Vol.2013-HPC-139 No.18
2013/5/30

並列化した多倍長陰的Runge-Kutta法の性能分析

幸谷 智紀[1,a)]

概要：実用上重要な「固い」常微分方程式を効率的に解くためには陰的解法が相応しい．我々は高次多倍長陰的 Runge-Kutta 法を混合精度反復改良法を用いて高速化し，ブロック三重対角化を行って効率化を図った多倍長精度の ODE ソルバーを開発した．今回はこのアルゴリズム全体に OpenMP による並列化を行い，マルチコア CPU 上において更なる高速化に成功した．本論文では多倍長 ODE ソルバーのアルゴリズムと数値的特性を示し，プロファイリングによってどの程度の性能向上が行われたかを明らかにする．

キーワード：常微分方程式，多倍長計算，陰的 Runge-Kutta 法，性能分析

Performance Analysis of Parallelized Fully Implicit Runge-Kutta Methods in Multiple Precision Computing Environment

Tomonori Kouya[1,a)]

Abstract: Implicit algorithms such as implicit Runge-Kutta methods are appropriate to solve "stiff" ordinary differential equations (ODEs) numerically which play important role in various scientific simulations. We have already developed the ODE solver based on high-order multiple precision fully implicit Runge-Kutta (IRK) methods accelerated by using mixed precision iterative refinement method and reduction to block tridiagonal form. Our whole parallelization of IRK methods by using OpenMP can also accelerate the ODE solver successfully. In this paper, we show the numerical property of our ODE solver based on IRK methods and reveal how fast the parallelized IRK process can run through profiling.

Keywords: Ordinary Differential Equation, Multiple Precision Computing, Implicit Runge-Kutta Method, Performance Analysis

1. 初めに

科学技術開発における大規模シミュレーションが一般化するにつれ，得られた計算結果の精度 (accuracy) や物理的性質についての関心が高まっている．近年，構造保存数値解法や多倍長計算の研究が盛んになっているのはその顕れであろう．

常微分方程式 (ODE) の初期値問題を多倍長計算で求めるためには高次の解法が相応しい．多倍長計算によって丸め誤差が小さく抑えられる分，相対的に打切り誤差 (離散化誤差) が増大するため，高次公式を導入することで積分区間の刻み幅が大きくとることができるようになる．結果として，ユーザの要求する精度が高ければ高いほど，1 刻み分の計算は遅くなっても，トータルの刻み数が大幅に減ることで全体の計算時間を減らすことができるからである．

もし種々の安定性を満足し，シンプレクティック解法の一つでもある Gauss 型陰的 Runge-Kutta(IRK) 法の多倍長計算が高速に実行できれば，様々な悪条件かつ硬い (stiff) 常微分方程式に相応しい ODE ソルバーの基盤アルゴリズムになり得る．Gauss 型 IRK 公式はシフト Legendre 多項式のゼロ点から自動的に導出できるため，任意の次数の公式も簡単に作ることができる．

我々は，定評ある任意精度浮動小数点演算ライブラリである MPFR[12]/GMP[1] を土台とした，多倍長数値計算ライブラリ BNCpack[9] を開発してきた．その上で，Gauss

[1] 静岡理工科大学
Shizuoka Institute of Science and Technology, 2200-2 Toyosawa, Fukuroi, Shizuoka Pref. 437–8555, Japan
[a)] tkouya@cs.sist.ac.jp

型 IRK 公式導出に必要となる Legendre 多項式の任意精度ゼロ点計算の実装 [10] や，混合精度反復改良法を導入することで IRK 法の大幅な高速化が可能であることも示してきた [13]．今回我々は，IRK 公式に基づく倍精度 ODE ソルバーの一つである SPARK3[8] が採用しているブロック三重対角リダクションを取り入れ [14]，更に Hairer による埋め込み型公式の導入 [6] に基づく局所誤差評価も取り入れ，実用的な刻み幅制御機能も取り込んだ任意次数 (段数)Gauss 型公式に基づく IRK 法の実装を行った．その結果，高次公式を用いることでより高精度な近似解を高速に導出できることを確認している [15].

更なる高速化を図るため，IRK 法全体のプロセスを OpenMP を用いて並列化することでパフォーマンス向上を目指した．その結果，最適とは言えないまでもコア数に応じた高速化が達成できていることが確認できた．

本稿では今回我々が実装した多倍長 ODE ソルバーの概要を述べたのち，二つの例題について並列化した ODE ソルバーの性能分析を行った結果について報告する．

2. 簡易 Newton 法を用いた IRK 法のアルゴリズム

解くべき n 次元常微分方程式の初期値問題を

$$\begin{cases} \frac{d\mathbf{y}}{dt} = \mathbf{f}(t, \mathbf{y}) \ \in \mathbb{R}^n \\ \mathbf{y}(t_0) = \mathbf{y}_0 \end{cases} \qquad (1)$$

積分区間:$[t_0, \alpha]$

とする．この常微分方程式は一意な解を持つものとする．即ち，$\forall \mathbf{v}, \mathbf{w} \in \mathbb{R}^n$, $\forall t \in [t_0, \alpha]$ に対して Lipschitz 定数 $L > 0$ を持ち

$$||\mathbf{f}(t, \mathbf{v}) - \mathbf{f}(t, \mathbf{w})|| \leq L||\mathbf{v} - \mathbf{w}|| \qquad (2)$$

を満足するものとする．

(1) を m 段陰的 Runge-Kutta 法を用いて解くアルゴリズムは次のようになる．積分区間を t_0, $t_1 := t_0 + h_0$, ..., $t_{k+1} := t_k + h_k$... と離散化し，t_k から t_{k+1} への近似解 $\mathbf{y}_{k+1} \approx \mathbf{y}(t_{k+1})$ を求めるため，次の 2 段階の計算を行う．

(A) 次の非線型方程式を $\mathbf{Y} = [Y_1 \ ... \ Y_m]^T \in \mathbb{R}^{mn}$ について解く．

$$\begin{cases} Y_1 &= \mathbf{y}_k + h_k \sum_{j=1}^m a_{1j} \mathbf{f}(t_k + c_j h_k, Y_j) \\ &\vdots \\ Y_m &= \mathbf{y}_k + h_k \sum_{j=1}^m a_{mj} \mathbf{f}(t_k + c_j h_k, Y_j) \end{cases}$$

以下，これを

$$\mathbf{F}(\mathbf{Y}) = 0 \qquad (3)$$

と書く．

(B) 求めた $\mathbf{Y}$ を用いて次の近似解 $\mathbf{y}_{k+1}$ を求める．

$$\mathbf{y}_{k+1} := \mathbf{y}_k + h_k \sum_{j=1}^m b_j \mathbf{f}(t_k + c_j h_k, Y_j)$$

ここで m 段 IRK 公式を規定する係数 $c_1, ..., c_m$, $a_{11}, ..., a_{mm}$, $b_1, ..., b_m$ を

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1m} \\ \vdots & \vdots & & \vdots \\ c_m & a_{m1} & \cdots & a_{mm} \\ \hline & b_1 & \cdots & b_m \end{array} = \begin{array}{c|c} \mathbf{c} & A \\ \hline & \mathbf{b}^T \end{array} \qquad (4)$$

と書くことにする．

非線型方程式 (3) を何らかの反復法で解く過程を IRK 法の内部反復と呼ぶ．RADAU5[7] や SPARK3[8] といった IRK 法ベースの倍精度 ODE ソルバーでは収束性の維持とパフォーマンスの向上の最適化を意図し，簡易 Newton 法が用いられている．この時，漸化式は

$$\mathbf{Y}_{l+1} := \mathbf{Y}_l - (I_m \otimes I_n - h_k A \otimes J)^{-1} \mathbf{F}(\mathbf{Y}_l) \qquad (5)$$

となる．ここで I_n, I_m はそれぞれ $n \times n$, $m \times m$ の単位行列，J は $\mathbf{f}$ の Jacobi 行列 $\partial \mathbf{f} / \partial \mathbf{y}(t_k, \mathbf{y}_k) \in \mathbb{R}^{n \times n}$ である．

したがって簡易 Newton 法 (5) の反復一回ごとに，連立一次方程式

$$(I_n \otimes I_m - h_k A \otimes J)\mathbf{Z} = -\mathbf{F}(\mathbf{Y}_l) \qquad (6)$$

を $\mathbf{Z}$ について解き，$\mathbf{Y}_{l+1} := \mathbf{Y}_l + \mathbf{Z}$ として $\mathbf{Y}_{l+1}$ を計算する必要がある．

3. 内部反復の高速化手法と刻み幅制御

内部反復 (A) が必ず唯一解を持ち，その解に収束させられる，という理論的な保証は IRK 法の提案者である Butcher によって示されている [3]．内部反復に単純反復法を用いた場合，$a = \max_{2 \leq i \leq m} \sum_{j=1}^{i-1} |a_{ij}| + \max_{1 \leq i \leq m} \sum_{j=i}^{m} |a_{ij}|$ とすると，この a と Lipschitz 定数 L を用いて

$$|h_0| < \frac{1}{aL}$$

を満足する時，単純反復法が縮小写像になることが示されている．しかしこれは A 安定である IRK 法のメリットを打ち消す可能性も示唆している．

従って，実用的な IRK 法の実装においては，$L >> 1$ となる硬い (stiff)ODE に対して十分大きな刻み幅 h_k を用いての収束性を維持しつつ，内部反復の高速性も達成できることが望ましい．そこで Hairer は内部反復として簡易 Newton 法を採用し，A の複素対角化を行って高速化を図り，低次の埋め込み型公式を用いた刻み幅制御機能を備えた実用的な ODE ソルバー，RADAU5[7] を開発した．その後，Jay は W 変換と呼ばれる係数の関係性から導出された A の相似変換を用いて，ブロック三重対角化を行う実用的な内部反復計算法を提案し，SPARK3[8] という倍精度 ODE ソルバーを提供している．我々は両者の内部反復スキームの高速化手法を比較し，後者の SPARK3 型のブ

ロック三重対角化リダクションが高次 IRK 公式の実装に適していることを示した [14]．更に倍精度計算を用いた混合精度反復改良法を適用することで大幅な性能向上が図れることもすでに示している [13]．以下，後述する性能分析に必要となるこの二つの内部反復の高速化手法について述べ，指定精度に近い値を得るための刻み幅制御についても述べる．

3.1 SPARK3 型ブロック三重対角化

内部反復として簡易 Newton 法を用いたときに解くべき連立一次方程式は (6) のようになる．Jay は IRK 公式の係数の関係性を利用して非対称実ブロック三重対角化し，高速に解けるようにする方法を提案している [11]．我々の実装ではこれを利用して高速化を図っている．

Gauss 型公式の場合，IRK 公式 (4) における係数 A は

$$X = W^T BAW = \begin{bmatrix} 1/2 & -\zeta_1 & & & \\ \zeta_1 & 0 & -\zeta_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \zeta_{m-2} & 0 & -\zeta_{m-1} \\ & & & \zeta_{m-1} & 0 \end{bmatrix} \tag{7}$$

とできる．ここで $\tilde{P}_{j-1}(x)$ は $j-1$ 次シフト Legendre 多項式であり，

$$\begin{aligned} W &= [w_{ij}] = [\tilde{P}_{j-1}(c_i)] \in \mathbb{R}^{m\times m} \\ B &= \mathrm{diag}(\mathbf{b}) \in \mathbf{R}^{m\times m} \\ \zeta_i &= 1/(2\sqrt{4i^2-1}) \ (i = 1, 2, ..., m-1) \end{aligned}$$

である．これを Hairer らは W 変換 [6] と呼んでいる．SPARK3 ではこれにより (6) の行列を次のように実非対称ブロック三重対角化して計算を行っている．

$$\begin{aligned} &(W^T B \otimes I_n)(I_m \otimes I_n - h_k A \otimes J)(W \otimes I_n) \\ &\quad = I_m \otimes I_n - h_k X \otimes J \\ &\quad = \begin{bmatrix} E_1 & F_1 & & & \\ G_1 & E_2 & F_2 & & \\ & \ddots & \ddots & \ddots & \\ & & G_{m-2} & E_{m-1} & F_{m-1} \\ & & & G_{m-1} & E_m \end{bmatrix} \end{aligned} \tag{8}$$

ここで各ブロックの行列 E_i, F_i, G_i は

$$\begin{aligned} E_1 &= I_n - \frac{1}{2} h_k J \\ E_i &= I_n \ (i = 2, 3, ..., m) \\ F_i &= h_k \zeta_i J \ (i = 1, 2, ..., m-1) \\ G_i &= -h_k \zeta_i J \ (i = 1, 2, ..., m-1) \end{aligned}$$

である．

これによってリダクションされた (6) は

$$(I_m \otimes I_n - h_k X \otimes J)((W \otimes I_n)^{-1}\mathbf{Z}) = (W^T B \otimes I_n)(-\mathbf{F}(\mathbf{Y}_l)) \tag{9}$$

となる．

通常の行列のリダクションとは異なり，ブロック三重対角化に伴う計算コストは発生しない．W．B は IRK プロセス開始前に決まっている定数であり，X は Jacobi 行列 J から自動的に求めることができる．従って実際に内部反復で計算する必要があるのは (9) の求解と，解の変換 $(W \otimes I_n)^{-1}\mathbf{Z}$，及び右辺の $(W^T B \otimes I_n)(-\mathbf{F}(\mathbf{Y}_l))$ である．

また SPARK3 では左前処理行列 $P \approx I_m \otimes I_n - h_k X \otimes J$ として，$\tilde{H}_i$ を

$$\tilde{H}_i = I_n - (2(2i-1))^{-1} h_k J \ (i = 1, 2, ..., m)$$

として作り，次のようなブロック LU 分解を行った P を提案し，実装している．

$$\begin{aligned} P = &\begin{bmatrix} I_n & & & & \\ G_1\tilde{H}_1^{-1} & I_n & & & \\ & \ddots & \ddots & & \\ & & G_{m-2}\tilde{H}_{m-2}^{-1} & I_n & \\ & & & G_{m-1}\tilde{H}_{m-1}^{-1} & I_n \end{bmatrix} \\ &\times \begin{bmatrix} \tilde{H}_1 & F_1 & & & \\ & \tilde{H}_2 & F_2 & & \\ & & \ddots & \ddots & \\ & & & \tilde{H}_{m-1} & F_{m-1} \\ & & & & \tilde{H}_m \end{bmatrix} \end{aligned} \tag{10}$$

今回我々が示した例のうち，Brusselator 問題はこの左前処理行列を倍精度計算 Krylov 部分空間法に用いて計算されている．この場合は

$$\begin{aligned} &P^{-1}(I_m \otimes I_n - h_k X \otimes J)((W \otimes I_n)^{-1}\mathbf{Z}) \\ &= P^{-1}(W^T B \otimes I_n)(-\mathbf{F}(\mathbf{Y}_l)) \end{aligned} \tag{11}$$

という連立一次方程式を解いていることになる．

3.2 倍精度–多倍長精度混合精度反復改良法

簡単のため，解くべき連立一次方程式を

$$C\mathbf{x} = \mathbf{d} \tag{12}$$

と書くことにする．これに対して Newton 法を適用し，次のように計算コストの軽い S 桁計算と，計算コストの重い L 桁計算（一般に $S << L$) を組み合わせて高速化を図ったアルゴリズムを $S-L$ 型混合精度反復改良法と呼ぶ [2]．以下，$[S]$，$[L]$ はそれぞれ S 桁，L 桁計算によって得られた (表現された) 値であることを示している．

1) $C^{[S]}\mathbf{x}_0^{[S]} = \mathbf{d}^{[S]}$ を解き，$\mathbf{x}_0^{[S]}$ を得る．
2) $\mathbf{x}_0^{[L]} := \mathbf{x}_0^{[S]}$
for $\nu = 0, 1, 2, ..., \nu_{max}$
(a) $\mathbf{r}_\nu^{[L]} := \mathbf{d}^{[L]} - C^{[L]}\mathbf{x}_\nu^{[L]}$
(b) $\mathbf{r'}_\nu^{[L]} := \mathbf{r}_\nu^{[L]}/||\mathbf{r}_\nu^{[L]}||$
(c) $\mathbf{r'}_\nu^{[S]} := \mathbf{r'}_\nu^{[L]}$
(d) $C^{[S]}\mathbf{z}_\nu^{[S]} = \mathbf{r'}_\nu^{[S]}$ を解き，$\mathbf{z}_\nu^{[S]}$ を得る．
(e) $\mathbf{z}_\nu^{[L]} := \mathbf{z}_\nu^{[S]}$
(f) $\mathbf{x}_{\nu+1}^{[L]} := \mathbf{x}_\nu^{[L]} + ||\mathbf{r}_\nu^{[L]}||\mathbf{z}_\nu^{[L]}$
(g) 収束判定

S 桁計算として IEEE754 倍精度計算，L 桁計算として多倍長計算を用いることで，比較的良条件である (6) を高速に解くことができることは既に示している [13]．今回はさらにリダクションして得た (9) に適用することで，より高次の IRK 公式を適用しても効率的に計算できることが確認できている．

さらに今回我々はベクトルの正規化 (b), (f) を取り入れることで，倍精度浮動小数点数と多倍長浮動小数点数の指数部の著しい長さの違いについて吸収することができるよう改良を行い，倍精度–多倍長混合精度反復改良法の計算桁数制限を事実上取り除くことができた．

また，混合精度反復改良法の 1), (d) では直接解法だけでなく，帯行列形式の Jacobi 行列にも対応した Krylov 部分空間法を使用できるようにした．これによって，偏微分方程式の離散化で得られるような大次元 ODE にも対応することができるようになった．

3.3 埋め込み型公式による刻み幅制御

実用的な ODE ソルバーには局所誤差評価に基づく刻み幅 h_k の制御機能が欠かせない．現時点で IRK 法のための実用的かつ簡易な刻み幅制御法としては Hairer が導出した，低次埋め込み型公式によるものが唯一と言える．Swart ら [4] による陰的誤差評価法という手法も提案されているが，計算が複雑になるため，多倍長計算に向いているかどうかは未知である．従って我々は Hairer が実装した方法を実装している．以下この低次埋め込み型公式に基づく刻み幅制御機能について述べる．

埋め込み型公式は既出の Y_1, Y_2, ..., Y_m をそのまま利用して低次の近似解 $\hat{\mathbf{y}}_{k+1}$ を求めるものである．Hairer は γ_0 を定数として与え，

$$\begin{array}{c|cc} 0 & 0 & \mathbf{0}^T \\ \mathbf{c} & \mathbf{0} & A \\ \hline & \gamma_0 & \hat{\mathbf{b}}^T \end{array}$$

という $m+1$ 段 IRK 公式を提案した．我々はなるべく A 安定領域が大きくなるよう，$\gamma_0 = 1/8$ としている [15]．ここで $\hat{\mathbf{b}} = [\hat{b}_1 \ \cdots \hat{b}_m]^T$ は簡易化公式 $B(m)$ を満足するように

$$\begin{bmatrix} 1 & \cdots & 1 \\ c_1 & \cdots & c_m \\ \vdots & & \vdots \\ c_1^{m-1} & \cdots & c_m^{m-1} \end{bmatrix} \begin{bmatrix} \hat{b}_1 \\ \hat{b}_2 \\ \vdots \\ \hat{b}_m \end{bmatrix} = \begin{bmatrix} 1-\gamma_0 \\ 1/2 \\ \vdots \\ 1/m \end{bmatrix}$$

を解くことで得られる値である．これを用いて $\hat{\mathbf{y}}_{k+1}$ を

$$\hat{\mathbf{y}}_{k+1} := \mathbf{y}_k + h_k\gamma_0\mathbf{f}(t_k, \mathbf{y}_k) + h_k\sum_{j=1}^{m}\hat{b}_j\mathbf{f}(t_k + c_j h_k, Y_j)$$

として求め，m 次の近似値として，以下の $\mathbf{y}_{k+1}$ の局所誤差評価値 $\mathbf{err}_k$ の計算に用いる．

$$||\mathbf{err}_k|| = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\frac{|\hat{y}_i^{(k+1)} - y_i^{(k+1)}|}{ATOL + RTOL\max(|y_i^{(k)}|, |y_i^{(k+1)}|)}}$$

ここで $ATOL$ は絶対誤差許容値，$RTOL$ は相対誤差許容値で，ユーザが要求する精度に応じて与える定数である．

これを用いて，次の刻み幅 h_{k+1} を

$$h_{k+1} := 0.9||\mathbf{err}_k||^{m+1}h_k$$

として用いている．

4. 数値実験と性能分析

以上示してきたように，我々の実装した高次 IRK 公式に基づく ODE ソルバーは SPARK3 型ブロック三重対角化を行い，更に混合精度反復改良法を用いて劇的に高速化しており，実用的な刻み幅制御機能も備えているものになっている．ここでは小規模問題と大規模問題を使って性能の比較を行い，更に詳細なプロファイリングを行って，本 ODE ソルバーのパフォーマンス特性を分析する．

ここで使用した計算機環境は次の通りである．OpenMP は Intel C++標準装備の機能を使用している．

H/W Intel Core i7 3820 (4 cores) 3.6GHz + 64GB RAM
OS Scientific Linux 6.3 x86_64
S/W Intel C++ 13.0.1, MPFR 3.1.1/GMP 5.1.1, BNCpack 0.8

4.1 Lorenz モデル

小規模問題の例として，複雑系モデルの一つである Lorenz モデルを取り上げる．

$$\begin{cases} \dfrac{d\mathbf{y}}{dx} = \begin{bmatrix} \sigma(-y_1 + y_2) \\ -y_1y_3 + ry_1 - y_2 \\ y_1y_2 - by_3 \end{bmatrix} \\ \mathbf{y}(0) = [0\ 1\ 0]^T \end{cases} \quad (13)$$

積分区間:$[0, 50]$

ここで $\sigma = 10$, $r = 470/19$, $b = 8/3$ である．このパラメータ値の時，このモデルは周期解を持たず，複雑な解曲線を

描くことが広く知られている．その際，数値解も精度を落とし，この積分区間の場合は 10 進約 13 桁の桁落ちを起こす．従って倍精度計算では信頼できる近似解が得られない．

常微分方程式としては解きやすい部類に入るため，陽的解法も十分実用として使える．ここでは多倍長計算が必要な問題として取り上げることにした．

Lorenz モデルの計算は解の精度低下分を考慮して十分大きな桁数，10 進約 200 桁 (665 bits) で計算を行うことにし，80, 100, 120 段の Gauss 型公式を用いて計算した．更に $RTOL$ の値を変えて計算時間，および近似解の成分ごとの相対誤差の比較を行った．ここでは直接法ベースの倍精度–多倍長精度混合精度反復改良法を用いている．その結果を**表 1** に示す．

表 1 Lorenz モデルの数値計算結果
Table 1 Numerical results of Lorenz model

200 dec.digits	$RTOL = 10^{-120}, ATOL = 0$		
# stages(m)	80	100	120
Comp.Time(s)	1991.4	2317.4	2555.0
2 Threads (*)	1130.3 (1.8)	1310.6 (1.8)	1443.8 (1.8)
4 Threads (*)	659.3 (3.0)	762.1 (3.0)	2215.4 (3.0)
# steps	1661	863	563
Average (s)	1.2	2.7	4.5
2 Threads	0.7	1.5	2.6
4 Threads	0.4	0.9	1.5
Max.Rel.Error	6.5E-110	1.3E-109	2.3E-109
Min.Rel.Error	1.2E-111	2.4E-111	4.4E-111
	$RTOL = 10^{-170}, ATOL = 0$		
# stages(m)	80	100	120
Comp.Time(s)	8233.6	5761.3	5285.9
2 Threads (*)	4832.6 (1.7)	3341.2 (1.7)	3069.6 (1.7)
4 Threads (*)	2858.1 (3.0)	2089.4 (2.8)	1826.0 (2.9)
# steps	6879	2603	1370
Average (s)	1.2	2.2	3.4
2 Threads	0.7	1.3	2.2
4 Threads	0.4	0.8	1.3
Max.Rel.Error	1.0E-161	9.0E-160	7.2E-160
Min.Rel.Error	1.9E-162	1.7E-161	1.3E-161

(*) ... Speedup Ratio

精度低下以上に十分桁数を確保して計算しているため，丸め誤差による精度低下の影響は見られない．どの段数，どの $RTOL$ の値においても，$RTOL$ より 10 桁程度低い相対誤差の近似解が得られていることがわかる．また，$RTOL$ を 10^{-120} から 10^{-170} にすることによってほぼ 50 桁，近似解の精度を増加させることにも成功している．

$RTOL = 10^{-120}$ の場合は 80 段 (160 次) 公式を用いた時が最も短時間で計算できている．

さらに精度を要求して $RTOL = 10^{-170}$ とすると，120 段公式の方が短時間で済んでいる．これは 80 段，100 段公式を用いた時に比べてトータルの刻み数 (# steps) の増加が低く抑えられているためである．時間方向には逐次的に計算していくだけなので，1 刻みごとの計算時間がトータルの刻み数倍されて全体の計算時間が決まる．そのため，120 段公式では 1 刻みごとの計算時間が 80 段と比較すると 2.8 倍程度，100 段と比較して 1.5 倍程度伸びているが，それ以上に刻み幅が大きくなっており，刻み数が少なくなっている (6879 → 2603 → 1370)．従って，トータルの計算時間は 120 段公式の方が抑制されることになる．結果として要求精度が高くなる ($RTOL$, $ATOL$ を小さくする) ことによって，高次公式を用いた方が計算時間は短縮できることが分かる．

アルゴリズム全体を並列化することで，2 スレッド利用時には 1.7〜1.8 倍，4 スレッド利用時には 2.0〜3.0 倍の性能向上を達成することができた．

4.2 1 次元 Brusselator 問題

大次元の問題の例として，時間発展偏微分方程式を離散化して ODE 化した 1 次元 Brusselator 問題 [6] を取り上げる．これは元の偏微分方程式 (14)

$$\begin{cases} \frac{\partial u}{\partial t} = 1 + u^2 v - 4 + 0.02 \cdot \frac{\partial^2 u}{\partial x^2} \\ \frac{\partial v}{\partial t} = 3u - u^2 v + 0.02 \cdot \frac{\partial^2 v}{\partial x^2} \end{cases} \tag{14}$$

を次のように ODE 化したものである．

$$\begin{cases} \frac{du_i}{dt} = 1 + u_i^2 v_i - 4 + 0.02 \cdot \frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta x)^2} \\ \frac{dv_i}{dt} = 3u_i - u_i^2 v_i + 0.02 \cdot \frac{v_{i+1} - 2v_i + v_{i-1}}{(\Delta x)^2} \\ \Delta x = 1/(N+1) \\ u_0(t) = u_{N+1}(t) = 1,\ v_0(t) = v_{N+1}(t) = 3, \\ u_i(0) = 1 + \sin(2\pi i \Delta x),\ v_i(0) = 3 \end{cases}$$
$$(i = 1, 2, ..., N)$$
$$\text{積分区間 } [0, 10] \tag{15}$$

空間方向 (x 方向) には差分法を用いて離散化を行い，その刻みは Δx とする．Jay は [8] において $N = 500$, $n = 2N = 1000$ 次元の時の問題を解き，倍精度数値解を得ているので，それと比較して数値解の検証ができるよう，同じ条件で問題を多倍長計算で解くことにした．Lorenz モデル同様，三種類の公式 (20, 30, 40 段) を使い，10 進 70 桁相当 (233 bits) の多倍長計算を行って，数値解の精度及びパフォーマンスを確認した．なお，メモリに制約の大きい環境であることと，Jacobi 行列が帯行列として表現できることを考量して，この計算は P を左前処理行列とする疎行列対応 BiCGSTAB 法ベースの混合精度反復改良法を用いている．その結果を**表 2** に示す．

Lorenz モデルの計算同様，$RTOL$, $ATOL$ を減らすことで，その大きさに比例した相対誤差が得られていることが分かるが，段数を増やすと得られる数値解の精度が下がるという傾向もみられる．$RTOL = ATOL = 10^{-50}$ の時は 20 段公式の使用が最も短時間で計算できており，数値解

表 2　Brusselator 問題の数値計算結果
Table 2　Numerical results of Brusselator Problem

70 dec.digits	$RTOL = ATOL = 10^{-50}$		
# stages(m)	20	30	40
Comp.Time(s)	8866.0	9271.5	10171.5
2 Threads (*)	5431.3 (1.6)	5446.9 (1.7)	5857.2 (1.7)
4 Threads (*)	4195.4 (2.1)	3978.8 (2.3)	4159.4 (2.4)
# steps	1629	860	553
Average (s)	5.4	10.8	18.4
2 Threads	3.3	6.1	9.4
4 Threads	2.6	4.5	6.7
Max.Rel.Error	2.9E-41	2.1E-38	9.0E-35
Min.Rel.Error	1.4E-43	1.0E-40	4.1E-37
	$RTOL = ATOL = 10^{-60}$		
# stages(m)	20	30	40
Comp.Time(s)	19712.0	11377.8	13667.6
2 Threads (*)	11875.8 (1.7)	6680.3 (1.9)	7519.6 (1.8)
4 Threads (*)	8966.9 (2.2)	4784.8 (3.3)	5000.6 (2.7)
# steps	3249	890	620
Average (s)	6.1	12.8	22.0
2 Threads	3.7	7.5	12.1
4 Threads	2.8	5.4	8.1
Max.Rel.Error	4.8E-53	1.1E-43	1.7E-44
Min.Rel.Error	7.6E-56	4.9E-46	7.4E-47

(*) ... Speedup Ratio

も，30 段, 40 段公式使用時に比べ，それぞれ約 3 桁，6 桁程度精度が良い．更に 10 桁，近似解の有効数字を増やそうとすると 20 段公式は途端に非効率的となり，30 段公式の利用が最も短時間で済んでいることがわかるが，精度はやはり 20 段公式が最も良く，30 段，40 段公式使用時に比べて 10〜11 桁程度良くなっている．

相対誤差は段数の低い公式利用時のほうが良い傾向にあるが，これは刻み幅制御の結果，高次公式より小さい刻み幅を選択させられるため，結果として細かい刻みによって良い近似解が得られているものと推察される．実際，$RTOL = ATOL = 10^{-60}$ の時，更に追加条件として最大刻み幅 ($\max h_k$) を 0.005, 0.002 に制限して実行した結果，最大相対誤差は 20 段公式と同レベルにすることができた (**表 3**)．

大次元の問題において高次公式を使う際には，近似解の要求精度を満足させるため，より小さい $RTOL, ATOL$ の値を設定すべきであろう．また，刻み幅制御方式も他の手法と比較してみる必要がある．

並列化効率は Lorenz モデルの計算例同様，2 スレッド使用時で 1.6〜1.9 倍，4 スレッド使用時で 2.1〜3.3 倍となっており，並列化できない左前処理の計算によるボトルネックの影響が 4 スレッド使用時には大きくなっている．

表 3　最大刻み幅を制限した場合の数値実験結果
Table 3　Numerical results obtained by using limited maximum stepsizes

70 dec.digits	$\max h_k = 0.005$		$\max h_k = 0.002$	
# stages(m)	30	40	30	40
Comp.Time(s)	21834.0	32983.4	43723.7	74230.3
2 Threads	12777.6	18551.9	25509.7	41921.4
4 Threads	9286.8	12421.8	18641.4	28060.2
# steps	1864	1748	3856	4156
Max.Rel.Error	1.1E-49	7.4E-49	3.4E-53	2.9E-53
Min.Rel.Error	5.3E-52	3.5E-51	5.4E-56	9.9E-57

4.3　プロファイリングによる比較

以上見てきたように，我々の実装した ODE ソルバーは Lorenz モデル，Brusselator 問題それぞれに対して良い精度の数値解が得られることが判明した．また，並列化したことによって計算時間を減少させることも成功した．

全体的に，高次公式を用いることでトータルの計算時間はある程度抑えられるが，段数が大きくなる分，解くべき連立一次方程式の次元数も格段に大きくなり，直接法や直接法ベースの前処理を行うことが時間短縮のボトルネックとなっていることも判明している．以下，プロファイリングによってどの部分に計算時間が費やされているのかを調査した結果を示す．

まず, 我々の実装した高次 IRK 法に基づく多倍長 ODE ソルバーのアルゴリズムの概略をまとめて示す．$\mathbf{y}_k \approx \mathbf{y}(t_k)$ から $\mathbf{y}_{k+1} \approx \mathbf{y}(t_{k+1}) = \mathbf{y}(t_k + h_k)$ への近似値を計算するステップのみを抜き出すと**図 1** のようになる．

$\mathbf{Y}_{-1} := [\mathbf{y}_k\ \mathbf{y}_k\ ...\ \mathbf{y}_k]^T \in \mathbb{R}^{mn}$
For $l = 0, 1, 2, ...$ 簡易 Newton 法のループ
(1) $\mathbf{Y}_l := [Y_1^{(l)}\ Y_2^{(l)}\ ...\ Y_m^{(l)}]^T$
　　where $Y_i^{(l)} = \mathbf{y}_0 + h_k \sum_{j=1}^{m} a_{ij}\mathbf{f}(t_k + c_i h_k, Y_i^{(l-1)})$
(2) $C := I_m \otimes I_n - h_k X \otimes J$, Compute $||C||_F$
(3) $\mathbf{d} := (W^T B \otimes I_n)(-\mathbf{F}(\mathbf{Y}_l))$
(4) Solve $C\mathbf{x}_0 = \mathbf{d}$ for $\mathbf{x}_0$ (S)
　　For $\nu = 0, 1, 2, ...$ 混合精度反復改良法のループ
　　(5) $\mathbf{r}_\nu := \mathbf{d} - C\mathbf{x}_\nu$
　　(6-1) $\mathbf{r}'_\nu := \mathbf{r}_\nu / ||\mathbf{r}_\nu||$ (S)
　　(6-2) Solve $C\mathbf{z} = \mathbf{r}'_\nu$ for $\mathbf{z}$ (S)
　　(6-3) $\mathbf{x}_{\nu+1} := \mathbf{x}_\nu + ||\mathbf{r}_\nu||\mathbf{z}$
　　(6-4) Check convergence $\Rightarrow \mathbf{x}_{\nu_{stop}}$
(7) $\mathbf{Y}_{l+1} := \mathbf{Y}_l + (W \otimes I_n)\mathbf{x}_{\nu_{stop}}$
　　Check convergence $\Rightarrow \mathbf{Y}_{l_{stop}}$
$\mathbf{Y} := \mathbf{Y}_{l_{stop}} = [Y_1\ Y_2\ ...\ Y_m]^T$
$\mathbf{y}_{k+1} := \mathbf{y}_k + h_k \sum_{j=1}^{m} b_j \mathbf{f}(t_k + c_j h_k, Y_j)$

図 1　実装した IRK 法のアルゴリズム
Fig. 1　Our implemented IRK algorithm

ここで，収束判定の結果，簡易 Newton 法は l_{stop} 回で，混合精度反復改良法は ν_{stop} 回で停止しているとする．

このうち (1)〜(7) が計算に時間を要すると考えられる個所である．倍精度–多倍長精度混合反復改良法を用いた場合は，(4), (6-1), (6-1) が倍精度で計算され，他の部分は多倍長計算される．この部分毎の計算時間を計測し，トータルの計算時間のうちどの程度の割合になっているのかを調査した．問題ごとの計算時間割合の傾向は問題と使用した連立一次方程式の解法に依存しているため，Lorenz モデルの場合は 80 段，$RTOL = 10^{-170}$ の計算を，Brusselator 問題の場合は 30 段，$RTOL = ATOL = 10^{-60}$ の計算を分析の対象としている．その結果をまとめたグラフを**図 2** に示す．

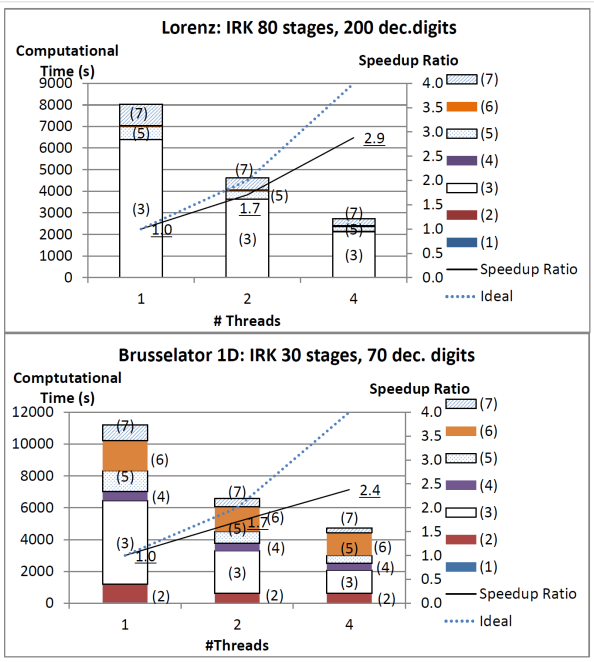

図 2 プロファイリング結果: Lorenz モデル (上), Brusselator 問題 (下)

Fig. 2 Results of profiling: Lorenz Model(upper), Brusselator Problem(lower)

小次元で高次公式を 200 桁計算した場合 (Lorenz モデル) と，大次元問題を前処理付き Krylov 部分空間法で比較的小さい桁数 (70 桁) を用いて解いた場合 (Brusselator 問題) では時間を要する箇所が共通している部分と，異なっている部分があることが分かる．共通しているのは多倍長計算による相似変換処理 (3)(7)，残差計算 (5) の割合が高いことである．

異なっているのは，小次元問題の場合，連立一次方程式を解いている部分は全く目立たないのに対し，大次元問題の場合は，多倍長行列のセット (2)，倍精度計算部分 (4)(6-1)(6-2) の比率が高まり，左前処理 ((11) 式) のための計算も増えており，その分 (3), (5), (7) の割合が減っていることである．概して，次元数が増えると連立一次方程式の計算量が相対的に増え，倍精度–多倍長精度混合反復法を用いていると，倍精度計算による連立一次方程式の求解回数が増えてくるため，このような結果になっている．4 スレッドを用いた場合の並列化性能が大次元問題では落ちているが，これは並列化できない左前処理における前進・後退代入がボトルネックになっていることが大きい．

以上のプロファイリングの結果により，今回行った SPARK3 型リダクションと混合精度反復改良法の利用によって格段に多倍長連立一次方程式求解の計算が軽くなった分，問題の次元数や使用する公式の段数によって性能のボトルネックの箇所が異なってくるということが判明した．

5. 結論と今後の課題

以上示してしてきたように，倍精度–多倍長精度混合反復法の利用に加え，SPARK3 型のブロック三重対角化リダクションを行うことで，高次陰的 Runge-Kutta 法の多倍長計算も実用段階に入ったと言える．すべての問題に対して最良の解法であるかどうかは，問題や計算環境によって異なるが，高精度計算が必要な複雑系時間発展偏微分方程式の解法としては利用用途はあると考えている．

そのためにはまず第一に性能向上がどこまで行えるかを詰める必要がある．ことに頑健な倍精度反復解法が性能向上には不可欠であるため，そのための機能追加も行うとともに，様々な問題に対して本解法を適用し，問題ごとの特性に合った連立一次方程式の解法を見極めていきたい．

また，高次の IRK 公式の数値的特性についてはよくわからないところが多いため，Testset[5] で示されているような実用的な問題に適用し，性能評価と精度の検証を行いつつその特性を調べていきたい．特に，倍精度計算ではせいぜい 10 数桁程度の精度しか得られず，近似解の $RTOL$, $ATOL$ と精度との関係が不明確であったことが多倍長計算を行うことで明確になった．複数提案されている刻み幅制御方式の比較を行いつつ，$RTOL$, $ATOL$ と精度との関係を見極めて行きたい．

なお，ここで示した計算を行ったプログラム群は，BIRK(extented Bncpack for Implicit Runge-Kutta methods) パッケージとして一部公開している．

謝辞 本研究のきっかけになった博士論文に対して，励ましの言葉をいただいた永坂秀子先生と田中正次先生に御礼申し上げる．また，本研究は静岡理工科大学研究プロジェクト (B) の助成を受けて行っている．並列化作業はは秋田県立大学システム科学技術学部電子情報システム学科にてサバティカル滞在中に，シミュレーション工学研究室（小澤一文教授，廣田千明准教授，中村真輔助教）から研究環境の提供を受けて行ったものである．感謝関係各位に感謝する．

参考文献

[1] Swox AB. The GNU Multiple Precision arithmetic library. http://gmplib.org/.
[2] A.Buttari, J.Dogarra, Julie Langou, Julien Langou, P.Luszczek, and J.Karzak. Mixed precision iterative refinement techniques for the solution of dense linear system. *The International Journal of High Performance Computing Applications*, Vol. 21, No. 4, pp. 457–466, 2007.
[3] J. C. Butcher. Implicit Runge-Kutta processes. *Mathematics of Computations*, Vol. 18, pp. 50–64, 1964.
[4] J. J. B. de Swart and G. Söderlind. On the construction of error estimators for implicit runge-kutta methods. *Modelling, Analysis and Simulation*, 1997.
[5] F. Mazzi et.al. Test set for IVP solvers. http://www.dm.uniba.it/~testset/testsetivpsolvers/.
[6] E. ハイラー, G. ヴァンナー, 三井斌友・監訳. 常微分方程式の数値解法 II 応用編. シュプリンガー・ジャパン, 2008.
[7] E. Hairer. RADAU5. http://www.unige.ch/~hairer/software.html.
[8] L. O. Jay. SPARK3. http://www.math.uiowa.edu/~ljay/SPARK3.html.
[9] Tomonori Kouya. BNCpack. http://na-inet.jp/na/bnc/.
[10] 幸谷智紀. 実用的な古典的誤差評価法の提案と gauss 型積分公式の分点計算への応用について. 情報処理学会論文誌, Vol. 48, No. SIG18(ACS20), pp. 1 – 11, 2007.
[11] L.O.Jay. A parallelizable preconditioner for the iterative solution of implicit Runge-Kutta-type methods. *Journal of Computational and Applied Mathematics*, Vol. 111, pp. 63–76, 1999.
[12] MPFR Project. The MPFR library. http://www.mpfr.org/.
[13] 幸谷智紀. 倍精度と多倍長精度浮動小数点数を用いた反復改良法による連立一次方程式の高精度高速解法について. 日本応用数理学会論文誌, Vol. 19, No. 3, pp. 313–328, 2009-09-25.
[14] 幸谷智紀. W 変換を用いた高次陰的 Runge-Kutta 法の実装. 第 41 回数値解析シンポジウム予稿集, 2012.
[15] 幸谷智紀. 陰的 Runge-Kutta 法の埋め込み型公式について. 平成 24 年度日本応用数理学会講演予稿集, 2012.