GPUを活用した全結合・全並列アニーリング手法の高速化検討

大川 開生[1,a)]　川村 一志[1]　Gutmann Gregory[1]　Thiem Van Chu[1]　劉 載勲[1]　本村 真人[1]

概要：SCA（Stochastic Cellular Automata Annealing）は全結合イジングモデル上の全変数を同時に更新可能なアニーリング手法であり，本手法の高い並列性を活用することで幅広い組合せ最適化問題を高速に解けるようになるものと期待される．本稿では，本手法を対象にGPUの並列計算ユニットを最大限に活用可能な実装方法を検討し，高速で高いスケーラビリティを持つSCAアクセラレータを実現する．また，アニーリングが確率的に動作することをふまえ，同一パラメタ下でアニーリングを複数回実行させる場合についても検討し，高品質な解を高速に得ることを目指す．計算機実験では，複数の実装方法のもとで速度・精度を一元評価するとともに，従来のアニーリング手法であるSA（Simulated Annealing）との比較を実施した．その結果，SCAのGPUアクセラレータは従来SAに比べ最大160倍の高性能化を達成した．

1. はじめに

組合せ最適化問題は，多数の選択肢から制約条件を満たす中でベストな選択肢を見つけ出す問題であり，機械学習・創薬・金融ポートフォリオなど様々な場面に潜んでいる．それらの多くはNP完全またはNP困難であり，現実的な時間で最適解を見つけ出すことが難しい．この課題に対し，対象の組合せ最適化問題をイジングモデルの基底状態探索に置き換え，高速に準最適解を導き出すアニーリング計算技術の研究が盛んに進められている．アニーリング計算は商用量子アニーリングマシンD-Wave [9]の登場によって大きな注目を集め，現在では量子アニーリングに着想を得て開発された半導体ベースのアニーリングプロセッサ [6–8,10]も実用規模の問題を扱うことができるという点から有力視されている．

代表的な半導体ベースのアニーリングプロセッサには，日立CMOSアニーリングマシン [10]や富士通デジタルアニーラ [8]が存在し，ともにSA（Simulated Annealing）[13]を計算原理として動作する．イジングモデルを対象とするSAでは，非ゼロの相互作用を持つ変数（スピン）を同時に更新できず，逐次アルゴリズムとなる問題がある．この問題はイジングモデルの形状（トポロジ）を局所結合にすることで改善される（例：CMOSアニーリングマシン）が，局所結合型のイジングモデルは扱うことのできる問題に大きな制限を課すことから，実用面で新たな問題を生む．したがって，アニーリング処理の高速性とアニーリング計算の実用性を両立するためには，全結合型のイジングモデルに対してもスピンを並列に更新することのできる仕組みが求められる．

この仕組みを実現する方式のひとつにSCA（Stochastic Cellular Automata Annealing）[4]がある．SCAは現状態と隣接セルの関係から確率的に次状態を決定する確率的セルオートマトン（PCA: Probabilistic Cellular Automata）[12]の概念を取り入れたアニーリング手法であり，イジングモデル上の全スピンをスピン間相互作用の有無に関係なく同時に更新可能である．本手法は [4]にて数理的な解析がなされており，SCAを用いることでSAと同一の基底状態に到達可能であることが示されている．SCAを計算原理として設計・実装されたアニーリングプロセッサLSI（STATICA）[6,7]は，全結合型のイジングモデルを対象としたうえで，全スピンを並列に更新する高速なアニーリング処理を実現している．ところが，[6,7]で実現されたSTATICAは扱えるモデルの規模が小さく，実用性の面で課題がある．

この点を踏まえ本稿では，SCAのGPUによる高速化を検討し，評価する．アルゴリズムレベルで高い並列性を持つSCAをGPU上に実装することで，複雑なトポロジを持つ大規模イジングモデルに対する高速なアニーリング処理を実現する．また，アニーリング計算には確率的な動作が含まれることから，同一モデルに対して（同一パラメタ下での）複数回のアニーリングが必要となる場面も多い．このような場面では，個々のアニーリング処理を直列に実行

[1] 東京工業大学
[a)] okawa.kaisei@artic.iir.titech.ac.jp

するのではなく，複数回分のアニーリング処理を並列に実行することで，GPU 上での計算効率が向上するものと期待される．本稿にて，同一モデル・同一パラメタに対して複数個の解を同時に求める SCA の多重インスタンス実行に対する GPU 高速化を併せて検討・評価する．

2. 準備

対象の組合せ最適化問題をイジングモデルの基底状態探索として定式化することで，イジングマシンは統一的かつ高効率に多様な問題を解くことができる．本章では，組合せ最適化問題とイジングモデルの関係性，ならびに，従来用いられてきたイジングモデルの基底状態探索手法である SA を紹介する．

2.1 イジングモデル

イジングモデルは，相互作用 J_{ij} と外部磁場 h_i から影響を受ける N 個のスピン $\boldsymbol{\sigma} = \{\sigma_1, \cdots \sigma_N\}$ によって構成される．各スピン $\sigma_i \in \boldsymbol{\sigma}$ は $+1$ または -1 のどちらかの状態をとり，N 個のスピンの状態にもとづいてシステム全体のエネルギー（ハミルトニアン）を以下のように定める．

$$H(\boldsymbol{\sigma}) = -\sum_{i<j} J_{ij}\sigma_i\sigma_j - \sum_i h_i\sigma_i. \qquad (1)$$

イジングモデルの基底状態探索とは，ハミルトニアン $H(\boldsymbol{\sigma})$ を最小化するようなスピン状態の組 $\boldsymbol{\sigma}$ を求めることを目的とする．

組合せ最適化問題の多くは，イジングモデルの基底状態探索へと変換可能であることが知られている [3]．そのため，様々な問題をイジングマシンによって統一的かつ効率的に解くことができる．次節では，代表的な組合せ最適化問題のひとつである最大カット問題をイジングモデルの基底状態探索として定式化する方法を紹介する．

2.2 最大カット問題

最大カット問題は，与えられたグラフ $G = (V, E)$ 中の頂点を 2 グループに分割するとき，異なるグループに属する頂点間の辺の重み総和が最大化されるような分割を探索する問題である．この辺の重み総和をカット値と呼び，N $(= |V|)$ 変数 $\boldsymbol{s} = \{s_1, \cdots, s_N\}$ $(s_i \in \{+1, -1\})$ を用いて次式のように計算する．

$$C(\boldsymbol{s}) = \frac{1}{2}\sum_{(i,j)\in E} W_{ij}(1 - s_i s_j). \qquad (2)$$

ここで，W_{ij} は辺 $(i, j) \in E$ の重みを表し，$s_i \in \{+1, -1\}$ を用いて属するグループを示す．カット値 $C(\boldsymbol{s})$ の最大化は式 (2) の第 2 項 $(\sum W_{ij}s_i s_j)$ の最小化と等価であることから，式 (1) において $(i, j) \in E$ で $J_{ij} = -W_{ij}$，$(i, j) \notin E$ で $J_{ij} = 0$，$\sigma_i = s_i$，$h_i = 0$ とすることで，イジングモデルの基底状態探索へと帰着させることができる．

Algorithm 1 SA（Simulated Annealing）

Input: initial spin configuration ($\boldsymbol{\sigma}$),
spin-spin interaction ($\boldsymbol{J}$), external magnetic field ($\boldsymbol{h}$),
# of annealing steps (S),
pseudo temperature scheduling ($T(s)$)
Output: optimized spin configuration ($\boldsymbol{\sigma}$)
1: **for** $s = 1$ to S **do**
2: **for** $i = 1$ to N **do**
3: Calculate local field
▷ $\tilde{h}_i = \sum_j J_{ij}\sigma_j + h_i$
4: Calculate transition probability
▷ $p_i = \mathrm{sigmoid}(-2\tilde{h}_i\sigma_i/T(s))$
5: Generate a random number $rand = [0, 1)$
6: **if** $(p_i > rand)$ **then**
7: $\sigma_i = -\sigma_i$

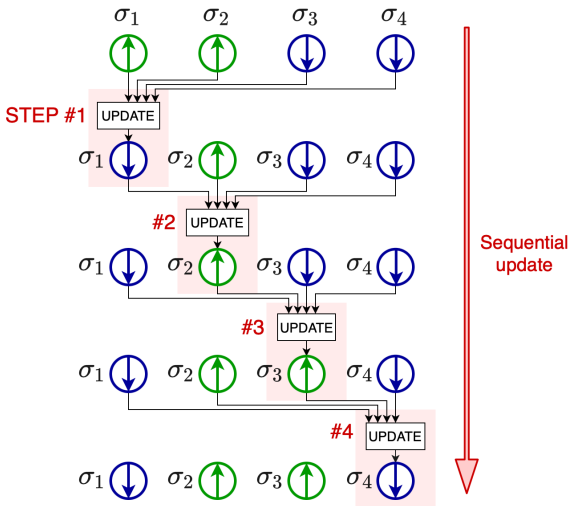


図 1: 全結合 4 スピンモデルを対象とした SA のスピン更新処理の様子．本図は単一アニーリングステップの更新を表す．

与えられる辺集合 E によっては，最大カット問題を表現するために全結合イジングモデルが必要となる．この場合，イジングモデルを局所結合のトポロジに制限しているイジングマシンで問題を解くことは非常に難しい．

2.3 SA（Simulated Annealing）

SA [13] は，Glauber ダイナミクスに基づいてスピンの更新を行ないながら，解空間を探索する手法である．SA では，低質な局所解に捕らわれることを防ぐために，疑似温度 T を導入して状態遷移確率を制御する．SA のアルゴリズムを **Algorithm 1** に示す．本アルゴリズムは，2-7 行目に示すスピン更新処理の繰り返しで構成される．スピン更新処理は，局所場計算（3 行目），状態遷移確率計算（4 行目），スピンの反転試行 (5-7 行目)，の 3 処理に細分される．スピンの反転試行では，遷移確率 p_i と一様分布に従う乱数との比較結果に応じて，スピン σ_i を反転させる．SA において，疑似温度 T は一般に，十分に高い値から十分に低い値へと減少するよう制御される．本稿では，初期温度 T_{init}

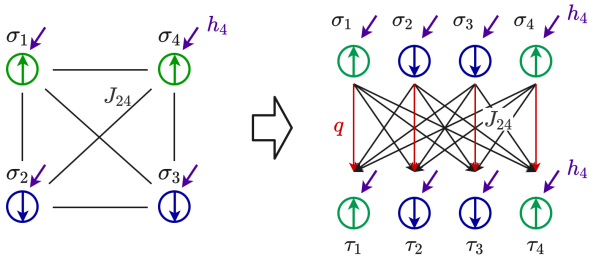


図 2: イジングモデルと PCA モデルの対応関係.

と減衰率 r_T $(0 < r_T < 1)$ に応じて $T(s) = T_{\text{init}} \times r_T^{s-1}$ として制御する.

全結合イジングモデルを対象とする場合，**Algorithm 1** の 3 行目で計算されるスピン σ_i の局所場 $(\tilde{h}_i)$ は，他の全てのスピン状態に影響を受けることから，遷移確率 p_i の計算は全てのスピン状態に依存する．従って，図 1 に示されるように，全結合イジングモデルを対象とする SA ではスピン更新は逐次的に実行される．この逐次的なスピン更新により，GPU などの並列計算エンジンを用いた SA の高速化は原理的に困難である.

3. SCA（Stochastic Cellular Automata Annealing）

全スピンの並列更新を可能とすることで，全結合イジングモデルを対象に高速なアニーリングを実現する手法が提案されている [4,14]．本研究では，最先端のアニーリングプロセッサ LSI である STATICA [6,7] の動作原理として用いられている SCA [4] に注目する.

3.1 PCA に基づくイジングモデルのサンプリング

SCA は，イジングモデルの効率的なサンプリングを可能にする確率的セルオートマトン（PCA）[12] を基にして開発されたアニーリング手法である．PCA では，イジングモデルの各スピン σ_i に対して，その次状態を表わすスピン τ_i を用意し，図 2 に示すように $\boldsymbol{\sigma}$ から $\boldsymbol{\tau}$ を生成するモデルへと拡張する．併せて，ハミルトニアンを次式のように書き換える.

$$H(\boldsymbol{\sigma},\boldsymbol{\tau}) = -\frac{1}{2}\sum_{i\neq j} J_{ij}\sigma_i\tau_j - \sum_i h_i\frac{\sigma_i+\tau_i}{2} + q\sum_i \frac{1-\sigma_i\tau_i}{2}. \tag{3}$$

ここで，J_{ij} はイジングモデルの J_{ij} と等しい σ_i と τ_j の相互作用を示し，q $(q > 0)$ は罰則の大きさを表す.

PCA モデルでは，$\boldsymbol{\sigma}$ 間に相互作用がなく，また $\boldsymbol{\tau}$ 間でも同様であるため，全スピンを並列に更新することができる．式 (3) 右辺の第 3 項は各スピンの反転を抑制する効果があり，全てのスピンが反転しないとき（すなわち $\boldsymbol{\sigma} = \boldsymbol{\tau}$ を満足するとき）に $H(\boldsymbol{\sigma},\boldsymbol{\tau})$ が $H(\boldsymbol{\sigma})$ に等しくなる．文献 [12] は，$q \to \infty$ かつ $N \to \infty$ を仮定したとき PCA の定常分布が元のイジングモデルのギブス分布に等しくなることを証明している．このことから，PCA により効率的なイジングモデルのサンプリングが実現されるものと期待される.

Algorithm 2 Stochastic Cellular Automata Annealing

Input: initial spin configuration ($\boldsymbol{\sigma}$),
spin-spin interaction ($\boldsymbol{J}$), external magnetic field ($\boldsymbol{h}$),
# of annealing steps (S),
pseudo temperature scheduling ($T(s)$), penalty (q)
Output: optimized spin configuration ($\boldsymbol{\sigma}$)

1: **for** $s = 1$ to S **do**
2: **for** $i = 1$ to N **do**
3: Calculate local field
 ▷ $\tilde{h}_i = \sum_j J_{ij}\sigma_j + h_i$
4: Calculate transition probability
 ▷ $p_i = \text{sigmoid}(-(\tilde{h}_i\sigma_i + q)/T(s))$
5: Generate a random number $rand = [0, 1)$
6: **if** $(p_i > rand)$ **then**
7: $\tau_i = -\sigma_i$
8: **else**
9: $\tau_i = \sigma_i$
10: **for** $i = 1$ to N **do**
11: $\sigma_i = \tau_i$

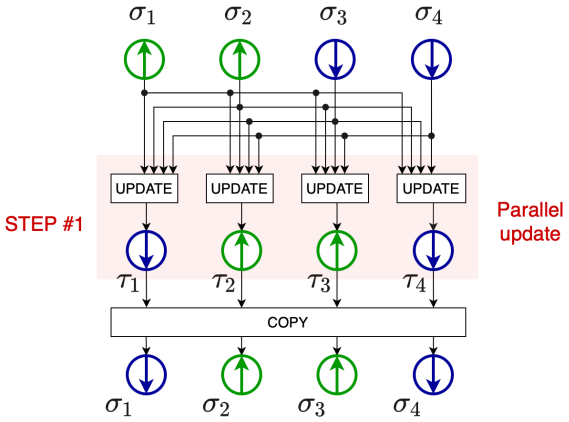


図 3: 全結合 4 スピンモデルにおける SCA の単一アニーリングステップで実行されるスピン更新処理.

3.2 SCA アルゴリズム

SCA は，PCA の概念を取り入れることによって，全結合イジングモデルの並列スピン更新を可能とする．SCA のアルゴリズムを **Algorithm 2** に示す．SCA のアルゴリズムは，SA と同様にスピン更新処理の繰り返しで構成されるが，各スピンの更新が独立であり並列に実行可能である．スピンの更新処理（3-9 行目）において，特にスピンの反転試行が SA と異なり，更新後のスピン状態を $\boldsymbol{\sigma}$ ではなく $\boldsymbol{\tau}$ に記録する（図 3）．ここで，$\boldsymbol{\tau}$ 間に相互作用はないため，全ての τ_i のスピン状態を並列に決定可能である．全てのスピンを更新後，次のアニーリングステップに備えて全ての τ_i を σ_i にコピーする（10-11 行目）.

SCA では，擬似温度 T だけでなく，罰則の大きさ q の制御が必要となる．T は SA と同様の役割を担うため，SCA

表 1: **Algorithm 2** における局所場 $(\tilde{h}_i)$ の計算.

単一インスタンス実行	$\begin{bmatrix} \tilde{h}_1 \\ \vdots \\ \tilde{h}_N \end{bmatrix} = \begin{bmatrix} J_{11} & \dots & J_{1N} \\ \vdots & \ddots & \vdots \\ J_{N1} & \dots & J_{NN} \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \vdots \\ \sigma_N \end{bmatrix} + \begin{bmatrix} h_1 \\ \vdots \\ h_N \end{bmatrix}$
多重インスタンス実行	$\begin{bmatrix} \tilde{h}_1^{(1)} & \dots & \tilde{h}_1^{(M)} \\ \vdots & \ddots & \vdots \\ \tilde{h}_N^{(1)} & \dots & \tilde{h}_N^{(M)} \end{bmatrix} = \begin{bmatrix} J_{11} & \dots & J_{1N} \\ \vdots & \ddots & \vdots \\ J_{N1} & \dots & J_{NN} \end{bmatrix} \begin{bmatrix} \sigma_1^{(1)} & \dots & \sigma_1^{(M)} \\ \vdots & \ddots & \vdots \\ \sigma_N^{(1)} & \dots & \sigma_N^{(M)} \end{bmatrix} + \begin{bmatrix} h_1 & \dots & h_1 \\ \vdots & \ddots & \vdots \\ h_N & \dots & h_N \end{bmatrix}$

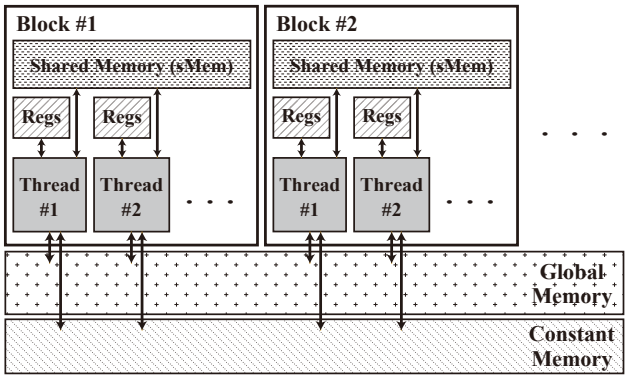


図 4: GPU の実行モデル.

においても SA と同様に $T(s) = T_{\text{init}} \times r_T^{s-1}$ として制御する．一方，q については，文献 [4] にて数学的な解析がなされており，q が $-\boldsymbol{J}$ の最大固有値の半分以上であれば SCA が SA と同一の基底状態に到達可能であることが示されている．本稿の計算機実験では，文献 [6] の実験に倣い，q を適当な大きさの値に固定してアニーリングを実行する．

4. GPU 実装

SCA はアルゴリズムレベルで高い並列性を有することから，GPU や FPGA などの並列計算エンジンによるスピン更新処理の効率化が見込まれる．本稿では，SCA を対象としたときの最も効率的な実装方法を探究するため，複数の GPU 実装方法を比較，評価する．図 4 に GPU の実行モデルを示す．GPU 実装では，計算資源（スレッド）とメモリ資源（グローバルメモリ・共有メモリ）を効率的に利用できるようにすることが重要となる．

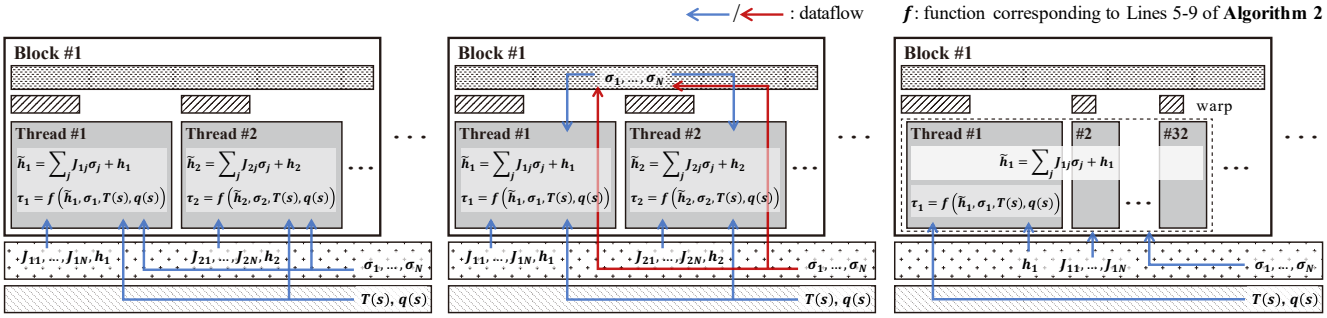
本章では，アニーリングを 1 回のみ実行する場合（単一インスタンス実行）と，同一パラメタ下でアニーリングを複数回実行する場合（多重インスタンス実行）の両方を想定し，GPU 実装方法を検討する．アニーリング計算には確率的な動作が含まれることから，良質な解を得るために複数回のアニーリングが求められることがあり，その意味で多重インスタンス実行の意義がある．多重インスタンス実行では各インスタンスがそれぞれ独立に実行されるため，並列計算が可能であり，GPU 上でのより一層の高速化が期待される．

4.1 単一インスタンス実行

SCA の最も単純な GPU 実装方法として，単一スピンの更新処理を 1 スレッドに割り当てる方法が考えられる（**Impl.1** と呼称）．**Impl.1** では，図 5(a) に示すように，カーネルの呼び出し毎に N スレッドを用い，単一のスピン更新（**Algorithm 2** の 3-9 行目に相当）を各スレッドが処理する．このとき，SCA の並列スピン更新を可能とする性質から，カーネル内での同期は不要である．カーネル実行後は $\boldsymbol{\sigma}$ と $\boldsymbol{\tau}$ の配列のポインタを交換することで 10-11 行目に相当する処理を実行し，同じ要領で次のカーネルを呼び出す．

Impl.1 の高速化を考えると，スピン更新処理で最も時間を要する局所場 $(\tilde{h}_i)$ 計算の実行時間を削減することが求められる．表 1 の上段に示すように，局所場計算は独立に実行される一方，スピンの現状態（$\sigma_1, \cdots, \sigma_N$）は全スピンの更新に共通で利用される（図 5(a)）．そこで，GPU の共有メモリ（sMem: shared memory）を活用して同一ブロックに属するスレッド間で頻繁に使われるデータを共有し，グローバルメモリへのアクセス回数を削減する．つまり，**Impl.2** では，ブロック毎にスピンの現状態をグローバルメモリから sMem へと読み出す処理（図 5(b) の赤い矢印に相当する処理）を追加する．その後の処理は，スピンの現状態をグローバルメモリではなく sMem から取得する点を除き，**Impl.1** と同様の手順となる．

別の実装方法として，並列リダクションを利用して局所場計算を高速化することが考えられる．並列リダクションとは，複数スレッドを用いて並列に配列要素の総和を計算する手法である．図 6 は並列リダクションを用いて配列要素の総和を計算する手順の一部を示しており，32 スレッドを用いて 64 要素の総和が並列に計算されている．昨今の GPU は，32 スレッドを 1 ワープと呼びワープ単位で動作させる仕組みを採用していることから，図 6 の一連の処理はスレッド間同期を必要としない．ここでは，**Impl.3** として，単一スピンの更新処理を 1 ワープに割り当てる実装方法を検討する．**Impl.3** では，図 5(c) に示すように，カーネル呼び出し毎に $32 \times N$ スレッドを用い，単一スピンに対する局所場計算を 32 スレッドで実行する．なお，並列リダク



(a) **Impl.1** (b) **Impl.2** (c) **Impl.3**

図 5: SCA の GPU 実装方法．本図には各実装方法における計算資源とメモリ資源の利用方法が図示されている．

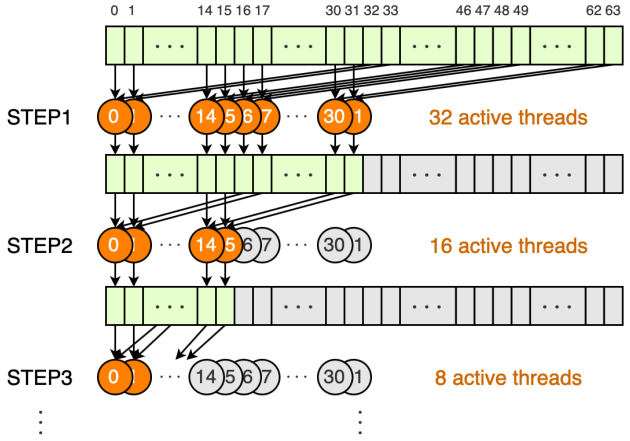


図 6: 64 要素配列に対する並列リダクションを利用した総和計算（32 スレッドを使用）．

ションの前処理として，N 要素を 64 要素 ($l = 1, 2, \cdots, 64$) に圧縮する処理 ($\sum_{j=64k+l} J_{ij}\sigma_j$ $(0 < k < \lceil N/64 \rceil)$) を併せて導入する．**Impl.3** の場合，局所場計算以外のスピン更新処理はワープ中の 1 スレッドのみで実行される．

4.2 多重インスタンス実行

多重インスタンス実行では，同一パラメタのもとで M 回のアニーリングを実行させる．各アニーリング処理は独立であることから，SCA においては各アニーリングステップで $M \times N$ スピンを並列に更新するような実行方法が考えられる．これを GPU に実装する場合，単一インスタンス実行と同様の方法で実現できる．つまり，**Impl.1** と **Impl.2** ではカーネル呼び出し毎に $M \times N$ スレッドを用い，**Impl.3** ではカーネル呼び出し毎に $32 \times M \times N$ スレッドを用いる．

多重インスタンス実行において，m 番目のインスタンスの σ_i，$\tilde{h}_i$ を $\sigma_i^{(m)}$，$\tilde{h}_i^{(m)}$ として表すと，局所場の計算は表 1 下段のように行列積の形で表現される．GPU 上で行列積を効率的に計算する方法として cuBLAS ライブラリ [11] を用いることが考えられる．ここでは，**Impl.4** として，cuBLAS ライブラリを活用した行列積の実装を試みる．**Impl.4** では，スピン更新処理を前半と後半に分け，前半の局所場計算を cuBLAS を用いて計算し，それ以外のスピン更新処理を $M \times N$ スレッドで処理する．

表 2: 単一インスタンス実行におけるアニーリング時間とスピン更新効率の比較．

Prob.	Algo.	Impl.	time [ms]	update/us
K2000	SA	CPU	461.09	4.338
		GPU	7,247.19	0.276
	SCA	GPU (**Impl.1**)	60.58	33.103
		GPU (**Impl.2**)	45.92	43.556
		GPU (**Impl.3**)	9.60	208.272
G61	SA	CPU	6,741.80	1.039
		GPU	94,401.63	0.074
	SCA	GPU (**Impl.1**)	490.42	14.274
		GPU (**Impl.2**)	285.94	24.490
		GPU (**Impl.3**)	110.25	63.499

5. 実験結果

4 章で述べた SCA の GPU 実装方法を CUDA C++を用いて実現し，単一の GPU ボード（NVIDIA Tesla-V100）上で実行させたときの性能を評価した．さらに，SA と SCA のスピン更新効率を比較する目的で SA アルゴリズムを CUDA C++と C++の両方で実装し，それぞれ Tesla-V100 と Intel Xeon W-2255 CPU で実行させた．2.3 節で説明した通り SA アルゴリズムに並列性はないものの，局所場計算は **Impl.3** と同様の方法で並列化が可能である．従って，SA の GPU 実装では 32 スレッドを用いて単一スピンの更新をおこなう．本実験では，ベンチマークとして 2 種類の最大カット問題，K2000（全結合 2,000 ノード）[2] および G61（疎結合 7,000 ノード）[1] を使用した．

5.1 単一インスタンス実行におけるスピン更新効率

表 2 は，K2000 と G61 に対する単一インスタンス実行のアニーリング時間とスピン更新効率を示す．表 2 では，アニーリングステップ数を $S = 1,000$ で固定した場合の SA の CPU 実行時間，GPU 実行時間，ならびに SCA の各実装方法に対する GPU 実行時間を比較している．

表 3: 多重インスタンス実行におけるアニーリング時間とスピン更新効率の比較（M：インスタンス数）.

Prob.	Algo.	Impl.	time/instance [ms]			update/us		
			$M=32$	$M=64$	$M=128$	$M=32$	$M=64$	$M=128$
K2000	SA	CPU	459.04	458.80	510.02	4.357	4.359	3.921
		GPU	242.30	136.00	74.26	8.257	14.707	26.933
	SCA	GPU (**Impl.1**)	4.79	4.67	4.22	419.350	429.630	473.988
		GPU (**Impl.2**)	3.44	3.35	2.88	583.072	599.435	695.287
		GPU (**Impl.3**)	4.17	3.99	3.88	482.046	502.587	516.326
		GPU (**Impl.4**)	1.90	0.97	0.56	1,054.810	2,067.387	3,602.913
G61	SA	CPU	5,412.98	5,388.04	5,520.75	1.293	1.299	1.268
		GPU	3,088.72	1,562.27	796.89	2.266	4.481	8.784
	SCA	GPU (**Impl.1**)	73.07	76.07	93.80	95.840	92.021	74.623
		GPU (**Impl.2**)	70.24	77.63	92.25	99.654	90.173	75.877
		GPU (**Impl.3**)	97.67	97.40	97.24	71.668	71.870	71.988
		GPU (**Impl.4**)	21.93	11.52	4.98	319.432	607.653	1,404.776

表 2 の結果から，単一インスタンス実行では SA の GPU 実行が極めて非効率であることが確認された．一方，SCA は GPU 上で効率的に処理されている．SCA を実装方法別で見ると **Impl.3** が最大の性能を示しており，SA を CPU 上で実行した場合と比較して K2000 で 48 倍，G61 で 61 倍の高速化効果が得られた（図 7）．特に N が小さい場合，**Impl.1** および **Impl.2** の N 並列計算では GPU の計算資源を十分に利用できず，**Impl.3** が優位となる．実際に，G61 と比較して K2000 の結果は，**Impl.1** に対する **Impl.3** の速度向上比が大きくなっている．

全結合イジングモデルを対象としたアニーリングにおいて，SA は単一ステップで単一スピンを更新するのに対し，SCA は単一ステップで N スピンの更新処理を並列に実行する．このことから，SCA は原理上，対 SA 比で約 N 倍の高速化が期待される．ところが，SCA の **Impl.3** の結果は，SA の GPU 上での実行結果に比べて K2000 で 755 倍，G61 で 856 倍の高速化に留まった．これは，カーネル呼び出しの際に起動するスレッド数（$32 \times N$）に対して GPU 内の演算器が不足しており，スピン更新処理を時間方向に分割しているためであると考えられる．

5.2 多重インスタンス実行におけるスピン更新効率

表 3 は，K2000 と G61 に対する多重インスタンス実行のアニーリング時間とスピン更新効率を示す．表 3 では，アニーリングステップ数を $S = 1,000$，インスタンス数を $M = 32, 64, 128$ とした場合の SA の CPU 実行時間，GPU 実行時間，ならびに SCA の各実装方法に対する GPU 実行時間を比較している．なお，表中のアニーリング時間は実際のアニーリング時間を 1 インスタンスあたりに換算して表示している．

表 3 の結果から，多重インスタンス実行では SA と SCA の両方で GPU による高速化効果が確認された．SCA を実装方法別で見ると **Impl.4** が最大性能を示し，$M = 128$ と

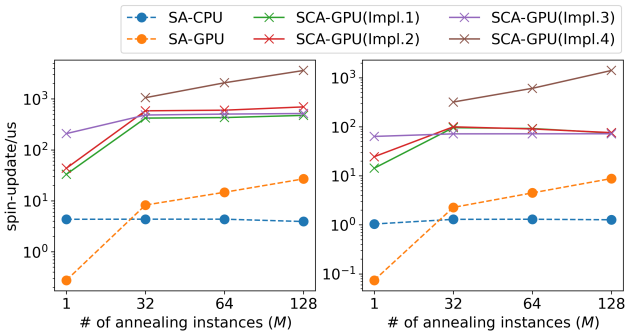


図 7: スピン更新効率の比較（左：K2000, 右：G61）.

して多重実行した結果は，SA を GPU 上で実行した場合と比較して K2000 で 132.6 倍，G61 で 160 倍の高速化効果が得られた（図 7）．一方で，単一インスタンス実行とは異なり，**Impl.3** の他の実装方法に対する優位性は見られなかった．また，**Impl.3** では，$M = 128$ の結果が $M = 32$ とした場合と比べて K2000 で 1.07 倍，G61 で 1.00 倍の高速化に留まり，インスタンス数を増加することによる性能向上が確認されなかった．これら **Impl.3** の結果は，$M \times N$ スピンの更新処理を同時実行する場合，GPU 内の並列演算器数の不足により時間方向に分割する必要があるためと考えられる．

5.3 GPU による SCA の高速化性能評価

アニーリングの性能は，アニーリング時間とその結果得られた解の品質の双方から決められるものであるため，ここでは文献 [5] で用いられている TTT（Time To Target）と呼ばれる性能指標を用いてこれを評価する．TTT とは，規定水準以上の品質の解を確率 P で得るために必要な時間を表す．本稿では，最大カット問題における解の品質をカット値により表すとき，既知の最適値（K2000：33,337, G61：5,798）の 99%を規定水準として TTT を算出した [5]．また，本評価では $P = 0.99$ と定めた．

表 4: TTT による評価結果の比較.

Prob.	Algo. / Impl.	TTT [ms]
K2000	SCA / GPU (**Impl.4**)	1.58
	SCA / ASIC : STATICA [7]	1.50
	bSB / FPGA : SBM [5]	0.26
	bSB / GPU [5]	18.70
	dSB / GPU [5]	7.66
G61	SCA / GPU (**Impl.4**)	101.93
	dSB / GPU [5]	71.50

K2000 と G61 に対して SCA を GPU 上で実行（**Impl.4**, $M = 128$）したときの TTT を，いくつかの先行研究の結果とともに表 4 に示す．なお，dSB および bSB は，文献 [5] で提案された最先端のイジングモデルの基底状態探索手法である．K2000 に対する TTT に注目すると，SCA の GPU アクセラレータは FPGA 上に実装された SBM [5] や専用 LSI として実装された STATICA [6] に劣る結果となっているが， Tesla-V100 上で実行された bSB および dSB には優る結果となっており，GPU 上で実行される SCA が高い性能を示すことを確認できる．一方，G61 に対する TTT に注目すると，同一 GPU で実行しているにもかかわらず，SCA の結果が dSB に劣る結果となった．今後の研究方針として，局所場計算に工夫を加えることが考えられる．STATICA [6] では，スピン更新結果にもとづいて局所場を差分更新する仕組みを用いており，この仕組みを活用することで SCA の GPU 上での実行もより一層高速化できるものと期待される．

6. おわりに

本稿では，全結合イジングモデルに対して全並列で状態更新可能なアニーリング手法 SCA に注目し，単一インスタンス実行と多重インスタンス実行の 2 つの場合で，GPU アクセラレーション方法を検討・評価した．SCA の単一インスタンス実行では，並列リダクションを活用することで小規模な問題においても GPU の計算資源を効率的に利用可能となった．SCA の多重インスタンス実行では，行列積に注目した cuBLAS ライブラリの活用により顕著な性能向上が見られ，対 SA 比で最大 160 倍のスピン更新効率の向上を達成した．TTT を用いた評価結果から，GPU を用いることで SCA を極めて効率的に実行可能であることを確認した．本研究の今後の展望として，5 章で述べた通り，局所場の差分更新を用いた SCA のさらなる高速化が考えられる．また，SCA により得られる解品質の向上を目的とし，本稿で提案した多重インスタンス実行を発展させることが考えられる．

謝辞 本研究の一部は JST CREST JPMJCR18K3 の支援を受けて実施したものである．

参考文献

[1] The G-set benchmark, `https://web.stanford.edu/~yyye/yyye/Gset/` (2003).
[2] Simulated annealing for complete graphs, `https://github.com/hariby/SA-complete-graph/tree/WK2000` (2016).
[3] A. Lucas: Ising formulations of many NP problems, *Frontiers in Physics*, Vol. 2, pp. 5.1–15 (2014).
[4] B. H. Fukushima-Kimura, S. Handa, K. Kamakura, Y. Kamijima and A. Sakai: Mixing time and simulated annealing for the stochastic cellular automata (2020).
[5] H. Goto, K. Endo, M. Suzuki, Y. Sakai, T. Kanao, Y. Hamakawa, R. Hidaka, M. Yamasaki and K. Tatsumura: High-performance combinatorial optimization based on classical mechanics, *Science Advances*, Vol. 7, No. 6, pp. 1–9 (2021).
[6] K. Yamamoto, K. Ando, N. Mertig, T. Takemoto, M. Yamaoka, H. Teramoto, A. Sakai, S. Takamaeda-Yamazaki and M. Motomura: STATICA: A 512-spin 0.25M-weight full-digital annealing processor with a near-memory all-spin-updates-at-once architecture for combinatorial optimization with complete spin-spin interactions, *2020 IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 138–140 (2020).
[7] K. Yamamoto, K. Kawamura, K. Ando, N. Mertig, T. Takemoto, M. Yamaoka, H. Teramoto, A. Sakai, S. Takamaeda-Yamazaki and M. Motomura: STATICA: A 512-spin 0.25M-weight annealing processor with an all-spin-updates-at-once architecture for combinatorial optimization with complete spin–spin interactions, *IEEE Journal of Solid-State Circuits*, Vol. 56, No. 1, pp. 165–178 (2021).
[8] M. Aramon, G. Rosenberg, E. Valiante, T. Miyazawa, H. Tamura and H. G. Katzgraber: Physics-inspired optimization for quadratic unconstrained problems using a digital annealer, *Frontiers in Physics*, Vol. 7, pp. 48:1–14 (2019).
[9] M. W. Johnson, M. H. S. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, P. Bunyk, E. M. Chapple, C. Enderud, J. P. Hilton, K. Karimi, E. Ladizinsky, N. Ladizinsky, T. Oh, I. Perminov, C. Rich, M. C. Thom, E. Tolkacheva, C. J. S. Truncik, S. Uchaikin, J. Wang, B. Wilson and G. Rose: Quantum annealing with manufactured spins, *Nature*, Vol. 473, No. 7346, pp. 194–198 (2011).
[10] M. Yamaoka, C. Yoshimura, M. Hayashi, T. Okuyama, H. Aoki and H. Mizuno: A 20k-spin Ising chip to solve combinatorial optimization problems with CMOS annealing, *IEEE Journal of Solid-State Circuits*, Vol. 51, No. 1, pp. 303–309 (2016).
[11] NVIDIA Corporation: cuBLAS::CUDA toolkit documentation, `https://docs.nvidia.com/cuda/cublas/index.html` (2021).
[12] P. Dai Pra, B. Scoppola and E. Scoppola: Sampling from a Gibbs measure with pair interaction by means of PCA, *Journal of Statistical Physics*, Vol. 149, No. 4, pp. 722–737 (2012).
[13] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi: Optimization by simulated annealing, *Science*, Vol. 220, No. 4598, pp. 671–680 (1983).
[14] T. Okuyama, T. Sonobe, K. Kawarabayashi and M. Yamaoka: Binary optimization by momentum annealing, *Physical Review E*, Vol. 100, No. 1, pp. 012111.1–9 (2019).