rank 辞書構築の GPU による並列化

田中 秀宗, 渡辺 治

東京工業大学

ERATO 合宿
2011 年 11 月 1 日

背景

- GPU の汎用計算への活用
  - 物理シミュレーション，フーリエ変換，など
- 大規模データをコンパクトな領域で高速に処理
  ➔ 簡潔データ構造の活用
  - 簡潔データ構造
    データに補助情報を加えて，高速に様々な問合せに答える
    例：rank/select 辞書，wavelet tree，balanced parenthesis
  - 大規模データの簡潔データ構造を構築するのは時間がかかる

目標

GPU を用いて，大規模データに対する簡潔データ構造（rank 辞書）を高速に構築

背景

- GPU の汎用計算への活用
  - 物理シミュレーション，フーリエ変換，など
- 大規模データをコンパクトな領域で高速に処理
  ➔ 簡潔データ構造の活用
  - 簡潔データ構造
    データに補助情報を加えて，高速に様々な問合せに答える
    例：rank/select 辞書，wavelet tree，balanced parenthesis
  - 大規模データの簡潔データ構造を構築するのは時間がかかる

目標

GPU を用いて，大規模データに対する簡潔データ構造（rank 辞書）を高速に構築

背景

- GPU の汎用計算への活用
  - 物理シミュレーション，フーリエ変換，など
- 大規模データをコンパクトな領域で高速に処理
  ➔ 簡潔データ構造の活用
  - 簡潔データ構造
    データに補助情報を加えて，高速に様々な問合せに答える
    例：rank/select 辞書，wavelet tree，balanced parenthesis
  - 大規模データの簡潔データ構造を構築するのは時間がかかる

目標

GPU を用いて，大規模データに対する簡潔データ構造（rank 辞書）を高速に構築

発表の概要

1. 背景
2. ランク辞書
3. CUDA プログラミングモデル
4. ランク辞書構築の並列化
5. 実験結果
6. まとめ

発表の概要

1. 背景
2. ランク辞書
3. CUDA プログラミングモデル
4. ランク辞書構築の並列化
5. 実験結果
6. まとめ

ランク

ビット列 $x \in \{0,1\}^n$ とビット $b \in \{0,1\}$ に対して,

$$\mathrm{rank}_b(x, i) := x[1..i] \text{ の中の } b \text{ の数.}$$

例：

	1	2	3	4	5	6	7	8	9	10
$x_1 =$	0	1	1	0	0	1	1	1	0	0
$x_2 =$	1	0	1	1	1	0	0	0	1	0

- $\mathrm{rank}_1(x_1, 4) = 2$
- $\mathrm{rank}_0(x_2, 7) = 3$

ランク

ビット列 $x \in \{0,1\}^n$ とビット $b \in \{0,1\}$ に対して,

$$\mathrm{rank}_b(x, i) := x[1..i] \text{ の中の } b \text{ の数.}$$

例：

	1	2	3	4	5	6	7	8	9	10
$x_1 =$	0	**1**	**1**	0	0	1	1	1	0	0
$x_2 =$	1	**0**	1	1	1	**0**	**0**	0	1	0

- $\mathrm{rank}_1(x_1, 4) = 2$
- $\mathrm{rank}_0(x_2, 7) = 3$

目標

ビット列 $x \in \{0,1\}^n$ とビット $b \in \{0,1\}$ に対して,

$$\mathrm{rank}_b(x, i) := x[1..i] \text{ の中の } b \text{ の数.}$$

ビット列 $x \in \{0,1\}^n$ から,

- $\mathrm{rank}_b(x, i)$ を $O(1)$ 時間で計算する
- $o(n)$ 領域の

データ構造（ランク辞書）を GPU で高速に構築する

ランク辞書の構築（Jacobson '89）

注意：$\mathrm{rank}_0(B, i) = i - \mathrm{rank}_1(B, i)$ なので，$\mathrm{rank}_1(B, i)$ のみ計算

	↓					↓					↓		↓	↓				
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
B	0	1	1	0	1	1	1	1	0	1	0	0	0	1	1	1	0	0

1. B を長さ L の大ブロックに分割（$L = \log^2 n$）
   - 各ブロックの開始位置までのランクを LT に記録
2. B を長さ S の小ブロックに分割（$S = \log n/2$）
   - 各小ブロックの開始位置までのランクを ST に記録
   - 記録する値は大ブロックからの相対値

ランク辞書を使ってランクを計算

$$\mathrm{rank}_1(B, i) = \mathrm{LT}[i/L] + \mathrm{ST}[i/S] + \text{残りの rank}$$

ランク辞書の構築（Jacobson '89）

注意：$\mathrm{rank}_0(B, i) = i - \mathrm{rank}_1(B, i)$ なので，$\mathrm{rank}_1(B, i)$ のみ計算

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
B	0	1	1	0	1	1	1	1	0	1	0	0	0	1	1	1	0	0
LT	0						4						7					

1. B を長さ L の大ブロックに分割（$L = \log^2 n$）
   - 各ブロックの開始位置までのランクを LT に記録
2. B を長さ S の小ブロックに分割（$S = \log n/2$）
   - 各小ブロックの開始位置までのランクを ST に記録
   - 記録する値は大ブロックからの相対値

ランク辞書を使ってランクを計算

$$\mathrm{rank}_1(B, i) = \mathrm{LT}[i/L] + \mathrm{ST}[i/S] + \text{残りの rank}$$

ランク辞書の構築（Jacobson '89）

注意：$\mathrm{rank}_0(B, i) = i - \mathrm{rank}_1(B, i)$ なので，$\mathrm{rank}_1(B, i)$ のみ計算

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
B	0	1	1	0	1	1	1	1	0	1	0	0	0	1	1	1	0	0
LT	0						4						7					

1. B を長さ L の大ブロックに分割（$L = \log^2 n$）
   - 各ブロックの開始位置までのランクを LT に記録
2. B を長さ S の小ブロックに分割（$S = \log n/2$）
   - 各小ブロックの開始位置までのランクを ST に記録
   - 記録する値は大ブロックからの相対値

ランク辞書を使ってランクを計算

$$\mathrm{rank}_1(B, i) = \mathrm{LT}[i/L] + \mathrm{ST}[i/S] + \text{残りの rank}$$

ランク辞書の構築（Jacobson '89）

注意：$\mathrm{rank}_0(B, i) = i - \mathrm{rank}_1(B, i)$ なので，$\mathrm{rank}_1(B, i)$ のみ計算

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
B	0	1	1	0	1	1	1	1	0	1	0	0	0	1	1	1	0	0
LT	0						4						7					

1. B を長さ L の大ブロックに分割（$L = \log^2 n$）
   - 各ブロックの開始位置までのランクを LT に記録
2. B を長さ S の小ブロックに分割（$S = \log n/2$）
   - 各小ブロックの開始位置までのランクを ST に記録
   - 記録する値は大ブロックからの相対値

ランク辞書を使ってランクを計算

$$\mathrm{rank}_1(B, i) = \mathrm{LT}[i/L] + \mathrm{ST}[i/S] + \text{残りの rank}$$

ランク辞書の構築（Jacobson '89）

注意：$\mathrm{rank}_0(B, i) = i - \mathrm{rank}_1(B, i)$ なので，$\mathrm{rank}_1(B, i)$ のみ計算

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
B	0	1	1	0	1	1	1	1	0	1	0	0	0	1	1	1	0	0
LT	0						4						7					
ST	0		1		2		0		2		3		0		1		3	

1. B を長さ L の大ブロックに分割（$L = \log^2 n$）
   - 各ブロックの開始位置までのランクを LT に記録
2. B を長さ S の小ブロックに分割（$S = \log n/2$）
   - 各小ブロックの開始位置までのランクを ST に記録
   - 記録する値は大ブロックからの相対値

ランク辞書を使ってランクを計算

$$\mathrm{rank}_1(B, i) = \mathrm{LT}[i/L] + \mathrm{ST}[i/S] + \text{残りの rank}$$

ランク辞書の構築（Jacobson '89）

注意：$\mathrm{rank}_0(B, i) = i - \mathrm{rank}_1(B, i)$ なので，$\mathrm{rank}_1(B, i)$ のみ計算

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
B	0	1	1	0	1	1	1	1	0	1	0	0	0	1	1	1	0	0
LT	0						4						7					
ST	0		1		2		0		2		3		0		1		3	

1. B を長さ L の大ブロックに分割（$L = \log^2 n$）
   - 各ブロックの開始位置までのランクを LT に記録
2. B を長さ S の小ブロックに分割（$S = \log n/2$）
   - 各小ブロックの開始位置までのランクを ST に記録
   - 記録する値は大ブロックからの相対値

ランク辞書を使ってランクを計算

$$\mathrm{rank}_1(B, i) = \mathrm{LT}[i/L] + \mathrm{ST}[i/S] + \text{残りの rank}$$

ランク辞書の構築（Jacobson '89）

注意：$\mathrm{rank}_0(B, i) = i - \mathrm{rank}_1(B, i)$ なので，$\mathrm{rank}_1(B, i)$ のみ計算

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
B	0	1	1	0	1	1	1	1	0	1	0	0	0	1	1	1	0	0
LT	0						4						7					
ST	0		1		2		0		2		3		0		1		3	

1. B を長さ L の大ブロックに分割（$L = \log^2 n$）
   - 各ブロックの開始位置までのランクを LT に記録
2. B を長さ S の小ブロックに分割（$S = \log n/2$）
   - 各小ブロックの開始位置までのランクを ST に記録
   - 記録する値は大ブロックからの相対値

ランク辞書を使ってランクを計算

$$\mathrm{rank}_1(B, i) = \mathrm{LT}[i/L] + \mathrm{ST}[i/S] + \text{残りの rank}$$

ランク辞書の構築（Jacobson '89）

注意：$\mathrm{rank}_0(B, i) = i - \mathrm{rank}_1(B, i)$ なので，$\mathrm{rank}_1(B, i)$ のみ計算

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
B	0	1	1	0	1	1	1	1	0	1	0	0	0	1	1	1	0	0
LT	0						4						7					
ST	0		1		2		0		2		3		0		1		3	

1. B を長さ L の大ブロックに分割（$L = \log^2 n$）
   - 各ブロックの開始位置までのランクを LT に記録
2. B を長さ S の小ブロックに分割（$S = \log n/2$）
   - 各小ブロックの開始位置までのランクを ST に記録
   - 記録する値は大ブロックからの相対値

ランク辞書を使ってランクを計算

$$\mathrm{rank}_1(B, i) = \mathrm{LT}[i/L] + \mathrm{ST}[i/S] + \text{残りの rank}$$

ランク辞書の構築（Jacobson '89）

注意：$\mathrm{rank}_0(B, i) = i - \mathrm{rank}_1(B, i)$ なので，$\mathrm{rank}_1(B, i)$ のみ計算

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
B	0	1	1	0	1	1	1	1	0	1	0	0	0	1	1	1	0	0
LT	0						4						7					
ST	0		1		2		0		2		3		0		1		3	

1. B を長さ L の大ブロックに分割（$L = \log^2 n$）
   - 各ブロックの開始位置までのランクを LT に記録
2. B を長さ S の小ブロックに分割（$S = \log n/2$）
   - 各小ブロックの開始位置までのランクを ST に記録
   - 記録する値は大ブロックからの相対値

ランク辞書を使ってランクを計算

$$\mathrm{rank}_1(B, i) = \mathrm{LT}[i/L] + \mathrm{ST}[i/S] + \text{残りの rank}$$

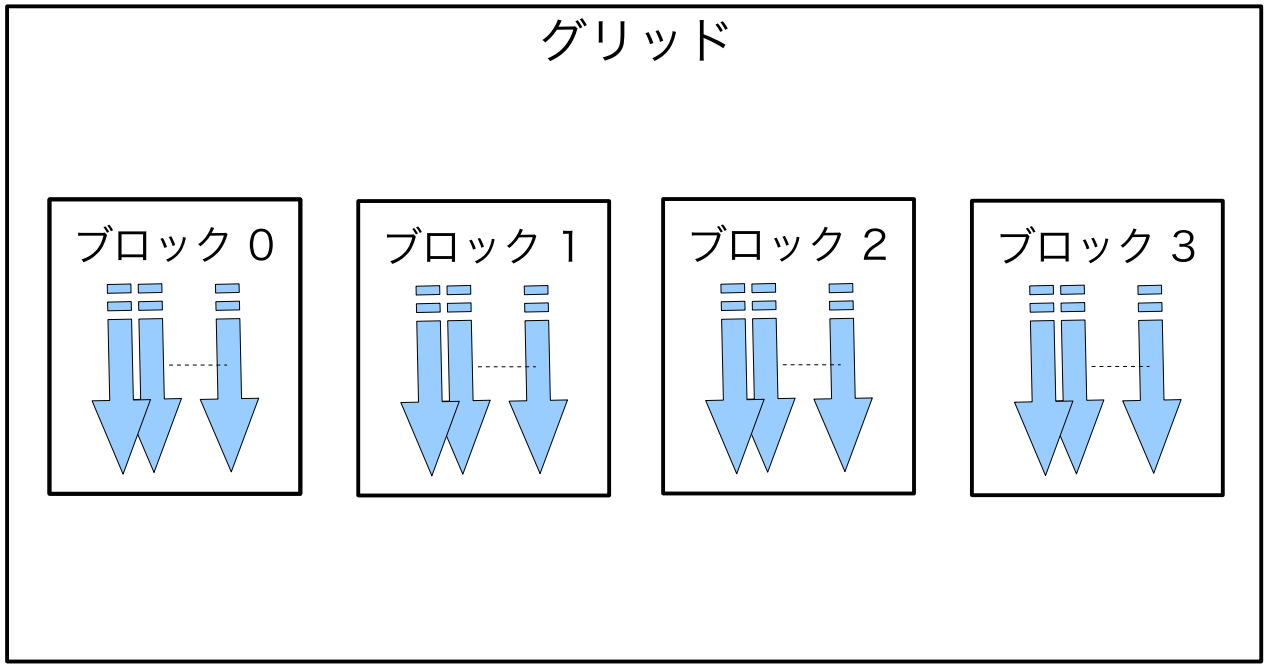
発表の概要

1. 背景
2. ランク辞書
3. CUDA プログラミングモデル
4. ランク辞書構築の並列化
5. 実験結果
6. まとめ

CUDA

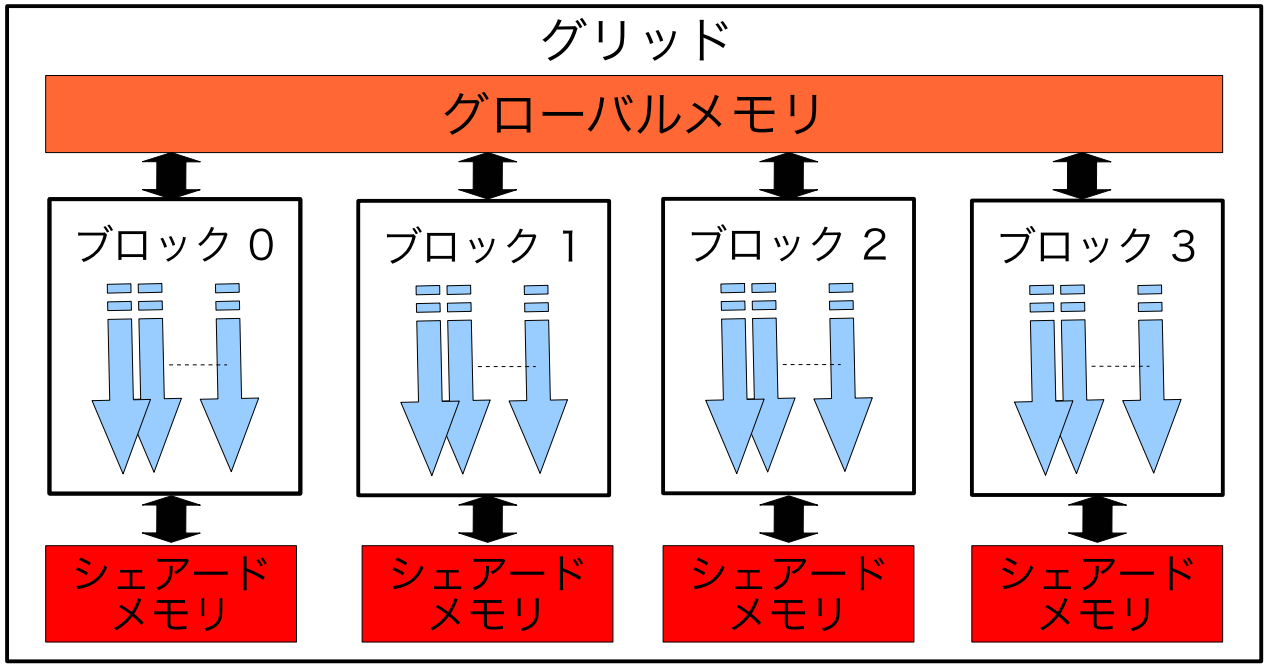
- GPU のための並列プログラミングモデル
- NVIDIA 製の GPU によりサポート
- C / C++ 用の開発環境
- Single Instruction Multiple Thread（SIMT）モデル
  - 32 スレッドが同時に動作
  - 分岐はオーバーヘッドになる

スレッド階層



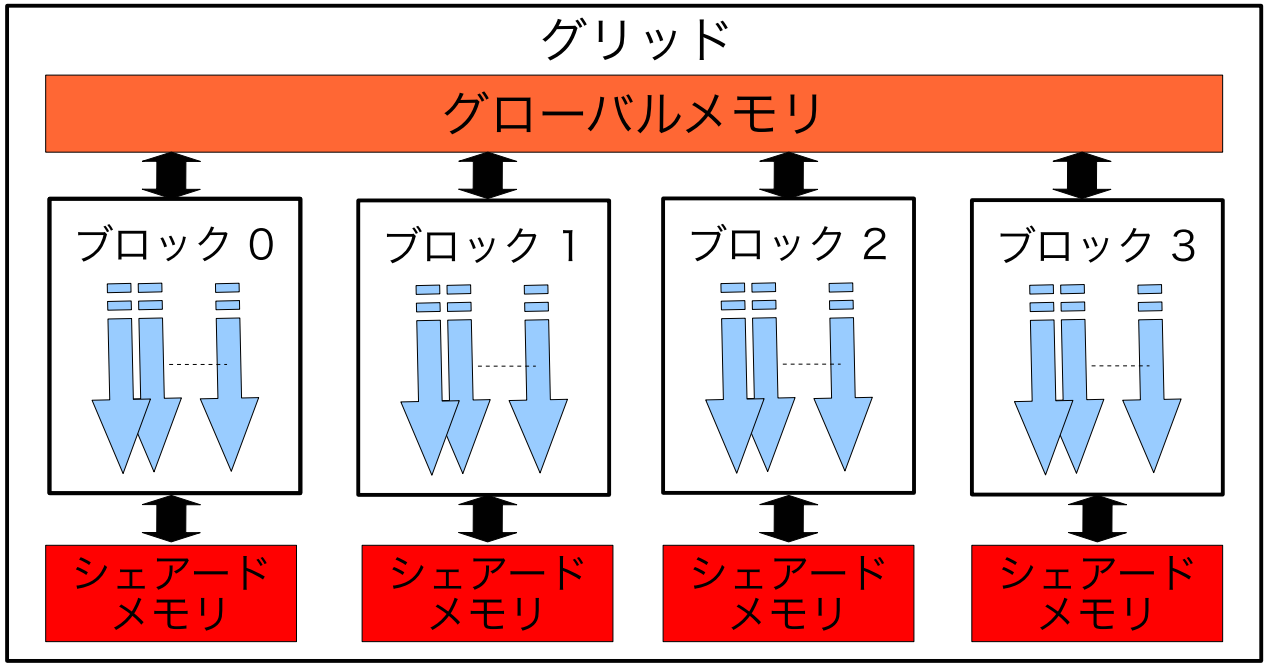
- スレッド  ：プログラムを実行する最小単位
- スレッドブロック（ブロック）：スレッドの集まり
- グリッド：スレッドブロックの集まり

メモリ階層



- グローバルメモリ
  全スレッドからアクセス可能, 容量大, 低速
  - 4GB
- シェアードメモリ
  同一ブロック内のスレッドからのみアクセス可能, 容量小, 高速
  - 1 スレッドブロック当たり 49kB

CUDA プログラミングモデル



注意

- スレッドブロック中のスレッド数，全スレッドブロック数に制限有り
  - スレッドブロック中のスレッドの最大数 $= 1024 \times 1024 \times 64$
    （ふつう 128〜256 程度）
  - スレッドブロックの最大数 $= 65535 \times 65535 \times 65535$
    （ふつう 200〜300 程度）
- 同じスレッドブロック中のスレッドのみ同期が可能

発表の概要

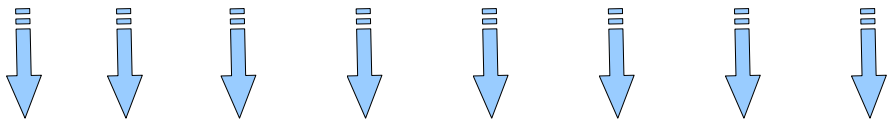
1. 背景
2. ランク辞書
3. CUDA プログラミングモデル
4. ランク辞書構築の並列化
5. 実験結果
6. まとめ

並列化のための道具

1. Population count (Popcount)
   - 32bit / 64bit 変数中のビット 1 の数を数える
   - 最近の GPU ではハードウェア実装
2. Prefix sum
   - 整数列 $(x_1, x_2, \ldots, x_k, \ldots, x_n)$ から
     整数列 $\left(x_1, x_1 + x_2, \ldots, \sum_{i=1}^{k} x_k, \ldots, \sum_{i=1}^{n} x_i\right)$ を求める
   - $O(\log n)$ 時間並列アルゴリズムが知られている

Prefix Sum

- 1 つの要素を 1 つのスレッドに割当て
- 各 i ステップ目で，2^{i-1} 個左隣の要素と足し合わせる

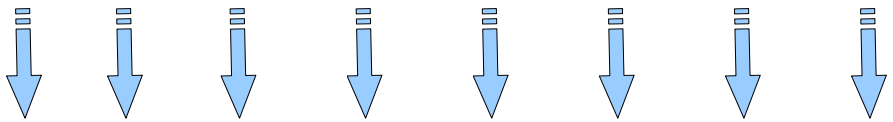


入力	5	3	9	7	6	2	3	1
1 ステップ目	5							
2 ステップ目								
3 ステップ目								
出力	5	8	17	24	30	32	35	36

注意：各ステップで同期が必要

Prefix Sum

- 1 つの要素を 1 つのスレッドに割当て
- 各 i ステップ目で，2^{i-1} 個左隣の要素と足し合わせる

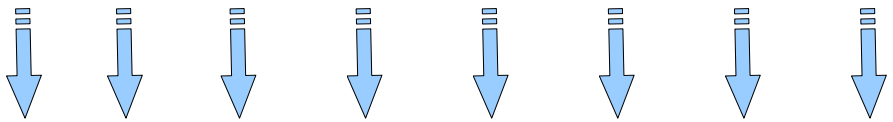


入力	5	3	9	7	6	2	3	1
1 ステップ目	5	8	12	16	13	8	5	4
2 ステップ目								
3 ステップ目								
出力	5	8	17	24	30	32	35	36

注意：各ステップで同期が必要

Prefix Sum

- 1 つの要素を 1 つのスレッドに割当て
- 各 i ステップ目で，2^{i-1} 個左隣の要素と足し合わせる

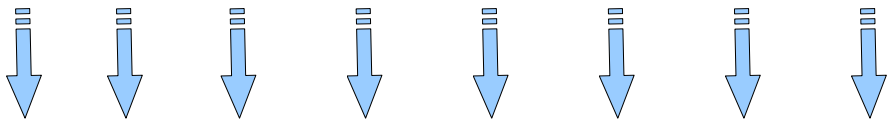


入力	5	3	9	7	6	2	3	1
1 ステップ目	5	8	12	16	13	8	5	4
2 ステップ目								
3 ステップ目								
出力	5	8	17	24	30	32	35	36

注意：各ステップで同期が必要

Prefix Sum

- 1 つの要素を 1 つのスレッドに割当て
- 各 i ステップ目で，2^{i-1} 個左隣の要素と足し合わせる

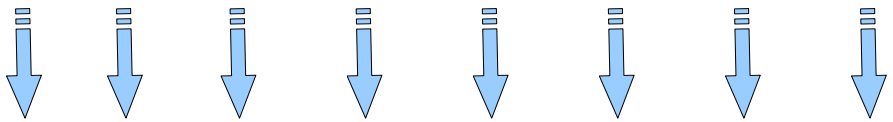


入力	5	3	9	7	6	2	3	1
1 ステップ目	5	8	12	16	13	8	5	4
2 ステップ目	5	8	17	24	25	24	18	12
3 ステップ目								
出力	5	8	17	24	30	32	35	36

注意：各ステップで同期が必要

Prefix Sum

- 1 つの要素を 1 つのスレッドに割当て
- 各 i ステップ目で，2^{i-1} 個左隣の要素と足し合わせる

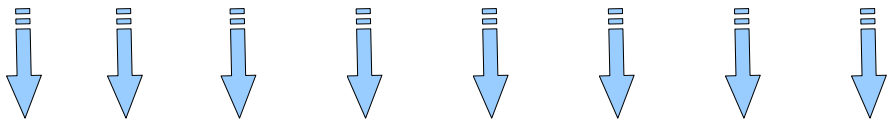


入力	5	3	9	7	6	2	3	1
1 ステップ目	5	8	12	16	13	8	5	4
2 ステップ目	5	8	17	24	25	24	18	12
3 ステップ目								
出力	5	8	17	24	30	32	35	36

注意：各ステップで同期が必要

Prefix Sum

- 1 つの要素を 1 つのスレッドに割当て
- 各 i ステップ目で，2^{i-1} 個左隣の要素と足し合わせる



入力	5	3	9	7	6	2	3	1
1 ステップ目	5	8	12	16	13	8	5	4
2 ステップ目	5	8	17	24	25	24	18	12
3 ステップ目	5	8	17	24	30	32	35	36
出力	5	8	17	24	30	32	35	36

注意：各ステップで同期が必要

並列化の指針

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
B	0	1	1	0	1	1	1	1	0	1	0	0	0	1	1	1	0	0
LT	0						4						7					
ST	0			1		2	0		2		3		0		1		3	

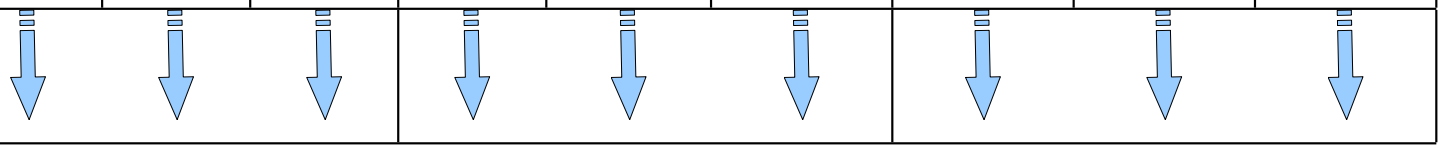
1. B を長さ L の大ブロックに分割（$L = \log^2 n$）
   - 各ブロックの開始位置までのランクを LT に記録
2. B を長さ S の小ブロックに分割（$S = \log n/2$）
   - 各小ブロックの開始位置までのランクを ST に記録
   - 記録する値は大ブロックからの相対値

スレッドとブロック

- 1 つの小ブロックを 1 つのスレッドに割当て
- 1 つの大ブロックを 1 つのスレッドブロックに割当て

並列化の指針

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
B	0	1	1	0	1	1	1	1	0	1	0	0	0	1	1	1	0	0
LT	0						4						7					
ST	0			1			2			0			2			3		



1. B を長さ L の大ブロックに分割（$L = \log^2 n$）
   - 各ブロックの開始位置までのランクを LT に記録
2. B を長さ S の小ブロックに分割（$S = \log n/2$）
   - 各小ブロックの開始位置までのランクを ST に記録
   - 記録する値は大ブロックからの相対値

スレッドとブロック

- 1 つの小ブロックを 1 つのスレッドに割当て
- 1 つの大ブロックを 1 つのスレッドブロックに割当て

並列アルゴリズムの概要

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
B	0	1	1	0	1	1	1	1	0	1	0	0	0	1	1	1	0	0

1. 各スレッドで小ブロックの値を Popcount ➔ シェアードメモリへ
2. 各大ブロック内で Prefix Sum を計算
3. シェアードメモリの値を大ブロック内で 1 つずらして ST に格納
4. 各大ブロック内で最後のシェアードメモリの値を
   1 つずらして LT に格納
5. LT の Prefix Sum を計算

並列アルゴリズムの概要

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
B	0	1	1	0	1	1	1	1	0	1	0	0	0	1	1	1	0	0
シェアードメモリ	1		1		2		2		1		0		1		2		0	

1. 各スレッドで小ブロックの値を Popcount ➔ シェアードメモリへ
2. 各大ブロック内で Prefix Sum を計算
3. シェアードメモリの値を大ブロック内で 1 つずらして ST に格納
4. 各大ブロック内で最後のシェアードメモリの値を 1 つずらして LT に格納
5. LT の Prefix Sum を計算

並列アルゴリズムの概要

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
B	0	1	1	0	1	1	1	1	0	1	0	0	0	1	1	1	0	0
シェアードメモリ	1		1		2		2		1		0		1		2		0	
➔ (Prefix sum)	1		2		4		2		3		3		1		3		3	

1. 各スレッドで小ブロックの値を Popcount ➔ シェアードメモリへ
2. 各大ブロック内で Prefix Sum を計算
3. シェアードメモリの値を大ブロック内で 1 つずらして ST に格納
4. 各大ブロック内で最後のシェアードメモリの値を 1 つずらして LT に格納
5. LT の Prefix Sum を計算

並列アルゴリズムの概要

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
B	0	1	1	0	1	1	1	1	0	1	0	0	0	1	1	1	0	0
シェアードメモリ	1		1		2		2		1		0		1		2		0	
➔ (Prefix sum)	1		2		4		2		3		3		1		3		3	
ST	0		1		2		0		2		3		0		1		3	

1. 各スレッドで小ブロックの値を Popcount ➔ シェアードメモリへ
2. 各大ブロック内で Prefix Sum を計算
3. シェアードメモリの値を大ブロック内で 1 つずらして ST に格納
4. 各大ブロック内で最後のシェアードメモリの値を 1 つずらして LT に格納
5. LT の Prefix Sum を計算

並列アルゴリズムの概要

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
B	0	1	1	0	1	1	1	1	0	1	0	0	0	1	1	1	0	0
シェアードメモリ	1		1		2		2		1		0		1		2		0	
➔ (Prefix sum)	1		2		4		2		3		3		1		3		3	
ST	0		1		2		0		2		3		0		1		3	
LT	0						4						3					

1. 各スレッドで小ブロックの値を Popcount ➔ シェアードメモリへ
2. 各大ブロック内で Prefix Sum を計算
3. シェアードメモリの値を大ブロック内で 1 つずらして ST に格納
4. 各大ブロック内で最後のシェアードメモリの値を 1 つずらして LT に格納
5. LT の Prefix Sum を計算

並列アルゴリズムの概要

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
B	0	1	1	0	1	1	1	1	0	1	0	0	0	1	1	1	0	0
シェアードメモリ	1		1		2		2		1		0		1		2		0	
➔ (Prefix sum)	1		2		4		2		3		3		1		3		3	
ST	0		1		2		0		2		3		0		1		3	
LT	0						4						3					
➔ (Prefix sum)	0						4						7					

1. 各スレッドで小ブロックの値を Popcount ➔ シェアードメモリへ
2. 各大ブロック内で Prefix Sum を計算
3. シェアードメモリの値を大ブロック内で 1 つずらして ST に格納
4. 各大ブロック内で最後のシェアードメモリの値を
   1 つずらして LT に格納
5. LT の Prefix Sum を計算

並列アルゴリズムの概要

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
B	0	1	1	0	1	1	1	1	0	1	0	0	0	1	1	1	0	0
ST	0		1		2		0		2		3		0		1		3	
LT	0						4						7					

1. 各スレッドで小ブロックの値を Popcount ➔ シェアードメモリへ
2. 各大ブロック内で Prefix Sum を計算
3. シェアードメモリの値を大ブロック内で 1 つずらして ST に格納
4. 各大ブロック内で最後のシェアードメモリの値を
   1 つずらして LT に格納
5. LT の Prefix Sum を計算

問題点

- LT が巨大で，シェアードメモリに載り切らない場合がある
  - $n = 2^{32}$ で 16MB

  ➔ 複数のスレッドブロックで計算する必要がある
- GPU 上でスレッドブロックをまたいだ同期は不可能

  ➔ 一度 CPU に処理を差し戻して同期

  ➔ 単純な実装だと大きなオーバーヘッドになりそう

Prefix sum を工夫して分割

問題点

- LT が巨大で，シェアードメモリに載り切らない場合がある
  - $n = 2^{32}$ で 16MB

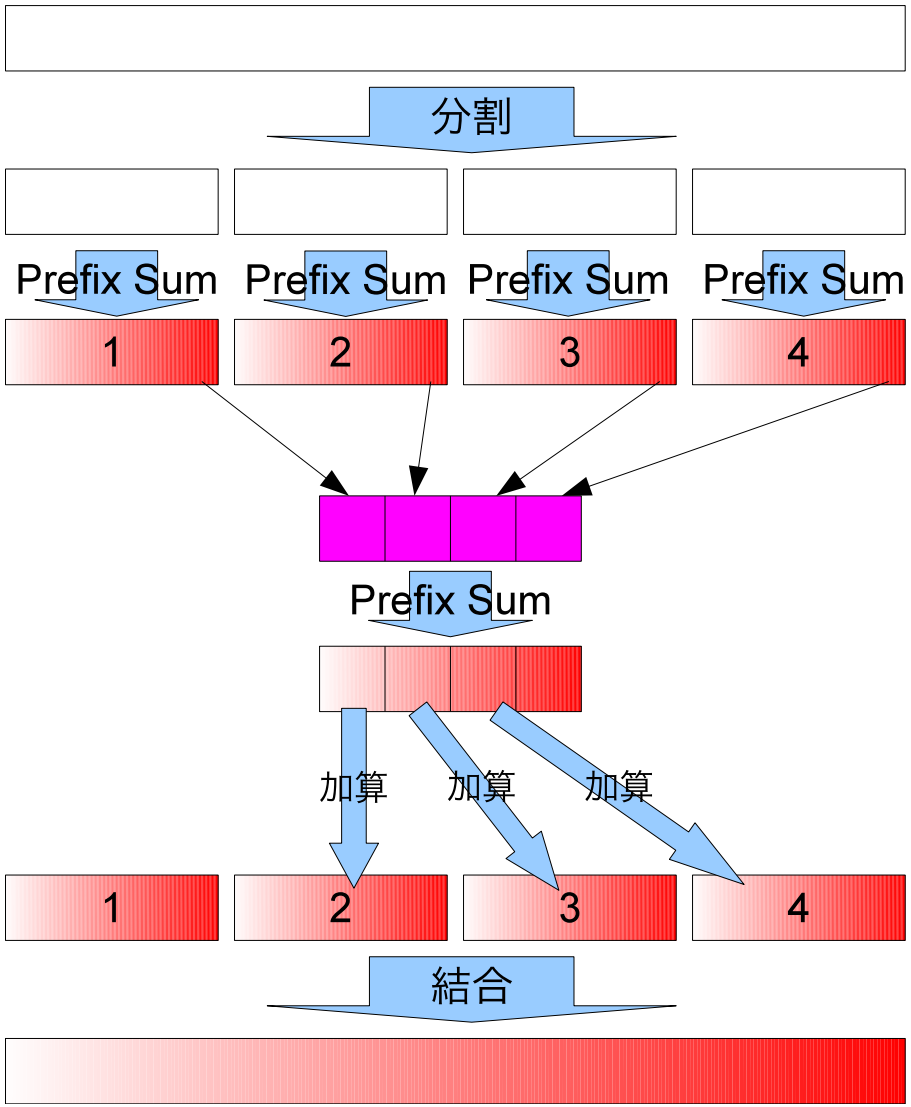
  ➔ 複数のスレッドブロックで計算する必要がある
- GPU 上でスレッドブロックをまたいだ同期は不可能

  ➔ 一度 CPU に処理を差し戻して同期

  ➔ 単純な実装だと大きなオーバーヘッドになりそう

Prefix sum を工夫して分割

Prefix Sum を分割



1. 配列を適当なブロックに分割
2. 各ブロックの Prefix Sum を計算
3. 各ブロックの最後の要素を取り出し，新たな配列を作る
4. 新たな配列の長さが長ければ 1 へ
5. 配列の Prefix Sum を計算
6. 自分の次のブロックの全要素に Prefix Sum の値を加える

発表の概要

1. 背景
2. ランク辞書
3. CUDA プログラミングモデル
4. ランク辞書構築の並列化
5. 実験結果
6. まとめ

実験

- 環境
  - CPU：AMD Phenom X4 9850 (2.5GHz)
  - GPU：Tesla C2070
    - クロック 1.15 GHz
    - グローバルメモリ 4 GB
    - シェアードメモリ 49 kB
- 比較対象

  Sux: Implementing Succinct Data Structures
  - Broadword Implementation of Rank / Select Queries.
  - S. Vigna
  - WEA 2008: 7th International Workshop on Experimental Algorithms (pp. 154-168).

実験 (1)

- スレッドブロック中のスレッドの数 $2 \log n \leqslant 64$
- スレッドブロックの数 256
  - 大ブロックの最大要素数は 4,194,304
    ➔ 大ブロックを各スレッドブロックで分割

	100Mbit	1Gbit	3Gbit
Sux(CPU)	0.932058 s	9.372586 s	28.005750 s
GPU	0.089909 s	0.397739 s	1.027892 s
CPU/GPU 比	10.37	23.56	27.25

- 通信時間 = 約 0.4 s（$n = 3\mathrm{G}$ のとき）

実験 (1)

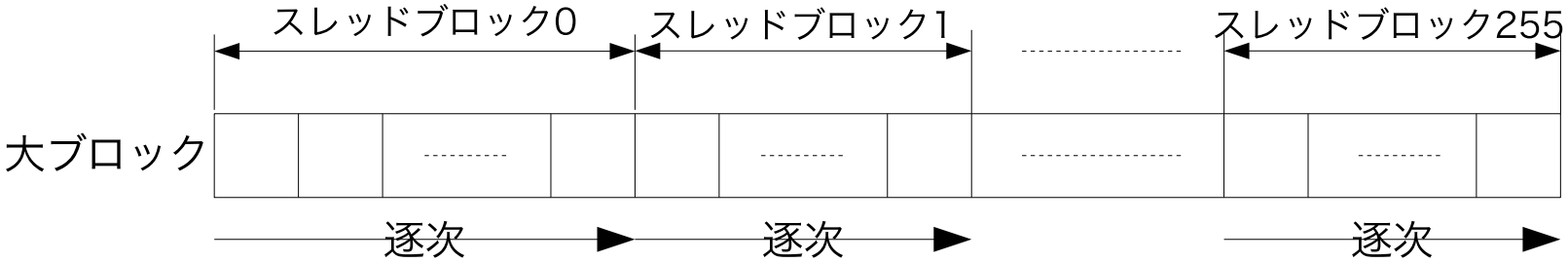
- スレッドブロック中のスレッドの数 $2 \log n \leqslant 64$
- スレッドブロックの数 256
  - 大ブロックの最大要素数は 4,194,304
    ➔ 大ブロックを各スレッドブロックで分割

	100Mbit	1Gbit	3Gbit
Sux(CPU)	0.932058 s	9.372586 s	28.005750 s
GPU	0.089909 s	0.397739 s	1.027892 s
CPU/GPU 比	10.37	23.56	27.25

- 通信時間 = 約 0.4 s（$n = 3G$ のとき）

実験 (1)

- スレッドブロック中のスレッドの数 $2 \log n \leqslant 64$
- スレッドブロックの数 256
  - 大ブロックの最大要素数は 4,194,304
    ➔ 大ブロックを各スレッドブロックで分割

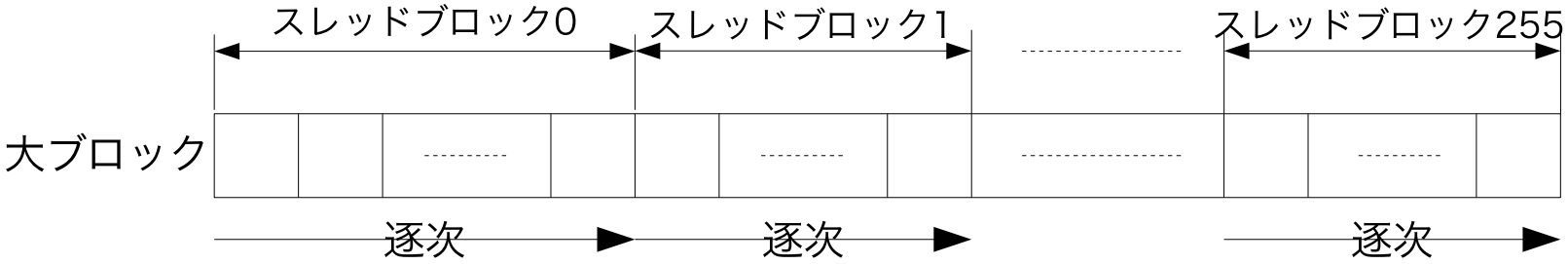


	100Mbit	1Gbit	3Gbit
Sux(CPU)	0.932058 s	9.372586 s	28.005750 s
GPU	0.089909 s	0.397739 s	1.027892 s
CPU/GPU 比	10.37	23.56	27.25

- 通信時間 = 約 0.4 s（$n = 3\mathrm{G}$ のとき）

実験 (1)

- スレッドブロック中のスレッドの数 $2 \log n \leqslant 64$
- スレッドブロックの数 256
  - 大ブロックの最大要素数は 4,194,304
    ➔ 大ブロックを各スレッドブロックで分割

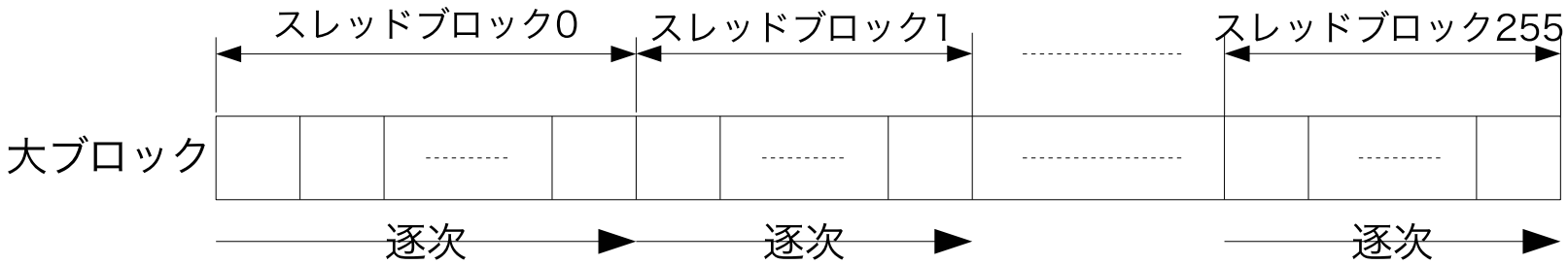


	100Mbit	1Gbit	3Gbit
Sux(CPU)	0.932058 s	9.372586 s	28.005750 s
GPU	0.089909 s	0.397739 s	1.027892 s
CPU/GPU 比	10.37	23.56	27.25

- 通信時間 = 約 0.4 s（$n = 3\mathrm{G}$ のとき）

実験 (1)

- スレッドブロック中のスレッドの数 $2 \log n \leqslant 64$
- スレッドブロックの数 256
  - 大ブロックの最大要素数は 4,194,304
    ➔ 大ブロックを各スレッドブロックで分割

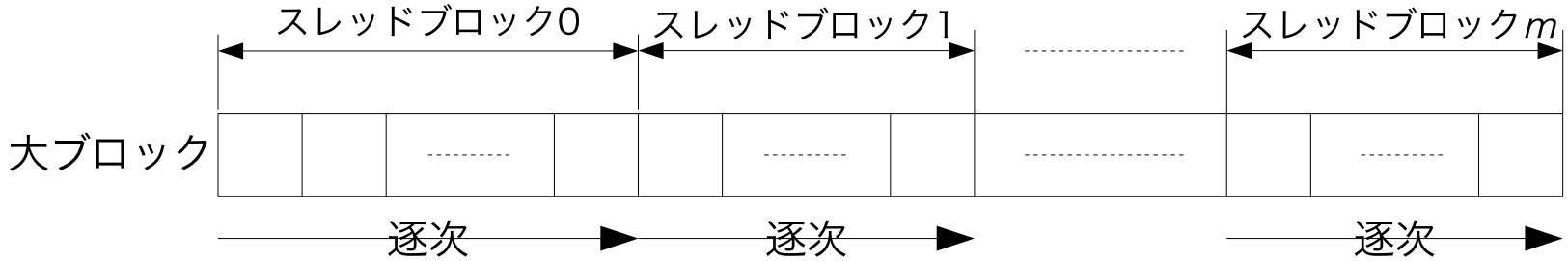


	100Mbit	1Gbit	3Gbit
Sux(CPU)	0.932058 s	9.372586 s	28.005750 s
GPU	0.089909 s	0.397739 s	1.027892 s
CPU/GPU 比	10.37	23.56	27.25

- 通信時間 = 約 0.4 s（$n = 3\mathrm{G}$ のとき）

実験 (2)：ブロック数を変化

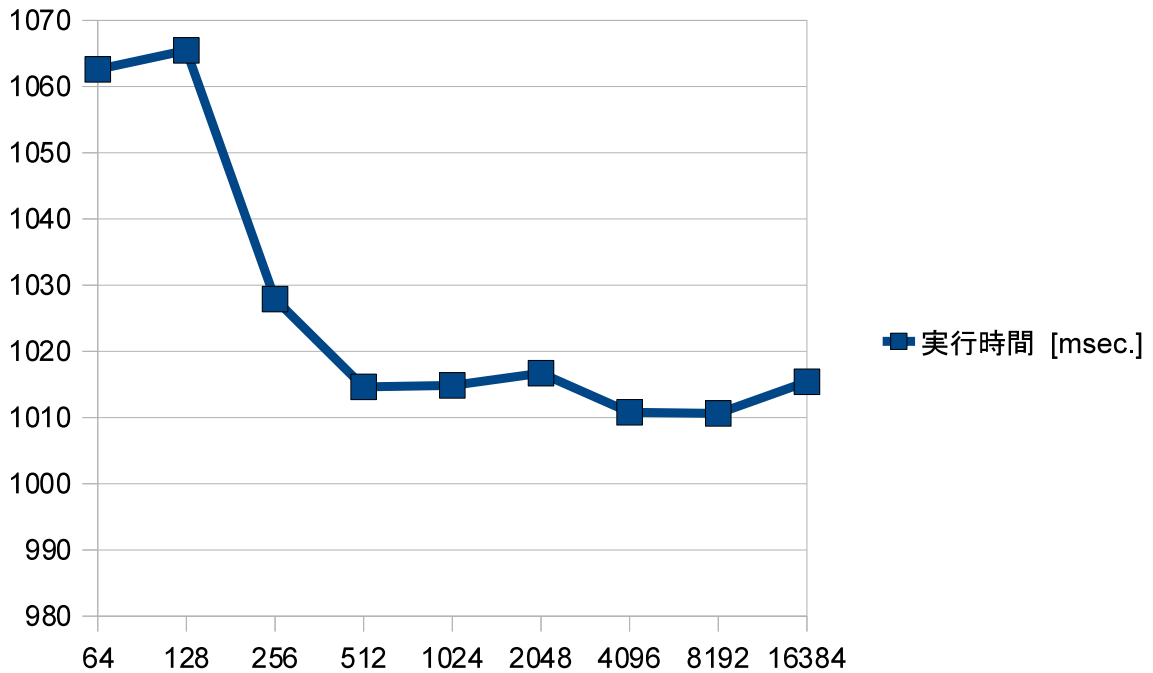
- $n = 3\mathrm{G}$
- スレッドブロック中のスレッドの数 $2 \log n \leqslant 64$
- スレッドブロックの数 m



- 最大 27.71 倍高速化

実験 (2)：ブロック数を変化

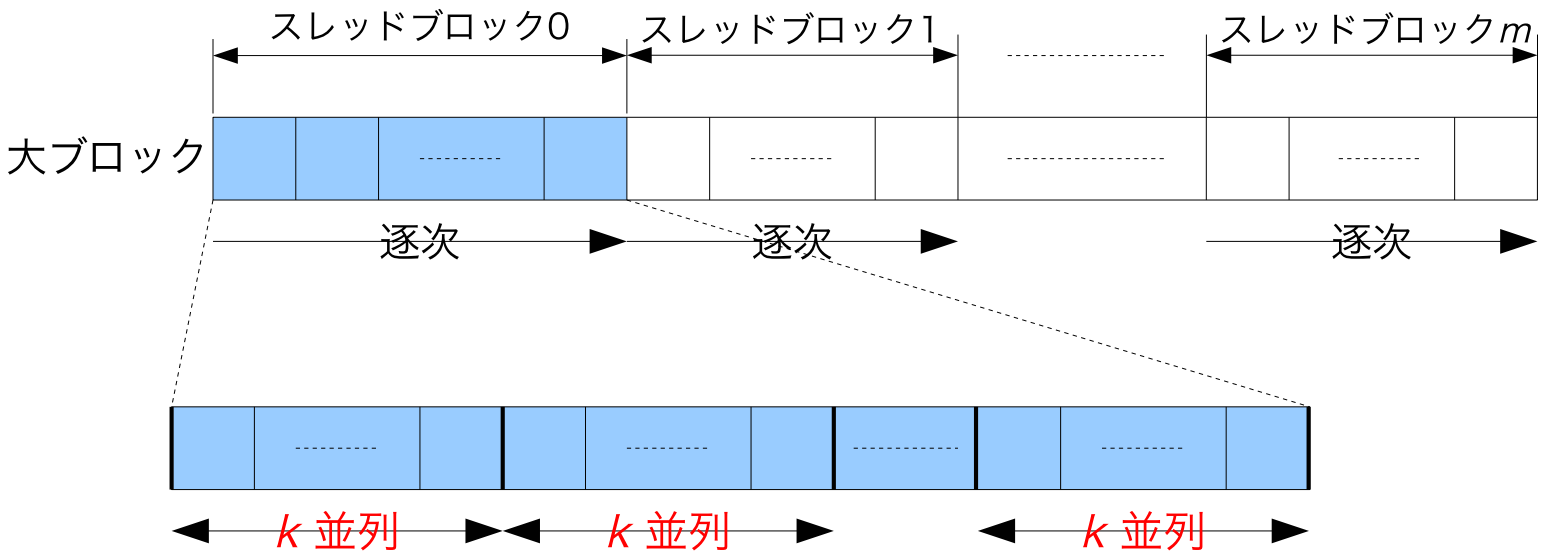
- $n = 3\mathrm{G}$
- スレッドブロック中のスレッドの数 $2 \log n \leqslant 64$
- スレッドブロックの数 m



- 最大 27.71 倍高速化

実験 (3)：ブロック当たりのスレッド数を変化

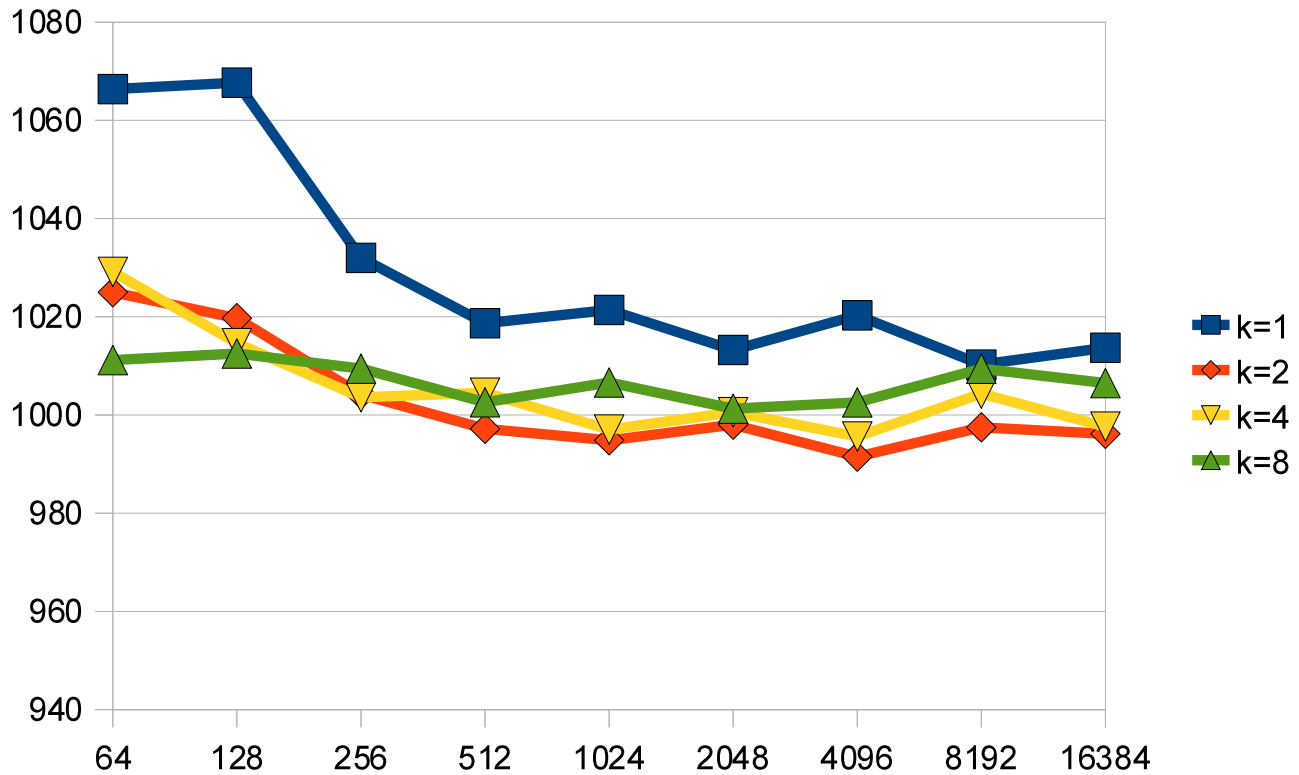
- $n = 3\mathrm{G}$
- スレッドブロック中のスレッドの数 $k \cdot 2 \log n \leqslant 64k$
- スレッドブロックの数 m



- 最大 28.25 倍の高速化

実験 (3)：ブロック当たりのスレッド数を変化

- $n = 3\mathrm{G}$
- スレッドブロック中のスレッドの数 $k \cdot 2 \log n \leqslant 64k$
- スレッドブロックの数 m



- 最大 28.25 倍の高速化

発表の概要

1. 背景
2. ランク辞書
3. CUDA プログラミングモデル
4. ランク辞書構築の並列化
5. 実験結果
6. まとめ

まとめ

- ランク辞書を GPU 上に構築する手法の提案
- 最大約 28 倍程度の高速化

今後の課題

応用を考える

- 「構築が速くできても意味が無い．問合せ時間が問題．」
- 何度もランク辞書を作る状況
- GPU 上で並列に問合せ

まとめ

- ランク辞書を GPU 上に構築する手法の提案
- 最大約 28 倍程度の高速化

今後の課題

応用を考える

- 「構築が速くできても意味が無い．問合せ時間が問題．」
- 何度もランク辞書を作る状況
- GPU 上で並列に問合せ