低電力化のための GPU，FPGA 自動オフロードの評価

山登　庸次[†a)]（正員：シニア会員）

Evaluation of Automatic GPU and FPGA Offloading for Power Saving

Yoji YAMATO[†a)], *Senior Member*

[†] 日本電信電話株式会社ネットワークサービスシステム研究所，武蔵野市
Network Service Systems Laboratories, NTT Corporation, Musashino-shi, 180–8585 Japan
a) E-mail: yoji.yamato.wa@hco.ntt.co.jp
DOI:10.14923/transinfj.2021JDL8010

あらまし　私は，通常コードを，環境に応じて自動変換し，高性能，低電力で運用可能とする環境適応ソフトウェアを提案し，GPU，FPGA への自動オフロードでの性能向上を検証してきた．本論文では，自動オフロード時の電力使用量を確認し，環境適応による低電力化を検証する．

キーワード　環境適応ソフトウェア，GPGPU，自動オフロード，低電力

1. まえがき

近年，ムーアの法則が減速するのではないかと言われている．そのような状況から，CPU だけでなく，GPU（Graphics Processing Unit）や FPGA（Field Programmable Gate Array）等のデバイスの活用が増えている．例えば，Microsoft 社は FPGA を使って検索効率を高める取り組みをしており [1]，Amazon 社は，FPGA，GPU 等をクラウド技術を用いて（例えば，[2]〜[5]）提供している [6]．また，IoT 機器利用システム（例えば，[7]〜[12]）も増えている．

しかし，ヘテロなデバイスをシステムで適切に活用するためには，デバイス特性を意識した設定やプログラム作成が必要であり，OpenCL（Open Computing Language）[13]，CUDA（Compute Unified Device Architecture）[14] 等の知識や IoT 機器向け組み込みシステムの知識が必要となり，大半のプログラマーにとっては，スキルの壁が高い．そのような壁を取り払い，ヘテロなデバイスを十分利用できるようにするため，プログラマーが処理ロジックを記述したソフトウェアを，配置先の環境（GPU，FPGA 等）にあわせて，適応的に変換，設定し，環境に適合した動作をさせるような，プラットホームが求められている．

そこで，私は，一度記述したコードを，配置先の環境に存在する GPU や FPGA，メニーコア CPU 等を利用できるように，変換，リソース設定等を自動で行い，アプリケーションを高性能かつ低電力で動作させることを目的とした，環境適応ソフトウェアを提案している．あわせて，環境適応ソフトウェアの要素として，アプリケーションコードのループ文や機能ブロックを，GPU，FPGA に自動オフロードする方式を提案し性能向上を評価している [15]〜[17]．しかし，今までは，自動オフロード時の処理時間のみ評価しており，電力使用量については評価されていなかった．本論文は，通常の CPU 向けプログラムを，GPU，FPGA 等のデバイスにオフロードし高性能化させる際に，電力使用量も考慮に入れる方式を提案し，電力使用量低減を，既存アプリケーションのオフロードを通じて確認する．

2. 既存技術

GPU を画像処理以外にも使う GPGPU（General Purpose GPU）を行うための環境に CUDA がある．また，GPU だけでなく，FPGA，メニーコア CPU 等のヘテロなデバイスを同様に扱うための仕様として OpenCL がある．CUDA，OpenCL は，C 言語の拡張を行いプログラムを行う形だが，プログラムの難度は高い．CUDA や OpenCL に比べて，より簡易にヘテロデバイスを利用するため，指示行ベースで，並列処理等を行う箇所を指定して，指示行に従ってコンパイラが，実行ファイルを作成する技術がある．仕様としては，OpenACC [18] や OpenMP 等，コンパイラとして PGI コンパイラ [19] や gcc 等がある．

CUDA，OpenCL，OpenACC 等の技術を用いることで，GPU 等へオフロードすることは可能になってきている．しかしオフロードは行えるようになっても，高速化や低電力化には課題がある．例えば，マルチコア CPU 向けに自動並列化機能をもつコンパイラとして，Intel コンパイラ [20] 等がある．これらは，ループ文中で並列処理可能な部分を抽出して，並列化する．しかし，メモリ処理等の影響で単に並列化しても性能がでないことも多い．GPU や FPGA 等で高速化の際は，OpenCL 等の技術者がチューニングを繰り返している．

このため，ヘテロなデバイスを活用してアプリケーションを高速化，低電力化することは難しいし，自動並列化技術等を使う場合も並列処理箇所探索の試行錯誤等の稼働が必要だった．そのため現状，ヘテロなデバイスに対するオフロードは手動での取組みが主流である．著者は以前に環境適応ソフトウェアのコンセプトを提案し，自動オフロードを検討している．GPU 等に処理をオフロードする際に，並列処理箇所探索を自動化するため，進化計算を用いた手法を提案しているが，処理時間短縮のみの評価であり [21]，電力使用量

の削減は未評価であった．

3. 電力を考慮した自動オフロード

著者はこれまでに，プログラムのループ文の GPU，FPGA 自動オフロード [15], [17]，プログラムの機能ブロックの自動オフロードや，多様言語，混在環境での自動オフロードを提案してきた．これらも踏まえて，**3.1**，**3.2** では，電力使用量を考慮した，ループ文の GPU，FPGA 自動オフロード技術を提案し，**3.3** では，移行先が混在環境での適切なオフロード先選択技術を提案する．

3.1 ループ文の GPU 自動オフロード

共通的な課題として，コンパイラがこのループ文は GPU の並列処理に適しているという適合性を見つけることは難しいのが現状である．GPU にオフロードすることでどの程度の性能，電力消費量になるかは，実測してみないと予測は難しい．そのため，このループを GPU にオフロードするという指示を手動で行い，測定の試行錯誤が行われている．著者はそれを踏まえ，GPU にオフロードする適切なループ文の発見を，進化計算手法である遺伝的アルゴリズム（GA）で自動的に行うことを以前提案した．並列可能ループ文群に対して，GPU 実行の際を 1，CPU 実行の際を 0 と値を置いて遺伝子化し，検証環境で反復測定し適切なパターンを探索する．ここで，測定で短時間処理できるパターンを高い適合度の遺伝子とするが，電力使用量も合わせて測定し，低電力なパターンも高い適合度とする処理を新たに加える．例えば，$(\text{処理時間})^{-1/2} * (\text{電力使用量})^{-1/2}$ のように，短処理時間，低電力使用量なほど遺伝子パターンの適合度が高くなるよう設定する．

ループ文の GPU オフロードについては，電力使用量を適合度に含める進化計算手法と，[21] 等で提案の CPU-GPU 転送の低減により，自動での高速化，低電力化を行う．

3.2 ループ文の FPGA 自動オフロード

FPGA で，どのループをオフロードすれば高速になるかの予測は難しいため，GPU 同様検証環境で自動測定することを提案している．しかし，FPGA は，OpenCL をコンパイルして実機で動作させるまで数時間以上かかるため，GPU 自動オフロードでの GA を用いて何回も反復して測定することは，処理時間が膨大となり行う事はできない．

そこで，FPGA にオフロードする候補のループ文を絞ってから，測定を行う形をとる．具体的には，発見されたループ文に対して，ROSE [22] 等の算術強度分析ツールを用いて算術強度が高いループ文を抽出する．更に，gcov 等のプロファイリングツールを用いてループ回数が多いループ文も抽出する．算術強度やループ回数が多いループ文を候補として，OpenCL 化を行う．OpenCL 化時には，CPU 処理プログラムを，カーネル（FPGA）とホスト（CPU）に，OpenCL の文法に従って分割する．候補ループ文に対して，作成した OpenCL をプレコンパイルして，リソース効率が高いループ文を見つける．これは，コンパイルの途中で，作成するリソースは分かるため，利用するリソース量が十分少ないループ文に更に絞り込む．候補ループ文が幾つか残るため，それらを用いて性能や電力を実測する．選択された単ループ文に対してコンパイルして測定し，更に高速化できた単ループ文に対してはその組み合わせパターンも作り 2 回目の測定をする．測定された複数パターンの中で，短時間かつ低電力のパターンを解として選択する．

ループ文の FPGA オフロードについては，算術強度等を用いて絞り込んでから，測定を行い，低電力パターンの評価値を高めることで，自動での高速化，低電力化を行う．

3.3 混在環境での自動オフロード

GPU，FPGA 等の混在環境での移行先選択についても，性能に加え電力を考慮して選択するようにする．

検証順番として，メニーコア CPU，GPU，FPGA の順で検証し，適切な移行先を選択していく．自動オフロードでは，探索はできるだけ短時間に行う事が期待される．そこで，検証時間がかかる FPGA は最後とし，前の段階で十分ユーザ要件を満足するパターンが見つかっていれば，FPGA 検証は行わない．GPU とメニーコア CPU に関しては，価格的にも検証時間的にも大きな差はないが，メモリが別空間となりデバイス自体が異なる GPU に比して，メニーコア CPU の方が，通常 CPU との差は小さいため，検証順はメニーコア CPU を先として，メニーコア CPU で十分ユーザ要件を満足すれば，GPU 検証は行わない．

三つの移行先を検証し，高性能な移行先を自動選択するのが既存方式であるが，本論文では更に検証環境での実測を通じて短処理時間だけでなく低電力の移行先も，自動選択の候補となるように変更する．例えば，$(\text{処理時間})^{-1/2} * (\text{電力使用量})^{-1/2}$ のように，短処理時間，低電力使用量なほどスコアが高くなるように評価式を設定すれば良い．

データセンタのコストとして，ハードウェアや開発

費等の初期費用が全体コストの 1/3，電力や保守等の運用コストが 1/3，サービスオーダ等のその他費用が 1/3 の例がある．この場合，例えば CPU と GPU 合わせてもハードウェア台数が半減となれば初期費用も低減される．電力使用量半減も運用コスト低減につながる．ただし，運用コストは電力以外要素も多く，電力半減が運用コスト半減になるわけでない．また，ハードウェア価格も，導入する GPU，FPGA 数によりボリューム効果等があり，事業者ごとに異なる．そのため，評価式は事業者ごとに異なる設定となる．

4. 評　　価

本論文では，測定パターンの評価値を定める際に，低電力な程評価値が高くなるような改造を以前実装ツールに加えて，オフロードを行い，低電力化ができることを確認する．

4.1 評 価 対 象

評価対象は，GPU では流体計算の姫野ベンチマーク，FPGA では MRI 画像処理の MRI-Q とする．これらは，IoT 等で良く利用され，GPU，FPGA 評価に適している．紙面等の都合上，評価は一つずつとする．

姫野ベンチマーク [23] は，非圧縮流体解析の性能測定ベンチマークソフトで，ポアソン方程式をヤコビ反復法で解いている．姫野ベンチマークは C 言語や Fortran もあるが，今回は電力測定のためある程度処理時間がかかる Python を用いることとし，処理ロジックを Python で記述した．データは Large で 512∗256∗256 のグリッドで計算する．CPU 処理は，Python の Numpy [24] で処理され，GPU 処理は Numpy Interface を GPU にオフロードする Cupy ライブラリ [25] を介して処理される．

MRI-Q [26] は，非デカルト空間の 3 次元 MRI 再構成アルゴリズムで使用されるスキャナー構成を表す行列 Q を計算する．MRI-Q は C 言語で記述されており，性能測定中に 3 次元 MRI 画像処理を実行し，Large の 64 ∗ 64 ∗ 64 サイズで処理時間を測定する．CPU 処理は C 言語で，FPGA 処理は OpenCL で処理される．

4.2 評 価 手 法

対象 2 アプリケーションのコードを入力し，移行先の GPU や FPGA に対して，Clang [27] 等で認識されたループ文オフロードを試行してオフロードパターンを決める．この際に，処理時間と電力使用量を測定する．最終オフロードパターンについて，電力使用量の時間変化を取得し，全て CPU で処理する場合に比べた低電力化を確認する．

GPU では GA により適切なパターンを選択する．FPGA では GA は行わず，算術強度等を用いて，測定パターンが 4 パターンとなるまで絞り込む．

オフロード対象ループ文：姫野ベンチマーク 13，MRI-Q 16.

パターン適合度：(処理時間)$^{-1/2}$ ∗ (電力使用量)$^{-1/2}$．処理時間と電力使用量が低い程高適合度になる．

表 1　評 価 環 境

Name	Hardware	CPU	RAM	Accelerator	OS	Compiler, etc
Verification Machine for GPU	Gaming PC LEVEL-F039-LQRT2W-XYVI	AMD Ryzen Threadripper 2990WX (32 Core)	32GB ∗2	NVIDIA GeForce RTX 2080 TI (CUDA core: 4352, Memory :GDDR6 11GB)	Ubuntu 16.04.6 LTS	CUDA toolkit 10.1, pyCUDA 2019.1.2, Cupy 7.8, Python 3
Verification Machine for FPGA	Dell Server PowerEdge R740	Intel(R) Xeon Bronze 3104	32GB ∗2	Intel PAC with Intel Arria 10 GX FPGA	CentOS 7.4	Intel Acceleration Stack 1.2, gcc 10.1
Client	HP ProBook 470 G3	Intel Core i5-6200U @2.3GHz	8GB		Windows 10 Pro	

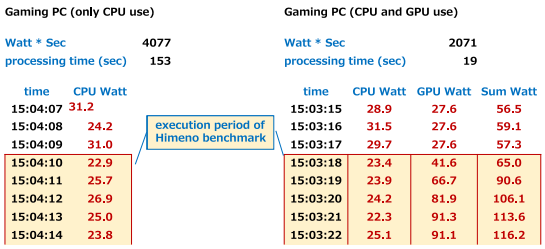

図 1　GPU オフロード時電力使用量（姫野ベンチマーク）

4.3 評 価 環 境

GPU は GeForce RTX 2080 Ti を用いる．電力使用量は，NVIDIA の nvidia-smi で GPU 電力を，s-tui [28] で CPU 電力を測定する．FPGA は Intel PAC with Intel Arria10 GX FPGA を用いる．電力使用量は Dell サーバの IPMI (Intelligent Platform Management Interface) の ipmitool [29] で，サーバ全体電力を測定し，FPGA 利用時と CPU のみ時で比較する．環境を表 1 に示す．

4.4 結果と考察

図 1 は GPU に姫野ベンチマークをオフロードした際の，Watt と時間を示している．最終解まで，姫野ベンチマークでは数時間，MRI-Q では 12 時間かかっている（FPGA コンパイルに 1 回約 6 時間かかる）．全て CPU 処理と比較して，処理時間は 153 秒から 19 秒に短縮されているが，電力は 27W 程度から 109W 程度となっていることが分かる．その結果，Watt ∗ sec は，全 CPU 処理の場合の 4080 Watt ∗ sec から，約 1/2 の 2070 Watt ∗ sec となっている．

図 2 は FPGA に MRI-Q をオフロードした際の，Watt と時間を示している．全て CPU 処理と比較して，処理時間は 14 秒から 2 秒になり，電力もサーバ全体で 121W 程度から 111W 程度となっていることが分かる．その結果，Watt ∗ sec は，全 CPU 処理の場合の 1690

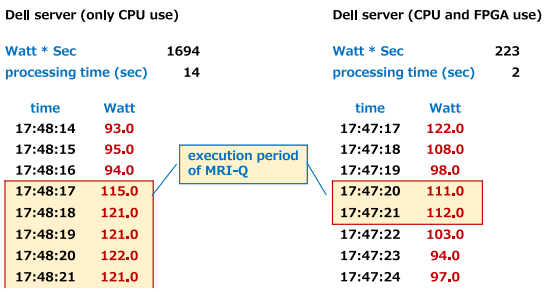

図 2　FPGA オフロード時電力使用量（MRI-Q）

Watt∗sec から，約 1/8 の 223 Watt∗sec となっている．

複数アプリケーションでの低電力化を確認した．姫野ベンチマークの GPU オフロード時は，Watt は増えるが，短時間化効果で全体として低電力化できている．MRI-Q の FPGA オフロード時は，Watt が減っており，短時間化と合わさり，大きく低電力化できている．一般的に FPGA は電力効率が良いと言われるが，今回 MRI-Q の実験でも FPGA の消費電力は低くなることが確認できた．そのため，混在環境でオフロードした際に GPU と FPGA で性能が同程度なら，電力を見て FPGA を選択する等が考えられる．

5. むすび

本論文では，電力を考慮したオフロード手法を提案し評価した．GPU，FPGA 自動オフロード時に検証環境で実測する際に，処理時間に加え電力使用量を取得し，短時間かつ低電力なパターンを高い適合度として，自動コード変換に低電力化を盛り込む．既存アプリケーションの自動オフロードを通じて，低電力化を確認し，方式有効性を確認した．今後は，より多くのアプリケーションを，混在環境を用いて検証する．

文　献

[1] A. Putnam, et al., “A reconfigurable fabric for accelerating large-scale datacenter services,” ISCA’14, 2014.
[2] Y. Yamato, et al., “Fast and reliable restoration method of virtual resources on OpenStack,” IEEE Trans. Cloud Comput., vol.6, Issue 2, pp.572–583, 2018.
[3] Y. Yamato, “Automatic verification for plural virtual machines patches,” Int. Conf. Ubiquitous and Future Networks (ICUFN 2015), July 2015.
[4] Y. Yamato, “Automatic system test technology of virtual machine software patch on IaaS cloud,” IEEJ Trans. Electrical Electron. Eng., vol.10, no.S1, pp.165–167, 2015.
[5] Y. Yamato, “Proposal of optimum application deployment technology for heterogeneous IaaS cloud,” WCSE 2016, pp.34–37, 2016.
[6] AWS EC2 website, https://aws.amazon.com/ec2/
[7] Y. Yamato, et al., “Proposal of shoplifting prevention service using image analysis and ERP check,” IEEJ Trans. Electrical Electron. Eng., vol.12, no.S1, pp.141–145, 2017.
[8] Y. Yamato, “Proposal of vital data analysis platform using wearable sensor,” ICIAE 2017, pp.138–143, 2017.
[9] Y. Yamato, et al., “Security camera movie and ERP data matching system to prevent theft,” IEEE CCNC 2017, pp.1021–1022, 2017.
[10] Y. Yamato, et al., “Proposal of real time predictive maintenance platform with 3D printer for business vehicles,” Int. J. Information and Electronics Engineering, vol.6, no.5, pp.289–293, Sept. 2016.
[11] Y. Yamato, “Experiments of posture estimation on vehicles using wearable acceleration sensors,” IEEE BigDataSecurity 2017, pp.14–17, 2017.
[12] Y. Yamato, et al., “Analyzing machine noise for real time maintenance,” ICGIP 2016, 2016.
[13] J.E. Stone, et al., “OpenCL: A parallel programming standard for heterogeneous computing systems,” Comput. Sci. Eng., vol.12, pp.66–73, 2010.
[14] J. Sanders and E. Kandrot, CUDA by example: An introduction to general-purpose GPU programming, Addison-Wesley, 2011.
[15] Y. Yamato, et al., “Automatic GPU offloading technology for open IoT environment,” IEEE Internet Things J., DOI: 10.1109/JIOT.2018.2872545, 2018.
[16] Y. Yamato, “Improvement proposal of automatic GPU offloading technology,” ACM ICIET 2020, pp.242–246, March 2020.
[17] Y. Yamato, “Automatic offloading method of loop statements of software to FPGA,” Int. J. Parallel, Emergent and Distributed Systems, Taylor & Francis, DOI: 10.1080/17445760.2021.1916020, 2021.
[18] S. Wienke, et al., “OpenACC-first experiences with real-world applications,” Euro-Par Parallel Processing, 2012.
[19] M. Wolfe, “Implementing the PGI accelerator model,” ACM the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, pp.43–50, 2010.
[20] E. Su, et al., “Compiler support of the workqueuing execution model for Intel SMP architectures,” Fourth European Workshop on OpenMP, 2002.
[21] Y. Yamato, “Study of parallel processing area extraction and data transfer number reduction for automatic GPU offloading of IoT applications,” J. Intell. Inf. Syst., Springer, DOI: 10.1007/s10844-019-00575-8, 2019.
[22] Rose framework website, http://rosecompiler.org/
[23] Himeno website, http://accc.riken.jp/en/supercom/
[24] Numpy website, https://numpy.org/
[25] Cupy website, https://cupy.dev/
[26] MRI-Q website, http://impact.crhc.illinois.edu/parboil/
[27] Clang website, http://llvm.org/
[28] s-tui website, https://github.com/amanusk/s-tui
[29] ipmitool website, https://github.com/ipmitool/ipmitool

（2021 年 11 月 4 日受付，2022 年 1 月 6 日早期公開）