OpenMP* を使用した効率良いヘテロジニアス並列プログラミング

CPU と GPU を同時に動作させるベスト・プラクティス

Elmira Volkova 学部生インターン
Alexander Bobyr インテル コーポレーション ソフトウェア・イネーブリング & 最適化エンジニア
Igor Ermolaev インテル コーポレーション 主席エンジニア

GPU などのアクセラレーターに計算をオフロードすると、オフロードした計算が終了するまでホスト CPU がアイドルになるケースがあります。しかし、CPU と GPU のリソースを同時に使用することで、アプリケーションのパフォーマンスを向上させることができます。ヘテロジニアス並列処理を利用する OpenMP* プログラムでは、`master` 節 (注:OpenMP* 5.1 仕様以降では `masked` に変更されています) を使用して CPU と GPU を同時に実行できます。この記事では、OpenMP* を使用して CPU と GPU で非同期計算を行う方法を紹介します。

SPEC ACCEL* 514.pomriq (英語) MRI 再構成ベンチマークは、C 言語で記述され、OpenMP* で並列化されています。このベンチマークは、一部の計算をアクセラレーターにオフロードしてヘテロジニアス並列処理を実行できます。この記事では、ホスト CPU とインテルのディスクリート GPU に計算を分割し、両方のプロセッサーがビジー状態になるようにします。また、インテル® VTune™ プロファイラーを使用して、CPU と GPU の利用率を測定し、パフォーマンスを解析します。

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

ここでは、ヘテロジニアス並列処理の開発とパフォーマンス・チューニングの 5 つのステージを紹介します。

1. 並列化に適したコード領域を探します。
2. CPU と GPU の両方がビジーになるように、これらの領域を並列化します。
3. 最適な作業配分係数を見つけます。
4. この配分係数でヘテロジニアス並列アプリケーションを起動します。
5. パフォーマンスの向上を測定します。

最初は、並列領域は GPU でのみ実行され、CPU はアイドルになります（**図 1**）。ご覧の通り、CPU では「OMP Primary Thread」のみが実行され、GPU は ComputeQ オフロードカーネルでフル稼働（**[GPU Execution Units] > [EU Array] > [Active]**）している状態です。

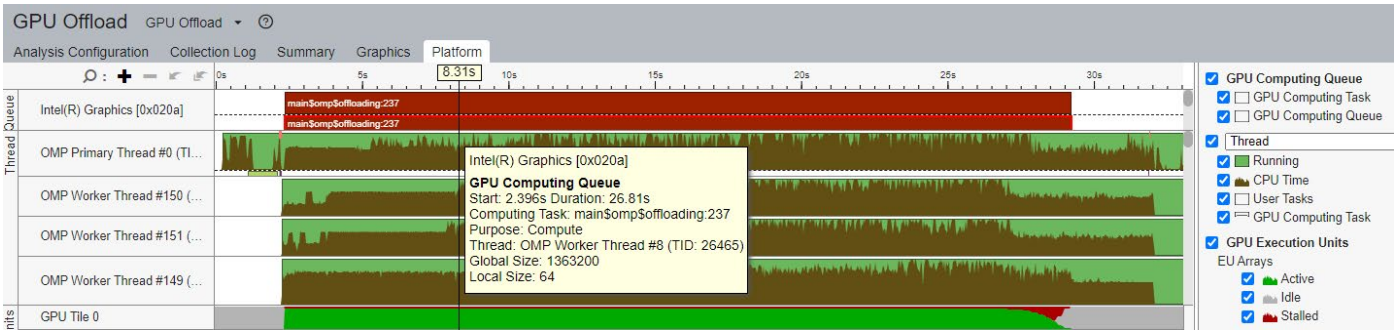

図 1. インテル® VTune™ プロファイラーで初期コードをプロファイルした結果

コードを調べた結果、各配列と各実行領域を複製し、1 つ目のコピーが GPU で実行され、2 つ目が CPU で実行されるようにしました。プライマリー・スレッドは OpenMP* `target` ディレクティブを使用して、GPU に作業をオフロードします。これを図式化したのが**図 2** です。`nowait` ディレクティブは、CPU と GPU で実行されるスレッド間の不要な同期を回避し、スレッド間のロードバランスを向上します。

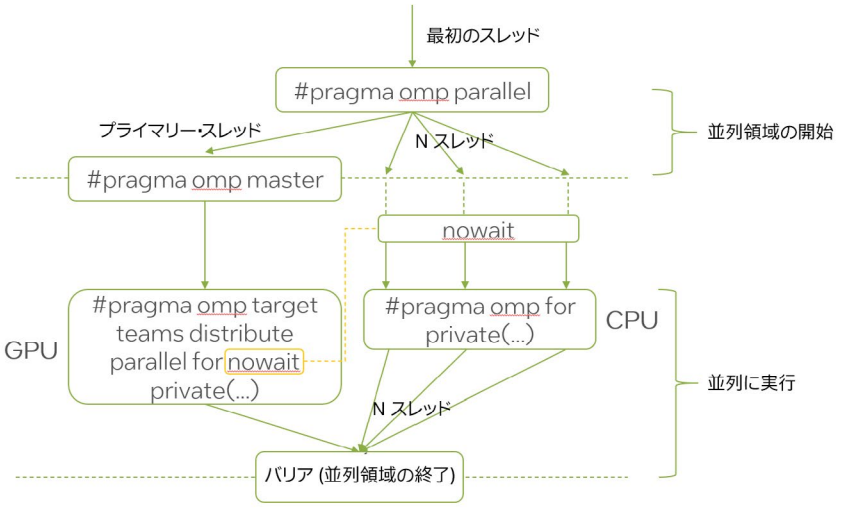

図 2. CPU と GPU をビジーに保つ OpenMP* 並列処理スキーム

CPU と GPU の作業配分のバランスは、stdin から読み込まれる part 変数で調節されます(**図 3**)。この変数は、GPU にオフロードされるワークロードの割合に numX を掛けた値になります。残りの作業は CPU で行われます。OpenMP* ヘテロジニアス並列処理の実装例を**図 4** に示します。

```
    float part;
    char *end;
    part   = strtof(argv[2], &end);

int TILE_SIZE = numX*part;

Qrc = (float*) memalign(alignment, numX * sizeof (float));
Qrg = (float*) memalign(alignment, TILE_SIZE * sizeof (float));
```

図 3. CPU と GPU 間の作業配分係数

```
{
    #pragma omp master
    #pragma omp target teams distribute parallel for nowait private(expArg, cosArg, sinArg)
    for (indexX = 0; indexX < TILE_SIZE; indexX++) {
        float QrSum = 0.0;
        float QiSum = 0.0;
        #pragma omp simd private(expArg, cosArg, sinArg) reduction(+:QrSum, QiSum)
        for (indexK = 0; indexK < numK; indexK++) {
            expArg = PIx2 * (GkVals[indexK].Kx * xg[indexX] + GkVals[indexK].Ky * yg[indexX] + GkVals[indexK].Kz * zg[indexX]);
            cosArg = cosf(expArg);
            sinArg = sinf(expArg);
            float phi = GkVals[indexK].PhiMag;
            QrSum += phi * cosArg;
            QiSum += phi * sinArg;
        }
        Qrg[indexX] += QrSum;
        Qig[indexX] += QiSum;
    }
    #pragma omp for private(expArg, cosArg, sinArg)
    for (indexX = TILE_SIZE; indexX < numK; indexX++) {
        float QrSum = 0.0;
        float QiSum = 0.0;
        #pragma omp simd private(expArg, cosArg, sinArg) reduction(+:QrSum, QiSum)
        for (indexK = 0; indexK < numK; indexK++) {
            expArg = PIx2 * (CkVals[indexK].Kx * xc[indexX] + CkVals[indexK].Ky * yc[indexX] + CkVals[indexK].Kz * zc[indexX]);
            cosArg = cosf(expArg);
            sinArg = sinf(expArg);
            float phi = CkVals[indexK].PhiMag;
            QrSum += phi * cosArg;
            QiSum += phi * sinArg;
        }
        Qrc[indexX] += QrSum;
        Qic[indexX] += QiSum;
    }
}
```

図 4. CPU と GPU を同時に利用する OpenMP* 実装のコード例

インテル® oneAPI DPC++/C++ コンパイラーで次のコマンドライン・オプションを指定してビルドします。

```
-O3 -Ofast -xCORE-AVX512 -mprefer-vector-width=512 -ffast-math
-qopt-multiple-gather-scatter-by-shuffles -fimf-precision=low
-fiopenmp -fopenmp-targets=spir64="-fp-model=precise"
```

表 1 は、CPU と GPU の作業比率（つまり、前述の `part` 変数）を変えた場合のパフォーマンスです。このシステムとワークロードでは、オフロード比率 0.65 が、CPU と GPU の間の最適なロードバランスとなり、プロセッサー・リソースの最適な利用率となります。インテル® VTune™ プロファイラーのプロファイルでは、CPU と GPU の間で作業がより均等に分散され、両方のプロセッサーが有効に活用されていることが分かります(**図 5**)。「OMP Primary Thread」がオフロードされたカーネル（`main:237`）を GPU 上で実行する間、ほかの「OMP Worker Threads」が CPU 上でアクティブになります。

オフロード比率	合計時間（秒）	GPU 時間（秒）
0.00	61.2	0.0
0.20	51.6	8.6
0.40	41.0	16.8
0.60	31.5	24.7
0.65	28.9	26.7
0.80	34.8	32.6
1.00	43.4	40.7

表 1. オフロードされる作業量（part 変数の値）別の hotspot 時間

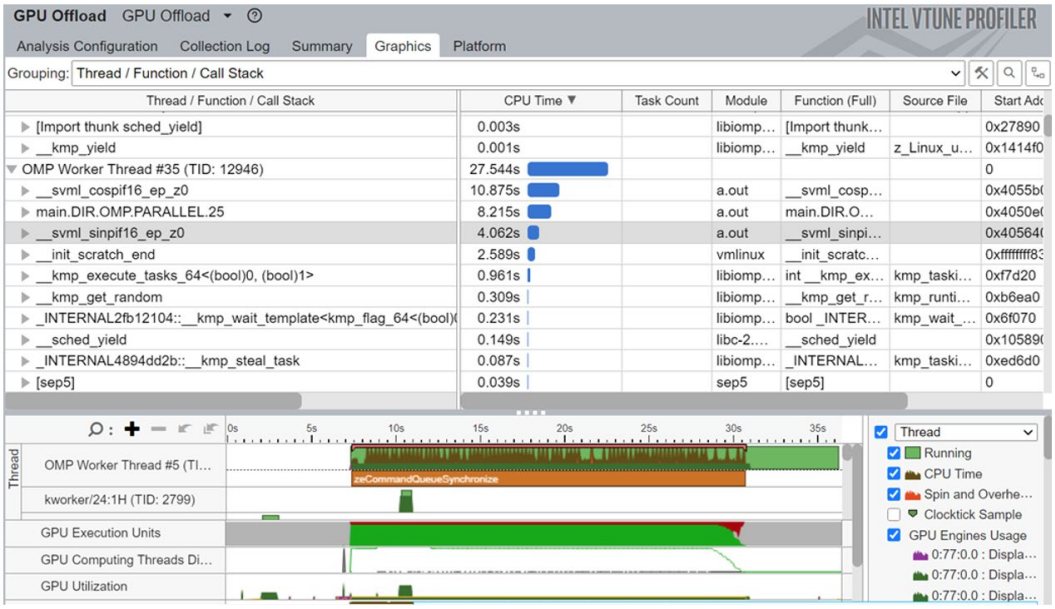

図 5. コードの 65% を GPU にオフロードした場合のプロファイル結果

図 6 は、`part` の値を変えた場合の実行時間です。`part` が `0` の場合は GPU に何もオフロードされず、`part` が `1` の場合はすべての作業がオフロードされます。CPU と GPU にバランス良く作業を振り分ければ、どちらか一方で作業するよりも優れたパフォーマンスが得られることは明らかです。

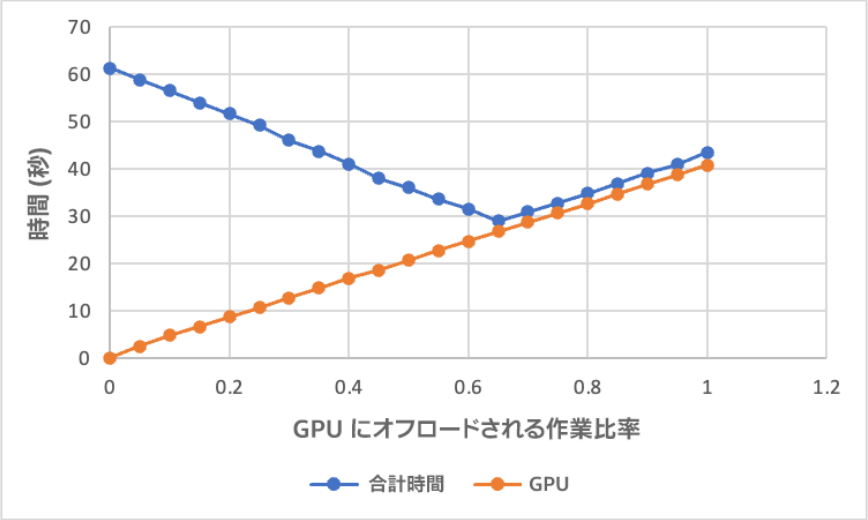

図 6. GPU にオフロードした場合の合計時間と GPU 時間（時間はすべて秒単位)

OpenMP* は、CPU + GPU システムで真の非同期ヘテロジニアス実行をサポートします。パフォーマンスの比較結果やインテル® VTune™ プロファイラーのプロファイル結果から、OpenMP* 並列領域で CPU と GPU をビジー状態に保つことで、最高のパフォーマンスが得られることは明らかです。ぜひこのアプローチを試してみてください。