ホモジニアス・コードを使用したヘテロジニアス・クラスタリング: コードの再構成が不要なインテル® Xeon Phi™ コプロセッサーによる MPI アプリケーションの高速化

Colfax International
Vadim Karpusenko
Andrey Vladimirov

2013 年 10 月 17 日

概要

このホワイトペーパーでは、汎用プロセッサー、およびインテル® Xeon Phi™ コプロセッサーと呼ばれるコンピューティング・アクセラレーターによる 2 種類の計算デバイスを使用して、ヘテロジニアス・クラスター環境で分散並列アプリケーションを実行するケースを紹介します。

汎用 GPU (GPGPU) と異なり、インテル® Xeon Phi™ コプロセッサーはネイティブ・アプリケーションを実行することができます。このモードでは、アプリケーションはコプロセッサーのオペレーティング・システムで実行されます。CPU 側でホストプロセスを実行し、アクセラレーター (コプロセッサー) へデータをオフロードする必要はありません。そのため、MPI アプリケーションでは、MPI プロセスを直接コプロセッサー上で実行することができます。その場合、コプロセッサーは、MPI ランク、ピア・ツー・ピア通信、ネットワーク共有ファイルシステムへのアクセスを備えた、独立した計算ノードのように動作します。そのため、プロセッサーとコプロセッサーから成るヘテロジニアス・システムでは、アプリケーションでデータオフロードをインストルメントする必要はありません。つまり、CPU のみのクラスター用に設計された MPI アプリケーションを、コードを変更することなく、コプロセッサーを含むクラスターで使用することができます。

ここでは、効率良いヘテロジニアス・システムに必要な可搬性のあるコード設計、ロードバランス、システム構成 (ネットワークおよび MPI) について取り上げます。モンテカルロ・シミュレーションによりエイジアンオプション価格評価を行うサンプル・アプリケーションを例に、CPU のみで実行した場合と、ヘテロジニアス・クラスター構成で実行した場合のパフォーマンスを示します。

目次

Colfax International (http://www.colfax-intl.com/) 社は、ワークステーション、クラスター、ストレージ、パーソナル・スーパーコンピューティング向けの革新的かつ専門的なソリューションを導くリーディング・プロバイダーです。他社では得られない、ニーズに応じてカスタマイズされた、広範なハイパフォーマンス・コンピューティング・ソリューションを提供します。すぐに利用可能な Colfax の HPC ソリューションは、価格/パフォーマンスの点で非常に優れており、IT の柔軟性を高め、より短期間でビジネスと研究の成果をもたらします。Colfax International 社の広範な顧客ベースには、Fortune 1000 社にランキングされている企業、教育機関、政府機関が含まれています。Colfax International 社は、1987 年に創立された非公開企業で、本社はカリフォルニア州サニーベールにあります。

1. はじめに: オフロード・プログラミングとネイティブ・プログラミング

インテル® メニー・インテグレーテッド・コア (インテル® MIC) アーキテクチャー・ベースのインテル® Xeon Phi™ コプロセッサーは、汎用 GPU (GPGPU) に似たコンピューティング・アクセラレーターです。

i) どちらのアクセラレーターも PCIe バスを介して CPU ベースのシステムに接続します。
ii) どちらも汎用プロセッサーよりも優れたパフォーマンスと消費電力を達成するには、並列性とタスク並列性を必要とします。
iii) どちらもオフロード・プログラミング・モデルをサポートします。

オフロード・プログラミング・モデルでは、アプリケーションはホストシステム (CPU) で起動され、データの初期化もホストで行われます。そして、PCIe バスを介して、データと実行コードの一部がデバイスにプッシュ ("オフロード") され、デバイス側で処理が実行されます。処理が完了すると、結果がホストに送られます [1]。Nvidia* CUDA フレームワークでは、専用の関数とストリームによりオフロードが行われます。インテル® Xeon Phi™ コプロセッサーでは、プログラマーがコンパイラー・プラグマを用いてオフロードを指定します。どちらのアクセラレーターでも、OpenCL* によるオフロードが可能です。汎用 CPU 向けアプリケーションにオフロードをインストルメントするには、オフロードする処理を決め、オフロード用のデータ構造を準備し、オフロード宣言子によりコードを拡張します。オフロード・プログラミングは、スタンドアロン・マシンとクラスターのどちらでも利用できます。各マシンで 1 つ以上の MPI プロセスが起動され、それぞれがオフロードを実行します (図 1 を参照)。

ただし、インテル® Xeon Phi™ コプロセッサーは 2 つの点において GPGPU と大きく異なります。

i) インテル® Xeon Phi™ コプロセッサーには uOS[2] と呼ばれる Linux* オペレーティング・システムが搭載されています。そのため、IP アドレスと仮想ファイルシステムを持ち、SSH、NFS、MPI などの主要な HPC サービスを実行できます。
ii) インテル® Xeon Phi™ コプロセッサーは、ネイティブ・プログラミング・モデルもサポートしています。このモードでは、アプリケーションはデバイスで直接起動され、すべてのデータの初期化と I/O もデバイス側で行われます。

ネイティブ・プログラミングは、GPGPU にはない方法で分散アプリケーションの設計が可能です。

- MPI を利用する分散アプリケーションをコプロセッサーでのみ実行し、ホスト CPU でほかのタスクを処理することができます。
- あるいは、MPI プロセスをホスト CPU とコプロセッサーの両方に配置し、ヘテロジニアス・プラットフォームで実行することもできます (図 2 を参照)。

2 つ目のケースでは、オフロード構造をインストルメントしなくてもインテル® Xeon Phi™ コプロセッサーを使用できるため、特に既存の MPI アプリケーションにとっては魅力的と言えるでしょう。インテル® Xeon Phi™ コプロセッサーでネイティブに実行するプロセスは、ホストのプロセッサーと同様に、自動的に初期化し、通信を開始します。

当然のことながら、プログラミングを必要としないこのような移植では、必ずしも "そのまま" で優れたパフォーマンスが得られるとは限りません。このホワイトペーパーでは、ヘテロジニアス・クラスターで効率良く実行できるアプリケーションの種類と、その実装の必要条件について説明します。また、インテル® Xeon Phi™ コプロセッサーを用いたヘテロジニアス実行を可能にするシステム構成についても述べます。その概念を示す例として、C 言語でモンテカルロ法によるエイジアンオプション価格評価を実装します。このコードは CPU ベースのクラスター向けに記述されていますが、コードを変更することなく、インテル® Xeon Phi™ コプロセッサーを含むヘテロジニアス・クラスターで実行することができます。つまり、このアプリケーションには、インテル® Xeon Phi™ コプロセッサー向けのコードは含まれていません。それにもかかわらず、ハードウェア構成にコプロセッサーを追加することで、大幅なパフォーマンス向上が得られます。

2. エイジアンオプション価格評価

セクション 3 では、MPI を利用する分散アプリケーションの実装に触れます。このアプリケーションは、モンテカルロ・シミュレーションによりエイジアンオプション価格評価を行います。

[1] これ以降、"ホスト" は CPU ベースのプラットフォームに搭載されたオペレーティング・システムまたは CPU プラットフォーム自体を指し、"アクセラレーター" や "デバイス" は GPGPU またはコプロセッサーを指します。
[2] uOS は、μOS (マイクロオペレーティング・システム) の一般的に使用されている表記法です。

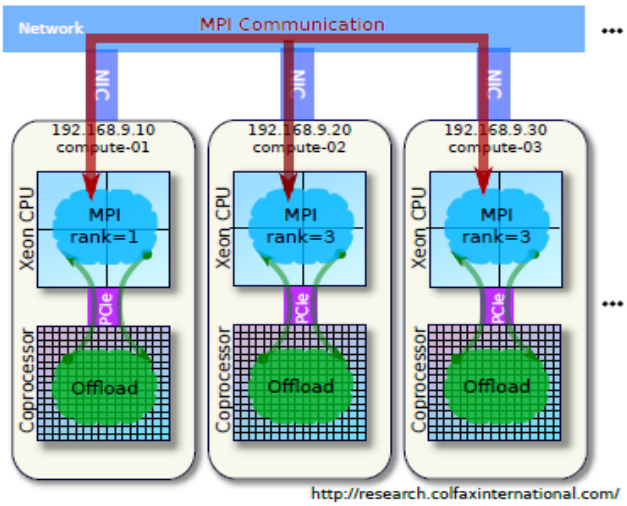

図 1: コプロセッサーへオフロードする従来のクラスターの MPI 通信。ホストは、リモートホストとの通信に MPI を利用し、**ローカル・コプロセッサー**との通信にオフロードを利用します。

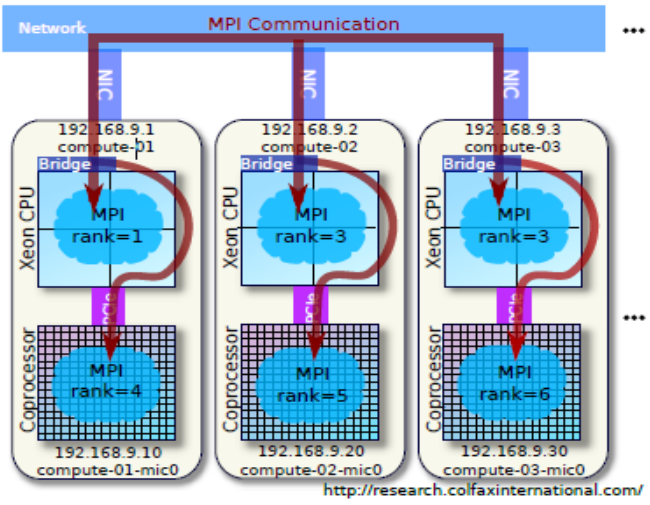

図 2: ヘテロジニアス・クラスターの MPI 通信。ホストは、リモートホストおよび**ローカル/リモート・コプロセッサー**との通信に MPI メッセージを利用します。

このセクションでは、シミュレーションを行う金融問題と数学モデルを説明します。HPC に関する部分にのみ興味がある方は、このセクションをスキップし、セクション 3 へ進んでもかまいません。

オプションとは、買い手 ("受益者") が株式市場資産を将来の特定日 ("オプション満期日") に、約定時に合意した行使価格で売り手 ("設定者") から買う権利 ("コールオプション")、または売り手 ("設定者") に売る権利 ("プットオプション") のことです。先物契約とは異なり、オプションの場合、受益者は権利を行使するかどうかを選択できます。通常、この選択は、オプション満期日の当該資産の市場価格に基づいて行われます。例えば、コールオプションの満期日に当該資産の市場価格が行使価格よりも高い場合、受益者は当該資産を設定者から買い付け、市場に売ることで利益 ("ペイオフ") を得られます。受益者が権利を行使しない場合であっても、設定者はオプション料を得られます。エイジアンオプションと呼ばれるオプションは、あらかじめ決められた間隔でサンプリングされた資産の平均価格 (算術平均または幾何平均) に基づいてペイオフが計算されます。

そのため、市場変動と短期市場操作のリスクを軽減できます。

リスク分析とエイジアンオプションの価格評価には、モンテカルロ・シミュレーションを利用できます。この方法では、資産の変動性に関する情報を基に、さまざまな確率変数による資産価格のシミュレーションを行います。

変数 $S(t)$ は、オプションの対象となる資産の価格で、確率方程式に従って時間とともに変化すると仮定されます。

$$dS(t) = \mu S(t)dt + \sigma S(t)dB(t) \qquad (1)$$

この方程式で、μ は資産のドリフト、σ はオプションの変動性、$B(t)$ は標準的なブラウン運動を示します。

この確率微分方程式の解は、次のように表すことができます。

$$S(t_i) = S(t_{i-1})e^{(\mu-\sigma^2/2)\Delta t+\sigma\sqrt{\Delta t}\,\mathcal{X}} \qquad (2)$$

ここで、X は平均ゼロ、標準偏差ゼロの正規分布乱数で、$\Delta t = t_i - t_{i-1}$ です。

当該資産のエイジアンオプションのペイオフを計算するには、満了日までの期間 T に対し N 間隔で資産価格の平均 (算術平均および幾何平均)を求めます。

$$\langle S \rangle_{\text{arithm}} = \frac{1}{N} \sum_{i=0}^{N-1} S(t_i) \quad (3)$$

$$\langle S \rangle_{\text{geom}} = \exp\left(\frac{1}{N} \sum_{i=0}^{N-1} \log S(t_i) \right) \quad (4)$$

ここでは、$t_i = T \times i/(N-1)$ です。行使価格 K に対する "コール" および "プット" オプションのペイオフは、次のようになります。

$$P_{\text{put}} = e^{-rT} \max\{0; K - \langle S \rangle\} \quad (5)$$

$$P_{\text{call}} = e^{-rT} \max\{0; \langle S \rangle - K\} \quad (6)$$

ここで、〈S〉は約定条件に応じて〈S_{arithm}〉または〈Sgeom〉となり、r はリスクフリー・レートです。

パラメーター・セット $\{S, K, \gamma, \mu, \nu, T, N\}$ と平均の算出方法 (算術平均または幾何平均) に対するエイジアンオプションのペイオフの数学的期待値は、モンテカルロ・シミュレーションにより計算できます。シミュレーションではランダムパスに大数 M を使用し、各ランダムパスで方程式 (2) に従って t=0 から t=T の範囲で確率変数に基づいてオプション価格を変動させ、方程式 (3) または (4) に従って N 時点でサンプリングした価格の平均を算出します。そして、これらの平均と方程式 (5) と (6) を使って、プットおよびコールオプションのペイオフを計算します。最後に、M ランダムパスのペイオフを平均し、数学的期待値のモンテカルロ推定を算出します。

3. 従来のクラスター向けの実装

他のモンテカルロ・シミュレーションと同様に、エイジアンオプション価格評価アプリケーションは並列性を備えています。並列化の第 1 候補として、パラメーター・セット $\{S, K, \gamma, \mu, \nu, T, N\}$ の処理が挙げられます。各パラメーター・セットは個別に処理することができるため、この実装では個々のパラメーター・セットを MPI プロセスに分散しています。並列化の第 2 候補は、各パラメーター・セットでシミュレーションされる資産変動の確率パスです。それぞれのパスは互いに独立しているため、この実装では個々のパスのシミュレーションをプロセッサー・コアと各コアの SIMD (Single Instruction Multiple Data) レーンに分散しています。SIMD レーンを利用する並列化は、ベクトル化と呼ばれます。

セクション 3.1 では、1 つの共有メモリー計算デバイスで 1 つのオプション・パラメーター・セットを分析する、C 言語のスレッド並列処理を紹介します。セクション 3.2 では、分散メモリーシステムの MPI プロセスにアプリケーションの処理を分配する方法を説明します。

3.1. スレッド並列性とベクトル並列性

リスト 1 は、この計算のパフォーマンス・クリティカルな部分です。ここでは、算術平均による "プット" オプションのみ掲載しています。完全なコードは [1] からダウンロードできます。このコードは、インテル® Xeon® プロセッサーのような SIMD 命令セットに対応したマルチコア・プロセッサー向けに最適化されています。

パフォーマンスに関してこの実装の重要な点は次のとおりです。

1. OpenMP* 構文によりランダムパスの計算をスレッド間で分配し、ワークロードを並列処理します (行 3 ～ 4 を参照)。スレッド間のロード・スケジュールは OpenMP* スケジューラーによって管理されます。デフォルトでは、利用可能な CPU のすべてのコアが計算に使用されます。そのため、計算ノードごとに 1 つのプロセスを開始する必要があります。
2. インデックス k のループ (特に行 13 のパフォーマンス・クリティカルなループ) がコンパイラーにより自動ベクトル化されると、各スレッドにおいてコアの SIMD レーンで複数のパスが同時に計算されます。インテル® C++ コンパイラーは自動ベクトル化をサポートしています。このデータ並列処理に使用される命令セットは、コードをコンパイルするプラットフォームに応じてコンパイル時に選択されます。
3. 行 8 の乱数生成は、インテル® マス・カーネル・ライブラリー (インテル® MKL) により行われます。この実装で使用されているメルセンヌツイスター・ベースの乱数ジェネレーターはベクトル化されています (つまり、SIMD 命令セットを利用しています)。各 OpenMP* スレッドは、プライベートの乱数ストリームを保持しています。

```
1  /* The i-loop is thread-parallel, i.e.,
2   distributed across the processor cores */
3  #pragma omp parallel for schedule(guided)\
4    reduction(+: payoff_arithm_put)
5  for (int i = 0; i < nPaths/vecSize; i++) {
6    for (int j = 1; j < nIntervals; j++) {
7  /* Intel MKL random number generator */
8      vsRngGaussian(
9        VSL_RNG_METHOD_GAUSSIAN_BOXMULLER,
10       stream, vec_size, rands, 0.0f, 1.0f);
11 /* The k-loop is data-parallel thanks to
12  automatic vectorization by the compiler */
13     for (int k = 0; k < vecSize; k++) [
14       spot_prices[k] *=
15         exp2f(drift + vol*rands[k]);
16       sumsm[k] += spot_prices[k];
17     }
18   }
19   for (int k = 0; k < vecSize; k++) {
20     arithm_mean_put[k] =
21       K - (sumsm[k] * recipIntervals);
22     if (arithm_mean_put[k] < 0.0f)
23       arithm_mean_put[k] = 0.0f;
24   }
25 /* Reduction across vector lanes and across
26  OpenMP threads is automatically
27  implemented by the compiler */
28   for (int k = 0; k < vecSize; k++)
29     payoff_arithm_put += arithm_mean_put[k]*
30       expf(-r*T)/(float)nPaths;
31 }
```

リスト 1: モンテカルロ法によるエイジアンオプションの価格評価

4. 計算中、スレッド間の通信は発生しません。ただし、計算終了時に、すべての SIMD レーンおよび OpenMP* スレッドにわたってランダムパスの結果がレデュースされます。この処理は、行 29 ～ 30 で、OpenMP* ライブラリーと自動ベクトル化によって行われます。

リスト 1 のコードの最適化はすべて汎用マルチコア・プロセッサーにより実行されます。このコードにインテル® Xeon Phi™ コプロセッサー専用のコードは含まれていません。ただし、セクション 4.4 で説明するように、同じコードを再コンパイルするだけでコプロセッサーでも効率良く実行できるようになります。そのため、リスト 1 のコードは "メニーコア対応"[3] と言っても差し支えないでしょう。

3.2. ボス・ワーカー・モデルによる動的ロードバランス

実際のアプリケーションでは、複数のオプション・パラメーターを評価する必要があります。アプリケーションをコンピューティング・クラスター全体に分散するため、ここでは、各計算ノードで 1 つまたはいくつかのオプション・パラメーター・セットが処理されるようにワークロードを分割します。

```
1  if (myRank == bossRank) {
2    int nR = 0; /* Number of processed tasks */
3    int iP = 0; /* Next task to assign */
4    while (nR < nPars) {
5
6   /* Wait for any worker to report for work
7      */ float buf[msgReportLength];
8      MPI_Recv(&buf, msgReportLength,
9        MPI_INT, MPI_ANY_SOURCE, msgReportTag,
10       MPI_COMM_WORLD, &status);
11     const int iW = status.MPI_SOURCE;
12
13     if (buf[0] > 0.0f) {
14   /* If worker reports with results of a
15      previous task, record these results */
16       nR++;
17       const int iR = floorf(buf[1]);
18       payoff_arithm_put [iR] = buf[2];
19     }
20
21     if (iP < nStrikes) {
22   /* Assign the next task iP to worker iW */
23       float buf[msgSchedLen] = {iP,
24         M[iP], N[iP], K[iP], S[iP], /*...*/};
25       MPI_Send((void*)&buf,
26         msgSchedLen, MPI_FLOAT, iW,
27         msgSchedTag, MPI_COMM_WORLD);
28       iP++;
29     }
30   }
31 }
```

リスト 2: ボスプロセスの実装。ボスプロセスがワーカーから要求を受け取り、作業を与えることで動的にロードバランスをとります。

一般に、各パラメーター・セットの処理時間はすべての計算ノードで同じになるとは限りません。実際、パラメーター・セットの計算時間は $M \times N$ に比例します。M は期待する精度を達成するのに必要なモンテカルロ・パスの数で、N は価格を平均する期間の数です。M と N の値はパラメーター・セットにより変わることがあります。ここでは、動的ロードバランスを実装することでこの問題に対処します。

[3] ここで、"マルチコア" は 2 つ以上のコアを持つ汎用プロセッサー (インテル® Xeon® プロセッサーなど) を指し、"メニーコア" はインテル® MIC アーキテクチャー、特にインテル® Xeon Phi™ コプロセッサーを指します。

このワーク・スケジュール方式は、インテル® Xeon Phi™ コプロセッサーを含むヘテロジニアス・クラスターでコプロセッサーを使用する場合に役立ちます (セクション 4 を参照)。

"ボス・ワーカー・モデル" は、比較的簡単で広く知られている動的ワーク・スケジュール方式です。このモデルでは、1 つのプロセス ("ボス") がほかのプロセス ("ワーカー") から要求を受け取り、作業を与えます。ボスは要求を受け取ると、そのワーカーに "作業" (ここではモンテカルロ・シミュレーションのオプション・パラメーター・セット) を送ります。ワーカーは作業が完了すると、ボスに結果を通知し、次の作業を要求します。ワーカーはアイドル状態になるとすぐに次の作業を与えられるため、このスケジュール方式は、作業コストやワーカーのパフォーマンスが均一でない場合でも、ワーカー全体にわたって負荷を分散できます。ただし、ボス・ワーカー・スケジュール方式には、セクション 5.2 で述べる、スケーラビリティーの制限がありますが、このテストでは問題ないでしょう。

エイジアンオプション価格評価アプリケーションのボスプロセスを処理する C コードをリスト 2 に示します。完全なコードは [1] からダウンロードできます。この実装で、ボスプロセスは MPI ランク 0 のシングルスレッド・プロセスで、ワーカーは MPI_WORLD コミュニケーターのすべての MPI プロセスです。

3.3. コンパイルと CPU のみの実行

テストでは、1 つのヘッドノードとインテル® Xeon® プロセッサーを搭載した 2 つの計算ノードから成るクラスター (図 3 を参照) でアプリケーションを実行しました。計算ノードは、NFS を介してヘッドノードから 2 つのディレクトリーをインポートします。

/opt/intel はインテル® MKL とインテル® MPI ライブラリーへのアクセスを提供し、/nfs は実行ファイルと必要なデータファイルを共有します。

図 3 に示すように、各計算ノードに 1 つのワーカーを配置し、ノード compute-01 にボスを配置します。リスト 3 は、テスト・クラスターでサンプル・アプリケーションをコンパイルし実行した結果です。

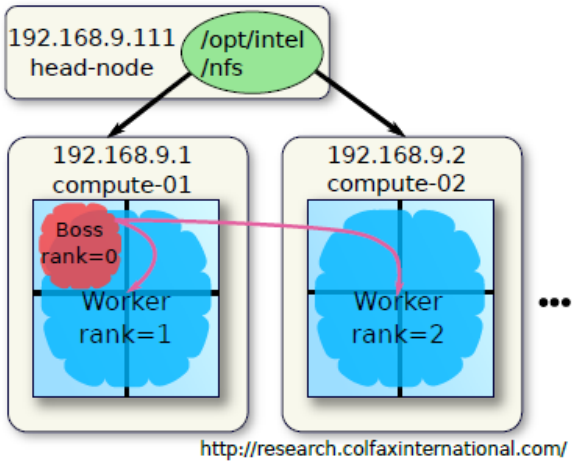

図 3: ボス・ワーカー・スケジュールを使用するエイジアンオプション価格評価アプリケーションのクラスター構成と MPI 実行設定。このホモジニアス・クラスターでは、CPU ベースの計算ノードのみ使用されます。

```
# Intel MPI environment
[colfax@head-node]# source \
> /opt/intel/impi/4.1.1/intel64/bin/mpivars.sh
# Viewing the cluster configuration
[colfax@head-node]# cat /etc/hosts | grep 192
192.168.9.1     compute-01
192.168.9.2     compute-02
[colfax@head-node]# cat /etc/exports
/opt/intel 192.168.9.0/24(rw,no_root_squash)
/nfs 192.168.9.0/24(rw,no_root_squash)

# Compiling the code, sharing with compute nodes
[colfax@head-node]# mpiicc -std=c99 -mkl \
>    -openmp -xAVX -o options options.c
[colfax@head-node]# cp -v options /nfs/options/
'options' -> '/nfs/options/options'

# Starting the MPI job
[colfax@head-node]# cat ./machines-CPUs
192.168.9.1 # Boss process on compute-01
192.168.9.1 # Worker on compute-01
192.168.9.2 # Worker on compute-02
[colfax@head-node]# export I_MPI_PIN=0
[colfax@head-node]# mpirun \
>     -machinefile machines-CPUs \
>     /nfs/options/options
...
```

リスト 3: マシンファイルを使用したホモジニアス MPI 計算の起動。最初の計算ノードには 2 つの MPI プロセス (ボスと 1 つのワーカー) が割り当てられ、その他のすべてのノードには 1 つのワーカープロセスが割り当てられます。

アプリケーションをコンパイルするには、mpiicc と呼ばれるインテル® C コンパイラー向けのインテル® MPI ラッパーを実行します。-openmp と -mkl コンパイラー・オプションを指定して、OpenMP* およびインテル® MKL ライブラリーを実行時にロードします。

アプリケーションの実行には、インテル® MPI ライブラリーにより提供される mpirun スクリプトを使用します。machines-CPUs ファイルに計算を実行するホストのリストがあります。このファイルの 1 行目は MPI ランク 0 (ボスプロセス) で、2 行目以降はワーカーです。

実行時に重要なことは、環境変数 I_MPI_PIN=0 を設定することです。これは、MPI プロセスをホストに固定にしないように (プロセスピニングを無効にするように) インテル® MPI ライブラリーに指示します。デフォルトでは、1 つのホストで複数の MPI プロセスが起動すると、MPI ライブラリーはこれらのプロセスによる CPU リソースへのアクセスを制限し、プロセス間で効率良く共有できるようにシステムを分割します。このテストでは、ボスプロセスと 1 つ目のワーカーを同じホスト compute-01 に配置します。プロセスピニングが有効な場合、2 ソケットの計算ノードで、ボスは 1 つのソケットを利用し、ワーカーはもう一方のソケットを利用します。しかし、ボスプロセスはシングルスレッドなので、割り当てられたマルチコア・ソケットを十分に活用できず、これは最適とは言えません。プロセスピニングを無効にすると、ワーカーがすべての CPU コアを利用できます。ボスプロセスは CPU 負荷が低いため、ワーカーと一緒に実行しても全体のパフォーマンスは低下しません。

4. コプロセッサーを利用するヘテロジニアス・クラスタリング

このセクションでは、従来のクラスター向けに開発されたエイジアンオプション価格評価アプリケーションを、コードを変更せずにインテル® Xeon Phi™ コプロセッサーを含むヘテロジニアス・クラスターで実行する方法を示します。具体的には、計算ノードと同じネットワークにコプロセッサーを配置し、コプロセッサー上に追加の MPI プロセスを直接配置します。図 4 はこのクラスター構成の図です (図 3 と比較してみてください)。ここでは、インテル® メニーコア・プラットフォーム・ソフトウェア・スタック (インテル® MPSS) [2] がすでにインストールされ、デフォルト設定で構成されていることを前提としています。

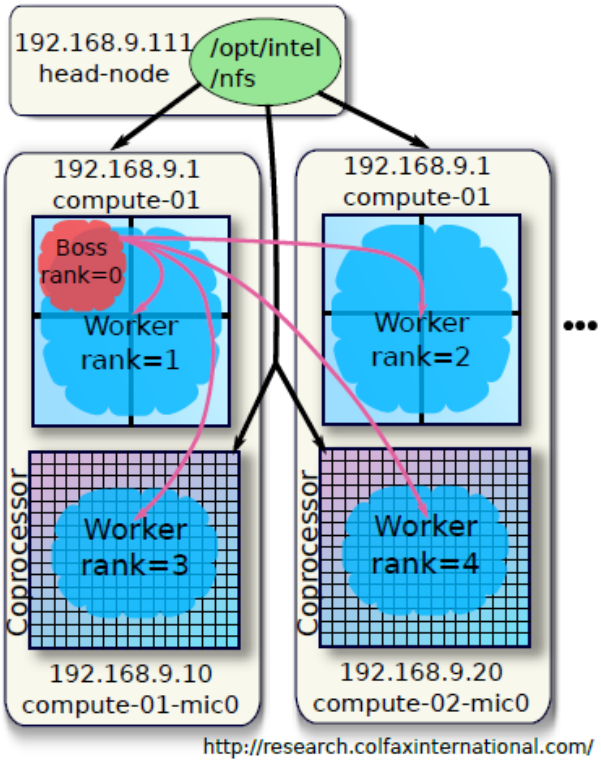

図 4: ヘテロジニアス計算のクラスター構成と MPI 実行設定。アプリケーションは、CPU ベースの計算ノードと、独立した計算ノードとして動作するインテル® Xeon Phi™ コプロセッサーから成るヘテロジニアス・クラスターを使用します。

4.1. コプロセッサーの SSH キー

インテル® Xeon Phi™ コプロセッサーで MPI プロセスをネイティブに実行するには、クラスターのヘッドノードとコプロセッサーのオペレーティング・システムが SSH 通信できなければなりません。SSH 通信では、通常の計算ノードの MPI クライアント認証と同様に、パスワードの代わりに SSH キーを使用して認証を行います。この設定をまだ行っていない場合は、リスト 4 の処理を実行します。

ここでは、各計算ノードのインテル® MPSS を停止し、コプロセッサーの構成をリセットします。この処理を行う理由は 2 つあります。1 つは、これにより、ホスト上のユーザーのホーム・ディレクトリーからすべてのインテル® Xeon Phi™ コプロセッサー上の認証済みキーファイルに SSH キーが自動的にコピーされます。もう 1 つは、セクション 4.2 および 4.3 で述べる後続の処理でインテル® MPSS サービスを停止する必要があるためです。

```
[colfax@head-node]% ssh-keygen
(output omitted)
[colfax@head-node]% cat ~/.ssh/id_rsa.pub >>\
> ~/.ssh/authorized_keys
# The following steps must be repeated
# for all compute nodes in the cluster.
[colfax@head-node]% scp ~/.ssh/id_rsa \
> ~/.ssh/id_rsa.pub ~/.ssh/authorized_keys \
> compute-01:~/
[colfax@head-node]% sudo su
[root@head-node]# ssh compute-01
[root@compute-01]# service mpss stop
Shutting down MPSS stack:     [ OK ]
[root@compute-01]% micctrl --cleanconfig
[root@compute-01]% micctrl --initdefaults
```

リスト 4: SSH キーを使用して、クラスターのコプロセッサーへのパスワード不要アクセスを構成

4.2. ブリッジ・ネットワーク構成

セクション 3.2 で述べたように、compute-01 のホスプロセスは、compute-01 および compute-02 のワーカープロセスと通信できなければなりません。計算デバイス compute-01-mic0 と compute-02-mic0 (コプロセッサー) が追加されたことで、ホスプロセスはこれらとも通信する必要があります。ここで問題となるのは、ホスプロセスは compute-01 にあり、compute-02-mic0 はリモートホストにあることです。

リモートホストとの通信を可能にするため、インテル® MPSS はブリッジ・ネットワーク構成をサポートしています。この構成では、コプロセッサーに計算ホストと同じサブネットの IP アドレスを割り当て、各計算ノードでネットワーク・ブリッジを作成し、そのブリッジを介してコプロセッサーをネットワークに接続します。compute-01 から compute-02-mic0 へメッセージを送る場合について考えてみます。

1) まず、compute-01 のネットワーク・インターフェイス・コントローラー (NIC) から compute-02 の NIC にメッセージが送られます。
2) 次に、compute-02 のオペレーティング・システムとコプロセッサー・ドライバーにより、メッセージは PCIe バスを介してコプロセッサー compute-02-mic0 に送られます。

図 2 はこのパスを図で表したものです。アプリケーションは、この一連の処理を関知しません。つまり、compute-01 は、compute-02-mic0 がリモートホスト上のインテル® Xeon Phi™ コプロセッサーであることを知りません。

ブリッジを設定するには、最初に各計算ノードでネットワーク・ブリッジを作成する必要があります。そのため、OS で (ここでは CentOS 6.4 を使用)、リスト 5 に示す構成を作成します。

```
[root@compute-02]% cat \
> /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
HWADDR="00:1E:67:56:B7:4B"
NM_CONTROLLED="no" ONBOOT="yes"
UUID="ac3cfdfd-b25f-4493-8bad-3f7b9d51d0d1"
BRIDGE=br0
MTU=9000
[root@compute-02]%
[root@compute-02]% cat \
> /etc/sysconfig/network-scripts/ifcfg-br0
DEVICE=br0
TYPE=Bridge ONBOOT=yes
DELAY=0
NM_CONTROLLED="no" MTU=9000
BOOTPROTO=static
IPADDR=192.168.9.2
NETMASK=255.255.25
5.0 NOZEROCONF=yes
```

リスト 5: ネットワーク構成ファイルで計算ホスト上に仮想ブリッジを作成

ifcfg-br0 は新しく作成したファイルです。このファイルで、ホスト compute-02 が仮想インターフェイス br0 を介してネットワークに接続し、IP アドレス 192.168.9.2[4] を取得するように構成します。ifcfg-eth0 ファイルは、OS のインストール時に作成されます。ここでは、このファイルに BRIDGE=br0 行を追加し、このデバイスへ IP アドレスを割り当てる行を削除します。この処理は、手動またはクラスター管理ソフトウェアにより、すべての計算ノードで行う必要があります。

インテル® Xeon Phi™ コプロセッサーのブリッジ・ネットワークを作成するステップをリスト 6 に示します。

リスト 6 から、コマンド micctrl--network... によってこのシステムの 2 つのコプロセッサーの IP アドレスが変更され、変更内容が /etc/hosts に反映されたことが分かります。

[4] クラスターに DHCP サーバーが存在する場合、DHCP を介して br0 を接続することもできます。

```
[root@compute-02]% micctrl --addbridge=br0 \
> --type=external --ip=192.168.9.1 \
> --netbits=24
[root@compute-02]% micctrl --network \
> --bridge=br0 --ip=192.168.9.20
          mic0: Changing network to
                static bridge br0 at
                192.168.9.20
          mic1: Changing network to
                static bridge br0 at
                192.168.9.21
[root@compute-02]% cat /etc/hosts | grep
192 192.168.9.1 compute-01
192.168.9.2 compute-02
192.168.9.20 compute-02-mic0 mic0
192.168.9.21 compute-02-mic1 mic1
```

リスト 6: 計算ノードにおけるコプロセッサーの構成と外部ネットワーク・ブリッジへの接続。これにより、リモートマシンのコプロセッサーに IP アドレスを指定してアクセスできるようになります。

これで、このマシンのコプロセッサーにホストと同じサブネットの IP アドレスが割り当てられるため、このサブネットを利用してリモートマシンからコプロセッサーに ping、SSH 通信、MPI メッセージの送信が可能になります。前述のとおり、この処理は `compute-01` を含むすべての計算ノードで行う必要があることに注意してください。

インテル® Xeon Phi™ コプロセッサーのネットワーク構成の詳細は、[3] を参照してください。

4.3. コプロセッサーとのネットワーク・ファイルの共有

ブリッジ・ネットワーク構成では、インテル® Xeon Phi™ コプロセッサーを含むクラスター全体にわたって NFS を構成できるため、コプロセッサーでも共有マウントを利用できます。この便利な機能により、アプリケーションを簡単に初期化し、コプロセッサーで実行する MPI プロセスで直接ファイル I/O を行えます。

セクション 3 で述べたように、ヘッドノードは NFS サーバーとして構成されており、`/opt/intel` と `/nfs` をサブネット `192.168.9.0/24` のすべてのホストにエクスポートします。あとは、各コプロセッサーの uOS からこれらのディレクトリーをマウントするだけです。

リスト 7 に、`micctrl` ツールを使ってコプロセッサーの `/etc/fstab` に NFS エントリーを作成する方法を示します。これらの NFS エントリーは、コプロセッサーの起動時に自動的にマウントされ、システム全体の再起動後も維持されます。ヘッドノード `192.168.9.111` から `/opt/intel` と `/nfs` をマウントします。コプロセッサーのマウントパスはヘッドノードと同じです。

NFS マウントのセットアップは、クラスターのすべての計算ノードで行う必要があります。NFS の追加はヘテロジニアス・クラスター構成の最後のステップなので、ここでインテル® MPSS サービスを再開します。これは、リスト 7 では最後のコマンドになります。

```
[root@compute-01]% micctrl --addnfs=/opt/intel \
> --dir=/opt/intel --server=192.168.9.111
[root@compute-01]% micctrl --addnfs=/nfs \
> --dir=/nfs --server=192.168.9.111
[root@compute-01]% service mpss
start Starting MPSS Stack:    [ OK ]
[root@compute-01]%
```

リスト 7: コプロセッサーの 2 つの NFS マウントのセットアップとインテル® MPSS の起動

4.4. コンパイルとヘテロジニアス実行

ヘテロジニアス・クラスターの構成が完了したら、インテル® MIC アーキテクチャー向けに MPI アプリケーションを再コンパイルして、コプロセッサーを利用するヘテロジニアス計算を開始します。リスト 8 にこの手順を示します。

追加の処理を確認するため、リスト 8 とリスト 3 (従来の MPI 実行の起動) を比較してみましょう。

a) リスト 8 では、コンパイルを 2 回実行しています。1 回目は、`-xAVX` オプションを指定し、インテル® アドバンスト・ベクトル・エクステンション (インテル® AVX) 命令に対応した CPU アーキテクチャー向けの実行ファイルを生成します。ファイル名は `options` です。2 回目は、`-mmic` オプションを指定し、インテル® Xeon Phi™ コプロセッサー向けのネイティブ実行ファイルを生成します。実行ファイル名は `options.MIC` です。

b) リスト 8 では、マシンファイルに、計算ノードの IP アドレスに加えてコプロセッサーの IP アドレスが含まれます。つまり、MPI はコプロセッサーにプロセスを配置します。各システムには 2 つのコプロセッサーが搭載されていますが、ここではノードごとに 1 つのコプロセッサーのみ使用します。これは、パフォーマンスを公正に比較するためです。各マシンの 2 つ目のコプロセッサーは、その IP アドレスをマシンファイルに追加することでいつでもロードできます。

```
[colfax@head-node]# source /opt/intel/impi/4.1.1/intel64/bin/mpivars.sh # Intel MPI environment
[colfax@head-node]# mpiicc -std=c99 -mkl -openmp -xAVX -o options options.c # Compile for CPU
[colfax@head-node]# mpiicc -std=c99 -mkl -openmp -mmic -o options.MIC options.c # Compile for MIC
[colfax@head-node]# cp -v options options.MIC /nfs/options/ # Copy to NFS-shared location
‘options’ -> ‘/nfs/options/options’
‘options.MIC’ -> ‘/nfs/options/options.MIC’

# Starting the MPI job
[colfax@head-node]# cat ./machines-HETEROGENEOUS # View the machine file
192.168.9.1 # Boss process on compute-01
192.168.9.1 # Worker on compute-01
192.168.9.2 # Worker on compute-02
192.168.9.10 # Worker on compute-01-mic0
192.168.9.20 # Worker on compute-02-mic0
[colfax@head-node]# export I_MPI_PIN=0 # Disable MPI process pinning
[colfax@head-node]# export I_MPI_MIC=on # Enable MPI for the MIC architecture
[colfax@head-node]# export I_MPI_MIC_POSTFIX=.MIC # Postfix of the MIC architecture executable
[colfax@head-node]# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MIC_LD_LIBRARY_PATH # Enables MKL on MIC
[colfax@head-node]# mpirun -machinefile machines-HETEROGENEOUS /nfs/options/options # Launch
```

リスト 8: ヘテロジニアス MPI 計算を開始。作業は CPU とコプロセッサー間で分配されます。MPI マシンファイルでは、コプロセッサーは独立した計算ノードと見なされます。

c) 環境変数 `I_MPI_MIC=on` を設定し、インテル® Xeon Phi™ コプロセッサーでネイティブプロセスが起動されることをインテル® MPI ライブラリーに知らせます。
d) 環境変数 `I_MPI_MIC_POSTFIX=.MIC` は、MIC 実行ファイル名の接尾辞を `.MIC` に指定します (前述のとおり、CPU アーキテクチャー向けの実行ファイル名は options、インテル® MIC アーキテクチャー向けの実行ファイル名は `options.MIC` です)。
e) `LD_LIBRARY_PATH` により、コプロセッサー上で NFS 共有場所 `/opt/intel` からインテル® MKL ライブラリーを簡単にロードできます。

マシンファイル名を除いて、`mpirun` の引数に変更はありません。これで、図 4 のエイジアンオプション価格評価の (コプロセッサーを利用する) ヘテロジニアス計算のセットアップと起動が完了しました。

5. パフォーマンス結果

すべてのテストは、2 つの計算ノード (Colfax International 製 SXP-8600 ワークステーション) から成るクラスターで実行しました。各マシンは 2 ソケットで、各ソケットには 8 コアのインテル® Xeon® プロセッサー E5-2687W V2 が搭載されています。

メモリーモジュールは 1600MHz 16GB で、サーバーごとのメモリー総容量は 128GB です。各システムには 2 つのインテル® Xeon Phi™ コプロセッサー QS-3120A (アクティブ冷却方式) が搭載されており、ベンチマークではシステムごとにこのうちの 1 つのみを使用しました。また、CentOS 6.4 Linux* オペレーティング・システム (カーネル 2.6.32-358)、インテル® C コンパイラー 13.1.3.192、インテル® MPSS 2.1.6720-15 を使用しました。ハードディスクは、すべてのマシンで Toshiba MG03ACA100 を採用しました。

5.1. コプロセッサーを利用する計算

図 5 は、次の 3 つの計算のパフォーマンスを示す棒グラフです。

1) リスト 3 の設定で CPU のみで計算を行った場合。
2) コプロセッサーのみで計算を行った場合。ここでは、この設定について触れていませんが、リスト 8 のマシンファイルから 2 行目と 3 行目を削除して実行します。
3) リスト 8 の設定でコプロセッサーを使用してヘテロジニアス計算を行った場合。

パフォーマンスは 1 秒あたりに価格評価されたオプションパスの数で測定し、棒グラフには各計算デバイス (CPU またはコプロセッサー) の割合も示しています。

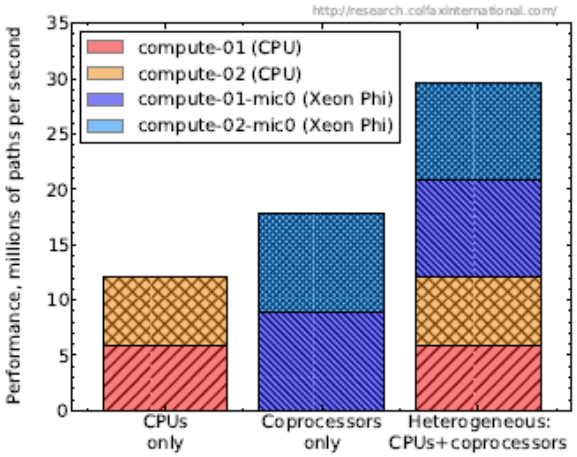

図 5: 3 つの構成 (CPU のみ、コプロセッサーのみ、両方) のパフォーマンス

パフォーマンスは $M \times N \times Q / \tau$ で計算され、τ はクラスターで Q 個のパラメーター・セットの処理にかかったウォールクロック時間、M は各パラメーター・セットで評価されたモンテカルロ・パスの数、N はエイジアンオプションの平均に使用された期間の数です。テストでは $M=2^{20}$、$N=365$、$Q=100$ を使用しました。シミュレーションで使用されたその他のパラメーターの値は、$S=15.3$、$T=1.0$、$\mu=\gamma=.08$ (0.05 < ν < 0.5 および 10 < K < 20) ですが、これらのパラメーターは、アプリケーションのパフォーマンスに影響しません。

この設定では、MPI 通信の頻度が少なく、ワークアイテムの数 Q が十分に大きいため、4 つの計算デバイスで負荷がほぼ均等になります。そして、CPU とコプロセッサーを併用した場合のパフォーマンスは、CPU のみおよびコプロセッサーのみを利用した場合のパフォーマンスを合わせたものと等しくなります。スケジュール・オーバーヘッドについては、セクション 6.2 で取り上げます。

5.2. 要因: MPI および NFS 速度

ここでは、パフォーマンス結果を理解し、このアプローチのスケーラビリティーの限界を予測するため、ハイブリッド・クラスター・パフォーマンスに関するテストを追加で行いました。

最初のテストでは、クラスター内の MPI 通信のレイテンシーと帯域幅を測定しました。この指標は、与えられた問題に対する、ボス・ワーカー・モデルの並列スケーラビリティーの限界を特定する上で重要です。MPI パフォーマンスの測定には、インテル® MPI ベンチマークを使用しました。さまざまな設定で "PingPong" テストを実行し、ホスト `compute-01` (ボスプロセス) と次のノード間の通信効率を測定しました。

1) `compute-01` (ボスと同じ CPU ノードのワーカー)
2) `compute-02` (リモート計算ノードのワーカー)
3) コプロセッサー `compute-01-mic0` (ローカル MIC ノードのワーカー)
4) コプロセッサー `compute-02-mic0` (リモート MIC ノードのワーカー)

図 6 はこのテストの結果をグラフで表したものです。

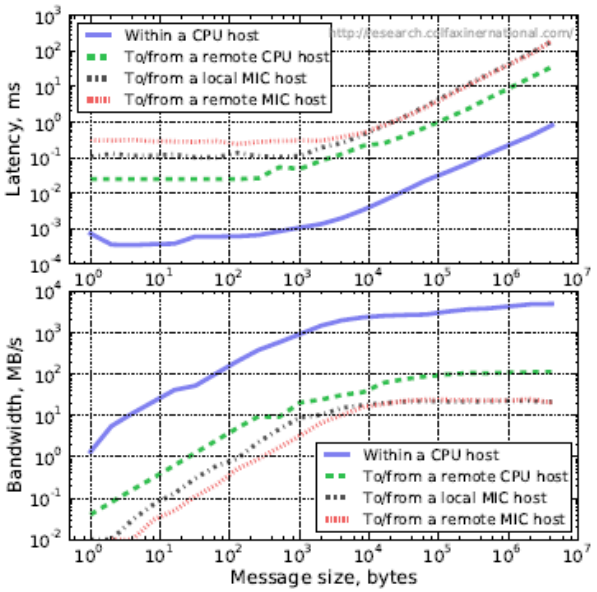

図 6: インテル® MPI ベンチマーク PingPong による CPU ベースのホスト (ボスプロセス) と他のプラットフォームのプロセス (ワーカープロセス) 間における MPI 通信のレイテンシーと帯域幅の測定結果

2 つ目のテストでは、NFS 共有ファイルのパフォーマンスを測定しました。この指標は、(プラグマベースのオフロード機能による初期化ではなく) ヘテロジニアス MPI を利用して、コプロセッサー上でデータファイルから MPI プロセスを初期化するアプリケーションでは重要です。

さらに、MPI プロセスがホストへデータを送る代わりに、ファイルへデータを出力する場合も書き込み速度を把握しておいたほうが良いでしょう。ここでは、次の場所にある大きなファイルの読み取り/書き込み速度を測定しました。

1) ローカル・ハードドライブ (ヘッドノード)
2) リモート CPU ホスト `compute-02`
3) リモート MIC ホスト `compute-02-mic0`

テストでは、Linux* ツール dd を使って、1GB のファイルの読み取り/書き込みを行いました。キャッシュの影響を避けるため、書き込み操作では、操作の前と後に sync でディスクキャッシュをフラッシュしました。読み取り操作では、読み取る前に NFS 共有をアンマウントし、再マウントしてディスクキャッシュを消去しました。ローカル・ファイル・システムでは、`/proc/sys/vm/drop_caches` に "3" を書き込んでディスクキャッシュを消去しました。図 7 にパフォーマンスの測定結果を示します。

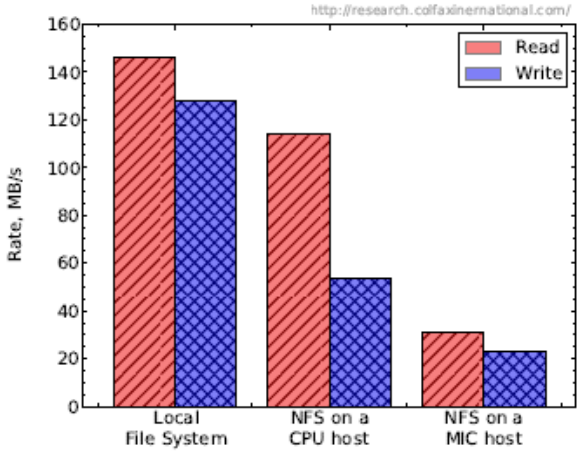

図 7: 計算ノードとコプロセッサーの uOS のファイルシステム、ローカル、NFS 共有にある 1GB ファイルの読み取り/書き込み速度

ヘッドノードのローカルドライブの I/O 速度は、読み取りが 145MB/秒、書き込みが 130MB/秒に達しています。リモート CPU ホスト (`compute-02`) では、NFS の I/O 速度は読み取りが 110MB/秒、書き込みが 50MB/秒に過ぎません。これは、マシンのインターコネクト・スイッチの帯域幅が 1 ギガビット/秒であるためです。リモート MIC ホスト (`compute-02-mic0`) では、NFS の I/O 速度は読み取りが 30MB/秒、書き込みが 25MB/秒に低下します。インターコネクトと PCIe バスの帯域幅のほうがかなり大きいため、これはインテル® MIC アーキテクチャーのネットワーク・スタックの実装による制限と言えます。

6. まとめ

このホワイトペーパーでは、従来の CPU ベースのクラスター向けに開発された一部のアプリケーションは、コードを変更せずに、インテル® Xeon Phi™ コプロセッサーを利用するヘテロジニアス・クラスターで実行できることを示しました。コプロセッサーの追加により、パフォーマンスが大幅に向上することも実証しました。このセクションでは、このアプローチを成功に導くための必要条件と、従来のオフロードベースのアクセラレーションと比較したヘテロジニアス・クラスターの制限について述べます。

6.1. コプロセッサー上でパフォーマンスを向上するための必要条件

インテル® C/C++ コンパイラーおよびインテル® Fortran コンパイラーは、高水準言語で記述された汎用 CPU 向けコードを、インテル® MIC アーキテクチャー・ベースのインテル® Xeon Phi™ コプロセッサー向けにコンパイルできます。ただし、別のホワイトペーパー ([4]、[5]、[6]、[7] など) で触れたように、次の並列プログラミングのガイドラインに沿って設計されているコードでのみ、コプロセッサーは互換性のあるインテル® Xeon® プロセッサー・ベースのシステムよりも優れたパフォーマンスを達成することができます。

a) OpenMP* または Pthreads* フレームワークによってアルゴリズムが複数のコアを利用している。
b) 同期とフォルス・シェアリングがわずかで、アプリケーションが 100 以上のスレッドまで線形にスケーリングできる。
c) コンパイラーによる自動ベクトル化が可能なデータ構造とループの実装を備えている。
d) ユーザーコードでベクトル命令を使用していない (インテル® Xeon Phi™ コプロセッサーはインテル® SSE およびインテル® AVX 命令をサポートしていません)。
e) メモリー・トラフィックが帯域幅に制限されない。または時間と空間の局所性に優れているため、キャッシュが効率良く使用されている。

このガイドラインについては [8] で詳しく説明されています。

このガイドラインは、マルチコア CPU 向けの HPC アプリケーションにも当てはまります。そのため、"コードを変更しない" アプローチで最も重要な必要条件は、高度に最適化されたマルチコア CPU の実装です。

6.2. ヘテロジニアス MPI の制限

ほとんどの MPI アプリケーションは、ここで示した方法でインテル® Xeon Phi™ コプロセッサー向けに変換できますが、すべてのアプリケーションが最適化や再構築を行わずにハイブリッド・クラスターで高速に実行できるわけではありません。以下は、ヘテロジニアス実行を成功に導くための必要条件です。

a) MPI 通信の量が多くなりすぎないようにします。ここでは、一般的な 1 ギガビットのイーサネット・スイッチを使用し、TCP/IP を介して MPI メッセージの受け渡しを行いました。図 6 に示すように、ホストからローカル・コプロセッサーへのメッセージ・パッシングの帯域幅は 20MB/秒に過ぎません。これは、従来のオフロード・データ・トラフィックの帯域幅 6 ～ 7GB/秒[8] よりも 300 倍も低く、また、テストで測定された CPU ベースの計算ノードの帯域幅よりも 5 倍も低くなります。そのため、リモート・コプロセッサーを IP アドレス指定可能にすることで、クラスターの CPU とローカル・コプロセッサー間の通信の帯域幅が大幅に向上します。MPI 通信の量が制限の要因になるかどうかは、アプリケーションにおけるデータ移動と計算の割合によって決まります。

b) 頻繁に同期するアプリケーションでは、ヘテロジニアス・クラスターの通信レイテンシーが問題になることがあります。図 6 に示すように、CPU ホストからリモート・コプロセッサーへのメッセージ・パッシングの通信レイテンシーは最も高く、ショートメッセージの PingPong に 0.3 ミリ秒かかります (CPU ホスト間では 0.02 ミリ秒)。従来のホモジニアス・クラスターで同期を必要としない並列アプリケーションでも、ヘテロジニアス・クラスターでは何らかのロードバランスが必要になるため、通信レイテンシーが重要になります。ボス・ワーカー・モデルでは、ボスがワーカーにワークアイテムを渡すため、通信レイテンシーによりアプリケーションの並列スケーラビリティーが制限されます。ボス・ワーカー・モデルのワーカーの最大数は、スケジュール・レイテンシーに対する τ (1 ワーカーによる 1 ワークアイテムの平均計算時間) の比率よりもかなり小さくなければなりません。

$$N_W \ll \frac{\tau}{0.3\ \mathrm{ms}} \qquad (8)$$

アプリケーションで式 (8) の制限を超えるスケーリングが必要な場合、ワークアイテムのサイズを大きくするか、ワークアイテムのサイズを動的に調整して、スケジュール・オーバーヘッドとロードバランスの妥協点を見つける必要があります。ワークアイテムの計算時間が予測可能な場合、動的スケジューリングを避け、コプロセッサーの相対パフォーマンスに応じてより多くの作業を割り当てることで、バランスのとれたワークロードを静的にスケジュールできる可能性があります。[8] のセクション 4.7 に、このスケジュール・モードの例があります。複雑なケースでは、ボスプロセスの階層構造やワークスチール・アルゴリズムのような全体的なワーク・スケジュールが必要になります。

c) NFS を介するコプロセッサーのファイル I/O の負荷が高くなりすぎないようにします。NFS は、コプロセッサーでファイルから初期化を行い、結果を出力できる便利な機能です。しかし、インテル® Xeon Phi™ コプロセッサーの現在の NFS 実装は速度が遅いため、高速 I/O は期待できません。図 7 に示すように、コプロセッサーの NFS 共有ディレクトリー内の読み取り/書き込みは、1 ギガビット・インターコネクトの 20% にしか達しません。コプロセッサーでファイルから MPI プロセスを初期化することが計算のボトルネックになる場合、従来のオフロードベースのアクセラレーションにアプリケーションを再構築する必要があるかもしれません。

6.3. このアプローチの長所と短所

これまで説明したように、独立した計算ノードとしてインテル® Xeon Phi™ コプロセッサーを利用するヘテロジニアス・クラスター構成により、"そのまま" でアクセラレーションが得られる場合があります。このアプローチの最も優れている点は、専用のネットワーク・ハードウェアを必要としないことです。ここでテストしたクラスターは、一般的な 1 ギガビット/秒のインターコネクトを介してノードを接続しています。ブリッジ・ネットワーク構成は便利ですが、通信レイテンシーが高く、データ転送帯域幅が狭いのが欠点です。

通信負荷の高いワークロードで InfiniBand* ファブリックは、レイテンシーと CPU オーバーヘッドを軽減し、Remote Direct Memory Access (RDMA) などの高度な先進技術を提供します。InfiniBand* ファブリックであるインテル® TrueScale 製品は、OpenFabrics Enterprise Distribution (OFED) と組み合わせて、インテル® MPSS に統合できます。インテル® TrueScale を使って、簡単なプログラミング手法によりパフォーマンスを向上できる可能性があります。Colfax Research の別のホワイトペーパーでは、これらの高度な HPC テクノロジーについて検証しています。

参考文献 (英語)

[1] Hetetogeneous Clustering with Homogeneous Code (landing page for this paper).
http://research.colfaxinternational.com/post/2013/10/17/Heterogeneous-Clustering.aspx.

[2] Intel Manycore Platform Software Stack (MPSS).
http://software.intel.com/en-us/articles/intel-manycore-platform-software-stack-mpss.

[3] System Administration for the Intel Xeon Phi Coprocessor.
http://software.intel.com/en-us/articles/system-administration-for-the-intel-xeon-phi-coprocessor.

[4] Andrey Vladimirov. Auto-Vectorization with the Intel Compilers: is Your Code Ready for Sandy Bridge and Knights Corner?
http://research.colfaxinternational.com/post/2012/03/12/AVX.aspx.

[5] Andrey Vladimirov and Vadim Karpusenko. Testdriving Intel Xeon Phi coprocessors with a basic N-body simulation.
http://research.colfaxinternational.com/post/2013/01/07/Nbody-Xeon-Phi.aspx.

[6] Vadim Karpusenko and Andrey Vladimirov. How to Write Your Own Blazingly Fast Library of Special Functions for Intel Xeon Phi Coprocessors.
http://research.colfaxinternational.com/post/2013/05/03/Fast-Library-Xeon-Phi.aspx.

[7] Andrey Vladimirov. Multithreaded Transposition of Square Matrices with Common Code for Intel Xeon Processors and Intel Xeon Phi Coprocessors.
http://research.colfaxinternational.com/post/2013/08/12/Trans-7110.aspx.

[8] Colfax International. *Parallel Programming and Optimization with Intel Xeon Phi Coprocessors*. ISBN:978-0-9885234-1-8.Colfax International, 2013.
http://www.colfax-intl.com/xeonphi/book.html.