Vol.2016-HPC-155 No.45
2016/8/10

数式の依存グラフを仕様とするOpenMPプログラムのメモリ参照効率向上

角田 優貴[1]　置田 真生[1]　安部 武志[2]　浅井 義之[2]　北野 宏明[2]　萩原 兼一[1]

概要：有向非巡回グラフ G は，単一代入プログラムの計算順の規定に用いられる．G を解析することで，並列実行可能な計算順序を得られる．これに基づいて生成した並列プログラム PP では，変数の主記憶への配置およびレベル内の評価順に自由度がある．本研究では，データ間の依存関係に注目して主記憶参照効率の良いデータ配置およびスケジュールを定めるための手法を提案する．評価には汎用生体シミュレータ Flint を用いた．Flint は，XML 系の言語で書かれた生体モデルから数式の依存グラフを作成し，それに基づき並列シミュレーションコードを生成する．心筋細胞のモデルから生成した 32 並列 OpenMP コードの実行において，最大 1.61 倍の高速化を達成した．

キーワード：データ配置　スケジューリング　参照局所性　生体シミュレーション

Improving Memory Efficiency of OpenMP Program specified in Equation Dependency Graph

Yuki Kakuda[1]　Masao Okita[1]　Takeshi Abe[2]　Yoshiyuki Asai[2]　Hiroaki Kitano[2]
Kenichi Hagihara[1]

Abstract: Directed acyclic graph G is used to determine an evaluation order of a single-assignment form program. Analysis of G results in a parallel schedule. The parallel program PP generated from it has flexibility in terms of the data layout and the schedule in each level. we propose methods of data arrangement and scheduling for improving memory efficiency. In experiments, we apply our methods to Flint, a general physiological simulator. Flint interprets a given model written in a XML-based language as an equation dependency graph, and then generates a parallel simulation program based on the graph. With a model of ventricular cardiac action potential, our method accelerates OpenMP programs running on 32 threads up to 1.61 times.

Keywords: Data arrangement; Scheduling; Data locality; Physiological simulation

1. はじめに

有向非巡回グラフ（以降，DAG） G は，単一代入プログラムの計算順の規定に用いられる．G の頂点は単一代入（変数 w とその値を求める式）を表す．G の有向辺 $e = <u, w>$ は，依存関係を表し，w の評価より前に u が評価されなければならない．G に対してトポロジカルソートを行うと，依存関係に矛盾しない計算順序を得られる．さらに，トポロジカルソートの特殊な例であるレベルソートによって，同時評価可能な頂点集合（レベル）を決定でき，並列実行可能な計算順序を得られる．各レベルの頂点集合は唯一に定まるが，この計算順序に基づいて生成した並列プログラム PP では，変数の主記憶への配置およびレベル内の評価順に自由度がある．PP を実行したときの主記憶参照効率が配置によって異なることが原因で，実行性能に

[1] 大阪大学 大学院情報科学研究科 コンピュータサイエンス専攻
Department of Computer Science, Graduate School of Information Science and Technology, Osaka University
[2] 沖縄科学技術大学院大学 統合オープンシステムユニット
Integrated Open Systems Unit, Okinawa Institute of Science and Technology

差異が生じる可能性があるため，参照効率の良いデータ配置を定める方法が重要である．また，その配置に対してアクセスする順番，つまり計算順序も参照効率に影響する．

DAG に基づく PP 生成の 1 つに Flint[1] がある．Flint は生体モデル記述言語である PHML (Physiological Hierarchy Markup Language)[2] を入力として，モデルの構造を解析し，シミュレーションコード（以降，SC）を出力する．近年，生体モデルの大規模化が進んでおり，高性能計算によるシミュレーションの高速化が求められている．Flint は多様な高性能計算環境に対応しており，プロセス内並列実行には OpenMP[3] を，プロセス間並列実行には MPI (Message Passing Interface)[4] を，GPU 実行には CUDA (Compute Unified Device Architecture)[5] を用いた SC を生成できる．

一般に，高性能計算においては主記憶参照が性能ボトルネックとなるため，その効率化が重要である．ステンシル計算および行列計算などの規則的なデータ配置と参照パターンを持つ計算については既存研究が数多く存在し，手動による最適化が有効である．しかし，不規則な参照パターンを含み得る DAG に基づいた計算については，大規模になると手動最適化は難しい．

そこで，DAG で規定される PP の主記憶参照効率の向上を目的として，その自動化に取り組む．まず PP の主記憶参照を分析し，効率向上の可能性を検討する．次に，PP のデータ配置とスケジュールを変更して，参照効率を向上する手法を提案する．

本報告では PP の生成系として Flint を対象に提案手法を実装し，評価する．ただし提案手法は，数式 DAG を入力としてスケジュールおよびデータ配置を出力する一般的な問題に対して，適用可能である．

以降では，2 節で研究の背景について説明し，3 節で取り組む問題を定義する．4 節では，既存手法の問題点を分析する．5 節で問題を解決する手法を提案し，6 節で手法の評価を行う．最後に 7 節で本報告についてまとめる．

2. 背景

PHML で記述されたモデル（以降，PHML モデル）は連立一階常微分方程式からなるシステムを表現できる．PHML の主なコンポーネントをモジュールと呼び，生物学・生理学上の要素を表す．モジュールは複数の変数を含み，各変数は値を定義する式を持つ．式は常微分方程式（ODE），初等関数，および条件式の組合せからなり，異なるモジュールの変数を参照可能である．したがって，PHML モデルは複数モジュールが構成するネットワークを表現する．なお，PHML は式を羅列するだけで，それらの計算順序は明示しない．

ODE の初期値問題を差分法で解く場合，PHML モデルは 1 ステップを計算する単一代入プログラムを規定する DAG G と解釈できる．以降では，前進オイラー法の利用を前提とする．

リスト 1: SC のサンプル

```
1  double vals[number_of_variables];
2  double nv[number_of_ODEs];
3
4  int main() {
5    #pragma omp parallel
6    for(i = 1; step * i < length; i++)
7      AdvanceStep();
8  }
9
10 void AdvanceStep() {
11   int i;
12   // Phase 0
13   #pragma omp for nowait
14   for(i = 0; i<10; i++) {
15     vals[c[i+20]] = vals[c[i+0]] * vals[c[i+10]] * 120;
16     vals[c[i+50]] = vals[c[i+30]] * (vals[c[i+40]] - 115);
17   }
18   #pragma omp for nowait
19   for(i = 0; i<10; i++) {
20     vals[c[i+70]] = 1 / exp((-vals[c[i+60]]) + 30);
21   }
22
23   #pragma omp barrier
24   // Phase 1
25   ...
26
27   #pragma omp for nowait
28   for(i = 0; i<30; i++) {
29     nv[c[i+260]] = vals[c[i+200]] * (1 - vals[c[i+230]]);
30   }
31
32   #pragma omp barrier
33   // State update
34   #pragma omp for
35   for(i = 0; i < num_odes; i++) {
36     vals[s[i]] += nv[s[i]] * step;
37   }
38 }
```

奥山ら [1] は G をもとに OpenMP プログラムを生成する手法を提案した．G のレベルソートを用いて式をレベルに分類し，各レベル内の式を式形（3 節で後述）によって並べ替える．同一式形の式群をまとめて 1 つの for 文に変換し，OpenMP の並列化指示文を付与する．

生成された SC の例をリスト 1 に示す．`step` および `length` は，それぞれシミュレーションのステップ幅 s，シミュレーション長 l である．変数の値は 1 次元配列 `vals` に格納する．`nv` は，変数値ではなくオイラー法における差分を一時的に格納するための配列である．関数 `AdvanceStep()` に 1 ステップの計算を記述する．この関数内は OpenMP のバリア指示文でフェーズに分割され，各フェーズがソート済み G のレベルに対応する．フェーズは for 文の並びで

表 1: 変数定義の例

$v = + \ u \ * \ w \ 3$
$F(v) = (+, *)$
$R(v) = (u, w, v)$

構成される．同一 for 文の繰り返しにおいて参照する変数の配置が必ずしも連続でないため，間接参照を用いる．1 ステップの末尾には，オイラー法の計算を行うフェーズ P_S（図中の `State update`）が存在する．さらに，PHML モデルによっては P_S の後に後計算を行うフェーズ P_A が存在する．

また，奥山らの手法では SC の主記憶参照効率を向上するため，スケジュールをもとにデータ配置を決定する．スケジュールとは代入文の順序を指し，データ配置とは `vals` における変数の配置順を指す．

3. 定義

本研究の対象となる，DAG を入力とする OpenMP プログラム生成系のためのスケジュール問題およびデータ配置問題を定義する．

この問題の入力は，DAG $G = (V, E)$ で与えられる．G の頂点は変数を表し，辺はステップ内における変数間の依存関係を表す．

変数 v は自身が表す変数の計算式で定義する．計算式は前置記法で記述し，演算子は四則演算および初等関数を，被演算子は他の変数および実数定数をとる．$F(v)$ および $R(v)$ を，それぞれ計算式における演算子の列（以降，式形）および参照する変数の列（以降，参照順）と定義する．これらの例を**表 1** に示す．$R(v)$ は，計算後の代入を表現するため末尾に自身 v を含む．

前提として，既存研究 [1] と同様に，G のレベルソートに基づいて V をフェーズ $P_i (0 \leq i < \gamma, \gamma$: フェーズ数) に分類する．

$$\bigcup_{i=0}^{\gamma-1} P_i \ = \ V \tag{1}$$

$$\forall < u, v > \in E (u \in P_i, v \in P_j \ \Rightarrow \ i < j) \tag{2}$$

さらに，各フェーズを頂点の式形を用いて分類する．

$$Q_i = \{A \subset P_i \mid v, u \in A, w \notin A, F(v) = F(u) \neq F(w)\} \tag{3}$$

Q_i の各要素が生成されるプログラムの各 for 文に対応する．

本研究におけるスケジュール問題とは，全ての Q_i に対して以下の写像 f_i および g_i を定めることである．N_n は n 以下の自然数からなる集合を表す．

$$f_i : N_{|Q_i|} \to Q_i \tag{4}$$

$$g_i : Q_i \to \{h \mid A \in Q_i, \exists_1 h \in A^{N_{|A|}}\} \tag{5}$$

以降では，写像 f で定義される列を単に f と表記する．Q_i から for 文を変換するにあたって，for 文の順序は列 f_i に従い，for 文内の計算順序は列 $g_i(A)$ に従う．したがって，P_i のスケジュール S_i および全体のスケジュール S は以下の列となる．ここで，列の和 $X \bigcup Y$ は列 X の後ろに列 Y を連結した列である．

アルゴリズム 1 layout(R)

```
i ← 0
for each v ∈ R do
  if (v ∉ L) then
    L(i++) ← v
```

$$S_i = \bigcup_{A \in f_i} g_i(A) \tag{6}$$

$$S = \bigcup_{i=0}^{\gamma-1} S_i \tag{7}$$

S に従う場合の P_i における変数の参照順 R_i^+ および全体参照順 R^+ は次式となる．

$$R_i^+ = \bigcup_{v \in S_i} R(v) \tag{8}$$

$$R^+ = \bigcup_{v \in S} R(v) \tag{9}$$

また，データ配置問題は次の写像 L を定めることである．

$$L : N_{|V|} \to V \tag{10}$$

前提として，PP において変数は主記憶上に連続的に配置する．$L(i) = v$ であれば変数 v の配置は i 番目である．

3.1 既存手法

既存手法 [1] における L の決定方法について説明する．Flint では，主記憶への参照のうち変数の読み込みを重視し，変数の読み込み順 R^- に従って L を決定する．

$$R^- = \bigcup_{v \in S} (R(v) - (v)) \tag{11}$$

すなわち，アルゴリズム 1 を用いて layout(R^-) で L を定める．

また，既存手法における f_i は式の長さ（$|F(v)|$）をもとに降順で決定する．一方で g_i は任意である．PHML モデルにおける定義順およびソートアルゴリズムに依存して決定する．

4. 参照効率の分析

主記憶の参照効率は，R^+ および L の関係（以降，参照パターン）で決定する．本節では，参照分布図を用いて参照パターンを可視化し，既存実装の問題点を示す．

簡単のため，参照効率の分析に際しては逐次実行を前提とする．本研究で想定する PP では，for 文をブロック分

$$V = \left\{ \begin{array}{ll} C = A + B, & J = D/H \\ F = D * E, & X = I + A \\ I = G - H, & Y = C * H \end{array} \right\}$$

$S_0 = (C, F, I, J)$

$S_1 = (X, Y)$

図 1: モデルおよびスケジュールの例

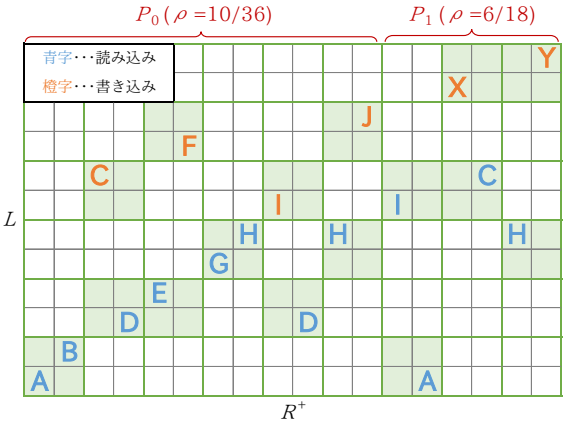


図 2: 参照分布図および参照集中率の例（$d = 2$）

割し，スレッドが動作する CPU を固定してマルチスレッド実行する．スレッド数に対して for 文の繰り返し数が十分大きければ，1 スレッド内の参照パターンは逐次実行の場合と同じである．したがって，逐次実行の解析結果を用いて，マルチスレッド実行の性能改善を期待できる．

4.1 参照分布図

参照分布図は，S に従い実行したときの変数の参照順 R^+ に対して，各変数の L 上の位置を点描した図である．時間的および空間的な参照の連続性を視覚的に表現できる．すなわち，参照分布図上で分布が集中していれば参照の局所性が高い．

参照分布図の例を**図 2** に示す．この図は，**図 1** のモデルとスケジュールに既存手法を適用した場合を表す．ただし，この例では変数 F, J, X, Y は読み込みの対象とならないため，データ配置が定まらない．ここでは L の末尾に登場順で追加した．実際のモデルでは全変数は 1 回以上読み込みの対象となるため，任意の変数 v に対して $L^{-1}(v)$ は一意に定まる．

4.2 参照集中率

参照分布図における集中度を定量的に評価するため，参照集中率 ρ を導入する． 参照集中率の定義は，参照分布図を一辺の長さ d の正方矩形で格子状に分割したとき，内部に点を含む矩形の全体に対する割合とする．例えば，図 2 に対して $d = 2$ とした場合，P_0 は $\rho = 10/36 = 0.28$，P_1 は $\rho = 6/18 = 0.33$ である．参照集中率が小さいほど参照分布が集中していることを表し，参照局所性が高い．参照集中率は参照分布図の任意の範囲に対して計算できる．

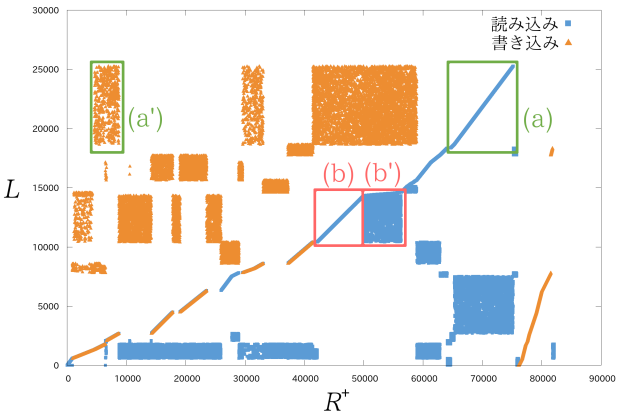


図 3: 既存実装における 1x301 の参照分布図

参照集中率の値は d によって変化する．本報告では，LLC に対するキャッシュミスが性能に与える影響が大きいと想定し，LLC の大きさ l を用いて次式のように定めた d_c を用いる．以降では，特に表記のない場合，参照集中率は $d = d_c$ で求めた値を示す．

$$d_c = \sqrt{l/\mathrm{sizeof(double)}} \qquad (12)$$

4.3 既存実装の問題

既存実装では，単純に読み込みの出現順に変数をデータ配置するため，各データの初回の読み込みにおいては参照ストライドが小さく，参照効率が良い．ただし，実際にはデータ参照は複数回発生する．(i) 書き込み および (ii)2 回目以降の読み込みでは，初回読み込みと同様の順番で参照されるとは限らないため，参照効率が悪い可能性がある．

実際の生体モデル 1x301 [*1] に対して既存実装を用いて生成した SC の参照分布図を**図 3** に示す．直感的に，参照パターンの分布は直線状と矩形状の 2 つあり，後者は分布が分散するために参照集中率が大きい．後者はさらに書き込みと読み込みの 2 つの場合に分かれる．

まず，書き込みが矩形状に分散する箇所 (a') に注目する．同じ主記憶位置を参照する (a) の参照集中率の値 0.10 と比較して，(a') の参照集中率は 0.93 と大きく，参照効率が悪い．この図では，同じ位置に対する最初の参照が書き込みである場合が，読み込みの場合より多い．

次に，読み込みが矩形状に分散する箇所 (b') に注目する．(a') と同様に，同じ主記憶位置を参照する (b) の参照集中率の値 0.16 と比較して，(b') の参照集中率は 1.0 と大きい．図が示すように，矩形状の参照パターンとなる読み込みは全て，同じ主記憶位置を 2 回目以降に読み込む場合である．

[*1] 301 個の視覚野神経細胞を直線状につなげたモデル

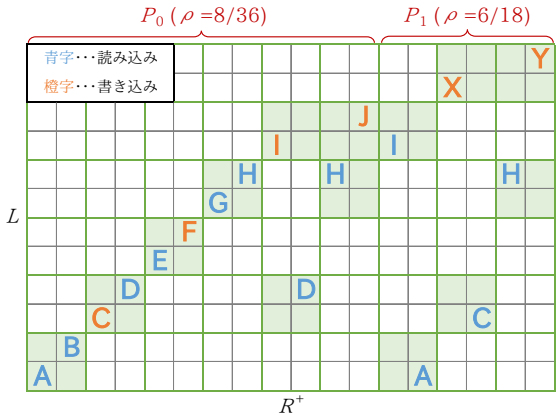


図 4: 図 1 に（P1）を適用した場合の参照分布図（$d = 2$）

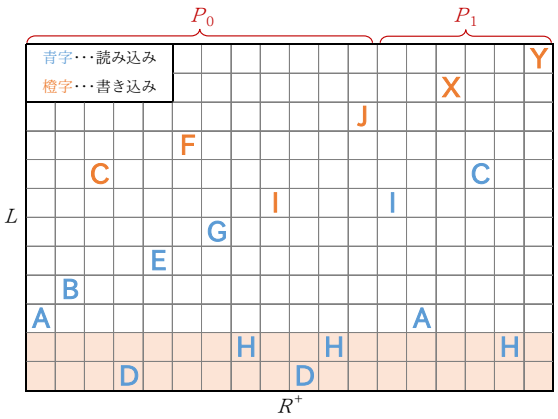


図 5: 図 1 に（P2）を適用した場合の参照分布図

アルゴリズム 2 フェーズ間の参照順を類似させるスケジューリング（P3）

```
for (i = 0..r − 1) do
  for each X ∈ f_i do
    Y_1 ← ( ), Y_2 ← ( ) {empty sequences}
    for each v ∈ g_i(X) do
      if R(v) ∩ R_r^+ ≠ φ then
        Y_1 ← Y_1 ∪ (v)
      else
        Y_2 ← Y_2 ∪ (v)
    g_i(X) ← Y_1 ∪ Y_2
```

5. 提案手法

DAG に基づいてデータ配置およびスケジュールを決定し，PP の主記憶参照効率を向上する手法を提案する．まず問題 (i) を解決するためのデータ配置について説明し，次に問題 (ii) を解決するためのスケジュールを説明する．

5.1 データ配置の変更

5.1.1 読み込み・書き込みを考慮した配置

(i) の問題を解決するため，読み込み順だけでなく書き込み順も考慮したデータ配置手法（P1）を提案する．具体的には，L の決定方法を変更し，読み込みと書き込みを同等に扱う参照順 R^+ に基づいて layout(R^+) で定める．

図 1 の例に対して（P1）を適用した場合の参照分布図を**図 4** に示す．図 2 と比較して，P_0 における参照集中率が減少する．

ただし，初回読み込みの参照集中率は既存手法と比較して増大する可能性がある．これは 5.2.1 節で後述する（P3）と組み合わせることで回避する．

5.1.2 参照回数に基づく配置

既存手法あるいは（P1）をさらに改善するため，参照回数に基づいてデータ配置を決定する手法（P2）を提案する．参照回数の多い変数を主記憶上のある範囲内に位置することで，それらを可能な限り同一キャッシュライン上に格納する．参照回数の多い変数を格納するキャッシュラインは参照頻度が比較的高いため，キャッシュラインの入れ替え回数が減少し，キャッシュヒット率の向上を期待できる．

具体的には，フェーズ P_i における変数の参照順を R_i^*，頂点 v の R_i^* 内の重複度を $m_i(v)$ として，次の R^m を用いて layout(R^m) で L を決定する．

$$R^m = \bigcup_{i=0}^{\gamma-1} \bigcup_{j=\max_{v\in R_i^*} m_i(v)}^{1} (v \in R_i^* \mid m_i(v) = j) \qquad (13)$$

$$R_i^* = \begin{cases} \bigcup_{v\in S_i} (R(v) - v) & \text{(既存手法)} \\ R_i^+ & \text{(P1)} \end{cases} \qquad (14)$$

例えば，フェーズの最初の数式が $X = A + B + C + D$ であるとする．このフェーズ内で他にも A が 4 回，B が 2 回，C が 8 回，D が 4 回出現すると仮定する．主記憶上の配置は参照回数の降順にしたがって C, A, D, B となる．

図 1 のスケジュールの場合，データ配置は $L = (D, H, A, B, E, G, I, C, F, J, X, Y)$ となる．参照分布図を**図 5** に示す．（P2）により，P_0 で出現回数の最も多い D, H が主記憶上の先頭部分に位置し，同一キャッシュライン上に格納される．

5.2 スケジュールの変更

問題 (ii) を解決するためには，L 上のある範囲を複数回参照する場合に，同一の参照順となることが望ましい．参照順はスケジュールに依存するが，全ての参照において同一の参照順となるスケジュールは一般に難しい．本報告では類似した参照順となるスケジュールを目指す．

5.2.1 フェーズ間の参照順を類似させるスケジューリング

フェーズ内のスケジュールがもつ自由度を利用して，フェーズ間で R_i^+ を類似させる手法（P3）を提案する．基準となる注目フェーズ P_r を決定し，アルゴリズム 2 を用いて $g_i(i < r)$ を変更することで，R_i^+ を R_r^+ に近づける．

具体例で期待される効果について説明する．図 1 に対して $P_r = P_1$ として（P3）を適用すると，$S_0 =$

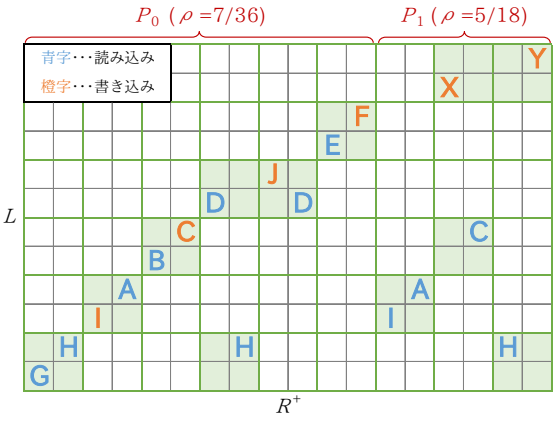


図 6: 図 1 に（P3）を適用した場合の参照分布図（$d = 2$）

表 2: 実験環境

CPU	Intel Xeon E5-4640v2 × 4 台
- 総コア数	40
- 動作周波数	2.2 GHz
- L1 キャッシュ	32 KB (データ), 32 KB (命令)
- L2 キャッシュ	256 KB
- L3 キャッシュ	20,480 KB
主記憶	1,024 GB
OS	CentOS 6.6
コンパイラ	gcc 4.9.3
プロファイラ	perf 2.6.32-573.3.1.el6.x86_64.debug

(I, C, J, F) となる．データ配置を（P1）で決定すると $L = (G, H, I, A, B, C, D, J, E, F, X, Y)$ である．この参照分布図を**図 6** に示す．図 2 と比較して P_0 における参照集中率が減少する．

（P3）による改善効果は，P_r に選択するフェーズによって変化する．本報告では，SC の実行時間が最短となる P_r を実験的に求めて使用する．

6. 評価

提案手法に対して，次の観点から評価する．

- PP のデータ配置の改善
- PP の実行性能
- PP の生成時間

評価実験の入力には，作為的な依存関係をもつ比較的小さな実験モデルおよび大規模な生体モデル 3 つを用いた．

CPU 実行での環境を**表 2** に示す．実行時間の計測においては，逐次実行，OpenMP の並列数 $n \in \{4, 8, 16, 32\}$ で計測する．計測値は，5 回実行したうち最小となる実行の結果を示す．キャッシュミス数の計測においては，L3 キャッシュの逐次実行時の値を計測する．

なお，提案手法適用前と適用後でシミュレーション結果を比較したところ，いずれの手法を組み合わせて適用した場合も計算結果に変化はなく，精度に問題はない．

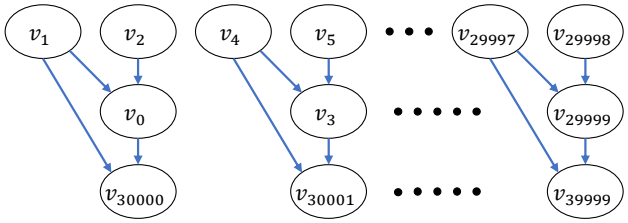
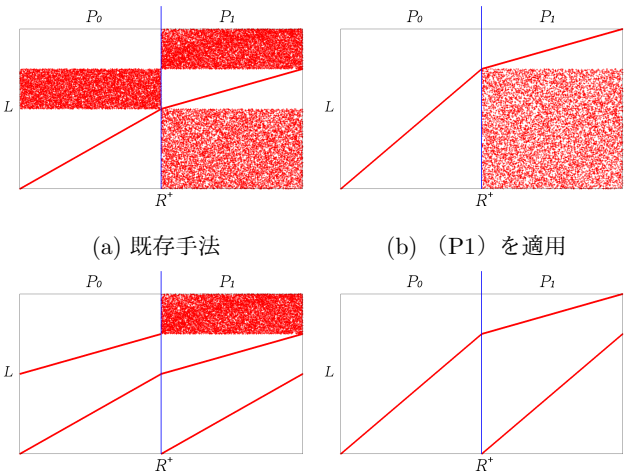


図 7: 実験モデル M の DAG



(a) 既存手法　(b) （P1）を適用

(c) （P3）を適用　(d) （P1）および（P3）を適用

図 8: 実験モデル M の参照分布図

6.1 実験モデルを用いた評価

作為的なモデルを用いて，提案手法が問題 (i) および問題 (ii)（4.3 節）を解消できることを示す．実験に用いたモデル M の DAG を**図 7** に示す．すべての変数の計算式は同じ $v = u + w$ である．スケジューリングの結果，変数は 2 つのフェーズ P_0 および P_1 に分かれる．各変数への書き込みは高々 1 回であるが，読み込みは 1 回の変数と 2 回の変数がある．既存手法におけるフェーズ内の式の計算順は任意であるため，ここでは `rand` 関数を用いて決定した．**図 8a** が示すように，M に既存手法を適用すると問題 (i) および問題 (ii) が発生する．

実行時間を**表 4** に示す．これは，同じ計算を 20,000 回実行した結果である．コンパイル時の最適化については，最適化オプション`-O2` を指定した．

まず，問題 (i) を解消するため（P1）を適用した場合の参照分布図を**図 8b** に示す．P_0 における参照パターンが直線状に集中しており，P_0 の参照集中率を 0.331 から 0.040 に改善した（表 4）．その結果，P_0 の実行時間を 64%に削減し，全体の実行時間を 66%に削減した．

次に，問題 (ii) を解消するため，$P_r = P_1$ として（P3）を適用した結果を**図 8c** に示す．P_1 における L の前半部分の参照パターンが直線状に集中しており，P_1 の参照集中率を 0.842 から 0.379 に改善した．その結果，P_1 の実行

表 3: 逐次実行における参照集中率

	101x101								bm64								fsk-epi100							
(P1)		✓		✓	✓	✓	✓	✓		✓		✓	✓	✓	✓	✓		✓		✓	✓	✓	✓	✓
(P2)			✓	✓			✓	✓			✓	✓			✓	✓			✓	✓			✓	✓
(P3)					P_S	P_A	P_S	P_A					P_S	P_A	P_S	P_A					P_S	P_A	P_S	P_A
P_B	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
P_0	0.136	0.066	0.114	0.021	0.015	0.031	0.017	0.014	0.048	0.018	0.039	0.008	0.012	0.004	0.004	0.003	0.031	0.013	0.026	0.007	0.008	0.003	0.003	0.002
P_1	0.211	0.142	0.194	0.114	0.142	0.141	0.122	0.115	0.036	0.045	0.030	0.036	0.033	0.033	0.027	0.028	0.018	0.033	0.014	0.028	0.021	0.025	0.019	0.022
P_2	0.251	0.042	0.244	0.043	0.047	0.046	0.047	0.044	0.047	0.059	0.037	0.041	0.041	0.042	0.032	0.030	0.029	0.036	0.025	0.026	0.020	0.022	0.015	0.017
P_3	-	-	-	-	-	-	-	-	0.024	0.048	0.019	0.041	0.035	0.028	0.029	0.025	0.015	0.028	0.013	0.025	0.020	0.017	0.016	0.015
P_4	-	-	-	-	-	-	-	-	0.021	0.005	0.018	0.005	0.006	0.005	0.005	0.005	0.015	0.004	0.014	0.004	0.006	0.003	0.005	0.004
P_5	-	-	-	-	-	-	-	-	0.010	0.002	0.010	0.002	0.002	0.001	0.002	0.001	0.009	0.002	0.010	0.002	0.002	0.000	0.002	0.000
P_S	0.105	0.314	0.090	0.245	0.156	0.286	0.109	0.220	0.041	0.079	0.043	0.083	0.014	0.078	0.014	0.080	0.024	0.047	0.025	0.052	0.006	0.046	0.007	0.050
P_A	0.047	0.093	0.014	0.015	0.093	0.021	0.015	0.012	0.028	0.036	0.015	0.022	0.036	0.014	0.022	0.010	0.019	0.027	0.013	0.019	0.024	0.008	0.018	0.007
全体	0.068	0.193	0.056	0.140	0.114	0.169	0.088	0.128	0.026	0.050	0.021	0.041	0.023	0.036	0.016	0.034	0.017	0.032	0.014	0.027	0.013	0.022	0.010	0.021

表 4: 実験モデル M の参照集中率および SC 実行時間 (s)

	(P1)		✓		✓
	(P3)			✓	✓
P_0	参照集中率	0.331	0.040	0.105	0.040
	実行時間	0.592	0.378	0.352	0.387
P_1	参照集中率	0.842	0.808	0.379	0.131
	実行時間	0.764	0.521	0.639	0.416
全体	参照集中率	0.604	0.439	0.256	0.086
	実行時間	1.357	0.900	0.991	0.803

時間を 84%に削減した．また，図 7 より，P_0 は P_1 と同様のパターンで参照するため，P_0 の書き込み部分の参照パターンも直線状となる．したがって，P_0 の実行時間に関しても，これを反映して 59%に削減した．全体の実行時間は 73%に削減した．

最後に，(P1) と (P3) 両方を適用した結果を**図 8d** に示す．(P1) で改善した P_0 における参照パターンを変化させることなく，P_1 における 2 回目の参照パターンを局所化できている．P_0 の参照集中率は (P1) のみ適用した場合と等しく，実行時間もほぼ同じである．P_1 の参照集中率は (P1) のみ，あるいは (P3) のみの場合よりも小さくなり，実行時間も短縮した．その結果，既存手法と比較して，全体の参照集中率を 0.604 から 0.086 に改善し，実行時間を 59%に削減した．

以上より，各手法単体でも高速化が期待できるが，スケジュールとデータ配置を両方変更する，すなわち (P1) および (P3) を同時に適用することで，問題 (i) および問題 (ii) を解消しさらなる実行時間の削減が可能となる．

6.2 生体モデルを用いた評価

実験対象としたモデルの情報を**表 5** に示す．シミュレーションコード実行時には，シミュレーションのステップ幅 s，シミュレーション長 l を指定する．s はモデル毎に設定されている基準値を利用した．l は，s を基準に全てのモデルで 1,000 ステップ実行するように設定した．本節の実験結果は，全て最適化あり (-O2) の場合の結果である．

6.2.1 データ配置の改善

本節では逐次実行の結果を元に論じる．各モデルのフェーズごとおよび全体の参照集中率を**表 3** に示す．表内では適用した手法にチェック (✓) を付す．なお，(P3) に関しては，チェックの代わりに注目フェーズ P_r を示す．ここで，各フェーズの参照集中率算出時の領域は，R^+ 方向は，フェーズ i の開始添字を b_i，終了添字を e_i としたときの $[b_i, e_i]$ である．L 方向は $[1, |L|]$ である．

表 5: 対象モデル

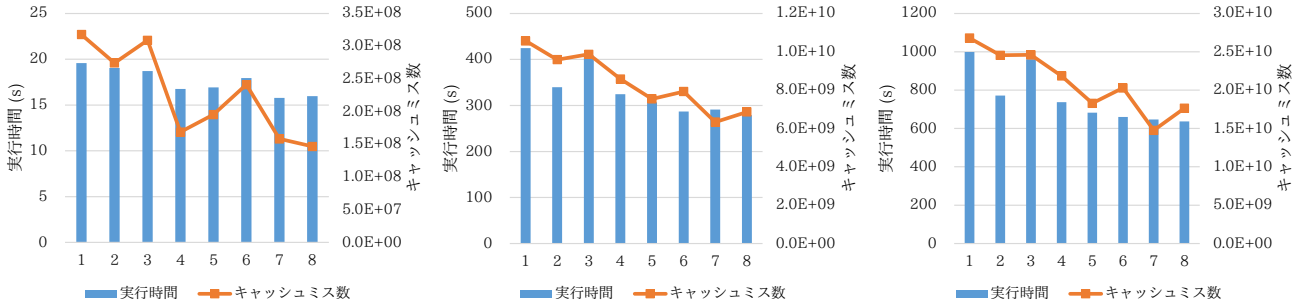
モデル名	細胞種別	$\|V\|$	γ	s	l
101x101	視神経	1,591,184	6	0.2 μs	0.2 ms
bm64	心室筋	22,648,353	8	5 μs	5 ms
fsk-epi100	心室筋	48,479,014	8	1 μs	1 ms

まず，(P1) によって，問題 (i) が解消され，全モデルで P_0 の参照集中率が改善される．次に，(P2) によって，ほとんどのフェーズで参照集中率が減少し，最大 69.2% 減少する．参照集中率が増大するフェーズもあるが，高々 6% にとどまり，全体としては参照集中率を改善した．最後に，(P3) によって，P_r の参照集中率が改善される．例えば，fsk-epi100 で，(P1) と (P3) ($P_r = P_S$) を併用した場合，P_S の参照集中率は 0.047 から 0.006 となる．

全体の L3 キャッシュミス数を**図 9** に示す．これと参照集中率との間には正の相関があると考えられる．しかし，相関係数を求めると，101x101 では-0.23 となり負の相関が現れる．他のモデルに関しても高々 0.40 と強い相関ではない．ここで，各モデルで外れ値となる (P1) を適用しない場合を除いて，同様に相関係数を求めると，0.84 から 0.93 と強い相関が得られる．したがって，(P1) の適用による参照分布図全体の変化を参照集中率が反映できていない．

全体の逐次実行時間を図 9 に示す．各モデルにおいて，全体の参照集中率が最小の組み合わせは，全ての手法を適用し，$P_r = P_S$ とした場合である．101x101 ではこの場合に最も実行時間が短いが，bm64 および fsk-epi100 では全ての手法を適用し $P_r = P_A$ とした場合に最も実行時間が短い．よって，実行時間は全体の参照集中率と対応しない場合がある．

一方で，フェーズ毎に改善の効果を調べるために参照集中率は有用である．具体例として，fsk-epi100 の各フェーズの実行時間 (**表 6**) と参照集中率との関係を述べる．(P1)



(a) 101x101　(b) bm64　(c) fsk-epi100

1: 既存手法	3: （P2）	5: （P1）&（P3），$P_r = P_S$	7: （P1）&（P2）&（P3），$P_r = P_S$
2: （P2）	4: （P1）&（P2）	6: （P1）&（P3），$P_r = P_A$	8: （P1）&（P2）&（P3），$P_r = P_A$

図 9: 逐次実行時間 (s) およびキャッシュミス数

表 6: 逐次実行時間 (s) の詳細（fsk-epi100）

（P1）		✓		✓	✓	✓	✓	✓
（P2）			✓	✓			✓	✓
（P3）					P_S	P_A	P_S	P_A
初期化	5.94	6.14	6.14	6.14	5.93	5.56	5.55	5.50
P_0	708.73	464.12	684.31	446.25	432.18	367.39	417.83	359.81
P_1	33.50	34.14	28.41	29.61	30.25	31.74	26.92	27.48
P_2	64.87	36.57	62.13	29.01	32.14	33.92	26.43	27.71
P_3	15.05	14.94	13.44	12.68	14.07	14.24	11.41	12.01
P_4	4.49	1.92	4.53	1.92	2.06	1.70	2.05	1.80
P_5	0.16	0.14	0.16	0.14	0.15	0.13	0.14	0.13
P_S	108.16	145.28	107.73	147.33	98.04	139.13	91.77	139.88
P_A	54.97	65.77	51.57	61.88	64.65	63.53	61.87	60.06
合計	998.57	771.74	961.06	737.62	682.20	660.03	646.62	636.99
速度向上	1.00	1.29	1.04	1.35	1.46	1.51	1.54	1.57

表 7: 全手法適用時のフェーズ毎の既存手法に対する速度向上（fsk-epi100, $P_r = P_A$）

	逐次	$n = 4$	$n = 8$	$n = 16$	$n = 32$
P_0	1.970	3.215	3.130	2.960	2.655
P_1	1.219	1.175	1.023	0.865	0.813
P_2	2.341	2.013	1.836	1.575	1.301
P_3	1.253	1.296	1.228	1.174	1.103
P_4	2.500	3.465	3.295	2.608	2.271
P_5	1.204	3.726	2.857	2.368	2.153
P_S	0.773	0.801	0.855	0.835	0.814
P_A	0.915	0.879	0.902	1.014	1.139

表 8: 手法適用時のシミュレーションコード生成時間 (s)

（P1）	（P2）	（P3）	101x101	bm64	fsk-epi100
			103.25	1623.96	3550.87
✓			103.80	1654.98	3607.04
	✓		104.73	1662.78	3630.46
✓	✓		106.80	1688.31	3903.52
✓		P_S	111.26	1823.67	3942.41
✓		P_A	108.36	1744.56	3814.10
✓	✓	P_S	112.23	1847.95	4056.61
✓	✓	P_A	109.86	1784.51	3868.69

によって，既存手法と比較して，最も影響を受ける P_0 の参照集中率は 0.031 から 0.013 となり，実行時間も 708.73 から 464.12 とそれに見合った改善となる．（P2）によって，既存手法と比較して，全体的に参照集中率が改善され，実行時間も全体的に減少する．（P1）と（P3）の併用，$P_r = P_S$ の場合には，（P1）のみと比較して，P_r の参照集中率は 0.047 から 0.006 と減少し，実行時間は 145.28 秒から 98.04 秒と改善される．

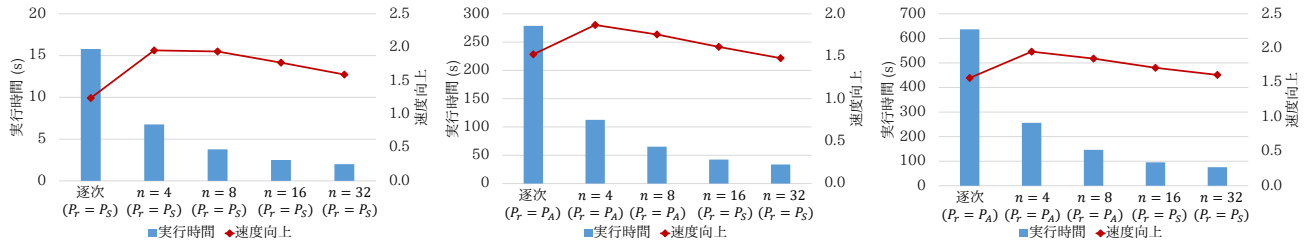
また，（P3）を適用した場合に，P_r 以外のフェーズが改善される副作用がある．とくに P_0 は顕著であり，参照集中率が 0.013 から 0.008 に減少し，実行時間が 464.12 秒から 432.18 秒に短縮する．この原因は現在判明しておらず，究明中である．

6.2.2 実行性能の改善

逐次実行で実行時間が最短となるのは全ての提案手法を適用した場合である．同じ条件での OpenMP 実行時間，および既存手法に対する速度向上を**図 10** に示す．なお，（P3）の注目フェーズについては，図 10 内に併記する．全モデルにおいて，並列数が増加しても速度向上は逐次実行と同等以上であり，提案手法により OpenMP 実行の性能を改善できる．

並列数が増加すると，bm64 および fsk-epi100 において，最適となる注目フェーズが P_A から P_S に変化した．この原因は，（P3）の副作用である．並列数が小さい場合には，$P_r = P_S$ の場合と比較して，$P_r = P_A$ における P_0 の実行時間が短い．しかし，並列数が増加すると，規模の大きい P_0 は効率的に並列化され，副作用の影響が小さくなる．その結果，全体の実行時間が逆転し $P_r = P_S$ が最適となった．

また，並列数を変化させたときの fsk-epi100 における全ての手法を適用した場合の既存手法に対する各フェーズの速度向上を**表 7** に示す．ただし，（P3）では $P_r = P_A$ とする．逐次実行において改善されているフェーズは，OpenMP 実行においても改善されており，概ね同様の傾向となっている．



(a) 101x101　(b) bm64　(c) fsk-epi100

図 10: OpenMP 実行時間 (s) および既存実装に対する速度向上

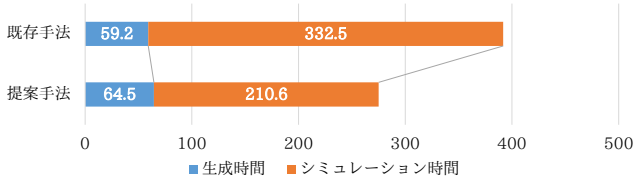


図 11: fsk-epi100 を 20,000 ステップ実行する際の SC 生成時間 (min) および SC 実行時間 (min)

6.3 有用性

手法適用時の SC 生成時間を**表 8** に示す．手法を適用した場合には生成時間が増加する．このオーバーヘッドが実用上問題がないことを検討する．実際のシミュレーションにおいては，今回実験で行った 1,000 ステップ程度ではなく，より多いステップ数でシミュレーションを行う．

例えば，fsk-epi100 を 0.02 秒分シミュレーションする場合には，SC を 20,000 ステップ実行する必要がある．**図 11** に示す通り，コード生成と SC 実行の時間の和が既存実装より短い．図 11 における提案手法とは，fsk-epi100 において最も効果があった手法，つまり，(P1)，(P2) および (P3) ($P_r = P_A$) を適用した手法である．

提案手法を適用すると SC 生成時間は増加するが，シミュレーション全体の大部分を占める SC 実行時間が減少する．また，一度生成した SC は l を変えて再度実行するなど，複数回再利用する状況も想定される．その場合，SC 生成時間は相対的に減少する．したがって，提案手法はシミュレーション作業全体の高速化に有用である．

7. おわりに

本報告では，DAG から並列プログラムを生成する際に，メモリ参照効率の良いデータ配置とスケジュールを決定するための 3 つの手法を提案した．まず，読み込み・書き込みを考慮した配置である．読み込みと書き込みの参照を区別することなく参照順で主記憶へと配置することで，初回参照の参照効率を向上させる．次に，参照回数に基づく配置である．参照回数の多い変数を集めて，参照頻度の高いキャッシュラインを作成することで，参照効率を向上させる．最後に，フェーズ間の参照順を類似させるスケジューリングである．同一領域の 2 回目以降の参照効率を向上させるため，基準となる注目フェーズの参照順に基づいて他のフェーズのスケジュールを変更する．

全ての提案手法を適用した場合に，既存手法に対する速度向上は，逐次実行で 1.24 倍から 1.58 倍，32 並列の OpenMP 実行で 1.48 倍から 1.61 倍であった．フェーズ間の参照順を類似させるスケジューリングにおける最適な注目フェーズはモデルや実行環境に依存する．

今後の課題として，以下の 2 つがある．まず，モデルや並列数によって最適な注目フェーズの決定方法を求める．次に，Flint が生成する GPU や MPI に対応したシミュレーションコードに本報告の手法を適用し，各環境向けの最適化を施す．

謝辞　本研究の一部は，科学研究費補助金（基盤研究（A）15H01687，若手研究（B）26730035）の支援による．

参考文献

[1] Okuyama, T., Okita, M., Abe, T., Asai, Y., Kitano, H., Nomura, T. and Hagihara, K.: Accelerating ODE-based Simulation of General and Heterogeneous Biophysical Models using a GPU, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 25, No. 8, pp. 1966–1975 (2014).

[2] PhysioDesigner.org: About Physiological Hierarchy ML (PHML), `http://physiodesigner.org/phml/index.html` (accessed on 2016-2-15).

[3] Chandra, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J. and Menon, R.: *Parallel Programming in OpenMP*, Morgan Kaufmann, San Mateo, CA (2000).

[4] Message Passing Interface Forum: MPI Documents, `http://www.mpi-forum.org/docs/docs.html` (2014).

[5] NVIDIA Corporation: CUDA C Programming Guide Version 7.5, `http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf` (2015).