ヒストグラムと接頭辞和計算に基づく整数ソーティングのためのGPUによる効率的な実装

小堺海叶† 藤本典幸†† 和田幸一†

†法政大学

††大阪府立大学

はじめに

- 本実験を行うに至った背景

thrustソート

GPUの「thrust」ライブラリで最も速いソーティングアルゴリズム

thrustソートよりも高速なアルゴリズムを考案したい！

研究の動機

- thrustソートは汎用的なソーティングアルゴリズムである

- 整数ソーティングに特化したアルゴリズムならば、より高速なアルゴリズムを考案できるかもしれない

整数ソーティング

0以上の非負整数で最大値(maxVal)が制限された

入力データのソートを行う

研究で行ったこと1

- PRAM上での動作を前提に考案されていた、接頭辞和による整数ソーティングアルゴリズムをGPU上で実装した

H-Pソート

ヒストグラムと接頭辞和（Prefix Sums）に基づいた整数ソーティングアルゴリズム

研究で行ったこと2

- そしてH-Pソートの更なる高速化を目指して改良を加えたものを考案し、GPU上で実装した

0-圧縮H-Pソート

H-Pソートに改良を加えた整数ソーティングアルゴリズム

- 本実験では、この2つのソーティングアルゴリズムとthrustソートとの比較を行った

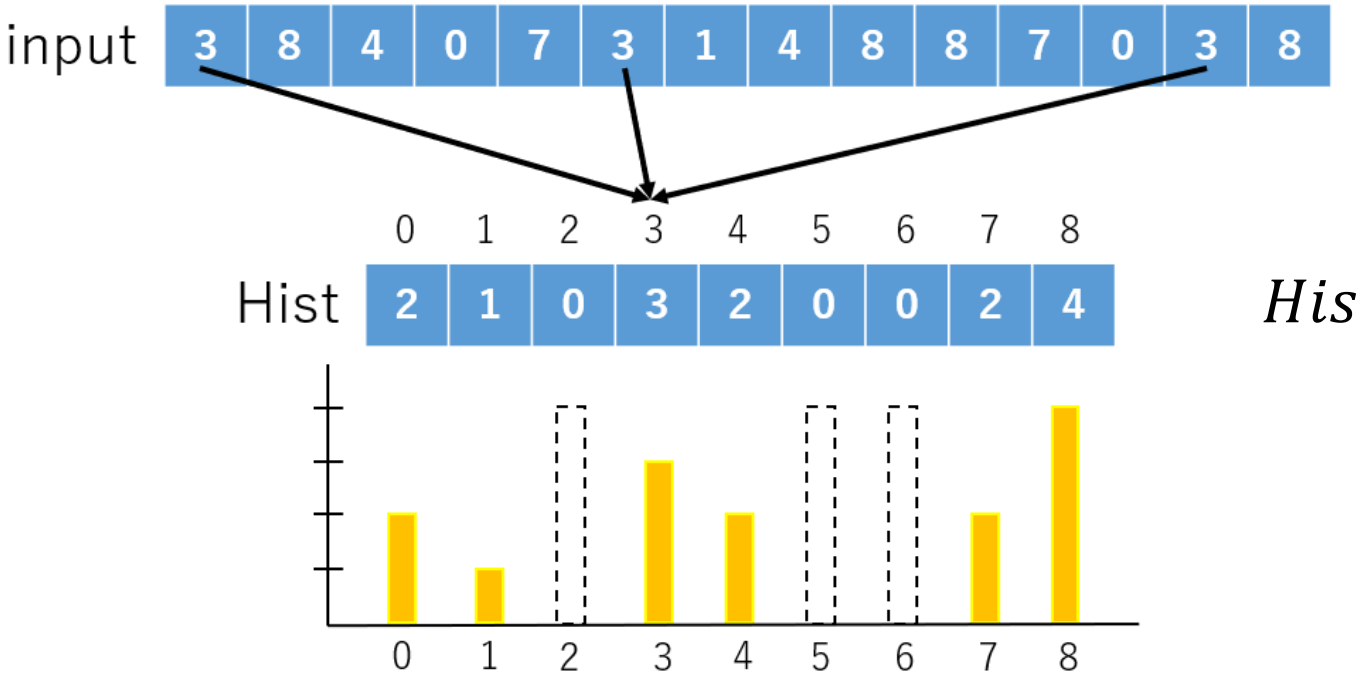
アルゴリズムについて

- thrustソートとの比較を行った2つのアルゴリズムは、いずれもヒストグラムと接頭辞和（Prefix Sums）に基づいたソーティングアルゴリズムである

➡ ヒストグラムと接頭辞和のアルゴリズムについて説明する

ヒストグラム

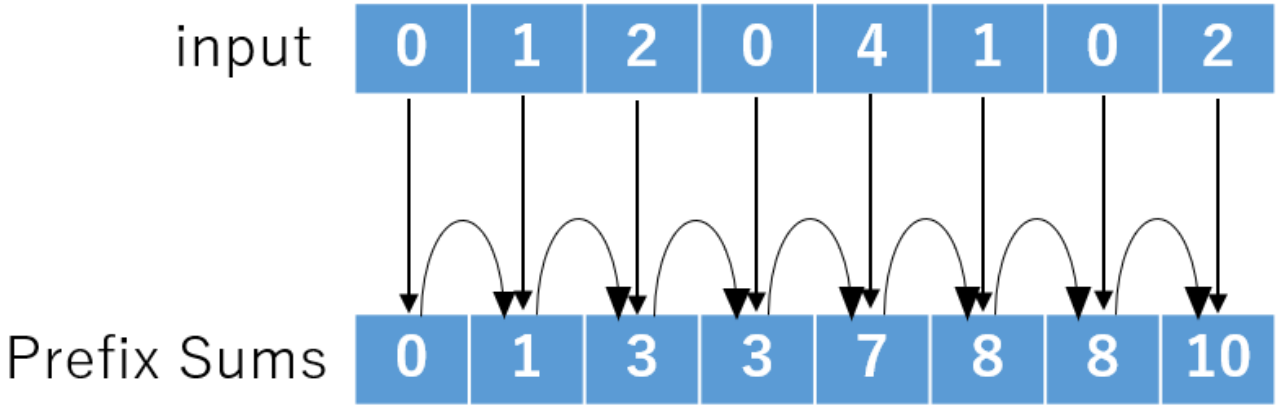
- 入力値の要素数を計数した度数分布表
- ヒストグラム生成の計算量　：　$O(n)$ ⇒ 入力データの大きさに依存



$$Hist[input[k]] \mathrel{+}= 1$$

接頭辞和（Prefix Sums）

- 入力配列が与えられたとき、出力配列の要素のk番目に、入力配列の0番目からk番目までの合計をそれぞれ出力する
- 接頭辞和の計算量　：　$O(n)$ ⇒ 入力データの大きさに依存



$$\text{Prefix Sums[k]} = input[0] + input[1] + \cdots + input[k]$$

H-Pソートのアルゴリズム

- ヒストグラムと接頭辞和（Prefix Sums）に基づいたソーティングアルゴリズムを「H-Pソート」と呼称する

<u>アルゴリズムのパラメータ</u>

n：入力データのデータ数

maxVal：入力データ内の最大値

<u>使用する配列</u>

入力配列x（size：n）、出力配列y（size：n）

補助配列A（size ：maxVal）、A_p（size：maxVal）、B（size：n+1）

H-Pソートのアルゴリズム

入力配列x（size：n）、出力配列y（size：n）

補助配列A（size ：maxVal）、A_p（size：maxVal）、B（size：n+1）

① x（n）のヒストグラムであるA（maxVal）を生成する

② Aに接頭辞和を適用し、A_p（maxVal）を生成する

③ A_pのヒストグラムであるB（n+1）を生成する

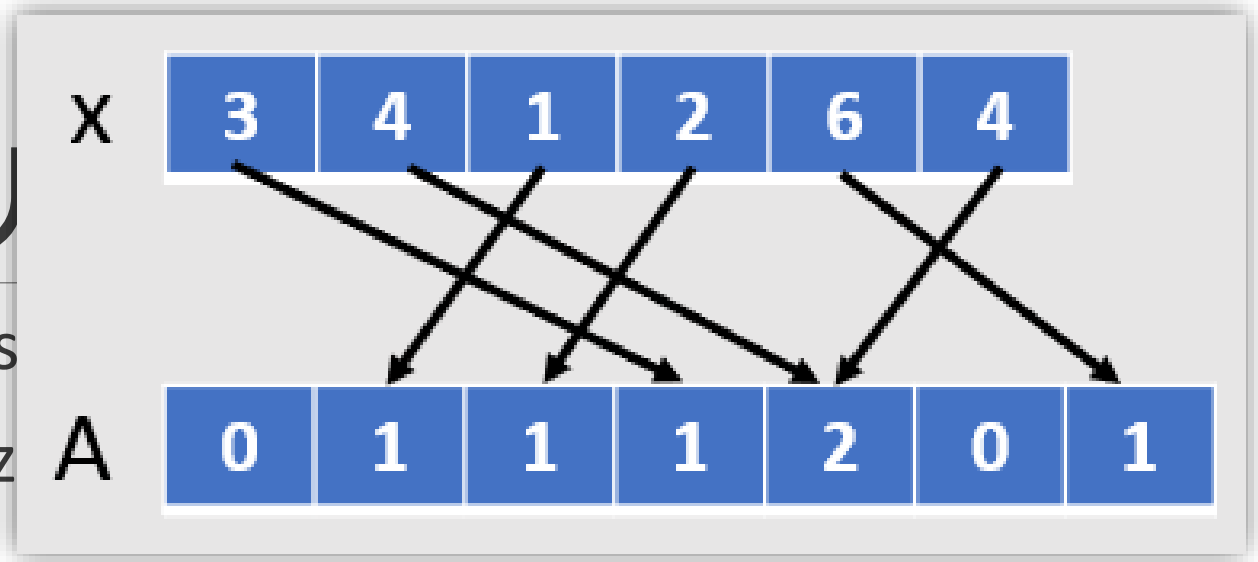
④ Bに接頭辞和を適用し、y（n）を生成する

※ヒストグラムを接頭辞和を2回ずつ行うことでソートを行う

H-Pソートのアルゴリ

入力配列x（size：n）、出力配列y（s

補助配列A（size ：maxVal）、A_p（siz



① x（n）のヒストグラムであるA（maxVal）を生成する

② Aに接頭辞和を適用し、A_p（maxVal）を生成する

③ A_pのヒストグラムであるB（n+1）を生成する

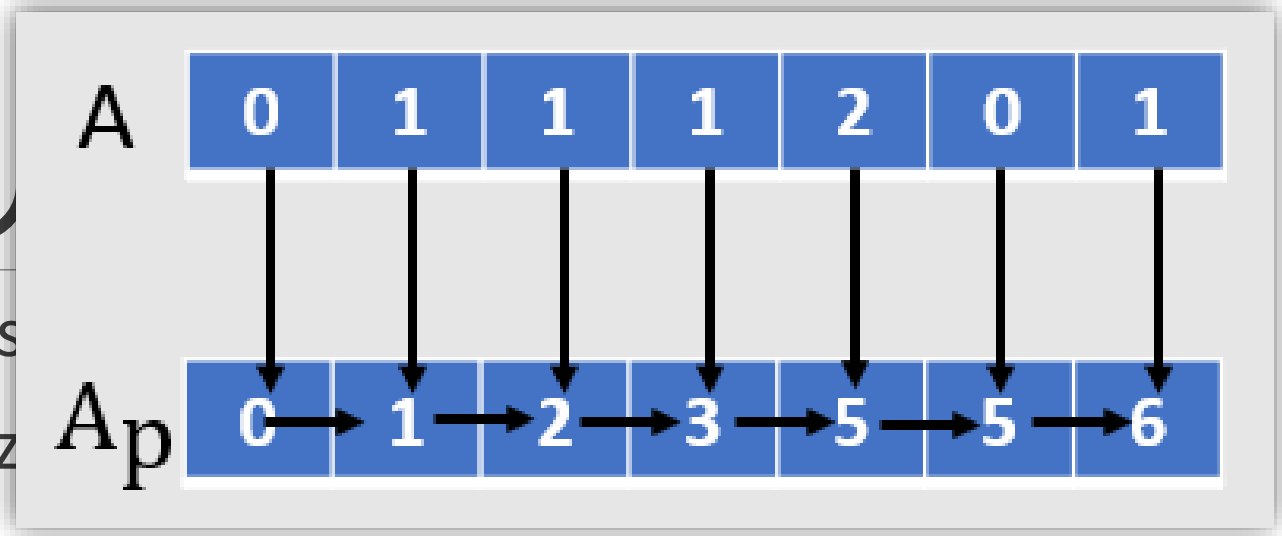
④ Bに接頭辞和を適用し、y（n）を生成する

※ヒストグラムを接頭辞和を2回ずつ行うことでソートを行う

H-Pソートのアルゴリ

入力配列x（size：n）、出力配列y（s

補助配列A（size ：maxVal）、A_p（siz



① x（n）のヒストグラムであるA（maxVal）を生成する

② Aに接頭辞和を適用し、A_p（maxVal）を生成する

③ A_pのヒストグラムであるB（n+1）を生成する

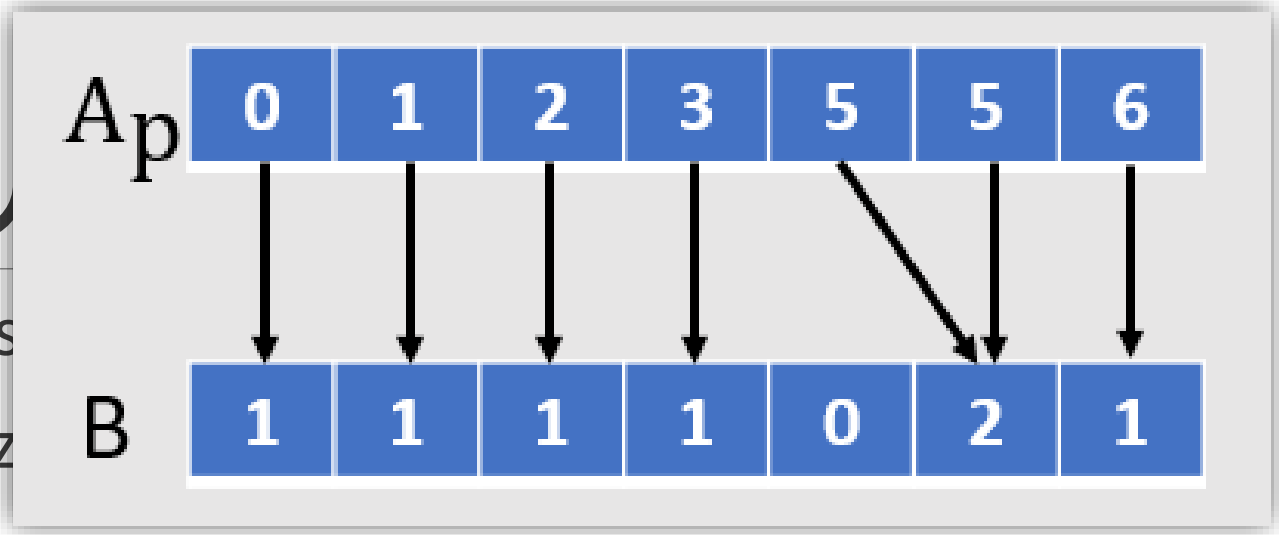
④ Bに接頭辞和を適用し、y（n）を生成する

※ヒストグラムを接頭辞和を2回ずつ行うことでソートを行う

H-Pソートのアルゴリ

入力配列x（size：n）、出力配列y（s

補助配列A（size ：maxVal）、A_p（siz



① x（n）のヒストグラムであるA（maxVal）を生成する

② Aに接頭辞和を適用し、A_p（maxVal）を生成する

③ A_pのヒストグラムであるB（n+1）を生成する

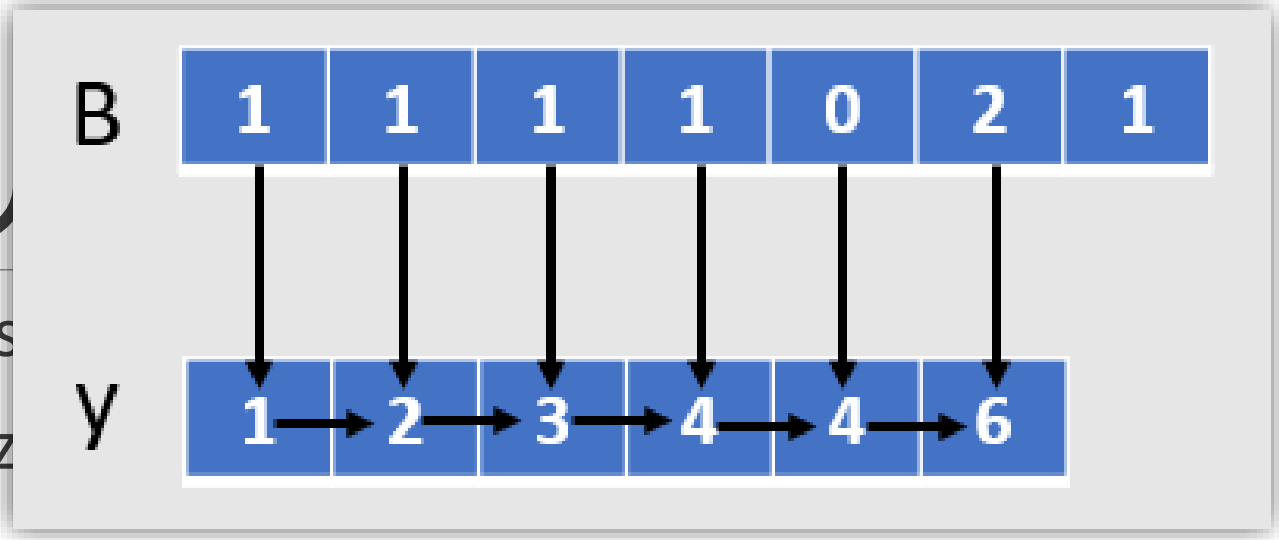
④ Bに接頭辞和を適用し、y（n）を生成する

※ヒストグラムを接頭辞和を2回ずつ行うことでソートを行う

H-Pソートのアルゴリ

入力配列x（size：n）、出力配列y（s

補助配列A（size ：maxVal）、A_p（siz



① x（n）のヒストグラムであるA（maxVal）を生成する

② Aに接頭辞和を適用し、A_p（maxVal）を生成する

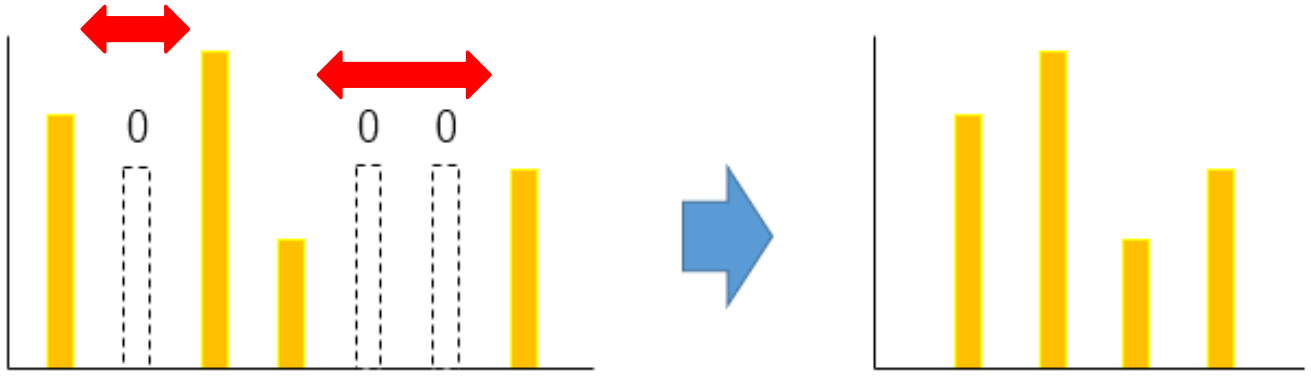
③ A_pのヒストグラムであるB（n+1）を生成する

④ Bに接頭辞和を適用し、y（n）を生成する

※ヒストグラムと接頭辞和を2回ずつ行うことでソートを行う

0-圧縮H-Pソート

- H-Pソートのヒストグラムの生成時における「個数：0」の部分を圧縮することで、ヒストグラムに対する接頭辞和の負荷の軽減を目指したアルゴリズムである



0-圧縮H-Pソートのアルゴリズム

アルゴリズムのパラメータ

n：入力データのデータ数、　len：入力データの種類数

maxVal：入力データ内の最大値

使用する配列

入力配列x（size：n）、出力配列y（size：n）

補助配列A（size ：maxVal）、B（size：n）、C（size：len+1）

0-圧縮H-Pソートのアルゴリズム

入力配列x（size：n）、出力配列y（size：n）

補助配列A（size ：maxVal）、B（size：n）、C（size：len+1）

① x（n）のヒストグラムであるA（maxVal）を生成する
② Aのヒストグラムの0部分を圧縮したC（len+1）を生成する
③ Cを用いてB（n）を生成する
④ Bに接頭辞和を適用し、y（n）を生成する

H-Pソートと0-圧縮H-Pソートの違い

H-Pソート

① x(n)のヒストグラムであるA(maxVal)を生成する

② Aに接頭辞和を適用し、A_p(maxVal)を生成する

③ A_pのヒストグラムであるB(n+1)を生成する

④ Bに接頭辞和を適用し、y(n)を生成する

0-圧縮H-Pソート

① x(n)のヒストグラムであるA(maxVal)を生成する

② Aのヒストグラムの0部分を圧縮したC(len+1)を生成する

③ Cを用いてB(n)を生成する

④ Bに接頭辞和を適用し、y(n)を生成する

H-Pソートと0-圧縮H-Pソートの差異

H-Pソート

① x(n)のヒストグラムであるA(maxVal)を生成する ⬅ $O(n)$

② Aに接頭辞和を適用し、A_p(maxVal)を生成する

③ A_pのヒストグラムであるB(n+1)を生成する

④ Bに接頭辞和を適用し、y(n)を生成する

0-圧縮H-Pソート

① x(n)のヒストグラムであるA(maxVal)を生成する ⬅ $O(n)$

② Aのヒストグラムの0部分を圧縮したC(len+1)を生成する

③ Cを用いてB(n)を生成する

④ Bに接頭辞和を適用し、y(n)を生成する

H-Pソートと0-圧縮H-Pソートの差異

H-Pソート

① x(n)のヒストグラムであるA(maxVal)を生成する ⬅ $O(n)$

② Aに接頭辞和を適用し、A_p(maxVal)を生成する ⬅ $O(maxVal)$

③ A_pのヒストグラムであるB(n+1)を生成する

④ Bに接頭辞和を適用し、y(n)を生成する

0-圧縮H-Pソート

① x(n)のヒストグラムであるA(maxVal)を生成する ⬅ $O(n)$

② Aのヒストグラムの0部分を圧縮したC(len+1)を生成する ⬅ $O(maxVal)$

③ Cを用いてB(n)を生成する

④ Bに接頭辞和を適用し、y(n)を生成する

H-Pソートと0-圧縮H-Pソートの差異

H-Pソート

① x(n)のヒストグラムであるA(maxVal)を生成する ⬅ $O(n)$

② Aに接頭辞和を適用し、A_p(maxVal)を生成する ⬅ $O(maxVal)$

③ A_pのヒストグラムであるB(n+1)を生成する ⬅ $O(maxVal)$

④ Bに接頭辞和を適用し、y(n)を生成する

0-圧縮H-Pソート

① x(n)のヒストグラムであるA(maxVal)を生成する ⬅ $O(n)$

② Aのヒストグラムの0部分を圧縮したC(len+1)を生成する ⬅ $O(maxVal)$

③ Cを用いてB(n)を生成する ⬅ $O(len)$

④ Bに接頭辞和を適用し、y(n)を生成する

H-Pソートと0-圧縮H-Pソートの差異

H-Pソート

① x(n)のヒストグラムであるA(maxVal)を生成する ← $O(n)$

② Aに接頭辞和を適用し、A_p(maxVal)を生成する ← $O(maxVal)$

③ A_pのヒストグラムであるB(n+1)を生成する ← $O(maxVal)$

④ Bに接頭辞和を適用し、y(n)を生成する ← $O(n)$

0-圧縮H-Pソート

① x(n)のヒストグラムであるA(maxVal)を生成する ← $O(n)$

② Aのヒストグラムの0部分を圧縮したC(len+1)を生成する ← $O(maxVal)$

③ Cを用いてB(n)を生成する ← $O(len)$

④ Bに接頭辞和を適用し、y(n)を生成する ← $O(n)$

H-Pソートと0-圧縮H-Pソートの差異

アルゴリズム	B生成の計算量	特性
H-Pソート	$O(maxVal)$	入力データの最大値に依存
0-圧縮H-Pソート	$O(len)$	入力データの種類数に依存

H-Pソート

入力データの最大値（maxVal）が小さいほど速くなる

0-圧縮H-Pソート

入力データの種類数（len）が少ないほど速くなる

ヒストグラムと接頭辞和のGPU

ヒストグラム

- GPUのライブラリの「AtomicAdd」という関数を用いて実現した

接頭辞和

- GPUの「thrust」ライブラリの「inclusive_scan」と「device_pointer_cast」いう関数を組み合わせることで実現した

実験環境

CPU	Intel Xeon CPU E5-2620 v3
GPU	NVIDIA　Tesla K40c

- NVIDIA統合開発環境「CUDA」　ver.10.0.130

実験内容

アルゴリズム	概要
thrustソート	CUDAのThrustライブラリのソート関数
H-Pソート	ヒストグラムと接頭辞和によってソートを行うアルゴリズム
0-圧縮H-Pソート	ヒストグラムの0部分を圧縮したアルゴリズム

- それぞれのアルゴリズムに対して、データ数(n)、最大値(maxVal)、データ数と最大値の割合($\delta = n/maxVal$)、種類数(len)をパラメータとして計測を行った

実験内容

- データ数と最大値の割合（$\delta = n/maxVal$）について

$\delta = 10$

データ数（n）	最大値（maxVal）
1万	1000
10万	1万
100万	10万

$\delta = 100$

データ数（n）	最大値（maxVal）
1万	100
10万	1000
100万	1万

実験内容

maxVal（最大値） = len（種類数）

δを大きく、つまり最大値を小さくしていく際の変化をみる

① δ = 1, maxVal = n

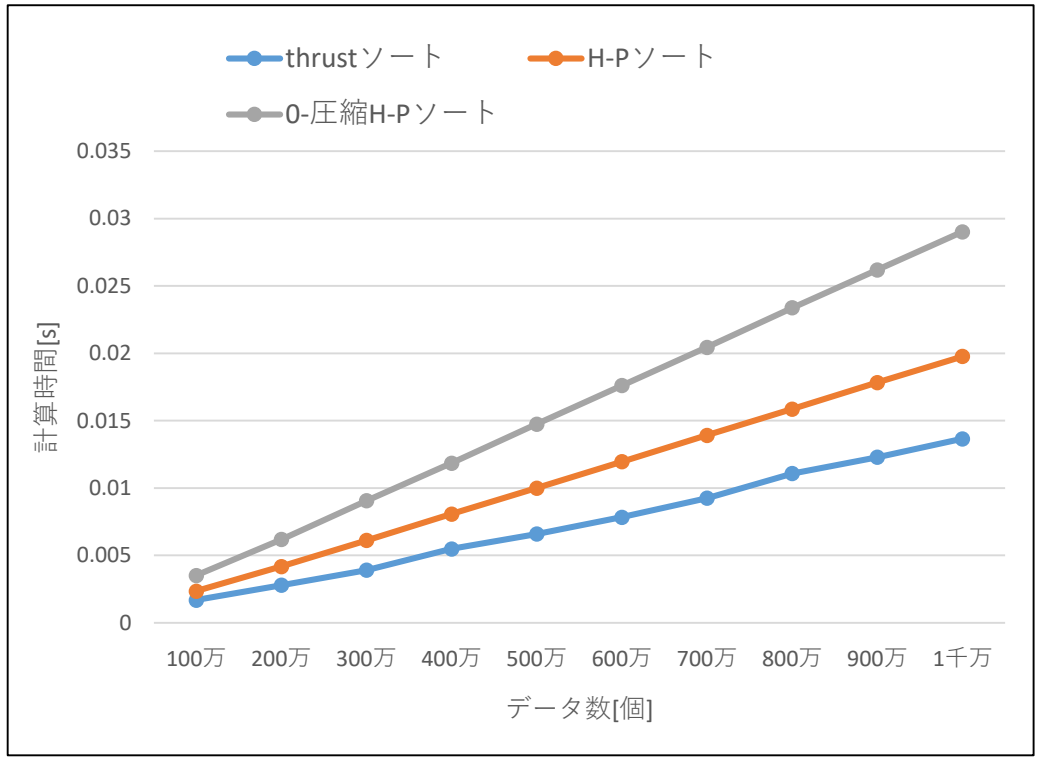
② δ = 10, maxVal = n/10

③ δ = 50, maxVal = n/50

④ δ = 100, maxVal = n/100

実験結果①（δ = 1, maxVal = n）

データ数 = 100万～1千万（100万刻み）



- データ数と最大値が等しい
  つまり、最大値が大きい

データ数 = 1千万の時、

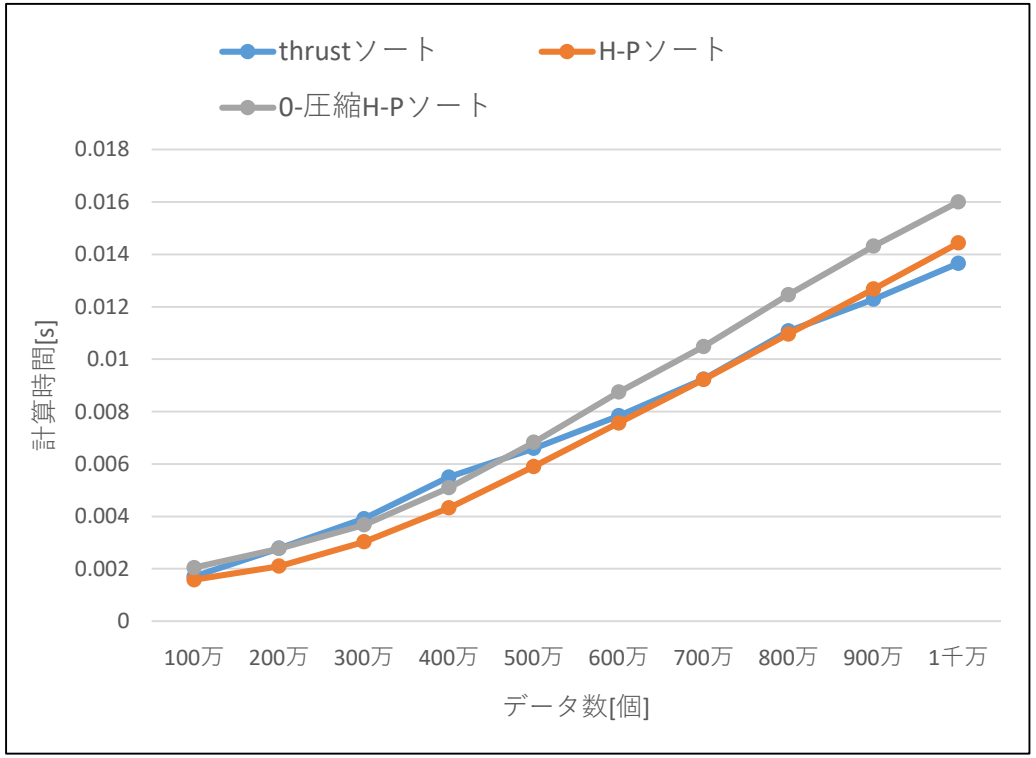
thrustソートに対して

H–Pソート … 30.9 %　遅い

0–圧縮ソート … 52.9 %　遅い

実験結果②（δ = 10, maxVal = n/10）

データ数 = 100万～1千万（100万刻み）



- 最大値がデータ数の1/10
  δ が10より大きくなるほど速くなる

データ数 = 1千万の時、

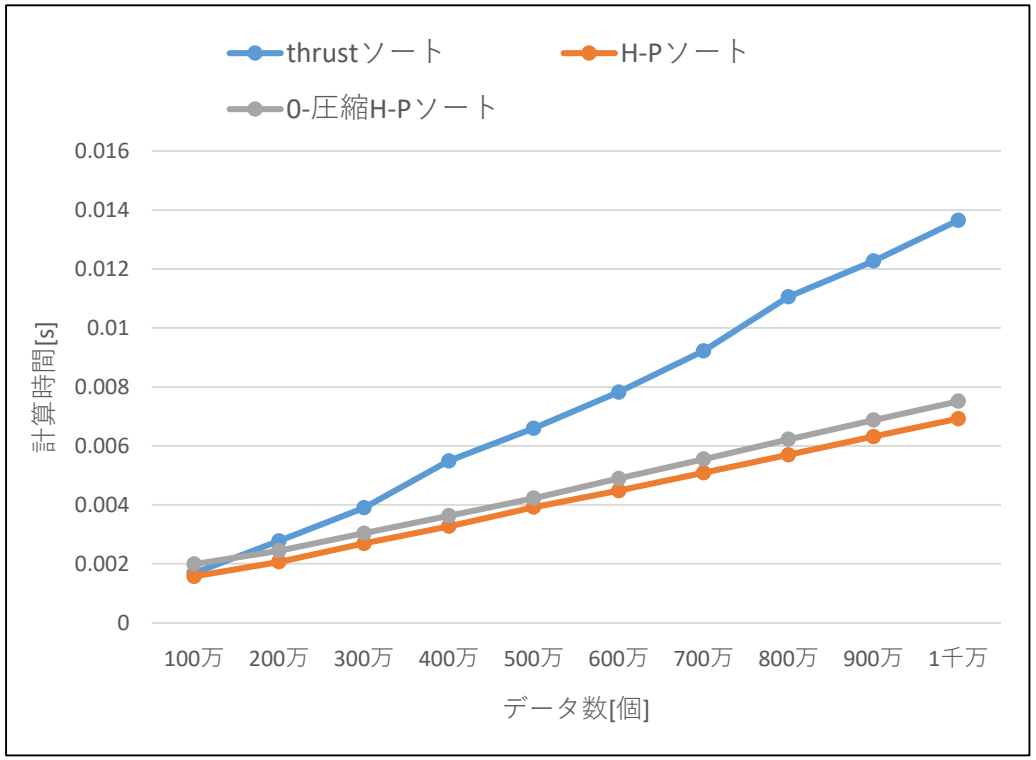
thrustソートに対して

H–Pソート … 5.4 % 遅い

0–圧縮ソート … 14.7 % 遅い

実験結果③（δ = 50, maxVal = n/50）

データ数 = 100万～1千万（100万刻み）



- 最大値が小さいため、
  H–Pソートが速くなっている

データ数 = 1千万の時、

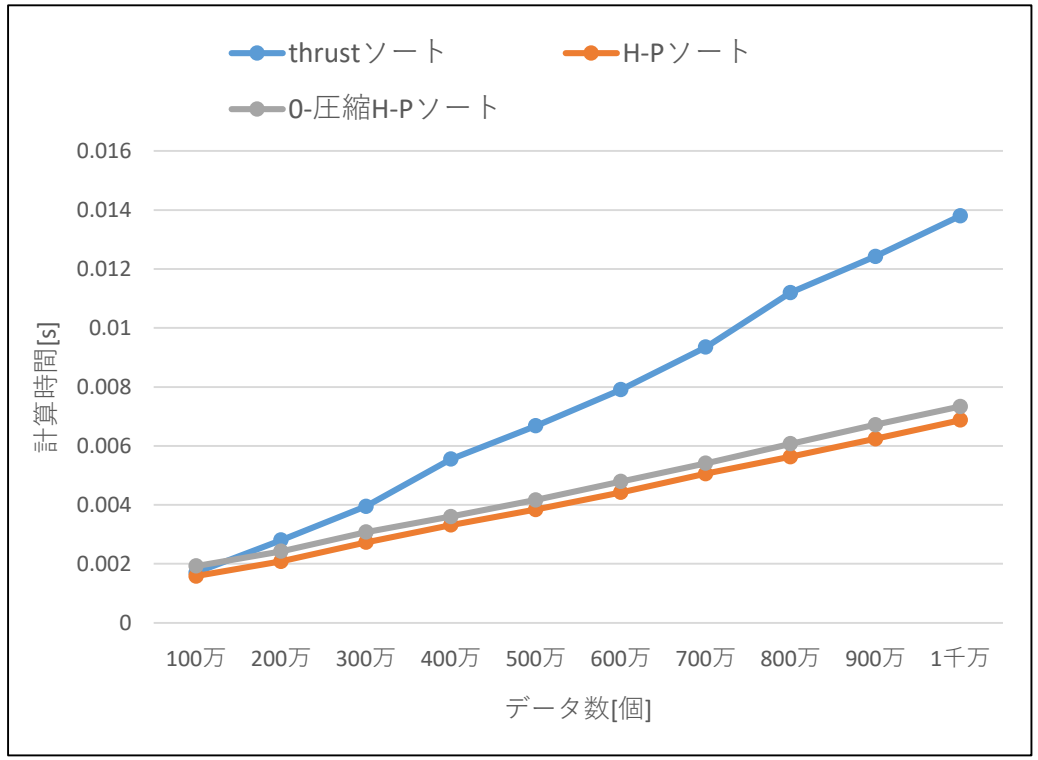
thrustソートに対して

H–Pソート … 49.3 %　速い

0–圧縮ソート … 45.0 %　速い

実験結果④（δ = 100, maxVal = n/100）

データ数 = 100万～1千万（100万刻み）



- δ が大きくなるほどH-Pソートと0-圧縮H-Pソートは速くなる

データ数 = 1千万の時、

thrustソートに対して

H-Pソート … 50.2 %　速い

0-圧縮ソート … 46.9 %　速い

実験内容

maxVal（最大値）>>　len（種類数）

最大値が大きい場合に、種類数を小さくしていく際の変化をみる

⑤　δ = 1, maxVal = 1len

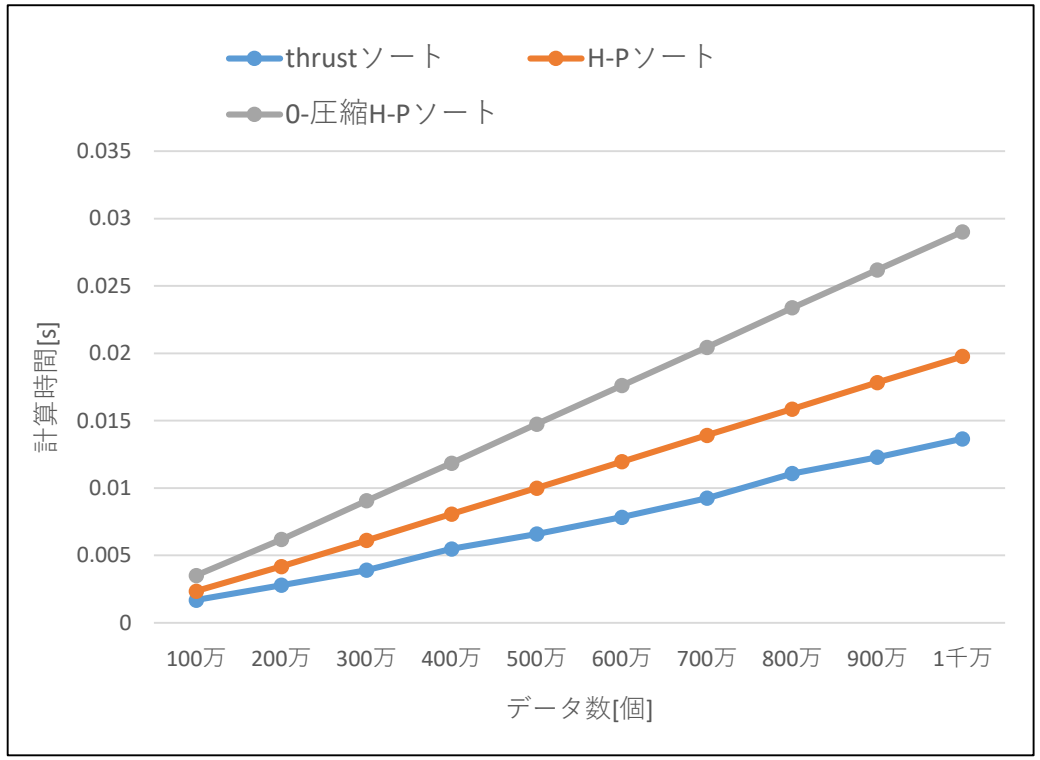
⑥　δ = 1, maxVal = 10len

⑦　δ = 1, maxVal = 100len

⑧　δ = 1, maxVal = 1000len

実験結果⑤（δ = 1, maxVal = 1len）

データ数 = 100万～1千万（100万刻み）



- 実験結果① = 実験結果⑤

データ数 = 1千万の時、

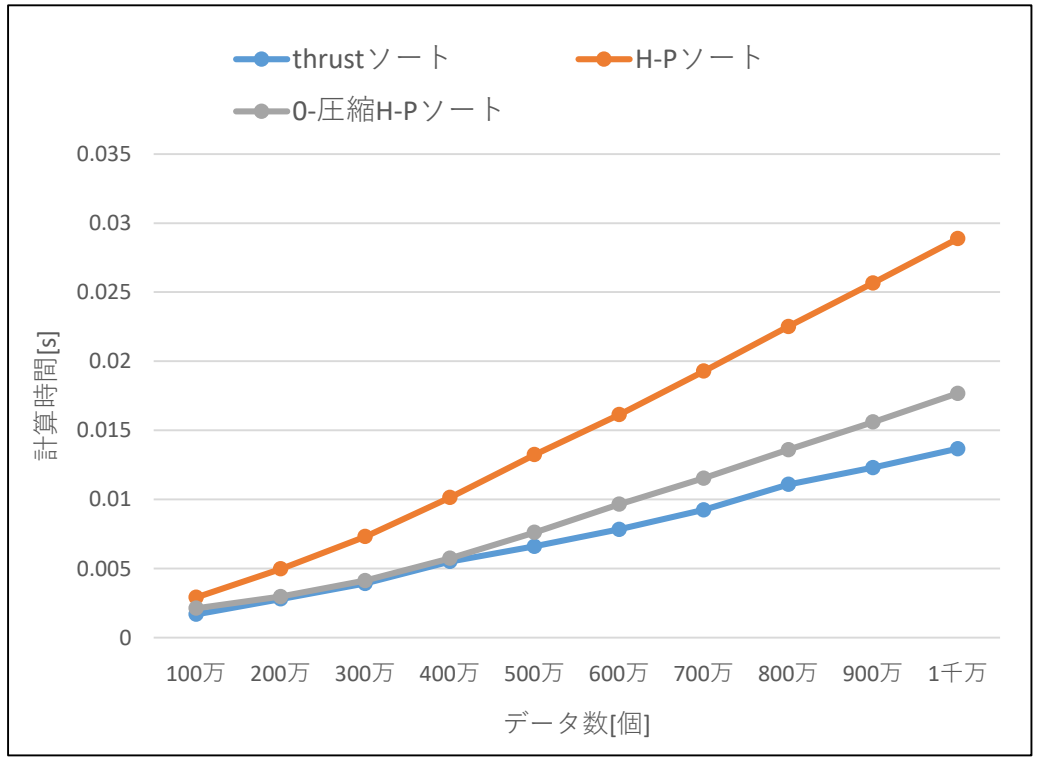
thrustソートに対して

H–Pソート … 30.9 %　遅い

0–圧縮ソート … 52.9 %　遅い

実験結果⑥（δ = 1, maxVal = 10len）

データ数 = 100万～1千万（100万刻み）



- 種類数がデータ数の1/10

データ数 = 1千万の時、

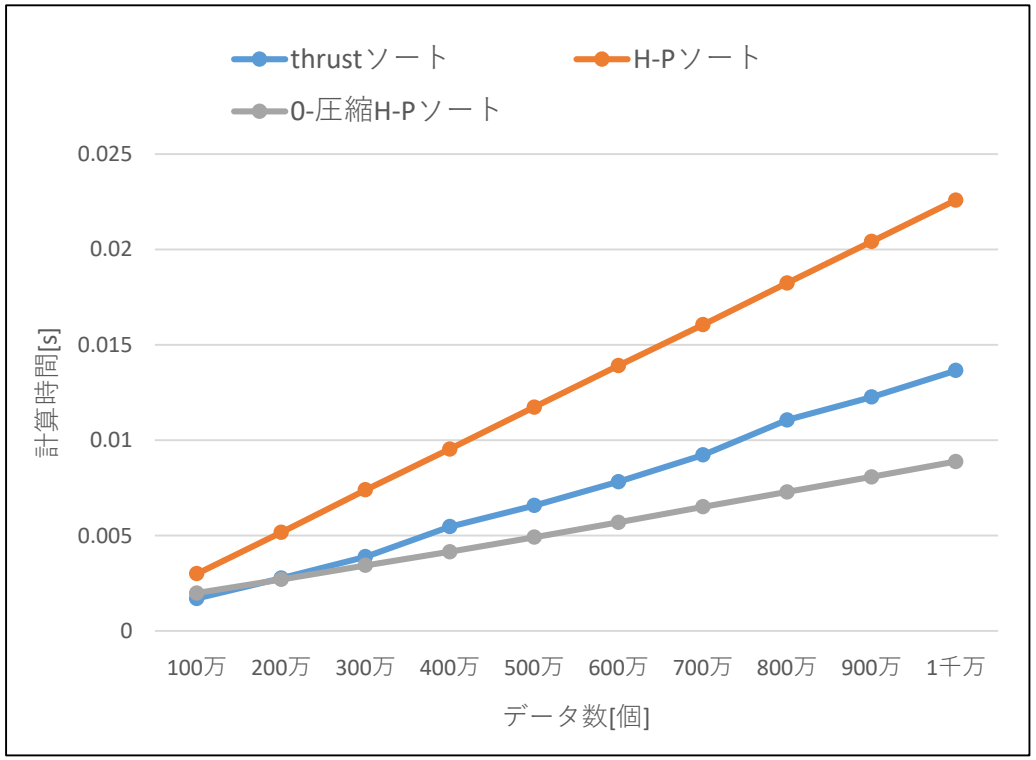
thrustソートに対して

H-Pソート … 52.7 %　遅い

0-圧縮ソート … 22.6 %　遅い

実験結果⑦（δ = 1, maxVal = 100len）

データ数 = 100万～1千万（100万刻み）



- 種類数が少なくなるほど、
  0-圧縮H-Pソートは速くなる

データ数 = 1千万の時、

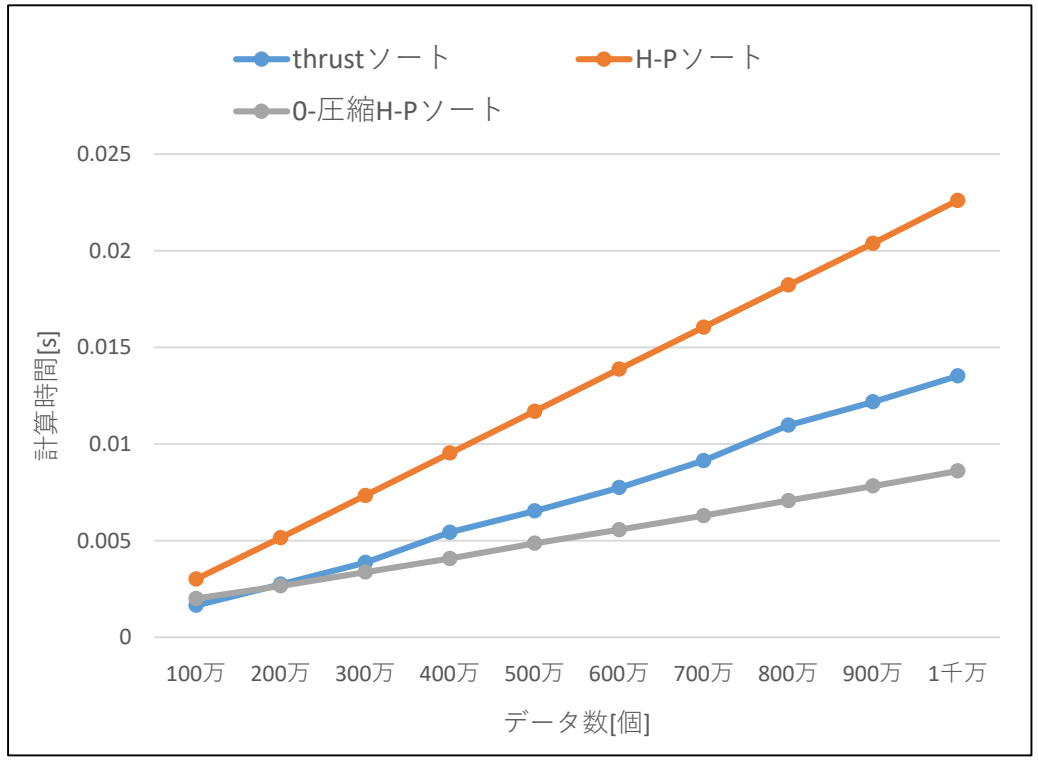
thrustソートに対して

H-Pソート … 36.9 %　遅い

0-圧縮ソート … 35.0 %　速い

実験結果⑧（δ = 1, maxVal = 1000len）

データ数 = 100万～1千万（100万刻み）



- 種類数が少なくなるほど、
  0-圧縮H-Pソートは速くなる

データ数 = 1千万の時、

thrustソートに対して

H-Pソート … 40.1 %　遅い

0-圧縮ソート … 36.4 %　速い

実験内容

n（データ数）>> maxVal（最大値）>>　len（種類数）

　種類数が小さい場合に、δを大きくする際の変化をみる

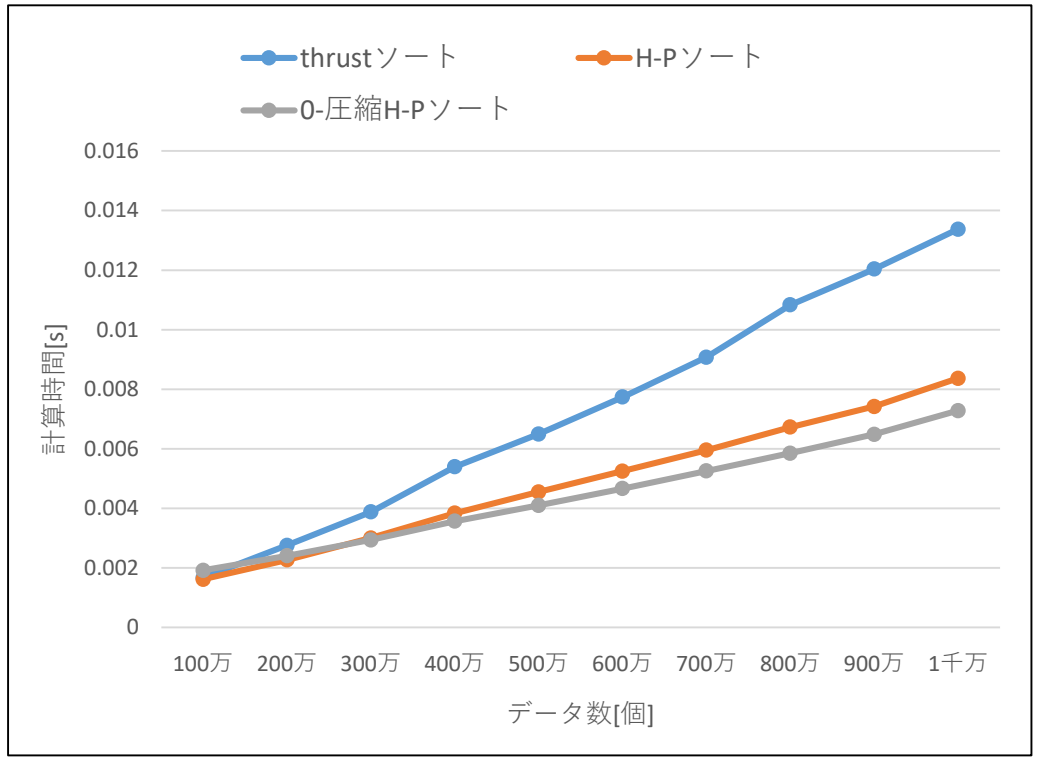
⑨　δ = 10, maxVal = 1000len

⑩　δ = 50, maxVal = 1000len

⑪　δ = 100, maxVal = 1000len

実験結果⑨（δ = 10, maxVal = 1000len）

データ数 = 100万～1千万（100万刻み）



データ数 = 1千万の時、

thrustソートに対して

H–Pソート … 37.4 %　速い

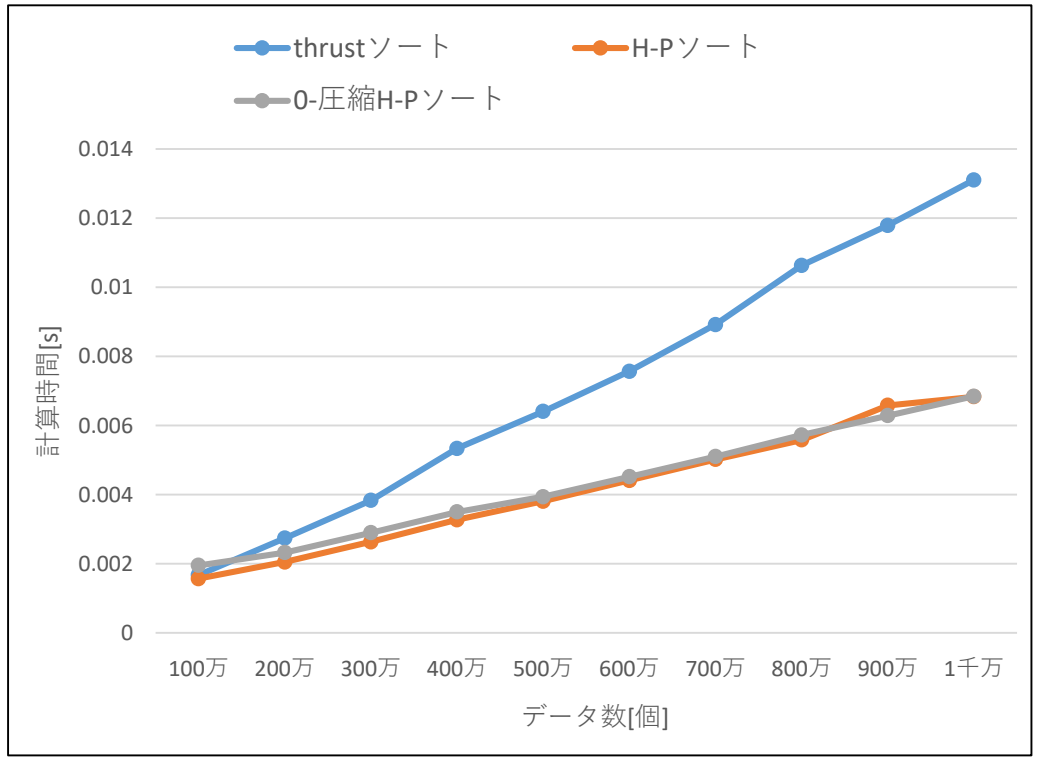
0–圧縮ソート … 45.6 %　速い

0–圧縮H–Pソートに対して

H–Pソート … 13.0 %　遅い

実験結果⑩（δ = 50, maxVal = 1000len）

データ数 = 100万～1千万（100万刻み）



データ数 = 1千万の時、

thrustソートに対して

H–Pソート … 47.9 %　速い

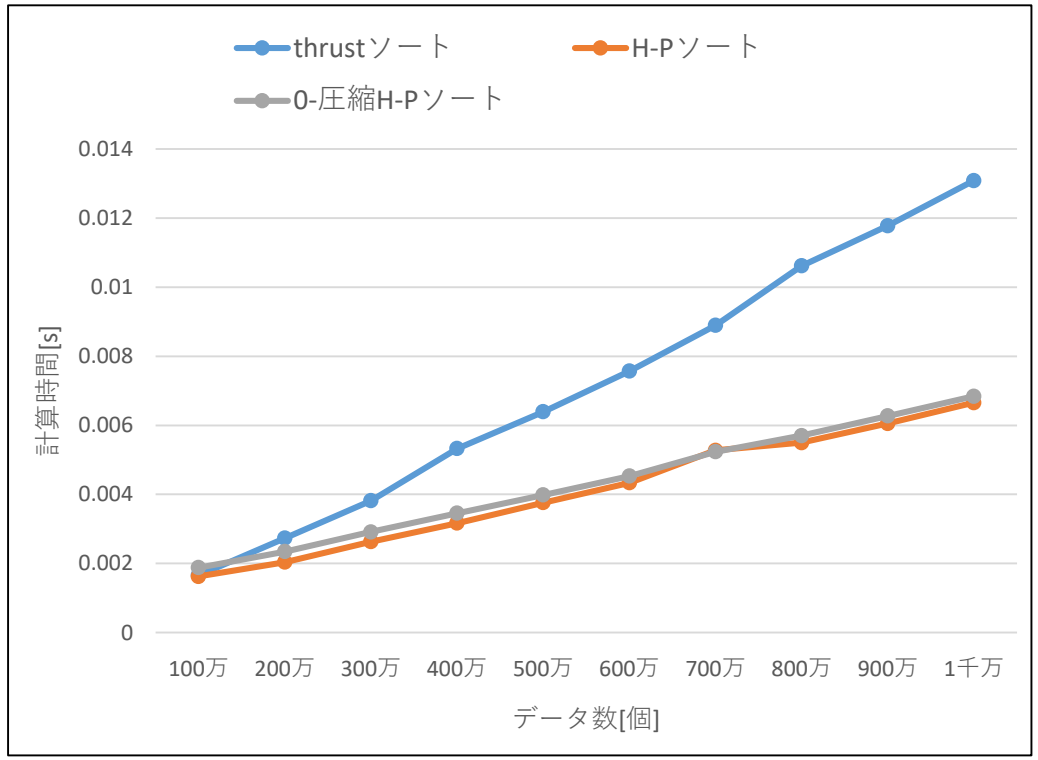
0–圧縮ソート … 47.8 %　速い

0–圧縮H–Pソートに対して

H–Pソート …　0.2 %　速い

実験結果⑪（δ = 100, maxVal = 1000len）

データ数 = 100万～1千万（100万刻み）



データ数 = 1千万の時、

thrustソートに対して

H-Pソート … 49.1 %　速い

0-圧縮ソート … 47.7 %　速い

0-圧縮H-Pソートに対して

H-Pソート …　2.7 %　速い

考察

- H-Pソートは0-圧縮H-Pソートに比べて最大値による影響が大きいため、最大値が小さくなるほど速くなる

- 0-圧縮H-Pソートは他の2つのアルゴリズムとは異なり、最大値が大きくとも、種類数が小さければ速くなる

考察

- 2つのアルゴリズムの計算量を考えれば明らかである

（n = データ数，maxVal = 最大値，len = 種類数）

H-Pソート

① x(n)のヒストグラムであるA(maxVal)を生成する	$O(n)$
② Aに接頭辞和を適用し、A_p(maxVal)を生成する	$O(maxVal)$
③ A_pのヒストグラムであるB(n+1)を生成する	$O(maxVal)$
④ Bに接頭辞和を適用し、y(n)を生成する	$O(n+1)$

0-圧縮H-Pソート

① x(n)のヒストグラムであるA(maxVal)を生成する	$O(n)$
② Aのヒストグラムの0部分を圧縮したC(len+1)を生成する	$O(maxVal)$
③ Cを用いてB(n)を生成する	$O(len+1)$
④ Bに接頭辞和を適用し、y(n)を生成する	$O(n)$

最大値の変化による影響が大きい

H-Pソート程最大値の影響は受けず、種類数が小さいほど速い

まとめ

- データ数に対して最大値が小さい場合、H-Pソートはthrustソートよりも速い整数ソーティングアルゴリズムである

  ⇒　thrustソートと比較して、最高で 50.2 % の高速化に成功した

- 最大値がデータ数よりも小さいという条件下において、最大値が大きい場合、0-圧縮H-Pソートはthrustソートよりも速い整数ソーティングアルゴリズムである

  ⇒　thrustソートと比較して、最高で 47.8 % の高速化に成功した

ご清聴ありがとうございました

発表者： 法政大学　小堺海叶