DPEcho: 2020 年代以降の SYCL* による一般相対論

科学計算における SYCL* の優位性

Salvatore Cielo ライプニッツ・スーパーコンピューティング・センター 宇宙プラズマ物理学アプリケーション・サポート担当 HPC エキスパート

Alexander Pöppl 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

Luca Del Zanna フィレンツェ大学 物理および天文学部准教授

Matteo Bugli トリノ大学 天体物理学研究者兼 MSCA フェロー

コードは複雑であるものの、定量的かつ定性的に大きな問題クラスのシミュレーションを可能にするヘテロジニアス・コンピューティングのおかげで、数値科学が見直されています。SYCL*（英語）プログラミング言語は、拡張性、移植性、オープン性を備えたヘテロジニアスへの標準アプローチを提供し、ワークシェアリングやスケジューリング、シミュレーション領域を実行空間に直接マッピングする直感的な C++ API により、アクセラレーターの能力を引き出します。後者は、空間と時間の性質が運動方程式と緊密に結合し、計算量とメモリー消費量が多い数値相対論において、特に便利です。

一般相対論的磁気流体力学(GR-MHD)の ECHO コード[1] には、最近の HPC 向けに最適化されたバージョン (ECHO-3DHPC[2]) がすでにあります。これは Fortran で記述されており、MPI + OpenMP* でハイブリッド並列処理を実装しています。この記事では、このコードに手を加える代わりに SYCL* で書き換えた、GR-MHD の MPI + SYCL* バージョンである DPEcho を紹介します。DPEcho は、不安定性、乱流、波動の伝播、恒星風と磁気圏、およびブラックホール周辺の天体物理学的プロセスのモデル化に使用され、ミンコフスキーやコード化された一般相対性理論 (GR) メトリックで従来の MHD と相対論的 MHD をサポートしています。ここでは、DPEcho の公開バージョンを紹介し、SYCL* で複雑な現実のアプリケーションのアルゴリズムを表現する方法を説明し、このアプローチによって達成されるパフォーマンスを定量化します。

```
 CMAKE_BUILD_TYPE         Release
 SYCL
oneAPI
 GPU
ON
 ENABLE_MPI                      OFF
 SINGLE_PRECISION          OFF
 FILE_IO
VISIT_BOV
 PHYSICS
GRMHD
 METRIC
CARTESIAN
 FD
4
 NRK
2
 REC_ORDER                       5

SYCL: Select the SYCL target
architecture
Keys: [enter] Edit an entry [d] Delete
an entry
      [l] Show log output   [c]
Configure
      [h] Help              [q] Quit
without generating
      [t] Toggle advanced mode
(currently off)
```

図 1. DPEcho のインタラクティブな ccmake ターミナル・ユーザー・インターフェイス

DPEcho を使用する

DPEcho の公開バージョン (英語) は、Apache* 2.0 ライセンスの下に GitHub* で公開されています。このコードは CMake をサポートしており、`ccmake` のインタラクティブなターミナル・ユーザー・インターフェイスを意識して設計されています (**図 1**)。DPEcho は、このインターフェイスを介していくつかの SYCL* コンパイラーをサポートしています。`ccmake` 画面にコードの内容 (SYCL* キューセレクターと MPI による GPU サポート、単精度、I/O トグル) が表示されます。次に、物理と数値のオプションがあります。ソルバーは従来の MHD と GR-MHD の切り替えが可能で、関心のある問題に応じて、サポートされている GR メトリックの 1 つを選ぶことができます (公開バージョンではデカルトまたはミンコフスキーのみサポート)。数値オプションには、オリジナルの ECHO コードと同様に、数値導出 (FD)、可変グリッド再構成 (REC)、ルンゲ・クッタ・ソルバー・ステップ (NRK: 1-3) の異なるオーダーが含まれています。

[1] "ECHO: An Eulerian conservative high order scheme for general relativistic magnetohydrodynamics and magnetodynamics," Apr. 2007, doi: 10.1051/0004-6361:20077093.
[2] "ECHO-3DHPC: Advance the performance of astrophysics simulations with code modernization," Oct. 2018, doi: 10.48550/arxiv.1810.04597.

DPEcho の開発、解析、インテルの CPU および GPU での実行には、インテル® oneAPI ベース・ツールキットとインテル® oneAPI HPC ツールキットを使用しました。ユーザーは、MPI ランクを CPU コアに固定するのと同様に、MPI ランクと GPU リソース（タイルなど）のマッピングを指定できます。[3] これは実行時に環境変数を通じて行われます。パフォーマンス解析には、インテル® VTune™ プロファイラーを使用しました。最後に、インテル® MPI ライブラリーは、統合共有メモリー（USM）に配置されたメモリーセグメントなどを直接操作することができます。

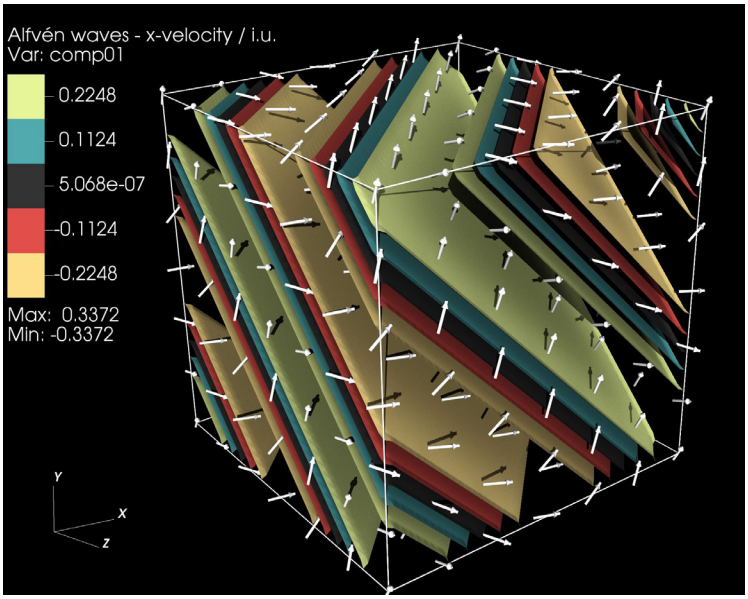

図 2. DPEcho 公開バージョンでモデル化された GR（x 速度成分）での平面アルヴェーン波のレンダリング。矢印は磁場（大きさは一様）をトレースします。

DPEcho の公開バージョンに存在する物理および数値手法により、**図 2** に示す平面アルヴェーン波のような相対論的磁化波のモデル化が可能です。異なる問題（GR-MHD 爆風など）をモデリングするインターフェイスも用意されていますが、ユーザー定義の実装が必要です。DPEcho の開発バージョンには、回転する天体物理学のブラックホールの周囲のモデリングを目的としたさまざまな GR メトリックや、非平面時空間を扱う一般的なインフラストラクチャーが含まれる予定です。これらの機能は一般公開の予定はありません。開発協力のリクエストには、個別に対応します。この点を除いて 2 つのバージョンは類似しており、互換性のあるインターフェイスを持ちます。

DPEcho の主な構造

DPEcho 開発中に最も苦労したのは、SYCL* のデバイス中心の構造をサポートする物理アルゴリズムと数値アルゴリズムの再設計でした。**図 3** は、DPEcho のホストとデバイスのコード、通信、およびフロー制御を示します。慎重な計画と USM の採用により、実行時間全体にわたって主要な変数をデバイスメモリーに保持することができました。これにより、(MPI デカルトグリッドでの分解に続く）各 MPI タスクのドメインの周りのゴーストレイヤーが、最小サイズのバッファーを介してコピーされ、データ転送が大幅に軽減されます。さらに処理するため、デバイス上のデータをロックしないように、出力ダンプも同様の原則に従います。単純な SYCL* 制御構造（例えば、ワークグループ、共有ローカルメモリー、およびキューの依存関係を除外する）のみを使用しても、データ転送ロックをバイパスし、読みやすいソースコードで高いパフォーマンスを実現できます。将来、さらに制御が必要になった場合、これらの高度な機能は、大きな問題なしに、効率良くベースラインに組み込まれます。これも SYCL* の主な利点です。

[3] "Intel MPI Library Developer Reference - GPU Support." https://www.intel.com/content/www/us/en/develop/documentation/mpi-developer-reference-linux/top/environment-variable-reference/gpu-support.html?wapkw=GPU%20pinning（英語）

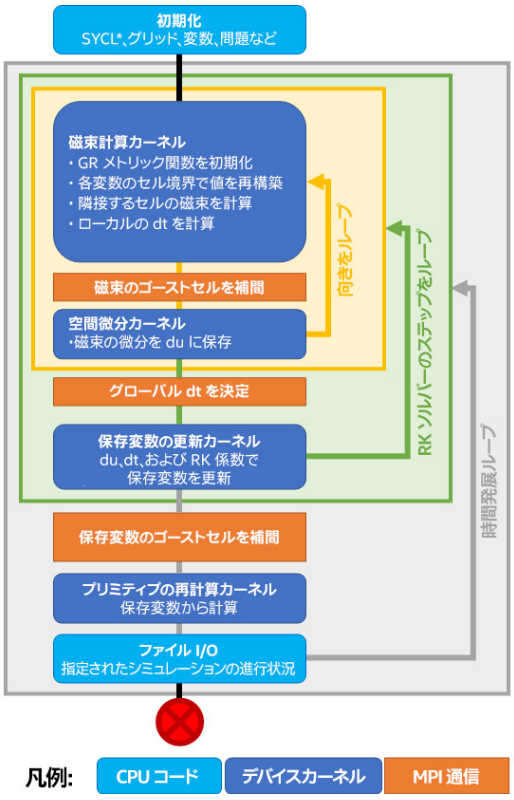

図 3. DPEcho のロジックの概要

USM の使用法を**図 4** に示します。まず、計算をオフロードするターゲットデバイスを選択します。ここでは、SYCL* ランタイムによって選択されるデフォルトの GPU を使用します。未定ベクトルの配列構造体として、データをデバイスメモリーに直接割り当てます。各ベクトル・コンポーネントを個別の配列に割り当てることで、ランタイムがタイル化された GPU アーキテクチャーでデータを正しく分割できるようになります。

アプリケーションの計算の大部分は、磁束計算カーネルで実行されます（**図 5**）。磁束とは別に、反復処理の以降のステップでは、安全なタイムステップの増分を決定するため、最大波動伝播速度の計算も必要です。SYCL* はリダクション・カーネルによってこれをサポートします。`parallel_for` は、計算の形状に加えて、共有リダクション・オブジェクトを受け取ります。デバイスの実行コンテキストでは、各ワークアイテムがローカル波速をサブミットします。最後に、SYCL* ランタイムがグローバルの最大値を決定します。

```
// SYCL* デバイスとキューを初期化
sycl::gpu_selector sDev;  sycl::queue qDev(sDev);


// USM で主要な変数を割り当て
double *v[VAR_NUM], *f[VAR_NUM], [...];
for (int i=0; i < VAR_NUM; ++i) {
  // 各変数のプリミティブと磁束
  v[i] = malloc_device<double>(grid.numCells, qDev);
  f[i] = malloc_device<double>(grid.numCells, qDev);
  [...]
}
```

図 4. 直接デバイスアクセスのため SYCL* USM で DPEcho の主要な変数を割り当て

```
//-- SYCL* の ranges: グリッドサイズとローカル・ワークグループ・サイズ
range<3> r = range(grid.nx, grid.ny, grid.nz),  rLoc = range(8, 8, 8);

[..]//-- メインのコードループ
  auto maxReduction = sycl::reduction(aMax+directionIndex, sycl::maximum<field>());
  qDev.parallel_for(nd_range<3>(r, rLoc), maxReduction, [=](nd_item<3> it, auto &max) {

   id<3> id    = it.get_global_id();              // SYCL* のインデックスとオフセット
   int myId    = globLinId(id, grid.nh, dOffset); // ゴーストレイヤーのカスタム・インデックス
   int dStride = stride(id, myDir, grid.nh);

   // GPU レジスターでローカル変数を宣言
   double vRecR[VAR_NUM], vRecL[VAR_NUM] , ...;
   Metric g(xCenter, yCenter, zCenter);    // GR の心臓部 : 各グリッドポイントのメトリック

   // 各変数を補間する空間を意識した関数 -> 並列化を考慮 !
   holibRec(myId, v, dStride, vRecL,  vRecR);
   physicalFlux(directionIndex, g, vRecL, vRecR, ...);

   [ ... ] // 磁束から特性を計算
   max.combine(localMax); // タイムステップ
```

図 5. DPEcho のデバイスコード例 : 磁束計算を行う parallel_for 構文（分かりやすくするため一部省略）

カーネルでは、各ワークアイテムが 1 つのグリッドセルの磁束を計算します。ワークアイテムのグローバル ID をグリッドセルにマッピングするため、カスタム「計算機クラス」の grid を使用し、ゴーストレイヤーをオフセットします。最初に Metric オブジェクトを作成します。Metric オブジェクトのコンポーネントは各セルでローカルに定義され、解析的に計算されるため、複数の追加変数を保存する必要はありません。次に、磁束計算のため、カーネルはまずセル境界で未定値を補間する必要があります。ここでは、コンパイル時に設定される、異なるオーダーの再構成をサポートしています。この補間は `holibRec` で実行されます。この関数は、ワークアイテム ID を使用して、シミュレーション・グリッド内の位置を決定します。現在は実装されていませんが、この機能は隣接するワークアイテム間でワークシェアリングを可能にします。例えば、セル中央の値をセル間でブロードキャストして、グローバルメモリーからの冗長なロードを回避できます。補間後の磁束計算は純粋にローカルであり、驚異的な並列性をもたらします。

パフォーマンスの比較

同じ問題設定を実行して、DPEcho とベースライン ECHO のスケーリングを比較します（**図 6**）。最初に、ライプニッツ・スーパーコンピューティング・センターにある SuperMUC-NG（英語）のインテル® Xeon® Platinum 8174 プロセッサー・ベースのノードで、弱いスケーリング・テストを実施します（**図 6** の左側）。16 ノードまでのスケーリングは完璧であり、MPI + OpenMP* のハイブリッドなベースライン・バージョンは最大 4 倍のパフォーマンス（1 ウォールクロック秒に 1 ステップで処理される 100 万単位のセル更新数で測定）を達成していることが分かります。DPEcho のメモリー・フットプリントが軽減されたことで、MPI タスクごとに 192^3 個ではなく 384^3 個のセルを配置できました。

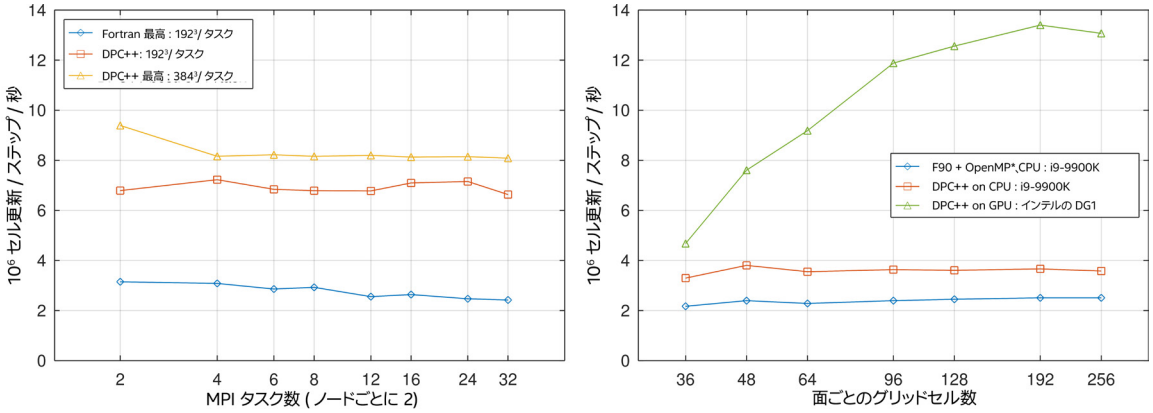

図 6. ECHO（Fortran バージョン）と DPEcho の比較。
左 : HPC ハードウェア（ライプニッツ・スーパーコンピューティング・センターの SuperMUC-NG のインテル® Xeon® Platinum 8174 プロセッサー・ベースのノード）での倍精度、CPU のみの弱スケーリング・テスト。4 倍のスピードアップに加え、メモリー消費量の減少により、シミュレーションを最大 8 倍まで拡大可能。右 : コンシューマー向けハードウェア（DG1 搭載の第 9 世代インテル® Core™ i9 プロセッサー）でグリッドサイズを大きくして単精度テストを実行。ディスクリート・グラフィックスは十分な大きさのワークロードを必要とするが、高速化されていない HPC ハードウェアに匹敵するパフォーマンスを達成。

HPC ハードウェアのほかに、コンシューマー向けのハードウェア（インテル® Iris® Xᵉ MAX グラフィックス（開発コード名 DG-1）搭載のインテル® Core™ i9-9900K CPU）でもテストしています。このテストでは、グリッドサイズのスケーリングを行い（**図 6** の右側）、常にフルノードまたはデバイスを使用します（ハードウェアの制限により単精度に切り替えています）。CPU でのベースライン・パフォーマンスは、HPC の結果と一致しています。DG-1 によりパフォーマンスは最大 7 倍向上し、高速化されていない HPC ハードウェアを上回っています。これは、SYCL* コードの優れた移植性と効率性を証明するものであり、あらゆるクラスのハードウェアを最も効率良く利用することができます。

今後の展望と考察

DPEcho には、磁場のソレノイド状態を維持する方法はまだ実装されていません。オリジナルの ECHO コードでは、磁場成分のスタッガリングと磁束の多次元近似リーマンソルバーに基づく戦略が採用されています。これはかなり複雑であるため、コミュニティーには SYCL* が基本的な役割を果たす、シンプルで拡張性のあるコードを提供しています。理想的でない効果（例えば、誘導方程式における物理的な粘性、抵抗、ダイナモの項）などのより複雑な物理オプションについても、同様の理由から実装されていません。開発バージョンでは、円盤からカー・ブラックホール解への降着シミュレーションがまもなくテストされる予定です。

個々のアップグレードがすべて実装されれば、DPEcho の可能性は確実に高まるでしょう。超巨大ブラックホールや恒星サイズのブラックホールからの降着流入と平行噴射の 3D 高解像度シミュレーションを詳細に研究し、衝撃波捕獲コードにつきものの数値拡散の影響を最小限に抑えながら、理想的な磁気回転不安定性と小さな再結合サイトの非理想的なプラズモイド不安定性の両方を解決できる可能性があります。当面は、公開バージョンで実験することを歓迎します。開発協力に興味のある方は、著者にご連絡ください。