Vol.2019-HPC-172 No.15
2019/12/19

C for Metal 言語を用いた Intel HD Graphics の GPGPU 用途における性能評価

近藤鯛貴[1]　竹田大将[1]　佐藤裕幸[1]

概要：Intel HD Graphics(IHD)とは Intel 社の開発する CPU 統合 GPU である．高いグラフィック性能を必要としない安価なコンピュータ向けに作られていることから低コストかつ低消費電力である．しかし，近年ではその性能もあがり理論性能で約 1TFlops の製品も登場し，計算資源として非常に有用である．また，2018 年に Intel が公式に IHD 用 GPGPU 言語である C for Metal(CM)を公開したことにより IHD の計算資源を GPGPU 用途で活用できるようになった．そこで本研究では CM の言語仕様を調査し，いくつかの GPGPU プログラムを実装した．Flops による理論性能と実測性能の評価を行ったところ，サンプルの行列積プログラムにて 81.60%の性能率を確認し，同 SoC 内の CPU と計算時間で比較したところ約 57 倍高速に動作し，IHD および CM の GPGPU 用途における有用性を示した．

キーワード：Intel HD Graphics，C for Metal 言語，GPGPU，SIMD

1. はじめに

近年，深層学習（DNN）の注目によりグラフィック演算装置である GPU を汎用計算に用いる GPGPU 技術の需要が高い．一般的に GPGPU 技術は NVIDIA 社や AMD 社が販売する専用の高性能 GPU ボードにおいて用いられることが多く，GPGPU プログラミング環境や GPU に最適化された数値計算アプリケーションなどが多く存在する．一方であまり性能を必要としない，低消費電力設計な SoC にパッケージされる GPU においては GPGPU の活用事例は少なく，開発環境や性能評価等の情報は少ない．また，近年の IoT ブームに伴い計算処理を遠隔サーバではなくエッジで行うエッジコンピューティングの注目が高く，特に DNN の推論処理をネットワークインフラに依存せず安定して高速に計算することへの需要が高い．このような場面において GPU ボードを搭載した大型コンピュータではなく，小型で低消費電力かつ高性能なコンピュータが望まれる．これらのことから小型で低消費電力な SoC 内 GPU による GPGPU 利用の需要は高いといえる．現状，小型 GPU ボードとして NVIDIA の Tegra シリーズがある．これは ARMCPU と NVIDIAGPU がパッケージされた SoC であり，NVIDIA の専用 GPU ボードと同じアーキテクチャであることから CUDA をはじめとする既存の GPGPU 環境が動作する．IoT 向けの計算ボードとして販売されているが，実社会において他汎用プロセッサと比べて普及しているとは言い難い．

一般的に普及している SoC 内 GPU として Intel HD Graphis が挙げられる．Intel HD Graphics とは Intel 社による CPU プロセッサ内蔵 GPU のシリーズ名であり，2010 年以降 Intel Core i シリーズをはじめとする IntelCPU 製品にて広く搭載されている．高いグラフィック性能を必要としない安価かつ低消費電力なコンピュータ向けに設計されており，性能は PCIe に外付けするようなボード型 GPU に比べて非常に低い．しかし，年々の性能向上に伴い近年では 1TFlops の性能を有する製品も登場しており，計算資源として非常に有用である．また，2018 年 5 月に Intel 社が公式に Intel HD Graphics 用 GPGPU 言語の C for Metal(CM)言語を公開したことにより，Intel プロセッサにて GPGPU 技術を活用することが可能になった．しかし，公開から日が浅いことから情報が少なく，GPGPU 用途において開発，性能効率面で有用であるかは未知数である．

そこで本研究では CM の言語仕様を調査し，いくつかのプログラムの実装，Flops による理論性能と実測性能による性能評価と計算時間による同 SoC 内 CPU との比較を行った．

本論文の構成は２章にて Intel HD Graphics アーキテクチャを説明し，３章では CM 言語仕様について説明する．4 章では評価プログラムの実装法を説明し５章で評価，最後に６章をまとめとする．

2. Intel HD Graphics アーキテクチャ

本章では Intel HD Graphics のアーキテクチャ構造について述べる．Intel では公式に Developer Documents[1]が公開されており，Gen ごとにアーキテクチャ構造を知ることができる．最新は Ice Lake アーキテクチャから搭載予定の Gen11 が公開されているが 2019 年現在一般的に普及し，かつ本研究でも用いられている Gen9 について解説する．Gen9 リファレンス[2]には Intel HD Graphics530 がモデルとなっている．

2.1 アーキテクチャ概要

IntelSoC の概要図を図 1，GPU の概要図を図 2 に示す．基本的な SoC 構成として CPU，GPU，EDRAM が搭載されており，Shared Last Level Cache(LLC)と EDRAM は CPU と

1 岩手県立大学ソフトウェア情報学部

GPU 共有で扱える. GPU は Slice 単位の構成となっており，1 つのスライスに 3 つの SubSlice と L3 Data Cache が搭載されている．L3 Data Cache には Shared local Memory と呼ばれる領域があり，SubSlice ごとに 64KB を割り当てられた共有メモリである．SubSlice には 8 個の Executin Unit(EU)と Sampler と呼ばれる L1，L2 Cache と Data port が搭載されている．EU は積和演算可能な実行ユニットであり次項にて詳細に説明する．Sampler は読み取り専用メモリフェッチユニットであり，L1，L2 Cache の間には DirectX や OpenGL の圧縮テクスチャを動的に解凍する専用ロジックがある．また，アドレスを画像 u，v 座標に変換する機能やバイリニア，トリリニア等のサンプリングフィルタリングモードをサポートしており，画像やテクスチャの扱いに適している. 他の L3 Data Cache の転送経路として Dataport も用意されている. これはメモリの load/store が可能なユニットで汎用に扱える．Sampler と Dataport はどちらも L3 Data Chache に対して 64byte/cycle でデータ通信を行える．

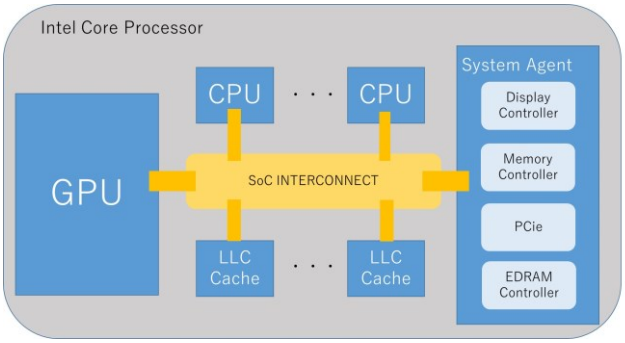

図 1:Intel プロセッサ概要図

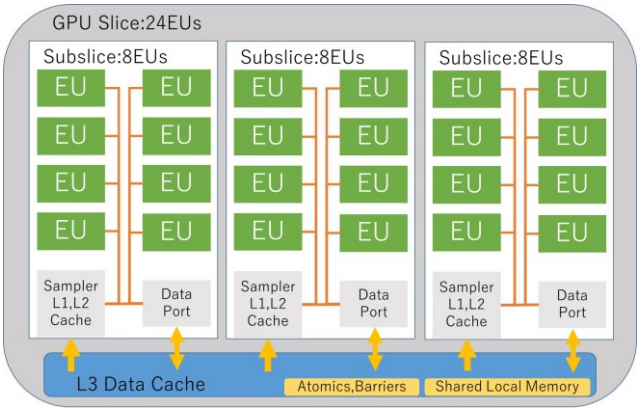

図 2:GPUSlice 概要図

2.2 Execution Unit(EU)

EU の構成図を図 3 に示す. EU には 32bit x 4way の SIMD FPU が 2 個搭載されており, 2 個で 1 個の SIMD 演算器として動作する．そのため，EU は 1 サイクルで 8 個の単精度浮動小数点演算が可能である. 半精度浮動小数点演算にも対応しており，16 個/サイクルで計算できる．また，2 個の SIMD FPU のうち 1 個が exp や log などの超越演算と 2way の倍精度浮動小数点演算に対応している. SIMD FPU という名前ではあるが整数計算にも対応している．

EU は 7 スレッドのハードウェア(HW)スレッドを生成可能であり, その内の 4 スレッドが 4 つの異なる命令を同時に発行できる．発行した命令は Thread Arbiter によって各ユニットにディスパッチされる仕組みとなっている. 各 HW スレッドには General Purpose Register File(GRF)と Architecture Register File(ARF)と呼ばれるレジスタが搭載されている．GRF は 1 個が 32bit x 8 個のベクトルレジスタであり，1 スレッドに 128 個搭載されている．そのため，EU では最大 28kbytes のデータを保持できる．ARF は各スレッドの実行状態等を保持する特殊レジスタである．

SIMD FPU は物理的に 4way-32bit ではあるが論理的に 1，2，4，8，16，32way の SIMD 命令に対応しており，この時レジスタも複数個を 1 個のレジスタとして扱うことができる．最後に Branch unit は SIMD での分岐を管理する専用ユニットで，Send unit はメモリ操作や sampler 操作などのシステム通信に対して send 命令にてディスパッチするユニットである．

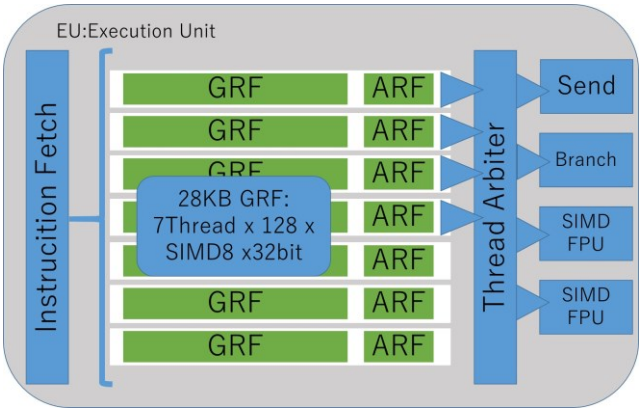

図 3:EU 概要図

3. C for Metal 言語

C for Metal(CM)言語とは Intel 社が公式に公開している C ベースの GPGPU プログラミング言語である．元は Intel 社内で用いられていた言語であるが，2018 年 5 月より一般に公開されるようになった．リファレンス[3]としてチュートリアルと API リファレンスが公開されているが，充実しているとは言い難く，情報量に乏しい．本論文ではリファレンスの内容に加えて実験的に判明した仕様等も含まれている．

CUDA や OpenCL との主な違いとして 1 スレッドがスカラではなく，ベクトルや行列の SIMD を計算単位として扱うことが挙げられる．基本的なプログラミング法は他 GPGPU 言語と類似していることから本章では CM 特有の機能を中心に解説する．

3.1 SIMD

CM では計算に使う SIMD 幅を GPU カーネル内の変数ごとに設定する．変数には int や float などの一般的な型でスカラ変数を定義することに加えて Matrix と Vector を用いた２次元，１次元配列を扱うことができる．この配列での計算は SIMD として扱われ，基本的な四則演算子や組み込み関数などにおいて１回の記述で全要素あるいは一部要素の計算を可能とする．

SIMD の操作として C 言語の配列アクセスのようにスカラを取得できることや Matrix において row や column 関数による行と列要素の取得, select 関数による sub-Matrix, sub-Vector の取得ができる．select 関数は select<v，v_st，h，h_st>(i，j)のように記述し，ここで v は縦要素数，v_st は縦要素の stride，h は横要素数 h_st は横要素の stride，i と j は開始位置の offset である. 動作例として図 4 に示す. Matrix, Vector に比較演算子を用いることで 1 と 0 からなる Mask 配列を生成することができる．Merge 関数でこの Mask を用いて２つの配列をマージできることから SIMD 内要素の疑似的な条件分岐が可能となる．

GPU カーネルにおいてこれらの変数は GRF にアロケーションされることから 1HW スレッドのレジスタ容量である 4kbytes 内に抑える必要がある．

```
Mat2=Mat.select<2,1,3,2>(0,0)
```

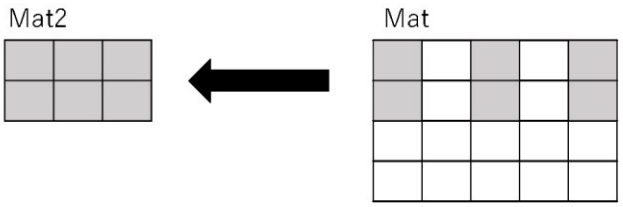

図 4:select 関数の動作例

3.2 スレッドについて

CM ではスレッドの定義に ThreadSpace と ThreadGroupSpace の２種類がある. ThreadSpace は境界のない単純な２次元のスレッド生成を行うモードであり，近傍スレッドの状態から実行順序を制御する Thread-dependency pattern の設定ができる．いくつかの制御パターンがあるが代表的なものとして WAVE_FRONT が挙げられる．これは最初に左上のスレッドが実行され，対象スレッド位置の一つ上と左が実行されたのちに動作を開始する制御モードで，積分画像を効率的に計算することを目的としている. 次に ThreadGroupSpace は通常の２次元スレッドの他に Group も定義するモードである．Group とは CUDA の Block，OpenCL の Group に相当するものであり，アーキテクチャにおける SubSlice である．SubSlice ごとにスレッドを管理することから L3 Cache の SLM を使用できる. また，それに付随して Atomic アクセスの利用も可能となる．

3.3 データの扱い

Intel HD Graphics は CPU と共有の DRAM をメインメモリとして扱う構造で，ゼロコピーのようなデータアクセスが容易に行える．しかし，高速な計算において CM では DRAM より高速にアクセスできる Surface と呼ばれるメモリ領域の利用を推奨している. Surface は Host, Device どちらからも Read/Write アクセスができる領域で，２次元メモリ的に扱う Surface2D が主に使われる．最大 16KB x 16KB(256MB)保持可能で定義時にゼロで初期化される．また，GPU からの範囲外アクセスに対応しており，それぞれ Read はゼロを返し，Write は余分なデータを破棄する．cmBuffer と呼ばれる Surface2D の１次元版もあるが，前述の高級な機能はなく，GPU からの 1 回の Read/Write が Surface2D は 2048bit までに対して，cmBuffer は 1024bit と機能に制限がある．

Surface の Read/Write には通常 Dataport を通り転送しているが, Read のみ Sampler を用いることもできる. Sampler では, データを RGBA のような 8bit の整数で連続した４チャンネルとみなし, 0.0~1.0 の 8bitFloat に変換される. Mask で取得するチャンネルを選択することができ，チャンネルごとに１６要素のベクトルに変形された状態で read される．

4. 実装プログラム

本研究では Intel HD Graphics の性能評価として，行列和とスカラ乗算，ShockFilter，ホログラム生成，行列積を用いる．

4.1 行列和とスカラ乗算

float32 の行列データに対して単純なスレッド並列により各要素の加算とスカラ乗算を行う．4096x4096 の行列計算に対して１スレッドの SIMD 幅は 4x4，8x8，16x16，32x32 の４パターンにて評価を行う．実装アルゴリズムはデータをすべて read した後，計算をすべて行い，最後に write するようにした．この評価にて高パフォーマンスにおける１スレッドの計算粒度設定を調査する．

4.2 ShockFilter

ShockFilter とは超解像処理に用いられる非線形エッジ先鋭化フィルタである[4]．主計算は 3x3 のガウシアンフィルタと４近傍ラプラシアンフィルタである．疑似コードを以下に示す．畳み込み計算は conv()に省略している．in は HEIGHTxWIDTH の入力画像である．

CM 実装では 1080x1920 のグレースケール画像に対して１スレッドが 6x24 の画素要素を担当する．畳み込み計算におけるデータアクセスは MatrixSIMD と select 関数により簡単に実装することができる．

本実装では４近傍ラプラシアンは上下左右の４方向にのみデータアクセスをしている．また，if の判別には max と min 関数により実装する．

No	ShockFilter
1	g=conv(in,gaus);
2	delta=conv(g,lap);
3	for(i=0;i<HEIGHT;i++){
4	for(j=0;j<WIDTH;j++){
5	if(d[i][j]>0)sign= 1;
6	elif(d[i][j]<0)sign= -1;
7	norm=sqrt(pow(in[i+1][j]-in[i][j],2)+
8	pow(in[i][j+1]-u[i][j],2):
9	out[i][j]=in[i][j]-(sign*norm*0.25);
10	}
11	}

No	select を使った畳み込み計算
1	matrix<float,8,26>in; //8x26 の matrix を定義
2	matrix<float,6,24>g;
3	for(i=0;i<3;i++){
4	for(j=0;j<3;j++){
5	g+=in.select<6,1,24,1>(i,j);
6	}
7	}

4.3 ホログラム生成

ホログラムとは光の伝搬を数値シミュレーションすることによって得られる物体の３次元情報を記録した媒体である[5]．本研究の実装において物体は L 個の集合で構成し，各物体点座標(plsx，plsy，plsz)をそれぞれ L 個の要素からなるベクトルとして保持している．以下に疑似コードを示す．

No	ホログラム生成
1	for (y=0; y<HEIGHT; y++) {
2	for (x=0; x<WIDTH; x++) {
3	h_r=0.0f;
4	h_i=0.0f;
5	for (l=0; l<L; l++) {
6	tmp_x = pls_x[l] - x*SI;
7	tmp_y = pls_y[l] - y*SI;
8	pow_xy= tmp_x*tmp_x + tmp_y*tmp_y;
9	theta = pow_xy * pls_z[l];
10	h_r += cos(theta);
11	h_i += sin(theta);
12	}
13	tmp_h = atan2(h_i, h_r);
14	hologram[y][x] = tmp_h * NT;
15	}
16	}

本研究では 1080x1920 のホログラム画像生成とし，L は 15984 個，データは入力が float32，出力を uchar で行った．CM 実装では 1 スレッドが 6x24 を担当し，物体点座標はベクトルデータであることから cmBuffer を用いた．cmBuffer は 1 回の read で 1024bit まで転送可能であることから 32 個の plsx，plsy，plsz をレジスタに read，L ループ 32 回分処理を L/32 回繰り返すことにより実装した. atan2 関数は CM テンプレートライブラリの cmtl.h に実装されており，高速に動作することを謳っていることからこのライブラリ関数を使った．

4.4 行列積

一般に公開されている CM のインストールパッケージにはいくつかのサンプルプログラムが付属している．その中の一つに Sgemm がある．この Sgemm プログラムは 1024x1024 の二つの行列による単純な行列積を行っており，スカラ乗算や行列の加算等は行っていない．また，データは Row-major で保存されており，転置も行っていない．処理を図で表したものを図 5 に，疑似コードを以下に示す．ここで行列 A，B，C に対して C=AB を計算する．図 5 において太矢印はループによるデータブロックの移動を表しており，各アルファベットは疑似コードと対応している．また，read は省略されており，実際の記述とは異なる．

No	行列積
1	matrix<float, 32, 8> a;
2	matrix<float, 16, 8> b;
3	matrix<float, 32, 16>c;
4	for (k=0;k<1024/32;k++) {
5	read(A,a); //read 32x8
6	read(B,b.select<8,1,8,1>(0,0)); //read 8x8
7	read(B,b.select<8,1,8,1>(8,0)); //read 8x8
8	for(j=0;j<8;j++) {
9	for(i=0;i<32;i++) {
10	c.select<1,1,8,1>(i,0)+=
11	a(i,j) * b.select<1,1,8,1>(j,0);
12	c.select<1,1,8,1>(i,8)+=
13	a(i,j) * b.select<1,1,8,1>(j+8,0);
14	}
15	}
16	}
17	write(C,c);

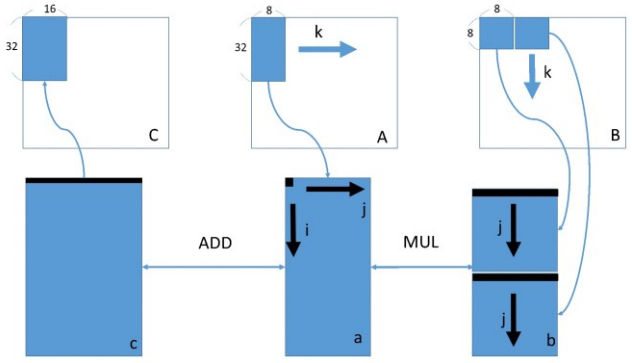

図５:行列積処理図

１スレッドが 32x16 要素を担当し，ブロック分割された行列積をループで足しこむことにより実装されている．b は 8x16 のデータを read するが，レジスタには 16x8 に格納されており縦に配置されている．これは 1 回の計算を物理 SIMD 幅, 1GRF 幅と同じサイズで計算するためであると考えられる．１回のブロック行列に必要なレジスタ数は a の 32x8 と b の 16x8，c の 32x16 であり GRF 換算で 112 個の GRF を使用している. 1 スレッドは 128 個の GRF を有することから，このプログラムではレジスタを最大まで使っていることが分かる.

5. 評価

評価方法は GPU の理論性能(Flops)に対する実測性能による評価と，CPU との計算時間比較による同 SoC 内 GPU を数値計算に活用することの有用性を評価する．ホログラム生成プログラムでは atan2 関数は関数内の処理が不明瞭であることから Flops の算出には atan2 を乗算２回に置き換えたプログラムで評価し, 元のプログラムは CPU との比較評価にて用いる．表１に評価環境を示す.

表 1 評価環境

		備考
OS	Ubuntu18.04.1	
CPU	Intel Core i7-6700K	4C/8T 4.00GHz
GPU	Intel HD Graphics530	1150MHz 24EU/168T 441.6Gflops
DRAM	16GB	
Compiler	g++ 7.4.0	
CM	Release_20190717	

表２:実験結果

	備考	計算時間 (msec)	GFlops	性能率 (%)
MUL+ADD	4x4	8.21	3.80	0.86
	8x8	7.40	4.22	0.95
	16x16	4.36	7.16	1.62
	32x32 ①	43.27	0.72	0.16
	32x32 ②	7.41	4.21	0.95
ShockFilter		0.52	118.16	26.75
ホログラム		1,973.31	203.35	46.04
行列積		5.41	360.36	81.60

表３:計算時間比較(msec)

	MUL+ADD	ShockFilter	ホログラム	行列積
GPU	4.36	0.52	1,973.77	5.41
CPU	8.00	12.00	73,979.00	306.00
比率(CPU/GPU)	1.83	23.07	37.48	56.56

表２と表３に実験結果を示す．表２，３の MUL+ADD は 4.1 の行列和とスカラ乗算である．1 スレッドの担当計算数と計算時間の傾向を見ていくと，16x16 までは計算数に比例して高速になっているが，32x32 にて著しく性能低下を起こしている. この実装ではすべてのデータを read した後，計算するため必要レジスタが 32x32 の時 384GRF と，１スレッドが有する GRF 数を超えてしまい性能低下が発生していると推測した．そこで 32x32②ではレジスタ利用数は 16x16 と同じ 96 個にし，計算と write を４回のループで 32x32 計算を行ったところ，計算時間は 7.41msec まで回復した．このことから CM ではレジスタ数が超過したときにどこかの記憶領域にレジスタデータの退避等をしていると考えられる.

ShockFilter の計算アルゴリズムは計算量とデータ量が同等であることに対してホログラムはすべての画素で同じデータを扱うことから画素サイズに比例して計算量が増大するプログラムである．GPGPU における Flops の実測性能ボトルネックはデータ読み書きに起因することから一般的にこの二つのプログラムの性能率（理論ピーク性能に対する実測性能）はホログラム＞ShockFiler となることは自明である．本実験結果においてもこのような関係が成り立っていることから Intel HD Graphics は GPGPU 用途において他の GPGPU によく使われる NVIDIA や AMD の GPU と同等に評価できることがわかる.

行列積実装では 81.60%という高い性能率を実現している．これは CM 言語の１スレッドが MatrixSIMD と大容量レジスタを扱えることによるピクセル間のデータ共有が容易になったことが結果の理由であると考えられる.

最後に CPU との比較では高性能率を有する行列積にて 56.56 倍高速に動作しており, この結果は並列性のあるプログラムにおいて CM 言語を用いた GPGPU 化は有用であるということを示している．また，各プログラムはすべて GPU の得意な並列性のあるプログラムで実験していることから CPU より高速になり，CPU/GPU の比率においても表２の性能率と同等の関係が見受けられる. 今回 CPU 実行にて SIMD 化を行っていないが，たとえ SIMD 化を行っても最大４倍，AVX512 が利用可能であっても最大 16 倍の高速化になるので，CM による GPU 実行の方が性能が高い.

6. まとめ

本研究では 2018 年に Intel より公開された Intel HD Graphics 向け GPGPU 言語である CM 言語の仕様を明らかにし，いくつかのプログラム実装とサンプルプログラムを含める Flops，計算時間による評価を行った．結果として Intel HD Graphics は GPGPU 用途において他 GPU と同等に評価できることが判明し，さらに行列積プログラムが性能率 81.60%出ていることから, 高性能率を容易に出せる特徴

を持つことが分かった．また，同 SoC 内の CPU との比較において GPU が最大 56.56 倍高速に動作し，IntelSoC の計算資源としての絶対的な価値を高めた．

これらの結果は Intel HD Graphics および CM 言語が GPGPU 用途として非常に有用であることを示しており，昨今のディープラーニング需要に付随して高まる計算機資源需要の供給に貢献できるものだと考えられる．

参考文献

[1] “Intel® Processor Graphics | Intel® Software”, https://software.intel.com/en-us/articles/intel-graphics-developers-guides?_ga=2.244710400.461455803.1555981020-663836114.1555980939

[2] “The Compute Architecture of Intel® Processor Graphics Gen9 “.https://software.intel.com/sites/default/files/managed/c5/9a/The-Compute-Architecture-of-Intel-Processor-Graphics-Gen9-v1d0.pdf

[3] “CM Programming reference | 01.org”. https://01.org/c-for-metal-development-package/downloads/cm-programming-reference-1

[4] 作田 泰隆，川本 祐大，渡辺 将史，後藤 富朗，平野 智，桜井 優 “TV 正則化法と Shock Filter を用い た超解像拡大法，“電子情報通信学会論文，Vol.J96- D No.3，pp.686-694，2013.

[5] 川田 真也， 佐藤 裕幸，” SHIELD TABLET を用いたステレオ画像のホログラム生成” 岩手県立大学ソフトウェア情報学部卒業論文要旨 2018 年.