GPUによる高速な結合重み分布生成の検討

安藤 翔平[1]　伊野 文彦[2]　藤原 融[2]　萩原 兼一[2]
[1] 大阪大学基礎工学部情報科学科，[2] 大阪大学大学院情報科学研究科

1　はじめに

効率のよいマルチキャスト通信手法の1つとして，ネットワーク符号化がある．この手法では，誤りに対して耐性を持たせるために誤り訂正符号を用いて情報源を符号化する．誤り訂正符号の性能を評価するためには，復号誤り率を求めることが重要である．

ネットワーク符号化では，受信語間の誤りに相関がある．受信語の復号誤り率を得るためには符号語間の結合重み分布を計算する必要がある [1]．符号語の数は符号の次数 k に関して指数的に増大するため，次数の大きな線形符号について現実的な時間で結合重み分布を求めることは難しい．

近年では，グラフィクス処理用のプロセッサである GPU（Graphics Processing Unit）を用いて，汎用計算を高速化する研究が盛んである．GPU は画像処理のように大量のデータに対して似た計算を適用することが得意であり，結合重みの計算においても高速化が期待できる．そこで本研究では，GPU 向けの開発環境である CUDA（Compute Unified Device Architecture）を用い，結合重み分布生成の高速化について検討する．

2　結合重み分布

(n,k)–2 元線形符号では，符号長 n の線形独立な k 個の基底から，2^k 個の符号語を得られる．ある 2 つの符号語 $\overline{u}$ および $\overline{v}$ の順序対 $(\overline{u},\overline{v})$ に対し，結合重み $w(\overline{u},\overline{v})$ を以下のように定義する．

$$w(\overline{u},\overline{v}) \triangleq (x,y,z)$$

ここで x,y,z とは，$\overline{u}=(u_1u_2\cdots u_n)$ としたとき，同じ位置 $l\,(1\le l\le n)$ におけるビット (u_l,v_l) の値がそれぞれ $(1,1),(1,0),(0,1)$ であるものの数である．例えば，$\overline{u}=(11110000)$, $\overline{v}=(11001110)$ のとき，$x=2$, $y=2$, $z=3$ となる．

符号語の対には，2^{2k} 通りの選び方がある．一方，結合重みの取りうる値は $0\le x,y,z\le n$, $0\le x+y+z\le n$ を満たす整数 x,y,z の数に等しく，それらは ${}_nH_3$ 通りである．

各符号語の対について結合重みを並列に計算するとき，結合ヒストグラムのビンに対するインクリメントはアトミック演算により排他実行する必要がある．さらに，このアトミック演算の回数は，符号語の対の数 2^{2k} に応じて増大し，この演算が実行時間の大部分を占める．数千以上もの並列度を持つ GPU では，競合の多発により実行効率が大きく低下しうる．そのため，アトミック演算の競合を抑えることが実行効率を高めるために重要である．

3　提案手法

提案手法では $n\le 128$ の符号を対象として，共有メモリ上で局所的に結合ヒストグラムを計算することで高速化を図る．現在の GPU では ${}_nH_3$ 個のビンを全て共有メモリ上に保持することは不可能である．しかし，符号語の一方を決めた場合に結合ヒストグラムの更新されうる領域が $O(n^2)$ に限定できるため，共有メモリ上での計算が可能となる．

符号語の対に対する結合重みは，計算機上ではビット列中の 1 の個数を数える命令（popcount）およびビット毎の論理積（∧）を用いて次のように表せる．

表 1: $(127,22)$–2 元線形符号に対する実行時間

ハードウェア	手法	実行時間（分）
i7 3770K	単一スレッド	465
	マルチスレッド	104
GTX 580	グローバルメモリ版	353
	提案手法	147
GTX 680	グローバルメモリ版	50
	提案手法	143

$$x=\mathrm{popcount}(\overline{u}\wedge\overline{v})$$
$$y=\mathrm{popcount}(\overline{u})-x$$
$$z=\mathrm{popcount}(\overline{v})-x$$

popcount の計算量は符号長 n に比例するため，符号語の 2^{2k} 通りの対に対する時間計算量は $O(2^{2k}n)$ である．

符号語 $\overline{u}$ を定めると，$\mathrm{popcount}(\overline{u})=x+y$ が固定される．この値を t とおくと，y,z は以下の条件を満たす．

$$0\le y\le t,\quad 0\le z\le n-t$$

つまり，符号語 $\overline{u}$ に対し，結合ヒストグラムが更新されうる領域の大きさは $(t+1)(n-t+1)$ である．この式は $t=\lfloor n/2\rfloor,\lceil n/2\rceil$ のとき最大となる．$n=127$ ならば要素数は $64\times 65=4160$ であり，1 要素を 2 バイトとすれば，この領域を共有メモリに格納できる．符号語の対の数は 2^{2k} であるが，共有メモリは最大 $2048(<2^{16})$ スレッドからなるスレッドブロック毎に独立のメモリであるため，ビンの大きさは 2 バイトで十分である．

4　評価実験

提案手法の有効性を検証するため，グローバルメモリ上で結合ヒストグラムを計算する実装と比較した．また，OpenMP を用いて CPU で並列計算した場合とも比較した．CPU での結合重みの計算には SSE4.2 の POPCNT 命令を利用した．

実験環境として，CPU 版の実装では Intel Core i7 3770K（4 コア，8 スレッド）を用いた．GPU 版の実装では NVIDIA Geforce GTX580 および Geforce GTX680 を用いた．入力としては，BCH 符号の生成多項式から得られる $n=127,k=22$ の生成行列を用いた．

実験結果を表 1 に示す．CPU の単一スレッド版に対しては，GPU を用いることで実行時間を削減できた．GTX580 では提案手法を適用したことで，共有メモリを用いない場合と比較して実行時間を約 42%に削減できた．しかし，CPU の並列版と比較すると実行時間は約 1.4 倍長い．GTX680 では，CPU の並列版と比較して実行時間を 50%以下に削減できたが，提案手法を適用したことで実行時間が約 2.8 倍に増大した．Kepler アーキテクチャではグローバルメモリに対するアトミック演算のスループットが 9 倍に向上したため，共有メモリを用いてアトミック演算を高速化する効果が小さく，追加の処理によるオーバヘッドが上回るためと考えられる．

今後の課題は，Kepler アーキテクチャにおいても実行時間を削減すること，および k のより大きな生成行列に対しても実用的な時間で結合重み分布を計算することである．

参考文献

[1] 貴戸祥郎，藤原融，ネットワーク符号化における線形ブロック符号の復号誤り確率の評価のための結合重み分布に対するマックウィリアムズの等式，第 34 回情報理論とその応用シンポジウム予稿集，3.4.1，Nov. 2011.