Vol.2019-ARC-235 No.20
Vol.2019-SLDM-187 No.20
Vol.2019-EMB-50 No.20
2019/3/17

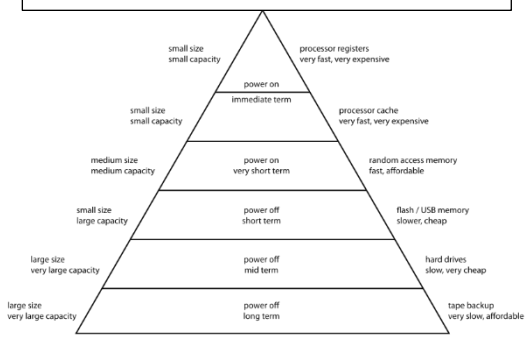
通信帯域の制限を受けずに大容量データを扱うための

CPU、GPU、FPGA をまたがった分散処理

戀川 光央
tonoi 株式会社

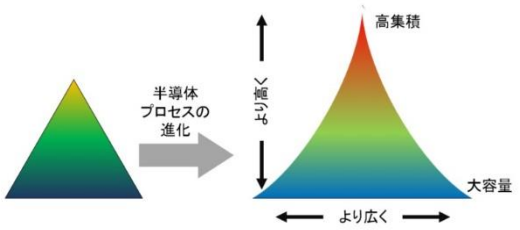
■ あらまし　Speech To Text 対応インターホン開発時に、僻地対応のため通信帯域の制限を受けずにクラウド内の Xeon と IoT の組み込みプロセッサー(Rapsberry Pi3) における分散処理を模索したところ、OpenCL Kernel ファイルを転送し利用する方法を発見した。この手法は簡便かつ応用範囲が広いためここに報告する。

図 1. Computer Memory Hierarchy



■ 背景　速度を上下、容量を左右とする Memory Hierarchy [1]はムーアの法則に従って鼓型に変形し中央部の転送速度が不足してゆく。

図 2. ムーアの法則による変形



転送速度の進化を表すギルダーの法則はムーアの法則より遅く、10 年で相対速度が 1/10 となる。そのため慢性的な転送速度不足が発生している。

■ 課題　転送速度対策としては、データの発生源から極力データを移動させずに処理を行う In Storage Processing が考えられる。しかし、多種プロセッサーに対応したデータ処理指向のプラットフォームが存在せず、コンテナ技術などを利用して独自に開発する必要がある。

■ 提案　データアクセスに関するプログラムを OpenCL で記述し、システムに存在する機器に合わせて事前に OpenCL Kernel Image を作成し、データアクセス時にテータが存在する機器に対応する Kernel Image を転送して実行する。

使用言語は対象となるプロセッサー対応が豊富だった OpenCL 1.2 を採用した。開発対象は Azure の Xeon と Rapsberry Pi3 (VideoCore IV)であったため他に選択肢がなかった。当初は Docker を利用してクラウドからラズパイに処理を渡していたが、僻地の通信環境では Docker Image の転送に時間がかかってしまったため、OpenCL の Kernel だけを転送させる手法を考案した。

後日、上記手法が大容量データ処理の多くのケースで有効であることが指摘された。

Vol.2019-ARC-235 No.20
Vol.2019-SLDM-187 No.20
Vol.2019-EMB-50 No.20
2019/3/17

All Flash Array などの高速ストレージでは、大容量データは複数の inode に分割されて保存されている。ファイルの読み出し時には、inode が集約(Aggregation)され、整列(Sort)され、ファイル転送可能な形式に変換(Encode)される。仮に OpenCL Kernel Image がストレージ機器自身のプロセッサーで動作すれば、上記の三つの処理を経由せずに直接 inode を操作可能となり、系全体の実行効率が上昇する。

下図左が従来の Apache Spark の演算を図式化したものであり、右が今回の提案である。従来の方法では 1-a でファイル読み取りが始まり、1-c で複数 inode の集約(Aggregation)と整列(Sort)が発生している。続いて 1-d でオブジェクトストレージ形式への変換(Encode)が行われている。右図の提案手法ではクラウド内の仮想プロセッサーに 2-a で処理が渡され OpenCL Kernel Image が 2-a-x で転送されている。3-a..e にて Aggregation、Sort、Encode を行わずにデータ処理が行われ、4-c で結果のみがクラウドに戻される。

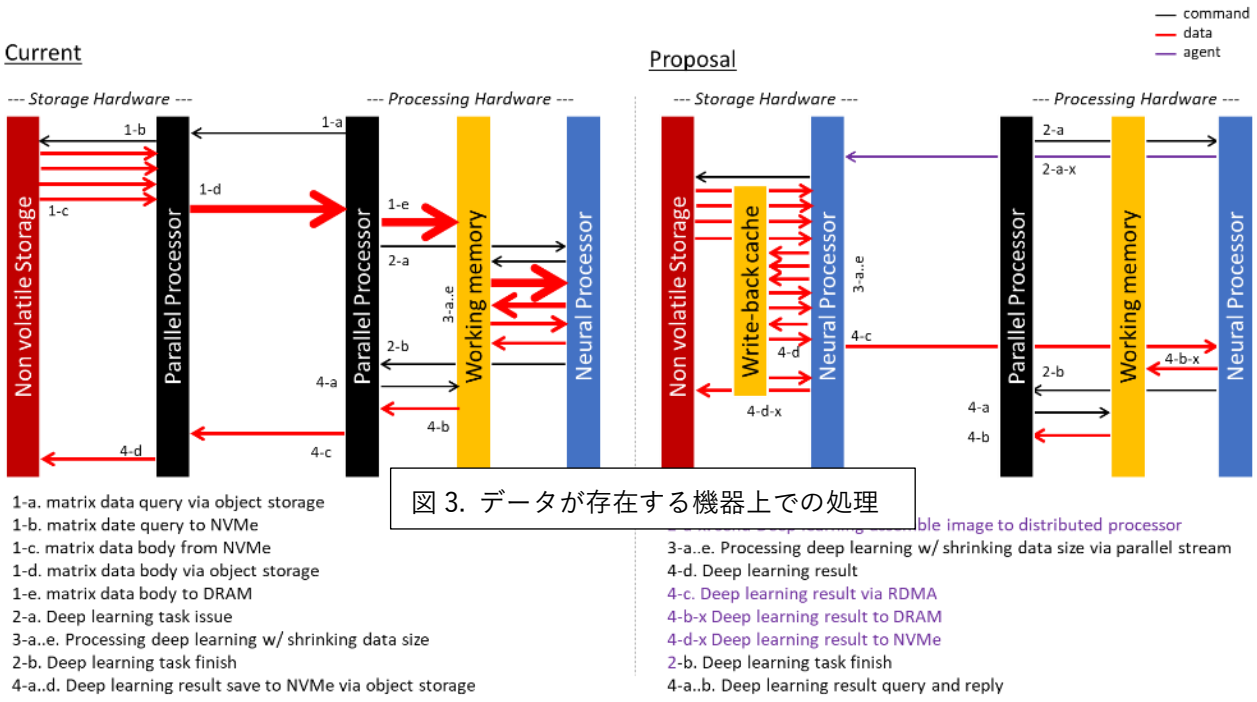


図 3. データが存在する機器上での処理

次ページ図 4 は上記処理の各ステップにおける見積もり時間である。Legacy が従来手法で Proposal が今回の提案となる。10M のファイルを 100 個北米のデータセンターと日本国内でやり取りした場合が Hybrid であり、同ファイル数を同一 LAN 配下で実行した場合が Public である。Issue-finish が実際の処理時間となる。従来手法が最悪ケースで 13 秒かかっているのに対し、提案は 260msec で完了する見込みとなる。処理時間のほとんどは 1-d のデータ転送にかかっており、かりにデータ転送が無いとすると、従来手法は 130msec と OpenCL Kernel Image 転送がない分だけ有利となる。しかし、現実にはデータ転送があるため、提案手法が多くの場合に有利と考えられる。

Vol.2019-ARC-235 No.20
Vol.2019-SLDM-187 No.20
Vol.2019-EMB-50 No.20
2019/3/17

	Legacy					Proposal			
	Hybrid		Public			Hybrid		Public	
1−a	130ms	130	1ms	1	1−a				
1−b	25μs	0.025	25μs	0.025	1−b				
1−c	25ns x src	0.0025	25ns x src	0.0025	1−c				
1−d	130ms x src	13000	1ms x src	100	1−d				
1−e	13.75ns x src	0.00138	13.75ns x src	0.001375	1−e				
2−a	25μs	0.025	25μs	0.025	2−a	25μs	0.025	25μs	0.025
					2−a−x	25μs+130ms	130.025	25μs+1ms	1.025
3−a..e	13.75ns x src^2	0.1375	13.75ns x src^2	0.1375	3−a..e	25μs + 25ns x src +13.75ns x src^2	0.165	25μs + 25ns x src +13.75ns x src^2	0.165
2−b	25μs	0.025	25μs	0.025	2−b	25μs	0.025	25μs	0.025
4−a	13.75ns	1.4E−05	13.75ns	1.38E−05	4−a	13.75ns	0.00001375	13.75ns	0.00001375
4−b	13.75ns x out	1.4E−05	13.75ns x out	1.38E−05	4−b	13.75ns x out	0.00001375	13.75ns x out	0.00001375
					4−b−x	13.75ns x out	0.00001375	13.75ns x out	0.00001375
4−c	130ms x out	130	1ms x out	1	4−c	130ms x out + 25μs x out	130.025	1ms x out + 25μs x out	1.025
4−d	300μs + 25ns x out	0.30003	300μs + 25ns x out	0.300025	4−d	13.75ns x out	0.00001375	13.75ns x out	0.00001375
					4−d−x	13.75ns x out + 300μs + 25ns x out	0.000025	13.75ns x out + 300μs + 25ns x out	0.000025
	Full total (ms)	13260.5		102.5164		Full total (ms)	260.26508		2.26508
	issue>finish (ms)	13130.5		101.5164		issue>finish (ms)	260.265055		2.265055

図 4. データが存在する機器上での処理(見積もり)

■ 実験　実情ではクラウド内では高性能 GPU が使用されエッジでは一般的なプロセッサーが使われるだろう。そこで以下の環境にて実験を行った。

Microsoft Azure

クラウド内 GPGPU:

　nVidia Tesla K80 (5.61Tflops)

エッジ プロセッサー:

　Intel Skylake 530 (0.44Tflops)

図 5. データサイズ x 転送速度における
従来処理時間/提案処理時間 比較（見積もり）

data(Mbyte)	PCIe Gen3x32 (32Gbps)	M2.SSD (3.5Gbps)	Serial SSD (0.5Gbps)
0.001	0.001	0.001	0.001
0.01	0.010	0.010	0.010
0.1	0.100	0.100	0.100
1	0.939	0.993	0.999
10	5.865	9.270	9.889
100	12.346	55.787	89.825
1000	13.880	111.970	468.692
10000	14.054	124.510	810.585
100000	14.072	125.920	874.367
1000000	14.074	126.063	881.301

10M のファイル x100 を 3.5Gbps の帯域環境で処理したところ、従来処理が 30.49 秒、提案が 3.56 秒となり、8.56 倍の高速化が見られた。理論値の 9.27 倍には届かなかったが十分な高速化が見られた。

■ 問題点　OpenCL 自体にはデータアクセスの手法が存在せず、ホストアプリケーションが読み取ったデータを共有メモリにて引き渡し処理をするため、提案の手法には一般にホストアプリケーションをプログラムごとに改編する必要がある。今回の実験ではホストアプリケーションをデータごとに組み替えて対応している。

　弊社では OpenCL から各種データにアクセスする新しい手法を開発中であり、ホストアプリケーションを組み替えることなく様々なデータを利用できるようになる予定である。

■ 今後　パートナー企業が本提案を FPGA 上にて展開する実装を研究開発している。理論上は FPGA 上でも問題なく動作するはずだが、実装進捗に合わせ改めて報告したい。

Vol.2019-ARC-235 No.20
Vol.2019-SLDM-187 No.20
Vol.2019-EMB-50 No.20
2019/3/17

■ 関連研究　東工大 TSUBAME サーバーにて、GPU のメモリ空間を拡張するために Page Fault 時に仮想メモリを読み込む DRGAGON という技術がある。DRAGON は GPU メモリを拡張し、本提案は読み込み先に処理を振り分けるため共存可能と考える。

Intel は OpenMCCA という NVDIMM などの Heterogeneous Memory を仮想メモリ空間にマップして統一的にアクセスを可能にしている。本提案は単一のメモリ空間を用いず、既存のストレージの名前空間上で、実際にファイルが存在するデバイス名を取得し、そこに処理を転送するものであり、スケールアップ型の OpenMCCA に対して本提案はスケールアウト型と考えられる。

文献

[1] Wikipedia
Memory Hierarch
https://en.wikipedia.org/wiki/Memory_hierarchy

[2] K. Kang　Intel Corporation, Canada, P. Yiannacouras　Intel Corporation, Canada
Host Pipes: Direct Streaming Interface Between OpenCL Host and Kernel
IWOCL 2017 Proceedings of the 5th International Workshop on OpenCL Article No. 25

[3] Tatsuyuki　Negoro, Sawako Fujimaki
Changing the Strategic Viewpoint to Adapt the Digitalization: From Value Chain Strategy to Layer Strategy
早稲田大学 WBS 研究センター早稲田国際経営研究 No.44（2013）pp.145-162

[4] Pak Markthub＊, Mehmet E. Belviranli†, Seyong Lee†, Jeffrey S. Vetter† and Satoshi Matsuoka‡＊
DRAGON: Breaking GPU Memory Capacity Limits with Direct NVM Access
The SC18 papers

[5] Jianbin Fang, Henk Sips, Pekka Jääskeläinen, Ana Lucia Varbanescu:
"Grover: Looking for Performance Improvement by Disabling Local Memory Usage in OpenCL Kernels"
The 43rd International Conference on Parallel Processing (ICPP-2014), Minneapolis, USA, September 9-12, 2014

[6] Mitsuo Koikawa, Takeyuki Ogura
Hybrid Computing のコア特許、実装特許
特願 2018-089022、2018-159325

■