インテル® Xeon Phi™ プロセッサーへのオフロード・オーバー・ファブリック: チュートリアル

この記事は、インテル® デベロッパー・ゾーンに公開されている「Offload over Fabric to Intel® Xeon Phi™ Processor: Tutorial」の日本語参考訳です。

インテル® C++ コンパイラーでサポートされる OpenMP* 4.0 のデバイス構文により、PCI Express* (PCIe*) を介してインテル® Xeon® プロセッサー・ベースのホストマシンからインテル® Xeon Phi™ コプロセッサーへワークロードをオフロードできます。オフロード・オーバー・ファイブリック (OoF) は、第 2 世代インテル® Xeon Phi™ プロセッサーをサポートするように、このオフロード・プログラミング・モデルを拡張します。つまり、OoF を利用することで、インテル® Xeon® プロセッサー・ベースのホストマシンは、インテル® Omni-Path アーキテクチャー (インテル® OPA) や Mellanox* InfiniBand* などの高速ネットワークを介して第 2 世代インテル® Xeon Phi™ プロセッサーへワークロードをオフロードできます。

このチュートリアルでは、OoF ソフトウェアのインストール、ハードウェアの設定、基本設定のテスト、OoF の有効化について説明します。サンプル・ソースコードを使用して OoF の仕組みを解説します。

ハードウェアの準備

このチュートリアルでは、ホストマシンとしてインテル® Xeon® プロセッサー E5-2670 (2.60GHz) ベースのサーバーと、ターゲットとしてインテル® Xeon Phi™ プロセッサー・ベースのサーバーの 2 台のマシンを使用します。どちらのマシンにも Red Hat* Enterprise Linux* 7.2 とリモートログイン用にギガビット・イーサネットが搭載されています。ホストマシンとターゲットマシンのホスト名は、それぞれ *host-device* と *knl-sb2* です。

最初に、高速ネットワークのセットアップを行う必要があります。ここでは、たまたま手元にあった InfiniBand* を使用していますが、代わりにインテル® OPA を使用することもできます。

作業を開始する前に、ホストマシンとターゲットマシン間の高速ネットワークをセットアップするため、それぞれのマシンの電源をオフにします。それぞれのマシンの PCIe* スロットに Mellanox* ConnectX*-3 VPI InfiniBand* アダプターを装着し、ルーターを介さずに InfiniBand* ケーブルで直接接続します。マシンを再起動したら、まずホストに Mellanox* ネットワーク・アダプターが装着されていることを確認します。

```
[host-device]$ lspci | grep Mellanox
84:00.0 Network controller: Mellanox Technologies MT27500 Family [ConnectX-3]
```

ターゲットでも同様の確認を行います。

```
[knl-sb2 ~]$ lspci | grep Mellanox
01:00.0 Network controller: Mellanox Technologies MT27500 Family [ConnectX-3]
```

ソフトウェアのインストール

ホストマシンとターゲットマシンの両方に Red Hat* Enterprise Linux* 7.2 が搭載されています。ホストマシンで Linux* カーネルのバージョンを確認します。

```
[host-device]$ uname -a
Linux host-device 3.10.0-327.el7.x86_64 #1 SMP Thu Oct 29 17:29:29 EDT 2015
x86_64 x86_64 x86_64 GNU/Linux
```

ターゲットでも現在の OS のカーネルバージョンを確認します。

```
[knl-sb2 ~]$ uname –a
Linux knl-sb2 3.10.0-327.el7.x86_64 #1 SMP Thu Oct 29 17:29:29 EDT 2015 x86_64
x86_64 x86_64 GNU/Linux
```

ホストマシンでは、OoF を有効にするため、こちらから最新の OoF ソフトウェアを入手してインストールします。このチュートリアルでは、Red Hat* Enterprise Linux* 7.2 用の Offload Over Fabric Software 1.4.0 (`xppsl-1.4.0-offload-host-rhel7.2.tar`) をインストールします。インストールの詳細は、『インテル® Xeon Phi™ プロセッサー x200 製品ファミリー向けオフロード・オーバー・ファブリック・ユーザーズ・ガイド』(英語) を参照してください。また、OoF サポート (インテル® コンパイラーによって提供されるオフロード・プログラミング・モデルのサポート) を有効にするため、インテル® Parallel Studio XE 2017 もホストにインストールします。

ターゲットマシンでは、こちらから最新のインテル® Xeon Phi™ Processor Software を入手してインストールします。このチュートリアルでは、Red Hat* Enterprise Linux* 7.2 用のインテル® Xeon Phi™ Processor Software (`xppsl-1.4.0-rhel7.2.tar`) をインストールします。インストールの詳細は、『インテル® Xeon Phi™ Processor Software ユーザーズガイド』(英語) を参照してください。

ホストマシンとターゲットマシン間の InfiniBand* ネットワークをセットアップするため、それぞれのマシンに Red Hat* Enterprise Linux* 7.2 用の Mellanox* OpenFabrics Enterprise Distribution (OFED) for Linux* ドライバー (MLNX_OFED_LINUX 3.2-2) をインストールします。このドライバーは、www.mellanox.com (英語) から入手できます (**[Products] > [Software] > [InfiniBand/VPI Drivers]** にアクセスして、**Mellanox OFED Linux** を選択します)。

ハードウェアの基本設定のテスト

ホストマシンとターゲットマシンに Mellanox* ドライバーをインストールしたら、Mellanox* InfiniBand* Host Channel Adapters (HCA) が適切に動作していることを確認するためネットワーク・カードをテストします。InfiniBand* ネットワークを起動し、`ibping` コマンドを実行してネットワーク・リンクをテストします。

最初に、ホストで InfiniBand* とサブネット・マネージャーを起動し、リンク情報を表示します。

```
[knl-sb2 ~]$ sudo service openibd start
Loading HCA driver and Access Layer:                       [  OK  ]

[knl-sb2 ~]$ sudo service opensm start
Redirecting to /bin/systemctl start  opensm.service

[knl-sb2 ~]$ iblinkinfo
CA: host-device HCA-1:
      0x7cfe900300a13b41      1    1[  ] ==( 4X       14.0625 Gbps Active/
LinkUp)==>       2    1[  ] "knl-sb2 HCA-1" ( )
CA: knl-sb2 HCA-1:
      0xf4521403007d2b91      2    1[  ] ==( 4X       14.0625 Gbps Active/
LinkUp)==>       1    1[  ] "host-device HCA-1" ( )
```

同様に、ターゲットで InfiniBand* とサブネット・マネージャーを起動し、InfiniBand* ネットワークの各ポートのリンク情報を表示します。

```
[knl-sb2 ~]$ sudo service openibd start
Loading HCA driver and Access Layer:                       [  OK  ]

[knl-sb2 ~]$ sudo service opensm start
Redirecting to /bin/systemctl start  opensm.service
```

```
[knl-sb2 ~]$ iblinkinfo
CA: host-device HCA-1:
      0x7cfe900300a13b41      1    1[  ] ==( 4X       14.0625 Gbps Active/
LinkUp)==>       2    1[  ] "knl-sb2 HCA-1" ( )
CA: knl-sb2 HCA-1:
      0xf4521403007d2b91      2    1[  ] ==( 4X       14.0625 Gbps Active/
LinkUp)==>       1    1[  ] "host-device HCA-1" ( )
```

iblinkinfo は、ファブリックのすべてのポート (ターゲットマシンとホストマシンにそれぞれ 1 つずつ) のリンク情報を表示します。次に、ibping コマンドを実行してリンクをテストします (このコマンドは、イーサネットの ping コマンドに相当します)。次のコマンドを実行して、ホストマシンで ibping サーバーを起動します。

```
[host-device ~]$ ibping –S
```

ターゲットマシンからホストのポート ID を ping します。

```
[knl-sb2 ~]$ ibping -G 0x7cfe900300a13b41
Pong from host-device.(none) (Lid 1): time 0.259 ms
Pong from host-device.(none) (Lid 1): time 0.444 ms
Pong from host-device.(none) (Lid 1): time 0.494 ms
```

同様に、ターゲットマシンで ibping サーバーを起動します。

```
[knl-sb2 ~]$ ibping -S
```

そして、ホストマシンからターゲットのポート ID を ping します。

```
[host-device ~]$ ibping -G 0xf4521403007d2b91
Pong from knl-sb2.jf.intel.com.(none) (Lid 2): time 0.469 ms
Pong from knl-sb2.jf.intel.com.(none) (Lid 2): time 0.585 ms
Pong from knl-sb2.jf.intel.com.(none) (Lid 2): time 0.572 ms
```

IPoIB (IP over InfiniBand*) 設定

ここまでで、InfiniBand* ネットワークが動作していることを確認しました。次に、OoF を使用するため、IPoIB (IP over InfiniBand*) を設定する必要があります。IPoIB は、ファブリックを介して計算をオフロードする際に使用されるターゲット IP アドレスを提供します。

最初に、ib_ipoib ドライバーがインストールされていることを確認します。

```
[host-device ~]$ lsmod | grep ib_ipoib
ib_ipoib               136906  0
ib_cm                   47035  3 rdma_cm,ib_ucm,ib_ipoib
ib_sa                   33950  5 rdma_cm,ib_cm,mlx4_ib,rdma_ucm,ib_ipoib
ib_core                141088  12
rdma_cm,ib_cm,ib_sa,iw_cm,mlx4_ib,mlx5_ib,ib_mad,ib_ucm,ib_umad,ib_uverbs,rdma_uc
m,ib_ipoib
mlx_compat              16639  17
rdma_cm,ib_cm,ib_sa,iw_cm,mlx4_en,mlx4_ib,mlx5_ib,ib_mad,ib_ucm,ib_addr,ib_core,i
b_umad,ib_uverbs,mlx4_core,mlx5_core,rdma_ucm ib_ipoib
```

ib_ipoib ドライバーがリストされない場合は、次のコマンドを実行して Linux* カーネルにこのモジュールを追加する必要があります。

```
[host-device ~]$ modprobe ib_ipoib
```

次に、ホストで ifconfig コマンドを実行して InfiniBand* インターフェイス ib0 をリストします。

```
[host-device ~]$ ifconfig ib0
ib0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 2044
Infiniband hardware address can be incorrect! Please read BUGS section in
ifconfig(8).
        infiniband A0:00:02:20:FE:80:00:00:00:00:00:00:00:00:00:00:00:00:00:00
txqueuelen 1024  (InfiniBand)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

このインターフェイスの IP アドレスを 10.0.0.1 に設定します。

```
[host-device ~]$ sudo ifconfig ib0 10.0.0.1/24
[host-device ~]$ ifconfig ib0
ib0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 2044
        inet 10.0.0.1  netmask 255.255.255.0  broadcast 10.0.0.255
Infiniband hardware address can be incorrect! Please read BUGS section in
ifconfig(8).
        infiniband A0:00:02:20:FE:80:00:00:00:00:00:00:00:00:00:00:00:00:00:00
txqueuelen 1024  (InfiniBand)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10  bytes 2238 (2.1 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

同様に、ターゲットで InfiniBand* インターフェイスの IP アドレスを 10.0.0.2 に設定します。

```
[knl-sb2 ~]$ ifconfig ib0
ib0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 2044
Infiniband hardware address can be incorrect! Please read BUGS section in
ifconfig(8).
        infiniband A0:00:02:20:FE:80:00:00:00:00:00:00:00:00:00:00:00:00:00:00
txqueuelen 1024  (InfiniBand)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[knl-sb2 ~]$ sudo ifconfig ib0 10.0.0.2/24
[knl-sb2 ~]$ ifconfig ib0
ib0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 2044
        inet 10.0.0.2  netmask 255.255.255.0  broadcast 10.0.0.255
Infiniband hardware address can be incorrect! Please read BUGS section in
ifconfig(8).
        infiniband A0:00:02:20:FE:80:00:00:00:00:00:00:00:00:00:00:00:00:00:00
txqueuelen 1024  (InfiniBand)
        RX packets 3  bytes 168 (168.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10  bytes 1985 (1.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

最後に、ホストからターゲットの新しい IP アドレス 10.0.0.2 を ping して、接続できることを確認します。

```
[host-device ~]$ ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.443 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.410 ms
<CTRL-C>
```

同様に、ターゲットからホストの新しい IP アドレス `10.0.0.1` を ping して、接続できることを確認します。

```
[knl-sb2 ~]$ ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.313 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.359 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.375 ms
<CTRL-C>
```

パスワードなしの SSH 設定 (オプション)

ターゲットマシンにワークロードをオフロードする場合、セキュアシェル (SSH) ではターゲットにログオンしてワークロードを実行するのにターゲットのパスワードが必要です。この処理を自動化するには、パスワードなしの `ssh` ログインを有効にする必要があります。最初に、ホストでパスワードを入力せずに認証キーのペアを生成します。

```
[host-device ~]$ ssh-keygen -t rsa
```

次に、`ssh-copy-id` コマンドでホストの新しい公開キーをターゲットの公開キーに追加します。

```
[host-device ~]$ ssh-copy-id @10.0.0.2
```

オフロード・オーバー・ファブリック (OoF)

ここまでで、高速ネットワークのセットアップが完了し、動作していることを確認しました。OoF 機能を有効にするには、ホストにインテル® Parallel Studio XE 2017 for Linux* をインストールする必要があります。そして、次のコマンドを実行してシェル環境を設定します。

```
[host-device]$ source /opt/intel/parallel_studio_xe_2017.0.035/psxevars.sh
intel64
Intel(R) Parallel Studio XE 2017 for Linux*
Copyright (C) 2009-2016 Intel Corporation. All rights reserved.
```

以下のサンプルプログラムを使用して OoF 機能をテストします。このサンプルプログラムは、ホストで定数 `A` とバッファー `x`、`y`、`z` の割り当てと初期化を行い、OpenMP* デバイス構文ディレクティブ (`#pragma omp target map…`) を使用して計算をターゲットにオフロードします。target ディレクティブは、ターゲット上にデバイスデータ環境を作成します。実行時に計算を開始する前に変数 `x`、`y` と `A` の値がターゲットにコピーされ、ターゲットが計算を完了すると変数 `y` の値がホストにコピーバックされます。この例では、ターゲットは `lscpu` コマンドで取得した CPU 情報を解析し、OpenMP* スレッドのチームをスポーンして、ベクトルスカラー積を計算し結果をベクトルに加算しています。

```
#include <stdio.h>

int main(int argc, char* argv[])
{
    int i, num = 1024;;
    float A = 2.0f;

    float *x = (float*) malloc(num*sizeof(float));
    float *y = (float*) malloc(num*sizeof(float));
    float *z = (float*) malloc(num*sizeof(float));

    for (i=0; i<num; i++)
    {
       x[i] = i;
       y[i] = 1.5f;
       z[i] = A*x[i] + y[i];
    }

    printf("Workload is executed in a system with CPU information:\n");
    #pragma omp target map(to: x[0:num], A) \
                       map(tofrom: y[0:num])
    {
        char command[64];
        strcpy(command, "lscpu | grep Model");
        system(command);
        int done = 0;

        #pragma omp parallel for
        for (i=0; i<num; i++)
        {
            y[i] = A*x[i] + y[i];

            if ((omp_get_thread_num() == 0) && (done == 0))
            {
               int numthread = omp_get_num_threads();
               printf("Total number of threads: %d\n", numthread);
               done = 1;
            }
        }
    }

    int passed = 0;

    for (i=0; i<num; i++)
        if (z[i] == y[i]) passed = 1;

    if (passed == 1)
        printf("PASSED!\n");
    else
        printf("FAILED!\n");

    free(x);
    free(y);
    free(z);

    return 0;
}
```

この OpenMP* プログラムを、オフロード領域を第 2 世代インテル® Xeon Phi™ プロセッサー向けにビルドするように指示する -qoffload-arch=mic-avx512 オプションを指定してインテル® コンパイラーでコンパイルします。プログラムを実行する前に、高速ネットワークを使用するように、OFFLOAD_NODES 環境変数にターゲットマシンの IP アドレス (この例では 10.0.0.2) を設定します。

```
[host-device]$ icc -qopenmp -qoffload-arch=mic-avx512 -o OoF-OpenMP-Affinity OoF-
OpenMP-Affinity.c

[host-device]$ export OFFLOAD_NODES=10.0.0.2

[host-device]$ ./OoF-OpenMP-Affinity
Workload is executed in a system with CPU information:
Model:                  87
Model name:             Intel(R) Xeon Phi(TM) CPU 7250 @000000 1.40GHz
PASSED!
Total number of threads: 268
```

オフロード処理は、インテル® Coprocessor Offload Infrastructure (インテル® COI) によって内部処理されます。デフォルトでは、オフロードコードはターゲットで利用可能なすべての OpenMP* スレッドを使用して実行されます。ターゲットには 68 のコアがあり、そのうち 1 つのコアでインテル® COI デーモンが実行しているため、67 のコアが利用可能で、合計スレッド数は 268 (コアあたり 4 スレッド) になります。coitrace コマンドを使用して、インテル® COI のすべての API 呼び出しを追跡できます。

```
[host-device]$ coitrace ./OoF-OpenMP-Affinity
COIEngineGetCount [ThID:0x7f02fdd04780]
        in_DeviceType = COI_DEVICE_MIC
        out_pNumEngines = 0x7fffc8833e00 0x00000001 (hex) : 1 (dec)

COIEngineGetHandle [ThID:0x7f02fdd04780]
        in_DeviceType = COI_DEVICE_MIC
        in_EngineIndex = 0x00000000 (hex) : 0 (dec)
        out_pEngineHandle = 0x7fffc8833de8 0x7f02f9bc4320

Workload is executed in a system with CPU information:
COIEngineGetHandle [ThID:0x7f02fdd04780]
        in_DeviceType = COI_DEVICE_MIC
        in_EngineIndex = 0x00000000 (hex) : 0 (dec)
        out_pEngineHandle = 0x7fffc88328e8 0x7f02f9bc4320

COIEngineGetInfo [ThID:0x7f02fdd04780]
        in_EngineHandle = 0x7f02f9bc4320
        in_EngineInfoSize = 0x00001440 (hex) : 5184 (dec)
        out_pEngineInfo = 0x7fffc8831410
                DriverVersion:
                DeviceType: COI_DEVICE_KNL
                NumCores: 68
                NumThreads: 272
```

<以下略>

OpenMP* スレッド・アフィニティー

上記のサンプルプログラムの結果から、ターゲットで実行されるデフォルトのスレッド数 (272) が分かります。ターゲットで実行するスレッド数は、明示的に指定することもできます。1 つの方法は、ホストで環境変数を使用してターゲットの実行環境を変更します。最初に、ターゲット固有の環境変数のプリフィクスを定義し、このプリフィクスを OpenMP* スレッド・アフィニティー環境変数に追加します。例えば、次の環境変数の設定は、オフロードランタイムがターゲット上で 8 つのスレッドを使用するように指定します。

```
[host-device]$ $ export MIC_ENV_PREFIX=PHI
[host-device]$ $ export PHI_OMP_NUM_THREADS=8
```

インテルの OpenMP* ランタイム拡張の KMP_PLACE_THREAD 環境変数と KMP_AFFINITY 環境変数を使用すると、スレッドを物理プロセシング・ユニット (コア) にバインドすることができます (詳細は、『インテル® C++ コンパイラー・デベロッパー・ガイドおよびリファレンス』の「スレッド・アフィニティー・インターフェイス」を参照してください)。例えば、次の環境変数の設定は、オフロードランタイムが隣接する 8 つのスレッドを使用するように指定します。

```
[host-device]$ $ export PHI_KMP_AFFINITY=verbose,granularity=thread,compact
[host-device]$ $ ./OoF-OpenMP-Affinity
```

OMP_PROC_BIND 環境変数を使用して OpenMP* アフィニティーを利用することもできます。例えば、OMP_PROC_BIND で上記と同じ隣接する 8 つのスレッドを使用するように指定するには、次のように設定します。

```
[host-device]$ $ export MIC_ENV_PREFIX=PHI
[host-device]$ $ export PHI_KMP_AFFINITY=verbose
[host-device]$ $ export PHI_OMP_PROC_BIND=close
[host-device]$ $ export PHI_OMP_NUM_THREADS=8
[host-device]$ $ ./OoF-OpenMP-Affinity
```

または、分散する 8 つのスレッドで実行する場合は、次のように設定します。

```
[host-device]$ $ export PHI_OMP_PROC_BIND=spread
[host-device]$ $ ./OoF-OpenMP-Affinity
```

結果を次の表に示します。

OpenMP* スレッド数	コア数
0	0
1	10
2	19
3	31
4	39
5	48
6	56
7	65

8 つのスレッドをコアあたり 2 スレッド (合計 4 コア) で実行するには、次の設定を使用します。

```
[host-device]$ export PHI_OMP_PROC_BIND=close;
[host-device]$ export PHI_OMP_PLACES="cores(4)"
[host-device]$ export PHI_OMP_NUM_THREADS=8
[host-device]$ $ ./OoF-OpenMP-Affinity
```

結果を次の表に示します。

OpenMP* スレッド数	コア数
0	0
1	0
2	1
3	1
4	2
5	2
6	3
7	3

まとめ

このチュートリアルでは、OoF アプリケーションのセットアップと実行について、ハードウェアの取り付けとソフトウェアのインストールも含め詳しく説明しました。ここで紹介した例では、Mellanox* InfiniBand* HCA を使用しましたが、代わりにインテル® OPA を使用することもできます。オフロード・プログラミング・モデルを使用するサンプル OpenMP* アプリケーションを使って、インテル® Xeon® プロセッサー・ベースのホストで実行し、高速ネットワークを介してインテル® Xeon Phi™ プロセッサー・ベースのターゲットに計算をオフロードする方法を示しました。また、インテル® Xeon Phi™ プロセッサー向けにコンパイルおよび実行する方法と、インテル® Xeon Phi™ プロセッサー上で OpenMP* スレッド・アフィニティーを制御する方法も紹介しました。

参考資料

- 開発コード名 Knights Landing でオフロード・オーバー・ファブリックを使用する (英語)
- インテル® Xeon Phi™ コプロセッサー向けオフロードランタイム (英語)
- インテル® Xeon Phi™ コプロセッサー向けインテル® COI アプリケーションのデバッグ (英語)
- OpenMP* スレッド・アフィニティーの制御
- インテル® C++ コンパイラー・ユーザー・リファレンス・ガイド (英語)

著者紹介

Loc Q Nguyen。ダラス大学で MBA を、マギル大学で電気工学の修士号を、モントリオール理工科大学で電気工学の学士号を取得しています。現在は、インテル コーポレーションのソフトウェア & サービスグループのソフトウェア・エンジニアで、コンピューター・ネットワーク、並列コンピューティング、コンピューター・グラフィックスを研究しています。