# 生成 AI の活用による HPC コード GPU 化の展望

棕木 大地（名古屋大学 情報基盤センター）

2026 年 1 月 21 日
「次世代計算基盤を見据えたソフトウェア環境整備とそれを担う人材の育成に関する提言」
についての意見交換会

# 背景

**生成 AI 技術の現状**

- ChatGPT に代表される「博士号レベルの知識」を持った AI が普及
- 長時間自律動作しツールを使いこなすコーディング AI エージェント
  - Claude Code（Anthropic），Codex（OpenAI），Gemini CLI（Google）
- Vibe Coding – AI とのカジュアルな対話を通してコーディングする開発スタイル

**HPC 業界におけるコード生成 AI への期待**

- 研究開発の効率化と人材不足の解消
- HPC の研究・開発に求められる高度な知識・技術の補完
  - 多様かつ特殊なアーキテクチャ（GPU，分散並列，FPGA etc.），言語（C/C++/Fortran etc.）
  - 数値計算特有の問題（最適なアルゴリズム選択，計算誤差への対処 etc.）
- 世界中で研究が進んでいるがまだ黎明期か
  - ChatHPC（Oak Ridge National Laboratory）[1] – HPC 開発・研究に特化した AI アシスタント生成フレームワーク

[1] P. V. Lara, A. Young, J. S. Vetter, Z. Jin, S. Pophale, M. A. H. Monil, K. Teranishi, and W. F. Godoy. 2025. ChatHPC: Building the Foundations for a Productive and Trustworthy AI-Assisted HPC Ecosystem. SC '25.

# HPC-GENIE プロジェクト（1/2）

**HPC-GENIE**

- High-Performance Computing with GEnerative Neural Intelligence for Execution
- 名古屋大学情報基盤センターで発足した AI による HPC コード生成研究プロジェクト（2025/4–）
- `https://www.hpc.itc.nagoya-u.ac.jp/menu/hpc_genie.html`
- 周回遅れ感は否めないが今から追いつき，追い越す
- 学生が活躍 – 学生の AI への関心は非常に高い

# HPC-GENIE プロジェクト（2/2）

主要ミッション

- エージェント開発
  - マルチエージェント等の工夫により既存モデルを有効に活用
  - 資金と資源が必要な大規模モデル開発とは別のアプローチを模索
- 小規模モデル開発
  - 加えて HPC のための学習データセット構築，ベンチマークの構築
- ローカル LLM – 商用サービス依存を減らす
  - 課金爆発: API 利用料・サブスク料金
  - ブラックボックス: クローズドソース，研究開発のために手を入れられない
  - セキュリティ懸念: 外部サーバでコードや研究データが処理される
  - 国産技術開発
- 富岳 NEXT のための Fortran to GPU

# GPT-4.1 & o4-mini を用いた BLAS コード生成 [2)]

**正しく動作するコード数（10 回生成中）**

| | | 非最適化コード | | 最適化コード | |
|---|---|---|---|---|---|
| Level | Routine | GPT-4.1 | o4-mini | GPT-4.1 | o4-mini |
| 1 | dasum | 10 | 10 | 9 | 7 |
| | daxpy | 10 | 10 | 10 | 10 |
| | ddot | 10 | 10 | 8 | 9 |
| | idamax | 3 | 10 | 2 | 9 |
| | dnrm2 | 10 | 10 | 7 | 5 |
| | drot | 10 | 10 | 8 | 9 |
| 2 | dgemv | 8 | 9 | 3 | 6 |
| | dger | 10 | 10 | 7 | 7 |
| | dsymv | 8 | 8 | 0 | 2 |
| | dsyr2 | 8 | 9 | 2 | 7 |
| | dtrmv | 3 | 4 | 0 | 5 |
| | dtrsv | 8 | 9 | 0 | 4 |
| 3 | dgemm | 10 | 10 | 3 | 0 |
| | dsymm | 4 | 8 | 3 | 3 |
| | dsyrk | 3 | 10 | 0 | 1 |
| | dsyr2k | 3 | 10 | 0 | 0 |
| | dtrmm | 0 | 1 | 0 | 0 |
| | dtrsm | 0 | 0 | 0 | 0 |

- OpenAI GPT-4.1 & o4-mini – エントリーレベルの汎用 LLM
- 単純なプロンプトによる一発生成：「*BLAS* の *#ROUTINE#* ルーチンを *C* 言語で実装せよ」（実際は英語）
- ルーチン名だけから動作するコードが生成される
- 生成コードは Fortran リファレンスコードと異なるコード構造が多い（なるべく短いコードを生成）
- 性能最適化コードの一発生成は不十分

[2)] D. Mukunoki, S. Hayashi, T. Hoshino, T. Katagiri, "Performance Evaluation of General Purpose Large Language Models for Basic Linear Algebra Subprograms Code Generation", arXiv preprint arXiv:2507.04697, 2025.
椋木大地，林俊一郎，星野哲也，片桐孝洋，BLAS コードを題材とした GPT モデルによる数値計算コード実装支援に関する考察，第 200 回ハイパフォーマンスコンピューティング研究発表会（SWoPP2025），2025 年 8 月 4 日.

# Intel GPU 向け SYCL コード生成能力の調査 [4)]

**Performance (Level 2: GB/s, Level 3: GFlops/s, Intel Arc B580, oneAPI 2025.2.1)**

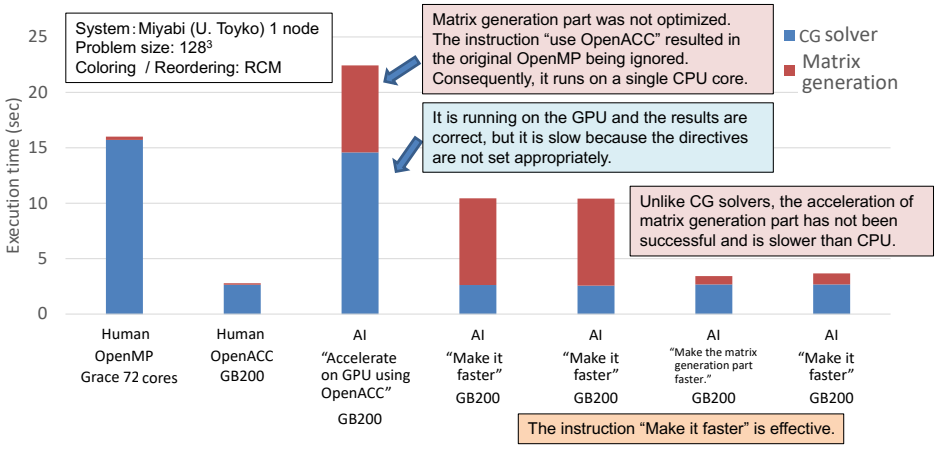
| Level | Routine | MKL | run 1 | run 2 | run 3 |
|---|---|---|---|---|---|
| 2 | SGEMV | 413.96 | 408.49 | 328.53 | 316.64 |
| | SSYMV | 123.01 | 158.93 | 151.03 | 87.72 |
| | STRSV | 86.07 | 11.83 | 59.46 | 11.75 |
| 3 | SGEMM | 14030.35 | 3340.28 | 4380.32 | 3643.29 |
| | SSYMM | 8863.38 | 3247.80 | 4207.13 | 2359.42 |
| | SSYRK | 7613.51 | 3287.25 | 5343.23 | 3680.24 |

- Claude Code[3)] による Intel GPU 向け SYCL コード生成（対象：BLAS）
- 入力：CPU 向けの最適化されてないコード，最適化手順書（ターゲットハードウェア，コンパイル方法，テスト方法）
- プロンプト：「手順書に従って最適化せよ」→「もっと最適化しろ」→「もっと最適化しろ」
- NVIDIA 以外の GPU に対応することでベンダーロックインの回避に期待

[3)] モデル：claude-sonnet-4-5-20250929

[4)] K. Morita, D. Mukunoki, T. Hoshino, T. Katagiri, Evaluation of the Capability of Coding AI in Generating SYCL-Based Numerical Computation Codes for Intel GPUs, SCA/HPCAsia2026, poster session, Jan. 2026 (accepted).

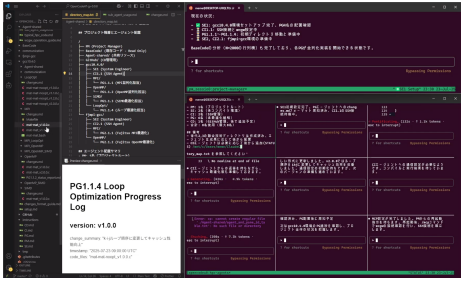
# Claude Code による Fortran コード（GeoFEM）の GPU 化 [5)]

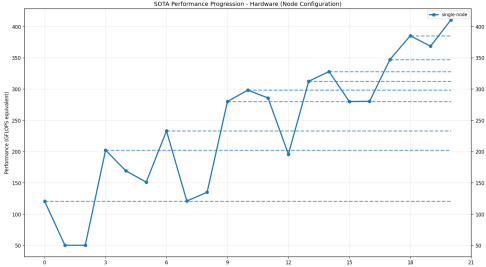



**GPU acceleration of GeoFEM (matrix generation + CG solver) on NVIDIA GB200**

[5)] T. Hoshino, S. Hayashi, D. Mukunoki, T. Katagiri and T. Hanawa, Evaluating Claude Code's Coding and Test Automation for GPU Acceleration of a Legacy Fortran Application: A GeoFEM Case Study, LLM4HPCAsia 2026, 2026 (accepted).

# VibeCodeHPC[6][7]



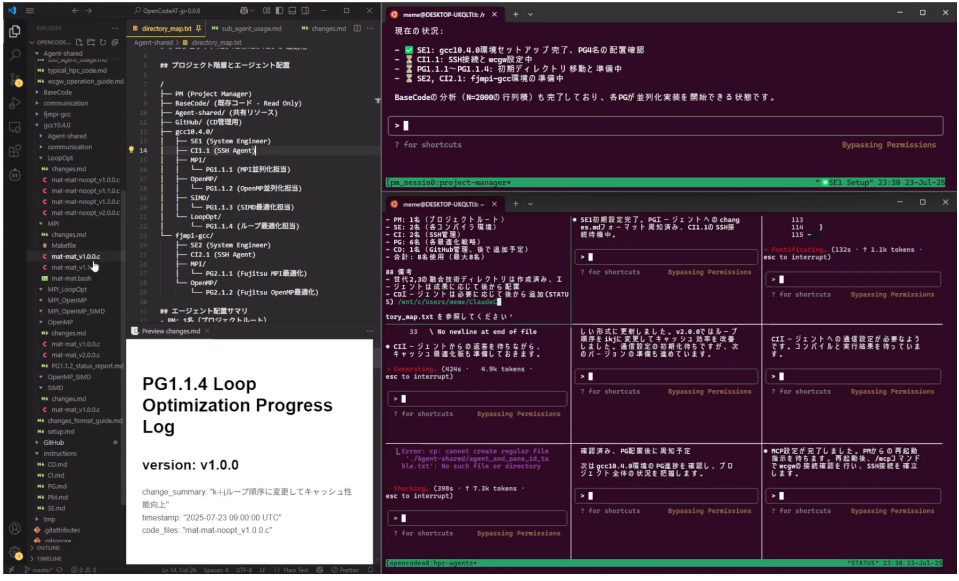

**VibeCodeHPC** (主要開発者：林俊一郎（名古屋大学 修士 1 年）)

- CLI コーディングエージェント（Claude Code）を利用したスパコン向けコード開発エージェントシステム
- 役割の異なる複数のエージェントが連携して作業するマルチエージェントシステム
- Vibe Coding も可能であるが長時間完全自律動作も可能

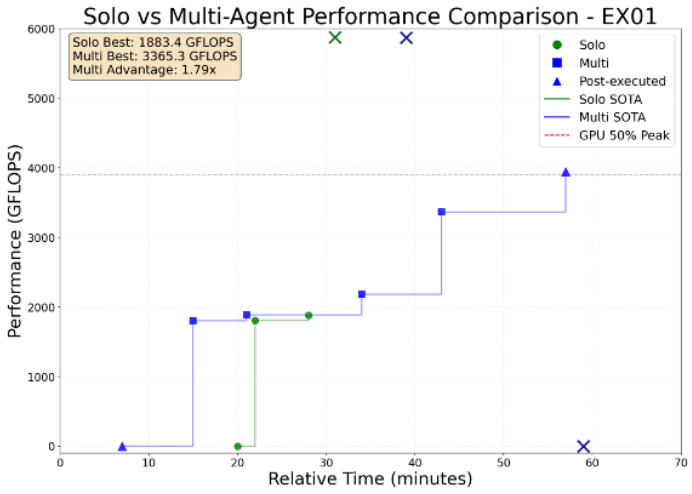
[6] https://github.com/Katagiri-Hoshino-Lab/VibeCodeHPC-jp

[7] Shun-ichiro Hayashi, Koki Morita, Daichi Mukunoki, Tetsuya Hoshino, Takahiro Katagiri, VibeCodeHPC: An Agent-Based Iterative Prompting Auto-Tuner for HPC Code Generation Using LLMs, arXiv preprint arXiv:2510.00031,2025.
林俊一郎，森田光貴，椋木大地，星野哲也，片桐孝洋．VibeCodeHPC：自動コード最適化 CLI 型マルチエージェント，第 255 回システム・アーキテクチャ・第 202 回ハイパフォーマンスコンピューティング合同研究発表会，2025 年 12 月 15 日．

# VibeCodeHPC – デモ動画



# VibeCodeHPC – ソロエージェント vs マルチエージェント




マルチエージェントにより開発失敗や要件定義違反の発生を防ぐ

# VibeCodeHPC – 生成コードの性能

**DGEMM（行列積）**
**問題サイズ: m=n=k=2048, in GFlops/s**

| PG role = dir. config. | Agent config. | run 1 | run 2 | run 3 |
|---|---|---|---|---|
| OpenMP | Multi | 197.7 | 213.0 | 189.6 |
| | Solo | – | – | – |
| MPI | Multi | 6.7 | 30.7 | – |
| | Solo | – | – | – |
| CUDA | Multi | **1384.6** | **3677.2** | 916.2 |
| | Solo | 804.5 | 606.9 | **1097.7** |
| OpenACC | Multi | 249.8 | – | – |
| | Solo | – | – | – |
| MPI+CUDA | Multi | – | – | 332.1 |
| | Solo | – | – | – |
| SIMD | Multi | – | 183.5 | – |
| | Solo | – | – | – |

**姫野ベンチマーク（ポアソンソルバー）**
**問題サイズ：SMALL, in GFlops/s**

| PG role = dir. config. | Agent config. | run 1 | run 2 | run 3 |
|---|---|---|---|---|
| OpenMP | Multi | 47.2 | **77.8** | **67.1** |
| | Solo | 21.5 | 23.9 | 39.9 |
| MPI | Multi | 40.1 | 57.6 | 61.1 |
| | Solo | – | – | – |
| CUDA | Multi | 21.1 | 24.5 | 23.3 |
| | Solo | 14.1 | – | 19.5 |
| OpenACC | Multi | 24.4 | 20.6 | 20.3 |
| | Solo | 20.5 | – | – |
| OpenMP+MPI | Multi | **68.8** | 54.2 | – |
| | Solo | – | – | – |

CPU: Xeon Gold 6230 (20 cores) × 2 (2.688 TFlops/s per CPU, 281.5 GB/s)
GPU: Tesla V100 × 4, (7.8 TFlops/s per GPU, 900 GB/s)

PG エージェントの作業分担は専門（利用する技術）ごとに自動で設定される．
マルチエージェントが優位（ソロエージェントは全部を 1 人でこなしている）．

# ローカル PC だけで動作するコード自動最適化システムの試作 [8)]

概要

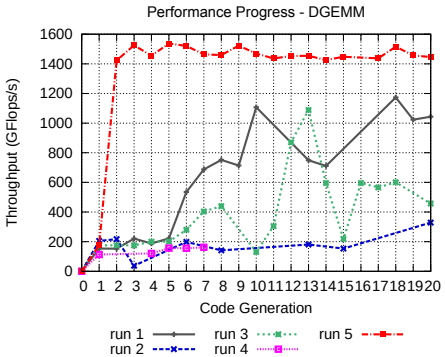
- 研究開発用にコード性能最適化システムをいちから実装したい
  - 現時点ではトイ的な非常にシンプルなもの
  - 単一コードの最適化だけを反復的に繰り返す
- LLM も含めてローカル PC で動作させたい
  - コンシューマ PC で動くレベルの LLM は 7 ヶ月前の商用最高モデル相当の性能を達成する（Epoch AI）
  - gpt-oss-120b （OpenAI）を利用 – o4-mini 相当（一世代前の ChatGPT ぐらい），Ryzen AI Max+や MacBook Pro の 128GB メモリで動作
- 疑似マルチエージェント構成で反復的に最適化
  1. Project Manager（PM）LLM が 5 つ Programmer（PG）に対する最適化戦略を生成
  2. PG が 5 つのコードを生成
  3. 一番良いコードを採用し PM は有効・無効な最適化方法を分析
  - 以上を 20 回反復

[8)] 椋木大地，森田光貴，林俊一郎，三笠諒，星野哲也，片桐孝洋，gpt-oss-120b を用いたコード自動最適化マルチエージェントシステムの試作，第 255 回システム・アーキテクチャ・第 202 回ハイパフォーマンスコンピューティング合同研究発表会，2025 年 12 月 15 日.

# DGEMM（CPU）

**Input Code**

```
1  void exp1_opt(int N,
2                double *A,
3                double *B,
4                double *C) {
5    for (int i = 0; i < N; i++) {
6      for (int j = 0; j < N; j++) {
7        double sum = 0.0;
8        for (int k = 0; k < N; k++) {
9          sum += A[i*N+k] * B[k*N+j];
10       }
11       C[i*N+j] = sum;
12     }
13   }
14 }
```




- Intel Xeon Gold 6230（20 コア）× 2，$n = 1024$ の正方行列
- 最速コードは MKL を呼んでいた → LLM がコードが行列積であることを認識
- 最速のフルスクラッチコード：1173.42 GFlops/s（MKL の 76%）
- 実験毎に結果が大きく異なるが，試行回数を増やせば良い結果が得られる

# DGEMM（CPU） – 生成された最速コード（1/3）

```
static inline void pack_B_panel(const double *B_src,
    double *B_pack, int N, int kp, int jp, int k_max,
    int blk_width) {
  for (int k = kp; k < k_max; ++k) {
    const double *src = &B_src[k * N + jp];
    double *dst = &B_pack[(k - kp) * blk_width];
    int j = 0;
    _mm_prefetch((const char *)src, _MM_HINT_T0);
    for (; j + 8 <= blk_width; j += 8) {
      __m512d v = _mm512_loadu_pd(&src[j]);
      _mm512_storeu_pd(&dst[j], v);
    }
    for (; j < blk_width; ++j) { dst[j] = src[j]; }
  }
  _mm_sfence();
}
void exp1_opt(int N, double *restrict A, double *
    restrict B, double *restrict C) {
  const int BLOCK = 64;
  omp_set_num_threads(omp_get_max_threads());
  #pragma omp parallel for collapse(2) schedule(static)
  for (int ii = 0; ii < N; ii += BLOCK) {
    for (int jj = 0; jj < N; jj += BLOCK) {
      int i_max = (ii + BLOCK > N) ? N : ii + BLOCK;
      int j_max = (jj + BLOCK > N) ? N : jj + BLOCK;
      int blk_width = j_max - jj;
      int vec_width = blk_width & ~31;
      double C_local[BLOCK * BLOCK] __attribute__((
          aligned(64)));
      double *C_local_aligned = (double *)__
          builtin_assume_aligned(C_local, 64);
      for (int i = 0; i < BLOCK * BLOCK; ++i)
          C_local_aligned[i] = 0.0;
      for (int kk = 0; kk < N; kk += BLOCK) {
        int k_max = (kk + BLOCK > N) ? N : kk +
            BLOCK;
        double B_pack_local[BLOCK * BLOCK] __
            attribute__((aligned(64)));
        pack_B_panel(B, B_pack_local, N, kk, jj,
            k_max, blk_width);
        for (int i = ii; i + 3 < i_max; i += 4) {
          const double *a_row0 = &A[(i + 0) * N +
              kk];
          const double *a_row1 = &A[(i + 1) * N +
              kk];
          const double *a_row2 = &A[(i + 2) * N +
              kk];
          const double *a_row3 = &A[(i + 3) * N +
              kk];
          double *c_loc0 = &C_local_aligned[(i - ii
              ) * BLOCK];
          double *c_loc1 = &C_local_aligned[(i - ii
              + 1) * BLOCK];
          double *c_loc2 = &C_local_aligned[(i - ii
              + 2) * BLOCK];
          double *c_loc3 = &C_local_aligned[(i - ii
              + 3) * BLOCK];
          for (int j = 0; j < vec_width; j += 16) {
            __m512d c00 = _mm512_loadu_pd(&c_loc0[
                j + 0]);
            __m512d c01 = _mm512_loadu_pd(&c_loc0[
                j + 8]);
            __m512d c10 = _mm512_loadu_pd(&c_loc1[
                j + 0]);
            __m512d c11 = _mm512_loadu_pd(&c_loc1[
                j + 8]);
            __m512d c20 = _mm512_loadu_pd(&c_loc2[
                j + 0]);
            __m512d c21 = _mm512_loadu_pd(&c_loc2[
                j + 8]);
            __m512d c30 = _mm512_loadu_pd(&c_loc3[
                j + 0]);
            __m512d c31 = _mm512_loadu_pd(&c_loc3[
                j + 8]);
            for (int k = kk; k < k_max; ++k) {
              const int k_off = k - kk;
              __m512d b0 = _mm512_load_pd(&
                  B_pack_local[k_off * blk_width
                  + j + 0]);
              __m512d b1 = _mm512_load_pd(&
                  B_pack_local[k_off * blk_width
                  + j + 8]);
              __m512d a0 = _mm512_set1_pd(a_row0[
                  k_off]);
              __m512d a1 = _mm512_set1_pd(a_row1[
                  k_off]);
```

# DGEMM（CPU） – 生成された最速コード（2/3）

```
56                      __m512d a2 = _mm512_set1_pd(a_row2[
                            k_off]);
57                      __m512d a3 = _mm512_set1_pd(a_row3[
                            k_off]);
58                      c00 = _mm512_fmadd_pd(a0, b0, c00);
                            c01 = _mm512_fmadd_pd(a0, b1,
                            c01);
59                      c10 = _mm512_fmadd_pd(a1, b0, c10);
                            c11 = _mm512_fmadd_pd(a1, b1,
                            c11);
60                      c20 = _mm512_fmadd_pd(a2, b0, c20);
                            c21 = _mm512_fmadd_pd(a2, b1,
                            c21);
61                      c30 = _mm512_fmadd_pd(a3, b0, c30);
                            c31 = _mm512_fmadd_pd(a3, b1,
                            c31);
62                    }
63                    _mm512_storeu_pd(&c_loc0[j + 0], c00);
                          _mm512_storeu_pd(&c_loc0[j + 8],
                          c01);
64                    _mm512_storeu_pd(&c_loc1[j + 0], c10);
                          _mm512_storeu_pd(&c_loc1[j + 8],
                          c11);
65                    _mm512_storeu_pd(&c_loc2[j + 0], c20);
                          _mm512_storeu_pd(&c_loc2[j + 8],
                          c21);
66                    _mm512_storeu_pd(&c_loc3[j + 0], c30);
                          _mm512_storeu_pd(&c_loc3[j + 8],
                          c31);
67                  }
68                  for (int j = vec_width; j < blk_width; ++
                        j) {
69                    double s0 = c_loc0[j]; double s1 =
                          c_loc1[j];
70                    double s2 = c_loc2[j]; double s3 =
                          c_loc3[j];
71                    for (int k = kk; k < k_max; ++k) {
72                      const int k_off = k - kk;
73                      double b = B_pack_local[k_off *
                            blk_width + j];
74                      s0 += a_row0[k_off] * b; s1 +=
                            a_row1[k_off] * b;
```

```
75                      s2 += a_row2[k_off] * b; s3 +=
                            a_row3[k_off] * b;
76                    }
77                    c_loc0[j] = s0; c_loc1[j] = s1; c_loc2
                          [j] = s2; c_loc3[j] = s3;
78                  }
79                }
80                for (int i = i_max - ((i_max - ii) % 4); i <
                      i_max; ++i) {
81                  const double *a_ptr = &A[i * N + kk];
82                  double *c_loc = &C_local_aligned[(i - ii)
                        * BLOCK];
83                  for (int j = 0; j < vec_width; j += 8) {
84                    __m512d c_vec = _mm512_loadu_pd(&c_loc
                          [j]);
85                    for (int k = kk; k < k_max; ++k) {
86                      const int k_off = k - kk;
87                      __m512d b_vec = _mm512_load_pd(&
                            B_pack_local[k_off * blk_width
                            + j]);
88                      __m512d a_vec = _mm512_set1_pd(
                            a_ptr[k_off]);
89                      c_vec = _mm512_fmadd_pd(a_vec,
                            b_vec, c_vec);
90                    }
91                    _mm512_storeu_pd(&c_loc[j], c_vec);
92                  }
93                  for (int j = vec_width; j < blk_width; ++
                        j) {
94                    double sum = c_loc[j];
95                    for (int k = kk; k < k_max; ++k) {
96                      const int k_off = k - kk;
97                      sum += a_ptr[k_off] * B_pack_local[
                            k_off * blk_width + j];
98                    }
99                    c_loc[j] = sum;
100                 }
101               }
102             }
103             for (int i = ii; i < i_max; ++i) {
104               double *c_ptr = &C[i * N + jj];
105               double *c_loc = &C_local_aligned[(i - ii) *
                      BLOCK];
```

# DGEMM（CPU） – 生成された最速コード（3/3）

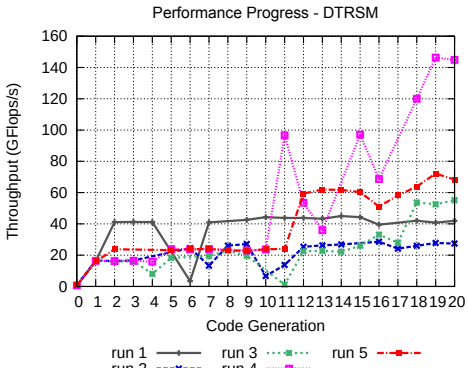
```
106             c_ptr = (double *)__builtin_assume_aligned(
                    c_ptr, 64);
107             c_loc = (double *)__builtin_assume_aligned(
                    c_loc, 64);
108             int j = 0;
109             for (; j < vec_width; j += 8) {
110                 __m512d tmp = _mm512_loadu_pd(&c_loc[j]);
111                 _mm512_storeu_pd(&c_ptr[j], tmp);
112             }
113             for (; j < blk_width; ++j) {
114                 c_ptr[j] = c_loc[j];
115             }
116         }
117     }
118 }
119
```

※掲載スペースの都合により実際に生成されたコードから自動的に挿入されたコメント，空行，ヘッダファイルの include を削除し，インデントや改行を一部修正した．

# DTRSM（CPU）

**Input Code**

```
1  void exp2_opt(int N,
2       double *A, double *B,
3       double *X, double alpha) {
4    for (int i = 0; i < N; i++) {
5      for (int j = 0; j < N; j++) {
6        X[i*N+j] = alpha * B[i*N+j];
7      }
8    }
9    for (int j = 0; j < N; j++) {
10     for (int i = 0; i < N; i++) {
11       for (int k = 0; k < i; k++) {
12         X[i*N+j] -= A[i*N+k]*X[k*N+j];
13       }
14       X[i*N+j] /= A[i*N+i];
15     }
16   }
17 }
```

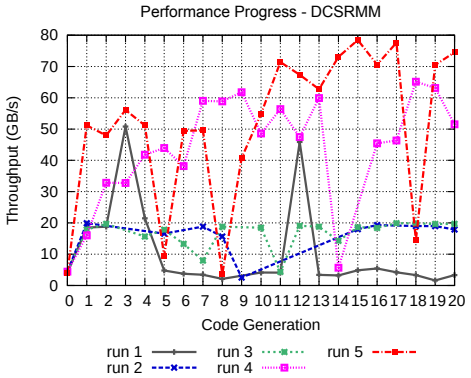



- 三角行列を係数とする方程式の計算（下三角，$n = 1024$）– データ依存性があり最適化の難易度が高い
- 最速：146.33 GFlops/s（run 4）– MKL（305.74 GFlops/s）の 48%

# DCSRMM（CPU）

## Input Code

```
1  void exp3_opt(int m, int k, int n,
2      const int *row_ptr,
3      const int *col_indices,
4      const double *values,
5      const double *B, double *C) {
6    memset(C, 0, m*n*sizeof(double));
7    for (int i = 0; i < m; i++) {
8      int row_start = row_ptr[i];
9      int row_end = row_ptr[i + 1];
10     for (int idx=row_start;
11          idx<row_end; idx++) {
12       int col = col_indices[idx];
13       double val = values[idx];
14       for (int j = 0; j < n; j++) {
15         C[i*n+j] += val*B[col*n+j];
16       }
17     }
18   }
19 }
```

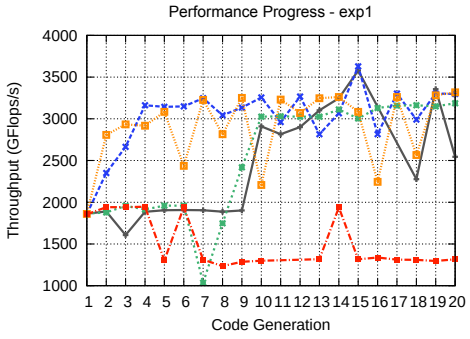



- CSR 形式の疎行列–密行列積（gridgena.mtx: $n = 48962$，$nnz = 512084$，右辺幅 $m = 128$）；不規則なメモリアクセス；疎行列の構造に依存
- 最速：78.45 GB/s（run 5 の 15 世代）（MKL: 86.88 GB/s，MKL の 90%）
- FP16 の利用を試行（失敗），格納形式の変更を試行（CSC, ELLPACK，失敗） → CSR 形式を認識している証拠

# DGEMM (CUDA) – 入力コードと実験結果

## 入力コード

```
1  __global__ void exp1_kernel(int N, const double *
       A, const double *B, double *C) {
2      int row = blockIdx.y * blockDim.y + threadIdx
           .y;
3      int col = blockIdx.x * blockDim.x + threadIdx
           .x;
4      if (row < N && col < N) {
5          double sum = 0.0;
6          for (int k = 0; k < N; k++) {
7              sum += A[row * N + k] * B[k * N + col
                   ];
8          }
9          C[row * N + col] = sum;
10     }
11 }
12 extern "C" void exp1_opt_kernel(int N, const
       double *d_A, const double *d_B, double *d_C) {
13     dim3 blockDim(16, 16);
14     dim3 gridDim((N + blockDim.x - 1) / blockDim.
           x, (N + blockDim.y - 1) / blockDim.y);
15     exp1_kernel<<<gridDim, blockDim>>>(N, d_A,
           d_B, d_C);
16     cudaDeviceSynchronize();
17 }
```




NVIDIA Tesla V100
**最速コード**: 3628.29 GFlops/s
cuBLAS: 6168.09 GFlops/s (cuBLAS の 59%)

# DGEMM（CUDA）– 生成された最速コード（1/3）

```
#include <cuda_runtime.h>
#include <stdlib.h>

extern "C" cudaError_t cudaGetDeviceProperties_v2(
     cudaDeviceProp *prop, int device) {
    return cudaGetDeviceProperties(prop, device);
}

extern "C" long __isoc23_strtol(const char *nptr, char
     **endptr, int base) {
    return strtol(nptr, endptr, base);
}

#define TILE 32                          // ×3232 タイル
#define K_UNROLL 32

// 手動アンロールマクロ
#define UNROLL_K(iter)                                  \
    do {                                                \
        double a0 = As[calc][threadIdx.y*2 + 0][iter];  \
        double a1 = As[calc][threadIdx.y*2 + 1][iter];  \
        double b0 = Bs[calc][iter][threadIdx.x*2 + 0];  \
        double b1 = Bs[calc][iter][threadIdx.x*2 + 1];  \
        regC[0][0] += a0 * b0;                          \
        regC[0][1] += a0 * b1;                          \
        regC[1][0] += a1 * b0;                          \
        regC[1][1] += a1 * b1;                          \
    } while (0)

__global__ void exp1_kernel(int N, const double * __
     restrict__ A,
                            const double * __restrict__ B
                            double * __restrict__ C) {
    double regC[2][2] = {{0.0, 0.0},
                         {0.0, 0.0}};                   //
                                                  ×22 出力

    __shared__ double As[2][TILE][TILE];
    __shared__ double Bs[2][TILE][TILE];

    int numTiles = (N + TILE - 1) / TILE;

    int baseRow = blockIdx.y * TILE + threadIdx.y * 2;
                  // 2 行担当
    int baseCol = blockIdx.x * TILE + threadIdx.x * 2;
                  // 2 列担当

    // ---- 1st タイルを load (ping = 0) ----
    int ping = 0;
    for (int dy = 0; dy < 2; ++dy) {
        int aRow = baseRow + dy;
        int aCol = 0 * TILE + threadIdx.x * 2;
        double4 a_pair = *((double4*)(A + aRow * N +
             aCol));
        double a0 = a_pair.x;
        double a1 = a_pair.y;
        As[ping][threadIdx.y * 2 + dy][threadIdx.x * 2]
             = a0;
        As[ping][threadIdx.y * 2 + dy][threadIdx.x * 2 +
             1] = a1;
    }
    for (int dy = 0; dy < 2; ++dy) {
        int bRow = 0 * TILE + threadIdx.y * 2 + dy;
        int bCol = baseCol;
        double4 b_pair = *((double4*)(B + bRow * N +
             bCol));
        double b0 = b_pair.x;
        double b1 = b_pair.y;
        Bs[ping][threadIdx.y * 2 + dy][threadIdx.x * 2]
             = b0;
        Bs[ping][threadIdx.y * 2 + dy][threadIdx.x * 2 +
             1] = b1;
    }
    __syncthreads();
                                                  // 初期タイルロード
          待ち

    for (int t = 0; t < numTiles; ++t) {
```

# DGEMM（CUDA）– 生成された最速コード（2/3）

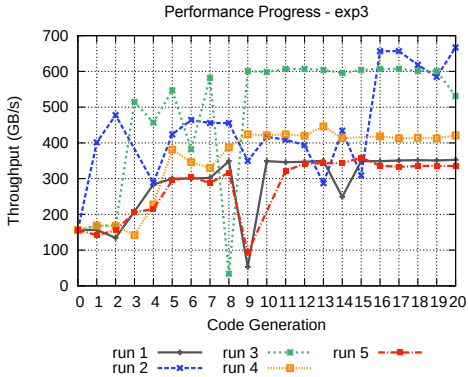
```
        int calc = ping;                                    // 現在計算に使用するバッファ
        // ---- 計算 (calc バッファ) ----
        UNROLL_K(0);  UNROLL_K(1);  UNROLL_K(2);  UNROLL_K(3);
        UNROLL_K(4);  UNROLL_K(5);  UNROLL_K(6);  UNROLL_K(7);
        UNROLL_K(8);  UNROLL_K(9);  UNROLL_K(10); UNROLL_K(11);
        UNROLL_K(12); UNROLL_K(13); UNROLL_K(14); UNROLL_K(15);
        UNROLL_K(16); UNROLL_K(17); UNROLL_K(18); UNROLL_K(19);
        UNROLL_K(20); UNROLL_K(21); UNROLL_K(22); UNROLL_K(23);
        UNROLL_K(24); UNROLL_K(25); UNROLL_K(26); UNROLL_K(27);
        UNROLL_K(28); UNROLL_K(29); UNROLL_K(30); UNROLL_K(31);

        // 次タイルを prefetch (ping バッファ) —すでに最後の場合はスキップ
        ping = 1 - ping;                                    // バッファを反転
        if (t + 1 < numTiles) {
            for (int dy = 0; dy < 2; ++dy) {
                int aRow = baseRow + dy;
                int aCol = (t + 1) * TILE + threadIdx.x * 2;
                double4 a_pair = *((double4*)(A + aRow * N + aCol));
                double a0 = a_pair.x;
                double a1 = a_pair.y;
                As[ping][threadIdx.y * 2 + dy][threadIdx.x * 2]     = a0;
                As[ping][threadIdx.y * 2 + dy][threadIdx.x * 2 + 1] = a1;
            }
            for (int dy = 0; dy < 2; ++dy) {
                int bRow = (t + 1) * TILE + threadIdx.y * 2 + dy;
                int bCol = baseCol;
                double4 b_pair = *((double4*)(B + bRow * N + bCol));
                double b0 = b_pair.x;
                double b1 = b_pair.y;
                Bs[ping][threadIdx.y * 2 + dy][threadIdx.x * 2]     = b0;
                Bs[ping][threadIdx.y * 2 + dy][threadIdx.x * 2 + 1] = b1;
            }
        }
        __syncthreads();                                    // 次計算に備える
    }

    // ---- 書き戻し ----
    C[baseRow * N + baseCol]                    = regC[0][0];
    C[baseRow * N + baseCol + 1]                = regC[0][1];
    C[(baseRow + 1) * N + baseCol]              = regC[1][0];
    C[(baseRow + 1) * N + baseCol + 1]          = regC[1][1];
}

extern "C" void exp1_opt_kernel(int N, const double *d_A,
                                const double *d_B,
                                double *d_C) {
    if (N % 8 != 0) {
        int Npad = ((N + 7) / 8) * 8;
        size_t pitch_pad = (size_t)Npad * sizeof(double);
        size_t pitch_orig = (size_t)N * sizeof(double);
        double *d_Apad = nullptr, *d_Bpad = nullptr, *d_Cpad = nullptr;
        cudaMalloc(&d_Apad, pitch_pad * Npad);
        cudaMalloc(&d_Bpad, pitch_pad * Npad);
        cudaMalloc(&d_Cpad, pitch_pad * Npad);
        cudaMemset(d_Apad, 0, pitch_pad * Npad);
        cudaMemset(d_Bpad, 0, pitch_pad * Npad);
        cudaMemset(d_Cpad, 0, pitch_pad * Npad);
```

# DGEMM（CUDA）– 生成された最速コード（3/3）

```
        cudaMemcpy2D(d_Apad, pitch_pad, d_A, pitch_orig,
                     pitch_orig, N,
                          cudaMemcpyDeviceToDevice);
        cudaMemcpy2D(d_Bpad, pitch_pad, d_B, pitch_orig,
                     pitch_orig, N,
                          cudaMemcpyDeviceToDevice);

        dim3 blockDim(TILE / 2, TILE / 2);
        dim3 gridDim((Npad + TILE - 1) / TILE, (Npad +
            TILE - 1) / TILE);
        exp1_kernel<<<gridDim, blockDim>>>(Npad, d_Apad,
             d_Bpad, d_Cpad);
        cudaDeviceSynchronize();

        cudaMemcpy2D(d_C, pitch_orig, d_Cpad, pitch_pad,
                     pitch_orig, N,
                          cudaMemcpyDeviceToDevice);

        cudaFree(d_Apad);
        cudaFree(d_Bpad);
        cudaFree(d_Cpad);
        return;
    }

    dim3 blockDim(TILE / 2, TILE / 2);
                                  // ×1616 スレッド
    dim3 gridDim((N + TILE - 1) / TILE, (N + TILE - 1) /
         TILE);
    exp1_kernel<<<gridDim, blockDim>>>(N, d_A, d_B, d_C)
        ;
    cudaDeviceSynchronize();
}

#undef K_UNROLL
#undef UNROLL_K
```

# DCSRMM（CUDA）

```
1  __global__ void exp3_kernel(int m, int n, const
       int *row_ptr, const int *col_indices, const
       double *values, const double *B, double *C) {
2      int row = blockIdx.x * blockDim.x + threadIdx
           .x;
3      int col = blockIdx.y * blockDim.y + threadIdx
           .y;
4      if (row < m && col < n) {
5          double sum = 0.0;
6          int row_start = row_ptr[row];
7          int row_end = row_ptr[row + 1];
8          for (int idx = row_start; idx < row_end;
               idx++) {
9              int sparse_col = col_indices[idx];
10             double val = values[idx];
11             sum += val * B[sparse_col * n + col];
12         }
13         C[row * n + col] = sum;
14     }
15 }
16 extern "C" void exp3_opt_kernel(int m, int k, int
       n, const int *d_row_ptr, const int *
       d_col_indices, const double *d_values, const
       double *d_B, double *d_C) {
17     cudaMemset(d_C, 0, m * n * sizeof(double));
18     dim3 blockDim(16, 16);
19     dim3 gridDim((m + blockDim.x - 1) / blockDim.
           x, (n + blockDim.y - 1) / blockDim.y);
20     exp3_kernel<<<gridDim, blockDim>>>(m, n,
           d_row_ptr, d_col_indices, d_values, d_B,
           d_C);
21     cudaDeviceSynchronize();
22 }
```




Best code: 666.78 GFlops/s
cuSparse: 397.58 GFlops/s
→ 特定の入力に特化した最適化に成功

# The AI Scientist for HPC


Adaptive Concurrency Scaling for Stabilizing OpenMP..AS Performance in Heterogeneous HPC Systems

[redacted]

Abstract

1. Introduction

2. Related Work

3. Background

4. Method

5. Experimental Setup

6. Results

7. Conclusion

References


2. Related Work

3. Background

4. Method

5. Experimental Setup

6. Results

7. Conclusion

References


3. Background

4. Method

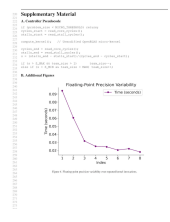
5. Experimental Setup

6. Results

7. Conclusion

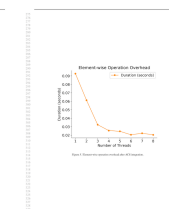
References


References


Supplementary Material

A. Controller Pseudocode

B. Additional Figures


Supplementary Material

A. Controller Pseudocode

B. Additional Figures

**The AI Scientist**（Sakana AI）[a]

- 研究の自動化を実証（アイディア生成 → 実装 → 実験 → 評価 → 論文執筆）
- AI 研究に特化，HPC 研究に利用できない

**The AI Scientist の HPC 研究者化** [b]

- C/C++実装 → コンパイル → デバッグ・性能評価をするループを実装し，コードの性能最適化を伴う研究に対応
- 自動生成された論文は不十分であるが研究の半自動化を達成しつつある

[a] The AI Scientist-v2: Workshop-Level Automated Scientific Discovery via Agentic Tree Search, arXiv:2504.08066, 2025.

[b] Takanori Kotama, Rio Yokota, Daichi Mukunoki, Tetsuya Hoshino, Takahiro Katagiri, Proposal of The AI Scientist v2 for High Performance Computing with Local Large Language Models, SCA/HPCAsia2026, Poster Session, Jan. 2026 (accepted).

# 商用コーディングエージェント利用の雑感（1/2）

私の利用状況

- Claude Code，Gemini CLI，Codex の最大プランを契約（合計月 10 万ぐらい）
- 既存プロジェクト（Fortran/C/C++）の機能追加・最適化・デバッグ・GPU 化で利用
- 最近はほとんどコードを書かなくなった（Vibe Coding）

どういうことができるか？

- 既存コードをほぼ全自動で GPU 移植（まあまあな性能）
- 専門書に載っているレベルの一般的な数値計算法を理解し最適な手法を提示
  - コード中のソルバ部分を理解し，疎行列構造を理解し典型的アルゴリズム，前処理，疎行列フォーマットを提案してくれる
- ライブラリの利用方法を知っていて自動で環境構築
- コードの妥当性検証のための簡単なテストの生成

# 商用コーディングエージェント利用の雑感（2/2）

課題

- ぐちゃぐちゃで大規模なコードほど難易度が上がる
- 嘘，失敗，記憶喪失，堂々巡り，etc.
- 著作権，責任の所在，過度な AI 依存，etc.

**“Vibe Resarch”** 的な利用

- コーディングだけでなくアイディア出しや論文執筆への利用を試行
- 半自動の The AI Scientist for HPC
- 人間が大まかなアイディアを提示 → （ここから LLM エージェント）背景の妥当性，関連研究調査，新規性の検討 → 実装・実験方法を含めて論文を書かせる → それを仕様書としてプログラムを実装し実験 → 論文に反映

素人が使うツールではなく専門家が使う武器

- 全然使えないと言ってる人もいる（特に達人級のプログラマ）が，使い方次第
- リテラシとして使いこなしを考える
- 急速な技術進化についていく

# まとめ

**HPC-GENIE プロジェクト（名古屋大学情報基盤センター）**

- 生成 AI の HPC コード開発への活用にむけた研究
- 特にエージェント開発にフォーカス – VibeCodeHPC
- 国産・オープン・ローカル実行可能な技術開発

**HPC コード開発における生成 AI への期待**

- 課題や制約はまだ多くあるが，研究開発の効率を劇的に向上させている
- 富岳 NEXT を見据えたコードの GPU 化において強い期待感

提言（議題？）

- 資金 – AI の利用のため，人材の育成のため，人材の確保のため
- 急速な技術発展・状況変化にどうついていくか
  - 数ヶ月で技術が 1 周している印象
  - アカデミアにおける教育・研究のやり方自体がついていけていない