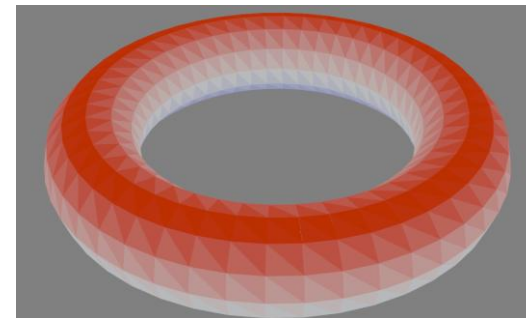


GPUを用いた境界要素解析の 高速化に関する研究

京都大学大学院 情報学研究科 修士一回

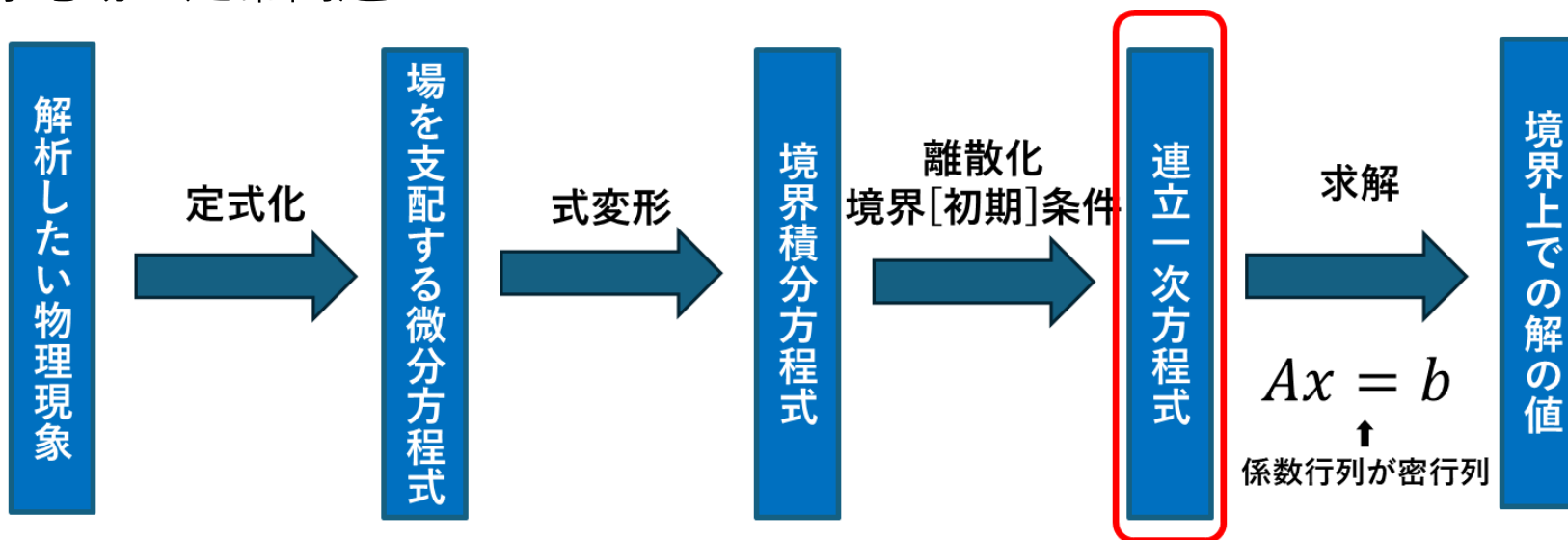
西田怜史

研究背景



境界要素法 (BEM):

- 偏微分方程式の求解手法の一つで、多くの物理シミュレーションで活用されている
- モデルの境界部分だけを離散化するため、少ない未知変数で高精度な解析が可能
- 例: 静電場の定常問題



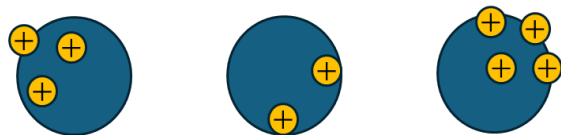
課題

密行列の連立一次方程式の計算量がボトルネック

複数モデルによる境界要素解析

実アプリケーションでは、状況を変えて複数回 BEM 実行することが多い

例: 静電場の定常問題を3パターン解く (モデルは変えない)



→ 係数行列が同じ連立一次方程式を何度も解く

単一モデルの連立一次方程式:

$$Ax_1 = b_1, \quad A \in \mathbb{R}^{n \times n}, \quad b_1, x_1 \in \mathbb{R}^n$$

$$Ax_2 = b_2, \quad A \in \mathbb{R}^{n \times n}, \quad b_2, x_2 \in \mathbb{R}^n$$

$$Ax_3 = b_3, \quad A \in \mathbb{R}^{n \times n}, \quad b_3, x_3 \in \mathbb{R}^n$$

⋮

係数行列は全部同じ密行列
BiCGStab 法で解く

→ **連立一次方程式を一度で解けば性能改善が期待できる**

複数の右辺ベクトルを同時に扱う場合:

GPU の並列性を効率的に活用出来る

$$AX = B, \quad A \in \mathbb{R}^{n \times n}, \quad B \in \mathbb{R}^{n \times m}, \quad X \in \mathbb{R}^{n \times m}$$

複数の右辺ベクトルを束ねて
行列形式にする

$$B = \begin{bmatrix} b_1 & b_2 & \cdots & b_m \end{bmatrix}, \quad X = \begin{bmatrix} x_1 & x_2 & \cdots & x_m \end{bmatrix}$$

BiCGStab 法の多右辺対応

$Ax = b, \quad A \in \mathbb{R}^{n \times n}, b, x \in \mathbb{R}^n$: 通常の解法

初期近似解を x_0 , 初期残差 $b - Ax_0$ を r_0 .

$(r_0^*, r_0) \neq 0$ となるように r_0^* を決定する.

$p_0 = r_0^*$ とする.

while 収束判定条件を満たさない do

for $k = 0, 1, \dots$ do

$$\alpha_k = \frac{(r_0^*, r_k)}{(r_0^*, Ap_k)} \quad \leftarrow \text{行列・ベクトル積}$$

$$t_k = r_k - \alpha_k Ap_k$$

$$\xi_k = \frac{(At_k, t_k)}{(At_k, At_k)}$$

$$x_{k+1} = x_k + \alpha_k p_k + \xi_k t_k$$

$$r_{k+1} = t_k - \xi_k At_k$$

$$\beta_k = \frac{\alpha_k}{\xi_k} \cdot \frac{(r_0^*, r_{k+1})}{(r_0^*, r_k)}$$

$$p_{k+1} = r_{k+1} + \beta_k (p_k - \xi_k Ap_k)$$

end for

end while

$AX = B, \quad A \in \mathbb{R}^{n \times n}, B \in \mathbb{R}^{n \times m}, X \in \mathbb{R}^{n \times m}$: 多右辺版

初期近似解を X_0 , 初期残差 $B - AX_0$ を R_0 .

$(R_0^*, R_0) \neq 0$ となるように r_0^* を決定する.

$P_0 = R_0^*$ とする.

while 収束判定条件を満たさない do

for $k = 0, 1, \dots$ do

$$\alpha_k = \frac{(R_0^*, R_k)}{(R_0^*, AP_k)} \quad \leftarrow \text{行列・行列積が登場}$$

$$T_k = R_k - \alpha_k AP_k$$

$$\xi_k = \frac{(AT_k, T_k)}{(AT_k, AT_k)}$$

$$X_{k+1} = X_k + \alpha_k P_k + \xi_k T_k$$

$$R_{k+1} = T_k - \xi_k AT_k$$

$$\beta_k = \frac{\alpha_k}{\xi_k} \cdot \frac{(R_0^*, R_{k+1})}{(R_0^*, R_k)}$$

$$P_{k+1} = R_{k+1} + \beta_k (P_k - \xi_k AP_k)$$

end for

end while

多右辺化
(右辺 b が複数ケース
あるとき同時に解く)

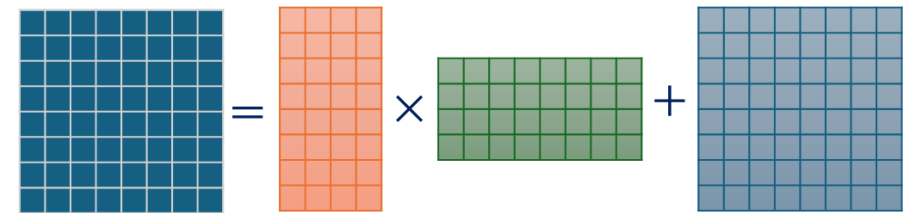
Tensor コアを活用

Tensor コア:

- NVIDIA GPU に搭載された行列積演算ユニット
行列・行列積を高速に計算できる
- CUDA では WMMA API 経由で利用可能
- 対応データ型 (NVIDIA A100): **INT8, FP16, BF16, FP32, FP64**
- 固定サイズの行列タイルが演算単位

Tensor コアで FP 64 行列・行列積を計算する場合:

- 入力行列タイル: 8×4 , 4×8
- 出力行列タイル: 8×8
- 行列をタイル単位で分割して解く



$$D = AB + C$$

→ 多右辺化した BiCGStab 法の行列・行列積で Tensor コアを使用して高速化

WMMA API の利用

```
__global__ void wmma_matvec(  
    int batch, /* 8-aligned */  
    int dim, /* 8-aligned */  
    const double* __restrict__ A /* [dim * dim] */,  
    const double* __restrict__ P /* [dim * batch] */,  
    double* __restrict__ Q /* out [dim * batch] */  
) {  
    const int coarse_row = blockIdx.x;  
    const int coarse_col = blockIdx.y;  
  
    wmma::fragment<wmma::accumulator, WMMA_M, WMMA_N, WMMA_K, double> c_frag;  
    wmma::fill_fragment(&f: c_frag, in: 0.0);  
  
    const int num_k_tiles = dim / WMMA_K;  
    for (int k_tile = 0; k_tile < num_k_tiles; ++k_tile) {  
        wmma::fragment<wmma::matrix_a, WMMA_M, WMMA_N, WMMA_K, double, wmma::row_major> a_frag; // 8x4  
        wmma::fragment<wmma::matrix_b, WMMA_M, WMMA_N, WMMA_K, double, wmma::row_major> b_frag; // 4x8  
  
        const double* a_tile_ptr = &A[tcl_at(row: coarse_row * WMMA_M, col: k_tile * WMMA_K, num_cols: dim)];  
        const double* b_tile_ptr = &P[tcl_at(row: k_tile * WMMA_K, col: coarse_col * WMMA_N, num_cols: batch)];  
  
        wmma::load_matrix_sync(&a: a_frag, p: a_tile_ptr, ldm: 8);  
        wmma::load_matrix_sync(&a: b_frag, p: b_tile_ptr, ldm: 8);  
  
        wmma::mma_sync(&d: c_frag, a: a_frag, b: b_frag, c: c_frag);  
    }  
  
    double* c_tile_ptr = &Q[tcl_at(row: coarse_row * WMMA_M, col: coarse_col * WMMA_N, num_cols: batch)];  
    wmma::store_matrix_sync(p: c_tile_ptr, a: c_frag, ldm: 8, layout: wmma::mem_row_major);  
}
```

wmma::fragment に Tensor コアの
タイルをロード or ストア

wmma::mma_sync() で行列演算
を実行

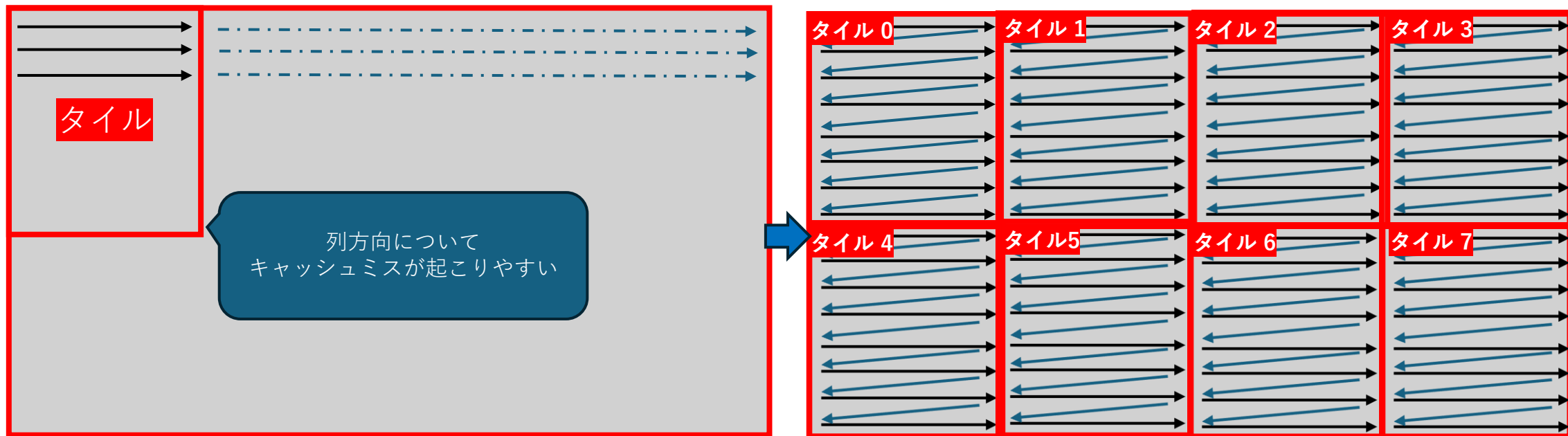
Tensor コア向けメモリレイアウトの導入

行列を標準的な row-major の方式でメモリに格納した場合:

- 列方向のメモリアクセスが遅くなり、Tensor コアを効果的に利用できない

➡ メモリレイアウトを改善する

タイル 8×8 を左から右、上から下の順にメモリ上へ連続して格納する



実験結果

バッチ数 100~1000 について、実行速度を比較

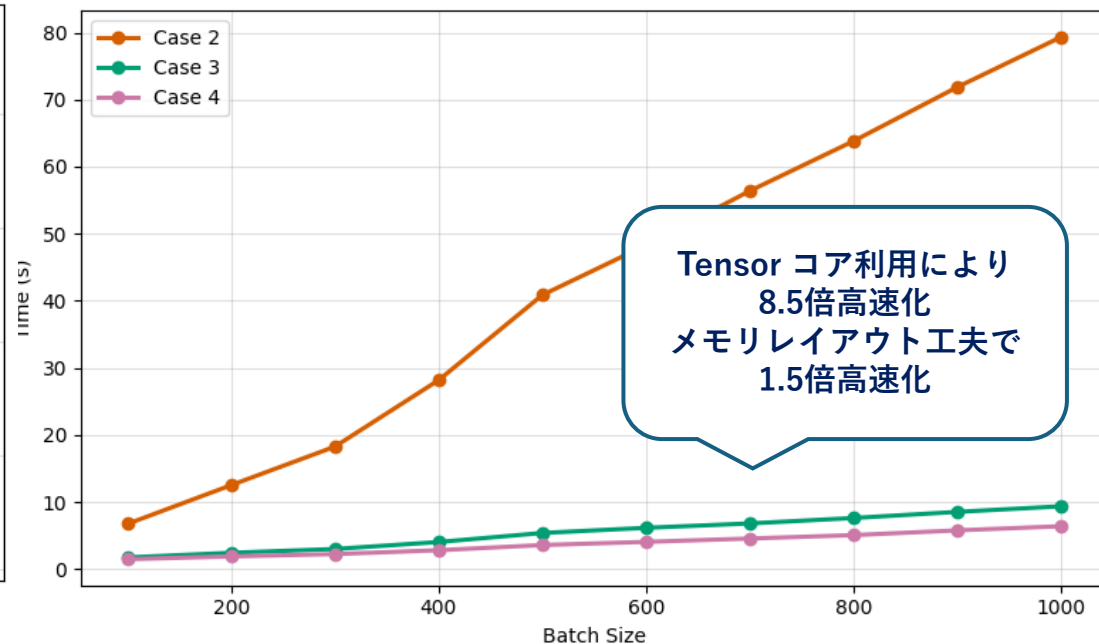
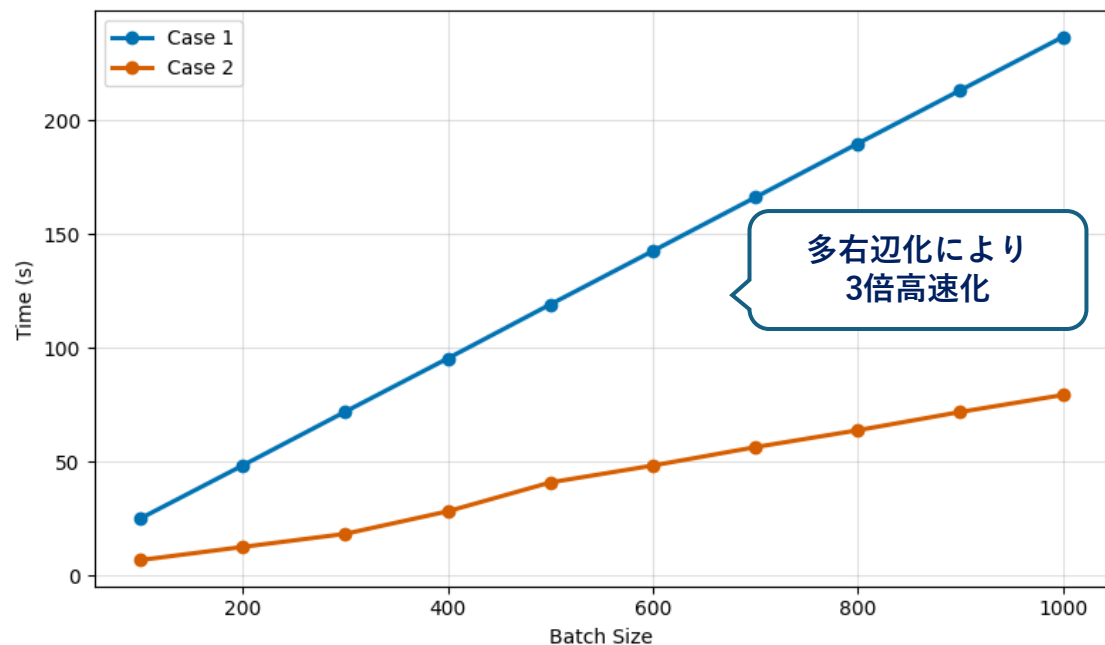
※ バッチ数: BEM で解くモデルの総数

Case 1: 従来法 (単一右辺ベクトル × 複数回実行)

Case 2: 多右辺ベクトル | row-major | Tensor コア未使用

Case 3: 多右辺ベクトル | row-major | Tensor コア使用

Case 4: 多右辺ベクトル | Tensor コア向けレイアウト | Tensor コア使用

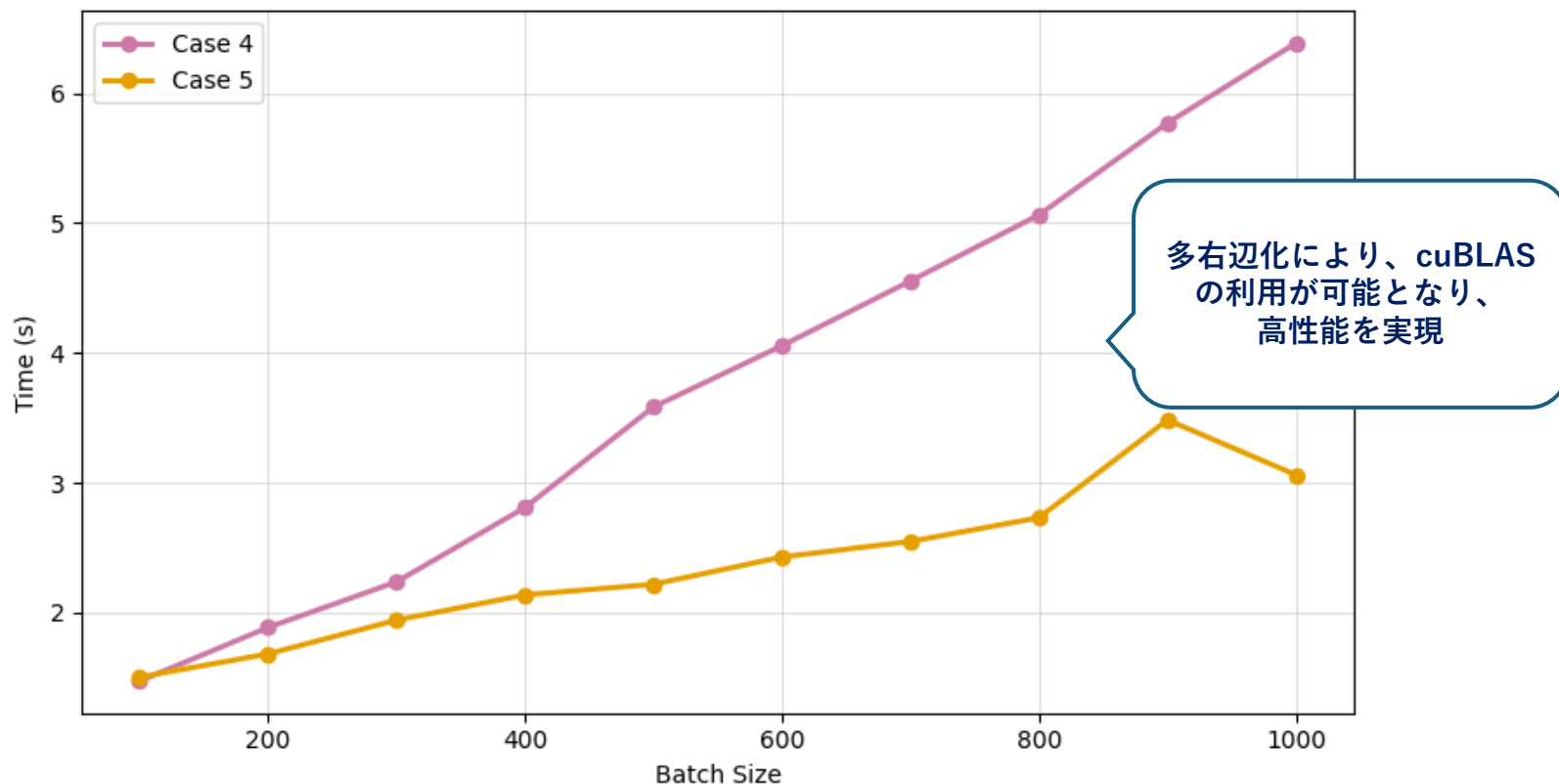


cuBLAS を用いた高速化

cuBLAS:

- NVIDIA GPU 上で行列演算を高速実行する線形代数ライブラリ
- ハードウェアレベルでチューニングされており高パフォーマンス

Case 5: 多右辺ベクトル | row-major | cuBLAS 利用



まとめ

- GPU を用いることによる境界要素解析の高速化について研究を行った。
- 境界要素解析のボトルネックである線形反復ソルバー (BiCGStab 法) の高性能化について検討した。
- GPU を効果的に用いるために、**複数モデルを一括して扱うこと**で多右辺化する。
- 多右辺化により、**行列・行列積演算を主体とする実装を可能とする。**
- **行列・行列積演算をTensorコアで実行することで、高い性能を実現した。**
 - 行列のメモリレイアウトの改善
 - cuBLAS の利用