2013 年 11 月 18 日 情報システム基盤学合同輪講発表資料

libWater: Heterogeneous Distributed Computing Made Easy

著者： Ivan Grasso, Simone Pellegrini, Biagio Cosenza, and Thomas Fahringer
出典： *27th ACM Int'l Conference on Supercomputing (ICS 2013), pp.161-172, June 2013.*
発表者： 高性能コンピューティング学講座 本多・近藤研究室　1353017 竹本拓未

1　はじめに

近年の HPC システムの多くが、CPU や GPU などの異なった種類の演算デバイスを混在させたヘテロジニアス構成のコンピュータノードをネットワークを介して接続したマルチノードの分散システムである。このようなマルチノードシステムを活用したアプリケーションを開発する場合、特別なプログラミング言語の習得に加えて、デバイスの個々の特性を活かし、ノード数が増加した際にも性能向上率を低下させない設計を行う必要がある。

そのため本研究では、分散型ヘテロジニアスシステムを対象としたプログラミングのための、生産性とスケーラビリティを兼ね備えたライブラリベースの手法を提案する。

2　従来手法の問題点

ヘテロジニアスシステムを対象としたプログラミングの従来手法として、共通フレームワークである OpenCL を用いた手法が標準化されている。OpenCL を用いることで、シングルノード上でのヘテロジニアスコンピューティングが可能になる。一方、マルチノードを対象とする場合には、OpenCL とは別に、ノード間通信を行うための機構 (e.g. MPI) を用いなければならない。

また、多種多様なデバイスが異なるノード上に分散して存在する場合には、適切なデバイスを探し、指定することはプログラマにとって困難である。

これらの理由から、マルチノードシステムでは、非分散システムに比べて生産性とスケーラビリティが著しく低下してしまうという問題がある。

3　libWater

3.1　libWater の概要

libWater は、マルチノードシステムでのヘテロジニアスコンピューティングを実現するための C/C++用ライブラリである。このライブラリは、オリジナルの OpenCL の概念を踏襲したシンプルなインターフェースを基本方針として設計されている。

また、リモートノード上のデバイスを使用する際には、MPI による通信ルーチンを自動的に適用することで、プログラマ自身がノード間通信を考慮する必要を取り除いている。

これらの仕様から、OpenCL のプログラミングパラダイムから逸れることなく、複数ノード上に搭載されている多種多様なデバイスをローカルなデバイスとして演算に使用することを可能とし、生産性の向上を図っている。

3.2　デバイス問い合わせ言語 DQL

デバイスの選択を簡単にするために、デバイス問い合わせ言語 DQL(Device Query Language) を利用することとしている。DQL は、SQL に似たクエリ言語であり、デバイス管理関数を呼び出す際に引数として渡すことで、指定する条件を満たすデバイスを簡単に選択することができる。図 1 に DQL の構文例を示す。

```
SELECT ALL
  WHERE (type = gpu AND vendor = nvidia)
```

(a)

```
SELECT POS 1
  FROM NODE 0
  WHERE global_memory > 1024MB
```

(b)

図 1: DQL の構文例

3.3　ランタイムシステム

libWater のランタイムシステムでは各ノード内において、コマンドスケジューラである WTR スケジューラが libWater コマンドを解釈し、対応する OpenCL コマンドを対象のデバイスで実行させる。また、ノード間通信が必要な場合には MPI による通信ルーチンを自動的に適用する。

図 2 はランタイムシステムの概要図を表している。また、具体的な手順を以下に示す。

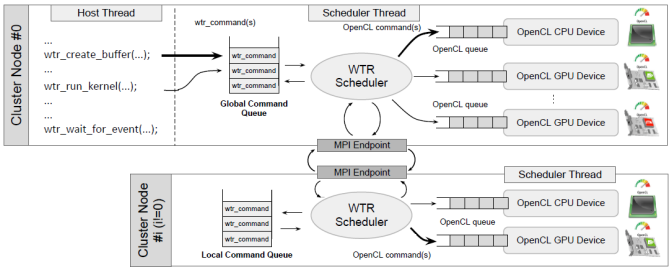

図 2: libWater のランタイムシステム

1. ホストプログラムから libWater で定義された関数が呼び出され、対応するコマンドが専用のキューに投入される。
2. 投入されたコマンドから、WTR スケジューラが演算を行うデバイスを解釈する。
3. 対象デバイスが同一ノード上の場合、対象デバイス用の OpenCL キューに、対応する OpenCL コマンドを投入する。リモートノード上の場合には、MPI を用いて該当ノードにコマンドが転送される。
4. 転送されたコマンドは、そのノード内のコマンドキューに投入される。

WTR スケジューラは、継続的にコマンドキューからコマンドを取り出し、上記 2-4 の手順を行う。

3.4 コマンドスケジューリング機構

コマンドがリモートノードに転送される際に、通信の到着順序によってはプログラムの整合性に問題が生じる。この問題を解決するために libWater は、コマンド間の依存関係を保証するためのコマンドスケジューリング機構を持つ。

プログラマは libWater の関数を呼び出す際、OpenCL の event 機構を用いて依存関係を指定する。スケジューラはコマンドが持つ event オブジェクトを参照し、先行制約を満たすコマンドのみを実行するようにする。

3.5 DCR 最適化

マルチノードシステムではノード間データ転送が時間的オーバーヘッドになるため、ノード数が増加してもリニアスピードアップにはならない。そこで libWater では、データ転送に使用される Point to Point 通信を、集団通信に置き換える DCR（Dynamic Collective Replacement）最適化手法を適用することで、実行時間の低減を図る。

各通信パターンが MPI の集団通信命令に置換可能であるかどうかを判別する手法には、既存研究 [1] などで提案された手法を用いる。

DCR 最適化の例を図 3 に示す。

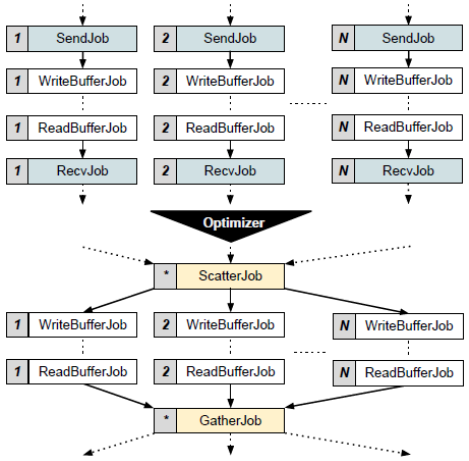

図 3: DCR 最適化の例

図 3 では、N 台のデバイスにデータの分割転送を行う際に、MPI の集団通信関数である Scatter/Gather 関数を使用する例である。

4 評価

6 種類のベンチマークに libWater を適用し、コード内の行数である LOC とノード数を増加させた際の実行時間のスケーリングから、生産性とスケーラビリティの評価を行った。

表 1 は、libWater 適用前と適用後の LOC を計測したものである。表 1 から、libWater を使用した際には平均で約 50%の LOC が削減され、高い生産性が得られていることが示された。

表 1: LOC

Application	OpenCL LOC	libWater LOC
PerlinNoise	412	301
NBody	450	324
kNN	234	101
Floyd	222	113
MatrixMul	219	104
LinReg	298	149

次に、ヘテロジニアス構成である Vienna Supercomputing Cluster 2 上でノード数を 1 から 64 まで変化させた際の実行時間を図 4 に示す。図 4(a) は最良ケースであり、図 4(f) が最悪ケースである。両ケースとも DCR 最適化を行った場合には、ほとんどリニアにスケールできていると考えられる。

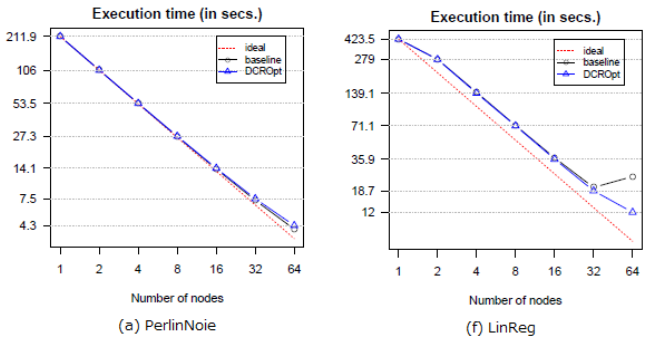

図 4: libWater を用いた際の実行時間のスケーリング

5 おわりに

本研究では、分散型ヘテロジニアスシステムのための libWater を設計、実装した。これによって、高い生産性とスケーラビリティを兼ね備えた分散型ヘテロジニアスコンピューティングが行えることを示した。

参考文献

[1] J. Bruck, C.-T. Ho, S. Kipnis, and D.Weathersby. Efficient Algorithms for All-to-All Communications in Multi-Port Message-Passing Systems. In SPAA, pages 298-309, 1994.