アプリケーション自動オフロードにおけるリソース量設定の検討

2020年11月28日

NTTネットワークサービスシステム研究所　特別研究員
山登庸次

目次

NTT

1 はじめに

NTT

- 近年、ムーアの法則終焉が近いと言われており、それも踏まえて、GPUやFPGAといったヘテロなハードウェアの活用が増えている。
- GPUやFPGAには、CUDA、OpenCLと言ったスキルが必要で障壁高い。
- これらを容易に活用するため、プログラマーは処理したいロジックだけ書き、配置環境に合わせて、ソフトウェアが適応して動作が求められる。
- Javaは、環境適応を実現したが、性能は考慮されていなかった。
- 私は、一度書いたソフトウェアを、配置先環境でコード変換等を自動で行う事で、高性能動作させる、環境適応ソフトウェアを提案している。
- 本稿はその中で、オフロードした際をコストパフォーマンス高く動作させるため、CPUとデバイスのリソース量を適切化する手法を検討する。

2 既存技術

NTT

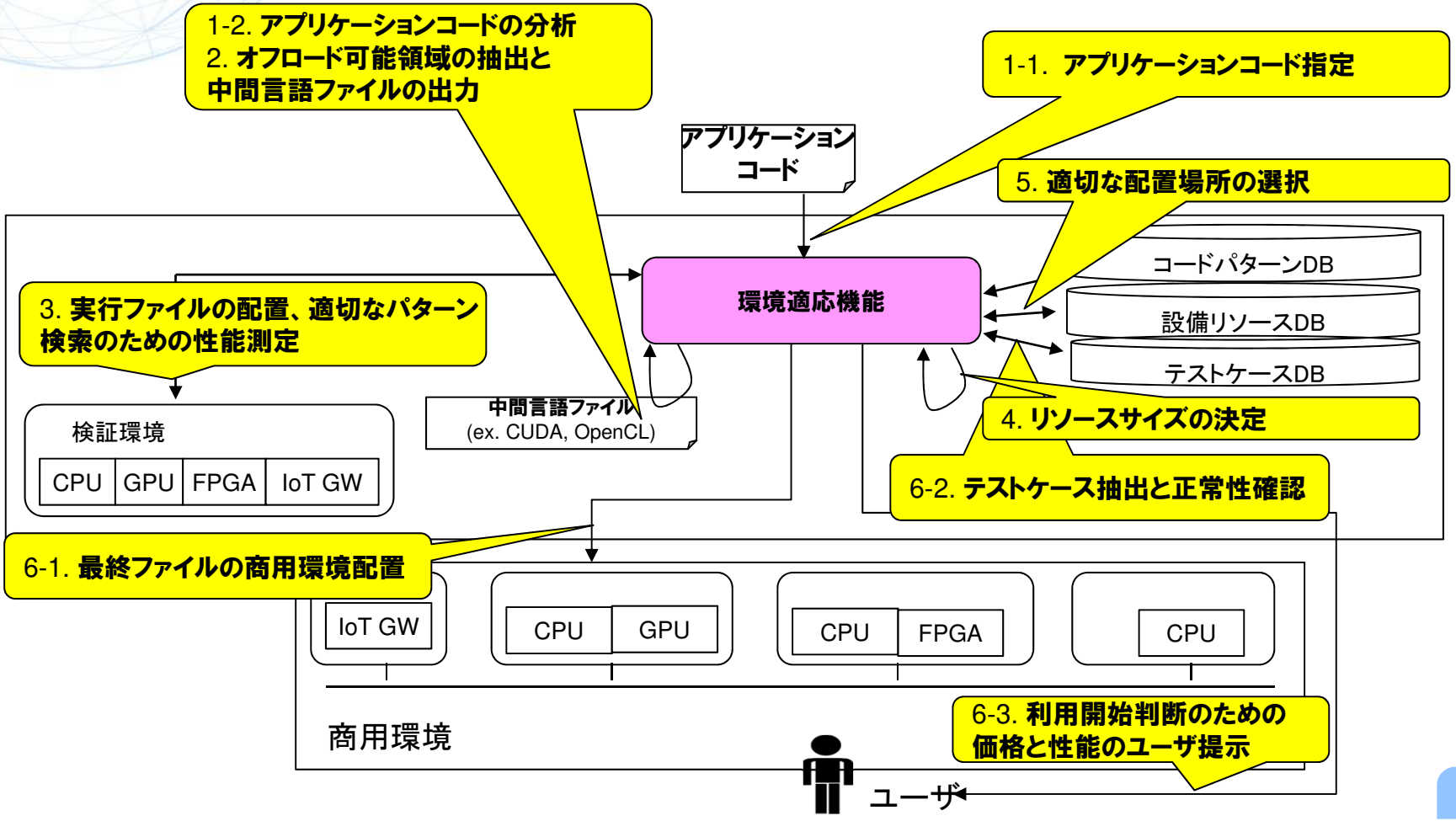
- Javaは、移行した先の性能チューニングやデバッグ等の稼働は小さくないことが課題であった（Write Once, Debug Everywhere）。

- GPGPU向けにCUDAが、ヘテロハード向け仕様にOpenCLがある。指示行仕様としてOpenACC、その解釈にPGIコンパイラがある。

- これらにより、GPU、FPGA処理は可能になっても、高性能化は難しい。
  - 自動並列化機能コンパイラは、ループ文等の並列可能部を抽出するが、メモリ間のやり取りオーバヘッドのため性能が出ないことも多い。
  - 並列処理箇所の試行錯誤を自動化する研究もあるが、高速化度が不十分との課題もある。
  - オフロードする際に言語の変換等の研究が主流で、オフロード後のデバイスのリソース量設定やコストパフォーマンスの検討はされていない。

2　環境適応ソフトウェアの処理フロー

- Step1-3で変換、4で量、5で場所を決め、6で検証配置。
- 運用中に、性能をモニタして、定期的にStep1-5を試行し、再構成が有効な場合に、7として再構成を提案。



2 本稿の目的とContribution

目的

- 一度書いたソフトウェアにて、GPU、FPGA、メニーコアCPU等のデバイスをコストパフォーマンス高く活用できるように、コード変換等を自動で行い、更に、リソース量を自動調整する。

Contribution

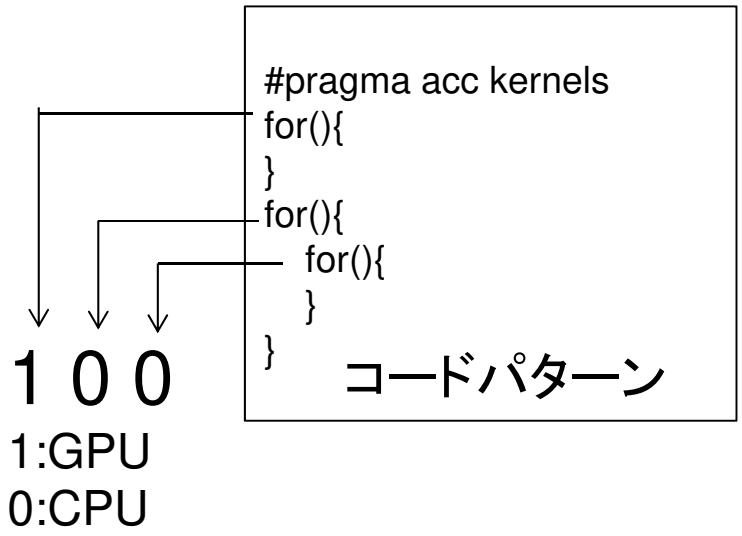
- CPUとオフロードデバイスのリソース比を適切にするための手法を提案する
- リソース比に基づいて、コストパフォーマンス高く利用するための、リソース量決定とその検証について提案する

- （既存アプリで、提案手法の有効性を実機評価する）　実装中

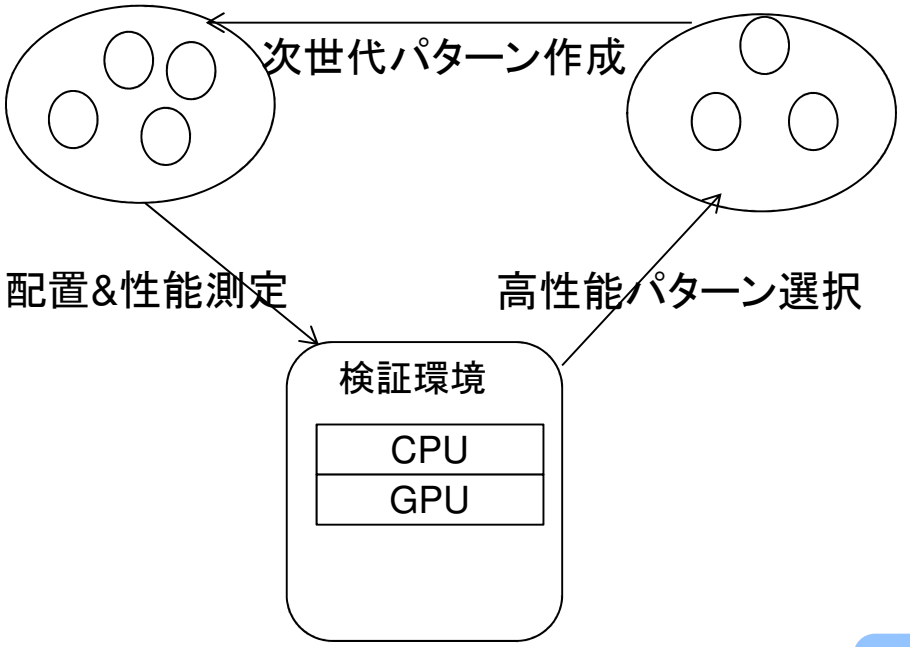
3　既存自動オフロード手法（ループ文のGPU）

- 性能はコンパイラ等が予測することは難しい現状を踏まえ、ループ文オフロードパターンを検証環境で繰返し性能測定し、進化的計算で高速化していく方式提案。

- ループ文に対しGPU処理指示句 ¥pragma acc kernels等で並列化指定し、実機性能測定を行い、高速なパターンを高適合度として、適切なループ文を組み換え抽出。
- GPU処理指示句誤りは、結果が誤る可能性があるため、元プログラムとの結果差分をとり、結果誤り時は、そのパターンの適応度を0にして後に残らない様にする

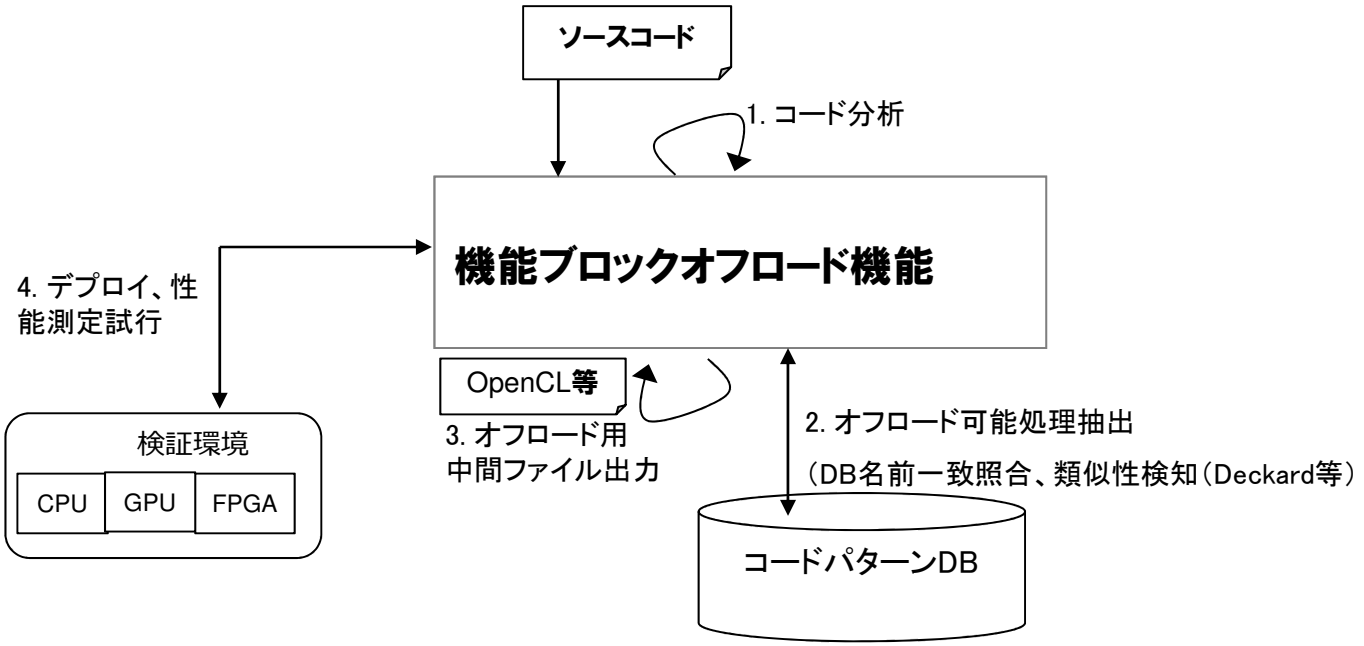


C言語ならOpenACCでGPU処理指示



3　既存自動オフロード手法（機能ブロック）

- コードの分析時に、コードに含まれるライブラリ呼び出しや機能処理を分析。
- コードパターンDB照合、類似性検知により、GPUにオフロードできる処理を発見。
  - ライブラリの名前一致に加え、abstract syntax tree類似性検知により、置換可能機能ブロックを増加。
  - ソフト維持に用いるソフト工学ツールだが、行単位、字句単位、フィンガープリント利用等手法があるが、機能ブロックの類似性を見る観点では、変更があっても良いabstract syntax treeが適当と考えた。
- オフロードできる処理について、GPU向けライブラリ等に置換。



3　リソース適性化の背景

NTT

関連した取り組み

- CPU仮想化だけでなく、GPU等でも仮想化の動きが出ている
  - XenはGPUボードの仮想化を一部行う
  - FPGAも未定義のGateを別用途に使う等の利用率向上が進んでいる
- 処理をオフロードしてもバランス悪いと全体性能が出ない
  - 100秒をGPUで10秒にしても、他の1000秒がCPUでは全体性能悪い
- MapReduceに特化して、CPUとGPUでの性能向上研究が有
  - K. Shirahata, et al., "Hybrid Map Task Scheduling for GPU-Based Heterogeneous Clusters,"IEEE CloudCom, 2010.
  - MapとReduceのタスクを、CPUとGPUで同じ処理時間になるようにタスクを配分することで全体の性能向上を図っている

3　リソース比適性化

NTT

提案方式

- MapReduce以外もCPUとオフロードデバイス処理時間が同オーダーにすることで効率化を図る。
  - 自動オフロードでは、コード変換の際に、既に検証環境での性能測定結果があるので、その結果を用いてCPUとオフロードデバイスのリソース比を定める
- 検証環境でのCPUとオフロードデバイスの処理時間の逆数に対して、適正リソース比を設定する。
  - 検証環境でのテストケース処理時間が、CPU処理：10秒、GPU処理：5秒の場合は、リソース比は、CPU：GPU＝2:1となる。
  - 処理時間が10倍以上差分がある場合は、リソース比を10倍以上にするのはむしろコストパフォーマンス悪化につながるので、5:1等のリソース比を上限にする。

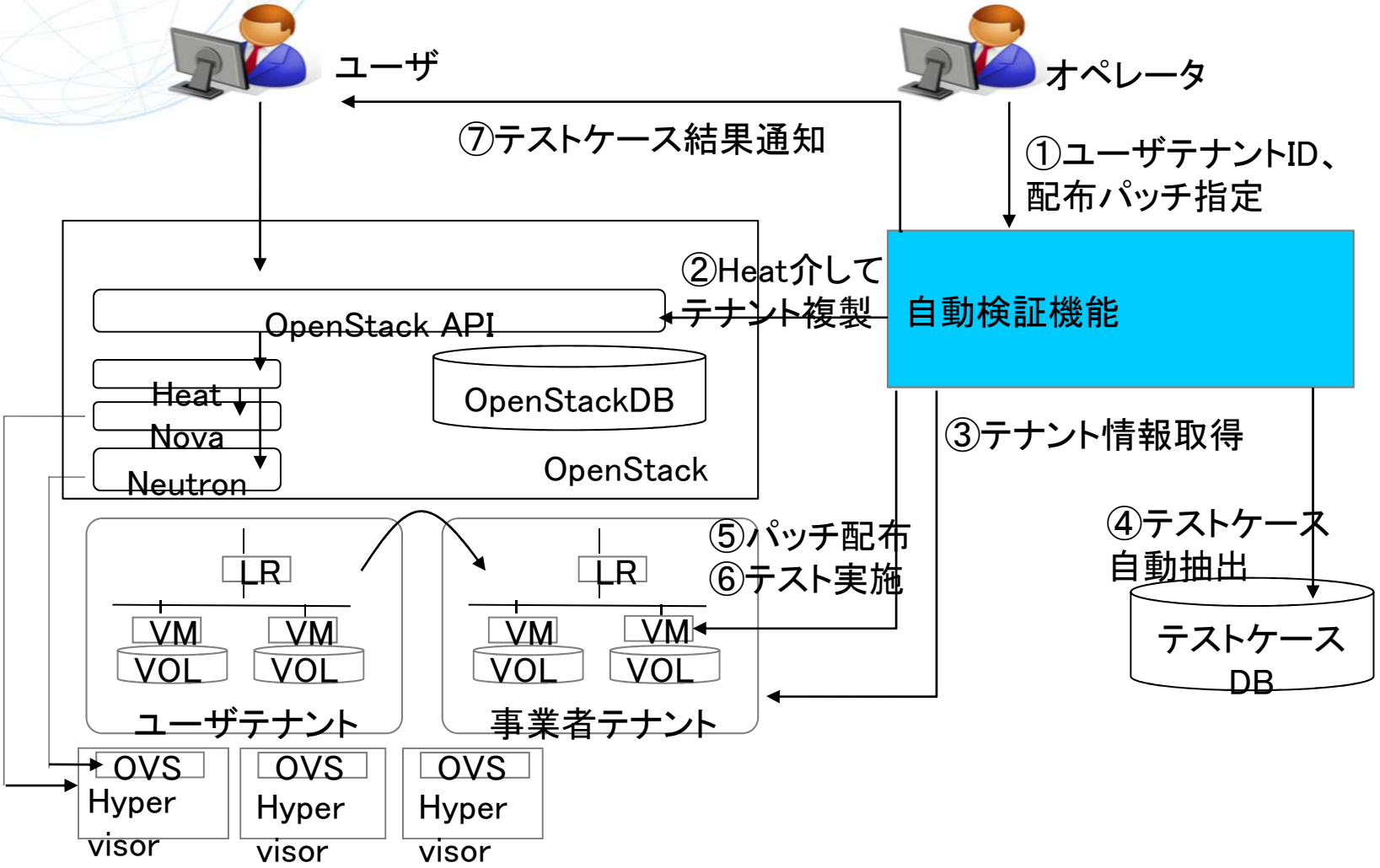
3　リソース量決定

提案方式

- ユーザが指定したコスト条件を満たすように、リソース比は出来るだけキープして、リソース量を決定する。
  - CPU1VMは1000円/月、GPUは4000円/月、リソース比は2:1が適切で、ユーザは月10000円以内の予算だった場合には、CPUは2、GPUは1を確保して商用環境に配置
  - ユーザは月5000円以内の予算だった場合には、リソース比はキープできないが、CPUは1、GPUは1を確保して配置
- 商用環境にリソースを確保して配置した後は、ユーザ利用前に性能、コストを確認するため、自動検証が行われる。

（3　自動検証）

- 性能、回帰テストを自動実施し結果をユーザ提示し開始判断。
  - Y. Yamato, “Automatic verification technology of software patches for user virtual environments on IaaS cloud,” Journal of Cloud Computing, Springer, Feb. 2015.　等利用



4 実装　（1/2）

NTT

- 提案方式を以下のツールを用いてC言語アプリに対し実装中
  - ツール実装
    - Perl　遺伝的アルゴリズム処理
    - Python　構文解析他
  - GPU：NVIDIA GeForce RTX 2080 Ti
    - PGIコンパイラ 19.10
  - CPU仮想化及びGPU仮想化
    - Citrix Xen Server 4
  - 構文解析
    - Clang 6.0
  - 類似性検出
    - Deckard ver2.0
  - コードパターンDB
    - MySQL 8.0

4 実装 （2/2）

NTT

- 動作概要
  - アプリケーションの利用依頼があると、構文解析ライブラリを用いてコード解析。
  - 機能ブロックオフロード、ループ文オフロードの順に試行を行う。
  - 機能ブロックオフロードが可能だった場合は、ループ文オフロードはオフロード可能機能ブロック部分を抜いたコードに対して試行。
  - 性能測定の結果、最高性能のパターンを解とする。
  - 解のパターンに対する性能測定結果を元に、CPUとGPUの適切なリソース比を算出。
  - 適切なリソース比と、ユーザ依頼の条件から、リソース量を定める。

5 まとめ

- 環境に適応させ、GPU、FPGA、メニーコアCPU等を適切に活用し、高性能に動作させるための環境適応ソフトウェアを提案している。

- 言語変換等した後、オフロードする際のリソース量決定の方式を提案
  - リソース比適切化
    - 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、リソース比を決定
    - VM数の大幅増加を防ぐため、リソース比には上限を設ける
  - リソース量決定と自動検証
    - ユーザのコスト要求を満たす中で、リソース比を出来るだけキープしてリソース量決定
    - 商用環境でリソースを確保して、自動検証後、性能とコストをユーザに提示

- 今後は検討手法を実装し、複数のコスト条件の中で、市中アプリケーション本の適切なオフロードを確認し、有効性を検証する。