電気学会論文誌 X（XX 部門誌）
IEEJ Transactions on xx
Vol.143 No.18 pp.1–6　DOI: 10.1541/ieejxxs.143.1

論　文

環境適応ソフトウェアのための運用中再構成ステップの評価

非会員　山登　庸次*

Evaluation of reconfiguration step on operation for environment adaptive software

Yoji Yamato*, Non-member

We have proposed environment-adaptive software that automatically converts normal code according to the environment and enables high-performance operation. In this paper, we study the overall processing performed for reconfiguration during operation, and newly propose resource amount reconfiguration, measure the processing time for reconfiguration during operation, and discuss reconfiguration timings.

キーワード：環境適応ソフトウェア, 自動オフロード，最適化，運用中再構成
Keywords: Environment Adaptive Software, Automatic Offloading, Optimization, Reconfiguration during Operation

1. はじめに

近年，ムーアの法則の鈍化が予想されている．そのような状況から，CPU だけでなく，FPGA（Field Programmable Gate Array）や GPU（Graphics Processing Unit）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 [1]，Amazon 社は，FPGA, GPU をクラウド技術を用いて（例えば, [2] [3]）提供している [4]．また，IoT 機器利用（例えば, [5]- [10]) も増えている．

しかし，GPGPU（General Purpose GPU）[11] 等，CPU 以外のデバイスを適切に活用するためには，デバイス特性を意識したプログラム作成が必要で，OpenMP（Open Multi-Processing）[12]，OpenCL（Open Computing Language）[13]，CUDA（Compute Unified Device Architecture）[14] や組み込み技術といった知識が必要になり，スキルの壁が高い．そこで，著者は，一度記述したコードを，配置先環境の FPGA や GPU 等を利用できるよう，変換，配置等を自動で行い，アプリを高性能に動作させる，環境適応ソフトウェアを提案した．合わせて，その要素として，アプリのループ文等を，FPGA，GPU に自動オフロードする方式，アプリのリソース量や配置を適切化する方式を提案評価している [15]- [22]．さらに，運用開始後に，利用状況に応じて構成を変更する FPGA，GPU，配置再構成方式も提案評価している．

本稿では，個々に検討してきた運用中再構成について，行う全体処理を検討し，その中で，未検討だったリソース量再構成を新たに検討し，さらに，運用中再構成処理時間を測定し，実施時期の材料を集め考察する．本稿は，国際会議 IECC2023 予稿を洗練化，結果追加等行い，発展させている．

本稿の構成は以下の通りである．2 節で，既存技術を概説するとともに，運用中再構成全体像と新規提案であるリソース量再構成含めて各処理ステップを説明する．3 節で，各処理ステップの処理時間を測定し，考察する．4 節で関連研究と比較する．5 節でまとめる．

2. 運用中再構成

〈2･1〉 既存提案　環境適応実現のため，著者は図 1 の処理フローを提案している．環境適応ソフトウェアは，環境適応機能を中心に，検証環境，商用環境，テストケース DB，コードパターン DB，設備リソース DB が連携する．

ここで，Step 1-6 は，運用開始前に必要となる，コードの変換，リソース量の設定，配置場所の設定，動作確認を行うが，Step 7 は，運用開始後に，運用中利用状況に応じて，構成を変更した方が良い際に再構成する．

〈2･2〉 運用中再構成全体処理　運用中再構成を Step 7 で行うが，運用中再構成の目的を明確にする．環境適応ソフトウェアでは，利用開始前に，ユーザの想定する試験パターンで最適化を行い，コードの変換，リソース量の設定，配置場所の設定を行う．しかし，実際に運用開始後は，事前に想定したリクエスト傾向ではなく，想定外の利用傾向になる可能性もある．例えば，SQL 処理をアクセラレートする FPGA ロジックで運用開始したが，半年後は，NoSQL のクエリが主流になっていた場合等である．このような場合でもより適切な構成に再構成することで，高速な性能を

* 日本電信電話（株） ネットワークサービスシステム研究所.
〒 180-8585　東京都武蔵野市緑町 3-9-11.
Network Service Systems Laboratories, NTT Corporation. 3-9-11 Midori-cho, Musashino-shi, Tokyo 180-8585 Japan.

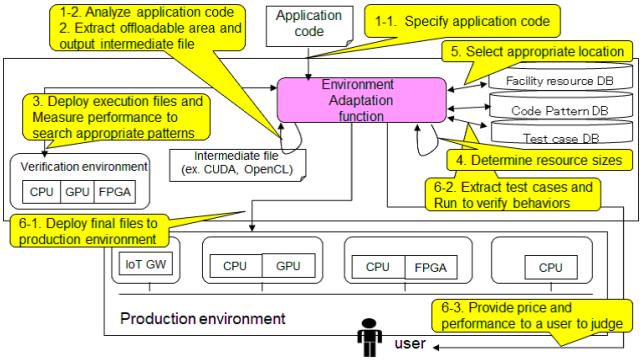


Fig. 1. Processing flow of environment-adaptive software

維持するため，実際の利用データを分析した運用中再構成を自動で行う．運用中再構成処理では，運用開始前に最適化を行う環境適応処理を，事前の想定データでなく，実際の利用データに適切に行う．行う環境適応処理は，FPGA ロジック再構成，GPU ロジック再構成，リソース量再構成，配置再構成である．これらは，全て一括しても良いし，各処理だけ行っても良く，ビジネス形態に応じる．

〈2･3〉 各運用中再構成処理

〈2･3･1〉 FPGA，GPU ロジック再構成 FPGA ロジック再構成については [20] で，GPU ロジック再構成については [21] で詳細を提案評価している．以下の 6 段階で，ロジック再構成は行われるが，FPGA も GPU も大きな考え方としては同じである．

1. 一定期間の，商用リクエストデータ履歴を分析し，処理時間負荷上位の複数アプリを特定し，そのアプリ利用時のデータサイズの最頻データを取得する．
2. 複数の負荷上位アプリで，最頻データのテストケースを高速化する，オフロードパターンを検証環境測定試行を通じて抽出．
3. 現オフロードパターンと抽出した複数の新オフロードパターンの処理時間を測定し，商用利用頻度に基づく性能改善効果を求める．
4. 新オフロードパターン性能改善度効果が現オフロードパターンのそれの閾値以上であるかで再構成提案を判断．
5. ユーザに FPGA/GPU 再構成実施を提案し，OK/NG の返答を得る．
6. 商用環境で別 OpenCL/OpenACC [23] を起動することで静的再構成を実施．

〈2･3･2〉 リソース量再構成 運用開始前リソース量設定については，[18] で提案評価しているが，その再構成は，本稿で新検討する．

運用開始前に，CPU とオフロード先の適切なリソース比を決めるため，[24] も参考に，何れかのデバイス処理がボトルネックとなる事を避けるため，想定テストケースの処理時間を元に，CPU とオフロードデバイスの処理時間が同等オーダーになるように，リソース比を定める．次に商用環境へのアプリの配置を行うが，商用環境への配置の際は，ユーザが指定したコスト要求を満たすように，リソース比を可能な限りキープして，リソース量を決定する．

運用開始後は，事前想定テストケースでなく，そのアプリ利用時最頻データを取得し，そのデータでのテストケースを用いて，適切なリソース比を計算する．運用開始後の大きな価格上昇はユーザも許容しがたいと考えるため，リソース量決定時は，当初価格と同程度か低価格になる場合だけ，再構成提案する形とする．

〈2･3･3〉 配置再構成 配置再構成については，[19] で詳細を提案評価している．オフロードアプリの配置については，運用開始前は他者アプリの配置状況に応じて適切に配置できるが，運用開始後は，他者アプリ配置も増えているため，それらの配置も考慮した全体最適の配置が必要となる．

再構成配置は (1)-(3) 式に応じ，GLPK [25] 等の線形計画ソルバで計算され決定されるが，ユーザ満足度に応じる値 S の計算式 (1) が，運用開始前から新たに加わる．個別ユーザの満足度は，再構成前応答時間 R_k^{before} が再構成後 R_k^{after} で X 倍になったら X が，再構成前価格 P_k^{before} が再構成後 P_k^{after} で Y 倍になったら Y が満足度関連値となる．再配置計算の目的関数は，再構成対象の全ユーザ群満足度に関連した値であり，複数アプリに対して（X+Y）の総和を最小化する配置を計算して，全体最適の配置を求める．

$$S = \sum_{k \in App} \left(\frac{R_k^{after}}{R_k^{before}} + \frac{P_k^{after}}{P_k^{before}} \right) \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (1)$$

$$\sum_{i \in Device} (A_{i,k}^d \cdot B_{i,k}^p) + \sum_{j \in Link} (A_{j,k}^l \cdot \frac{C_k}{B_k^l}) = R_k^{after} \leq R_k^{upper} \cdots (2)$$

$$\sum_{i \in Device} a_i (\frac{A_{i,k}^d \cdot B_k^d}{C_i^d}) + \sum_{j \in Link} b_j (\frac{A_{j,k}^l \cdot B_k^l}{C_j^l}) = P_k^{after} \leq P_k^{upper} \quad (3)$$

3. 処理時間測定

今までは，個々のステップの再構成の評価のみしていたが，本稿では環境適応ソフトウェアの行う再構成処理ステップ全てを実装し，各処理の処理時間を測定し，実施タイミング等を考察する．

〈3･1〉 測定条件 FPGA 再構成では，信号処理 tdFIR [26] を運用開始前に既存手法 [16] で FPGA にオフロードしておき，CPU で動作する画像処理 MRI-Q [27] 含めて，運用開始する．一定期間リクエストの負荷をかけ，性能改善効果が高いオフロードパターンへの再構成提案を行い改善を確認する．

FPGA オフロード対象ループ文数：tdFIR 6，MRI-Q 16

算術強度絞り込み：算術強度分析の上位 4 つのループ文に絞り込み

リソース効率絞り込み：リソース効率分析の上位 3 つのループ文に絞り込み

実測オフロードパターン数：4（1 回目は上位 3 つのループ文オフロード測定し，2 回目は 1 回目で高性能だった 2 つのループ文オフロード組合せパターン測定．）

リクエスト負荷：tdFIR 200 req/h，MRI-Q 10 req/h の負荷を 3 つのデータサイズで行う．tdFIR では，162KB，2.06MB，33.0MB のサンプルデータを 75:120:5 の比でリクエストする．MRI-Q では，32*32*32，64*64*64，64*64*64 のダブルサイズ（64*64*64 をコピーして追加）のサンプルデータを 3:5:2 の比でリクエストする．

負荷分析時間：2 時間

負荷上位アプリケーションの数：2

性能改善効果閾値：2

GPU 再構成では，フーリエ変換 NAS.FT [28] を運用開始前に既存手法 [22] で GPU にオフロードし，CPU 動作の流体計算姫野ベンチマーク [29] を含めて，運用開始する．一定期間リクエスト負荷をかけ，性能改善効果高いオフロードパターンへ再構成提案を確認する．

GPU オフロード対象ループ文数：NAS.FT 81，姫野ベンチマーク 13

個体数 M：NAS.FT 30，姫野ベンチマーク 10

世代数 T：NAS.FT 30，姫野ベンチマーク 10

適合度：(処理時間)$^{-1/2}$　処理時間が短い程高適合度になる．(-1/2) 乗により，処理時間が短い特定個体の適合度が高くなり過ぎ，探索範囲が狭くなるのを防ぐ．

選択：ルーレット選択．ただし，世代での最高適合度遺伝子は交叉も突然変異もせず次世代に保存するエリート保存も合わせて行う．

交叉率 Pc：0.9

突然変異率 Pm：0.05

リクエスト負荷：NAS.FT 20 req/h，姫野ベンチマーク 30 req/h の負荷を 3 つのデータサイズで行う．NAS.FT では，クラス W，A，B のサンプルデータサイズで，3:5:2 の比でリクエストする．姫野ベンチマークでは，M，L，XL のサンプルデータサイズで，2:5:3 の比でリクエストする．

負荷分析時間：2 時間

負荷上位アプリケーションの数：2

性能改善効果閾値：2

リソース量再構成では，NAS.FT と姫野ベンチマークを，当初想定データサイズでリソース量を決め運用開始後，実際利用データサイズが想定と大きく異なる際に，リソース量が再構成されることを確認する．

当初想定のデータサイズは，NAS.FT はサイズ W，姫野ベンチマークはサイズ XL とし，実際の負荷をかけるデータサイズは，NAS.FT はサイズ B，姫野ベンチマークは M とする．GPU リソースの分割，再割当には NVIDIA vGPU [30] を用いる．リソースは，CPU は 1 Core 単位，GPU は 4GB RAM 単位で割当でき，CPU 1 Core：GPU 4GB RAM のコスト比は 1:2.5 とする．

配置再構成では，NAS.FT を GPU，MRI-Q を FPGA にオフロードした際の，複数ユーザ平均満足度を向上する，全体配置最適化をシミュレーションする．600 アプリを順に配置し，新たに 100 アプリ配置した際に，100，200，400，700 アプリを再配置計算対象とした際の最適配置を計算する．シミュレーション条件は以下の通りである．

トポロジーは 4 層で構成され，クラウドレイヤー拠点数は 5，キャリアエッジレイヤーは 20，ユーザエッジレイヤーは 60，インプットノードは 300 とする．

クラウドでは，サーバは CPU 8 台，GPU 16GB RAM 4 台，FPGA 2 台，キャリアエッジでは，CPU 4 台，GPU 8GB RAM 2 台，FPGA 1 台，ユーザエッジでは，CPU 2 台，GPU 4GB RAM 1 台とする．サーバ 1 台の全リソース使用月額はクラウドでは 5 万，10 万，12 万とした．キャリアエッジ，ユーザエッジは割高で，クラウドの 1.25 倍，1.5 倍とした．リンクについては，クラウド-キャリアエッジ間は 100Mbps，キャリアエッジ-ユーザエッジ間は 10Mbps の帯域とする．リンクコストは，100Mbps のリンクは月額 8,000 円，10Mbps のリンクは月額 3,000 円とした．

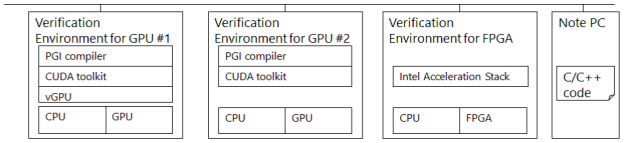
アプリが利用するリソースとして，NAS.FT は，利用リソース量は GPU 1GB RAM，利用帯域 2Mbps，転送データ量 0.2MB，処理時間 5.8 秒で，MRI-Q は，利用リソース量は FPGA サーバの 10%，利用帯域 1Mbps，転送データ量 0.15MB，処理時間 2.0 秒である．

アプリの配置依頼は，300 のインプットノードから上位ノードにランダムに生じさせる．配置依頼数として，NAS.FT：MRI-Q=3:1 の割合で最初に 600 回アプリの配置依頼をする．ユーザ要求条件として，配置依頼する際に価格か応答時間かその両方が選ばれる．NAS.FT の場合，価格に関しては月 7,500 円 (a) か 8,500 円 (b) か 10,000 円 (c) 上限か，応答時間に関しては 6 秒 (A) か 7 秒 (B) か 10 秒 (C) 上限かが選択される．MRI-Q の場合，価格に関しては月 12,500 円 (x) か 20,000 円 (y) 上限か，応答時間に関しては，4 秒 (X) か 8 秒 (Y) 上限が選択される．ユーザ要求として，NAS.FT では a, b, c, A, B, C, aC, bB, bC, cA, cB, cC をそれぞれ 1/12 ずつ，MRI-Q では x, y, X, Y, xY, yX, yY をそれぞれ 1/7 ずつの確率で選択する．

〈3･2〉 測定環境　測定環境を図 2 に示す．FPGA 再構成には，Intel FPGA PAC D5,005 を用いて，Intel Acceleration Stack 2.0 で FPGA を制御する．GPU 再構成には，NVIDIA GeForce RTX 2,080 Ti を用いて，PGI Compiler 19.10 [31] で GPU を制御する．リソース量再構成では，NVIDIA Tesla T4 の GPU リソースを，NVIDIA vGPU 12.2 [30] で分割して利用する．配置再構成のみ実機でなく，GLPK 5.0 [25] を用いてシミュレーションを行う．

〈3･3〉 結　果　FPGA 再構成，GPU 再構成，リソース量再構成，配置再構成の再構成による改善結果と処理時間を示す．

図 3 は，再構成提案前後の FPGA オフロードの処理時間改善度とそれに関連する一定期間の処理時間合計を示している．再構成前は，tdFIR がオフロードされており，リクエストの処理時間合計の 159 秒が 2 時間の負荷である．分析した結果，tdFIR と MRI-Q の 2 つが負荷上位アプリケーションとなる．運用開始後の最頻データを使ってオフロー



Name	Hardware	CPU	RAM	GPU/FPGA	OS	NVIDIA vGPU	CUDA toolkit	PGI compiler	Intel Acc
Verification Environment for GPU #1	Supermicro SYS-1029GP-TR	Intel Xeon Silver 4210R (10 core, 2.4GHz)	128 GB	NVIDIA Tesla T4 *2 (CUDA core: 2560, Memory: GDDR6 16GB)	RedHat Enterprise Linux 7.9	12.2	10.1	19.10	
Verification Environment for GPU #2	LEVEL-F039-LCRT2W-XYVI	AMD Ryzen Threadripper 2990WX	64 GB	NVIDIA GeForce 2080 (CUDA core: 4352, Memory: GDDR6 11GB)	Ubuntu 16.04.6 LTS		10.1	19.10	
Verification Environment for FPGA	Dell PowerEdge R740	Intel(R) Xeon Bronze 3206R *2	128 GB	Intel PAC D5005 (Intel Stratix 10 GX FPGA)	CentOS 7.9				2.0
Client	HP ProBook 470 G3	Intel Core i5-6200U	8GB		Windows 10 Pro				

Fig. 2. Verification environments

	Offload application	Improvement of processing time	Summation of processing time
Before reconfiguratoin	tdFIR	41.1 sec/h	159 sec
After reconfiguratoin	MRI-Q	252 sec/h	549 sec

Fig. 3. FPGA logic reconfiguration results

	Offload application	Improvement of processing time	Summation of processing time
Before reconfiguratoin	NAS.FT	308 sec/h	2,420 sec
After reconfiguratoin	Himeno benchmark	1,180 sec/h	2,790 sec

Fig. 4. GPU logic reconfiguration results

	Offload application	Set resource amount	Cost	Cost performance
Before reconfiguration	NAS.FT	CPU : GPU = 2core : 8GB	7,000	1
After reconfiguration	NAS.FT	CPU : GPU = 2core : 8GB	7,000	
Before reconfiguration	Himeno benchmark	CPU : GPU = 1core : 16GB	11,000	1
After reconfiguration	Himeno benchmark	CPU : GPU = 1core : 8GB	6,000	1.5

Fig. 5. Resource amount reconfiguration results

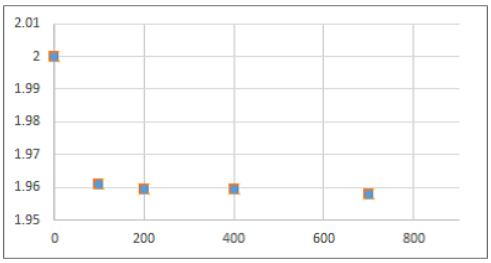


Fig. 6. Applications re-placement results

ドパターンを探索し，商用利用回数をかけることで，処理時間削減改善度は，再構成前 tdFIR で 41.1 秒/時間，再構成後 MRI-Q で 252 秒/時間となる．図 3 から，改善度閾値 2.0 をチェックし，tdFIR から MRI-Q にオフロードアプリを変更した再構成が提案される．アプリのコンパイル時間が支配的であるため，処理時間はそれに依存するが，両アプリとも 1 回のコンパイルが 6 時間程度で，4 パターン測定で 1 日程度かかっている．

図 4 は，再構成提案前後の GPU オフロードの処理時間改善度とそれに関連する一定期間の処理時間合計を示している．再構成前は，NAS.FT がオフロードされており，リクエストの処理時間合計の 2,420 秒が 2 時間の負荷である．分析した結果，姫野ベンチマークと NAS.FT の 2 つが負荷上位アプリケーションとなる．運用開始後の最頻データを使ってオフロードパターンを探索し，商用利用回数をかけることで，処理時間削減改善度は，再構成前 NAS.FT で 308 秒/時間，再構成後姫野ベンチマークで 1,180 秒/時間となる．図 4 から，改善度閾値 2.0 をチェックし，NAS.FT から姫野ベンチマークにオフロードアプリを変更した再構成が提案される．for 文数等のアプリサイズに処理時間は依存するが，NAS.FT の場合で，6 時間程度で再構成後のオフロードパターン探索を行っている．

図 5 は，NAS.FT をデータサイズ W，姫野ベンチマークをデータサイズ XL で最適化して運用開始した後，運用開始後の最頻データがそれぞれサイズ B，M になった際の，再構成前後の設定リソース量，コスト，コスト対効果を示している．NAS.FT では，データサイズが W から B に大きくなり，計算量が増えたため，CPU に対して GPU のリソースを増やした方が良いと計算し，GPU を 16GB RAM に増やした方がコスト対効果が上がると考えられたが，価格が大きく上昇するため，再構成は提案されない．一方，姫野ベンチマークでは，データサイズが XL から M に小さくなり，計算量が減ったため，CPU に対して GPU のリソースを減らして良いと計算し，GPU を 8GB RAM に減らす提案をし，コスト対効果を 1.5 倍にしている．リソース量再構成の提案は数秒以内にされている．

図 6 は，配置再構成の GLPK でのシミュレーションで，再配置したアプリの $R_k^{after}/R_k^{before}+P_k^{after}/P_k^{before}$ の平均値を縦軸に取ったグラフである．若干ばらつきはあるが，再配置対象アプリ数の約 1 割弱が，実際に再構成されていることがわかる．図 6 より，再配置を行ったアプリは $R_k^{after}/R_k^{before}+P_k^{after}/P_k^{before}$ の平均が 1.96 程度に改善されていることが分かる．この値は 2 から大きく改善される値ではないが，例えば，NAS.FT をキャリアエッジからクラウドに配置を変えた際は応答時間が 6.6 秒から 7.4 秒になるが，価格が約 8,400 円から約 7,000 円となるため，値は 2 から 1.954 になる．再配置計算時間は，再配置対象のアプリが増えるにつれ，線形計画の条件式が増えることになるが，700 アプリでも 1 分で終わっている．

〈3･4〉 考　　察　FPGA，GPU，配置再構成は別論文で評価しているが，今回新たな提案のリソース量再構成も，データサイズ傾向を分析して，価格が同程度から低くなる場合に，コスト対効果を上げる再構成を提案しており，ユーザ利便性は高い．

運用中再構成の各処理時間については，アプリによるが，今回の測定では，FPGA 再構成は 1 日，GPU 再構成は 6 時間，リソース量再構成は数秒，配置再構成は 700 アプリ再構成計算の場合に 1 分となっている．FPGA の場合は，アプリのコンパイルに 6 時間程度かかるため，検証環境測定数が 4 でも 1 アプリで 1 日程度かかっている．GPU の場

合は，再構成パターン探索に遺伝的アルゴリズムを用いるが，多世代多個体で多数の測定を行うため，1 アプリで 6 時間程度かかっている．リソース量再構成は，最頻データ選出とその際の処理時間に応じたリソース比，量計算であり数秒である．配置再構成は，ソルバの線形計画計算時間に依存するが，100 アプリの場合 10 秒以内だが，700 アプリの場合 1 分と，アプリ数増に伴い，長くなる．

リソース量や配置の変更は，価格に影響するため，頻繁な変更は望ましくない．そのため，再構成試行は 1-数か月に一回程度を想定している．ただし，運用開始時は，全ての適応処理を行ってから開始するが，運用中再構成は，個々の処理で独立に行ってよい．例えば，FPGA，GPU ロジックやリソース量の再構成は 3 か月に 1 回と一定期間毎にして，配置再構成は，100 アプリ増えるごとに等一定増加数毎にする等，クラウドサービス等のビジネス形態に応じて定めることができる．

4. 関連研究

自動で GPU にオフロードする領域を探索する技術として，[32] [33] が上げられる．[32] [33] は，GA によりオフロードする部分の最適化を行っている点で，著者の以前研究と同様である．しかし，[32] は，流体計算の姫野ベンチマーク等，GPU での高速化が数多く行われているアプリケーションを対象としており，世代数 200 で評価を行い，長時間の繰返し実行を必要としている．今回も利用している著者の以前提案技術は，CPU 向け汎用アプリケーションを GPU で高速化する際に，一定時間で利用開始できることを狙っており，その点は既存研究と異なる．GPU へのオフロードについては，[34] [35] [36] 等があげられる．[34] は，C++ expression template の GPU オフロードのため，メタプログラミングと JIT コンパイル利用をあげており，[35] [36] は OpenMP 使った GPU へのオフロードに取り組んでいる．新たな開発モデルや指示句の手動挿入等を行わず，著者が狙うような既存コードを自動で GPU 向けに変換する研究は少ないと言える．

FPGA のオフロードに関しては多くの研究がある [37], [38], [39], [40]．Liu et al [37] は，ネストされたループを FPGA にオフロードする方法を提案し，さらに 20 分の手作業でネストされたループをオフロードできることを示している．Alias et al [38] は，アルテラ HLSC2H を使用するときに HLS ツールが C 言語コードやループタイリングなどを指定して FPGA を構成する方法を提案している．Sommer et al によって提案された方法 [39] を使用して，OpenMP コードを解釈し，FPGA オフロードを実行できる．Putnum et al [40] は，CPU-FPGA ハイブリッドマシンを使用して，わずかに変更された標準 C 言語でプログラムを高速化している．これらの方法では，OpenMP または他の仕様を使用して並列化するパーツなどの命令を手動で追加する必要がある．SYCL [41] は，ヘテロジニアスハードウェアでの単一ソースプログラミングモデルで，DPC++ [42] は Intel の SYCL コンパイラである．しかし，これらは新たなコードの手動作成が不要な本稿とは狙いが異なる．本稿のように既存のコードを FPGA に自動的にオフロードしたり，再構成したりする研究はほとんどない．

アプリケーションを CPU と GPU 処理のコードに変換し，コード自体は適切にできても，CPU と GPU とのリソース量が適切なバランスでない場合は，コストパフォーマンスが低い．例えば，あるアプリケーション処理を行う際に，CPU での処理時間が 1000 秒，GPU での処理時間が 1 秒では，GPU にオフロードする処理を GPU で高速化しても，全体的には CPU がボトルネックとなっている．[24] では，CPU と GPU を使って MapReduce フレームワークでタスク処理している際に，CPU と GPU の実行時間が同じになるよう Map タスクを配分することで，全体の高性能化を図っている．著者の再構成でも CPU と GPU の実行時間バランスを調整している．

ネットワーク上に存在するリソースの最適利用に関して，ネットワーク上にあるサーバ 群に対して VN（Virtual Network）の埋め込み位置を最適化する研究がある [43] [?]．これらの研究では，通信トラヒックを考慮した VN の最適配置を決定する．これらの研究のメインターゲットは設備設計であり，トラフィック増加量等を見て設計する．具体的には，ユーザ毎に異なるアプリケーションを前提にした，アプリケーション毎の処理時間やコスト等，また，エッジやクラウドサーバ等の多様な配置環境の考慮がされていない．本稿は，環境適応ソフトウェアの要素として，ユーザ毎に異なるアプリケーションを変換し，ユーザリクエストに応じて適切に再配置する事を目的としている．

以上のように，GPU，FPGA へのオフロードによる高速化は数多くの取組みがあるが，既存コードを自動でオフロードするような取組みはほとんどない．また，運用開始前に GPU へオフロードするための変換だけであり，基本的に高速化は一度だけ行われており，本稿で対象としているような，ユーザの利用特性をフィードバックして，運用中のアプリケーションの GPU オフロードロジックを新たな GPU オフロードロジックに再構成して，コストパフォーマンスを高めるような検討はない．

5. ま と め

本稿では，環境適応ソフトウェアの運用中再構成について，行う再構成の全体処理を検討し，その中で，リソース量再構成を新たに検討追加し，運用中再構成の各処理時間を測定した．

FPGA，GPU ロジックについては，商用最頻データを用いて，検証環境で定期的に最適化試行することで，適切なパターンを見つける．リソース量は，商用最頻データでの CPU とオフロード先処理時間から，適切なリソース比，量を見つける．配置は，線形計画手法で，応答時間と価格条件で全体最適化を計算することで，適切な配置を見つける．全再構成処理を実装し，処理時間を測定した．FPGA 再構

成は 1 日，GPU 再構成は 6 時間，リソース量再構成は数秒，配置再構成は 700 アプリで 1 分であった．

運用中再構成の確認した処理時間に応じて，各再構成ステップの商用実施タイミングについては商用担当者と議論し決定する．

文　献

（1） A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao and D. Burger, ”A reconfigurable fabric for accelerating large-scale datacenter services,” Proceedings of the 41th Annual International Symposium on Computer Architecture (ISCA’14), pp.13-24, June 2014.

（2） O. Sefraoui, M. Aissaoui and M. Eleuldj, ”OpenStack: toward an open-source solution for cloud computing,” International Journal of Computer Applications, Vol.55, No.3, 2012.

（3） Y. Yamato, Y. Nishizawa and S. Nagao, ”Fast Restoration Method of Virtual Resources on OpenStack,” IEEE Consumer Communications and Networking Conference (CCNC 2015), pp.607-608, Jan. 2015.

（4） AWS EC2 web site, https://aws.amazon.com/ec2/instance-types/

（5） M. Hermann, T. Pentek and B. Otto, ”Design Principles for Industrie 4.0 Scenarios,” Rechnische Universitat Dortmund. 2015.

（6） H. Noguchi, T. Demizu, N. Hoshikawa, M. Kataoka and Y. Yamato, ”Autonomous Device Identification Architecture for Internet of Things,” 2018 IEEE 4th World Forum on Internet of Things (WF-IoT 2018), pp.407-411, Feb. 2018.

（7） Y. Yamato, ”Automatic Verification for Plural Virtual Machines Patches,” The 7th International Conference on Ubiquitous and Future Networks (ICUFN 2015), pp.837-838, July 2015.

（8） H. Noguchi, M. Kataoka and Y. Yamato, ”Device Identification Based on Communication Analysis for the Internet of Things,” IEEE Access, DOI: 10.1109/ACCESS.2019.2910848, Apr. 2019.

（9） Y. Yamato, Y. Fukumoto and H. Kumazaki, ”Proposal of Real Time Predictive Maintenance Platform with 3D Printer for Business Vehicles,” International Journal of Information and Electronics Engineering, Vol. 6, No. 5, pp.289-293, Sep. 2016.

（10） H. Noguchi, T. Demizu, M. Kataoka and Y. Yamato, ”Distributed Search Architecture for Object Tracking in the Internet of Things,” IEEE Access, DOI: 10.1109/ACCESS.2018.2875734, Oct. 2018.

（11） J. Fung and M. Steve, ”Computer vision signal processing on graphics processing units,” 2004 IEEE International Conference on Acoustics, Speech, and Signal Processing, Vol. 5, pp.93-96, 2004.

（12） T. Sterling, M. Anderson and M. Brodowicz, ”High performance computing : modern systems and practices,” Cambridge, MA : Morgan Kaufmann, ISBN 9780124202153, 2018.

（13） J. E. Stone, D. Gohara and G. Shi, ”OpenCL: A parallel programming standard for heterogeneous computing systems,” Computing in science & engineering, Vol.12, No.3, 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, ”Study and Evaluation of Automatic GPU Offloading Method from Various Language Applications,” International Journal of Parallel, Emergent and Distributed Systems, Taylor and Francis, DOI: 10.1080/17445760.2021.1971666, Sep. 2021.

（16） Y. Yamato, ”Automatic Offloading Method of Loop Statements of Software to FPGA,” International Journal of Parallel, Emergent and Distributed Systems, Taylor & Francis, DOI: 10.1080/17445760.2021.1916020, Apr. 2021.

（17） Y. Yamato, ”Improvement Proposal of Automatic GPU Offloading Technology,” The 8th International Conference on Information and Education Technology (ICIET 2020), pp.242-246, Mar. 2020.

（18） Y. Yamato, ”Proposal and Evaluation of Adjusting Resource Amount for Automatically Offloaded Applications,” Cogent Engineering, Taylor and Francis, Vol.9, Issue 1, DOI: 10.1080/23311916.2022.2085467, June 2022.

（19） Y. Yamato, ”Study and Evaluation of Deployment Reconfiguration during Operation of Environment-adaptive Applications,” IPSJ Journal, Vol.63, No.12, pp.1840-1845, 2022.

（20） Y. Yamato, ”Study and Evaluation of FPGA Reconfiguration during Service Operation for Environment-Adaptive Software,” International Journal of Parallel, Emergent and Distributed Systems, Taylor & Francis, DOI: 10.1080/17445760.2023.2242639, Aug. 2023.

（21） Y. Yamato, ”Evaluation of GPU Logic Reconfiguration after Service Start,” ICIET 2023, pp.551-556, 2023.

（22） Y. Yamato, ”Study and Evaluation of Improved Automatic GPU Offloading Method,” International Journal of Parallel, Emergent and Distributed Systems, Taylor & Francis, DOI: 10.1080/17445760.2021.1941010, June 2021.

（23） S. Wienke, P. Springer, C. Terboven and D. an Mey, ”OpenACC-first experiences with real-world applications,” Euro-Par 2012 Parallel Processing, pp.859-870, 2012.

（24） K. Shirahata, H. Sato and S. Matsuoka, ”Hybrid Map Task Scheduling for GPU-Based Heterogeneous Clusters,”IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom), pp.733-740, Dec. 2010.

（25） GLPK website, https://sites.google.com/site/nssvdabb/glpk

（26） Time domain finite impulse response filter web site, http://www.omgwiki.org/hpec/files/hpec challenge/tdfir.html

（27） MRI-Q website, http://impact.crhc.illinois.edu/parboil/

（28） NAS.FT website, https://www.nas.nasa.gov/publications/npb.html

（29） Himeno benchmark web site, http://accc.riken.jp/en/supercom/

（30） NVIDIA vGPU software web site, https://docs.nvidia.com/grid/index.html

（31） M. Wolfe, ”Implementing the PGI accelerator model,” ACM the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, pp.43-50, 2010.

（32） Y. Tomatsu, T. Hiroyasu, M. Yoshimi and M. Miki, ”gPot: Intelligent Compiler for GPGPU using Combinatorial Optimization Techniques,” The 7th Joint Symposium between Doshisha University and Chonnam National University, Aug. 2010.

（33） Bobby R. Bruce and J. Petke, ”Towards automatic generation and insertion of OpenACC directives,” RN, 18.04: 04. 2018.

（34） J. Chen, B. Joo, W. Watson III and R. Edwards, ”Automatic offloading C++ expression templates to CUDA enabled GPUs,” 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, pp.2359-2368, May 2012.

（35） C. Bertolli, S. F. Antao, G. T. Bercea, A. C. Jacob, A. E. Eichenberger, T. Chen, Z. Sura, H. Sung, G. Rokos, D. Appelhans and K. O’Brien, ”Integrating GPU support for OpenMP offloading directives into Clang,” ACM Second Workshop on the LLVM Compiler Infrastructure in HPC (LLVM’15), Nov. 2015.

（36） S. Lee, S.J. Min and R. Eigenmann, ”OpenMP to GPGPU: a compiler framework for automatic translation and optimization,” 14th ACM SIGPLAN symposium on Principles and practice of parallel programming (PPoPP’09), 2009.

（37） Cheng Liu, Ho-Cheung Ng and Hayden Kwok-Hay So, ”Automatic nested loop acceleration on fpgas using soft CGRA overlay,” Second International Workshop on FPGAs for Software Programmers (FSP 2015), 2015.

（38） C. Alias, A. Darte and A. Plesco, ”Optimizing remote accesses for offloaded kernels: Application to high-level synthesis for FPGA,” 2013 Design, Automation and Test in Europe (DATE), pp.575-580, Mar. 2013.

（39） L. Sommer, J. Korinth and A. Koch, ”OpenMP device offloading to FPGA accelerators,” 2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP 2017), pp.201-205, July 2017.

（40） A. Putnam, D. Bennett, E. Dellinger, J. Mason, P. Sundararajan and S. Eggers, ”CHiMPS: A C-level compilation flow for hybrid CPU-FPGA architectures,” IEEE 2008 International Conference on Field Programmable Logic and Applications, pp.173-178, Sep. 2008.

（41） SYCL web site, https://www.khronos.org/sycl/

（42） DPC++ web site, https://www.intel.com/content/www/us/en/developer/tools/oneapi/dpc-library.html#gs.flx6xq

（43） C. C. Wang, Y. D. Lin, J. J. Wu, P. C. Lin and R. H. Hwang, ”Toward optimal resource allocation of virtualized network functions for hierarchical datacenters,” IEEE Transactions on Network and Service Management, Vol.15, No.4, pp.1532-1544, 2018.