名古屋大学 「不老・弐」2026年10月運用開始！

AIエージェントで25万行気象アプリをGPU化して分かったこと

星野 哲也 (名古屋大学情報基盤センター)
共同研究者：坪木 和久, 加藤 雅也(名古屋大学宇宙地球環境研究所)
椋木 大地, 片桐 孝洋(名大), 塙 敏博(東大)

研究背景

図1 代替テキスト
「スーツ姿の男性。片腕を組み、もう一方の腕を前に伸ばして手のひらを上に向け、語っている様子。」

- “「スパコンは使われてこそ。商業的成功なしでは意味がない」―。富岳NEXTの開発を主導する理化学研究所計算科学研究センター長の松岡聡氏は断言する（図1）。”
  - 引用元：日経 XTECH **「世界一は狙わない」NVIDIAと協業でAI時代の計算機へ**
    https://xtech.nikkei.com/atcl/nxt/mag/nmc/18/00192/00001/

つまり、アプリのGPU移植が課題である！

研究背景

図1 代替テキスト
「スーツ姿の男性。片腕を組み、もう一方の腕を前に伸ばして手のひらを上に向け、語っている様子。」

- “「スパコンは使われてこそ。商業的成功なしでは意味がない」―。富岳NEXTの開発を主導する理化学研究所計算科学研究センター長の松岡聡氏は断言する（図1）。”
  - 引用元：日経 XTECH **「世界一は狙わない」NVIDIAと協業でAI時代の計算機へ**
    https://xtech.nikkei.com/atcl/nxt/mag/nmc/18/00192/00001/

つまり、アプリのGPU移植が課題である！

十数年前からずっと言われている！

研究背景

図1 代替テキスト
「スーツ姿の男性。片腕を組み、もう一方の腕を前に伸ばして手のひらを上に向け、語っている様子。」

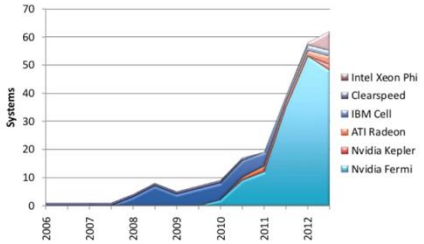
2013年　松岡研M1の時の学会発表スライド

TOKYO TECH Pursuing Excellence

Background

- Increased number of system using accelerators
  - ex. Titan, Tianhe, TSUBAME2.0
  - Porting of legacy CPU-based applications to accelerators is an important issue

Accelerators/Co-Processors



available at: http://www.top500.org

TITAN, 1st in top500 (Nov. 2012)
18688 NVIDIA K20 GPUs

TSUBAME2.0, 4th in top500 (Nov. 2010)
4258 NVIDIA Fermi GPUs

4

十数年前からずっと言われている！

研究背景

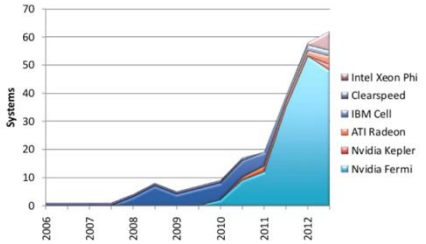
図1 代替テキスト
「スーツ姿の男性。片腕を組み、もう一方の腕を前に伸ばして手のひらを上に向け、語っている様子。」

2013年　松岡研M1の時の学会発表スライド

Background

- Increased number of system using accelerators
  - ex. Titan, Tianhe, TSUBAME2.0
  - Porting of legacy CPU-based applications to accelerators is an important issue

Accelerators/Co-Processors



available at: http://www.top500.org

TITAN, 1st in top500 (Nov. 2012)
18688 NVIDIA K20 GPUs

TSUBAME2.0, 4th in top500 (Nov. 2010)
4258 NVIDIA Fermi GPUs

4

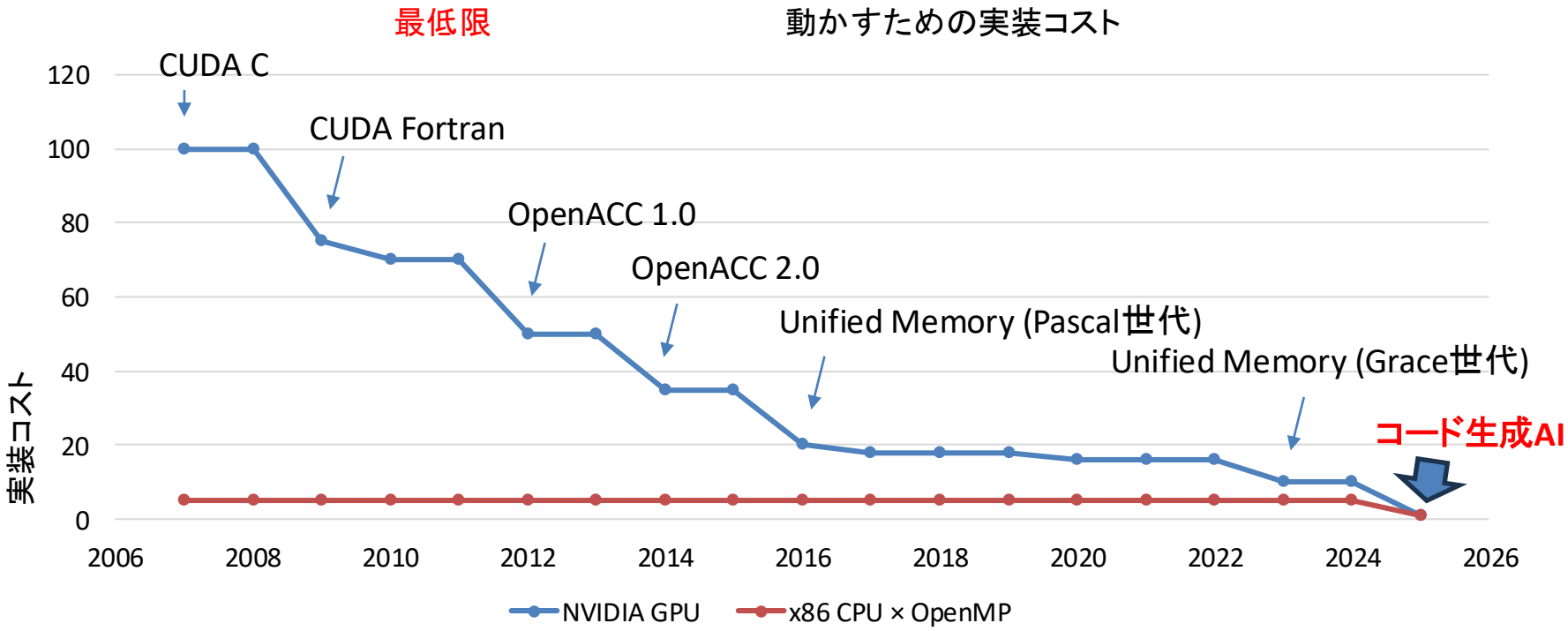
十数年前からずっと言われている！

アプリのGPU移植はなぜ進まなかったか？

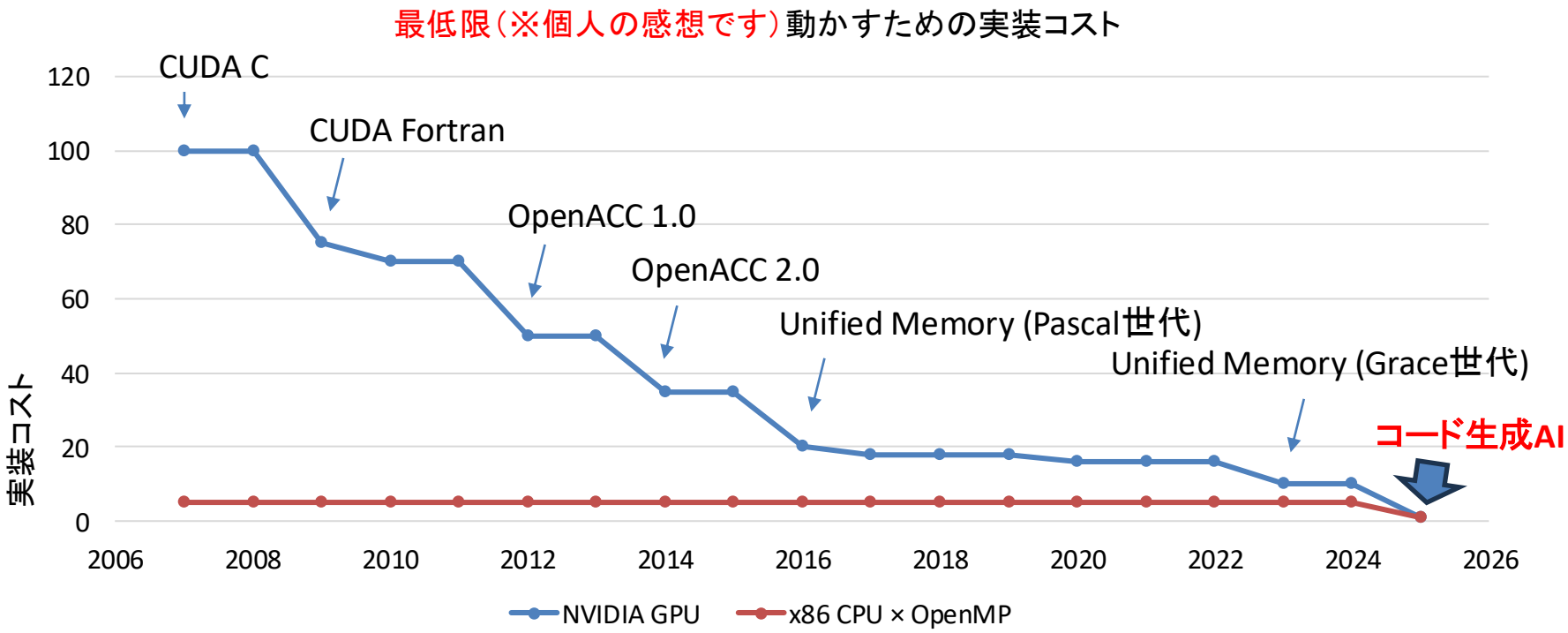
- 日本のアプリはほぼ Fortran
  - Fortran 対応が有償のサードパーティ製コンパイラ※（～2020年頃）
    - ※2009年PGI CUDA Fortran登場、2013年NVIDIAがPGI買収、2017年～2020年頃は無償版としてCommunity Editionあり
  - なのでまずC/C++に直します ！　←この時点でかなり無理！
- 保守コストの増加
  - CPU版 + GPU版
    - 場合によっては、GPU版 = CUDA版 + OpenACC版 + OpenMP版 ・・・
- GPU実装の属人化
  - ○○さんにGPU版実装してもらったが、その後放置

移植・保守コストが高く、かつ人材不足！

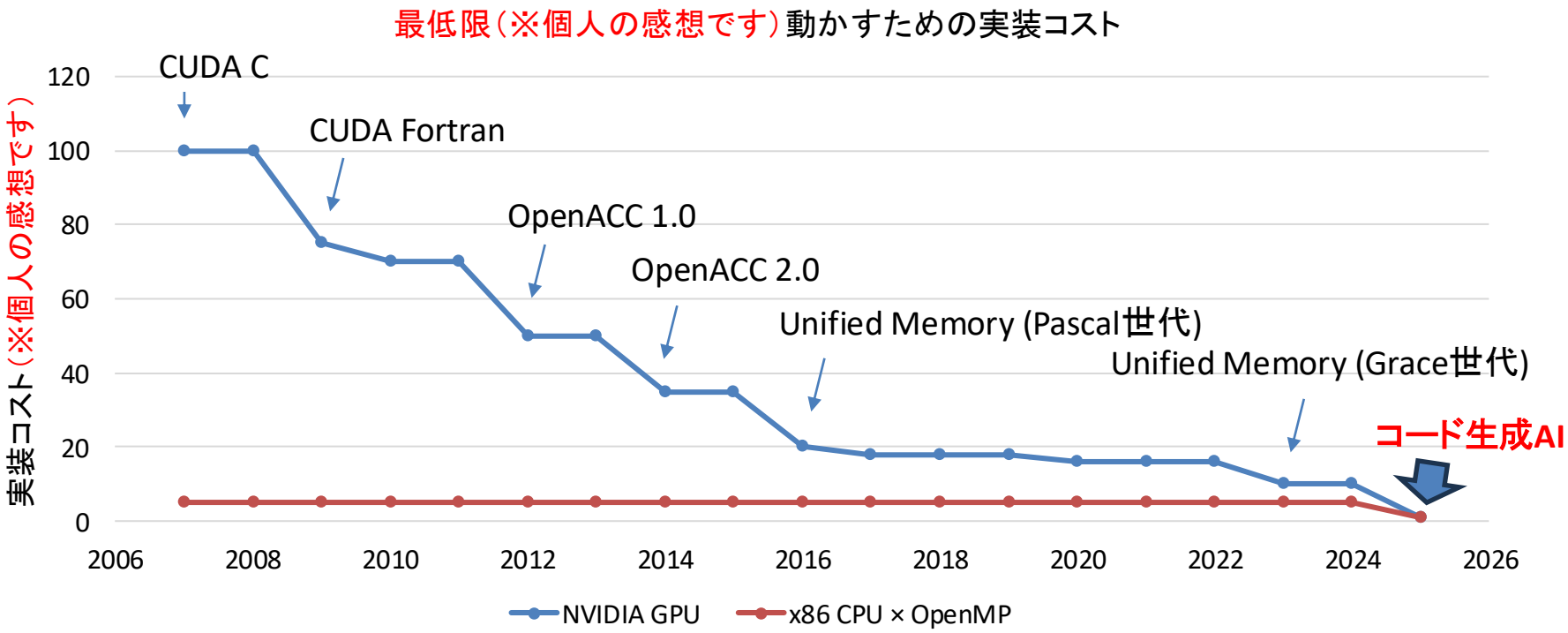
FortranカーネルのGPU（NVIDIA）向け実装コストは下がっている



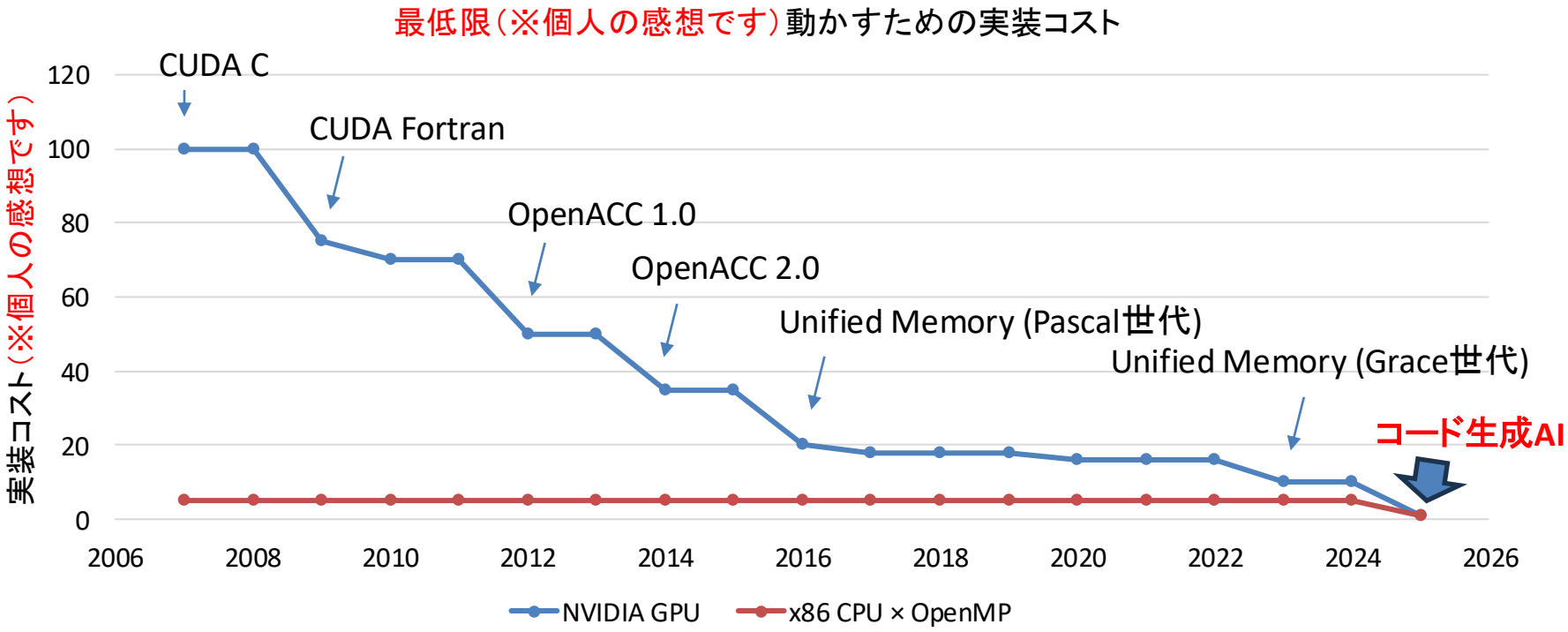
FortranカーネルのGPU（NVIDIA）向け実装コストは下がっている



FortranカーネルのGPU（NVIDIA）向け実装コストは下がっている

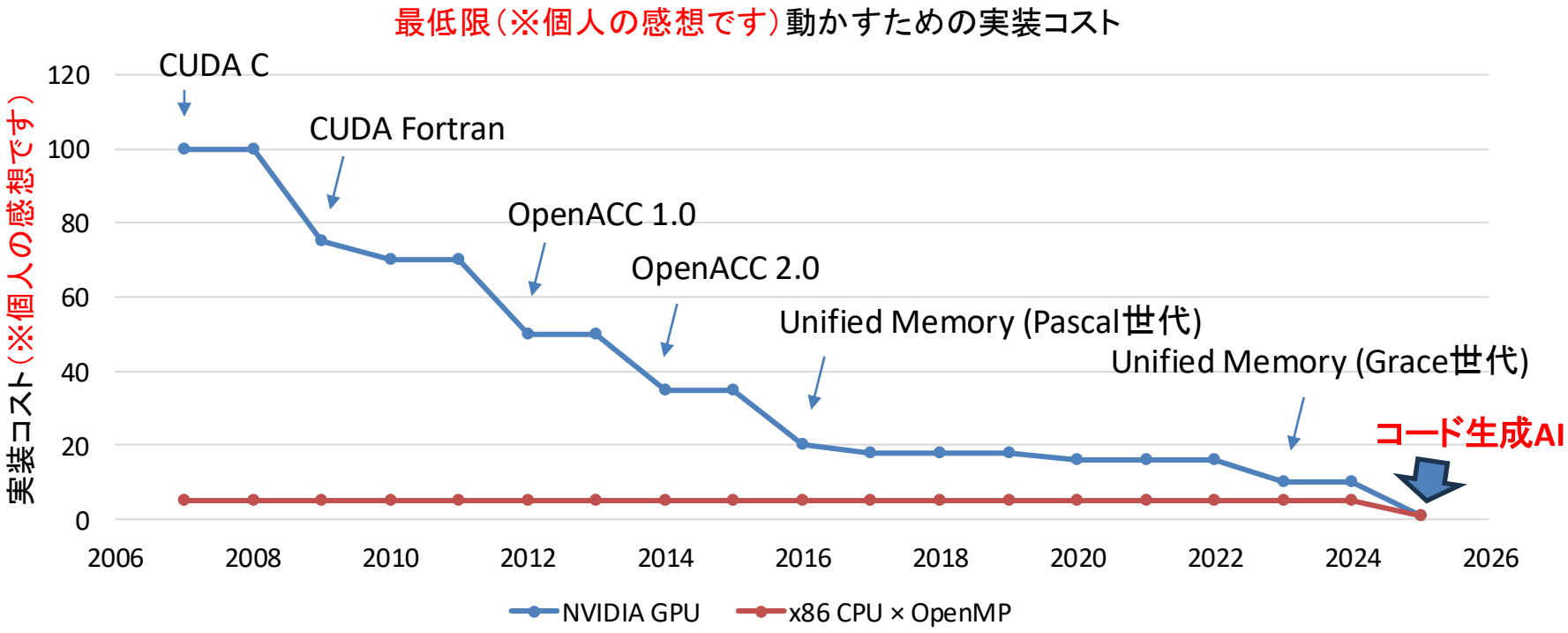


FortranカーネルのGPU（NVIDIA）向け実装コストは下がっている



最低限のGPUカーネル実装はもはやAIで十分

FortranカーネルのGPU（NVIDIA）向け実装コストは下がっている

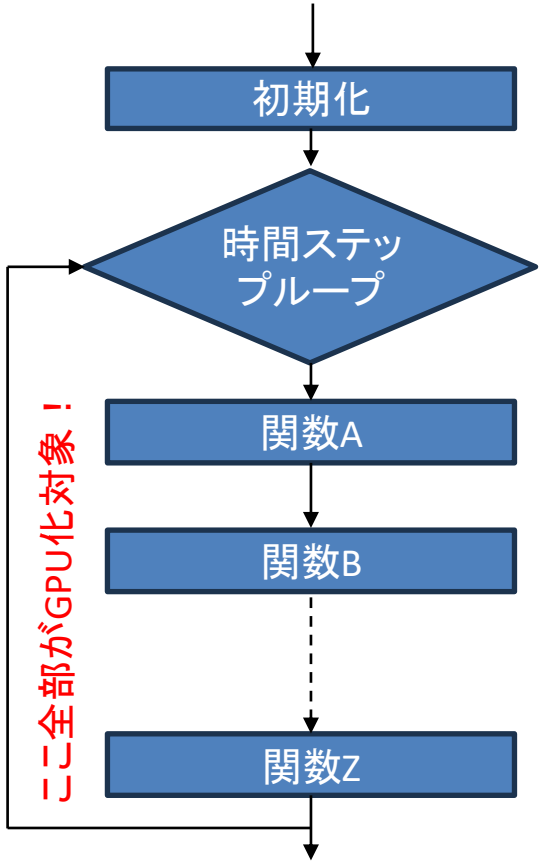


最低限のGPUカーネル実装はもはやAIで十分（※個人の感想です！）

GPUカーネル実装はGPU移植の一要素でしかない

- GPU移植の大まかな流れ
  1. コード解析・プロファイリング・方針決定
  2. For all 時間発展ループ内関数
     1. GPUカーネル実装
     2. 検証
  3. 全体最適化
- 検証コストが重い
  - 並列化により計算順序は変わるので、数値精度も検証対象
  - 検証のためにベンチマーク化することが望ましい

検証含めたGPU移植ワークフロー全体をAI活用で効率化したい



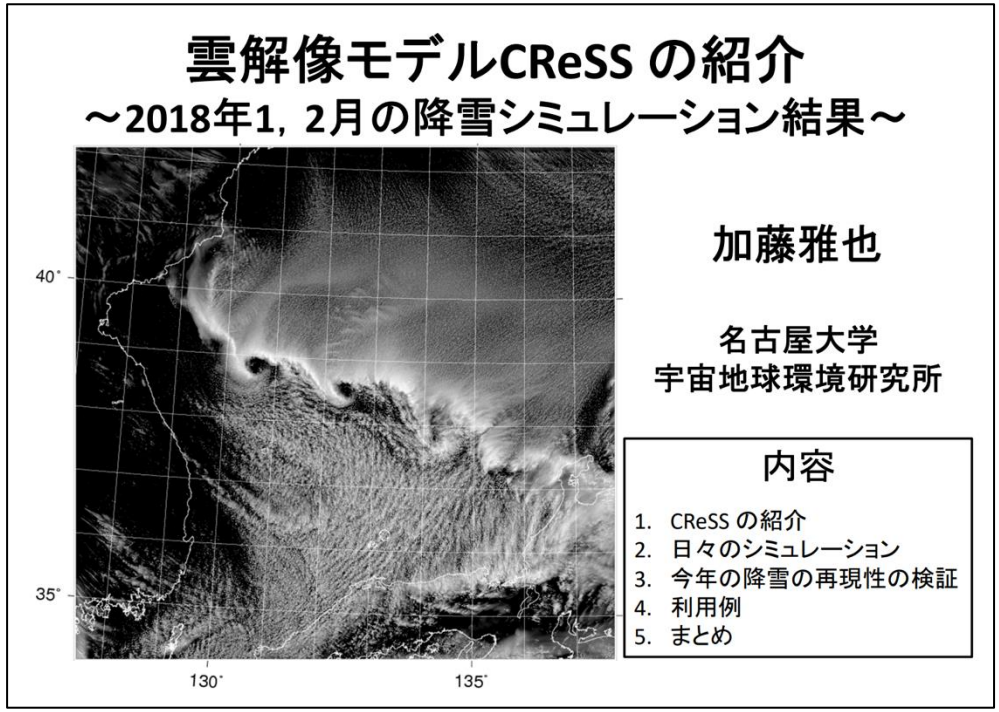
CLIベースのコード生成AIエージェント

- Claude Code, Codex, Gemini など
- 2025年5月（Claude Opus 4.0発表）くらいから本格的に使われだしたと思われる
- コード開発のための対話型AI アシスタントとして機能
- ファイルシステムへ直接アクセスしてのコード開発が可能
- コード開発からテスト実行まで一連のタスクを自立的に実行可能

気象コードCReSS https://www.rain.hyarc.nagoya-u.ac.jp/CReSS/fcst_exp.html

- 名古屋大学宇宙地球環境研究所 坪木先生のチームが開発
- 1998年開発開始
- Fortran 90 + OpenMP + MPI
- 25+万行, 599 Fortran ファイル, 387 OpenMP parallelループ

ー昔前なら、GPU移植には年単位の期間が必要な規模



引用元：https://diasjp.net/wpsite/wp-content/uploads/2018/04/DIASForum2018_Kato.pdf

先に結果から

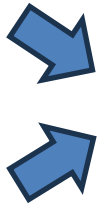
- Claude Opus 4.5 - 4.6の支援により、2026年1月～3月の期間でGPU化達成
  - ただし、対象としたのは2022年大西洋台風シミュレーションの１パス実行に必要な162/387 OpenMPループ
  - OpenACC + Unified memory実装、シングルGPUのみ検証
  - 期間は３ヶ月程だが、実働時間は100+時間くらい
- AIセッション切断時の文脈維持のため、タスクの仕様書化による記憶の引き継ぎが重要
- 完全自動化ではなく、人間とAIの役割分担
- 時間が掛かったのはGPUカーネル生成ではなく、検証のためのベンチマーク化
  - 特に、ベンチマークを実行するためのダンプデータの取得に苦戦

AI支援による大規模アプリのGPU化の課題と対応方針

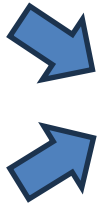
- AIは忘れる
  - コンテキスト溢れ
  - セッション切断
- 人間側がついていけない
  - AIに一度にたくさんの作業をやらせると把握できない
- 実行環境にGPU必須
  - 高価なGPUはユーザ端末には付いてない
- セキュリティなど
  - AI使用に関する業界内でのコンセンサスが取れてない

AI支援による大規模アプリのGPU化の課題と対応方針

- AIは忘れる
  - コンテキスト溢れ
  - セッション切断
- 人間側がついていけない
  - AIに一度にたくさんの作業をやらせると把握できない
- 実行環境にGPU必須
  - 高価なGPUはユーザ端末には付いてない
- セキュリティなど
  - AI使用に関する業界内でのコンセンサスが取れてない



仕様書化による記憶の外部化
タスクの細分化



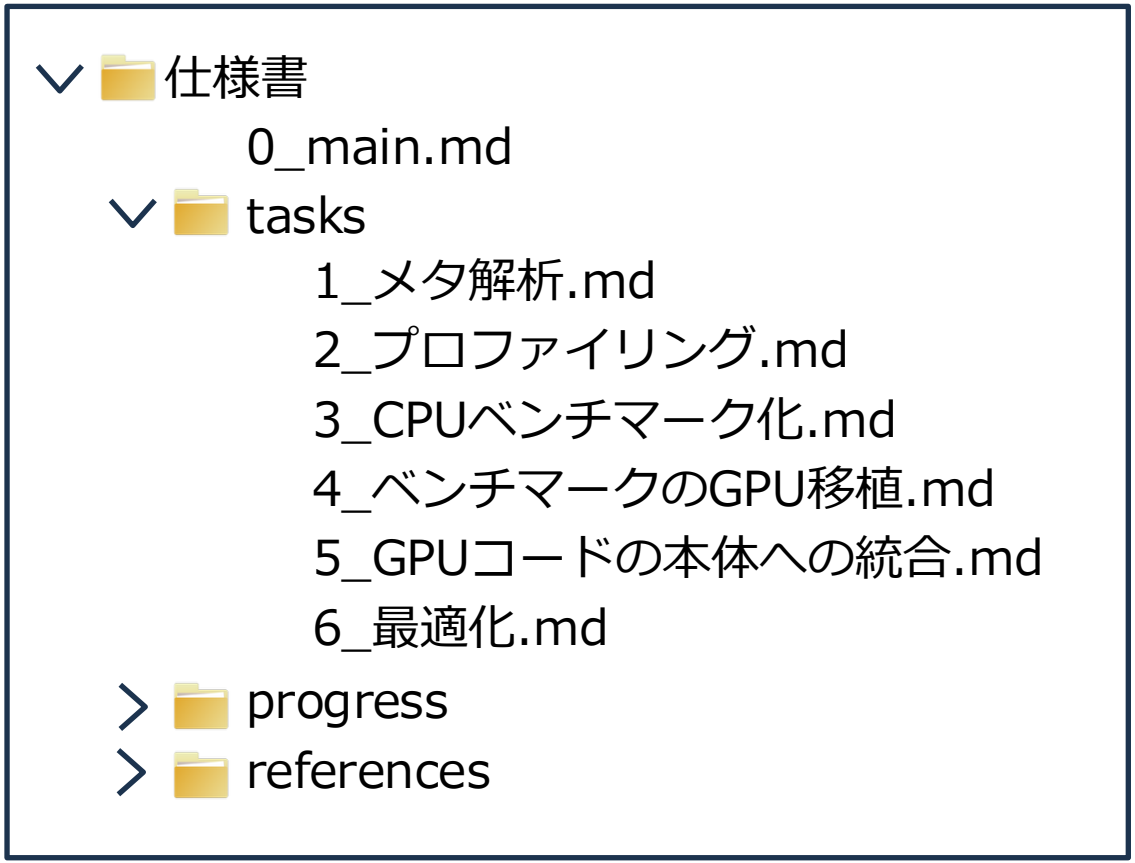
スパコン（Miyabi）のインタラクティブジョブ内でのみAIエージェント起動

GPU移植環境

- JCAHPC（東大・筑波大）の Miyabi-Gノード群
  - GH200
  - CPU-GPU 間がNVLink C2C （片方向450 GB/s）で接続
  - インタラクティブジョブ実行は２時間がMAX
    - 従って、２時間毎にAIセッションが切断される
- ソフトウェア環境
  - GPU向けコンパイラ：nvfortran 25.9（デフォルト）
  - Claude Opus 4.5 - 4.6
  - 利用料金：200 USD / 月 （当時）
  - Claude Code は --dangerously-skip-permissions を付与して起動
    - ○○していいですか？の確認をいちいちされなくなる
    - ユーザ権限で実行可能なコマンドは全て実行され得る

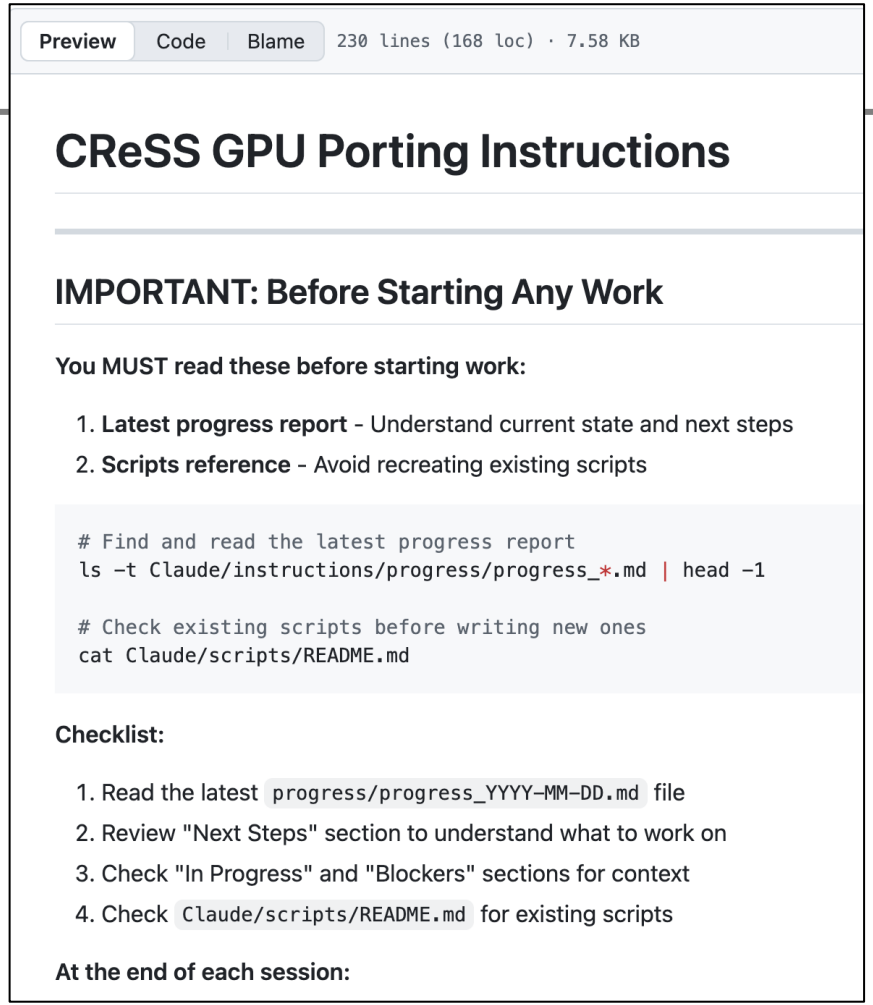
AI支援型GPU移植ワークフローの仕様書構造

- 基本的に人間がこれまでやってきたGPU移植ワークフローを踏襲
- 仕様書も細部はAIに書かせる
- コンテキストの消費を抑えるために階層構造にしたが、効果は未検証



0_main.md

- 「まずここを読め」のファイル
- プロジェクト全体の目的・進め方
- CReSSのディレクトリ構成・ビルド方法
- 各フェーズ仕様書の簡易説明
- Progress Report について
  - まずProgress Reportがあるかどうか確認せよ
  - セッション終了時にProgress Reportを更新せよ
- Git の扱いなど



Preview | Code | Blame | 230 lines (168 loc) · 7.58 KB

CReSS GPU Porting Instructions

IMPORTANT: Before Starting Any Work

You MUST read these before starting work:

1. **Latest progress report** - Understand current state and next steps
2. **Scripts reference** - Avoid recreating existing scripts

```
# Find and read the latest progress report
ls -t Claude/instructions/progress/progress_*.md | head -1

# Check existing scripts before writing new ones
cat Claude/scripts/README.md
```

Checklist:

1. Read the latest `progress/progress_YYYY-MM-DD.md` file
2. Review "Next Steps" section to understand what to work on
3. Check "In Progress" and "Blockers" sections for context
4. Check `Claude/scripts/README.md` for existing scripts

At the end of each session:

1_メタ解析.md (1/2)

- AIにコードレビューをやらせるフェーズ
- meta_infoコメントを全OpenMP領域に付与
- meta_infoの内容
  - AI判断のGPU化難易度 Easy/Medium/Hard
    - Opus 4.5の判定 → Easy: 236, Medium: 121, Hard: 30
    - 何を持ってHardとしたのかなどのコメント
    - コンテキストの消費を抑えられるのではないか、という期待してのことだが、意味があったのかどうかは不明
  - 発見した並列化阻害要因

```
!@llm start meta_info ----------------------------------------------------
! Location: eddyvis.f90 :: subroutine s_eddyvis
! Summary : Calculates eddy viscosity and turbulent length scale using
!           Smagorinsky or Deardorff (TKE-based) formulations with
!           stability corrections.
! GPU diff: Medium
! Findings:
!   - No omp_get_thread_* usage.
!   - No function calls inside parallel region.
!   - Reads module constants (oned3, csnum, ckm, ckmin, prnum, kappa, eps)
!     from commath/comphy.
!   - No synchronization constructs.
!   - Uses intrinsic abs(), exp(), log(), max(), min(), sqrt() - all GPU ok.
!   - Complex nested conditionals (tubopt, isoopt, sfcopt, mfcopt, mpopt).
!   - Thread divergence likely due to many branching paths.
!   - All grid points are independent within selected code path.
! Next:
!   - Consider restructuring conditionals for GPU - evaluate outside kernel.
!   - Use template/variant approach for different physics configurations.
!   - Intrinsic functions are well-supported on GPU.
!@llm end meta_info ------------------------------------------------------
```

1_メタ解析.md (2/2)

- 実際に重要だったのは、OpenACC化阻害要因の有無の判定
  - omp_get_thread_id(), atomic, critical, グローバル変数への書き込み, 関数呼び出し, mallocなどシステムコール, 間接参照など
  - 以下はClaudeの統計レポートで、CReSSにおいては阻害要因なし

Check Item	Result
`omp_get_thread_num` / thread-ID dependency	**0 occurrences** — not used
`!$omp atomic` / `critical` / `ordered`	**0 occurrences** — not used
Function calls inside parallel regions (`call`)	**0 occurrences** — all calls are outside parallel regions
System/runtime functions (`malloc` , `free` , `system` , `getenv` , etc.)	**0 occurrences** — not used
Writes to global/module variables	**None** — module access is read-only
`reduction()` clauses	**25 files** — directly supported by `!$acc loop reduction()`

これを受けて人間が、OpenACC化の方針で確定する

2_プロファイリング.md

- コードの検証をする上で妥当なサイズの問題を用意
  - 2022年9月の西太平洋における台風シミュレーション
  - 899 × 899 × 128 格子 (約1億格子）, 2 kmメッシュ
  - シミュレーション内時間1,800秒
  - 主要なステップは全部通過
- CPU実行時情報として以下を各OpenMP領域のmeta_infoに追加
  - 呼び出し回数
    - テストコードで実際に呼び出されたのは 162/387 OpenMP領域
    - ダンプデータ取得時にも重要
  - 実行時間
  - ループ長
    - GPU 並列化する上で、短すぎるループを検出

3_CPUベンチマーク化.md：概要

- ここが最も大変だった場所
- 全ての OpenMP parallel 領域を独立実行できるベンチマークにする
- 検証に用いる入出力データ
  - 台風シミュレーション実行時の、最終呼び出し時のダンプデータ
    - 雲の発生やら必要な過程を全て通過した後の数値を用いたいため
  - OpenMP実行に必要な全ての変数についてダンプ
- やりたい内容はこれ↓だけ

```
For all OpenMP 領域
    ベンチマーク生成
    ダンプデータ生成
    ベンチマーク検証
End for
```

3_CPUベンチマーク化.md：テスト実行時間問題

- しかし、ダンプ出力のためのシミュレーション実行に１時間掛かる
- これ↓はダンプだけで100時間以上掛かるので、こう↓する

```
For all OpenMP 領域
    ベンチマーク生成
    ダンプデータ生成
    ベンチマーク検証
End for
```

```
For all OpenMP 領域
    ベンチマーク生成
    必要変数リスト生成
End for

ダンプデータ生成

For all OpenMP 領域
    ベンチマーク検証
End for
```

3_CPUベンチマーク化.md：変数リストの作り方

- OpenMP の **default(none)** を使って、敢えてコンパイルエラーを起こすことで必要変数をリスト化
  - コードが長くAI（Opus 4.5-4.6）の静的解析だけだと変数を見逃すため
- 変数に対し、in / out / inout などの属性を付与
  - ここはAIの静的解析を信じる
  - メタ解析によりユーザ定義の関数呼び出しがないことがわかっているため、変数が左辺に出てくるか右辺に出てくるかだけ見れば良い
- 条件付きアクセスの変数の条件を抽出
  - AIの静的解析を信じる方針だが、**ここが苦しい**

```
1    # name  type    rank    attributes  omp_clause  inout   condition
2    advopt  integer 0   local   shared  in  -
3    sfcopt  integer 0   local   shared  in  advopt.le.3 .or. advopt.gt.3
4    cphopt  integer 0   local   shared  in  fmois(1:5).eq.'moist'
```

変数リストの例

3_CPUベンチマーク化.md：条件付きアクセスが苦しい理由

- CReSS はメモリ節約のため、ダミー配列を使用している
  - 実際には1要素配列なダミー配列の例

```
      if(savmem.eq.0.or.                                                &
&  (advopt.le.3.and.(abs(cphopt).ge.4.and.abs(cphopt).lt.20))) then
        allocate(nwtr(0:ni+1,0:nj+1,1:nk,1:nnw),stat=cstat)
      else
        allocate(nwtr(0:0,0:0,1:1,1:1),stat=cstat)
      end if
```

- OpenMP領域のあるサブルーチン側では↓のように変数宣言される

```
subroutine s_swp2nxt (…, nwtr,… )

      real, intent(in) :: nwtr(0:ni+1,0:nj+1,1:nk,1:nnw)
```

よって、sizeやallocated 組み込み関数ではダミーかどうか判定できない
リファクタリングも全体へ影響するのでしたくない

3_CPUベンチマーク化.md：条件付きアクセスが苦しい理由

- こんな if文のネストから条件式を抽出する必要がある
  - 1000行以上あるコードのif文構造だけ抜き出したもの

```
if(advopt.le.3) then

      if(fmois(1:5).eq.'moist') then

            if(abs(cphopt).lt.10) then

                  if(abs(cphopt).eq.4) then

                        …=nwtr(i,j,k,1)

                  end if

             else if(abs(cphopt).gt.10.and.abs(cphopt).lt.20) then

                  if(abs(cphopt).ge.11) then
                        …=nwtr(i,j,k,n)
                  end if
             end if
        end if
  end if
```

正しく抽出すると

(advopt.le.3 .and. fmois(1:5).eq.'moist' .and. (abs(cphopt).eq.4 .or. (abs(cphopt).ge.11 .and. abs(cphopt).lt.20)))

AIの静的解析では厳しい

↑こういうのが数百箇所あり、ミスると致命傷

3_CPUベンチマーク化.md：なぜ致命傷か

- 致命傷な例
  1. ダンプのためのシミュレーション実行（1時間）
  2. 条件抽出ミスでダミー配列をダンプしようとし、Segmentation fault
  3. Segmentation fault がダンプ関数で発生するため、ダンプ関数のバグと誤認
  4. ダンプ関数をなんか修正し、１へ戻る
- 致命傷回避も重症な例
  1. ダンプのためのシミュレーション実行（1時間）
  2. Segmentation fault や変数ダンプ漏れ
  3. 当該変数部分**だけ**を修正し、１へ戻る

挿入する配列ダンプが2000箇所以上（うち条件付きが数百）あり、
ダンプ挿入成功率99%でもミス20箇所 → 20時間掛かる

3_CPUベンチマーク化.md：対処法

- 致命傷への対処法
  - 単にプロンプトで教えるだけだと次のセッションで忘れてるので、ダミー配列への対処方法をちゃんと仕様書に書く

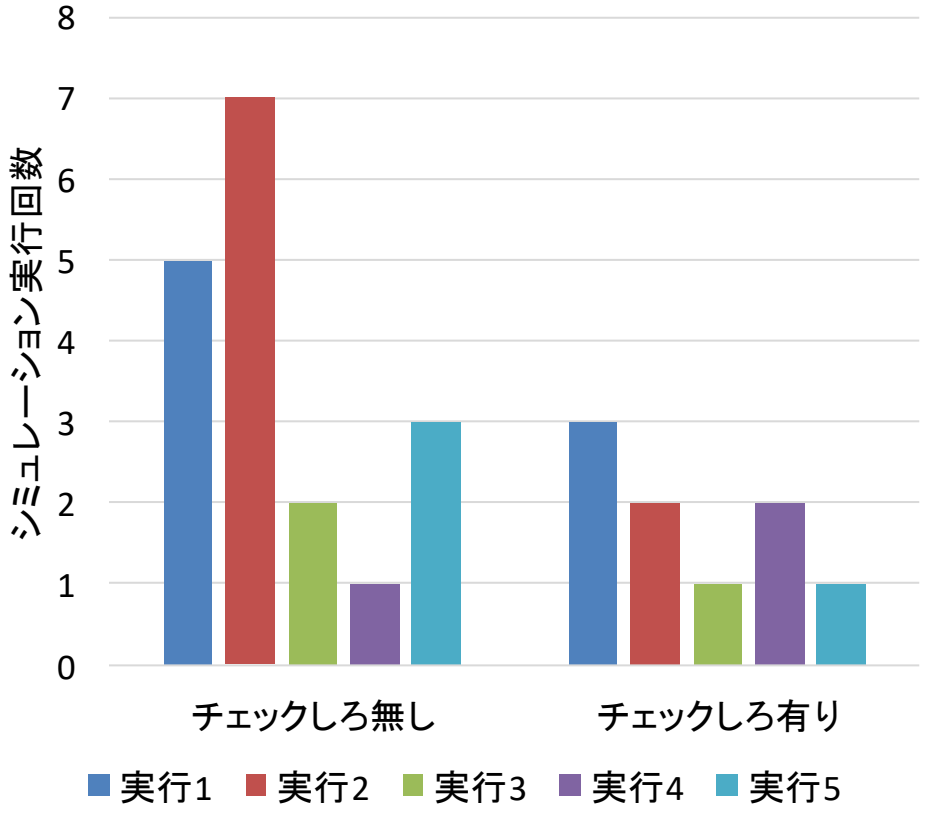
  ↑ Opus 4.5-4.6ではこれを書かないとベンチマーク化が成功しなかった

- 重症への対処法
  - 変数リストを「ダブルチェックしろ」と仕様書に書く
  - シミュレーション実行には時間が掛かるので、一回ミスを発見したら「同様のミスが無いか再チェックしろ」と仕様書に書く

  ↑ 実時間がどのくらい掛かってるかをAIは自主的に気にしてくれない

3_CPUベンチマーク化.md：「チェックしろ」の効果

- ダミー配列対処方法は仕様書記載済み
- テストシミュレーションをさらにミニテスト化して検証
  - シミュレーション内時間：1800秒 → 180秒
  - ベンチマーク化対象カーネル：162カーネル → 主要15カーネル
  - ベンチマーク化終了まで、全5回実行
  - タスク強制終了時間：２時間 → 30分
    - 30分経ったら、「直ちに作業をやめて進捗レポート書け」と指示



4_ベンチマークのGPU移植.md

- OpenACC kernels指示文を使うよう指示
  - parallel指示文を使いたがるが、parallel指示文は並列粒度をちゃんと設定しないとまともに動かないため
- データ指示文は使わず、Unified memoryで実装するよう指示
  - データ指示文はカーネル間に依存関係が生じるため
  - カーネル単位で処理を閉じることで、短タスク化
- 結果の検証
  - ダンプデータと全配列要素比較
  - 10^{-5}（単精度なので）以上の数値誤差が１要素でもあれば、エラーとして報告
  - **数値誤差の妥当性は人間が検証**

4_ベンチマークのGPU移植.md：検出された数値誤差

- 以下の５例検出（全て１要素の数値エラー）

bruntv.f90	境界条件判定差 (t <= tlow)
disptke.f90	exp(), log(), sqrt() の超越関数の実装差による Catastrophic cancellation
cloudcov.f90	ゼロ近傍の相対誤差
siadjst.f90	境界条件判定差 (qi > dqi)
swadjst.f90	境界条件判定差 (qc > dqc)

- アプリ開発者と議論の上、問題ないことを確認

↑これが現実的な時間でできるようになったのが革命的！

具体例 (bruntv.f90):

```
! 温度を計算
t = pt(i,j,k) * exp(rddvcp * log(p0iv * (pbr(i,j,k) + pp(i,j,k))))

! 低温補正の分岐
if (t <= tlow) then    ! tlow = 233.16K
  a(i,j,k) = a(i,j,k) + (lf0 + cwmci * (t - t0))
end if
```

- CPU計算結果: t = 233.16002K
  → 条件は False → 潜熱項を加算しない
- GPU計算結果: t = 233.16000K
  → 条件は True → 潜熱項を加算する

5_GPUコードの本体への統合.md

- アプリケーション本体への統合を目的としたフェーズ
- #ifdef KERNELID を使って CPU/GPU 実装のオンオフをカーネル毎に切り替えられるように指示
- シミュレーション内時間1,800秒経過時、右のa_1,a_2が10^{-4}以下ならvalidation OK
- validation NGなら、#ifdefを使って二分探索
  - カーネル単位の検証は、そのカーネルの最終呼び出し時のダンプとしか比較しないため、途中ステップでしか呼ばれない分岐処理が省略されているケースが2-3件検出された

```
#ifdef USE_GPU_KERNELID
   GPUカーネル
#else
   CPUカーネル
#endif
```

$$a_1 = \frac{\left|pp_{\max} - 1.140663 \times 10^3\right|}{1.140663 \times 10^3}$$
$$a_2 = \frac{\left|pp_{\min} + 3.927225 \times 10^3\right|}{3.927225 \times 10^3}$$

6_最適化.md

- NVIDIA Nsight を使用してプロファイリングを指示
  - 全GPUカーネルに対しルーフライン解析
  - CPU-GPU間のUnified memoryによるデータ移動の妥当性解析
- プロファイリング結果を元に最適化を指示
  1. 遅いカーネルを対象に、GPUベンチマークの最適化
  2. 結果検証
  3. 本体へ統合
- 禁止事項
  - data指示文、async節の使用
  - データ構造変更など、全体に影響する最適化
  - カーネル融合
  - MPIの通信隠蔽

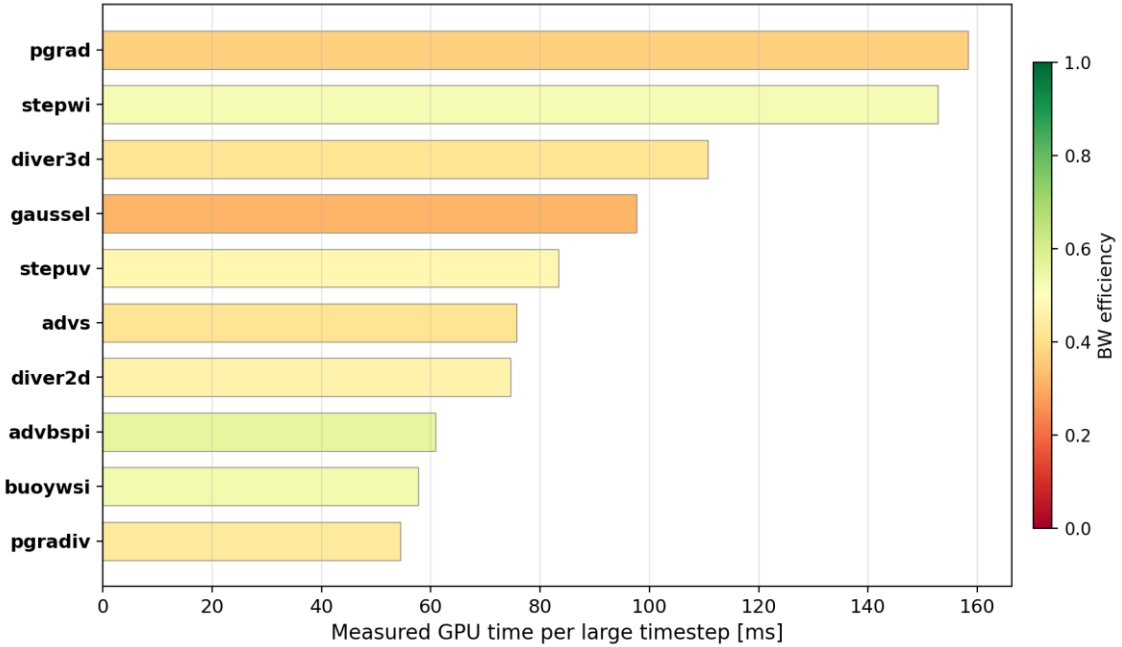
速くなったのは数カーネルだけ

カーネル間依存が生じ、検証が難しいため、現段階では禁止

GPU化で得られた性能

- 全162カーネルをプロファイリング
- メモリバンド幅効率は35-60%程度で、OpenACC実装としては妥当
- 右は実行時間が大きいtop 10 カーネルだけ抜き出したもの
- GH200使用時の1ステップあたりの高速化が、Grace CPU (72 thread)比で5.1倍

実行時間 top 10 カーネル（全体の65%を占める）の実行時間とメモリバンド幅効率



AIと人間の役割分担まとめ

Phase	AIの役割	人間の役割
1_メタ解析 2_プロファイリング	解析、メタ情報付与、レポート	以降の方針決定、後続Phase仕様書化指示
3_CPUベンチ	ベンチ作成、ダンプデータ取得	安定動作確認
4_GPUベンチ	OpenACCコード作成	GPU化に伴う数値誤差の妥当性検証
5_本体統合	#ifdef付きで本体へ統合	シミュレーション全体の妥当性検証
6_最適化	（現時点では）カーネル単位最適化	最適化に伴う数値誤差の妥当性検証

人間側の主な役割はAIの監視・安定化と、数値誤差の妥当性検証

制限事項

- 以下は今回のケーススタディでは検証していない
  - **高度な**GPUカーネル生成
  - カーネル間依存の生じる、広範囲影響な最適化
    - 計算と通信のオーバーラップ
    - OpenACCのasync節
    - データ構造の変更
    - カーネル融合
- 運用上の懸念：ダンプデータ爆発
  - 1 テストケース × 1 MPIプロセス × 1タイムステップ × 162/387 カーネル分のダンプ → **500+GB**

みんなが本格的にこれをやり出すとストレージ死ぬかも

まとめ

- AIアシストにより、25万行超の大規模アプリであるCReSSをGPU移植し、その際の知見を共有した
- AIのコンテキスト・セッション制約や、人間側の認知のために、タスクの細分化や仕様書化が重要
- CReSSではカーネルのOpenACC化は特に問題にならず、検証のためのベンチマーク化が問題であった
  - 特にClaude Opus 4.5-4.6は、ジョブの実時間をあまり考慮せずにタスク実行やエラー修正をする傾向
- GPU化に伴う数値誤差の検出が現実的な時間で可能となった
- 今後はより難しいカーネルを含むアプリでの検証、ローカルLLMによる検証、カーネル間で依存が生じる最適化の実施方法の検討が課題