplot as plt
import math
import glob
from scipy import integrate
from scipy.interpolate import interp1d
from matplotlib import rc
import sys
import os
import time
rc('text', usetex=True)

#====================#
# input
#====================#
file_name = 'CPU_time'
fig_switch = 1 # if exe, put 1

cpu_time = np.zeros((4, 5, 2)) # ((50,100,200,400), (1,2,4,6,8), (no_write, write))
# 50*50, no_write
cpu_time[0, 0, 0] = 10.57
cpu_time[0, 1, 0] = 5.516
cpu_time[0, 2, 0] = 3.438
cpu_time[0, 3, 0] = 2.740
cpu_time[0, 4, 0] = 2.567
# 50*50, write
cpu_time[0, 0, 1] = 13.07
cpu_time[0, 1, 1] = 7.556
cpu_time[0, 2, 1] = 5.065
cpu_time[0, 3, 1] = 4.340
cpu_time[0, 4, 1] = 4.179
# 100*100, no_write
cpu_time[1, 0, 0] = 35.41
cpu_time[1, 1, 0] = 18.11
cpu_time[1, 2, 0] = 10.02
cpu_time[1, 3, 0] = 7.321
cpu_time[1, 4, 0] = 6.162
# 100*100, write
cpu_time[1, 0, 1] = 37.84
cpu_time[1, 1, 1] = 19.75
cpu_time[1, 2, 1] = 11.63
cpu_time[1, 3, 1] = 8.887
cpu_time[1, 4, 1] = 8.241
# 200*200, no_write
cpu_time[2, 0, 0] = 134.4
cpu_time[2, 1, 0] = 68.59
cpu_time[2, 2, 0] = 36.06
cpu_time[2, 3, 0] = 26.31
cpu_time[2, 4, 0] = 21.81
# 200*200, write
cpu_time[2, 0, 1] = 137.2
cpu_time[2, 1, 1] = 70.59
cpu_time[2, 2, 1] = 38.07
cpu_time[2, 3, 1] = 28.58
cpu_time[2, 4, 1] = 24.33
# 400*400, no_write
cpu_time[3, 0, 0] = 548.4
cpu_time[3, 1, 0] = 275.5
cpu_time[3, 2, 0] = 143.5
cpu_time[3, 3, 0] = 106.5
cpu_time[3, 4, 0] = 89.47
# 400*400, write
cpu_time[3, 0, 1] = 555.0
cpu_time[3, 1, 1] = 279.2
cpu_time[3, 2, 1] = 146.0
cpu_time[3, 3, 1] = 110.1
cpu_time[3, 4, 1] = 93.84
```

```
77
78 num_thread = np.zeros((5))
79 num_thread = [1, 2, 4, 6, 8]
80
81 #====================#
82 # program
83 #====================#
84 #cpu_time_1 = time.perf_counter()
85 #cpu_time_2 = time.perf_counter()
86 #print('[Message] time : {:.2f} [s]'.format(cpu_time_1-cpu_time_2))
87
88 #====================#
89 # figure
90 #====================#
91 if (fig_switch==1):
92     # figure and axis environment
93     fig, axs = plt.subplots(2, 2, figsize=(16, 16), facecolor='white', subplot_kw={'
           facecolor':'white'})
94     # margin between figures
95     plt.subplots_adjust(left=0.12, right=0.92, bottom=0.098, top=0.92, wspace=0.4,
           hspace=0.35)
96     # plot
97     axs[0,0].plot(num_thread, cpu_time[0,0,0]/cpu_time[0,:,0], linestyle='--', marker='s
           ', c='k', label=r'$50\times50$')
98     axs[0,0].plot(num_thread, cpu_time[1,0,0]/cpu_time[1,:,0], linestyle='--', marker='s
           ', c='b', label=r'$100\times100$')
99     axs[0,0].plot(num_thread, cpu_time[2,0,0]/cpu_time[2,:,0], linestyle='--', marker='s
           ', c='g', label=r'$200\times200$')
100    axs[0,0].plot(num_thread, cpu_time[3,0,0]/cpu_time[3,:,0], linestyle='--', marker='s
           ', c='r', label=r'$400\times400$')
101    axs[0,0].plot(num_thread, num_thread, linestyle='-', c='k')
102
103    axs[0,1].plot(num_thread, cpu_time[0,0,1]/cpu_time[0,:,1], linestyle='--', marker='s
           ', c='k', label=r'$50\times50$')
104    axs[0,1].plot(num_thread, cpu_time[1,0,1]/cpu_time[1,:,1], linestyle='--', marker='s
           ', c='b', label=r'$100\times100$')
105    axs[0,1].plot(num_thread, cpu_time[2,0,1]/cpu_time[2,:,1], linestyle='--', marker='s
           ', c='g', label=r'$200\times200$')
106    axs[0,1].plot(num_thread, cpu_time[3,0,1]/cpu_time[3,:,1], linestyle='--', marker='s
           ', c='r', label=r'$400\times400$')
107    axs[0,1].plot(num_thread, num_thread, linestyle='-', c='k')
108
109    axs[1,0].plot(num_thread, cpu_time[1,:,0]/cpu_time[0,:,0], linestyle='--', marker='s
           ', c='b', label=r'$100\times100$')
110    axs[1,0].plot(num_thread, cpu_time[2,:,0]/cpu_time[0,:,0], linestyle='--', marker='s
           ', c='g', label=r'$200\times200$')
111    axs[1,0].plot(num_thread, cpu_time[3,:,0]/cpu_time[0,:,0], linestyle='--', marker='s
           ', c='r', label=r'$400\times400$')
112
113    axs[1,1].plot(num_thread, cpu_time[1,:,1]/cpu_time[0,:,1], linestyle='--', marker='s
           ', c='b', label=r'$100\times100$')
114    axs[1,1].plot(num_thread, cpu_time[2,:,1]/cpu_time[0,:,1], linestyle='--', marker='s
           ', c='g', label=r'$200\times200$')
115    axs[1,1].plot(num_thread, cpu_time[3,:,1]/cpu_time[0,:,1], linestyle='--', marker='s
           ', c='r', label=r'$400\times400$')
116
117    # axis labels
118    axs[0,0].set_ylabel(r'$S$', fontsize=24, labelpad=15)
119    axs[0,1].set_ylabel(r'$S$', fontsize=24, labelpad=15)
120    axs[1,0].set_ylabel('cpu␣time␣normalized␣by␣×5050', fontsize=24, labelpad=15)
121    axs[1,1].set_ylabel('cpu␣time␣normalized␣by␣×5050', fontsize=24, labelpad=15)
122    # title
123    axs[0,0].set_title('(a)␣no␣writing', fontsize=26, color='k', y=1.03)
124    axs[0,1].set_title('(b)␣writing', fontsize=26, color='k', y=1.03)
125    axs[1,0].set_title('(c)␣no␣writing', fontsize=26, color='k', y=1.03)
126    axs[1,1].set_title('(d)␣writing', fontsize=26, color='k', y=1.03)
127    # log plot
128    axs[1,0].set_yscale('log', base=2)
129    axs[1,1].set_yscale('log', base=2)
130
131    for i in range(2):
132        for j in range(2):
133            # legend
134            axs[i,j].legend(fontsize=20, fancybox=True, edgecolor='silver')
135            # axis labels
136            axs[i,j].set_xlabel('the␣number␣of␣threads', fontsize=24, labelpad=15)
137            # grid
138            axs[i,j].grid(which='major', color='silver', linewidth=0.1)
139            axs[i,j].grid(which='minor', color='silver', linewidth=0.1)
140            # direction and width of ticks
```

```
141             axs[i,j].tick_params(axis='both', which='major', direction='out', length=3,
                    width=0.8, labelsize=22)
142             # width of outer frame
143             axs[i,j].spines["bottom"].set_linewidth(1.2)
144             axs[i,j].spines["top"].set_linewidth(1.2)
145             axs[i,j].spines["right"].set_linewidth(1.2)
146             axs[i,j].spines["left"].set_linewidth(1.2)
147     # save
148     fig.savefig('./../Figs/{}.png'.format(file_name), format='png', dpi=300, transparent
            =False)
149     fig.savefig('./../Figs/{}.pdf'.format(file_name), format='pdf', transparent=True)
150
151 ###
152 print('[Message] program has completed.')
```