Fortranアプリケーションを対象とした生成AIによるGPUコード開発の初期評価

星野 哲也, 林 俊一郎, 椋木 大地, 片桐 孝洋(名大),
塙 敏博(東大)

研究背景

2025年8月22日

理化学研究所

← 前の記事 ↑ 一覧へ戻る → 次の記事

理化学研究所、富士通およびNVIDIAとの国際連携による「富岳NEXT」開発体制を始動

－計算による課題解決を支える次世代「AI-HPCプラットフォーム」構築へ－

English Page

理化学研究所（五神真理事長、以下「理研」）は、富士通株式会社（時田隆仁代表取締役社長、以下「富士通」）[1]、NVIDIA Corporation（ジェンスン・ファンCEO、以下「NVIDIA」）[2]とともに、スーパーコンピュータ「富岳」[3]の次世代となる新たなフラッグシップシステム（開発コードネーム：「富岳NEXT」）に関して、理研を開発主体とした国際連携により設計および開発を開始することとしました。日本のフラッグシップシステムとし

- レガシーアプリケーションのGPU対応はいよいよ待ったなしであるが、様々な課題があって進み具合は良くない
  - 特にレガシーアプリに多いFortranで記述されたプログラムが課題

GPU対応における課題

- ベンダーによってまちまちな並列プログラミング言語／モデル対応

各GPUベンダの自社コンパイラによる対応状況

ベンダ	OpenACC	OpenMP (GPU)	CUDA	SYCL	HIP	言語標準の並列化 (stdpar)	OpenCL
NVIDIA	✓	✓	✓	—	—	✓	C のみ
AMD	—	✓	—	—	C のみ	C のみ	C のみ
Intel	—	✓	—	C のみ	—	✓	C のみ

- 保守コストの増加
  - CPU版 + GPU版 ×プログラミング言語・モデル数
    - 場合によってはCへの変換も必要

↑ AIを使ってどうにかしたい

コード生成AI Claude Codeの特徴

- Claude Code
  - Anthropic が開発したCLIツール
  - 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
  - Claude Opusなどの大規模言語モデルと連携
    - 日本語で指示可能
- Claude Opus 4.1
  - 2025年8月にリリースされたAnthropic の最新大規模言語モデル
  - Claude Codeと連携し、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

↑ GPU向けコード開発でどれだけ使えるかを評価したい

Claude Code のスパコンでの利用

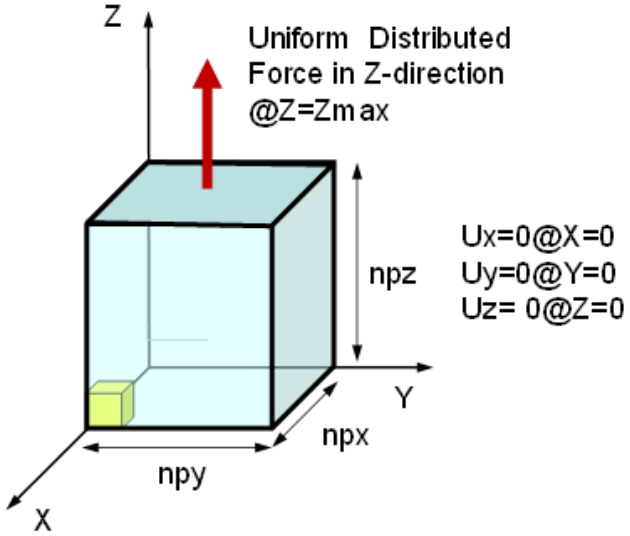
- ログインノードや計算ノードにインストールして使用
  - 推論は外部サーバで実行される（要外部接続）
  - 使用の際にはAnthropicとの契約が必要
    - 個人向け最上位プランでUSD 200/月
- ファイルシステム上のソースコードを直接編集・実行
  - 権限が与えられるのはデフォルトでは起動したディレクトリ以下
  - ファイルの編集や削除の際に、実行の許可を求められる
    - Claude Code起動時に--dangerously-skip-permissions を付与すると、実行許可の確認はスキップされる
  - ユーザ権限で利用できるコマンドであれば使用できるため、qsubコマンドなどでジョブ実行も可能
    - Claude Codeへの指示をファイルに書いておき、--dangerously-skip-permissions で対話処理をスキップすれば、コード開発自体をジョブとして実行できる
- 推論にはランダム性があり、seedの固定ができない

Claude CodeのGPUコード開発能力評価

- Claude CodeによってGeoFEM／CubeのGPU対応版を開発
  - GeoFEM／Cube
    - MPI+OpenMPで並列化されたFortranベースのアプリケーション
    - GPU含め、様々な環境向けに最適化された実績がある（今回はMPI+OpenMPの基本的なコードをベースに実験）
- 以下を評価
  - Claude Codeによって生成されたコードの性能
  - Claude Codeのコード開発自体の時間

GeoFEM／Cube

- 有限体積法，一様場ポアソン方程式ソルバー
  - HPC環境での高速化に関する様々な実績がある
    - [大島他 SWoPP 2014], [中島 HPC-139], [中島 HPC-147], [中島 HPC-157], [星野 HPC-158]
  - 差分格子：データ構造は非構造，7点ステンシル
  - 対称正定な疎行列を係数とする連立一次方程式
  - ICCG法，上下三角成分を別々に格納
- 色付け・リオーダリング
  - CM-RCM + Coalesced/Sequential
- 行列格納手法
  - CRS, Sliced-ELL, Sell-C-σ
- 係数行列生成部とソルバ部が主要



GeoFEM／Cube のファイル構成

- 右の表は、Claude Codeに「ディレクトリ構造をlatex表形式にして」と依頼して出力されたもの
- .f の固定長形式Fortran 90
- implicit real を使用
- OpenMPで並列化
  - 係数行列生成関連ファイル
    - mat_ass_*.f
  - ソルバ関連ファイル
    - solver_*.f
- doc フォルダにwordとpdfでアプリの説明有り

表 2: Directory Structure of `GeoFEM-Cube-Hybrid_CG_3`

Directory	File	Description
./doc/	GeoFEM-Cube-3.docx	Documentation (Word)
	GeoFEM-Cube-3.pdf	Documentation (PDF)
./run/	go.sh	Execution script
	mesh.inp	Mesh input file
	test.lst	Test list file
./src/	Makefile	Build configuration
	hpcmw_all.f	HPC middleware main module
	hpcmw_fem_cntl.f	FEM control module
	hpcmw_fem_mesh.f	FEM mesh module
	hpcmw_fem_util.f	FEM utility module
	hpcmw_finalize.f	Finalization module
	hpcmw_init.f	Initialization module
	hpcmw_solver_cntl.f	Solver control module
	hpcmw_solver_matrix.f	Solver matrix module
	hpcmw_util.f	Utility module
	input_cntl.f	Input control module
	input_grid.f	Grid input module
	mat_ass_bc.f	Matrix assembly BC module
	mat_ass_init.f	Matrix assembly init module
	mat_ass_main.f	Matrix assembly main module
	mat_con0.f	Matrix construction module 0
	mat_con1.f	Matrix construction module 1
	mat_trans.f	Matrix transformation module
	solver33.f	3x3 solver module
	solver_CG_3_SMP_novec.f	CG solver (SMP, no vectorization)
	solver_SR_3.f	SR solver module
	test1.f	Test program
	util.f	Utility functions

係数行列生成部

- ループ構造は非常に複雑
- 並列化して動かすだけなら難しくない
- GPUで効率良く並列化するのは単純ではない

並列化可能ループ→

→

GPUで効率良く並列化するなら、jeのループを ieのループ直後に持ってきて、ie, je ループを一重化して並列化するのが普通

```
do icol= 1, ELMCOLORtot
  !$omp parallel do private (...)
  do icel0= ECidx(icol-1)+1, ECidx(icol)
    ! 中略
    do ie= 1, 8
      ip = nodLOCAL(ie)
      if (ip.le.N) then
        do je= 1, 8
          jp = nodLOCAL(je)
          ! 中略
          kk= 0
          iiS= indexU(ip-1) + 1
          iiE= indexU(ip  )
          do k= iiS, iiE
            if ( itemU(k).eq.jp ) then
              kk  = k
              IDlu= 1
              exit
            endif
          enddo
          if (kk.eq.0) then
            iiS= indexL(ip-1) + 1
            iiE= indexL(ip  )
            do k= iiS, iiE
              if ( itemL(k).eq.jp) then
                kk= k
                IDlu= -1
              endif
            enddo
          endif
          ! 中略
          do kpn= 1, 2
            do jpn= 1, 2
              do ipn= 1, 2
                ! a11, a12, ... a33 を計算
              enddo
            enddo
          enddo
          if (IDlu.eq.1) then
            AU(...)= a11, a12,..., a33 を代入
          endif
          if (IDlu.eq.-1) then
            AL(...)= a11, a12,..., a33 を代入
          endif
          if (IDlu.eq.0) then
            D(...)= a11, a12,..., a33 を代入
          endif
        enddo
      endif
    enddo
  enddo
enddo
```

ソルバ部

- ICCG法
  - 共役勾配法(Conjugate Gradient, CG)
  - 前処理が不完全コレスキー分解(Incomplete Cholesky Factorization, IC)
- 処理内容は一般的な線形代数演算の組み合わせ
  - 疎行列ベクトル積、内積、AXPYなど
  - 並列化も簡単で、自明並列可能ループとリダクションループの組み合わせ

```
Compute r(0)= b-[A]x(0)
for i= 1, 2, …
    solve [M]z(i-1)= r(i-1)
    ρi-1= r(i-1) z(i-1)
    if i=1
      p(1)= z(0)
    else
      βi-1= ρi-1/ρi-2
      p(i)= z(i-1) + βi-1 p(i-1)
    endif
    q(i)= [A]p(i)
    αi  = ρi-1/p(i)q(i)
    x(i)= x(i-1) + αip(i)
    r(i)= r(i-1) - αiq(i)
    check convergence |r|
end
```

GeoFEM／Cube の出力例

- 以下の出力例はdocフォルダのwordファイルにあるもの
  - 実際に出力されるのは黒字のみ
  - Claude Codeにプロンプトで教えない限りは、wordファイルを読み解くか、コードから出力内容を判断する必要がある

```
        128        128        128          npx, npy, npz
          2          2          1          ndx, ndy, ndz
         12                                PEsmpTOT
                                           （各MPIプロセスのOpenMPスレッド数）
### NORMAL
color number:         0
### MATRIX assembly     3.046744E-01       MATassTime（行列生成に要する時間）

 1295     9.866630E-09                     反復回数（ITER），残差（RESID）

### min/max/ave      3.005061E+01      3.005061E+01      3.005061E+01
                                           CG法計算時間
                                           （最小：Tmin，最大：Tmax，平均：Tave）
                                           5回以上計測したTmaxの最小値で評価
  524288    -3.810000E+01    -3.810000E+01      1.270000E+02
                                           参照点の変位（Ux,Uy,Uz）
                                           計算が正しく終了していることの指針
 * normal termination
```

実験内容

- 実験方針
  - 同じ入力ソースコード、生成指示で、各10回生成
  - 出力されたプログラムは各5回計測して中央値を取る

- 実験対象
  - 入力ソースコード差分
    - **.f**：拡張子.f の固定長形式Fortran90, implicit real（オリジナル）
    - **.f90**：拡張子.f90 の自由形式Fortran90, implicit none
    - **.f90_woOMP**：**.f90** からOpenMPの指示文を削除したもの
  - 出力プログラミング言語指定
    - **omp**：**.f90** をベースに、OpenMP target を使ってGPU化
    - **acc**：**.f90** をベースに、OpenACC を使ってGPU化
    - **cuda**：**.f90** をベースに、CUDA Fortran を使ってGPU化
  - さらに高速化
    - **fast1**：**acc** をベースに、さらに速くせよと指示
    - **fast2**：**fast1** をベースに、さらに速くせよと指示
    - **fast3**：**fast2** をベースに、さらに速くせよと指示

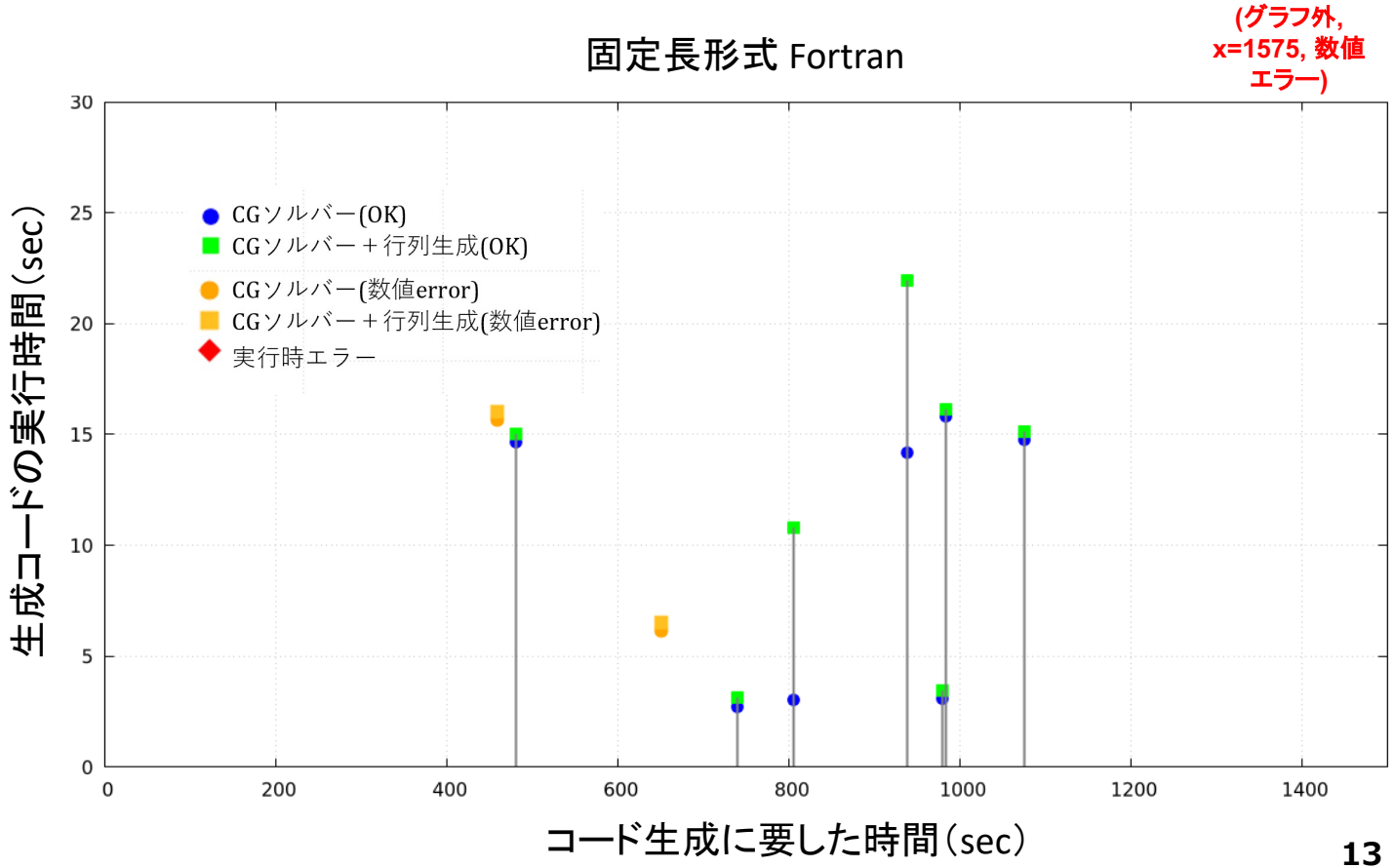
評価環境

- JCAHPC（東大・筑波大）の Miyabi
  - GH200
  - CPU-GPU 間がNVLink C2C （片方向450 GB/s）でCache-coherent に接続

- ソフトウェア環境
  - GPU向けコンパイラ：nvfortran 24.9（デフォルト）
    - Claude Codeが別のコンパイラを使うことはあり得たが結果として全てこれを使っていた
  - Claude Code ver.1.0.83 ～ 1.0.90
    - 頻繁に更新されるため、固定できなかった
    - モデルは全てClaude Opus 4.1

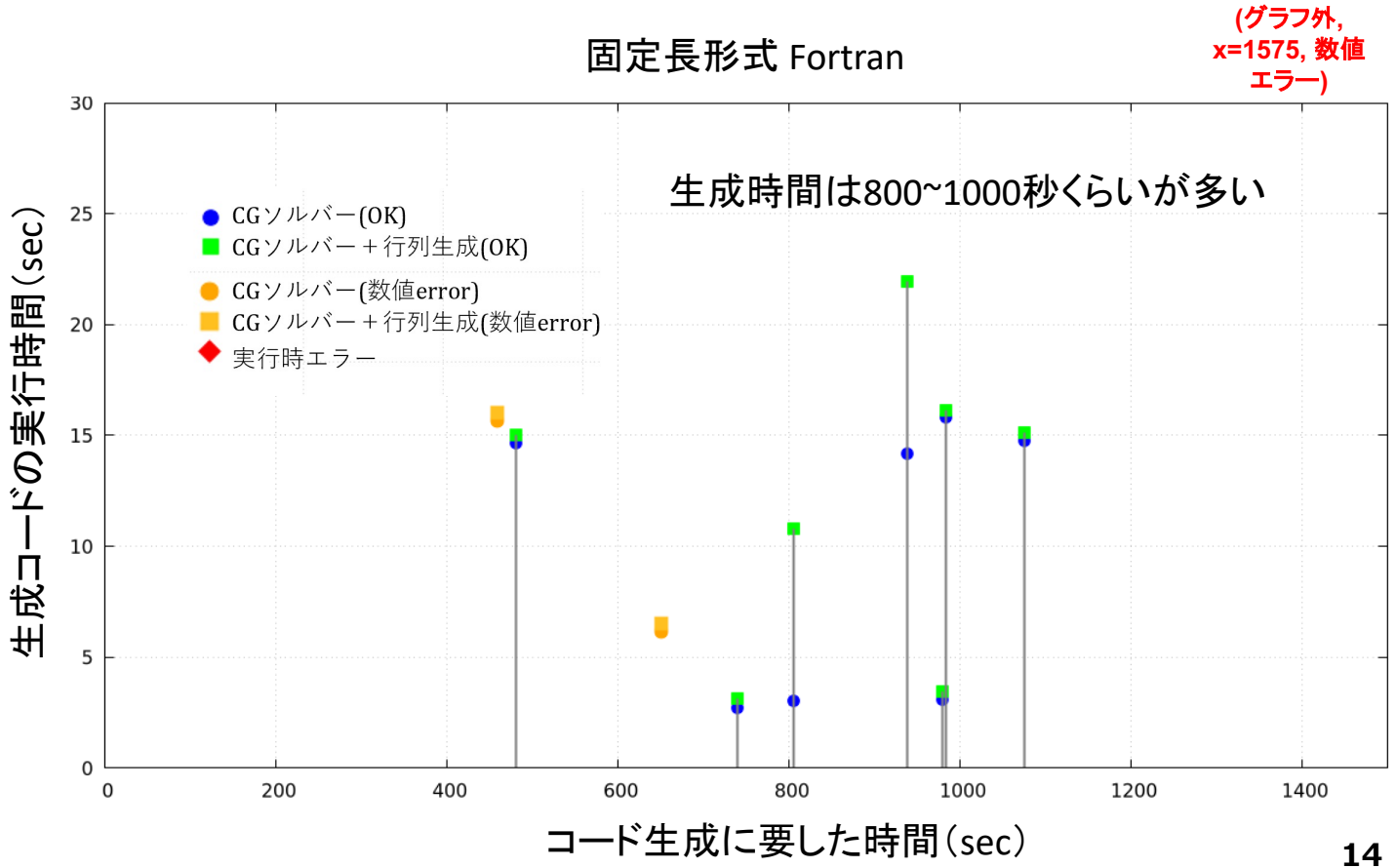
入力ソースコード差分（.f）

- 青点がソルバのみの時間
- 緑展がソルバ+行列生成
- 黄色や赤は正しく実行できていない



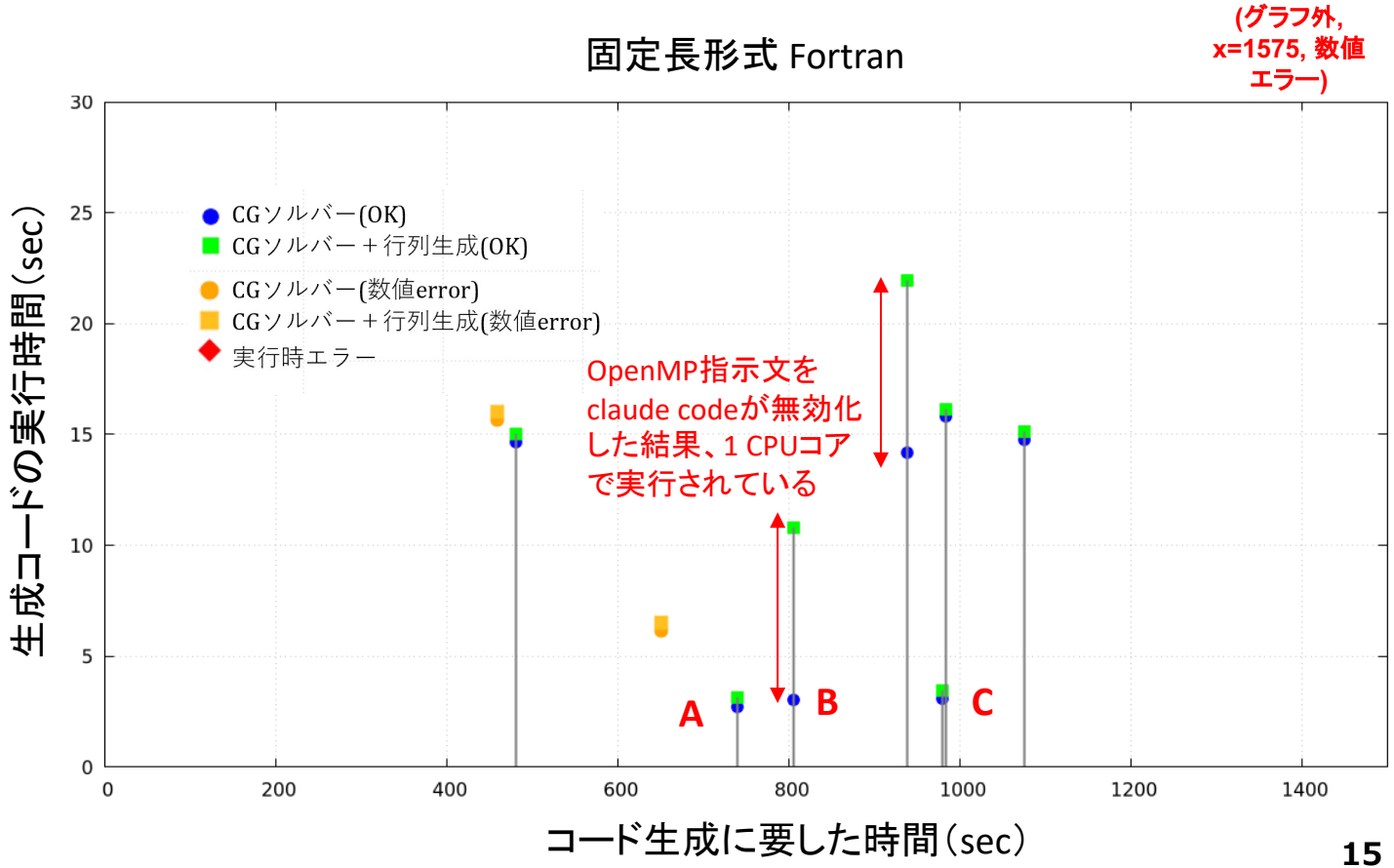
入力ソースコード差分（.f）

- 人間のコーディングと同じで、固定長形式Fortranの文字数制限や、implicit記法によってバグが生じ、生成時間が長くなる
- グラフ外の点は、係数行列生成に挑戦し、挙句失敗している



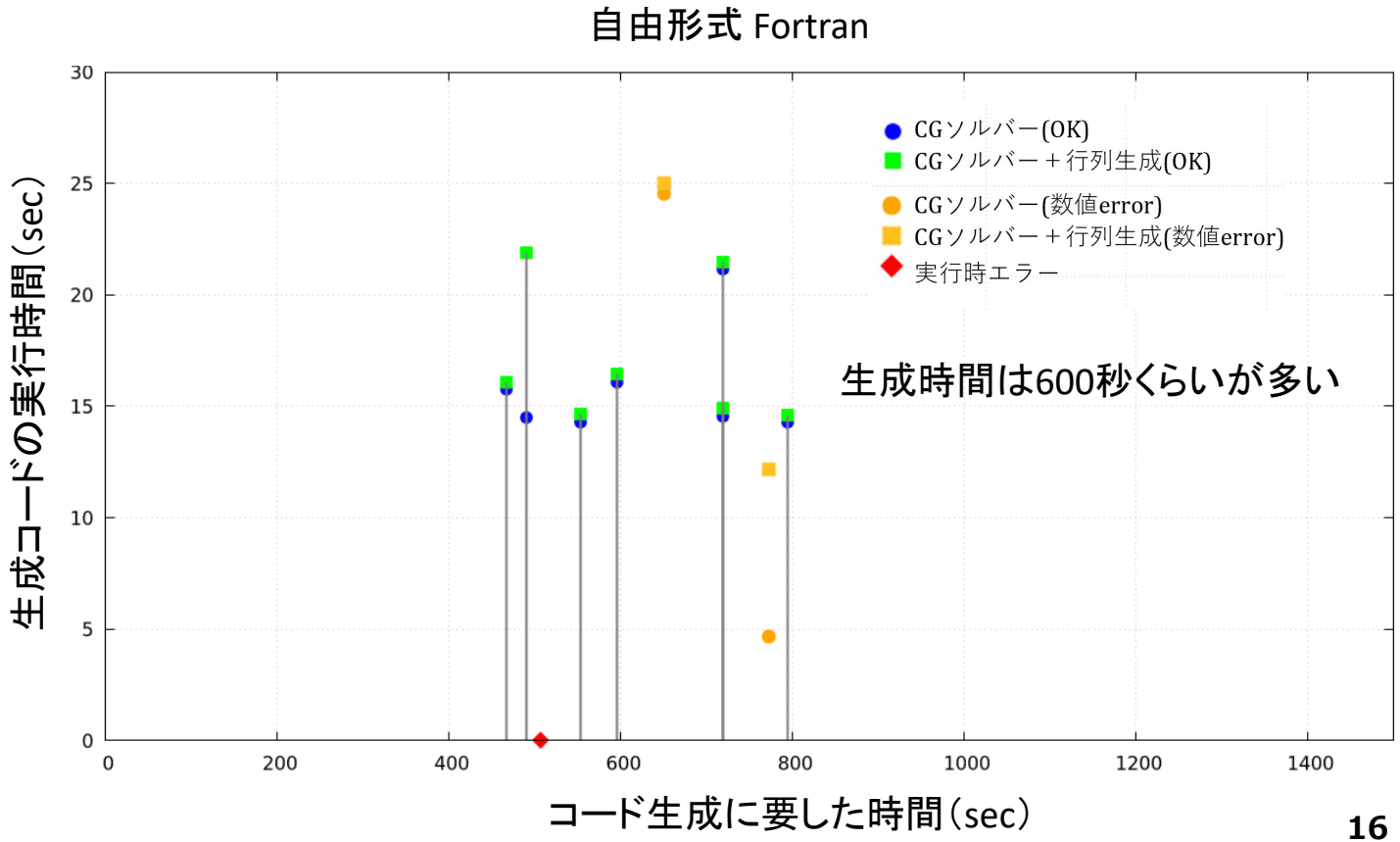
入力ソースコード差分（.f）

- 係数行列生成部のGPU化には一度も成功しなかった
- **A, C**はOpenACCのkernels指示文を使っていて、ソルバ部分に関しては適切な性能になっている
- **B**はOpenACCのparallel指示文を使っていて、かつ適切に指示子を指定している
- それ以外はparallel指示文で、適切に指示節が設定されていない



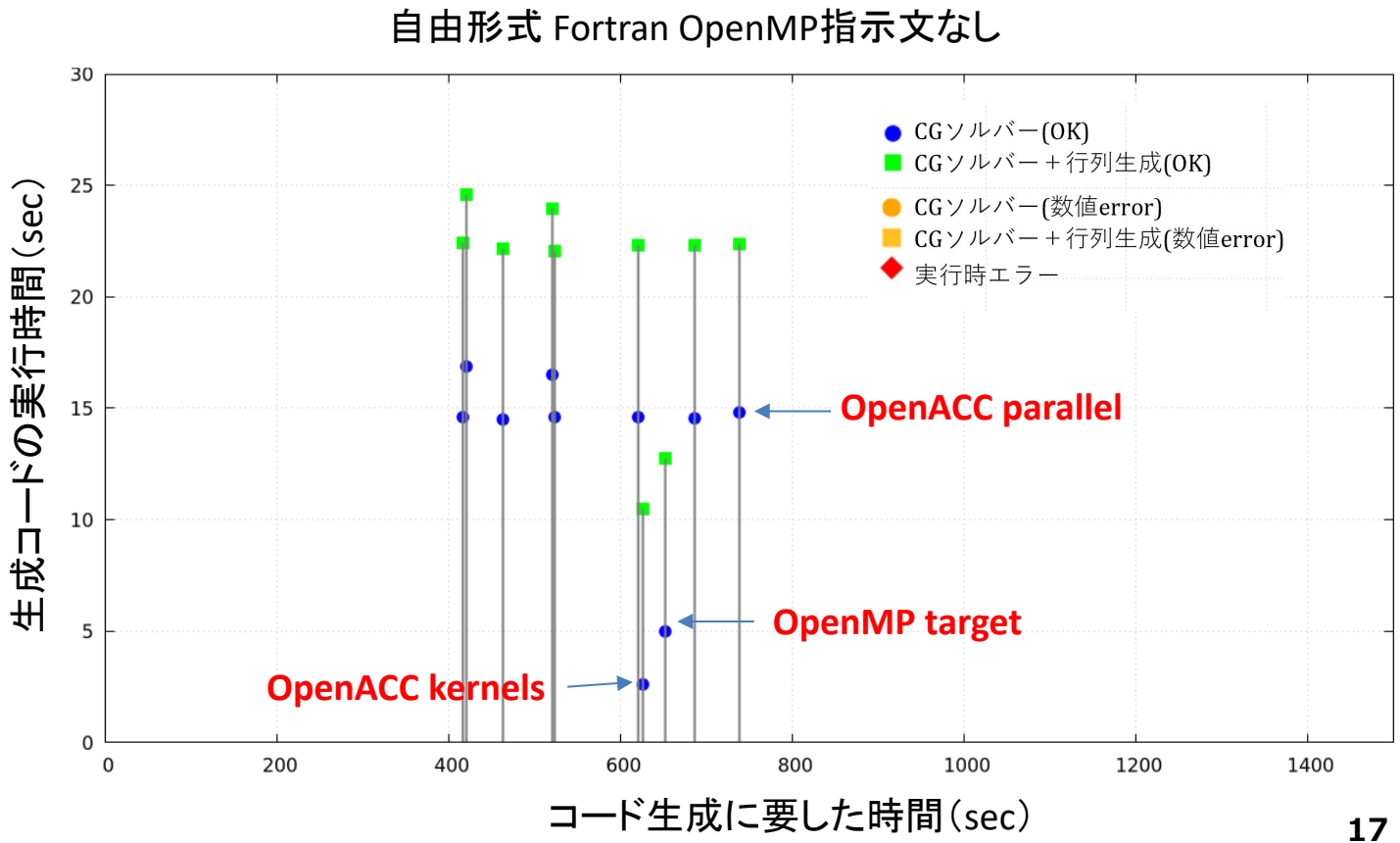
出力プログラミング言語指定（OpenACC）

- 生成時間は明らかい短くなっている
- **.f**では一部適切なコードを吐いているが、**.f90**では吐かれていない（理由は不明）



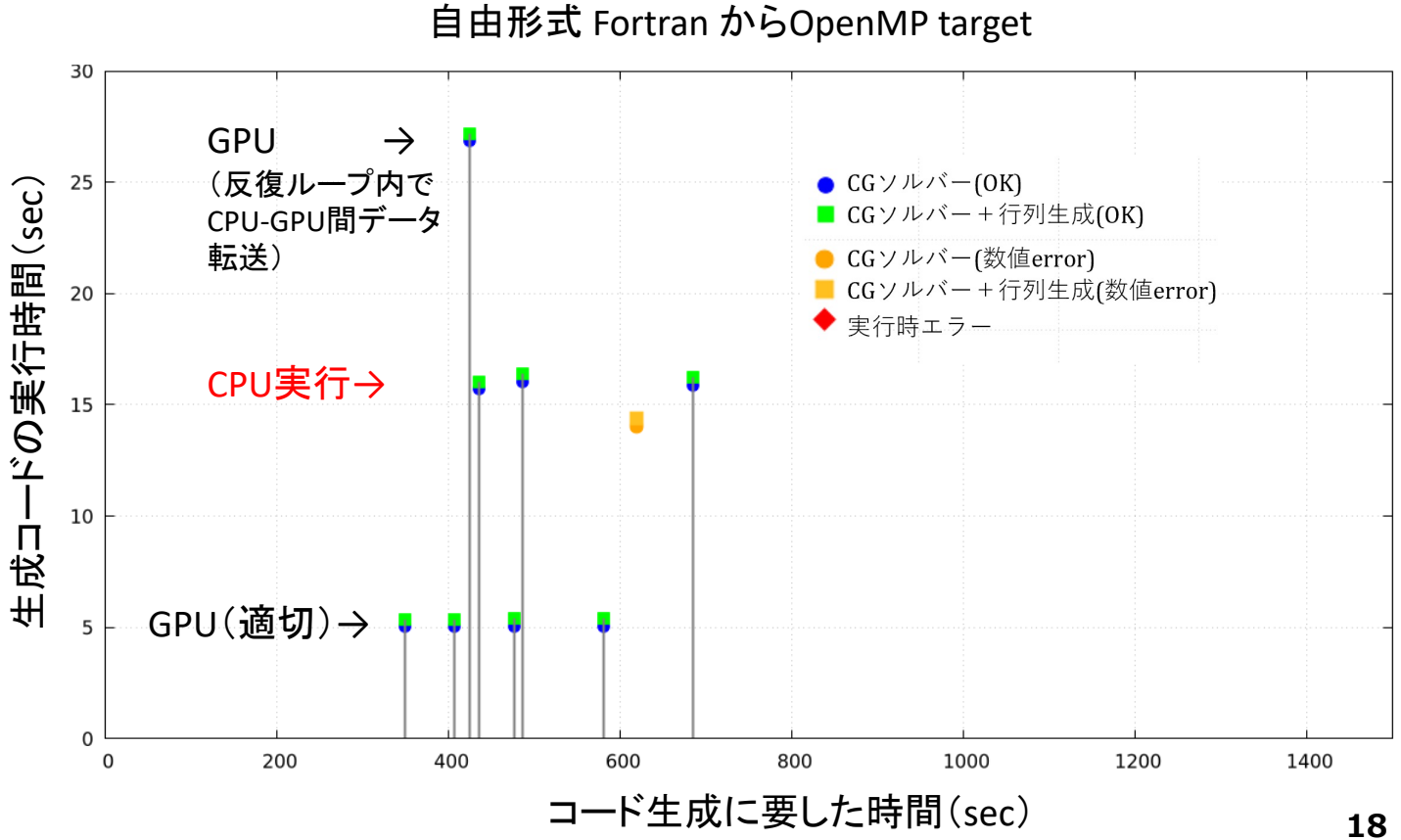
出力プログラミング言語指定（CUDA Fortran）

- OpenMP指示文がなくなった結果、行列生成部のGPU化を全く試さなくなったため、結果として生成成功率が増加、生成時間が短縮している
- 行列生成部はCPU 1コアで実行されている



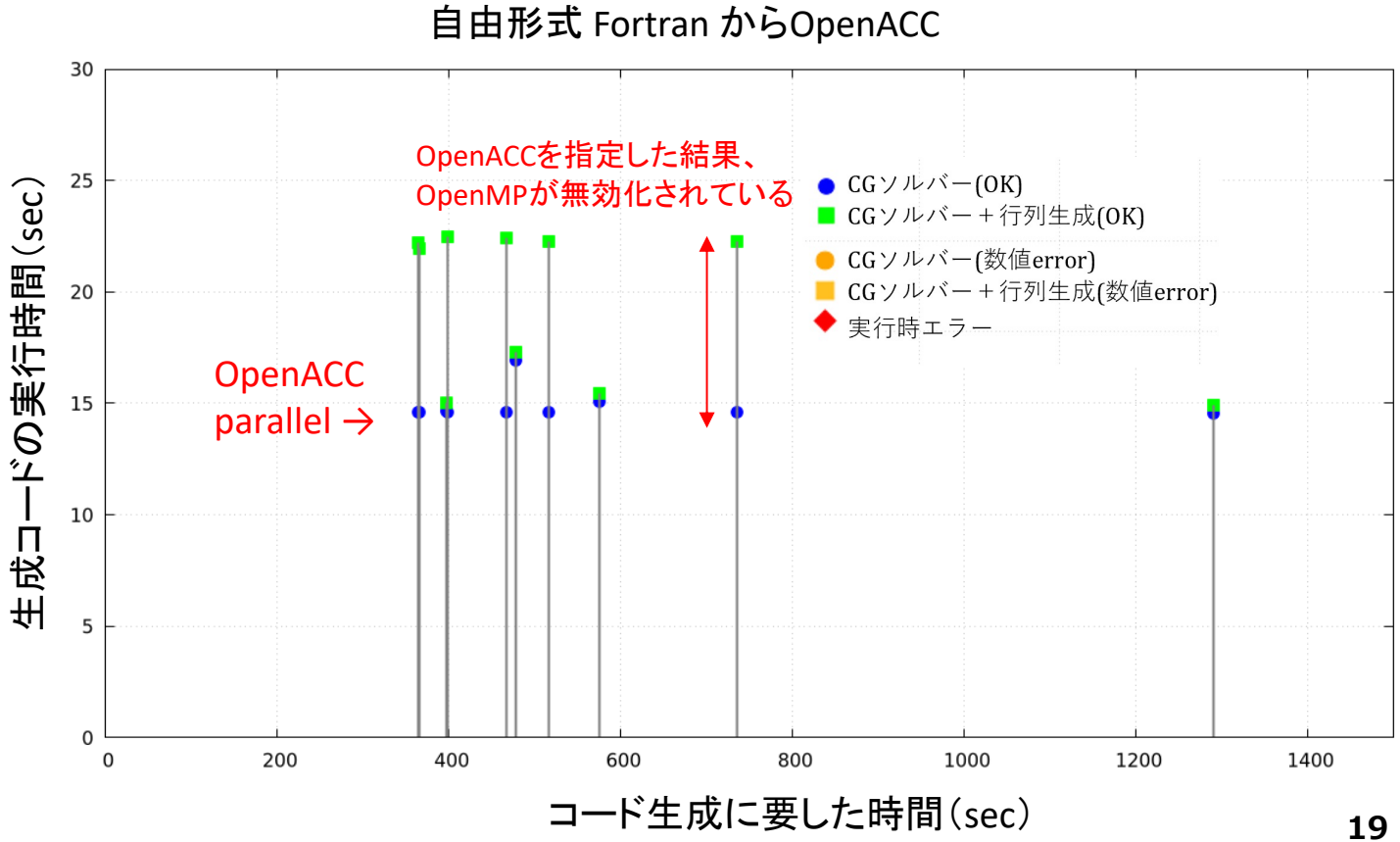
出力プログラミング言語指定（OpenMP target）

- GPU実行に成功しているのは実は5ケースのみ
- 生成スピードは概ね早い



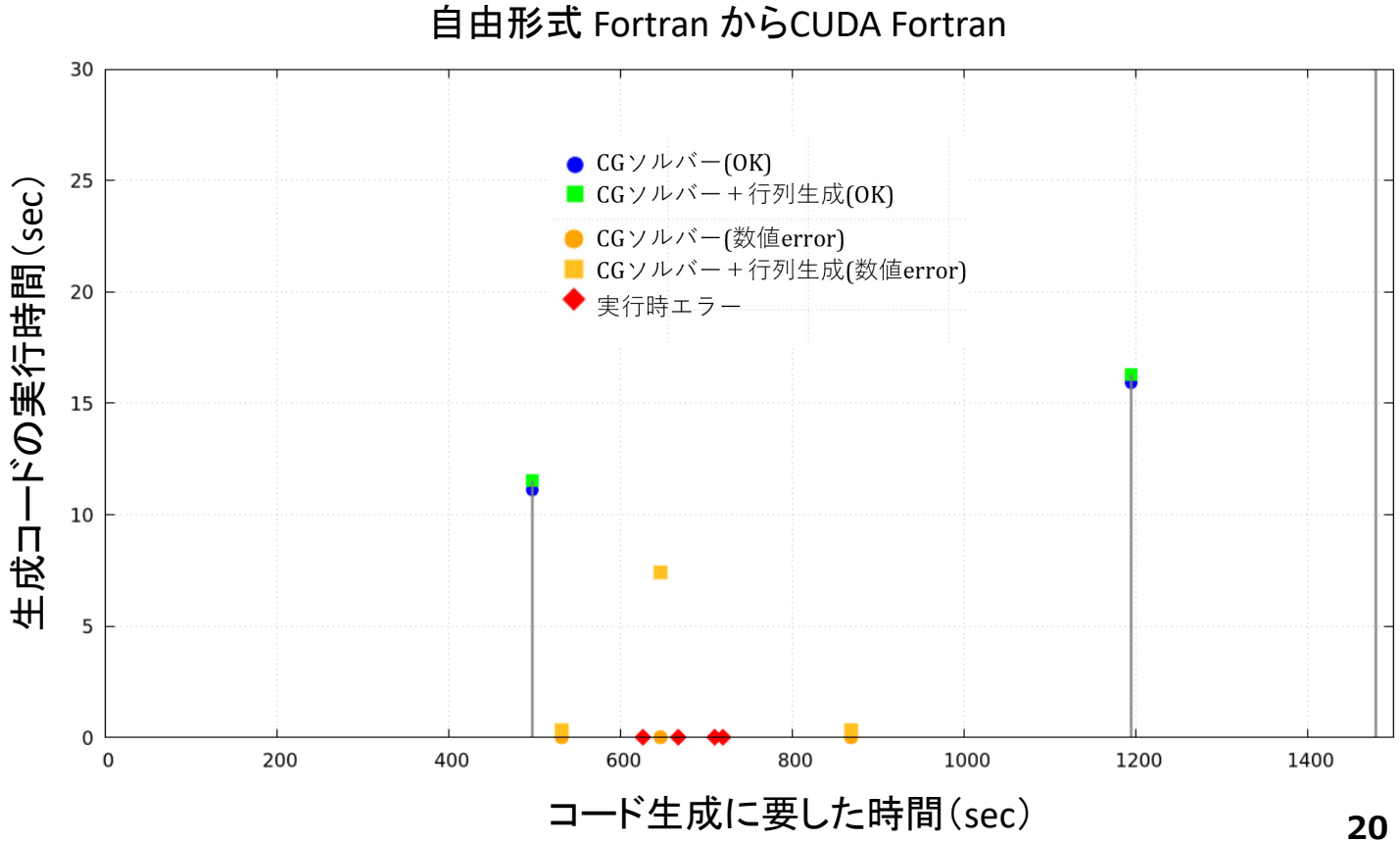
出力プログラミング言語指定（OpenACC）

- 実行可能なGPUコード生成の可能性は高い
- 全てOpenACC parallel 指示文を使用していて、指示節の指定が適切でないため、性能がよくない（なぜかは不明）
- OpenACCを指定した結果、MakefileのOpenMPフラグが消されてしまい、行列生成部の多くが1 CPUで実行されている



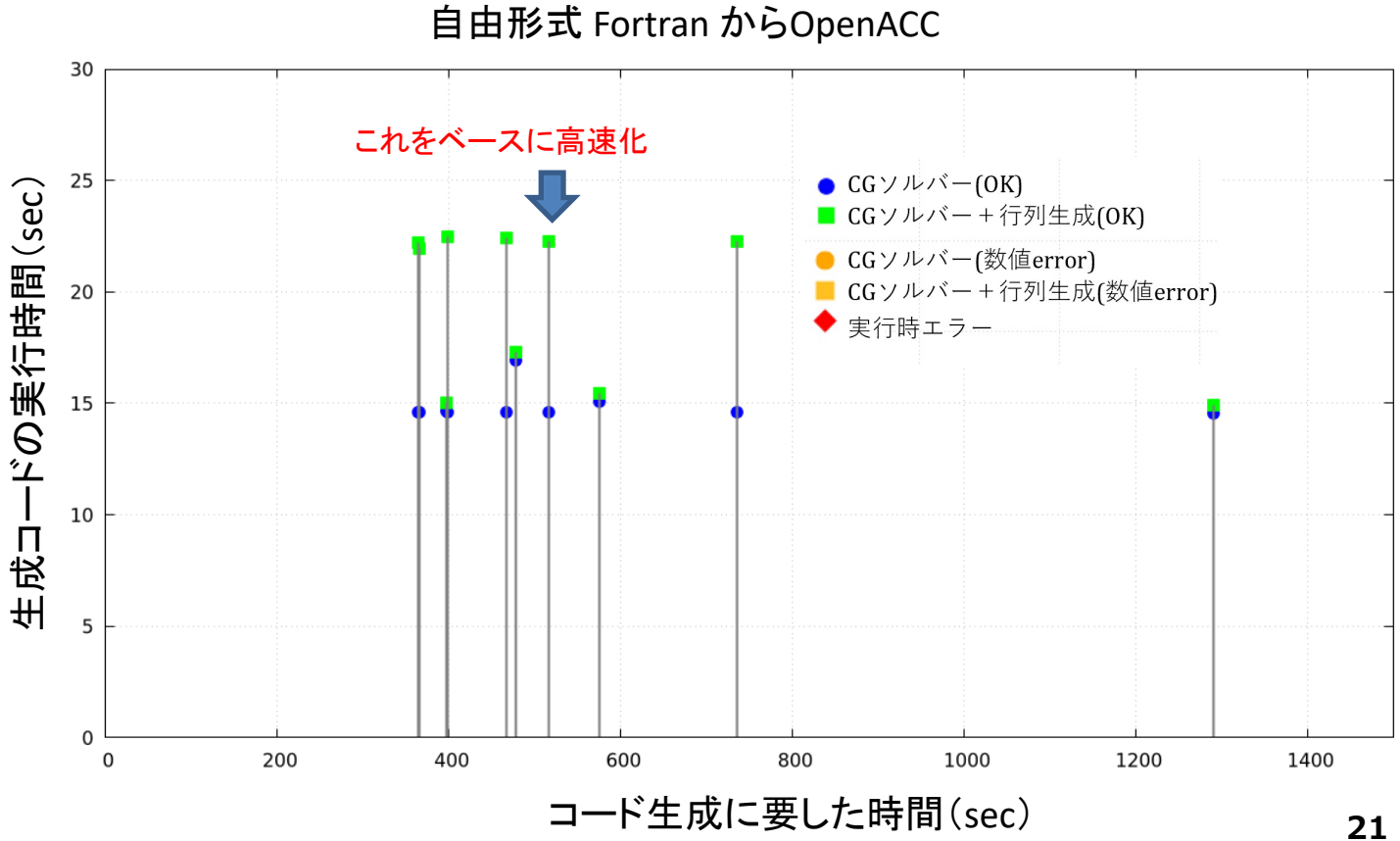
出力プログラミング言語指定（CUDA Fortran）

- 生成成功率は非常に低い
- 成功したケースでも速くない
- 出力をCUDA Cにすればマシになる可能性もあるが、今後の課題



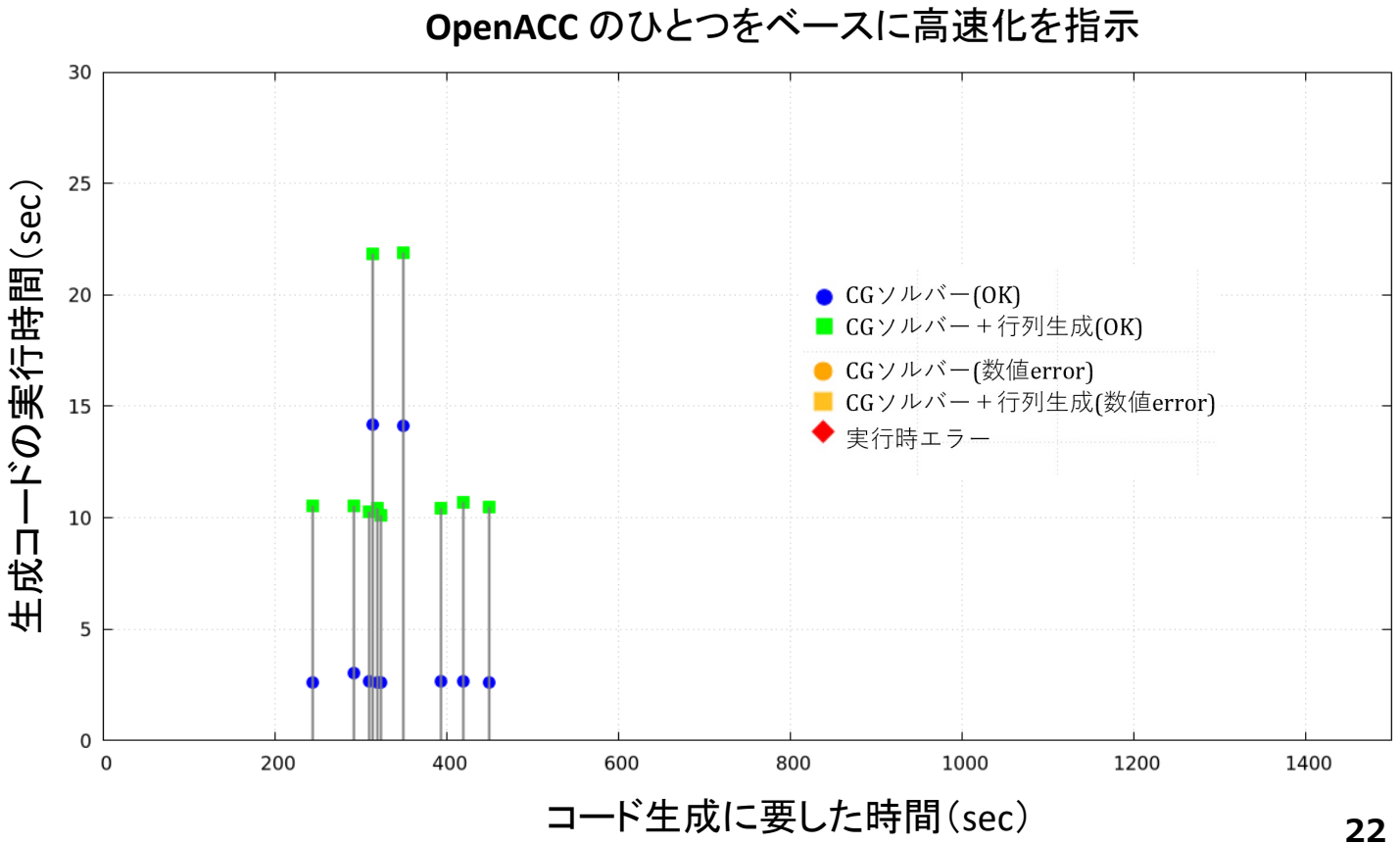
さらに高速化（Level 0）

- OpenACC parallel 指示文を使っていて、指示子が適切でなく遅いコードをベースにさらに高速化



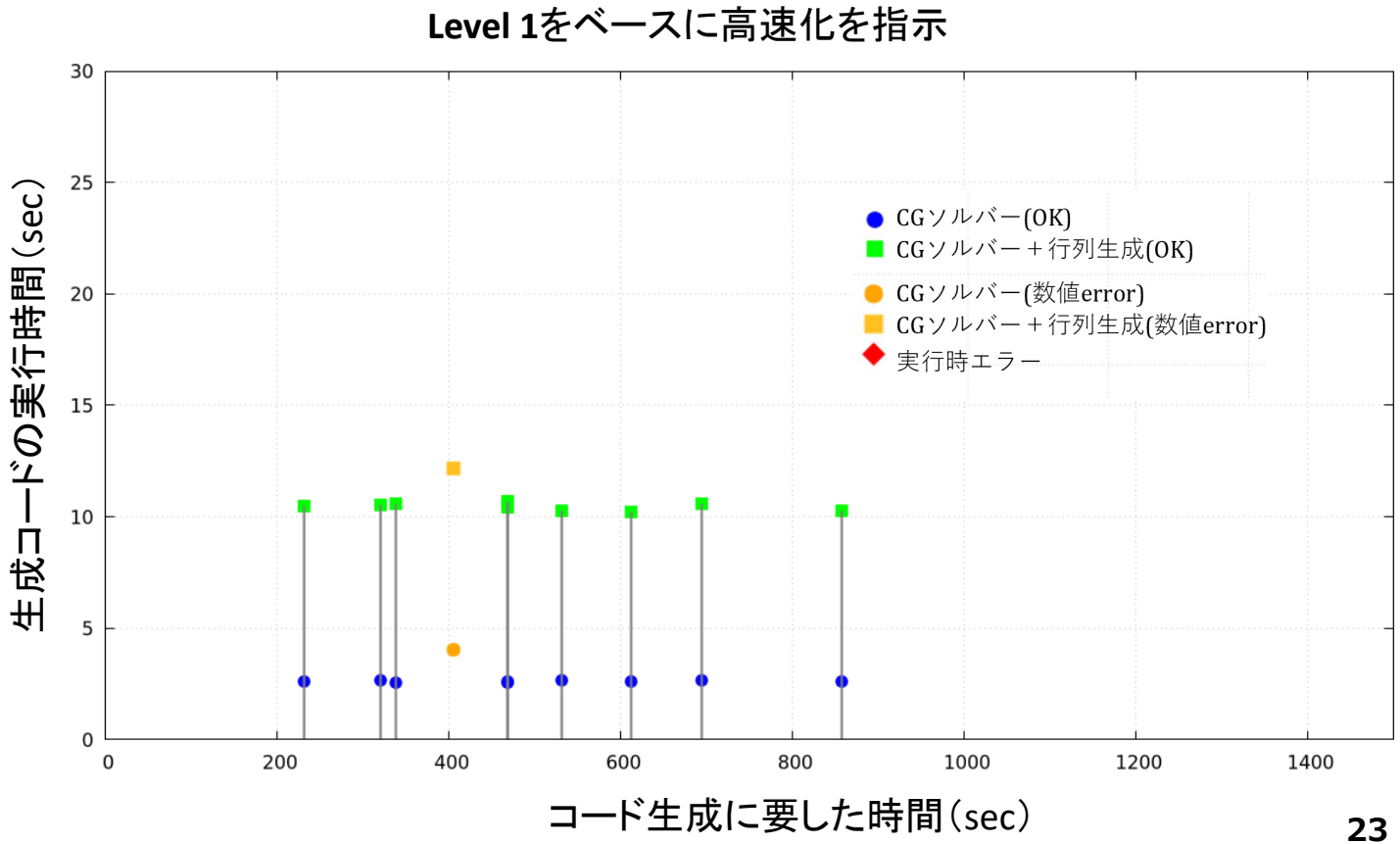
さらに高速化（Level 1）

- 多くのケースで、parallel指示文の指示子を適切に指定して、高速化を達成



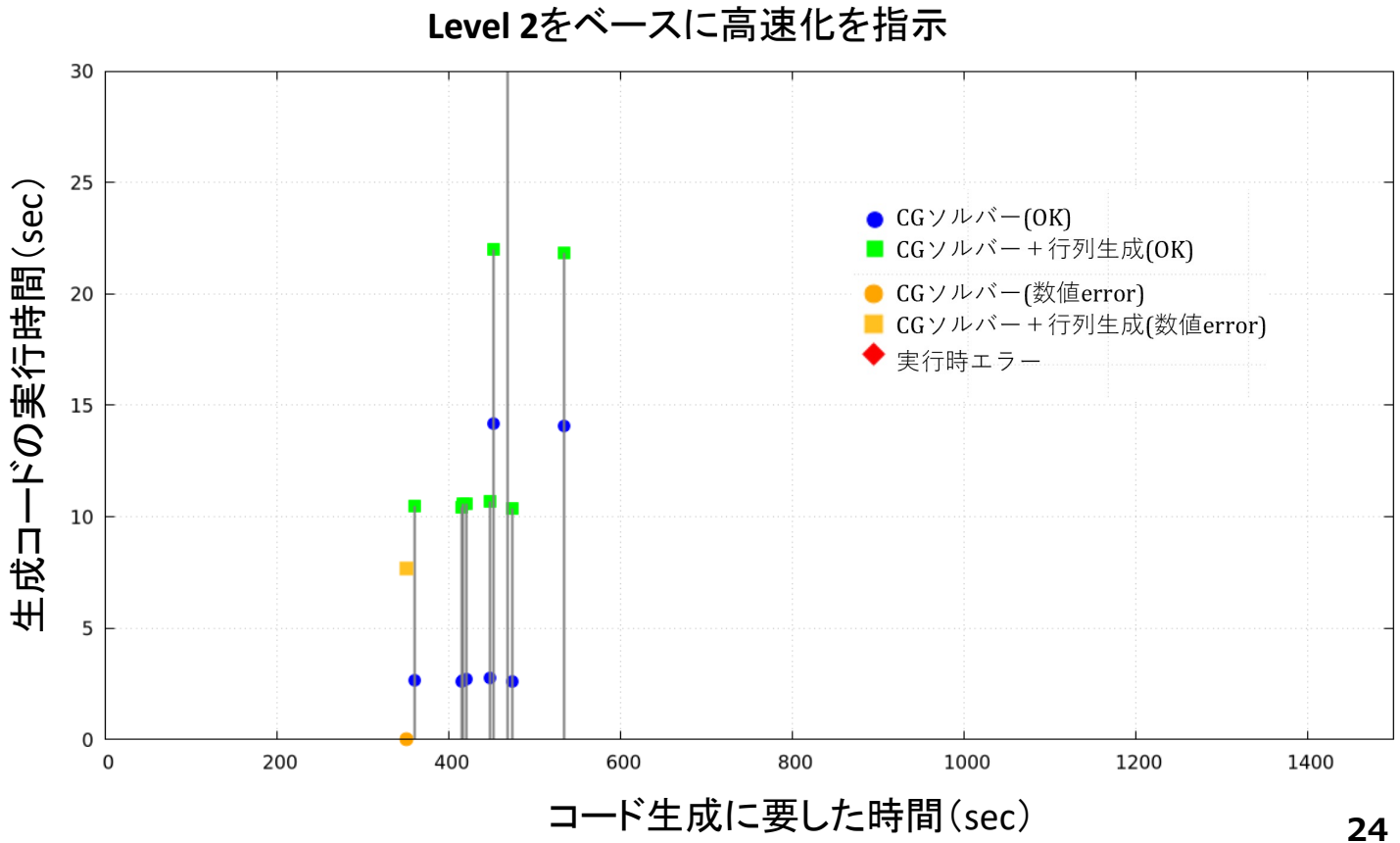
さらに高速化（Level 2）

- 多くのケースで
async節が付与され、
少し高速化（これは
良くやられる手段）



さらに高速化（Level 3）

- 多くのケースで失敗し、遅くなっている
- 明らかに行列生成部の実行時間が支配的であるにもかかわらず、高速化は実施されていない



考察

- CGソルバの部分に関しては、OpenMP指示文があろうとなかろうと、概ね生成に成功
  - OpenACCでの成功率が高い
    - FortranコードのGPU化での既存コードが多いためと考えられる
    - 「さらに高速化」を指示すれば、概ね適切なコードが得られる

  → 並列版CGソルバを既に知っているから？
- 係数行列生成部分に関しては、OpenMP指示文があるにもかかわらず、並列化が実施されない（実施されても成功しない）

  → 並列版係数行列生成を知らないから？

人間だったら、
　　このループは並列化できるな　→　指示文を適用
コード生成AIは、
　　これはCG　→　対応する並列CGはこれ
　　…という違いがあるのでは？

まとめ

- MPI+OpenMPで並列化されたFortranベースのGeoFEM／Cubeをベースに、コード生成AIによるアプリケーションのGPU対応版開発能力の評価を実施した
- 同じ計算であっても、入力するソースコードの差分（Fortranの固定長形式、自由形式など）により開発時間に大きな差が出た
- CG部分に関しては、良好な生成結果であったが、独自コード色の強い係数行列生成部は全くうまくいかなかった
- 今後の課題
  - 独自コード色の強い部分について、GPUコード生成の成功率を上げる手法の開発
    - 生成AIが扱えるくらいタスクを小さくすれば良いのではないか？
    - このループとこのループを並列化せよ、とコード中に指示を書くとか