インテル® Xeon Phi™ コプロセッサーを搭載したクラスターの ギガビット・イーサネットおよび InfiniBand* 上の ピア・ツー・ピア MPI 通信の構成とベンチマーク

Vadim Karpusenko および Andrey Vladimirov

Colfax International

2014 年 3 月 11 日

概要

インテル® Xeon Phi™ コプロセッサーでは、オフロードベースのクラスタリングとは対照的に、MPI プロセスをすべてコプロセッサーで実行する、シンメトリック・ヘテロジニアス・クラスタリング・モデルが可能です。CPU ベースのアプリケーションをメニーコア・コンピューティング・アクセラレーターにより簡単に移植できるため、これらのシンメトリック・モデルは魅力的です。

しかし、デフォルトのソフトウェア構成で、専用のネットワーク・ハードウェアを利用しない場合、クラスターのコプロセッサー間のピア・ツー・ピア通信はギガビット・イーサネット・ネットワーク・ハードウェアの能力と比較して桁違いに劣ります。この状況は、InfiniBand* インターコネクトと、それらをサポートするソフトウェアにより改善できます。

このホワイトペーパーでは、ギガビット・イーサネットおよび InfiniBand* インターコネクトで接続された、インテル® Xeon Phi™ コプロセッサーを搭載したクラスターの構成手順について説明します。また、InfiniBand* をサポートする高度な構成を使用した場合と使用しない場合の MPI メッセージのレイテンシーと帯域幅を比較します。さらに、インテル® Xeon Phi™ コプロセッサーを搭載した InfiniBand* 対応クラスターにおける MPI アプリケーションのチューニング、InfiniBand* プロトコルの影響についてのケーススタディー、ハイパフォーマンス・コンピューティング・アプリケーションで PCIe バスを介した不均一な RDMA パフォーマンスに対処するための推奨事項についても触れます。

目次

略語リスト

API	Application Programming Interface
CCL	Coprocessor Communication Link
GPGPUs	General-Purpose Graphics Processing Units
HCA	Host Channel Adapter
IMB	Intel MPI Benchmark
MIC	Many Integrated Core
MKL	Intel Math Kernel Library
MPI	Message Passing Interface
MPSS	Manycore Platform Software Stack
NFS	Network File Sharing
NIC	Network Interconnect
OFA	OpenFabrics Alliance
OFED	Open Fabrics Enterprise Edition
PCIe	Peripheral Component Interconnect Express
QPI	Quick Path Interconnect
RDMA	Remote Direct Memory Access

1. インテル® MIC アーキテクチャーを利用したメッセージパッシング

メッセージ・パッシング・インターフェース (MPI) は、分散メモリーシステムで並列プログラミングを可能にするライブラリー・ルーチンの構文とセマンティクスを定義する規格です。MPI は、スケーラビリティーと移植性を備えたハイパフォーマンスなアプリケーションを提供することを目標としています。MPI を使用するアプリケーションは、ネットワーク機能を明示的に管理する必要がないため、開発者は並列アルゴリズムの設計に集中することができます。同時に、MPI は、Mellanox InfiniBand* やインテル® True Scale のような、非常に高いスループットと低いレイテンシーを備えた、カスタマイズされた商用ネットワークで動作します。

多くの既存の MPI 実装の中で、インテル® MPI ライブラリー ([2] も参照) のみ、クラスターのインテル® Xeon Phi™ コプロセッサー をサポートしています。インテル® Xeon Phi™ コプロセッサーは、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・ベースの製品で、汎用のインテル® Xeon® プロセッサー よりも優れたパフォーマンスを提供します。

インテル® Xeon Phi™ コプロセッサーを使用するアプリケーションでは、汎用 GPU (GPGPU) 向け CUDA フレームワークに似た、オフロード・プログラミングを利用できます。オフロード・プログラミングでは、宣言子を使用して、CPU アプリケーションの計算集約部分と関連データをホストシステムのメモリーからコプロセッサーに送ります。GPGPU と異なり、インテル® Xeon Phi™ コプロセッサーは、CPU アーキテクチャー向け関数と同じ C、C++、Fortran コードからコンパイルされたパフォーマンス・クリティカルな関数を実行できます。オフロード・クラスタリング・モデルの概要は、図 1 を参照してください。

さらに、インテル® Xeon Phi™ コプロセッサーは、コンピューティング・クラスター上で独立した IP アドレスを持つメニーコア・ノードとして動作し、MPI プロセスはコプロセッサー上でホスト CPU の関与なく動作します (イラストは図 2、ケーススタディーは [3] を参照)。コードを再構成することなくデータのオフロードを実装し、コプロセッサーを使用して HPC アプリケーションをスピードアップできるため、シンメトリック・クラスタリング・モデルは、既存の CPU ベースクラスター向け MPI アプリケーションのユーザーにとって魅力的です。

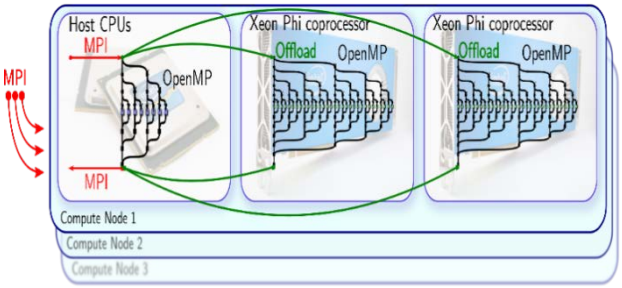

図1 : オフロード・プログラミングを利用した従来のヘテロジニアス・クラスタリング。MPI プロセスは、オフロードメカニズムによりインテル® Xeon Phi™ コプロセッサーを利用して、計算ノードのプロセッサーでのみ実行されます。コプロセッサー間のピア・ツー・ピア通信は不可能で、アプリケーションはオフロード API (アプリケーション・プログラミング・インターフェイス) を利用してコプロセッサー間でコードとデータを移動します。

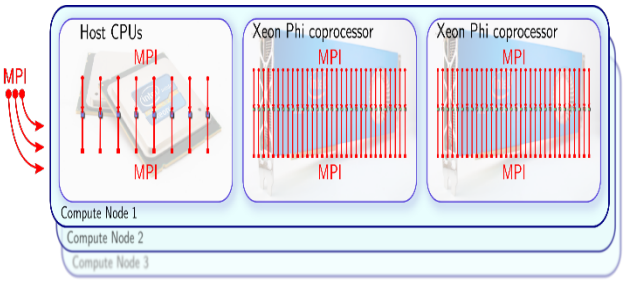

図2 : シンメトリック・ヘテロジニアス・クラスタリング。MPI プロセスはコプロセッサーで直接起動されます。オペレーティング・システムで仮想化されたネットワーク・ファブリックにより、コプロセッサー間のピア・ツー・ピア通信が可能です。このモードでは、CPU 向けの MPI アプリケーションを、コードを変更することなくコプロセッサーで実行できます (ただし、チューニングは必要です)。

前述したように、MPI は通信に異なるネットワーク構成とファブリックを使用できます。これは特に、シンメトリック・クラスターのコプロセッサー間のピア・ツー・ピア通信に当てはまります。このホワイトペーパーでは、イーサネット・ファブリック上の TCP プロトコルと InfiniBand* ファブリック上の DAPL プロトコルでコプロセッサー間の MPI 通信パフォーマンスを比較します。

セクション 2 では、インテル® Xeon Phi™ コプロセッサーがネットワーク・ファブリックを使ってピア・ツー・ピア通信をできるようにするソフトウェア・スタックを説明します。セクション

3 では、インテル® Xeon Phi™ コプロセッサーを搭載したシンメトリック・ヘテロジニアス・クラスターでイーサネットと InfiniBand* によるネットワーク構成手順を示します。セクション 4 では、MPI ベンチマークの結果を提供し、セクション 5 では、通信の改善がアプリケーションのパフォーマンスに与える影響に関するケーススタディーを紹介します。そして、まとめ (セクション 6) では、MIC 対応クラスターにおける InfiniBand* ベースのピア・ツー・ピア通信の長所と短所を示し、HPC ワークロードの最適化について述べます。

2. コプロセッサー間のピア・ツー・ピア・メッセージング

2.1. イーサネット

インテル® Xeon Phi™ コプロセッサーは、PCI Express (PCIe) エンドポイント・デバイスです。ネットワーク接続用のイーサネット・ポートや InfiniBand* ポートは装備されていませんが、コプロセッサーの Linux* オペレーティング・システム (*µOS*) とホストのインテル® 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) により、コプロセッサー上でネットワークを仮想化します。

インテル® MPSS をデフォルト設定でインストールすると、"スタティック・ペア" ネットワーク・トポロジーが提供されます。この構成の内容は次のとおりです。

i) ホストのイーサネット・ネットワーク・インターコネクト (NIC) はクラスターのプライベート・ネットワークに接続されます (この例では、クラスターのサブネットは 10.33.0.0/15 です)。

ii) ホストに、仮想 NIC `mic0`、`mic1`、... が作成され、そのスタティック IP アドレスとサブネットは 172.31.1.254/24、172.31.2.254/24、... です。

iii) ホストの `/etc/hosts` ファイルには、ローカルシステム内部のインテル® Xeon Phi™ コプロセッサーのホスト名と IP アドレスが含まれます (host `mic0` at 172.31.1.1、host `mic1` at 172.31.2.1、...)。

iv) 各コプロセッサーの µOS に、仮想 NIC `mic0` (または `mic1`、`mic2`、...) が表示され、スタティック IP アドレス 172.31.1.1 (または 172.31.2.1、172.31.3.1、...) を取得するように構成されます。

v) 各コプロセッサーの µOS で、ホストの `/etc/hosts` ファイルには、ホストのホスト名とコプロセッサーの IP アドレスが含まれます (host at 172.31.1.254、`mic0` at 172.31.1.1、...)。

つまり、各インテル® Xeon Phi™ コプロセッサーがマシン内部のプライベート・サブネットに配置され、ホストとコプロセッサー間の通信、およびコプロセッサー間のピア・ツー・ピア・メッセージングが可能になります。物理的には、通信は PCIe バス上で行われますが、この通信は TCP/IP 対応の NIC として仮想化されます。リスト 1 に、スタティック・ペアの構成を示します。

別の可能なネットワーク構成は、フラット・ネットワーク・トポロジーで、これは外部ブリッジ構成とも呼ばれます (図 3 を参照)。フラット・ネットワーク・モデルでは、Linux* のネイティブ・ブリッジ・メカニズムが使用されます。この構成の内容は次のとおりです。

i) ブリッジ `br0` がホストに作成され、ホストの NIC 経由でクラスター・ネットワークに接続されます。

ii) ホストのプライマリー・ネットワーク・インターフェイス `eth0` が `br0` に接続されます。

iii) コプロセッサーの仮想インターフェイス `mic0`、`mic1`、... が `br0` に接続され、クラスターのプライベート・ネットワークの IP アドレスを取得します。

その結果、同じ方法で構成されたリモートマシンのコプロセッサーに、アドレスを指定して TCP/IP パッケージを直接送ることができます。フラット・ネットワーク構成の最終結果をリスト 2 に示します。

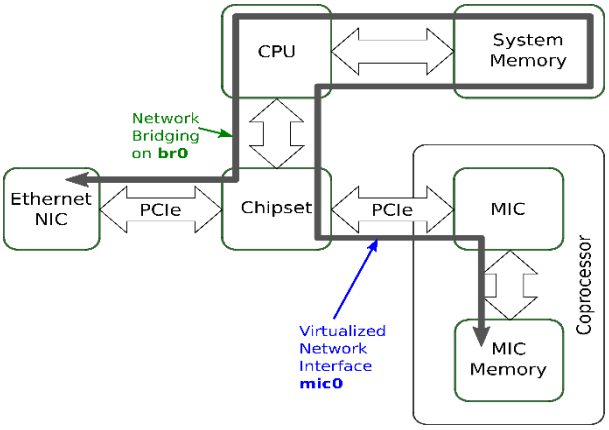

図3 : TCP/IP パケットのパスとシンメトリック・ヘテロジニアス・クラスター内部のインテル® Xeon Phi™ コプロセッサーからブリッジされたネットワーク。

セクション 3.1 で、スタティック・ペア・ネットワークと外部ブリッジを含むフラット・ネットワークの構成手順について説明します。後者の構成は、イーサネット上でインテル® MIC アーキテクチャー対応のシンメトリック MPI アプリケーションを実行するためだけに必要なわけではありません。計算ノードのイーサネット接続にフラット・ネットワークを利用すると、シンメトリック MPI の InfiniBand* 接続を構成することもできます。

```
[root@c001-n002 ~]# # We are on host c001-n002
[root@c001-n002 ~]# cat /etc/hosts
...
10.33.1.2 c001-n002 # Private cluster network
172.31.1.1  c001-n002-mic0 mic0 # Network...
172.31.2.1  c001-n002-mic1 mic1 # ...within...
172.31.3.1  c001-n002-mic2 mic2 # ...this...
172.31.4.1  c001-n002-mic3 mic3 # ...machine.
[root@c001-n002 ~]#
[root@c001-n002 ~]# ifconfig
...
eth0  Link encap:Ethernet  HWaddr 00:25:90:C3...
      inet addr:10.33.1.2  Bcast:10.35.255.255..
...



mic0  Link encap:Ethernet  HWaddr 4C:79:BA:1A...
      inet addr:172.31.1.254  Bcast:172.31.1.255
...
mic1  Link encap:Ethernet  HWaddr 4C:79:BA:1A...
      inet addr:172.31.2.254  Bcast:172.31.2.255
...
mic2  Link encap:Ethernet  HWaddr 4C:79:BA:1A...
      inet addr:172.31.3.254  Bcast:172.31.3.255
..
mic3  Link encap:Ethernet  HWaddr 4C:79:BA:1A...
      inet addr:172.31.4.254  Bcast:172.31.4.255
...
[root@c001-n002 ~]# ssh mic0
[root@c001-n002-mic0 ~]# # Now we are on mic0
[root@c001-n002-mic0 ~]# cat /etc/hosts
...
172.31.1.254        host c001-n002
172.31.1.1        c001-n002-mic0 mic0
172.31.2.1        c001-n002-mic1 mic1
172.31.3.1        c001-n002-mic2 mic2
172.31.4.1        c001-n002-mic3 mic3
[root@c001-n002-mic0 ~]#
[root@c001-n002-mic0 ~]# ifconfig
...
mic0  Link encap:Ethernet  HWaddr 4C:79:BA:1A...
      inet addr:172.31.1.1  Bcast:172.31.1.255..
...
```

リスト 1: ホスト `c001-n002` と 4 つのインテル® Xeon Phi™ コプロセッサーのデフォルトのスタティック・ペア・ネットワーク構成。マシン内のホストとコプロセッサーは TCP/IP を使用して互いに通信できますが、コプロセッサーから (ほかのマシンの) リモート・コプロセッサーへの通信はできません。この構成は、オフロード・クラスタリング・モデル (図 1) や 1 つの計算ノードでのみ実行するシンメトリック MPI アプリケーション (図 2) に適しています。

```
[root@c001-n002 ~]# # We are on host c001-n002
[root@c001-n002 ~]# cat /etc/hosts
...
10.33.1.2 c001-n002 # Private cluster network
10.33.1.22  c001-n002-mic0 mic0 # Now also on..
10.33.1.42  c001-n002-mic1 mic1 # ...private...
10.33.1.62  c001-n002-mic2 mic2 # ...cluster...
10.33.1.82  c001-n002-mic3 mic3 # ...network.
[root@c001-n002 ~]#
[root@c001-n002 ~]# ifconfig
...
br0   Link encap:Ethernet  HWaddr 00:25:90:C3...
      inet addr:10.33.1.2  Bcast:10.35.255.255..
...
eth0  Link encap:Ethernet  HWaddr 00:25:90:C3...
...

mic0  Link encap:Ethernet  HWaddr 4C:79:BA:1A...
...

mic1  Link encap:Ethernet  HWaddr 4C:79:BA:1A...
...

mic2  Link encap:Ethernet  HWaddr 4C:79:BA:1A...
...

mic3  Link encap:Ethernet  HWaddr 4C:79:BA:1A...
...

[root@c001-n002 ~]# ssh mic0
[root@c001-n002 ~]# # Now we are on mic0
[root@c001-n002-mic0 ~]# cat /etc/hosts
...
10.33.1.2        host c001-n002
10.33.1.22        c001-n002-mic0 mic0
10.33.1.42        c001-n002-mic1 mic1
10.33.1.62        c001-n002-mic2 mic2
10.33.1.82        c001-n002-mic3 mic3
[root@c001-n002-mic0 ~]#
[root@c001-n002-mic0 ~]# ifconfig
...
mic0  Link encap:Ethernet  HWaddr 4C:79:BA:1A...
      inet addr:10.33.1.22  Bcast:10.33.1.255...
...
```

リスト 2: ホスト `c001-n002` と 4 つのインテル® Xeon Phi™ コプロセッサーの外部ブリッジ・ネットワーク構成。このネットワーク・トポロジーで構成されたリモートマシンのコプロセッサー間で TCP/IP パケットを直接送ることができます。この構成は、図 2 で示されているようにネイティブモード (オフロードなし) でコプロセッサーを利用するシンメトリック MPI アプリケーションに適しています。同時に、外部ブリッジで構成されたクラスターはオフロード・アプリケーションに使用することもできます。

2.2. InfiniBand*

インテル® Xeon Phi™ コプロセッサー向けインテル® MPI は、OpenFabrics Alliance (OFA) の OpenFabrics Enterprise Edition (OFED) 通信スタックをビルトインサポートしています。OFED を利用することで、InfiniBand* ファブ

リック上で Remote Direct Memory Access (RDMA) 転送を活用してマイクロ秒のレイテンシーと高い帯域幅の通信チャネルを提供できます。

インテル® Xeon Phi™ コプロセッサーを搭載したクラスターの RDMA 転送には 2 つのルートがあります。

1) 単一プラットフォームのノード内 (計算ノード内) 通信用の *scif0* 仮想 InfiniBand* アダプター。このメカニズムは、(1 つのノードに属する) ローカル・コプロセッサー間、およびホストと 1 つのローカル・コプロセッサー間の PCIe バス上の通信を抽象化します。

2) コプロセッサー通信リンク (CCL) – インテル® Xeon Phi™ コプロセッサーからハードウェア InfiniBand* ホスト・チャネル・アダプター (HCA) への直接アクセスを仮想化してノード間 RDMA を可能にするプロキシードライバー (図 4 を参照)。

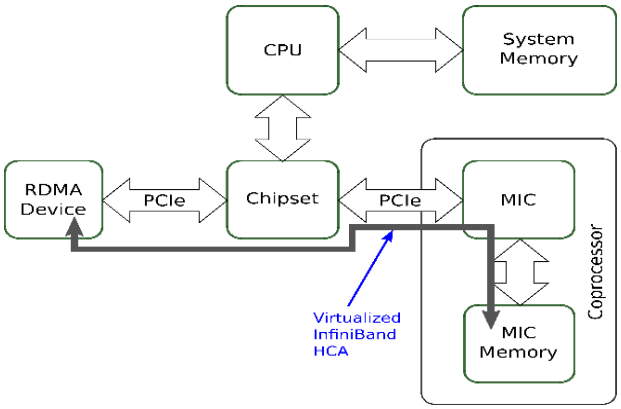

図4 : RDMA 転送とシンメトリック・ヘテロジニアス・クラスター内部のインテル® Xeon Phi™ コプロセッサー間の CCL。[4] に基づくダイアグラム。

インテル® Xeon Phi™ コプロセッサーと InfiniBand* HCA を搭載したシステムの OFED 構成をリスト 3 に示します。リストには、ホスト上の実際の HCA `mlx4_0`、コプロセッサー上の仮想化された HCA `mlx4_0`、ホストとコプロセッサー上の仮想アダプター `scif0` が示されています。

```
[root@c001-n002 ~]# # We are on host c001-n002
[root@c001-n002 ~]# service openibd status

  HCA driver loaded
...
[root@c001-n002 ~]# service ofed-mic status
Status of OFED Stack:
host                                        [  OK  ]
mic0 mic1 mic2 mic3                         [  OK  ]
[root@c001-n002 ~]# ibv_devices
    device                 node GUID
    ------              ----------------
    scif0               4c79bafffe1a0ec9
    mlx4_0              f45214030012ecf0
[root@c001-n002 ~]#
[root@c001-n002 ~]# ssh mic0
[root@c001-n002 ~]# # Now we are on mic0
[root@c001-n002-mic0 ~]# ibv_devices
    device                 node GUID
    ------              ----------------
    mlx4_0              f45214030012ecf0
    scif0               4c79bafffe1a0ec8
[root@c001-n002-mic0 ~]#
```

リスト 3: 4 つのインテル® Xeon Phi™ コプロセッサーと Mellanox InfiniBand* HCA を搭載したシステムの InfiniBand*。サービス `openibd` (OFED の一部) は、InfiniBand* HCA が同じ InfiniBand* ネットワークに接続されているほかのシステムと通信できるようにするモジュールをロードします。サービス `ofed-mic` (インテル® MPSS の一部) は、ローカル・コプロセッサー間の通信用に仮想 InfiniBand* インターフェイス `scif0` を作成し、コプロセッサー上で InfiniBand* HCA をデバイス `mlx4_0` として仮想化します。

クラスターのすべてのマシンでイーサネットと InfiniBand* ネットワークを構成すると、これらのネットワークは MPI アプリケーションによって検出されます。インテル® MPI は、開発者が通信ファブリックとプロトコルのハードウェアおよびソフトウェア実装とコードを区別できるように抽象化を提供します。

セクション 3 では、MIC アーキテクチャー・ベースのシンメトリック・ヘテロジニアス・クラスターでネットワークをセットアップおよび構成する方法を説明します。セクション 4 では、ネットワークを構成したクラスターで MPI アプリケーションを実行する方法を示します。

3. ヘテロジニアス・クラスターの構成

このセクションでは、シンメトリック・ヘテロジニアス MPI ワークロード用にインテル® Xeon Phi™ コプロセッサーと InfiniBand* インターコネクトを搭載したクラスターを構成する方法を説明します。ここでは、すべてのノードに CentOS 6.5 Linux* がインストールされ、イーサネット上に基本的なネットワークが構成されていると仮定しています。ここで説明するインテル® MPSS、ブリッジ接続されたネットワーク、計算ノードの OFED を手動構成するための手順は、クラスターのプロビジョニング・システムに適応できます。

手順の大部分は [5] で提案されている手法に基づいており、最新のインテル® MPSS バージョン 3.1.2 の機能を反映するように更新されています。

3.1. ネットワーク構成

クラスターの計算ノードのホスト名は `c001-n001`、`c001-n002`、... です。ホスト名の前半部分はクラスターのキャビネット番号を示し、後半部分はシャーシのインデックス番号を示します。このクラスターには 1 つのキャビネットしかありませんが、この命名規則を使用することで複数のキャビネットがある場合も容易に対応できます。

計算ノードのイーサネット・アダプターの IP アドレスは `10.33.C.N` (`C` はキャビネット番号、`N` はノード番号) で、ネットマスクは `255.254.0.0` です。対応するノードの InfiniBand* アダプターの IP アドレスは `10.34.C.N` で、ネットマスクは `255.254.0.0` です。イーサネットと InfiniBand* の IP アドレス範囲はオーバーラップしないことに注意してください。

インテル® Xeon Phi™ コプロセッサーのホスト名は、ホストシステムのホスト名にサフィックス `-micX` を追加したものです。例えば、最初のキャビネットの計算ノード番号 2 のホスト名は `c001-n002` で、対応する MIC カードは `c001-n002-mic0`、`c001-n002-mic1`、... です。IP アドレスは `10.33.C.M` (`M=N+(X+1)*20`、`X` はゼロから始まるコプロセッサーの番号) です。この規則では、シャーシあたり最大 11 のインテル® Xeon Phi™ コプロセッサー (最新のシステムでは最大 8 つまでサポート)、キャビネットあたり最大 19 の計算ノードを指定できます。

最初に、クラスターのすべてのマシンとインテル® Xeon Phi™ コプロセッサーの IP アドレスを含む `/etc/hosts` ファイルを作成し、ヘッドノードに配置します。インテル® MPSS は、このファイルを構成プロセス中に変更することがあるため、ヘッドノードにインテル® MPSS をインストールして構成する場合は、ホストファイルのバックアップ・コピーを保存しておいてください。

注: インテル® Xeon Phi™ コプロセッサーのホスト名は、ホストシステムのホスト名にサフィックス `-micX` を追加したものです。ホスト名が正しくない場合、インテル® MPI は正しく機能しません。

```
[root@head-node ~]# cat /etc/hosts
127.0.0.1   localhost
::1         localhost
10.32.0.1     head-node
10.33.1.1     c001-n001
10.33.1.21    c001-n001-mic0
10.33.1.41    c001-n001-mic1
10.33.1.61    c001-n001-mic2
10.33.1.81    c001-n001-mic3
10.33.1.2     c001-n002
10.33.1.22    c001-n002-mic0
10.33.1.42    c001-n002-mic1
10.33.1.62    c001-n002-mic2
10.33.1.82    c001-n002-mic3
```

リスト 4: マスター `/etc/hosts` ファイルを MPI ジョブが開始されるヘッドノードに配置。

フラット・ネットワーク・トポロジーを実装するには、ホストシステム上にネットワーク・ブリッジを構成する必要があります。ブリッジ構成では、システムのすべてのインテル® Xeon Phi™ コプロセッサーに割り当てられた IP アドレスをホスト NIC が処理します。この処理が行われるように、管理者はブリッジデバイス `br0` を作成し、イーサネット NIC `eth0` を接続する必要があります。これは、ネットワーク・ブリッジ経由で仮想マシン用の外部ネットワーク・アクセスを構成する手順に似ています。ブリッジ `br0` とイーサネット・インターフェイス `eth0` に対応するネットワーク構成ファイルは、`/etc/sysconfig/network-scripts` フォルダーに配置されます。リスト 5 は、ノード `c001-n001` の構成ファイルの例を示しています。

この構成ファイルは、DHCP による動的アドレス割り当てを使用していません。代わりに、IP アドレスとネットワーク・マスクの静的割り当てを使用しています (`BOOTPROTO=static`)。ホストシステムの IP アドレスとネットワーク・マスクは、ブリッ

ジ・インターフェイス br0 に割り当てます。ネットワーク・マネージャー・インターフェイスを無効にするには、すべてのインターフェイスで NM_CONTROLLED=no を指定します。ブート時にインターフェイスを自動的に有効にするには、ONBOOT=yes を指定します。これらのファイルを変更した後、サービス network を再起動します。サービス NetworkManager が有効な場合は無効にします。

```
[root@c001-n001 ~]# \
> cat /etc/sysconfig/network-scripts/ifcfg-br0
DEVICE=br0
TYPE=Bridge
BOOTPROTO=static
ONBOOT=yes
NM_CONTROLLED=no
IPADDR=10.33.1.1
NETMASK=255.254.0.0
[root@c001-n001 ~]#
[root@c001-n001 ~]# \
> cat /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
TYPE=Ethernet
BRIDGE=br0
BOOTPROTO=none
ONBOOT=yes
NM_CONTROLLED=no
HWADDR="00:E0:81:E4:84:EC"
```

リスト 5: 計算ノードのブリッジを構成。

この時点で、すべてのノードがそれぞれの IP アドレスでネットワークに接続され、ノード間の ping が成功しなければなりません。

インテル® Xeon Phi™ コプロセッサーを構成する準備として、ホストでパケット転送を有効にして (リスト 6 を参照)、1 つのシステム内のコプロセッサーが TCP プロトコルを使用して互いに通信できるようにします。

```
[root@c001-n001 ~]# echo 1 > \
> /proc/sys/net/ipv4/ip_forward
[root@c001-n001 ~]# cat /etc/sysctl.conf \
> | grep forward
# Controls IP packet forwarding
net.ipv4.ip_forward = 1
```

リスト 6: 計算ノードのパケット転送を構成。

コプロセッサーのネットワークはまだ構成されていません。これは、インテル® MPSS 構成 (セクション 3.4 を参照) で行います。

3.2. NFS 共有

NFS 共有を使用すると、計算ノードだけでなく、インテル® Xeon Phi™ コプロセッサーともファイルを共有できます。コプロセッサーとの NFS 共有は、次の 3 つの点で便利です。

1) /home を共有すると、~/.ssh/ の SSH キーが共有され、ヘッドノードから計算ノードおよび計算ノードのコプロセッサーに、パスワードがないアクセスを構成できます。

2) /home を共有すると、(コプロセッサーをネイティブモードで使用する場合) シンメトリック・ヘテロジニアス MPI 計算のセットアップが簡素化されます。

3) コプロセッサーと /opt/intel を共有すると、インテルのライブラリー (特に、インテル® マス・カーネル・ライブラリー (インテル® MKL)、MPI および OpenMP* ライブラリー) を簡単にインテル® Xeon Phi™ コプロセッサーへ転送できます。

コプロセッサーとのファイル共有を有効にするには、リスト 7 のように (NFS サーバーの) ヘッドノードの /etc/exports ファイルを変更します。ヘッドノードはサブネット 10.33.0.0/15 にあると仮定しています (ここでは、ヘッドノードの IP アドレスは 10.32.0.1 です)。

```
[root@head-node ~]# cat /etc/exports
/opt/intel 10.33.0.0/15(rw,no_root_squash)
/home 10.33.0.0/15(rw,no_root_squash)
[root@head-node ~]# exportfs -ra
[root@head-node ~]# mkdir /opt/mic
[root@head-node ~]# ln -s /opt/mic /opt/intel/mic
```

リスト 7: /home と /opt/intel を共有するようにヘッドノードの NFS サーバーを構成。

/opt/intel のマウント中にインテル® MPSS 構成の不足機能を補うため、ヘッドノード (およびすべての計算ノード) にローカル・ディレクトリー /opt/mic を作成し、シンボリック・リンク /opt/intel/mic が /opt/mic を指すようにする必要があります。計算ノード固有の一部の構成ファイルは /opt/intel/mic にインストールされるため、このシンボリック・リンクがないと、異なるノードが互いの構成を上書き

してしまいます。シンボリック・リンクが共有ディレクトリー /opt/intel でない場所を指すようにすることで、各ノードの構成を一意に保つことができます。

次に、サーバーから /opt/intel と /home をマウントするため、計算ノードの /etc/fstab に次の行を追加します。

```
[root@c001-n002 ~]# tail -f /etc/fstab
head-node:/home      /home      nfs defaults 1 2
head-node:/opt/intel /opt/intel nfs defaults 1 2
```

リスト 8: /home と /opt/intel をマウントするように NFS クライアントを構成。

別の構成は、ヘッドノードから NFS マウントする代わりに、計算ノードにインテル® Cluster Studio をインストールします。ローカルにインストールされたインテル® Cluster Studio では、コプロセッサーに /opt/intel を提供する NFS サーバーの役割はローカルの計算ノードが行います。つまり、各計算ノードが NFS サーバーとして構成され、計算ノードの /etc/exports が /opt/intel を 10.33.C.N/24 と共有する必要があります (この IP アドレスの説明はセクション 3.1 を参照)。この場合、/opt/intel/mic から /opt/mic を指すシンボリック・リンクを作成する必要はありません。

標準 NFS サーバー、クライアント構成、セキュリティーの詳細は、このホワイトペーパーでは説明しません。

3.3. アカウント管理

インテル® Xeon Phi™ コプロセッサーは、ホストシステムのユーザー権限ポリシーを継承します。このため、特定のユーザーアカウントでアプリケーションを実行するには、アカウントを作成して対応する SSH キーをセットアップする必要があります (リスト 9 を参照)。

/home がヘッドノードとすべての計算ノードで NFS 共有されていると仮定して、ヘッドノードに各ユーザー用の SSH キーを一度だけ作成します。/home を NFS 共有しているすべてのノードとコプロセッサーでパスワードなしの認証ができるように、公開 SSH キーを authorized_keys ファイルにコピーします。

コプロセッサーへのアクセスに SSH キー認証を使用し、パスワードを入力する必要がない場合でも、インテル® Xeon Phi™ コプロセッサーにアクセスするすべてのユーザーに、空でないパスワードを設定しなければいけないことに注意してください。このパスワードはコプロセッサーでの認証に必要です。

```
[root@c001-n001 ~]# adduser cfxuser
[root@c001-n001 ~]# passwd cfxuser
...
[root@c001-n002 ~]# adduser cfxuser
[root@c001-n002 ~]# passwd cfxuser
...
[root@head-node ~]# adduser cfxuser
[root@head-node ~]# passwd cfxuser
[root@head-node ~]# su cfxuser
[cfxuser@head-node ~]# ssh-keygen
...
[cfxuser@head-node ~]# cat \
> .ssh/id_rsa.pub >> .ssh/authorized_keys
[cfxuser@head-node ~]# chmod \
> 600 ~/.ssh/authorized_keys
```

リスト 9: 計算ノードのユーザーとヘッドノードのキーを構成。

また、各ユーザーの数値ユーザー ID とグループ ID がヘッドノードおよび計算ノードで同一であることを保証しなければなりません (必要な場合は adduser の --uid と --gid 引数を使用します)。同一でない場合、NFS は正しく機能しません。

3.4. インテル® MPSS 構成

次に、インテル® Xeon Phi™ コプロセッサー向けにインテル® MPSS をインストールして構成します。インテル® MPSS バージョン 3.1.2 の場合はインテル® レジストレーション・センターからアーカイブをダウンロードし (以前のバージョンの場合は (http://software.intel.com/en-us/articles/intel-manycore-platform-software-stack-mpss-archive) からアーカイブをダウンロードし)、RPM をアンパックした後、yum を使用してインストールします (リスト 10 を参照)。

```
[root@c001-n001 ~]# tar -xf \
>                   mpss-3.1.2-rhel-6.5.tar
[root@c001-n001 ~]# cd mpss-3.1.2/
[root@c001-n001 ~]# yum install *.rpm
```

リスト 10: インテル® MPSS のインストール。

このシステム構成では、インストール後に計算ノードで

Linux* ディストリビューションの更新を無効にして、インストールされているカーネルのバージョンがインテル® MPSS がコンパイルされたバージョンと同じになるように保証しています。インストールされているカーネルのバージョンがインテル® MPSS がコンパイルされたバージョンと異なる場合、インテル® MPSS カーネルモジュールと OFED モジュールを再コンパイルする必要があります。再コンパイル手順は、インテル® MPSS ユーザーガイド [6] のセクション 8 で説明されています。

インテル® MPSS の構成には、micctrl ツールを使用します。micctrl ツールにより、コマンドライン引数でインテル® Xeon Phi™ コプロセッサー の *µOS* の構成を管理できます。リスト 11 は、インテル® Xeon Phi™ コプロセッサーの構成ファイルの初期化 (ブリッジ接続の構成、ネットワーク構成のセットアップ、ホストシステムであらかじめ構成されている NFS フォルダーの追加) を示しています。

```
[root@c001-n001]# micctrl --initdefaults
...
[root@c001-n001]# micctrl --addbridge=br0 \
> --type=external --ip=10.33.1.1 --netbits=15
[root@c001-n001]# micctrl --network=static \
> --bridge=br0 \
> --ip=10.33.1.21:10.33.1.41:10.33.1.61:\
>10.33.1.81

[root@c001-n001]# micctrl --addnfs=/opt/intel \
>                 --dir=/opt/intel
[root@c001-n001]# micctrl --addnfs=/home \
>                 --dir=/home
```

リスト 11: インテル® MPSS (ブリッジと NFS) の構成。

インテル® MPSS 3.1.2 では、micctrl --addnfs で設定されるコプロセッサーの NFS サーバーは常に計算ノードであり、コマンドラインで異なる NFS サーバーを指定することはできません。このため、/home と /opt/intel を計算ノードからではなくヘッドノードからマウントしなければいけない場合、管理者は /var/mpss/mic*/etc/fstab ファイルを手動で編集して NFS サーバーの場所を正しい IP アドレスに変更する必要があります。例えば、リスト 12 は、/home をヘッドノード (10.32.0.1) から、/opt/intel をローカルの計算ノード (10.33.1.1) からマウントする方法を示しています。

```
[root@c001-n001]# cat /var/mpss/mic0/etc/fstab \
          | tail -2
10.32.0.1:/home      /home      nfs nolock 1 1
10.33.1.1:/opt/intel /opt/intel nfs nolock 1 1
```

リスト 12: 変更後の /etc/fstab は、ローカルの計算ノードの代わりにヘッドノードから /home をマウント。インテル® MPSS 3.1.2 では、この変更は手動で行う必要があります。

これでインテル® MPSS を開始する準備がすべて整いました。すべてのデバイスでインテル® MPSS を開始すると、クラスターのホストまたはコプロセッサーからすべてのホストとすべてのコプロセッサーに ping できます。

```
[root@head-node]# ssh c001-n001
[root@c001-n001]# service mpss start
Starting Intel(R) MPSS:                [  OK  ]
mic0: online (mode: linux image: /usr/share/...
mic1: online (mode: linux image: /usr/share/...
mic2: online (mode: linux image: /usr/share/...
mic3: online (mode: linux image: /usr/share/...
[root@c001-n001]# exit
[root@head-node]# ping -q -c 1 c001-n001
PING c001-n001 (10.33.1.1) 56(84) bytes of data.

--- c001-n001 ping statistics ---
1 packets transmitted, 1 received ...
rtt min/avg/max/mdev = 0.330/0.330/0.330/0.000ms
[root@head-node]#
[root@head-node]# ping -c 1 c001-n001-mic0
PING c001-n001-mic0 (10.33.1.21) 56(84) bytes...

--- c001-n001-mic0 ping statistics ---
1 packets transmitted, 1 received ...
rtt min/avg/max/mdev = 0.714/0.714/0.714/0.000ms
[root@head-node]#
[root@head-node]# ssh c001-n002
[root@c001-n002]# service mpss start
...
```

リスト 13: 計算ノードでインテル® MPSS を開始してコプロセッサーのブリッジ・ネットワークを確認。

3.5. InfiniBand* 構成

インテル® MIC アーキテクチャー向けインテル® MPI の InfiniBand* サポートをインストールして構成するには、すべての計算ノードに OFED がインストールされている必要があります。インテル® MPSS 3.1.2、CentOS 6.5 Linux*、Mellanox HCA の場合 (scif0 仮想ネットワークのノード内通信の場合は HCA なし)、OFA Web サイトから OFED バージョン 1.5.4.1 をダウンロードしてインストールします。パッケージをインストールするには、Perl インストール・スクリプト install.pl を実行します。インストール中、ライブラ

リー dapl*、compat-dapl*、32 ビット・ライブラリー・サポートのインストールを "No" にします。リスト 14 のように、OFED をインストールした後、mpss-3.1.2/ofed サブディレクトリーから追加のインテル® MPSS パッケージをインストールします。Linux* カーネルのバージョンがインテル® MPSS OFED パッケージがコンパイルされたカーネルのバージョンと異なる場合、インストールの前に OFED モジュールを再コンパイルする必要があります。詳細は、インテル® MPSS ユーザーガイド [6] のセクション 2.3 を参照してください。

インテル® MPSS ユーザーガイドでは特に明記されていませんが、構成プロセスのこの時点でシステムを再起動することを推奨します。

```
[root@c001-n001 ~]# cd OFED-1.5.4.1
[root@c001-n001 OFED-1.5.4.1]# ./install.pl
...
[root@c001-n001 OFED-1.5.4.1]# cd ../mpss-3.1.2
[root@c001-n001 mpss-3.1.2]# yum install\
> --disablerepo=* --skip-broken ofed/*.rpm
...
```

リスト 14: ソースとインテル® MPSS OFED パッケージから RPM として OFED 1.5.4.1 をインストール。

OFED をインストールした後、リスト 15 の操作を行います。

1) InfiniBand* 接続のスタティック IP アドレスを構成します。

2) すべての計算ノードでサービス openibd を開始します。

3) 1 つの計算ノードでサービス opensmd を開始します (ただし、ネットワーク・スイッチがサブネット・マネージャーをすでに提供している場合は、計算ノードでこのサービスを開始しないでください)。

4) サービス ofed-mic を開始して CCL を有効にします (ホストとコプロセッサーで仮想ノード内インターフェイス scif0 とノード間インターフェイス mlx4_0 が作成されます)。

5) ibstatus を実行して InfiniBand* デバイスのステータスを確認します。ノード内インターフェイスとノード間インターフェイスが存在し、ステータスが "ACTIVE" になっていなければいけません。

インテル® Xeon Phi™ コプロセッサー向け InfiniBand* サポートが開始していることを保証するため、コプロセッサーの 1 つにログインして ibv_devinfo を実行します (リスト 16)。ノード内インターフェイス scif0 とノード間インターフェイス mlx4_0 が PORT_ACTIVE ステートになっていることを確認します。

4. MPI パフォーマンスの測定

パフォーマンスの測定は、それぞれ 4 つのインテル® Xeon Phi™ コプロセッサー 31S1P を搭載した、2 つの Colfax ProEdge* SXP8600p ワークステーションからなるクラスターで行われました。クラスター内のネットワークには 2 つのインターコネクトが使用されました。

```
[root@c001-n001 ~]# cat \
> /etc/sysconfig/network-scripts/ifcfg-ib0
DEVICE=ib0
HWADDR=80:00:00:48:FE:80:00:00:00:00:00...
TYPE=InfiniBand
UUID=43a82878-51ee-427d-bec2-312d6b868508
ONBOOT=yes
NM_CONTROLLED=no
BOOTPROTO=static
IPADDR=10.34.1.1
NETMASK=255.254.0.0
[root@c001-n001 ~]# service network restart
[root@c001-n001 ~]# service openibd start
[root@c001-n001 ~]# service ofed-mic start
[root@c001-n001 ~]# ibstatus
Infiniband device 'mlx4_0' port 1 status:
        default gid:  fe80:0000:0000:...
        base lid:     0x1
        sm lid:               0x1
        state:               4: ACTIVE
        phys state:   5: LinkUp
        rate:               40 Gb/sec (4X QDR)
        link_layer:    InfiniBand

Infiniband device 'scif0' port 1 status:
        default gid:  fe80:0000:0000:...
        base lid:     0x3e8
        sm lid:               0x1
        state:        4: ACTIVE
        phys state:   5: LinkUp
        rate:         40 Gb/sec (4X QDR)
        link_layer:   Ethernet

[root@c001-n001 ~]#
```

リスト 15: InfiniBand* サービスの開始/チェック。

1. インテル® ギガビット・イーサネット・アダプター (モデル I350); D-Link ギガビット・イーサネット・スイッチ (モデル DGS-1024D) に接続。

2. Mellanox InfiniBand* ConnectX-3 Single-Port VPI 4X QDR アダプター (モデルバージョン MCX353A-FCAT); 36 ポートの Mellanox Infiniscale IV スイッチ (モデル IS5025) に接続。

CentOS 6.5 Linux* オペレーティング・システム (カーネル 2.6.32-431.el6.x86_64) でインテル® MPSS 3.1.2、インテル® MPI 4.1.1.036、OFED 1.5.4.1 で、インテル® MPI に含まれるインテル® MPI Benchmark (IMB) を使用してパフォーマンスを測定しました。

```
[root@c001-n001 ~]# ssh mic0
[root@c001-n001-mic0 ~]# ibv_devinfo
hca_id:         mlx4_0
  transport:        InfiniBand (0)
  fw_ver:           2.30.8000
  node_guid:        f452:1403:0012:ed70
  sys_image_guid:   f452:1403:0012:ed73
  vendor_id:        0x02c9
  vendor_part_id:   4099
  hw_ver:           0x0
  phys_port_cnt:    1
    port:   1
      state:        PORT_ACTIVE (4)
      max_mtu:      2048 (4)
      active_mtu:   2048 (4)
      sm_lid:       1
      port_lid:     1
      port_lmc:     0x00
      link_layer:   IB

hca_id:   scif0
  transport:        SCIF (2)
  fw_ver:           0.0.1
  node_guid:        460f:d8ff:fe1d:8294
  sys_image_guid:   460f:d8ff:fe1d:8294
  vendor_id:        0x8086
  vendor_part_id:   0
  hw_ver:           0x1
  phys_port_cnt:    1
    port:   1
      state:        PORT_ACTIVE (4)
      max_mtu:      4096 (5)
      active_mtu:   4096 (5)
      sm_lid:       1
      port_lid:     1001
      port_lmc:     0x00
      link_layer:   SCIF
```

リスト 16: インテル® Xeon Phi™ コプロセッサーでノード内インターフェイス `scif0` とノード間インターフェイス `mlx4_0` が仮想化される。

4.1. イーサネットでのパフォーマンス

最初に、InfiniBand* インターコネクトが搭載されていないため、MPI 通信をイーサネット上で行うシナリオを考えます。このシナリオは、環境変数 `I_MPI_FABRICS=tcp` を設定して (インテル® MPI がイーサネット上の TCP/IP プロトコルを使用して通信するように指定して) エミュレートできます。

リスト 17 は、ベンチマークの起動手順を示しています。ベンチマーク・アプリケーションの出力から、レイテンシーと帯域幅が分かります。

```
[cfxuser@head-node ~]# export I_MPI_MIC=1
[cfxuser@head-node ~]# export I_MPI_FABRICS=tcp
[cfxuser@head-node ~]# export IMBHOST=\
>${I_MPI_ROOT}/bin64/IMB-MPI1
[cfxuser@head-node ~]# export IMBMIC=\
>${I_MPI_ROOT}/mic/bin/IMB-MPI1
[cfxuser@head-node ~]# mpirun \
> -np 1 -host c001-n001 ${IMBHOST} PingPong :\
> -np 1 -host c001-n002-mic0 ${IMBMIC}
...
```

リスト 17: イーサネット・ファブリックでのインテル® MPI Benchmark の実行。この例で、PingPong エンドポイントの一方はノード 1 の CPU、もう一方はノード 2 のコプロセッサー 0 です。

図 5 は、TCP プロトコルを使用した場合の PingPong ベンチマークのパフォーマンスを示しています。ノード内通信では、MPI は仮想化されたイーサネット NIC `mic0`、`mic1`、… をデータ転送に使用します。ノード間通信 (ホストからリモートホストおよびリモート・コプロセッサー) では、`eth0` と `mic0`、`mic1`、… のブリッジ接続が使用されます。

ブリッジからコプロセッサーに送られるすべての通信で、ショートメッセージのレイテンシーは 300 ～ 500 マイクロ秒です。直接接続 (CPU とローカル・コプロセッサー間、および 2 つの CPU 間) のレイテンシーは 50 ～ 100 マイクロ秒です。ロングメッセージの帯域幅は、ホスト間接続ではギガビット・イーサネット・ネットワークのハードウェアの上限とほぼ一致する約 110MB/秒に達しています。しかし、コプロセッサー関連の通信の帯域幅は 20 ～ 25MB/秒に過ぎません。これは、ネットワーク・インターコネクトや PICe バスのハードウェア制限よりも一桁小さな値です。[3] で説明されているように、イーサネット上のシンメトリック・ヘテロジニアス・クラスタリングの利便性は通信効率を犠牲にして得られています。

しかし、これは InfiniBand* ファブリックでは問題になりません。

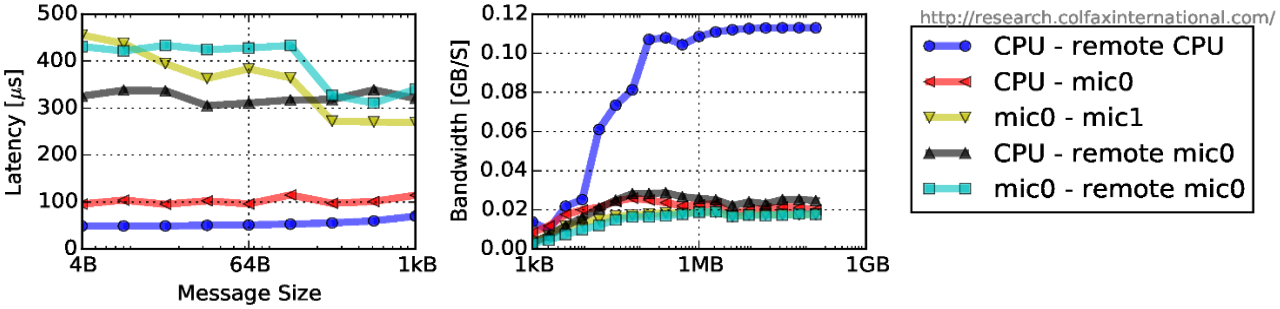

図5 : インテル® MPI Benchmark、PingPong をイーサネット・ファブリック上でテスト (`I_MPI_FABRICS=tcp`)。CPU はホスト上で実行される MPI プロセスを示し、`mic*` はコプロセッサーで実行される MPI プロセスを示します。"remote" は異なるシャーシにあることを意味します。

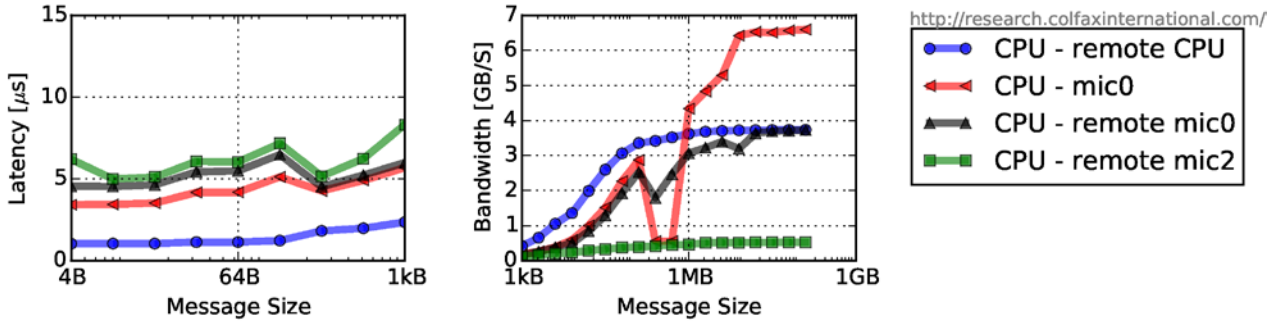

図6 : インテル® MPI Benchmark、PingPong を InfiniBand* ファブリック上でテスト (`I_MPI_FABRICS=dapl`)。CPU ホストで開始する通信。図 5 の説明も参照してください。

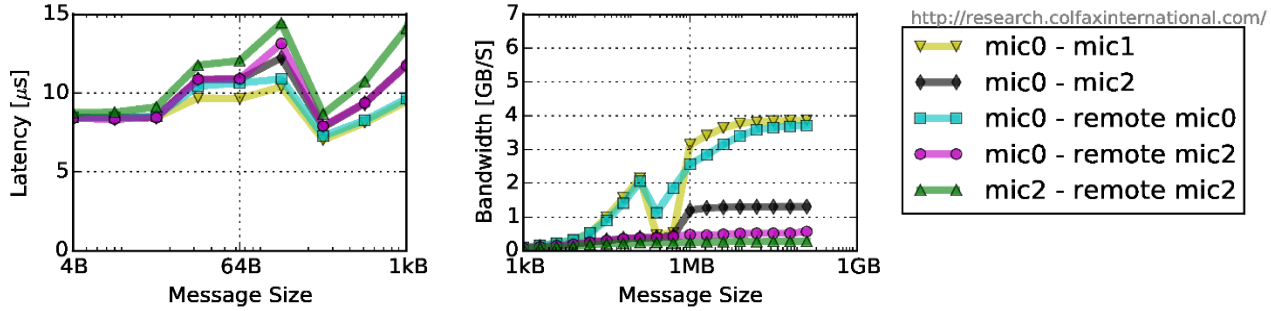

図7 : インテル® MPI Benchmark、PingPong を InfiniBand* ファブリック上でテスト (`I_MPI_FABRICS=dapl`)。MIC ホストで開始する通信。図 5 の説明も参照してください。

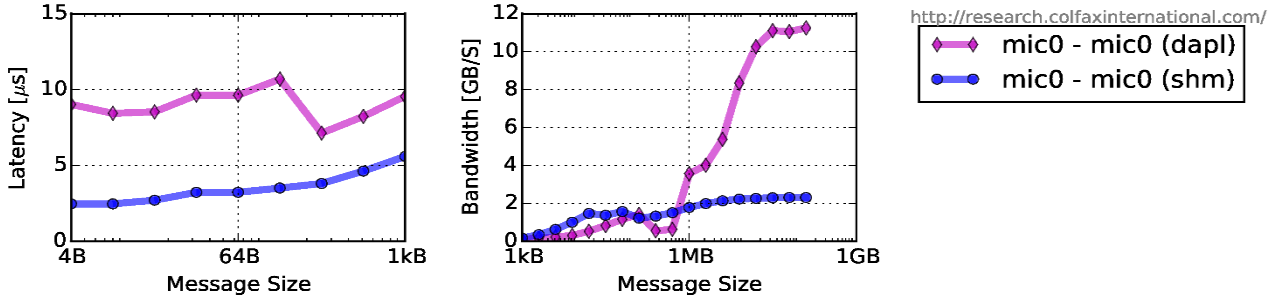

図8 : インテル® MPI Benchmark、PingPong を共有メモリーおよび InfiniBand* ファブリック上でテスト (`I_MPI_FABRICS=shm` および `I_MPI_FABRICS=shm:dapl`)。MIC ホスト内の通信。図 5 の説明も参照してください。

4.2. InfiniBand* でのパフォーマンス

次に、InfiniBand* ファブリック上の MPI 通信のベンチマーク・テストを行います。インテル® MPI は、このファブリックを自動的に検出して使用します。追加の構成を行う必要はありません。しかし、自動構成に失敗した場合や、ほかのファブリックが使用されないようにする場合は、環境変数 I_MPI_FABRICS=dapl を設定し、DAPL プロトコルによる通信を要求できます。

ここでは、3 つの通信 (ホストからのメッセージ、インテル® Xeon Phi™ コプロセッサー間のメッセージ、単一インテル® Xeon Phi™ コプロセッサー内のメッセージ) のパフォーマンス結果を説明します。

4.2.1. ホストからのメッセージ

図 6 は、ホスト CPU からリモート CPU、ローカル・コプロセッサー、リモート・コプロセッサーへの PingPong テストのレイテンシーと帯域幅を示しています。明らかに、イーサネット上で TCP プロトコルを使用した場合よりも優れた値になっています。

ショートメッセージの通信レイテンシーは 1.5 ～ 8 マイクロ秒に短縮されています。これは、イーサネット上で TCP プロトコルを使用した場合よりも二桁、オフロード・プログラミング・モデルのコプロセッサーにコントロールを転送する時間よりも一桁優れています (オフロード・パフォーマンスの測定については [7] を参照)。

CPU からローカル・コプロセッサーへのロングメッセージの帯域幅は、PCIe v2.0 x16 デバイスの PCIe バス帯域幅の上限に近い 6.5GB/秒に達しています。ホストからリモートホストおよびリモート・コプロセッサーの帯域幅は、4X QDR InfiniBand* インターコネクトの上限である 3.8GB/秒に達しています。

ホストからローカル・コプロセッサー mic1、mic2、mic3、およびリモート mic0、mic1 への通信の帯域幅とレイテンシーはすべて、ホストから mic0 への値に非常に近いため、図では示されていません。

しかし、ホストからリモート mic2 および mic3 へのメッセージでは、帯域幅が 0.5GB/秒に低下します。これは、InfiniBand* HCA がすべてのホストの CPU1 にインストールされ、コプロセッサー mic2 と mic3 は CPU2 にインストールされていることが原因です。

4.2.2. コプロセッサー間のメッセージ

図 7 mic0 からほかのローカルおよびリモート・コプロセッサーへの MPI メッセージのレイテンシーと帯域幅を示しています。すべてのケースでレイテンシーはほぼ同じ (10 マイクロ秒前後) ですが、帯域幅は通信デバイスのペアに依存します。

ノード内通信の場合、次のことが言えます。

i) mic0 からローカル mic1 (または mic2 からローカル mic3、図では示されていません) の帯域幅はほぼ 4GB/秒に達しています。

ii) しかし、mic0 からローカル mic2 (または mic3、図では示されていません) の帯域幅は 1.3GB/秒に過ぎません。これは、PCIe バス上のコプロセッサーの位置が原因です。mic0 と mic1 は CPU1 でコントロールされる PCIe スロットに接続されていますが、mic2 と mic3 は CPU2 でコントロールされる PCIe スロットに接続されています。エンドポイント・デバイス間の PCIe 転送は、同じ CPU でコントロールされるドメイン内よりも、インテル® QuickPath インターコネクト (QPI) で接続される CPU のほうが遅くなります。

同様に、ノード間通信では、InfiniBand* HCA が CPU1 の PCIe スロットに接続されているため、これによって通信の効率が決まります。

i) mic0 とリモート (つまり、別の計算ノードの) mic1 や mic0 (図では示されていません) 間の通信では、帯域幅は InfiniBand* 接続の上限である 4GB/秒に近くなります。

ii) しかし、mic0 からリモート mic2 では、データをリモート HCA (CPU1) から同じマシンの mic2 (CPU2) に転送するため、帯域幅は 0.4GB/秒まで低下します。

iii) mic2 からリモート mic2 (または mic3、図では示されていません) では帯域幅がさらに低下し、約 0.25GB/秒になります。この場合、データはまずローカル PCIe バスを介して CPU1 に送られ、次に InfiniBand* ネットワークを通り、その後そのマシンのリモート HCA から CPU2 に送られます。

4.2.3. コプロセッサー内のメッセージ

特別なケースは、単一コプロセッサー内の複数の MPI プロセス間の通信です。デフォルトでは、MPI は通信プロトコルを組み合わせて使用します (I_MPI_FABRICS=shm:dapl を設定した場合と同じ)。この設定では、同じ計算デバイス (CPU またはコプロセッサー) 内の MPI ランク間の通信には共有メモリーのコピーを使用し、デバイス外部との通信には DAPL を使用します。

しかし、図 8 から分かるように、共有メモリーのコピーのみ、あるいは DAPL のみを使用するほうが良い場合もあります。実際、ショートメッセージのレイテンシーは shm プロトコルのほうが低く (2 マイクロ秒、dapl は 10 マイクロ秒)、ロングメッセージの帯域幅は dapl プロトコルのほうが高くなっています (最大 11GB/秒、shm は 2GB/秒)。図 8 の結果は、I_MPI_FABRICS=shm および I_MPI_FABRICS=dapl を指定して IMB を実行して得られたものです。

4.3. ファブリックのチューニング

DAPL プロトコルは、異なるメッセージサイズとファブリック用にチューニングされた、OFED スタックの複数のライブラリーでサポートされています。インテル® MPI は、ランタイムに検出したシステム構成に応じて、使用する DAPL のプロバイダーを自動的に決定します。選択された DAPL プロバイダーは変更することができます。

このシステムでは、MPI は通信に 3 つの DAPL プロバイダーを選択しました (ショートメッセージに ofa-v2-mlx4_0-1、計算ノード内のロングメッセージに ofa-v2-scif0、計算ノード外のロングメッセージに ofa-v2-mcm-1)。この選択は、IMB で環境変数 I_MPI_DEBUG=2 (または 3 以上) を設定して診断することができます。この選択を変更するには、環境変数 I_MPI_DAPL_PROVIDER_LIST を使用します。この変数には、最大 3 つのプロバイダー (ショートメッセージ、ノード内ロングメッセージ、ノード間ロングメッセージ) からなるカンマ区切りリストを指定できます。詳細は、James Tullos 氏の記事を参照してください。

サポートしている DAPL プロバイダーの詳細リストは、/etc/dat.conf ファイルに含まれています。

メッセージが "ショート" か "ロング" か区別するしきい値は、環境変数 I_MPI_DAPL_DIRECT_COPY_THRESHOLD で指定します。

このシステムでは、インテル® MPI ライブラリーによって選択された DAPL プロバイダーが最適であることが確認できました。

4.4. 集合通信のチューニング

並列転送関数 (MPI_SendRecv など) および MPI 集合通信関数 (MPI_Allgather など) は、クラスター内のデータ移動に構成可能なアルゴリズムを使用しています。特定のマシンのファイルで最適なパフォーマンスが得られるように、ユーティリティー mpitune は最適なアルゴリズムを見つけることができます。このユーティリティーは、インテル® MPI の一部です。mpitune によりチューニングされるパラメーターは、ファイル $I_MPI_ROOT/etc64/options.xml にリストされています。mpitune の使用方法は [2] を参照してください。

4.5. ヘテロジニアス・クラスターの MP パフォーマンスのまとめ

測定結果をまとめたものを図 9 に示します。この図で、デバイスは角丸長方形で、MPI 通信の効率は線で表されています。線の太さはロングメッセージの帯域幅に比例しており、線上の数値は、ロングメッセージの帯域幅とショートメッセージのレイテンシーです。

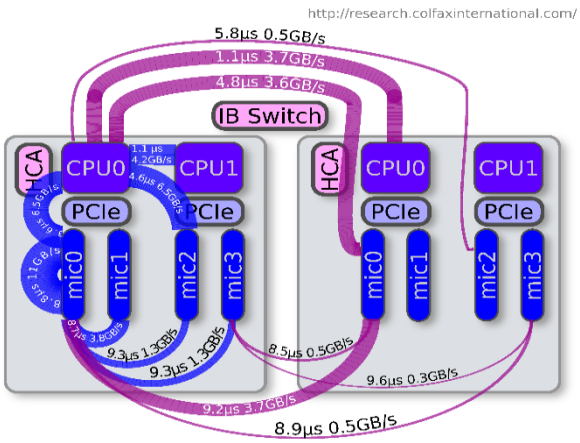

図9 : テストベンチ・クラスターで DAPL メッセージを使用した場合のショート MPI メッセージのレイテンシーとロング MPI メッセージの帯域幅。線の太さは帯域幅に比例しています。

5. ケーススタディー: エイジアンオプション

InfiniBand* インターコネクトによってもたらされる MPI 通信のレイテンシーと帯域幅の向上は、データ転送を行う並列アプリケーションで利点となります。この利点を説明するため、InfiniBand* クラスターで、[3] のエイジアンオプション価格評価アプリケーションのベンチマーク・テストを行いました。

このアプリケーションは、ボス・ワーカー・モデルを使用し、複数の計算デバイス (CPU とインテル® Xeon Phi™ コプロセッサー) に、異なるオプション・パラメーターのセットの計算を分散するモンテカルロ・コードです。ボスプロセスからワーカープロセスに送られる入力パラメーターは、ドリフトレート、原株資産の変動性、権利行使価格で、ワーカーにより返される出力は、モンテカルロ法で計算されたディスカウント・ペイオフです。各メッセージで交換されるデータの量が非常に小さいため (32 バイト)、このアプリケーションのパフォーマンスはネットワークのレイテンシーに左右されます。結果が分かりやすくなるように、今回はワーカープロセスをインテル® Xeon Phi™ コプロセッサーにのみ配置しました。

[3] では、ボスプロセスにより分散されるワークアイテムが大きなケースを取り上げたため、通信時間は重要ではありませんでした。ここでは、非常に高速に処理される、ワークアイテムが小さなケースについて考えてみます。ワークアイテムのサイズ (乱数値の量で測定) の範囲は $W=2^{16}$ から $W=2^{24}$ で、各ワークアイテムの計算時間の範囲は数マイクロ秒から数ミリ秒です。$W=M \times N$ となり、$M=2^{15}$ は希望する精度を達成するために必要なモンテカルロ・パスの数で、$N \in 2^1, 2^4, 2^9$ はエイジアンオプション定義ごとの価格を平均する期間です。パフォーマンスを向上させるため、MPI+OpenMP* のハイブリッド・アプローチを使用し、各インテル® Xeon Phi™ コプロセッサーに 7 つの 32 スレッドのワーカーをセットアップしました。プロセスあたり 228 スレッドのセットアップと比較すると、このアプローチではスレッドの同期オーバーヘッドが小さくなります。228 という数は、インテル® Xeon Phi™ 31S1P コプロセッサー (57 コア、4 ウェイ・ハイパースレッディング) の論理コアの数です。

図 10 は、さまざまな数の計算デバイスで、一秒ごとに生成され使用される乱数の量を測定して得られた、このモンテカルロ・アプリケーションのパフォーマンスを示しています。このクラスターでは、ノードあたりのインテル® Xeon Phi™ コプロセッサーは 4 つであるため、6 および 8 コプロセッサーのケースでは 2 つの計算ノードを使用しています。

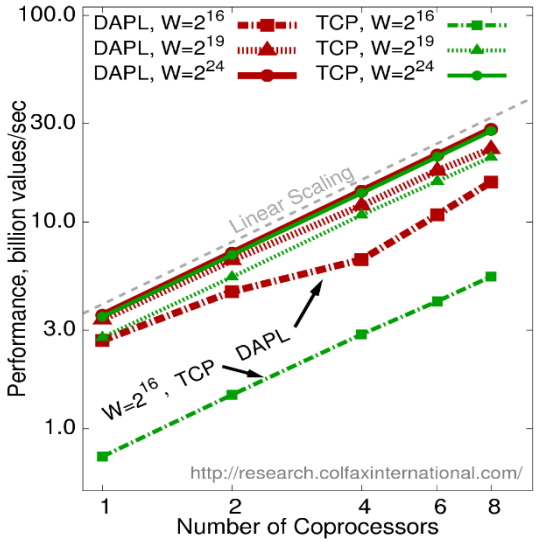

図 10: イーサネットと InfiniBand* ファブリックで [3] のエイジアンオプション価格評価を計算。ワークアイテムのサイズ W が小さいため、アプリケーションのパフォーマンスはネットワークのレイテンシーに左右されます。

図 10 に示すように、ワークアイテムのサイズが大きい場合、イーサネット ("TCP") と InfiniBand* ("DAPL") でアプリケーションのパフォーマンスは変わりません。しかし、より少ないワークアイテムでは、各ワーカーがボスプロセスからのスケジューリング・コマンドを待つときに、アイドル状態で待機する時間が長くなるため、どちらのファブリックもパフォーマンスが低下します。ワークアイテムのサイズが小さくなるとともに、イーサネットのパフォーマンスは InfiniBand* よりも大きく低下します。これは、InfiniBand* のレイテンシーのほうが低いことと一致します。

アプリケーションのソースコードは [3] で取得できますが、図 10 の結果を再現するには、M と N の値およびマシンファイルを変更する必要があります。

6. まとめ

TCP プロトコルによる MPI 通信は、クラスターの CPU ホスト間の通信でのみ、ギガビット・イーサネットのネットワーク帯域幅に達します。これは、インテル® Xeon Phi™ コプロセッサー向けのオフロードモデル (MPI ランクを CPU にのみ配置) を使用するアプリケーションでは問題ありません。しかし、MPI ランクをコプロセッサーにネイティブに配置するヘテロジニアス MPI アプリケーションでは、イーサネット上のピア・ツー・ピア通信はハードウェア制限よりも大幅に遅くなります。

このホワイトペーパーでは、インテル® MPSS に加えて、InfiniBand* コントローラーと関連ソフトウェアをインストールすることで、クラスターのホストとコプロセッサー間の MPI 通信が大幅に向上することを実証しました。通信には、CPU とコプロセッサー間のノード内通信、コプロセッサー間のノード内通信、両デバイス間のノード間通信が含まれます。また、CentOS 6.5 Linux*、インテル® MPSS 3.1.2、OFED 1.5.4.1、Mellanox インターコネクトでネットワークとソフトウェア構成を再現するために必要なステップを紹介しました。

`scif0` 仮想 InfiniBand* アダプターを利用すると、ホストからローカル・コプロセッサーへの MPI 通信は、PCIe の帯域幅の上限に近い 6.5GB/秒に達します。HCA と同じ CPU に接続されているすべてのデバイス (CPU とコプロセッサー) 間の MPI 通信は、QDR 4X インターコネクトの帯域幅の上限である 4GB/秒に達します。

しかし、PCIe バスを介して CPU から別の CPU にメッセージを渡す処理を含む通信では、帯域幅は大幅に低下します。例えば、同じデュアルソケット・マシンの `mic0` から `mic2` への通信では、帯域幅は 1.3GB/秒に低下します。リモートマシンの `mic0` から `mic2` への通信では、帯域幅は 0.5GB/秒に過ぎません。

インテル® Xeon Phi™ コプロセッサーを搭載したヘテロジニアス・クラスターで通信の帯域幅が不均一な場合は、アプリケーションの通信パターンを考慮してチューニングを行います。次に例を示します。

a) メッセージをリングの最も近いデバイスに渡す通信パターンの場合、MPI ランクの順序を変更します。パターン `mic0` -> `mic2` -> `mic3` -> `mic1` -> リモート `mic0` は、`mic0` -> `mic1` -> `mic2` -> `mic3` -> リモート `mic0` よりも効率的です。これは、後者の (デフォルト) パターンでは `mic3` からリモート `mic0` へのリンクが最も遅い (約 0.5GB/秒) のに対して、前者の (最適化) パターンでは `mic0` -> `mic2` または `mic3` -> `mic1` (約 1.3GB/秒) より遅いリンクがないためです。この最適化を行うには、MPI マシンファイルの対応する行を変更します。

b) 帯域幅の制約を受けている MPI アプリケーションでは、(HCA が CPU1 にインストールされている場合) CPU2 にインストールされているコプロセッサー `mic2` と `mic3` のワークシェアを減らすと、クラスターのロードバランスが向上する可能性があります。

c) また、`mic0` … `mic3` からローカル CPU にデータを転送し、ローカル CPU からリモート CPU にデータを送り、リモートマシンのコプロセッサーにデータを移動する、オフロード形式の通信パターンを考慮してみてください。

d) 最後に、特定のアプリケーションについては、MPI プロセス間のより優れた帯域幅とより均一な RDMA パフォーマンスから、オフロードアプローチのほうがシンメトリック・ヘテロジニアス・クラスタリングよりも適していると判断することができるかもしれません。

一部のケースでは帯域幅が低下していますが、InfiniBand* ではインテル® Xeon Phi™ コプロセッサーの通信パスはすべて、ギガビット・イーサネットの場合よりも桁違いに高速です。これは、InfiniBand* インターコネクトを計算ノードに装備する大きな動機となります。物理的な HCA のないスタンドアロンのワークステーションでも、インテル® MPSS に OFED ソフトウェアを追加することは意味があります。この場合、インテル® MPSS は、OFED を使用して仮想アダプター `scif0` を作成し、コプロセッサー間の MPI 通信をより高速にします。

ほかにも、バスを介した不均一な通信を改良するさまざまなソフトウェア手法が調査され、解決策が提案されています [8]。Colfax Research の別のホワイトペーパーでは、2 つの HCA を CPU1 と CPU2 に装備することによりノード間通信

を改善する可能性を検討します。さらに、インテル® Xeon Phi™ コプロセッサー向けインテル® ソフトウェア (インテル® MPSS および MPI) のアップデート、ならびにコンピューティング・プラットフォームにおける PCIe バスのアップグレードについても注目します。

謝辞

問題解決と結果の解釈にご協力いただいたインテル コーポレーションの Andrey Semin 氏および Michael Hebenstreit 氏に感謝します。

参考文献 (英語)

[1] Landing page for this paper "Configuration and Bench-marks...". http://research.colfaxinternational.com/post/2014/03/11/InfiniBand-for-MIC.aspx.

[2] Intel MPI Library Reference Manual for Linux* OS. https://software.intel.com/sites/products/documentation/hpc/ics/impi/41/lin/Reference_Manual/index.htm.

[3] Hetetogeneous Clustering with Homogeneous Code. http://research.colfaxinternational.com/post/2013/10/17/Heterogeneous-Clustering.aspx.

[4] Intel Xeon Phi Coprocessor System Software Developers Guide. https://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-system-software-developers-guide.

[5] Michael Hebenstreit. Configuring Intel Xeon Phi coprocessors inside a cluster. http://software.intel.com/en-us/articles/configuring-intel-xeon-phi-coprocessors-inside-a-cluster.

[6] Intel Manycore Platform Software Stack (MPSS) User's Guide. http://registrationcenter.intel.com/irc_nas/3778/MPSS_Users_Guide.pdf.

[7] 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/nd/xeonphi/book.aspx.

[8] S. Potluri et al., in Proceedings of SC'13. MVAPICH-PRISM: a proxy-based communication framework using InfiniBand and SCIF for intel MIC clusters. http://dl.acm.org/citation.cfm?doid=2503210.2503288.