SYCL* パフォーマンス: SYCL* に最適なワークグループ・サイズの選択

この記事は 2020 年 1 月 9 日に Codeplay のウェブサイトで公開された「SYCL Performance Post: Choosing a Good Work Group Size for SYCL」の日本語参考訳です。原文は更新される可能性があります。原文と翻訳文の内容が異なる場合は原文を優先してください。

SYCL* の「ワークグループ」サイズとは何か? パフォーマンスにどのような影響を与えるのか?

「ワークグループ」は、スレッド階層内の 1、2、または 3 次元のスレッドのセットです。「ワークアイテム」のセットが含まれており、これらのワークアイテムはそれぞれ GPU の「コア」にマップされます。

OpenCL* デバイスで SYCL* を使用する場合、通常、「ワークグループ」のサイズが計算ユニットの利用率を決定します。最高のパフォーマンスを実現するには、ワークグループ・サイズをハードウェアの計算ユニットのサイズに一致させる必要があります。大きすぎると計算がコアに収まらず、小さすぎるとコアが最大限に使用されず、どちらの場合も、コードのパフォーマンスを低下させる可能性があります。ソフトウェアは、ハードウェア・リソースをフル活用できる場合にのみ、OpenCL* デバイス (**GPU** など) で 100% の利用率を達成できます。

ワークグループ・サイズの違いが SYCL* コードのパフォーマンスに与える影響を示す例については、アルゴンヌ国立研究所の論文「SYCL* を使用した医用画像アプリケーションのパフォーマンス向上」(英語) の 8 ページと 9 ページを参照してください。

はじめに

この記事の目的は、最適なパフォーマンスを得るために使用するワークグループ・サイズを決定するさまざまな手法を要約および分析し、理想的なワークグループ・サイズの値を選択する際に考慮すべき要因を説明することです。

この記事で紹介する手法は、SYCL* だけでなく、**GPGPU** で使用されている最も一般的なものです。

それでは、カーネルが最高のパフォーマンスを発揮できるように、最適なワークグループ・サイズを計算する方法を検討してみましょう。

カーネル実行範囲の適切な値を見つけるには、調査が必要になります。これは、カーネルを起動するワークグループの最適なサイズを決定する**万能の**公式が存在しないためです。

この値は、アルゴリズムをプロファイルし、ベンチマークして、いくつかの実行数を比較する、ちょっとした検証によって明らかになります。

幸いなことに、検証を正しい方向に導くのに役立ついくつかのガイドラインがあります。この記事では、SYCL* と OpenCL* に関連する最も重要なガイドラインを取り上げます。これらのガイドラインは、SYCL* だけでなく **GPGPU** プログラミング全般にも関連します。

最初は、カーネルに最適なワークグループ・サイズの選択を OpenCL* ランタイム実装に任せるのが良い出発点です。通常、このデフォルトサイズは許容可能なパフォーマンスをもたらしますが、改善できる可能性があります。

前述のように、理想的な解決策は、異なるサイズのワークグループを使用して一連の検証を実施することです。これにより、実行しているデバイス上のカーネルに最適なパフォーマンスをもたらすサイズが分かります。ただし、ここでの本当の問題は、ワークグループの理想的なサイズがハードウェアに依存することです。したがって、複数のデバイスをサポートする場合、これに対応する関数、つまり使用されているハードウェアに応じて適切なワークグループ・サイズを選択する関数を作成する必要があるかもしれません。

ランタイムにワークグループ・サイズを決定させる

一部のハードウェアでは、ワークグループ・サイズを指定せず、ランタイムに最適な値を選択させることで、妥当な結果が得られ、一連の検証を実行する必要がありません。

デフォルトサイズのメリット

- プライマリー・メモリー (OpenCL* ではグローバルメモリー) のみを使用するカーネルの最適なワークグループ・サイズを OpenCL* 実装に自動的に決定させることは珍しいことではなく、多くの場合はこれにより十分な結果が得られます。

デフォルトサイズのデメリット

- これは、キャッシュメモリー (ローカルメモリー) を通じてメモリーの局所性を利用するカーネルには理想的ではない可能性があります。キャッシュメモリーは最速のメモリーであり、高速にアクセスするためデータの一時的な保存にも使用されます。

手動でワークグループ・サイズを検証する

推奨事項

ワークグループのオーバーヘッドを軽減する

ワークグループの切り替えオーバーヘッドを軽減するには、できるだけ大きなワークグループを作成する必要があります。つまり、ワークグループごとに少なくとも **32** 個、できればそれ以上のワークアイテムを持つようにします。**32** 個は、単純な開始点としては適していますが、最適なサイズである可能性は低いかもしれません。**32** と `device_max_work_group_size` の間の 32 の整数倍をワークグループ・サイズとして試し、ハードウェア・ベンダー (**AMD、Intel、ARM など**) とデバイスタイプ (**CPU、GPU、FPGA など**) に最適なものを確認します。

倍数の選択に使用する正確な開始数を知りたい場合は、`kernel_preferred_work_group_size_multiple` を照会します。次のコードを使用します。

```
const size_t max_device_work_group_size =
kernel.get_work_group_info<sycl::info::kernel_work_group::preferred_work_group_si
ze_multiple>(device);
```

通常、この値の倍数がデバイスにとって最適です。

倍数は、使用するメモリー・アクセス・パターンと各ワークアイテムのワークロードによって異なります。

ガイドライン:

- 負荷の高い、計算依存のカーネルを実行する場合、小さい倍数を試してください。
- メモリーアクセスがボトルネックになる場合、メモリー・レイテンシーを隠匿するため倍数を大きくしてみてください。
- レジスターを使用する場合、レジスターの利用率が高くなるほど、実行できるスレッドが少なくなります。

プロファイルとベンチマーク

コードのパフォーマンスを向上する最初のルールは、何に時間が費やされているかを測定することです。OpenCL* ベンダーは通常、コードの各部分にかかる時間を測定するツールを提供しています。Codeplay は、SYCL* 実装である ComputeCpp 用の組み込みプロファイラーを実装しています。

ComputeCpp を使用した SYCL* プロファイルの詳細については、ブログ記事「プロファイルを使用した SYCL* コードの最適化」(英語) を参照してください。実行時間をプロファイルするカスタムタイマーを作成する方法と、ComputeCpp Professional Edition の自動プロファイル機能を使用する方法について説明しています。これは、実行時間とメモリー帯域幅の記録に加えて、ボトルネックの発見に役立つその他の多くの機能を提供する統合 JSON プロファイラーです。

考慮事項

さらに、ワークグループの同時実行能力は、以下の要因によっても影響を受ける可能性があります。

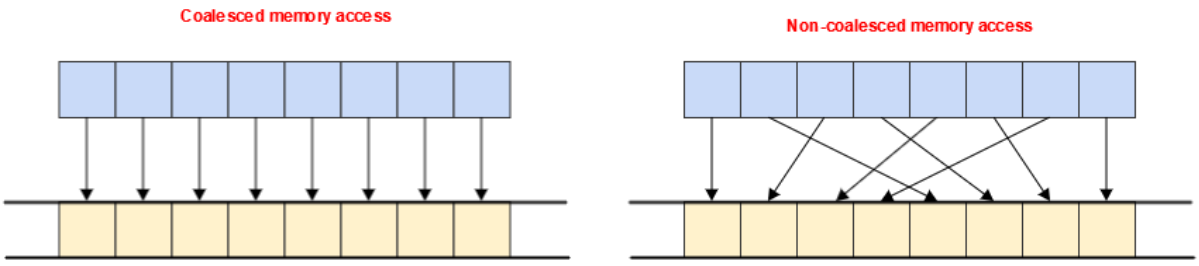
- バリアによるワークグループ同期が必要になるカーネル分岐。
  - バリアは文字通り実行をブロックし、すべてのスレッドがバリアに到達しないと続行できません。
- 計算ユニットごとに利用可能な共有ローカルメモリーの量。
  - 各ワークグループは、一定量のメモリーを自分専用に割り当てることができます。ただし、最大量を割り当てた場合、同じ計算ユニット上で他のワークグループを同時にスケジューリングできなくなる可能性があることに注意してください。

デバイスキャッシュ

限られた数のワークグループでデバイスの利用率を高く保つには、より大きな (2 の累乗の) ワークグループ・サイズが必要です。ローカル・ワーク・サイズはグローバル・ワーク・サイズの除算でなければならないため、2 の累乗である必要があります。ただし、アクセスされるデータセットのサイズには上限があります。スレッド間のデータアクセスを最適化するには、単一のワークグループ内の L1 キャッシュのサイズを超えないことが最善です。

ワークグループ内の一連のワークアイテムがメモリーにアクセスする命令を同時に実行する場合、そのグループ内のスレッドによって作成されるアクセスパターンを考慮することが重要です。

例えば、L1 キャッシュからデータをロードする場合、最も効率の悪いパターン (1 つのスレッドが 1 つの値を読み取る) を使用しているか、最も効率良いパターン (16 個のスレッドすべてが連続した 4 バイト値を読み取る) を使用しているかにかかわりなく、64 バイトのキャッシュライン全体がフェッチされる可能性があります。以下の図は、これらのパターンの例を示したものです。



`device_global_mem_cacheline_size` を介してデバイスのキャッシュライン・サイズの値を照会し、この値を使用して特定のハードウェアに最適なワークグループ・サイズを指定できます。

SIMD アプローチにより、特定のワークアイテムのグループ (ワークグループ全体である「スライス」の「サブスライス」) を同時に実行する機能は仕様で保証されていないため、これは各デバイスの OpenCL* 実装方法に基づいて考慮されます。**インテルの SoC (システムオンチップ)** ハードウェアはこの種の最適化を目的に設計されているため、GPU で実行する場合に役立ちます。

同期

もう 1 つの考慮事項は、ワークグループ内のワークアイテム間の同期が必要な場合です。

カーネルコード内の論理分岐を最小限 (理想的にはゼロ) に抑えている場合、カーネルは各計算ユニット (1 つの計算ユニットが 1 つのワークグループを実行する) の実行に同じ時間を要します。

ワークグループ・サイズは、OpenCL* デバイスの計算ユニット数に基づいて計算できます。

`device_max_compute_units` の値を照会し、それを使用してワークグループ・サイズを計算します。実行している OpenCL* 実装が仕様に準拠していると仮定すると、この照会で得られる値は、デバイスを最大限に活用するワークグループの最小数を示します。したがって、実行するワークアイテムの総数を、照会した OpenCL* デバイスの計算ユニットの数で割れば、最適なワークグループ・サイズの近似値が得られます。

実際には、特に GPU では、通常、計算ユニットごとのワークグループ数をより多くしたいと思うでしょうが、GPU デバイス向けにワークグループ・サイズを選択する方法としては有効です。これは、カーネルが (理想的には) 分岐のないロジックを持ち、同期を伴うことを前提としています。

まとめ

全体として、ハードウェアに依存しない方法でワークアイテムを割り当てることが目的の場合、ユーザー側でワークグループ・サイズを指定しないことが、良い結果をもたらす可能性があります。これが最適であるとは保証されませんが、`kernel_preferred_work_group_size_multiple` の倍数を試したことがない場合、またはカーネルが必ずしも特定のグループサイズ向けに設計されていない場合、これは信頼できるデフォルトオプションです。

そうでない場合は、ハードウェア・アーキテクチャーだけでなく、ワークアイテムのメモリー・アクセス・パターンやワークロードを含むカーネルの設計も考慮しなければなりません。

また、プロファイルを使用して、ワークロードが計算依存かメモリー依存かを判断する必要があります。これにより、コードのパフォーマンスを向上するため何を変更すべきかが分かります。

最終的には、アクセス時間と計算時間がいつ最適であるかを判断することが重要であり、実行時間とメモリー帯域幅のベンチマークを忘れずに実行し、比較と分析のためにデータを保存します。

用語集

- **計算ユニット:** プロセッサー・ユニット
- **グローバルメモリー:** デバイスのすべてのスレッドから見えるメモリープールからの割り当てを表します。
- **ローカルメモリー:** ワークグループごとに割り当てられ、そのワークグループ内のすべてのワークアイテムから見える連続したメモリー領域です。
- **ワークグループ:** スレッド階層内の 1、2、または 3 次元のスレッドのセットです。SYCL* では、バリアを使用して、ワークグループ内のすべてのワークアイテム間でのみ同期が可能です。
- **ワークアイテム:** スレッド階層内の 1 つのスレッドです。

参考資料:「プロファイルを使用した SYCL* コードの最適化」(英語)
