Vol.2012-GI-27 No.7
2012/3/2

対称解の特性を用いた N-Queens 問題の求解と GPU による高速化

萩野谷 一二　田中 慶悟*　藤本 典幸*

最近，N-Queens 問題への新しい取組が行われている．1 つは，田中らの GPU を用いた高速化の提案であり，もう 1 つは，萩野谷の新しい解法アルゴリズム「部分解合成法」の提案である． 本論文では，田中らの提案手法をベースに，部分解合成法における「代表解の選択条件」の取り込みを行った．NVIDIA GeForce GTX580 と Intel Core i7 2600 3.4GHz CPU を使用した評価実験では，N=23 の問題の求解時間は，12 時間 43 分 39 秒と N=24 の世界記録を 2003 年に樹立した電通大の PC クラスタ（Intel Pentium4 Xeon 2.8GHz CPU x 68）の約 4.5 倍高速であった．

Solving the N-Queens Problem with properties among Symmetric Solutions and its GPU Acceleration

Kazuji HAGINOYA, Keigo TANAKA*, and
Noriyuki FUJIMOTO*

Recently, research on the N-Queens problem has evolved in two directions. One is using a GPU proposed by Tanaka et al., and the other is a novel sequential algorithm, named 'subsolution composition method', proposed by Haginoya. This paper presents a new GPU algorithm that adopts 'selection of representative solutions' from the subsolution composition method. The experiments on NVIDIA GeForce GTX580 GPU and Intel Core i7 2600 3.4GHz CPU show that the proposed algorithm solves 23-queens problem for 12 hours, 43 minutes, and 39 seconds. The time is about 4.5 times faster than the PC cluster （Intel Pentium4 Xeon 2.8GHz CPU x 68） at the university of electro-communications, which established the world record for 24-queens problem in 2003.

* 大阪府立大学　大学院理学系研究科 情報数理科学専攻
Department of Mathematics and Information Sciences,
Graduate School of Science, Osaka Prefecture University

1. はじめに

N-Queens 問題とは，N x N のチェス版上で，互いに攻撃し合わないような N 個のクイーンの配置の総数を求める問題である．現時点では，N=26 までの解が求められている．N=27 以降の解を求めるにはより一層の高速化が必要とされる．なお，N-Queens 問題の最新の動向については文献[1]を参照されたい．

1.1 用語の説明

Somers のアルゴリズム[3]

下記のスタック情報 u, r, l, bitmap を使って，0 行から順にクイーンを配置していくアルゴリズムである．x 行の情報から，（x+1）行の情報を定める手順は以下のようになる．

```
bit = －bitmap[x] & bitmap[x]        //（x+1）行目のクイーンの位置を選択
bitmap[x] ^= bit                     //選択したクイーンの位置を候補から消去
u[x+1] =（u[x] | bit） << 1   //（x+1）行目の右斜め下方向のビットマップの作成
r[x+1] =  r[x] | bit          //（x+1）行目の真下方向のビットマップの作成
l[x+1] =（l[x] | bit） >> 1   //（x+1）行目の左斜め下方向のビットマップの作成
bitmap[x+1] = mask & ~（u[x] | r[x] | l[x]）
                     //（x+2）行目のクイーンの位置候補のビットマップの作成
x++                  //スタックを 1 段深くする（次の行の準備）
```

但し，u[0]=r[0]=l[0]=0, bitmap[0]=mask = $2^N - 1$ とする．

また，解の対称性を利用して探索量を半分に削減している．

スタック情報：0 行～x 行に配置したクイーンの位置を管理する情報．

u[x] ：クイーンの右斜め下方向への効き筋を表すビットマップ
r [x] ：クイーンの真下方向への効き筋を表すビットマップ
l [x] ：クイーンの左斜め下方向への効き筋を表すビットマップ
u, r, l のビットマップは，0：効き筋にクイーンなし 1 ：効き筋にクイーン有りとする．
bitmap[x]：クイーンを配置可能な位置を表すビットマップ
0 はクイーンの配置不可，1 はクイーンの配置可能とする．

スタックの深さ：GPU 側でクイーンを探索する時必要な行数．

CPU 側でどの深さまで探索するか，終了判定をどの時点で行うかなどによって変動

する．田中らの提案手法の場合，m 行を CPU 側で計算しているため，m 行～（N-1）行の深さ（=N-m）となる．

スタック域：スタック情報 × スタックの深さの配列．
GPU では，関数の再帰呼び出しができないのでスタックを利用したループ制御となる．（GPU のグローバルメモリはアクセス速度が非常に遅いため）スタック域は，高速な共有メモリに配置する．しかし，共有メモリは小容量であるためスレッドの多重度を制約する要因となっている．

解の区分：よく知られているように，N-Queens 問題の解は，上下反転・左右反転・回転操作の組合せによって自身の解も含めて８～２個の対称解が得られる．文献[2]に倣って，8 つの解を得られるものを type-a（非回転対称解），4 つの解を得られるものを type-b（180°回転対称解），2 つの解を得られるものを type-c（90°回転対称解）とする．また，対称解を生成する元となった解を代表解と定義する．代表解の選び方は任意であるが，性能面を考えると探索アルゴリズムと整合性のよい選択条件を採用することは重要である．

Seed：CPU 側で探索を行った結果を引き継いで GPU 側のスレッドで解の探索を行うための情報．（Seed は田中らの提案手法の“タスクの入力データ”に相当する）当然の事ではあるが，異なる Seed から得られるそれぞれの解の集合の間に共通の解は存在しない．また，解の総数は，すべての Seeds から得られた解の総和となる．Somers アルゴリズムの場合，u,r,l,bitmap の 4words が標準的な情報となる．ただし，bitmap は，u,r,l から求めることができるので必須情報ではなく，後述の Seeds 削減の施策では 4word 目を別の情報に使用している．
解の区分と同様に，Seed もいくつかの対称解を代表する以下の 3 種の Seed に区分する．

Seed-A: type-a の代表解を生成する Seed.
解の総数を求める際，探索で得られた解の数を 8 倍して集計する．

Seed-B: 左右反転，斜軸反転（上下反転＋90°回転）により生成される解を代表する Seed．解の総数を求める際，探索で得られた解の数を 4 倍して集計する．
type-b の代表解を生成する Seed だけでなく，type-a の 8 つの対称解を 2 つの Seeds で代表するケースもある．

Seed-C: 左右反転により生成される解を代表する Seed．解の総数を求める際，探索で得られた解の数を 2 倍して集計する．type-c の代表解を生成する Seed だけでなく，type-a の 8 つの対称解を 4 つの Seeds で代表するケースや，type-b の 4 つの対称解を 2 つの Seeds で代表するケースもある．

1.2 開発環境と評価マシン

今回の評価で使用した環境は以下の表 1 の通りである．以後，文中で特に断らない限り，この環境で測定されたものである．

表 1　開発環境と評価マシン

【PC】	【GPU】NVIDIA GeForce GTX580
CPU:Intel Core i7 2600 3.4GHz CPU	MP数：16
キャッシュ容量: 8MB	コア数：512
メモリ容量: 4GB	MPclock数：1.54GHz
グラフィックスドライバ: 270.81	グローバルメモリ容量：1.5GB
OS :Windows7 Home premium （32ビット版）	共有メモリ容量: 48KB/MP
【開発環境】	
コンパイラ: Visual C++ 2008 Express Edition	
CUDA：NVIDIA GPU Computing SDK 3.2	

なお，GPU プログラミングの留意点については文献[5][6][7]を参照されたい．

2. 関連研究と提案手法の概要

最近の N-Queens 問題の関連研究として２つの取り組みがある．１つは，GPU を用いた高速化の試み[1]であり，もうひとつは，新しい解法アルゴリズム「部分解合成法」の提案[2]である．

2.1 GPU を用いた高速化の試み

GPU を用いた高速化の試みは，まだ少なく，田中らの研究[1]が最新の取組と思われる．田中らの提案手法の概要は，以下の通りである．

CPU 側の処理：
①CPU 側で m 行までのクイーンの配置を求め，その位置情報を N 進数と見て符号化を行う．ただし，解の対称性を考慮し符号化は半分としている．
②符号化したクイーンの位置情報は一括して，GPU の VRAM に転送する．
③GPU を起動し，GPU 側の解の探索完了を待つ．
④GPU の探索完了後，VRAM に格納された解の数を 2 倍し，解の総数を求める．

GPU 側の処理：
⑤各スレッドは VRAM に格納されたそれぞれの符号化データを受け取り，
⑥その符号化データを元のクイーンの位置情報に復元し，

⑦Somers のアルゴリズムにより解の探索を行う．探索の深さは（N-m）行となる．
⑧解の探索が終了すると次の符号化データを動的（スレッドの到着順）に取り出して，⑥，⑦の処理を継続する．
⑨すべての符号化データの処理が完了したら，解の数を VRAM に格納し，処理を終了する．

この**提案のポイント**は，
（a） クイーンの位置情報の符号化による CPU/GPU 間の通信量の最小化
（b） 多数スレッドに対して，符号化データを動的に配分し負荷の均等化を図る
（c） 共有メモリのバンク衝突の防止による性能向上
にある．

2.2 新アルゴリズム「部分解合成法」の提案

萩野谷の部分解合成法[2]では，Somers アルゴリズムとは全く別のアルゴリズムが提案されている．その中核は，①解の類別とその判定方法（代表解の選択条件），および，②部分解から全体解を合成する手順の提示にある．以下では，本提案手法で必要となる①について概要を説明する．

【代表解の選択条件】（クイーンの数が奇数の場合）

クイーンの数（N）が奇数の場合，中央の行と列に配置されたクイーンの位置によって，図 1 のように，解を type Ⅰ，type Ⅱ に分類する．

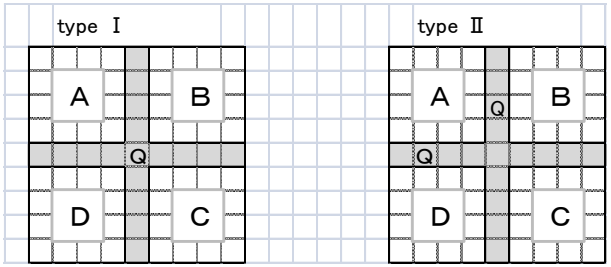


図 1 クイーン数が奇数の場合の解の分類

type Ⅰ は，クイーンが中央に 1 つ配置された形である．type Ⅰ には，type-a~type-c の解が含まれる．（その判別方法は，文献[2]を参照）

type Ⅱ は，2 つのクイーンが中央以外の位置に配置された形である．この場合は type-a の解しか存在しない．

type Ⅱ の解の代表解の選択条件は，n=（N-1）/2，2 つのクイーンの位置を （n,I），（J,n） とすると，**I＜J＜n を満たす解を代表解**と定義する．

N-Queens 問題の解の殆どが type-a であることに注目すべきである．Somers アルゴリズムでは，左右反転解を考慮して探索量を半減しているが，部分解合成法では，代表解の選択条件によって探索量を約 1/8 に削減している．（**探索量で 4 倍の差**は極めて大きい）

2.3 提案手法 の概要

本提案手法は，クイーンの数を奇数に限定し，Somers アルゴリズムをベースとする田中らの提案手法に部分解合成法の「代表解の選択条件」を取り込み，探索量の大幅削減を図ることを目指した．

主な取り組みは，Somers アルゴリズムに適合するように設計した以下の機能にある．

① （I,J）フィルタ （「代表解の選択条件」を取込み）
まず，中央行までの探索を行い，上記の代表解の選択条件を適用する．これにより，以降の探索量を約 1/8 に削減する．

② CPU/GPU の役割分担の最適化
前半の複雑な探索処理（代表解の選択条件の適用）は CPU 側で行い，後半の単純・大量の探索処理を GPU 側で行う構造とする．これにより，GPU 側の探索が浅くなり，共有メモリの使用量が削減されるので，スレッドの多重度を上げることができる．また，if 文の少ない処理を GPU にオフロードする事も性能面では重要である．

③ CPU/GPU の並行処理
上記の CPU 側の処理と GPU 側の処理を並行して実行する方式とする．これにより，CPU 側の処理時間は，GPU 側の処理時間に隠れて見えなくなる．

④ 共有メモリ使用量の削減
64 ビットレジスタを使用して，スタック情報を 4words から 2words に削減する．これにより，スレッドあたりの共有メモリの使用量が半減し，スレッドの多重度を倍増できる．

また，**田中らの提案手法から継承した点**は，

⑤ 多数スレッドに対する負荷の動的配分
⑥ 共有メモリのバンク衝突の防止

である．下記の表 2 に，田中らの提案手法との比較を示す．

表 2　本提案手法と田中らの提案手法との比較

項番	比較項目	田中らの提案手法	本提案手法
1	クイーンの数の制限	なし	奇数に限定
2	探索量の削減	1/2	約1/8
3	CPU側の探索行数(m)	m=5（最適値に設定）	m=n～n-2　但し, n=(N-1)/2
4	GPUの起動回数	一括処理	分割処理
5	負荷(Seed)の動的割り当て	あり	あり(田中らの提案手法を踏襲)
6	共有メモリのバンク衝突防止	あり	あり(田中らの提案手法を踏襲)
7	Seedsの総数(N=19の場合)	232,174	約100M個
8	Seedの情報量	4バイト(符号化)	4バイト x 4
9	スタック情報	4バイト x 4	4バイト x 2
10	スタックの深さ	N-5	n+2～n　但し, n=(N-1)/2
11	スレッドの多重度	64 x 32	96 x 64
12	CPU/GPUの負荷バランス	CPUは無視できるレベル	負荷バランスが重要
13	CPU/GPUの並行処理	なし(不要)	あり

（補足）田中らの提案手法との大きな相違として Seeds に対する考え方がある．

（1） Seed の情報には，スタックの情報に近い 4words の情報を採用し，符号化は行わないこととした．本提案手法では GPU の探索は浅くなっているので，符号化による通信量削減のメリットより，復号化のコストが非常に大きいと判断したためである．一方，情報量は 1word から 4words と 4 倍となったことでグローバルメモリのアクセス回数の増大が懸念される．これについては，3.6 節で述べる．

（2） 大量の Seeds を捌くため分割処理を採用した．これは，CPU 側の探索を深くして（大量の Seeds 生成により）CPU 処理を重くしても，GPU 側の探索が浅くなることで，GPU 処理時間が削減され，トータルの処理時間を短縮できればよいという考えである．CPU 処理と GPU 処理の負荷の調整を行い，バランスのよいポイントを模索した．最終的には，CPU と GPU の並行処理方式を導入し，CPU 処理時間を隠し，GPU 処理時間がほぼトータル時間となる方式へと発展させた．

3.　提案手法の詳細

以下では各施策の詳細を説明する．（各施策の定量的な評価は，表 9 を参照）

3.1　施策 1：　（I,J）フィルタ　（代表解の選択条件の取り込み）

3.1.1 Seeds 生成処理

CPU 側で深さ 0～n （=(N–1)/2）行まで探索を行い，中央の行・列に配置されたクイーンの位置を決定する．そのクイーンの位置を（n,I），(J,n）とすると，

- I=J=n の場合：type Ⅰ の解である．type Ⅰ の解には，type-a～type-c の解の可能性があるが，その区分を容易に特定することはできない．そこで，左右対称解を考慮し，第 0 行のクイーンの位置が左半分の場合のみ Seed-C を生成する．
- I<J<n の場合：type Ⅱ の代表解である．type-a の解の元となる Seed-A を生成する．
- 上記以外の場合：type Ⅱ の解であるが代表解ではないので Seed の生成は不要．

【Seed-A の生成】

u, r, l, bitmap の 4words を設定する．GPU 側では，深さ n の探索を行う．CPU 側では，GPU から返された解の数を 8 倍して集計する．

【Seed-C の生成】

u, r, l, bitmap の 4words を設定する．GPU 側では，深さ n の探索を行う．CPU 側では，GPU から返された解の数を 2 倍して集計する．

なお，本施策は大量の Seeds 生成を行うことになる．クイーンの数が大きくなると Seeds を格納するためのメインメモリが不足する事態が想定されるので，適当な量（PoolSize）に区切って分割処理を行う方式を採用した．Seed は各区分ごとのバッファに格納する．そのバッファが一杯になると GPU に渡され，解の探索が行われる．

3.1.2 CPU/GPU の役割分担の最適化

GPU は，単純で大量処理に向いているが，if 文などが多用される複雑な処理はあまり向いていない．従って，前半の複雑な処理（Seeds 生成処理）は CPU 側で行い，後半の単純・大量の探索は GPU で行うことが自然である．一方，こうすることによって，CPU 側の処理が重すぎないかと懸念されるが，その評価については 4.1 節を参照されたい．

3.2　施策 2：K-line 設定による Seeds の削減

施策 1 の（I,J）フィルタは，深さ n （=(N–1)/2）まで CPU 側で探索を行い，大量の Seeds を生成し，それを GPU 側に送って処理する構造である．N=19 の場合，Seeds の総数は約 383M 個となり，さすがに性能への悪影響が懸念される．そこで，Seeds を削減する仕組みとして K-line（＝Seed 生成を開始する行）を設定し，中央行までの探索を待たずに Seed 生成を行う．K-line を設定することによって，GPU 側には，①探索の深さが可変になることの考慮，および，②中央行におけるクイーンの配置を制限する

処理が追加となる. 図 2 は, K=n-2 の場合について, クイーンの位置と Seed 生成 / GPU の探索範囲の関係を示している.

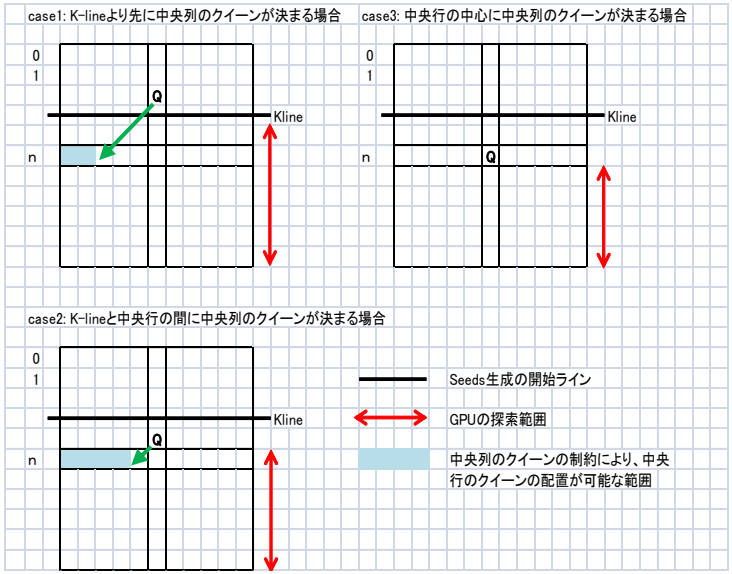


図 2　K-line とクイーンの位置関係

【K-line 設定時の Seed 生成論理】

n =(N–1)/2, CPU 側の探索処理中の深さを nest（0~（n-1)), 中央行のクイーンの位を I, 中央列のクイーンの位置を J とすると, 以下のようになる.

- nest < K　：　探索を継続する.
- K ≦ nest & nest < n-1 :
  J が確定している場合（case1）は, Seed-A の生成を行う.
  J が確定していない場合, 探索を継続する.
- nest = n-1　:
  J が確定している場合（case2), Seed-A の生成を行う.
  J が確定していない場合（case3), 中央にクイーンを配置したものとして Seed-C の生成を行う.（ただし, 左右対称解を考慮し, 第 0 行のクイーンの位置が左半分の場合のみ生成する）

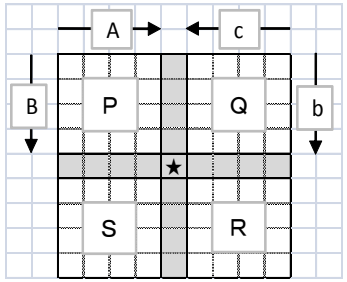
【Seed-A の生成】（K-line 設定にあわせて変更）

u, r, l の 3words に続き, 4word 目には, 中央行のクイーンの位置を制限する情報 (J), および GPU 側の探索の深さを定める情報（d=max（J,K)）を設定する. CPU 側では, GPU から返された解の数を 8 倍して集計する.

3.3　施策 3 : 簡易フレームチェックによる Seeds の削減

施策 3 は, 相対的に割合が高くなっている type I の解の Seed 削減策である.

図 3 のように, 中央の行・列によって分けられた 4 つの領域を P,Q,R,S とする. また, #(P), #(Q) を P, Q それぞれに配置されたクイーンの数とする.



フレーム A,B の定義
A:P に配置されたクイーンの列方向のビットマップ
B:P に配置されたクイーンの行方向のビットマップ

★：中央に配置されたクイーン

図 3 ボードの分割とフレーム

【簡易フレームチェック】

- #(P)　=　#(Q) の場合
  いままでどおり Seed-C の生成を行う.（ただし, 左右対称解を考慮し, 0 行のクイーンの位置が左半分の場合のみ生成する）
- #(P)　<　#(Q) の場合
  A < B のとき, Seed-B の生成を行う.
  A = B のとき, Seed-C の生成を行う.
  A > B のとき, Seed の生成は不要である.
- #(P)　>　#(Q) の場合
  Seed の生成は不要である.

【Seed-B の生成】

u, r, l, bitmap の 4words を設定する. また, GPU 側では, 深さ n の探索を行う. CPU 側では, GPU から返された解の数を 4 倍して集計する.

3.4　施策 4: CPU/GPU の並行処理

本提案手法のプログラム構造は, CPU 側の Seeds 生成部, GPU 制御部, GPU 側のカーネル関数部の構成をとっている. 表 3 は, N=19 の場合の各部の処理時間の内訳である.

表 3　求解時間の内訳（N=19）

各処理部	処理時間(s)
Seeds生成部	5.12
GPU制御部	0.59
カーネル関数部	17.4
求解時間	23.11

現在の制御構造では，GPU で探索処理をしている時 CPU 側はアイドル状態である．CPU 側の Seeds 生成処理は，GPU の探索処理より短い時間ではあるが，無視できないレベルの時間である. この時間は, GPU の探索処理と並行して，CPU 側で次の Seeds の生成処理を行う方式をとれば，ほとんど隠れてしまう筈である．Seeds 生成処理を別スレッドで動作させ，CPU 側の Seeds 生成処理と GPU 側の探索処理を非同期に動作させる構造に変更する．

3.5 施策 5 : 64 ビットレジスタの活用

スタック情報 u, r, l, bitmap の 4 words を 64 ビットレジスタ（uu, ll）と 32 ビットレジスタ(rr)を使って，b と bitmap の 2words 構成に変更する．u, r, l の効き筋情報は，uu, rr, ll に保持する．（64 ビットレジスタの使用により，シフト操作による情報の消失はない） なお，b は各行のクイーンの位置を格納する．bitmap の意味は変更ない．以下にスタック操作の疑似コードを示す．(1.1 節の疑似コードと比較参照されたい)

【push 操作】

```
bit =－bitmap[x] & bitmap[x]
bitmap[x] ^= bit
b[x] = bit       // pop 操作のために save
uu =（uu | bit）<< 1
rr =   rr | bit
ll =（ll | (bit<<32)）>> 1
bitmap[x+1] = mask & ~（uu | rr] | (ll>>32)）
x++
```

【pop 操作】

```
x--
bit=b[x]   //クイーンの位置を復元

uu = (uu>>1) ^ bit
rr =   rr ^ bit
ll = (ll <<1) ^(bit<<32)
```

この方式のメリットは，共有メモリの使用量を半減できることである．共有メモリへのアクセス回数の削減，並びに，スレッド多重度の倍増が期待できる．一方，デメリットは，pop 操作に追加の処理が発生することである．

3.6 int4 データ型の利用

Seed は，4 つのパラメタ構成（4 words）となっている．もし，1 word ずつアクセスすると，グローバルメモリアクセスは非常に遅いので，性能低下の原因となっている恐れがある． VRAM から共有メモリに取り込むところを 1 回のアクセスとなるように Seed のデータ型を int4 に変更して，実測したところ，変更前とほとんど性能差がないことが判った．（表 9 参照）

4. GPU のチューニング

4.1 CPU/GPU 間の負荷の最適化

本提案では，（Seeds が大量に生成されるため），Seeds をある程度まとまった単位（PoolSize）に分割して渡す方式を採用した．分割処理方式は，既存手法の一括処理と較べて，① CPU 処理と GPU 処理の負荷の偏り（CPU 処理が重くなりすぎ，全体の処理時間に影響しないか），②CPU/GPU 間の通信オーバヘッド等の懸念がある．

【CPU 処理と GPU 処理の負荷の調整】

K-line の設定値により，CPU 処理の負荷を制御できる．以下の表 4 は，K-line の値を変化させて，CPU 側の処理時間と GPU 側の処理時間を測定したものである．

表 4　K-line 設定値と求解時間の変化（N=19 の場合）

K-line設定値	n	n-1	n-2	n-3	n-4
Seeds総数	約383M個	約255M個	約131M個	約104M個	約100M個
スタックの深さ	13	14	15	16	17
スレッドブロック数	112	96	96	96	80
求解時間（s）	46.44	29.53	23.62	22.91	27.86
Seeds生成部	22.88	7.25	5.40	4.93	4.88
GPU制御部	1.51	1.03	0.51	0.25	0.22
カーネル関数部	22.05	21.24	17.72	17.72	22.78

補足：本測定には初期の版数(4words のスタック情報版)を使用

K-line の値を減少させると，Seeds 総数が減少する．それに応じて，CPU 側（Seeds 生成部）の負荷は減少し，GPU 側（カーネル関数部）の負荷も減少する．（GPU 側の負荷が減少する理由は，Seeds 取り込み回数の減少である）しかし，GPU 側は，その後，探索の深さの増加がスレッドの多重度にも影響し，性能低下が現れる．K-line の最適値は，n-2,n-3 あたりと考えられる．

【CPU/GPU 間の通信オーバヘッド（GPU 制御部の処理時間） 】

CPU/GPU 間の通信オーバヘッドには, Seeds を GPU の VRAM に転送する時間と GPU を起動・終了する時間が含まれる．実測によれば，1 回の GPU 起動・終了時間は 36μs 程度であり, 殆ど問題にならない. 一方, VRAM への転送時間は上記の表 4 の K-line=n-2 の場合，約 0.5 秒となっている．最新版の N=19 の求解時間（9.98 秒）からみても，約 5%とあまり問題ない値である. 一方, N が大きい場合は, PoolSize を大きくすると，VRAM への転送時間（実際は非同期処理の待ち合わせ時間）が短縮され，若干の改善がみられる．（表 5 参照）N が 21 以上では，PoolSize＝10M 程度がよいと思われる．

表 5　PoolSize と求解時間の関係（N=21 の場合）

PoolSize	1.0M	2.5M	5M	10M
Seeds総数	2608M個（41.7GB）			
求解時間(s)	604.5	591.6	586.3	584.1

4.2 MPSZ（スレッドブロック数）と求解時間の関係

スレッドの多重度は，カーネル関数起動時のパラメタ(MPSZ：スレッドブロック数と BLKSZ：1 スレッドブロック当たりのスレッド数）で調整可能である．田中らの研究報告では，スレッド数は 32（ワープサイズ）の倍数がよいとされている．筆者らの評価でも同様の結果が得られたので，BLKSZ=32 に固定し，MPSZ でスレッドの多重度を調整することとした．（表 6 参照）

表 6　MPSZ と求解時間の関係（N=19 の場合）

MPSZ	求解時間(s)	求解時間2(s)	参考値(s)
64	17.73	10.87	23.00
80	14.78	10.15	18.92
96	12.83	9.98	16.19
112	11.71		
128	10.82		
144	10.82		
160	10.85		
192	10.85		

参考値は，スタック情報に 4words を使用した時の測定値である．この時，MPSZ の最大値は 96 である．

求解時間 2 は BLKSZ=64 に変更後の再測定結果である．

スタック情報に 2words を使用している最新版の MPSZ の最大値は 192（12x16）であるが，MPSZ=128 以降は，ほぼ横ばいとなっている．これは，アーキテクチャの制限により 1 MP では最大 8 ブロックまでしか同時実行できないことが原因と判明した．BLKSZ=64 に変更して再測定を行った結果（求解時間 2）では，共有メモリの制約による MPSZ の上限値（MPSZ=96）まで若干の改善がみられる．

5. 評価実験

共有メモリの必要量は，アルゴリズムによって異なるので，平等な比較のためには測定環境を合わせる必要がある．今回の測定では，GPU のパラメタを“N=27 が測定可能な範囲”を想定して設定することとした．

（パラメタの意味）

ＭＰＳＺ ：スレッドブロック数　　ＢＬＫＳＺ ：1 スレッドブロック当たりのスレッド数

Ｑｕｅｅｎｓ ：GPU 側の探索の深さ（N=27 探索時のスタックの深さ）

ＰｏｏｌＳｉｚｅ：Seeds の分割単位（GPU への一括転送の最大数）

5.1 最新版の求解時間

以下の表 7 は，最新版の求解時間を求めたものである．

GPU のパラメタ：MPSZ=96, BLKSZ=64, Queens=16, PoolSize=2500000

表 7　提案手法の求解時間

クイーン数	解の数	求解時間(s)	備考
15	2,279,184	0.20	
17	95,815,104	0.51	
19	4,968,057,848	9.98	
21	314,666,222,712	591.56	
23	24,233,937,684,440	45818.24	(12H 43M 39S)
25	2,207,893,435,808,350	約50日	(予測値)
27	未解決	約60日 x 100台	(予測値)

N=23 までは実測値である．（ただし，N=23 は PoolSize=10M で測定）

N=19～23 の求解時間の伸び率は，解の総数の伸び率によく比例している．

N=25,27 の求解時間は，この関係を利用して求めた予測値である．

N=23 の求解時間は, N=24 の世界記録を 2003 年に樹立した文献[4]の電通大の PC クラスタ（Intel Pentium4 Xeon 2.8GHz CPU x 68）の実測結果（56.9H）の約 4.5 倍高速となっている．

5.2 他のプログラムとの比較

田中らの提案手法：MPSZ=64,BLKSZ=32,Queens=22

本提案手法　　：MPSZ=96,BLKSZ=64,Queens=16,PoolSize=2500000

表 8　他のプログラムとの比較

N	単一CPUのプログラム			GPUを用いたプログラム			
	Somers版[3]	部分解合成法		田中らの提案手法	本提案手法		
	実測値(s)	実測値(s)	相対比1	実測値(s)	実測値(s)	相対比2	相対比3
17	28.00	3.71	7.5	1.90	0.51	3.8	55.3
19	1,617.00	107.00	15.1	102.40	9.98	10.3	162.0
21	111,425.00	5,919.82	18.8	7,103.24	591.56	12.0	188.4

相対比 1：田中らの提案手法 / 部分解合成法

相対比 2：田中らの提案手法 / 本提案手法

相対比 3：Somers / 本提案手法

(1) 本提案手法は，ベースとなった田中らの提案手法に比べて，10.3～12.0 倍高速である．また，クイーン数の増加に伴い改善効果も向上している．
(2) 単一 CPU との比較では，N=19 の場合，Somers 版の 162～188 倍高速である．

5.3 各施策の効果

ここでは開発の順に従って，どのように性能改善が推移したかを示す．参考値として，田中らの提案手法の求解時間も測定した．

表 9　各施策の効果（N=19 の場合）

	求解時間(s)	改善率	備考
田中らの提案手法	102.40		参考値
初版　施策1	46.50	0.55	(I,J)フィルタ Seeds=383M個
第0.1版	46.43	0.00	int4データ型の利用
第0.2版　施策2	29.33	0.37	K-line(n-1) Seeds=255M個
第0.3版　施策2	23.11	0.50	K-line(n-2) Seeds=130M個
第0.4版　施策3	22.24	0.04	簡易フレームチェック Seeds=100M個
第0.5版　施策4	17.12	0.23	CPU/GPUの並行処理
第0.6版	16.15	0.06	電通大アルゴリズムの採用
最新版　施策5	9.98	0.38	64ビットレジスタの活用

改善率：1－改善後 / 改善前

(1) 施策 1 の改善率は 55%. 探索量の削減およびスレッド多重度の向上による効果が，大量 Seeds による CPU/GPU 間の通信オーバヘッドを大きく上回った結果である．しかし, Seeds の削減量は 4 倍弱なので, 改善率 55%では不十分な結果といえる．
(2) 施策 2 の改善率は 50%．4.1 節で述べたように，Seeds 生成処理の軽減と，カーネル関数部での Seeds 取り込み処理の削減による効果である．
(3) 施策 3 の改善率は 4%と極めて小さい. Seeds の削減量が少ない事と type I の Seed の重さ（必要となる計算量）が軽いためと考えられる．
(4) 施策 4 の改善率は 23%．3.4 節で述べたように，CPU 側の Seeds 生成処理時間の削減となっている．
(5) 施策 5 の改善率は 38%. 共有メモリへのアクセス回数の減少および多重度の向上の相乗効果である．

6. まとめ

田中らの提案手法をベースに，クイーンの数を奇数のみに限定することによって，Somers アルゴリズムと部分解合成法の「代表解の選択条件」とを融合させる提案を行った．改善のポイントは，①部分解合成法の「代表解の選択条件」による探索量の大幅な削減，②if 文の多い複雑な処理は CPU で行い，大量で単純な処理は GPU で行うという CPU/GPU の役割分担の最適化，③CPU と GPU の並行処理，④GPU の特性・制約を考慮したアルゴリズムの選択（共有メモリ使用量の節約）などである．②～④の考え方は，GPU を他の問題に適用する際にも有効なアプローチと考える．

本提案手法の評価実験では，田中らの提案手法に較べて 10.3～12.0 倍，単一 CPU との比較では Somers アルゴリズムの 162～188 倍と著しい性能向上を実現できた. また，N=23 の求解時間を N=24 の世界記録を 2003 年に樹立した電通大の PC クラスタ（Intel Pentium4 Xeon 2.8GHz CPU x 68）使用時の結果と比較しても約 4.5 倍高速である．

N=23 の求解時間（12H 43M 39S）から推測すると，N=25 の求解時間は GPU 付き PC 1 台 x 約 50 日, N=27 の求解時間も GPU 付き PC 100 台 x 60 日程度で求まる可能性が見えてきた．もし環境が整えば 一度挑戦してみたい課題である．

なお, 本論文で実験に使用したプログラム(nq_symG)のソースを Web サイト[8]で公開予定である．

参考文献

1） 田中慶悟,藤本典幸,GPU を用いた N-Queens 問題の求解
情報処理学会シンポジウムシリーズ Vol.2011,No.6 pp.76-83,2011
2） 萩野谷一二,NQueens 問題への新しいアプローチ（部分解合成法）について
情報処理学会研究報告 Vol.2011-GI-26 No.11,2011
3） J.Somers, The N Queens Problem: a Study in Optimization
http://www.jsomers.com/nqueen_demo/nqueens.html
4） 吉瀬謙二,片桐孝洋,本多弘樹,弓場敏嗣,PC クラスタを用いた N-Queens 問題の求解,
電子情報通信学会論文誌 D-I,J87-D-I（12）,pp.1145-1148,2004
5） 小山田耕二,岡田謙二, CUDA 高速 GPU プログラミング入門, ㈱秀和システム,2010
6） NVIDIA Corp., NVIDIA CUDA C Programming Guide 3.2
7） NVIDIA Corp., Tuning CUDA Applications for Fermi
8） nq_symG ソースプログラム（公開予定）,
http://deepgreen.game.cocoon.jp/nqueens_index.html