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 *1
藤本 典幸 *1

*1 大阪府立大学　大学院理学系研究科　情報数理科学専攻

1. はじめに

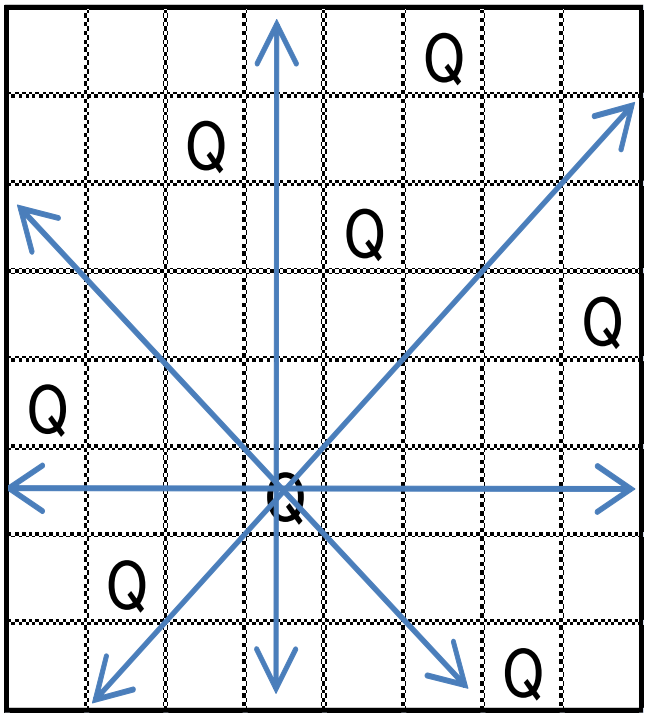
1.1 N-Queens 問題とは

1.2 Somers のアルゴリズム

1.3 解の区分

1.4 Seed

1.1 N-Queens 問題とは



・N x N のチェス版に N 個のクイーンを配置。互いに攻撃し合わない配置の総数を求める問題
・N=26 までの解の総数は既知
・N=27 以降の解の総数は未解決

・解法：**Somers アルゴリズム**が有名
・高速化：**対称解の取り扱いがポイント**
（上下／左右反転・回転操作などの対称変換による別解）

【最近の取り組み】

✧ 田中らの **GPU を用いた高速化**の提案

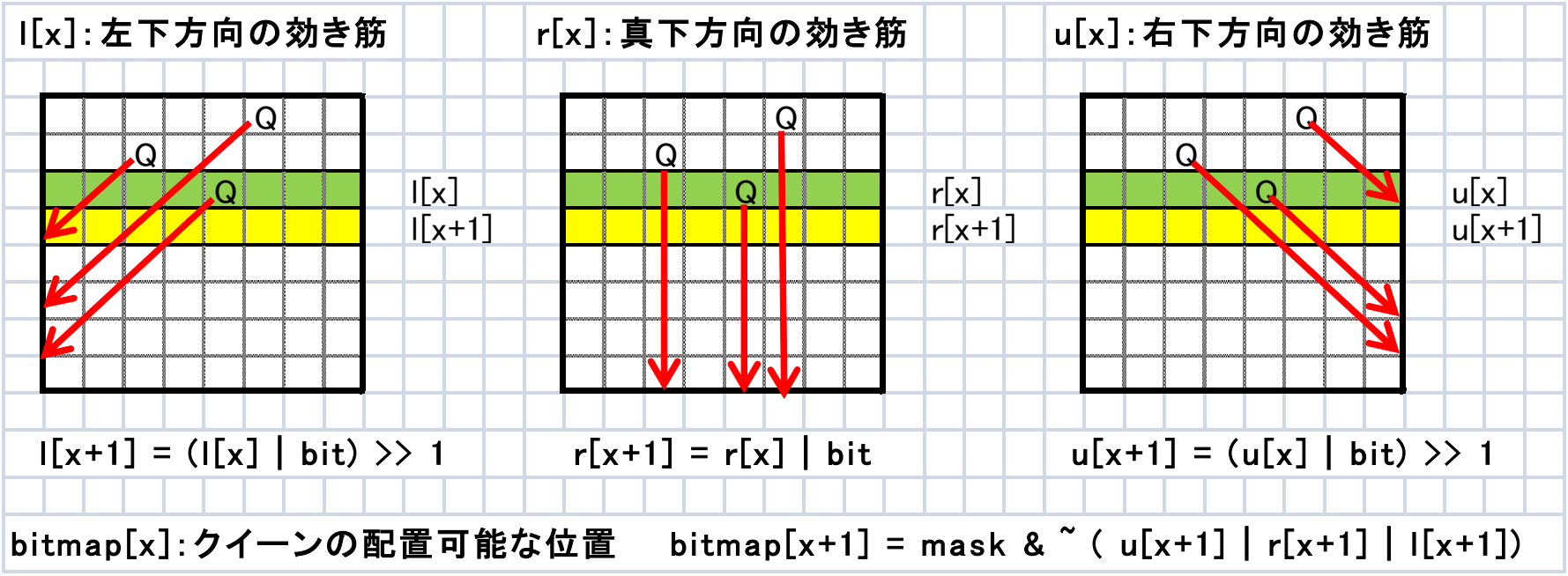
✧ **新アルゴリズム「部分解合成法」**の提案

1.2 Somers のアルゴリズム

✧ 左右反転解を考慮して探索量を半減

✧ スタックを利用したループ制御（GPU では関数の再帰呼び出し不可）

スタック情報：クイーンの効き筋の管理(u, r, l, bitmap の 4words が標準)



スタックの深さ：GPU 側でクイーンを探索するとき必要な行数

ス タ ッ ク 域：スタック情報 ×スタックの深さの配列（高速な共有メモリに配置）

共有メモリは小容量のためスレッドの多重度を制約する要因

1.3 解の区分

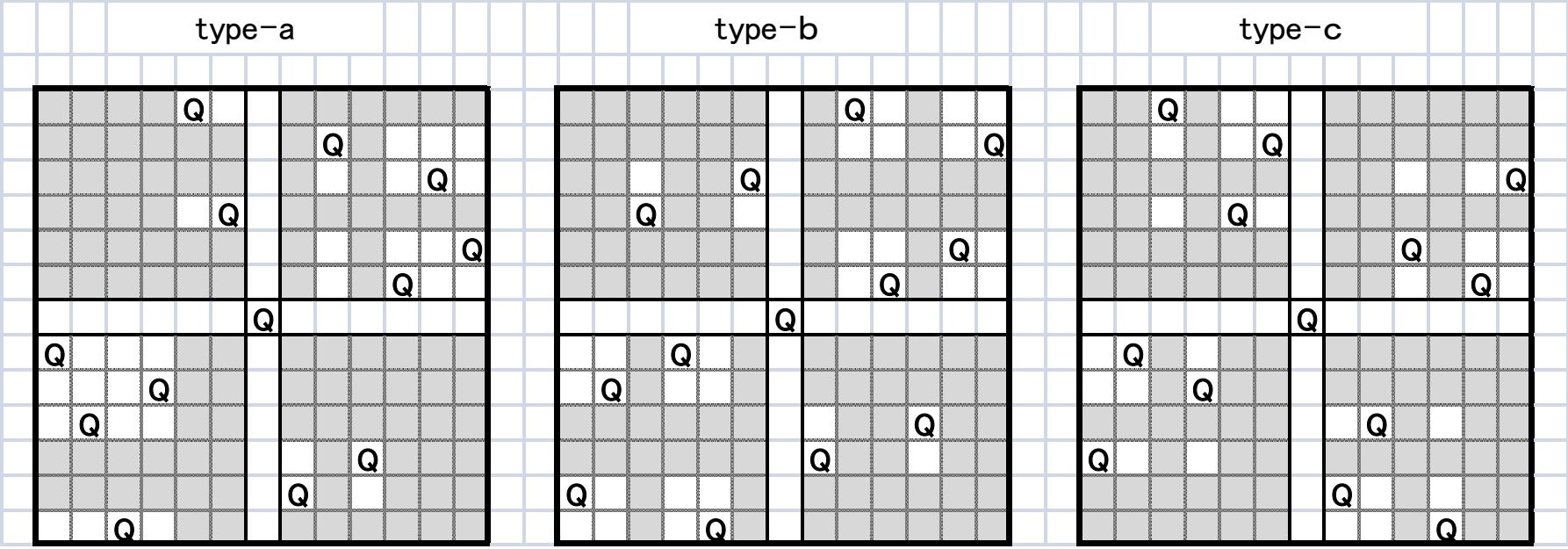
対称解による区分(上下反転、左右反転、回転操作による解分類)

type-a：非回転対称解（**8 つの対称解**が存在するもの）

type-b：180° 回転対称解（**4 つの対称解**が存在するもの）

type-c：90° 回転対称解（**2 つの対称解**が存在するもの）

代表解：対称解を生成する元となった解



N-Queens 問題の解は殆どが type-a

1.4 Seed

Seed：CPU から GPU へ探索を引き継ぐための情報

Seeds 全体として、生成される解の重複・欠落がないこと

ex.　u,r,l,bitmap の 4words が標準的な情報（bitmap は必須情報ではない）

Seeds 削減の施策では 4word 目を別の情報に使用

【Seed の区分】

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

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

type-b の代表解を生成する Seed　／　2 つの Seeds で Seed-A を代用するケース

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

type-c の代表解を生成する Seed／4 つの Seeds で Seed-A を代用するケース／

2 つの Seeds で Seed-B を代用するケース

２．関連研究と提案手法の概要

2.1 田中らの提案手法の概要

2.2 新アルゴリズム「部分解合成法」の概要

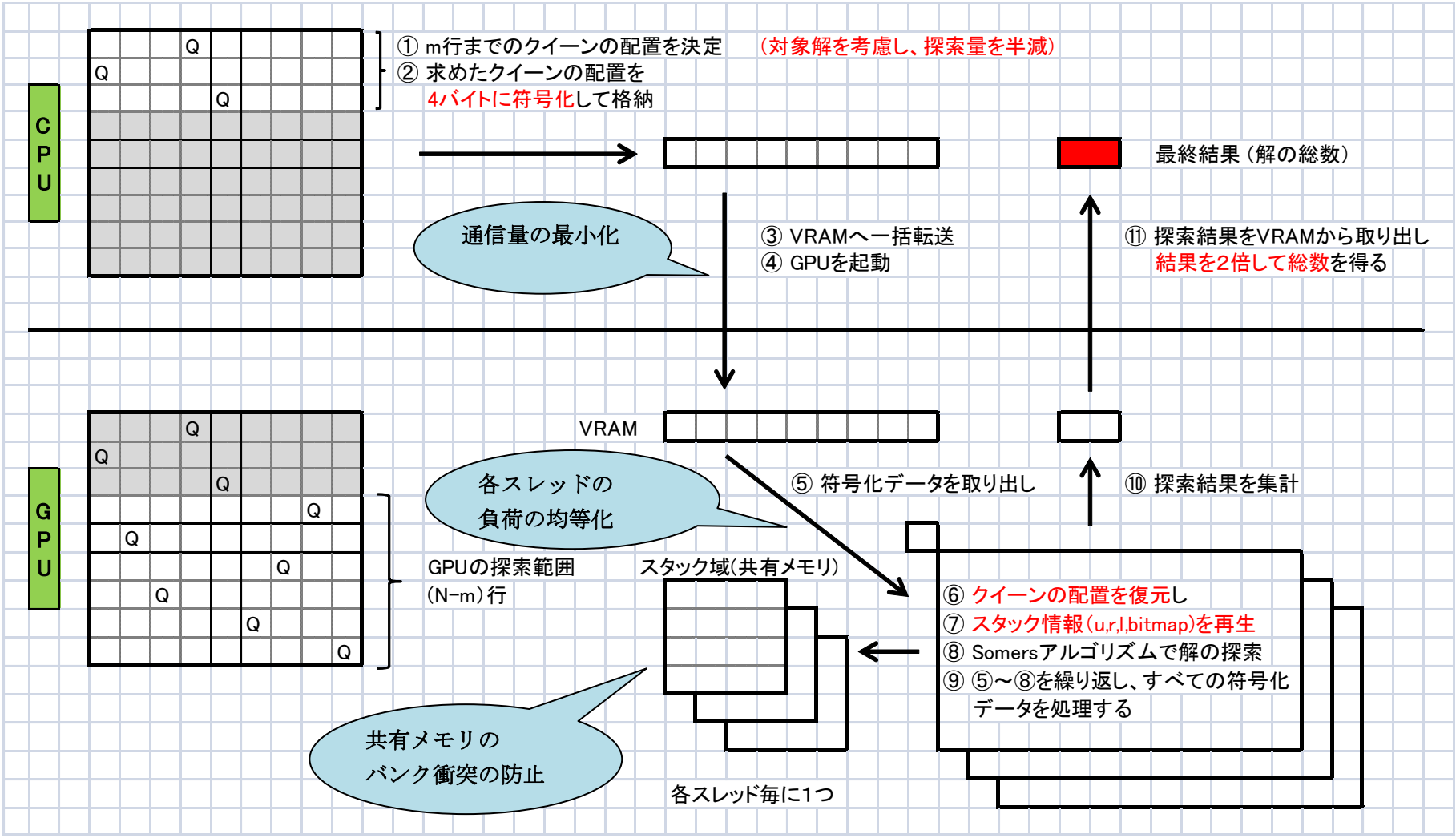
2.3 本提案手法の基本構想

2.4 本提案手法の概要

2.5 本提案手法の主な取り組み

2.6 本提案手法と田中らの提案手法との比較

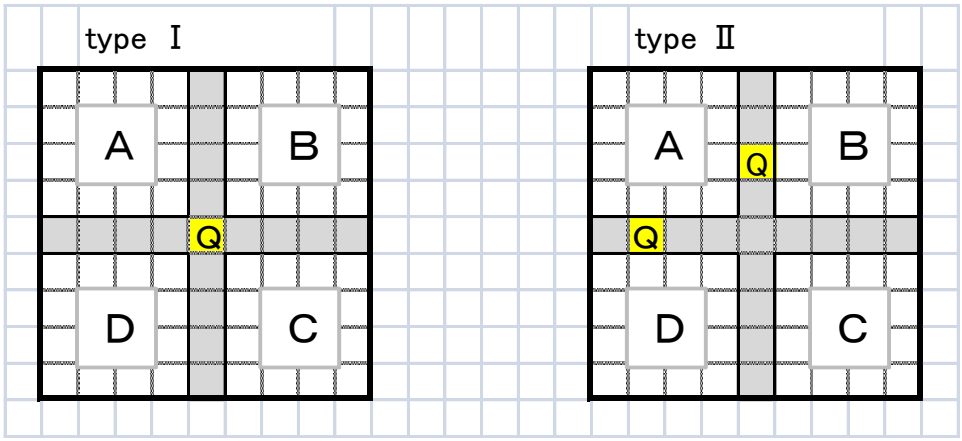
2.1 田中らの提案手法の概要



2.2 新アルゴリズム「部分解合成法」の概要

①解の類別とその判定方法(代表解の選択条件)
②部分解から全体解を合成する手順の提示

【解の類別と代表解の選択条件】(クイーンの数(N)が奇数の場合)



typeⅠ：クイーンが中央に１つ配置された形（type-a~type-c の解の可能性がある）
typeⅡ:2 つのクイーンが配置された形（type-a の解のみ）

typeⅡの解の代表解の選択条件：I＜J＜n を満たす解　(type-a の代表解)
　　ただし、n=(N-1)/2、　中央の行・列のクイーンの位置を (n,I), (J,n)

→ 探索量を約 1/8 に削減（解の殆どは type-a）
(Somers アルゴリズムの約４倍の削減効果)

2.3 本提案手法の基本構想

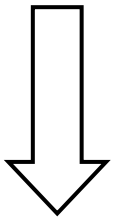
田中らの提案手法をベース

(GPU 版 Somers アルゴリズム)

＋

(クイーンの数を奇数に限定)

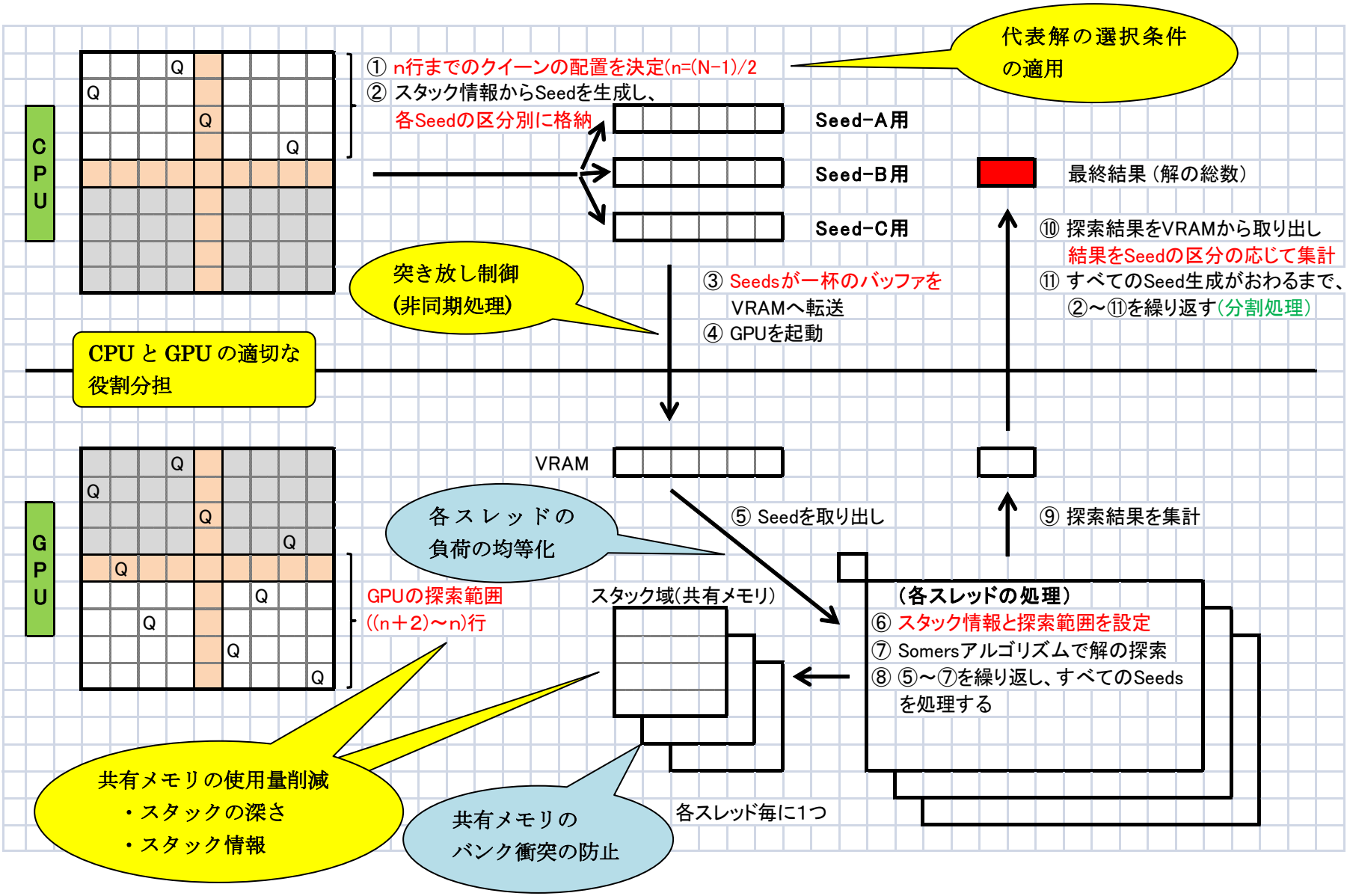
部分解合成法の「代表解の選択条件」の取込み



狙い：探索量の大幅な削減

(Somers：1/2 → 部分解合成法：約 1/8)

2.4 本提案手法の概要


代表解の選択条件
の適用
① n行までのクイーンの配置を決定(n=(N-1)/2
② スタック情報からSeedを生成し、
各Seedの区分別に格納
CPU
Seed-A用
Seed-B用
Seed-C用
最終結果（解の総数）
⑩ 探索結果をVRAMから取り出し
結果をSeedの区分の応じて集計
⑪ すべてのSeed生成がおわるまで、
②～⑪を繰り返す(分割処理)
突き放し制御
(非同期処理)
③ Seedsが一杯のバッファを
VRAMへ転送
④ GPUを起動
CPU と GPU の適切な
役割分担
VRAM
⑤ Seedを取り出し
⑨ 探索結果を集計
GPU
各スレッドの
負荷の均等化
GPUの探索範囲
((n＋2)～n)行
スタック域(共有メモリ)
（各スレッドの処理）
⑥ スタック情報と探索範囲を設定
⑦ Somersアルゴリズムで解の探索
⑧ ⑤～⑦を繰り返し、すべてのSeeds
を処理する
共有メモリの使用量削減
・スタックの深さ
・スタック情報
共有メモリの
バンク衝突の防止
各スレッド毎に1つ

2.5 本提案手法の主な取り組み

主な取り組み

① **(I,J) フィルタ**　　（「代表解の選択条件」を取込み）
中央行までの探索を行い、代表解の選択条件を適用
以降の探索量を約 1/8 に削減

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
後半の単純・大量の探索処理を GPU 側

③ **CPU/GPU の並行処理**
CPU 側の処理と GPU 側の処理を並行して実行

④ **共有メモリ使用量の削減**（64 ビットレジスタの活用）
スタック情報を 4words から 2words に削減

田中らの提案手法からの継承

⑤ **多数スレッドに対する負荷の動的配分（負荷の均等化）**

⑥ **共有メモリのバンク衝突の防止**

2.6 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月2日	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）Seeds に対する考え方

✧ 多数のスレッドに均等な負荷を提供（既存手法)

✧ **CPU と GPU との協調作業の媒介**(トータルな負荷バランス)

3. 提案手法の詳細

3.1 施策１：(I,J)フィルタ　(代表解の選択条件の取込み)

Seeds 生成処理

CPU 側で深さ 0～ n (= (N - 1) / 2) 行まで探索し、
中央の行、列に配置されたクイーンの位置 (n,I), (J,n) を決定

- **I=J=n の場合**：**Seed-C を生成** (左右対称解を考慮し生成は半分とする)
  type Ⅰ の解である（type-a～type-c の解の可能性がある）

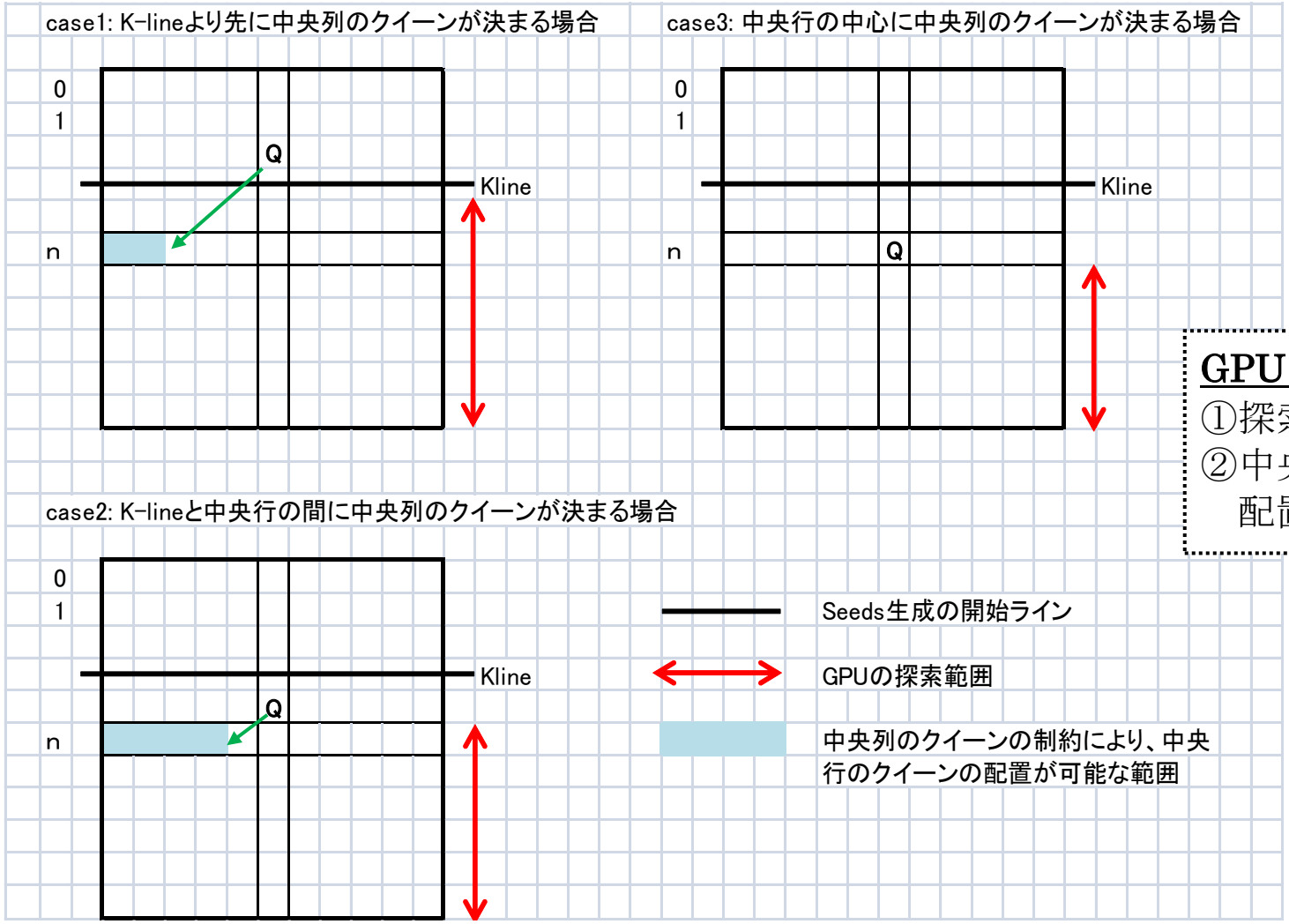
- **I<J<n の場合**：**Seed-A の生成**
  type Ⅱ の代表解である

- **上記以外の場合**：**Seed の生成は不要**
  type Ⅱ の解であるが、代表解ではない

（補足）大量の Seeds 生成に対応するため**分割処理方式を採用**
Seed-A, Seed-C の生成およびその扱いは後述

3.2 施策 2：K-line 設定による Seeds の削減

K-line：Seed 生成を開始する行（中央行までの探索を待たずに Seed 生成を開始）



GPU 側の対処

①探索の深さを可変

②中央行のクイーンの配置を制限

3.3 K-line 設定時の Seed 生成論理

記号の定義

K　　：Seed 生成を開始する行　　n = (N-1) / 2

I　　：中央行のクイーンの位置　　J　　：中央列のクイーンの位置

ｎｅｓｔ：CPU 側の探索処理中の深さ(0~(n-1))

- **nest < K の場合**　：　探索を継続
- **K ≦ nest & nest < n-1 の場合**　：

  J が確定：(case1)、**Seed-A の生成**

  J が未確定：探索を継続
- **nest = n-1 の場合**

  J が確定：(case2)、**Seed-A の生成**

  J が未確定：中央にクイーンを配置したものとして **Seed-C の生成**

  （ただし、左右対称解を考慮し生成は半分のみとする）

【Seed-A の生成】（K-line 設定用）

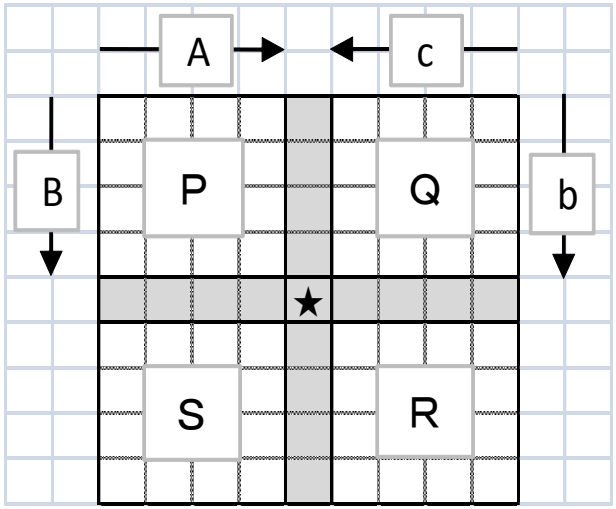
u, r, l に続けて **4word 目には bitmap の代わりに J と d=max(J,K)を設定**

J：GPU 側の探索で中央行のクイーンの位置を制限する情報

d：GPU 側の探索の深さを規定する情報

3.4 施策 3：簡易フレームチェック

（type I の解の Seeds 削減策）



★：中央に配置されたクイーン

P,Q,R,S:　中央の行・列により分割された各領域
#(P),#(Q)：P,Q それぞれに配置されたクイーンの数

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

- **#(P) = #(Q)**：Seed-C の生成
  （ただし、左右対称解を考慮し生成は半分のみとする）
- **#(P) < #(Q)**：A < B のとき、Seed-B の生成
  A = B のとき、Seed-C の生成
  A > B のとき、Seed の生成は不要
- **#(P) > #(Q)**：Seed の生成は不要

3.5 Seed 情報とその扱い

記号の定義

K　Seed 生成を開始する行

J　中央列のクイーンの位置　　n ＝ （N – 1）　/ 2

N　クイーンの数　　d = max(J,K)

	Seed-A	Seed-B	Seed-C
Seed情報	u, r, l, (J,d)	u, r, l, bitmap	u, r, l, bitmap
GPUの探索の深さ	N-d	n	n
CPU側の処理	8 倍して集計	4 倍して集計	2 倍して集計
その他	中央行のクイーンの位置を J 未満に制限		

3.6 CPU/GPU の適切な役割分担

GPU の特性

✧ 単純で大量処理に向いている

✧ if 文などが多用される複雑な処理は苦手（スレッド分散）

➔ CPU/GPU それぞれの得意分野を分担

CPU：前半の複雑な処理(Seeds 生成処理)

GPU：後半の単純・大量の探索に専念

メリット：GPU 側の探索が浅くなる

(スタック域(=共有メモリ)の削減➔スレッド多重度の向上)

懸念事項：CPU 側の処理が重すぎないか ？

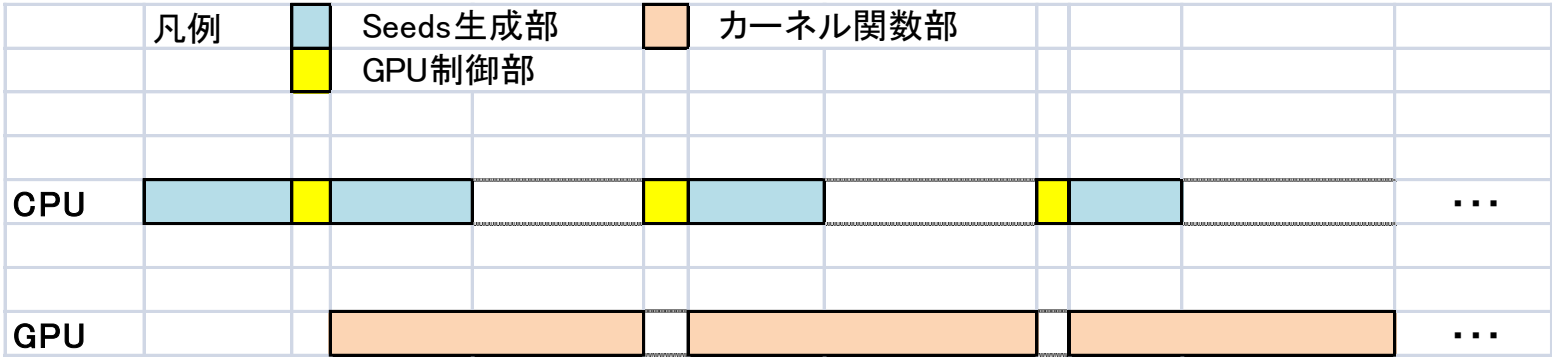
3.7 施策 4：CPU/GPU の並行処理

求解時間の内訳（N=19 の場合）

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

Seeds 生成部の処理時間（5.12 秒）は、全体の 22%（無視できないレベル）

➔ Seeds 生成部を別スレッド化し、GPU 側の探索処理と非同期に動作



3.8 施策 5：64 ビットレジスタの活用

スタック情報の変更： 4 words 構成（u,r,l,bitmap） → 2words 構成

- ✧ b(クイーンの位置情報), bitmap の 2words
- ✧ u,l の代わりに 64 ビットレジスタ(uu,ll)を使用
- ✧ r の代わりに 32 ビットレジスタ(rr)を使用

push 操作	pop 操作
bit = −bitmap[x] & bitmap[x]	x--
bitmap[x] ^= bit	bit = b[x]
b[x+1] = bit	uu = (uu >> 1) ^ bit
uu = (uu | bit) << 1	rr = rr ^ bit
rr = rr | bit	ll = (ll <<1) ^ (bit<<32)
ll = (ll | (bit<<32)) >> 1	
bitmap = mask & ~(uu | rr |(ll>>32))	
x++	

共有メモリの使用量が半減

➔ 共有メモリへのアクセス回数の削減

➔ スレッド多重度の倍増

3.9 int4 データ型の利用

Seed は 4 words 構成（田中らの提案手法では 1 word）➔ **性能劣化の懸念**

（GPU のグローバルメモリアクセスは数百クロックと非常に遅い）

➔ int4 データ型の利用（4words を 1 回のメモリアクセスで可能）

int4 に変更しても性能差はなし(効果なし)

➔ Fermi アーキで追加された L2 キャッシュの効果？

➔ **Seed を 4words 構成にしても性能面での問題はない**

4. GPU のチューニング

4.1 CPU/GPU 間の負荷の最適化
4.2 CPU/GPU 間の通信オーバヘッド
4.3 MPSZ（スレッドブロック数）と求解時間の関係

4.1 CPU/GPU 間の負荷の最適化

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(s)	22.88	7.25	5.40	4.93	4.88
GPU制御部(s)	1.51	1.03	0.51	0.25	0.22
カーネル関数部(s)	22.05	21.24	17.72	17.72	22.78

ただし、n = (N – 1) / 2

K-line の値の減少 → Seeds 総数が減少

→ CPU 側(Seeds 生成部)の負荷は減少

→GPU 側(カーネル関数部)の負荷も減少

（**GPU 側の負荷減少の理由は、Seeds 取り込み回数の減少**）

その後、GPU の探索が深くなるとスレッドの多重度が下がり、性能低下

K-line の最適値は、n-2, n-3

4.2 CPU/GPU 間の通信オーバヘッド

(GPU 制御部の処理時間)

✧ GPU の起動・終了時間　　→　36μs 程度 / 1 回（問題なし）

✧ Seeds を GPU（VRAM）に転送する時間

N が小：問題なし（N=19 の場合、求解時間 9.98 秒の約 5%）

N が大：PoolSize を大きくすると若干の改善あり

(非同期処理の待ち合わせ時間の減少)

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

N が 21 以上では、PoolSize＝10M 程度がよい

4.3 MPSZ（スレッドブロック数）と求解時間の関係

BLKSZ=32 に固定し MPSZ を 16 の倍数で変化（N=19 の場合）

MPSZ	求解時間(s)（2words）	求解時間2(s)（2words）	参考値(s)（4words）
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		

再測定
BLKSZ=64

参考値は、スタック情報に
4words を使用した時の測定値。
この時、MPSZ の最大値は 96。

1 MP 当たりの常駐
ブロック数の最大値

最新版の MPSZ の最大値は 192(12x16)だが、

✧ **MPSZ=128 以降はほぼ横ばい(アーキの制限：8 x 16 = 128 が上限)**

✧ **BLKSZ=64 で再測定し、若干の改善あり(求解時間 2)**

5. 評価実験

5.1 評価環境
5.2 最新版の求解時間
5.3 他のプログラムとの比較
5.4 各施策の効果（N=19 の求解時間の推移）
5.5 各施策の効果(改善率)

5.1 評価環境

【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	

【GPU パラメタの定義】

（共有メモリの必要量などは“**N=27 まで測定可能な範囲**”に設定）

- ✧ MPSZ　：スレッドブロック数
- ✧ BLKSZ　：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 1 回の転送の最大数)

5.2 最新版の求解時間

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)

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

N=23 までは実測値（ただし、N=23 は PoolSize=10M で測定）

- N=23 の求解時間：電通大の PC クラスタ(Intel Pentium4 Xeon 2.8GHz CPU x 68）の求解時間(56.9H)に較べて、約 4.5 倍高速
- N=27 の求解時間の予測値：100 台 x 約 60 日

N=19～23 の求解時間の伸び率は、解の総数の伸び率に比例

N=25,27 の求解時間は、この関係を利用して求めた予測値

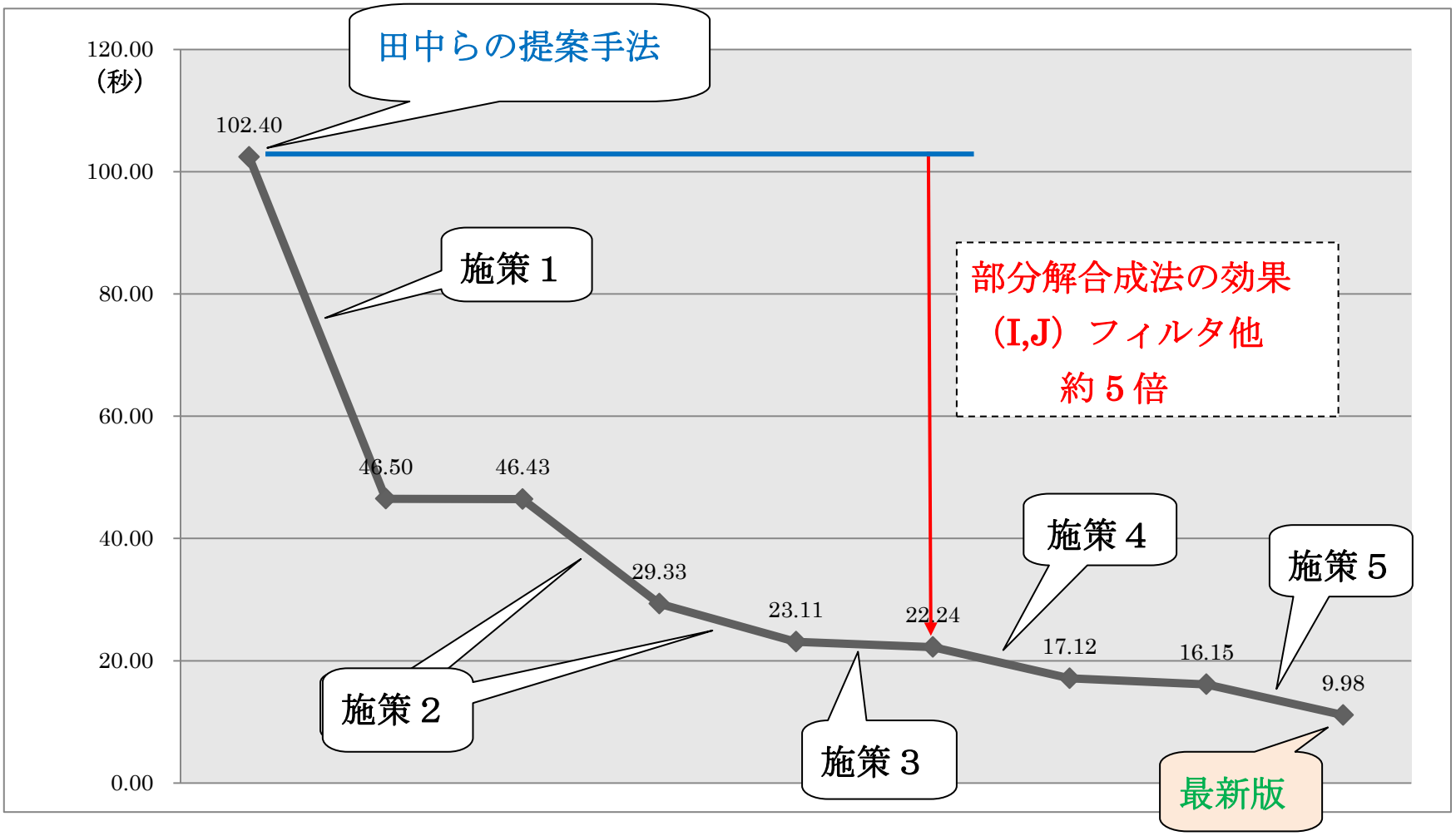
5.3 他のプログラムとの比較

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	0.5	1.90	0.51	3.7	54.9
19	1,617.00	107.00	1.0	102.40	9.98	10.3	162.0
21	111,425.00	5,919.82	1.2	7,103.24	591.56	12.0	188.4

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
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

- **田中らの提案手法との比較：10.3～12.0 倍高速**
  また、クイーン数の増加に伴い改善効果も向上
- **単一 CPU との比較：Somers 版の 162～188 倍高速**
- 部分解合成法は GPU を使った田中らの提案手法と同等以上の性能

5.4 各施策の効果（N=19 の求解時間の推移）



5.5 各施策の効果(改善率)

	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 の効果：55%　探索量の差は約 4 倍弱なので、改善率 55%ではまだ不十分

②施策 2 の効果：50%　Seeds 生成処理の軽減とカーネル関数部の Seeds 取込み処理の削減

③施策 3 の効果：　4%　１つの Seed の重さ(必要となる計算量)が軽いため効果小

④施策 4 の効果：23%　CPU 側の Seeds 生成処理時間の削減

⑤施策 5 の効果：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

6. まとめ

本提案手法

- ✧ 田中らの提案手法をベース
- ✧ クイーンの数（N）を奇数に限定
- ✧ Somers アルゴリズムと部分解合成法の「代表解の選択条件」との融合
- ✧ CPU と GPU の協調作業（適切な役割分担、並行処理）

成果

- ➢ 田中らの提案手法に較べて 10.3～12.0 倍
- ➢ Somers アルゴリズム(単一 CPU)との比較では 162～188 倍
- ➢ N=23 の求解時間は、電通大の PC クラスタ (Intel Pentium4 Xeon 2.8GHz CPU x 68) の約 4.5 倍高速

今後の課題

- ➢ N=27 への挑戦（環境が整えば） ： GPU 付き PC 100 台 x 60 日程度

補遺 最新版(論文提出後)の求解時間

スタック情報 b, bitmap を int2 データ型に変更→約４％の改善

（実行ステッ数の削減効果がバンク衝突のペナルティをカバー）

クイーン数	解の数	求解時間(s)	備考
15	2,279,184	0.22	<--0.20
17	95,815,104	0.50	<--0.51
19	4,968,057,848	9.72	<--9.98
21	314,666,222,712	566.70	<--591.56
23	24,233,937,684,440	43782.58	<--45818.24
		（12H 09M 43S）	
25	2,207,893,435,808,350	約50日	(予測値)
27	未解決	約55日 x 100台	(予測値)

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

N=23 までは実測値（ただし、N=23 は PoolSize=10M で測定）

✧ N=23 の求解時間：電通大の PC クラスタの約 4.7 倍高速

✧ N=27 の求解時間の予測値：100 台 x 約 55 日

✧ 単一 CPU との比較　166～196 倍高速

✧ 田中らの手法との比較　10.5～12.5 倍高速

ご清聴ありがとうございました

本論文で実験に使用したプログラム(nq_symG)のソースを
下記の Web サイトにて公開予定

http://deepgreen.game.cocon.jp/nqueens_index.html