Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186-5140
Volume 7, Number 1, pages 53–58, January 2018

A CUDA C Program Generator for Verifying Logical Operators

Daisuke Takafuji, Koji Nakano and Yasuaki Ito
Graduate School of Engineering, Hiroshima University
1-4-1 Kagamiyama, Higashi-Hiroshima, 739-8527, JAPAN
Email: {daisuke,nakano,yasuaki}@cs.hiroshima-u.ac.jp

***Abstract*—We develop a tool, named L2CU, which automatically generates a CUDA C program for verifying logical operators. The L2CU has been used to generate CUDA C programs for solving SAT. Compared to a sequential implementation on a single CPU, the generated CUDA C programs for solving SAT algorithms run** 2296.7 **times faster.**

I. はじめに

アルゴリズム実行中の各ステップにおいてアクセスされるアドレスが入力に依存しないシーケンシャルアルゴリズムを，オブリビアスアルゴリズムという．複数の独立した入力に対し，順にまたは並列に実行することを，シーケンシャルアルゴリズムのバルク実行という．我々はこれまでに，複数の独立した入力に対し，オブリビアスなシーケンシャルアルゴリズムのバルク実行を行う CUDA C プログラムを自動生成するツールを開発してきた [1]．このツールを使うと，オブリビアスなシーケンシャルアルゴリズムの C 言語プログラムから，そのバルク実行を行う CUDA C プログラムに自動で変換できる．生成された CUDA C プログラムは，CUDA で動作可能な GPU で実行可能である．また，回路に対するバルク実行においても，GPU による高速化が示されている [2]．

本稿では，論理式で表された回路から CUDA C プログラムを自動生成するツール L2CU を開発した．このツールの性能評価のために，大量のインスタンスの実行により解を求めることができる SAT 解法の GPU 実装に応用した．その結果，1 ブロック当たりのスレッド数を 64 とした GPU 実装では，CPU 実装に比べて 31 変数で，節の数が 1280 個，4SAT の場合に 2296.7 倍の高速化を実現した．

GPU (Graphics Processing Unit) は，画像処理計算の高速化を目的に設計された専用回路である [3]．最近の GPU は汎用計算のために設計され従来の CPU 計算にも応用でき，多くのアプリケーション開発者に注目されている [3]．NVIDIA は，自社が提供する GPU を動作させるための並列計算機アーキテクチャを提供しており，これを *CUDA* (Compute Unified Device Architecture) [4] という．CUDA は，NVIDIA 社 GPU での仮想命令セットと並列計算のメモリを提供する．

GPU は複数のストリーミング・マルチプロセッサ（streaming multiprocessor）で構成されており，各ストリーミング・マルチプロセッサには CUDA コアと呼ばれる演算装置が含まれている（Fig. 1 参照）．GPU のメモリは高速で小容量なオンチップメモリ（shared memory）と低速で大容量なオフチップメモリ（global memory）で構成されている（Fig. 1 参照）．一方，CUDA の並列計算モデルは，グリッド（grid），スレッド・ブロック（thread block），スレッド（thread）と呼ばれる階層で構成されている．1 つのグリッドは複数のスレッド・ブロックで構成されており，各スレッド・ブロックも複数のスレッドで構成されている．1 スレッドは 1 つの CUDA コアで実行され，1 スレッド・ブロックは 1 つのストリーミング・マルチプロセッサで実行される．

CUDA による GPU 実装において，高速化のために重要な点を記す．スレッドは並列に動作するとき 32 スレッドごとに動作するため，32 スレッドを 1 ワープ（warp）と呼ぶ．1 ワープ内の 32 スレッドは並列に動作しており，if 文などの分岐がある場合にはその then 節と else 節の両方を実行するため，動作遅延の要因になり，可能な限りこのような記述は避けるべきである．スレッドによって 2 次元配列の行にある連続した場所がアクセスされるとき，golobal memory のアドレス空間にある対応する場所は同時にアクセスされる．このようなアクセス方法を coalescing access とよぶ（Fig. 2 (a) 参照）．しかし，もし列にある連続した場所がアクセスされるならば遠く離れた場所が同時にアクセスされることになる．このようなアクセス方法を stride access とよぶ（Fig. 2 (b) 参照）．

以下のように，p 個の配列（要素数 n）を保存する 2 つの方法を考える（Fig. 3 参照）．

- **Row-wise arrangement**: $b_j[i]$ $(0 \leq i \leq n-1, 0 \leq j \leq p-1)$ を j 行 i 列に保存するように，p 行 n 列の 2 次元配列に蓄える．
- **Column-wise arrangement**: $b_j[i]$ $(0 \leq i \leq n-1, 0 \leq j \leq p-1)$ を i 行 j 列に保存するように，n 行 p 列の 2 次元配列に蓄える．

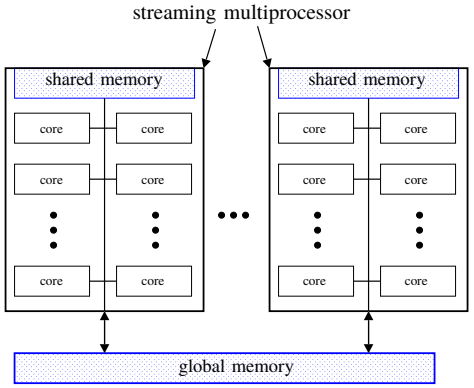

Fig. 1. CUDA hardware architecture.

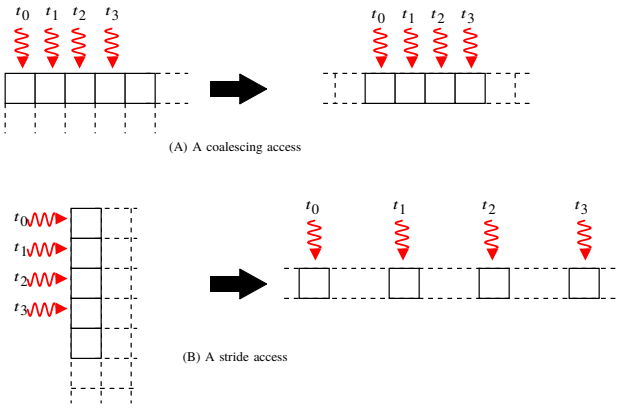

Fig. 2. An example of a coalescing access and a stride access.

0	1	2	3	4	5	6	7
$b_0[0]$	$b_1[0]$	$b_2[0]$	$b_3[0]$	$b_4[0]$	$b_5[0]$	$b_6[0]$	$b_7[0]$
8	9	10	11	12	13	14	15
$b_0[1]$	$b_1[1]$	$b_2[1]$	$b_3[1]$	$b_4[1]$	$b_5[1]$	$b_6[1]$	$b_7[1]$
16	17	18	19	20	21	22	23
$b_0[2]$	$b_1[2]$	$b_2[2]$	$b_3[2]$	$b_4[2]$	$b_5[2]$	$b_6[2]$	$b_7[2]$
24	25	26	27	28	29	30	31
$b_0[3]$	$b_1[3]$	$b_2[3]$	$b_3[3]$	$b_4[3]$	$b_5[3]$	$b_6[3]$	$b_7[3]$
32	33	34	35	36	37	38	39
$b_0[4]$	$b_1[4]$	$b_2[4]$	$b_3[4]$	$b_4[4]$	$b_5[4]$	$b_6[4]$	$b_7[4]$
40	41	42	43	44	45	46	47
$b_0[5]$	$b_1[5]$	$b_2[5]$	$b_3[5]$	$b_4[5]$	$b_5[5]$	$b_6[5]$	$b_7[5]$

Column-wise

0	1	2	3	4	5
$b_0[0]$	$b_0[1]$	$b_0[2]$	$b_0[3]$	$b_0[4]$	$b_0[5]$
6	7	8	9	10	11
$b_1[0]$	$b_1[1]$	$b_1[2]$	$b_1[3]$	$b_1[4]$	$b_1[5]$
12	13	14	15	16	17
$b_2[0]$	$b_2[1]$	$b_2[2]$	$b_2[3]$	$b_2[4]$	$b_2[5]$
18	19	20	21	22	23
$b_3[0]$	$b_3[1]$	$b_3[2]$	$b_3[3]$	$b_3[4]$	$b_3[5]$
24	25	26	27	28	29
$b_4[0]$	$b_4[1]$	$b_4[2]$	$b_4[3]$	$b_4[4]$	$b_4[5]$
30	31	32	33	34	35
$b_5[0]$	$b_5[1]$	$b_5[2]$	$b_5[3]$	$b_5[4]$	$b_5[5]$
36	37	38	39	40	41
$b_6[0]$	$b_6[1]$	$b_6[2]$	$b_6[3]$	$b_6[4]$	$b_6[5]$
42	43	44	45	46	47
$b_7[0]$	$b_7[1]$	$b_7[2]$	$b_7[3]$	$b_7[4]$	$b_7[5]$

Row-wise

Fig. 3. $p = 8$ 個の 1 次元配列における Column-wise arrangement と Row-wise arrangement. ただし，それぞれの配列の要素数 n は $n = 6$ である．

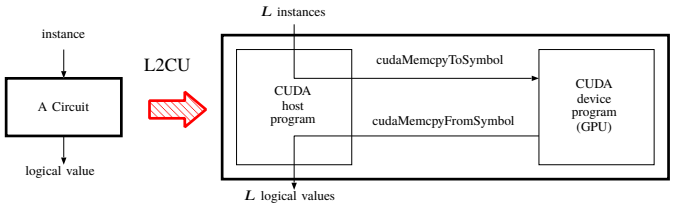

Fig. 4. A CUDA C Program Generator L2CU

II. CUDA C Generator

与えられた回路を検証するための CUDA C プログラムを自動生成する L2CU を提案する．L2CU は，Fig. 4 に示すように，L 個のインスタンスに対し与えられた回路の論理演算を効率的に計算する CUDA C プログラムを生成するものである．与えられた回路を検証するために，

大量のインスタンスに対し動作検証をするため，その手間が非常に大きく，効率的な検証ツールが必要である．大量のインスタンスに対し GPU を用いて効率的に論理演算を行う手法として，BPBC が知られている [2].

A. *GPU* 実装

この L2CU で生成される CUDA C プログラムは，coalescing access による効率的なメモリアクセスとビットパックによる計算効率化を実現している．

生成される CUDA C プログラムについて説明する（Fig. 5 参照）．N 入力 M 出力の回路において，L 個のインスタンスがあると仮定する．

このプログラムではまず，各入力 u_i $(0 \leq i < N)$ のインスタンスを 1 ビットで表すことにより，32 個分のインスタンスを 1 word（32 ビット）に保存する．各 u_i の入力値を 1 行に蓄えることにより，Fig. 5 のようにカラムワイズに保存する．同様に，各出力 X_j $(0 \leq j < M)$ の論理値を 1 ビットで表すことにより，32 個分の論理値も 1 word（32 ビット）に保存する．各 X_j の L 個分の論理値を 1 行に保存し，Fig. 5 のようにカラムワイズに保存する．1 CUDA thead は 1 word 分（32 個）のインスタンスの論理演算を同時に計算する．インスタンスおよび論理地はすべて global memory 上に保存されるが，Fig. 5 のように，入力値 u_i の読み込み，および出力値 X_j への書き込みは coalescing access であり，効率的なメモリアクセスとなっている．

B. 回路の書式

ここでは，回路の記述方法を述べる．1 つめのカラムが項目名を表し，2 つめのカラムが値を表すが，カラム区切りをタブとする．入力は項目 “`input`”，出力は項目 “`output`” の値であり，値それぞれは “,（カンマ）” で区切られている．論理式は項目 “`assign`” の値で与えられる．論理式には，“`&` (AND)”, “`|` (OR)”, “`^` (XOR)”, “`~` (NOT)” を使用できる．各項目の終端は “`;`（セミコロン）” で表わし，複数行にまたがっても構わない．（つまり，“`input`”, “`output`”, “`assign`” を複数行に分けてもよい．）`#`で始まる行はコメント行として扱われ，無視される．

8 入力 4 出力の回路を記述した例を，Fig. 6 に示す．L2CU によって生成された CUDA C プログラムを Fig. 7 に示す．

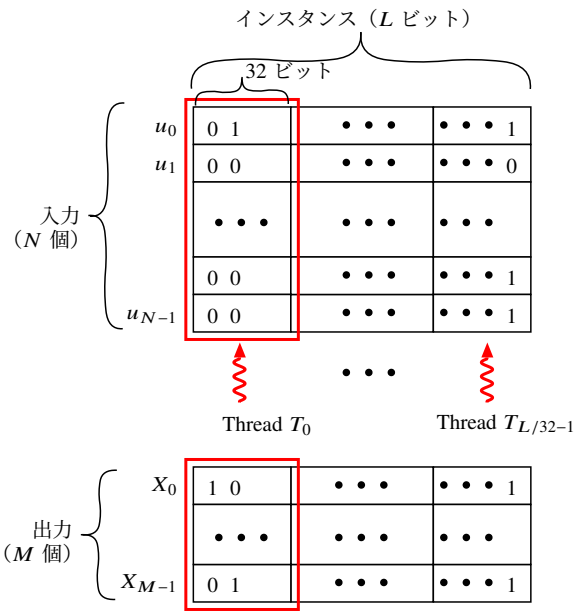

Fig. 5. GPU implementation.

回路を記述した Fig. 6 では，まず第 2, 3 行において入力値が `A0`, `A1`, `A2`, `A3`, `A4`, `A5`, `A6`, `A7`, `A8` の 8 入力であることを指定しており，第 4 行において出力値が `S0`, `S1`, `S2`, `S3` の 4 出力であることを指定している．第 5–8 行において，`S0`, `S1`, `S2`, `S3` それぞれの論理式を与えている．

Fig. 7 では第 3 行で指定された 1 CUDA thread が第 4–7 行において入力値を読み込み，Fig. 6 で与えられた論理式を計算し保存する．このとき，各 CUDA thread は隣り合うメモリアドレスを読み込み，隣り合うアドレスに書き込んでいるため，coalescing access を実現していることがわかる．

また 1 word（32 ビット）に 32 個のインスタンスを保存しているため，1 回の演算で 32 回分の演算を行うことができ，演算回数を減少させることができ，効率的な計算につながっている．

III. SAT 解法への応用

A. *GPU* 実装

L2CU の性能評価のため，SAT 解法に応用する．

充足可能性問題（SAT）とは，論理変数の値に真または偽を割り当てるとき，与えられた論理式が真になるような論理変数の真偽値が存在するかどうか判定する問題である．論理変数またはその否定をリテラルといい，リテラルの論理和を節という．どの節においても，含まれるリテラルの数が k 以下のとき kSAT という．

SAT は NP 完全問題として有名であり，様々な近似解法や最適解法が研究されている．そのため，論理変数の値のすべての組合せに対し論理式を計算し，その結果の中に真を返すものがあれば SAT の解は “YES” であり，そうでなければ “NO” である．

そこで本稿では，論理変数の値のすべての組合せを計算することにより SAT を解くことにする．L2CU の使用により，与えられた 1 つの論理式から CUDA C プログラムを生成し，すべての組合せをインスタンスとして与える．つまり，論理変数が N 個のときインスタンス数は 2^N 個であり，2^N 個のインスタンスに対する論理式を計算する．

16 変数の 4SAT の回路を Fig. 8 に示し，L2CU によって生成された CUDA C プログラムを Fig. 9 に示す．SAT においては論理式はちょうど 1 つであるため，1 出力であり，CUDA C プログラムにおいても論理式計算は 1 行のみである．このときインスタンス数は 2^{16} である．Fig. 8 では第 1 行において 16 個の論理変数を定義している．第 3 行において 4SAT の論理式を定義している．

B. 性能評価

L2CU によって生成された CUDA C プログラムを NVIDIA TITAN X に実装し，CPU 実装と計算時間を比較した．NVIDIA TITAN X は 28 個のストリーミング・マルチプロセッサをもち，それぞれ 128 個のコアをもつ．

入力データは，変数の数 N は $N = 28, 29, 30, 31$，節の数 C は $C = 256, 512, 768, 1024, 1280$，$k$SAT ($k = 3, 4, 8, 12, .., 28$) のインスタンスをランダムに生成したものである．GPU 実装では，1 ワード 32 ビットとし，1 ブロック当たりスレッド数を 32, 64 とした．起動ブロック数は，1 ブロック当たり 32 スレッドならば $N = 28, 29, 30, 31$ のときそれぞれ 256k, 512k, 1024k, 2048k 個である．1 ブロック当たり 64 スレッドならば $N = 28, 29, 30, 31$ のときそれぞれ 128k, 256k, 512k, 1024k 個である．CPU 実装の計算時間を計測するために，Intel Xeon 2.1 GHz を使用した．計算時間の比較結果を Table I, II に示す．

GPU 実装では 1 ブロック当たりのスレッド数が 64 の場合の計算時間は，32 の場合に比べてほとんどの場合で短くなっている．また，この計算時間は CPU 実装と比べると，31 変数で，節の数が 1280 個，4SAT の場合に最大 2296.7 倍の高速化を実現した．

高速化の理由としては次の通りである．本稿の GPU 実装では論理演算とデータの読み書きのみであるが，論理演算の計算時間が非常に短く，計算時間は global memory への読み込み，書き込みが大部分を占める．global memory への読み込み，書き込みを効率化するために，Column-wise arrangement で保存し，coalescing access を実現した効果が大きいと考えられる．

IV. まとめ

本稿では，論理式で表された回路から CUDA C プログラムを自動生成するツール L2CU を開発した．このツールの性能評価のために，大量のインスタンスの実行により SAT 解法の GPU 実装に応用した．その結果，1 ブロック当たりのスレッド数を 64 とした GPU 実装では，CPU 実装に比べて 31 変数で，節の数が 1280 個，4SAT の場合に最大 2296.7 倍の高速化を実現した．

References

[1] D. Takafuji, K. Nakano, Y. Ito, and J. Bordim, “C2CU: a CUDA C program generator for bulk execution of a sequential algorithm,” *Concurrency and Computation: Practice and Experience*, vol. 29, no. 17, pp. e4022–n/a, 2017.

[2] T. Fujita, K. Nakano, and Y. Ito, “Bitwise parallel bulk computation on the GPU, with application to the CKY parsing for context-free grammars,” in *Proc. of International Parallel and Distributed Processing Symposium Workshops*, May 2016, pp. 589–598.

[3] W. W. Hwu, *GPU Computing Gems Emerald Edition*. Morgan Kaufmann, 2011.

```
## input: 8 values, output: 4 values
input   A0,A1,A2,A3,A4,A5;
input   A6,A7;
output  S0,S1,S2,S3;
assign  S0=(A3 & A4) | (A4 & A6) | (A0 & A5) | (A3 & A5) | (A7 & A1) ;
assign  S1=(A2 & A4) | (A7 & A3) | (A2 & A1) | (A4 & A1);
assign  S2=(A7 & A4) | (A1 & A6) | (A2 & A0) | (A6 & A0) | (A6 & A1) | (A4 ^ A2);
assign  S3=A7 & A3 & A2 ;
```

Fig. 6. Logical Operators.

```
__global__ void compute_bit(unsigned int *d_in, unsigned int *d_out)
{
    unsigned int tid = blockIdx.x * blockDim.x + threadIdx.x;
    d_out[0*W2+tid] = (d_in[3*W2+tid] & d_in[4*W2+tid] ) |(d_in[4*W2+tid] & d_in[6*W2+tid] ) |(
      d_in[0*W2+tid] & d_in[5*W2+tid] ) |(d_in[3*W2+tid] & d_in[5*W2+tid] ) |(d_in[7*W2+tid] &
      d_in[1*W2+tid] ) ;
    d_out[1*W2+tid] = (d_in[2*W2+tid] & d_in[4*W2+tid] ) |(d_in[7*W2+tid] & d_in[3*W2+tid] ) |(
      d_in[2*W2+tid] & d_in[1*W2+tid] ) |(d_in[4*W2+tid] & d_in[1*W2+tid] ) ;
    d_out[2*W2+tid] = (d_in[7*W2+tid] & d_in[4*W2+tid] ) |(d_in[1*W2+tid] & d_in[6*W2+tid] ) |(
      d_in[2*W2+tid] & d_in[0*W2+tid] ) |(d_in[6*W2+tid] & d_in[0*W2+tid] ) |(d_in[6*W2+tid] &
      d_in[1*W2+tid] ) |(d_in[4*W2+tid] ^ d_in[2*W2+tid] ) ;
    d_out[3*W2+tid] = d_in[7*W2+tid] & d_in[3*W2+tid] & d_in[2*W2+tid] ;
}
```

Fig. 7. A CUDA C program generated by our L2CU.

```
input   X0,X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15;
output  F0;
assign  F0 = (X1 | X0 | X13 | X7 ) & (~ X4 | X1 | ~ X2 | X6 ) & (~ X9 | ~ X11 | X13 | X6 ) & (~
  X12 | X0 | X10 | X14 ) & (X8 | X14 | X13 | X1 ) & (~ X3 | ~ X12 | X0 | X5 ) & (X15 | ~ X11 |
  ~ X2 | X10 ) & (~ X4 | X6 | X8 | X0 );
```

Fig. 8. 16 変数の 4SAT ($k = 4$)

```
 __global__ void sat_solver(unsigned int *d_in, unsigned int *d_out)
{
    unsigned int tid = blockIdx.x * blockDim.x + threadIdx.x;
    unsigned int d_out[0*W2+tid] = (d_in[1*W2+tid] | d_in[0*W2+tid] | d_in[13*W2+tid] | d_in[7*
      W2+tid] ) &(~d_in[4*W2+tid] | d_in[1*W2+tid] | ~d_in[2*W2+tid] | d_in[6*W2+tid] ) &(~d_in
      [9*W2+tid] | ~d_in[11*W2+tid] | d_in[13*W2+tid] | d_in[6*W2+tid] ) &(~d_in[12*W2+tid] |
      d_in[0*W2+tid] | d_in[10*W2+tid] | d_in[14*W2+tid] ) &(d_in[8*W2+tid] | d_in[14*W2+tid] |
       d_in[13*W2+tid] | d_in[1*W2+tid] ) &(~d_in[3*W2+tid] | ~d_in[12*W2+tid] | d_in[0*W2+tid]
       | d_in[5*W2+tid] ) &(d_in[15*W2+tid] | ~d_in[11*W2+tid] | ~d_in[2*W2+tid] | d_in[10*W2+
      tid] ) &(~d_in[4*W2+tid] | d_in[6*W2+tid] | d_in[8*W2+tid] | d_in[0*W2+tid] ) ;
}
```

Fig. 9. Fig. 8 の CUDA C program

[4] NVIDIA Corporation, “NVIDIA CUDA C programming guide version 5.0,” 2012.

TABLE I
N = 28, 29 の場合の計算時間 (ms) の比較

節の数	k	28 変数					29 変数				
		CPU	GPU				CPU	GPU			
			32 threads		64 threads			32 threads		64 threads	
		計算時間	計算時間	高速化率	計算時間	高速化率	計算時間	計算時間	高速化率	計算時間	高速化率
256	3	1725.00	2.81	612.8	2.83	610.6	3432.35	5.76	596.2	5.74	598.0
	4	2083.94	2.83	736.6	2.83	736.6	4216.45	5.75	733.0	5.76	732.2
	8	3087.99	2.84	1087.1	2.86	1078.9	6211.50	5.79	1072.0	5.79	1072.9
	12	4121.90	3.03	1359.0	2.84	1452.6	8425.75	5.79	1454.7	5.80	1453.3
	16	5141.24	4.11	1250.1	4.04	1273.4	10278.46	6.78	1516.2	6.63	1550.2
	20	6216.56	5.14	1210.5	4.58	1357.5	12525.02	8.38	1494.8	8.69	1441.9
	24	7377.38	5.79	1273.2	5.63	1309.3	15141.16	10.07	1503.4	9.90	1529.2
	28	8636.36	6.75	1278.6	7.21	1198.0	17773.61	11.94	1488.4	11.68	1522.4
512	3	2798.87	2.83	988.9	2.85	983.2	5768.75	5.80	994.8	5.77	999.5
	4	3514.35	2.84	1235.7	2.84	1236.2	6974.67	5.80	1202.5	5.82	1199.3
	8	5732.69	4.18	1372.6	3.70	1551.2	11476.26	6.94	1652.5	6.41	1789.4
	12	7785.15	5.59	1393.7	5.48	1420.6	15462.50	10.31	1499.8	9.49	1629.4
	16	9771.40	8.00	1222.2	7.49	1305.0	19540.32	13.78	1418.0	13.34	1464.4
	20	12324.73	9.78	1260.6	9.44	1305.0	25176.31	17.48	1440.4	16.76	1502.1
	24	14777.33	10.59	1395.9	11.40	1296.7	29503.47	23.30	1266.0	20.44	1443.4
	28	17069.53	13.45	1269.6	12.97	1315.6	35737.03	23.61	1513.9	23.87	1497.2
768	3	3930.07	2.84	1385.0	2.85	1379.0	8001.29	5.78	1384.4	5.83	1372.3
	4	5101.07	3.17	1611.6	2.88	1770.2	10255.57	5.78	1774.6	5.79	1770.2
	8	8568.19	5.71	1500.9	5.55	1544.3	17611.80	10.26	1717.2	10.05	1753.2
	12	11207.00	9.34	1200.4	8.42	1331.8	23213.11	15.77	1472.1	14.96	1551.9
	16	14608.81	11.17	1308.2	12.45	1173.7	29321.44	21.02	1395.2	18.73	1565.2
	20	17775.41	14.89	1193.4	14.20	1251.4	36349.17	26.56	1368.8	25.71	1413.8
	24	21525.78	18.88	1140.0	17.12	1257.4	43023.83	29.75	1446.1	31.16	1380.6
	28	35243.83	20.60	1711.1	19.80	1779.7	53291.51	35.64	1495.4	34.81	1530.8
1024	3	5105.72	3.25	1569.8	2.92	1745.8	10139.30	5.80	1749.3	5.83	1740.2
	4	6556.46	3.83	1713.3	3.75	1749.1	12897.49	6.90	1868.4	6.75	1911.3
	8	11382.18	8.50	1338.5	7.32	1555.4	22717.99	13.94	1629.3	13.34	1702.6
	12	14663.95	11.70	1253.7	11.18	1311.4	29485.83	19.58	1505.7	20.28	1453.8
	16	18877.22	15.74	1199.2	15.63	1208.0	38506.65	28.07	1371.7	25.45	1513.2
	20	24961.88	20.90	1194.3	19.65	1270.6	47444.27	32.61	1454.7	31.39	1511.7
	24	28031.33	23.84	1175.8	21.92	1278.8	56849.40	40.44	1405.8	40.97	1387.4
	28	32934.03	27.64	1191.4	26.48	1243.9	66986.38	48.36	1385.2	46.91	1428.1
1280	3	6062.38	4.00	1516.5	3.93	1542.8	12134.11	6.44	1884.9	6.38	1900.8
	4	8528.72	5.28	1615.6	4.72	1806.0	17112.19	8.52	2007.8	7.87	2175.6
	8	13737.59	9.65	1423.5	10.57	1300.0	27072.08	17.37	1558.2	16.86	1605.4
	12	18123.15	14.72	1231.5	14.05	1290.1	36866.76	24.41	1510.4	25.30	1457.2
	16	23280.31	19.92	1169.0	19.08	1220.0	46936.34	34.71	1352.2	32.34	1451.3
	20	28972.54	23.61	1227.0	22.75	1273.6	59264.09	41.61	1424.3	42.93	1380.6
	24	35270.39	29.72	1186.7	28.51	1236.9	97753.52	51.64	1893.1	51.28	1906.4
	28	42293.08	36.27	1166.1	35.52	1190.8	126895.93	58.36	2174.4	57.26	2216.3

TABLE II
N = 30, 31 の場合の計算時間 (ms) の比較

節の数	k	30 変数					31 変数				
		CPU	GPU				CPU	GPU			
			32 threads		64 threads			32 threads		64 threads	
		計算時間	計算時間	高速化率	計算時間	高速化率	計算時間	計算時間	高速化率	計算時間	高速化率
256	3	6902.41	11.96	577.2	11.94	578.0	14825.53	24.32	609.7	24.52	604.7
	4	8380.19	11.95	701.1	11.95	701.0	17030.37	24.62	691.6	24.60	692.3
	8	12753.07	11.95	1067.1	11.94	1068.2	25435.57	24.64	1032.2	24.64	1032.3
	12	16911.32	11.92	1418.2	11.94	1416.9	33850.76	24.58	1377.4	24.62	1374.8
	16	20900.03	13.18	1585.6	12.89	1621.9	42713.38	26.35	1621.3	26.26	1626.4
	20	25437.63	16.39	1551.7	15.96	1593.5	52081.82	31.57	1649.9	30.94	1683.2
	24	30719.80	19.50	1575.3	19.08	1609.7	60300.33	39.14	1540.5	37.91	1590.5
	28	35366.64	23.05	1534.5	22.21	1592.1	72679.34	45.85	1585.3	44.32	1639.9
512	3	11312.37	11.90	950.6	11.94	947.7	22506.24	24.38	923.1	24.49	919.2
	4	14275.73	11.95	1194.6	11.86	1204.2	28622.21	24.51	1167.6	24.60	1163.4
	8	22831.04	12.80	1783.8	12.53	1821.7	45830.92	26.05	1759.3	25.66	1785.9
	12	31651.56	19.23	1646.2	18.91	1673.5	64455.35	38.49	1674.4	37.74	1707.7
	16	40225.05	26.60	1512.3	25.55	1574.1	80432.88	52.24	1539.6	50.25	1600.5
	20	49418.77	34.13	1448.0	33.38	1480.6	98904.53	65.77	1503.8	65.51	1509.7
	24	62343.02	39.82	1565.5	39.93	1561.4	125508.62	79.85	1571.8	102.84	1220.4
	28	70001.94	46.74	1497.6	47.31	1479.6	141766.19	96.57	1468.0	114.78	1235.1
768	3	15258.77	11.90	1282.3	11.90	1281.7	33556.93	13.50	2485.5	13.41	2501.7
	4	21618.75	12.19	1774.1	11.97	1806.8	41947.81	18.69	2244.7	18.38	2282.0
	8	35794.88	19.84	1804.2	19.54	1832.0	70125.38	36.06	1944.8	35.59	1970.4
	12	45440.82	34.37	1322.1	29.29	1551.7	93013.62	53.43	1740.8	51.92	1791.6
	16	58798.64	40.09	1466.8	38.44	1529.4	118037.42	70.45	1675.5	67.49	1749.0
	20	71666.98	50.17	1428.6	47.49	1509.2	146022.40	89.61	1629.5	85.87	1700.4
	24	86892.56	62.08	1399.7	56.95	1525.9	183336.24	110.95	1652.4	109.49	1674.4
	28	102136.69	70.15	1456.1	66.76	1529.9	208815.54	125.43	1664.7	120.55	1732.1
1024	3	20757.70	13.67	1518.2	11.92	1741.2	40471.19	18.40	2199.2	18.21	2222.9
	4	27398.16	13.23	2071.5	12.82	2136.9	52709.35	24.48	2153.3	24.29	2169.8
	8	45739.06	27.68	1652.5	26.74	1710.6	92645.92	47.69	1942.8	46.53	1991.1
	12	60213.73	39.58	1521.3	38.37	1569.1	121165.96	71.41	1696.7	68.66	1764.7
	16	76643.45	53.16	1441.7	51.56	1486.4	154064.57	95.07	1620.5	91.04	1692.2
	20	95394.99	66.83	1427.5	64.06	1489.1	191926.79	123.76	1550.8	118.31	1622.3
	24	154065.23	81.44	1891.8	79.31	1942.5	230617.22	143.65	1605.4	137.76	1674.1
	28	135131.49	93.35	1447.6	94.17	1434.9	285816.97	167.83	1703.0	159.11	1796.4
1280	3	24867.33	14.61	1701.6	12.41	2003.3	50919.15	23.49	2167.3	23.28	2187.5
	4	34588.13	17.01	2033.9	16.73	2066.9	69714.92	30.61	2277.6	30.35	2296.7
	8	55462.65	34.80	1593.9	37.49	1479.2	110539.55	60.61	1823.7	59.79	1848.9
	12	75076.51	48.80	1538.3	58.08	1292.6	151398.35	89.73	1687.2	86.49	1750.5
	16	93774.93	64.96	1443.5	62.49	1500.7	191909.36	119.00	1612.7	116.64	1645.3
	20	117540.14	87.24	1347.3	84.34	1393.6	237603.54	148.97	1594.9	144.73	1641.8
	24	219272.53	103.96	2109.2	101.57	2158.7	283261.64	178.77	1584.5	172.82	1639.1
	28	207250.00	116.54	1778.3	110.22	1880.4	450548.78	213.96	2105.8	203.85	2210.2