汎用 LLM による BLAS コード自動生成能力の考察 [1)2)]

椋木 大地 [1]　林 俊一朗 [2]　星野 哲也 [1]　片桐 孝洋 [1]

[1] 名古屋大学 情報基盤センター

[2] 名古屋大学大学院 情報学研究科

第 6 回スーパーコンピュータ「不老」ユーザ会
2025 年 9 月 11 日

[1)] 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.

[2)] 本研究は学際大規模情報基盤共同利用・共同研究拠点（JHPCN）および革新的ハイパフォーマンス・コンピューティング・インフラ（HPCI）の支援による (課題番号:jh250015)．また本研究は JSPS 科研費 JP23K11126，JP24K02945 の助成を受けた．

背景と目的（1/2）

背景

- 高度な技術を有するソフトウェアエンジニアが不足
- 数値計算・HPC コードの開発においては幅広い専門知識が必要
  - 単に動けばいいコードとは異なり，複雑な性能最適化が必要
  - 数値計算コードでは計算誤差など特有の問題がある
  - Fortran コードなどのレガシー資産，過去のプロジェクトの維持・最新技術適用が難しい

大規模言語モデル（LLM）によるプログラム開発支援技術の急速な発展

- LLM は ChatGPT 等における自然言語処理からコード生成へと応用範囲を拡大
- さまざまなサービス：OpenAI Codex, Anthropic Claude Code, Google Gemini CLI, etc.
- 数値計算・HPC コードの生成においては課題がある
  - 専門家のみが利用するため実装例が少なく十分に学習されていない
  - 難易度が高い：実装・性能最適化のために専門知識・専門技術が必要

背景と目的（2/2）

本研究の概要

- BLAS（Basic Linear Algebra Subprograms）の C 言語コード生成
  - 科学技術計算で広く使用されるベクトル／行列演算を行う基本線形代数演算ライブラリ
  - Fortran による参照実装コードが公開されている（性能最適化・並列化されていない）
- 普及価格帯の汎用 LLM（GPT-4.1, o4-mini）を利用
- 自然言語による最小限の指示あるいは参照 Fortran コード添付による C 言語コード生成
- 基礎的な性能最適化（スレッド並列化，SIMD 化，ブロッキング）を適用したコード生成

問題設定の動機

- BLAS は最も簡単な数値計算コードの題材として適当
- BLAS は多数のルーチンで構成され，詳細なインタフェース定義，実行オプションがあり，実装難易度がルーチンにより異なるため，包括的な評価が可能
- BLAS は Fortran の参照実装を含むいくつかのコードとドキュメントがインターネット上に公開されており，LLM が何を学習しているか考察可能
- コーディング向けではない普及価格帯の汎用 LLM の評価＝ローカル LLM 相当？

関連研究

HPC コード生成研究

- HPC-GPT（Ding et al., SC'23 Workshops），HPC-Coder（Nichols et al., ISC 2024），LM4HPC（Chen et al., IWOMP 2023）．MARCO（Rahman et al., arXiv, 2025），etc.

行列計算コード生成

- ChatBLAS（Valero-Lara et al., SC24-W）
  - OpenMP，CUDA，HIP による BLAS のベクトル演算ルーチン（level-1 BLAS）を ChatGPT（GPT 3.5-Turbo, 4o）により生成
  - 10 コード生成し OpenMP で 93–100%，CUDA で 66%，HIP で 75–82%が動作
  - プロンプトエンジニアリング，ファインチューニングによる改善
  - → Level-1 BLAS のみ．level-1 は簡単ではないか？
- QiMeng-GEMM（Zhou et al., AAAI 2025）
  - GPT-4o，Claude 3.5 Sonnet に対するプロンプトエンジニアリングによる行列積（BLAS の GEMM ルーチン）のコード生成
  - CPU（RISC-V C910）で OpenBLAS の 211%，GPU（RTX4070）で cuBLAS の 115%の性能
  - → 評価対象 CPU がマイナー（RISC-V C910），GEMM は実装例や実装解説がウェブ上に多くあり，簡単ではないか？

評価対象 LLM モデル

OpenAI GPT-4.1 と o4-mini（両者 2025 年 4 月リリース）

- GPT-4.1：応答速度と効率性に重点を置いた汎用モデル
- o4-mini：推論性能に優れた「reasoning 型」モデル
- いずれもコード生成特化型ではなく，OpenAI の最上位モデルではないが，価格性能比，応答速度に優れた標準的な汎用モデルと位置付けられる

	GPT-4.1	o4-mini
コンテキストウィンドウ	1,047,576	200,000
最大出力トークン	32,768	100,000
ナレッジカットオフ	2024 年 6 月 1 日	2024 年 6 月 1 日
入力料金（100 万トークンあたり）	$2.00	$1.10
出力料金（100 万トークンあたり）	$8.00	$4.40
推論トークン	なし	あり

評価対象 BLAS ルーチン

20 種類の倍精度実数ルーチン（次頁の表参照）

- Level-1 BLAS（ベクトル演算）：dasum, daxpy, ddot, idamax, dnrm2, drot, drotm
- Level-2 BLAS（行列-ベクトル演算）：dgemv, dger, dsymv, dsyr, dsyr2, dtrmv, dtrsv
- Level-3 BLAS（行列-行列演算）：dgemm, dsymm, dsyrk, dsyr2k, dtrmm, dtrsm
- ルーチンにより以下の実行パラメータをサポート
  - incx/incy：ベクトルの要素間隔（1, 2）
  - trans：行列の転置（'N', 'T'）
  - side：行列の位置（'L', 'R'）
  - uplo：三角行列の使用部分（'L', 'U'）
  - diag：単位三角行列か否か（'U', 'N'）

動作検証

- BLAS++のテストプログラム（run_tests.py）で動作検証
- Fortran の参照実装と比較しマシンイプシロンに基づく許容相対誤差ノルムで評価
- パラメータの全組合せを考慮，行列は正方と比率の異なる非正方パターンも考慮
  - 複雑なルーチンほど評価パターン数も増加（例：dtrsm で 256 ケース）

評価対象 BLAS ルーチン

Level	Routine	Description	Parameters	Equation
1	dasum	sum of magnitudes of vector elements	incx	$v = \|x_1\| + \cdots + \|x_n\|$
	daxpy	vector-scalar product and add vector	incx, incy	$y = \alpha x + y$
	ddot	vector-vector dot product	incx, incy	$v = x^T y$
	idamax	index of element with max absolute value	incx	$v = 1^{st} k \ni \|\mathrm{re}(\mathrm{x_k})\| + \|\mathrm{im}(\mathrm{x_k})\|$
	dnrm2	Euclidean norm of vector	incx	$v = \|\|x\|\|_2$
	drot	rotation of points in plane	incx, incy	$x_i = cx_i + sy_i,\ y_i = -sx_i + cy_i$
	drotm	modified Givens rotation	incx, incy	$\begin{bmatrix} x_i / y_i \end{bmatrix} = H \begin{bmatrix} x_i / y_i \end{bmatrix}$
2	dgemv	general matrix-vector multiply	trans, incx, incy	$y = \alpha Ax + \beta y$
	dger	general matrix rank-1 update	incx, incy	$A = \alpha x y^T + A$
	dsymv	symmetric matrix-vector multiply	uplo, incx, incy	$y = \alpha Ax + \beta y$
	dsyr	symmetric rank-1 update	uplo, incx	$A = \alpha x x^T + A$
	dsyr2	symmetric rank-2 update	uplo, incx, incy	$A = \alpha x y^T + \alpha y x^T + A$
	dtrmv	triangular matrix-vector multiply	uplo, diag, incx	$x = Ax$
	dtrsv	triangular matrix-vector solve	uplo, trans, diag, incx	$x = A^{-1} x$
3	dgemm	general matrix-matrix multiply	transa, transb	$C = \alpha AB + \beta C$
	dsymm	symmetric matrix-matrix multiply	side, uplo	$C + \alpha AB + \beta C$
	dsyrk	symmetric rank-k update	uplo, trans	$C = \alpha A A^T + \beta C$
	dsyr2k	symmetric rank-2k update	uplo, trans	$C = \alpha AB^T + \alpha BA^T + \beta C$
	dtrmm	triangular matrix-matrix multiply	side, uplo, trans, diag	$B = \alpha AB$
	dtrsm	triangular matrix-matrix solve	side, uplo, trans, diag	$B = \alpha A^{-1} B$

コード生成プロンプト

結果を先に言うと，**BLAS のルーチン名を与えるたけで正しく動作するコードが生成される**

3 ケースのプロンプト

- **NameToCcode**：ルーチン名のみを与えて C 言語コードを生成
- **NameToOptCcode**：ルーチン名のみを与えて基礎的な性能最適化（並列化・SIMD・キャッシュブロッキング）を考慮した C 言語コードを生成
- **FrtcodeToOptCcode**：Fortran 参照実装コードを与えたうえで基礎的な性能最適化（同上）を考慮した C 言語コードを生成（参照実装コードには冒頭に仕様が英語で書かれている）

プロンプトに与える補助情報

- BLAS++のテストプログラムに組み込んで評価するために便宜上，コードの整形を依頼する指示を付与

コード生成プロンプト（1）

NameToCcode：ルーチン名のみを与えて C 言語コードを生成

Prompt

"**Implement #ROUTINE# routine in BLAS in C language.** *Function name must be "GPT-BLAS_#ROUTINE#". "GPTBLAS" is capitalized. All function argument names must be in lower case, all must be taken as pointers, and those that have not changed in the code must be marked with const. Insert printf("[gptblas]"); at the beginning of the routine. Do not output anything other than the source code. Do not output Markdown code block symbols either.*"

- ※ ROUTINE にはルーチン名（例：dgemm）が入る.

コード生成プロンプト（2）

NameToOptCcode：ルーチン名のみを与えて基礎的な性能最適化（並列化・SIMD・キャッシュブロッキング）を考慮した C 言語コードを生成

Prompt

"**Implement #ROUTINE# routine in BLAS in C language. Thread parallelization, SIMD vectorization, and cache blocking should be considered for speed-up.** *Function name must be "GPTBLAS_#ROUTINE#". "GPTBLAS" is capitalized. All function argument names must be in lower case, all must be taken as pointers, and those that have not changed in the code must be marked with const. Insert printf("[gptblas]"); at the beginning of the routine. Do not output anything other than the source code. Do not output Markdown code block symbols either.*"

- ※ ROUTINE にはルーチン名（例：dgemm）が入る.

コード生成プロンプト（3）（1/2）

FrtcodeToOptCcode：Fortran 参照実装コードを与えたうえで基礎的な性能最適化（同上）を考慮した C 言語コードを生成（参照実装コードには冒頭に仕様が英語で書かれている）

Prompt

"**Implement C code that has the same functionality as the attached Fortran code; the specifications are written at the beginning of the Fortran code. Thread parallelization, SIMD vectorization, and cache blocking should be considered for speed-up.** *Function name must be "GPTBLAS_#ROUTINE#". "GPTBLAS" is capitalized. All function argument names must be in lower case, all must be taken as pointers, and those that have not changed in the code must be marked with const. If you implement the XERBLA function, it must be named "xerbla" and the function body must not be implemented, but the prototype declaration "void xerbla(const char *srname, const int info);" must be added at the beginning of the code. Use macros named "MIN()" for "min()" and "MAX()" for "max()". Insert printf("[gptblas]"); at the beginning of the routine. Do not output anything other than the source code. Do not output Markdown code block symbols either.*"

- ※ ROUTINE にはルーチン名（例：dgemm）が入る.

コード生成プロンプト（3）（2/2）

dsymm ルーチンの Fortran 参照実装コード冒頭にある仕様の説明（一部）

```
*> \brief \b DSYMM
*  =========== DOCUMENTATION ===========
* Online html documentation available at
*            http://www.netlib.org/lapack/explore-html/
*  Definition:
*  ===========
*       SUBROUTINE DSYMM(SIDE,UPLO,M,N,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
*       .. Scalar Arguments ..
*       DOUBLE PRECISION ALPHA,BETA
*       INTEGER LDA,LDB,LDC,M,N
*       CHARACTER SIDE,UPLO
*       ..
*       .. Array Arguments ..
*       DOUBLE PRECISION A(LDA,*),B(LDB,*),C(LDC,*)
*       ..
*> \par Purpose:
*  =============
*> \verbatim
*> DSYMM  performs one of the matrix-matrix operations
*>    C := alpha*A*B + beta*C,
*> or
*>    C := alpha*B*A + beta*C,
*> where alpha and beta are scalars,  A is a symmetric matrix and  B and
*> C are  m by n matrices.
*> \endverbatim
*  Arguments:
*  ==========
*> \param[in] SIDE
*> \verbatim
*>          SIDE is CHARACTER*1
*>           On entry,  SIDE  specifies whether  the  symmetric matrix  A
*>           appears on the  left or right  in the  operation as follows:
*>              SIDE = 'L' or 'l'   C := alpha*A*B + beta*C,
*>              SIDE = 'R' or 'r'   C := alpha*B*A + beta*C,
```

生成コードの動作検証結果（10 コード中動作した数）（1/4）

Level	Routine	NameToCcode		NameToOptCcode		FrtcodeToOptCcode	
		GPT-4.1	o4-mini	GPT-4.1	o4-mini	GPT-4.1	o4-mini
1	dasum	10	10	9	7	8	8
	daxpy	10	10	10	10	8	9
	ddot	10	10	8	9	6	10
	idamax	3	10	2	9	7	7
	dnrm2	10	10	7	5	8	6
	drot	10	10	8	9	6	9
	drotm	8	5	3	6	8	8
2	dgemv	8	9	3	6	3	4
	dger	10	10	7	7	6	10
	dsymv	8	8	0	2	0	3
	dsyr	10	8	0	5	7	7
	dsyr2	8	9	2	7	6	10
	dtrmv	3	4	0	5	2	1
	dtrsv	8	9	0	4	4	7
3	dgemm	10	10	3	0	5	7
	dsymm	4	8	3	3	1	4
	dsyrk	3	10	0	1	4	5
	dsyr2k	3	10	0	0	7	6
	dtrmm	0	1	0	0	1	0
	dtrsm	0	0	0	0	5	1

生成コードの動作検証結果（2/4）

生成結果について

- ルーチン名のみを与えても正しく動作するコードが生成されている
- しかし必ずしも Fortran の参照実装コードと同じ実装方針（コード構造）ではない
  - dgemm では参照実装コードは全ての転置モード（transa/transb）の組合せを if 文で場合分けしそれぞれに 3 重 for ループを書いている
  - 生成コードの多くは最内側で転置モードに応じてアドレスを調整
  - 生成コードはなるべく短いコードで出力される傾向
- 仕様書やドキュメントを学習した結果に基づいてコード生成している可能性がある
  - 正しく動作しないケースが多くある
  - 生成コードは参照実装コードにあるエラー処理が省略されている

生成コードの動作検証結果（3/4）

dgemm の Fortran 参照実装コード（一部）

```
      IF (NOTB) THEN
          IF (NOTA) THEN
*           Form  C := alpha*A*B + beta*C.
              DO 90 J = 1,N (略)


                  DO 80 L = 1,K
                      TEMP = ALPHA*B(L,J)
                      DO 70 I = 1,M
                          C(I,J) = C(I,J) + TEMP*A(I,L
                              )
   70                 CONTINUE
   80                 CONTINUE
   90             CONTINUE
          ELSE
*           Form  C := alpha*A**T*B + beta*C
              DO 120 J = 1,N
                  DO 110 I = 1,M
                      TEMP = ZERO
                      DO 100 L = 1,K
                          TEMP = TEMP + A(L,I)*B(L,J)
  100                 CONTINUE
                      IF (BETA.EQ.ZERO) THEN
                          C(I,J) = ALPHA*TEMP
                      ELSE
                          C(I,J) = ALPHA*TEMP + BETA*C
                              (I,J)
                      END IF
  110             CONTINUE
  120         CONTINUE
          END IF
      ELSE
          IF (NOTA) THEN
```

dgemm の生成された C 言語コード（例）

```
#include <stdio.h>

void GPTBLAS_dgemm(const char *transa, const char *
     transb, const int *m, const int *n, const int *k,
     const double *alpha, const double *a, const int *
     lda, const double *b, const int *ldb, const double
      *beta, double *c, const int *ldc)
{
    printf("[gptblas]");
    int i, j, l;
    int nota = (*transa == 'N' || *transa == 'n');
    int notb = (*transb == 'N' || *transb == 'n');
    for (j = 0; j < *n; j++) {
        for (i = 0; i < *m; i++) {
            double cij = (*beta) * c[i + j * (*ldc)];
            double sum = 0.0;
            for (l = 0; l < *k; l++) {
                double a_val = nota ? a[i + l * (*lda)
                    ] : a[l + i * (*lda)];
                double b_val = notb ? b[l + j * (*ldb)
                    ] : b[j + l * (*ldb)];
                sum += a_val * b_val;
            }
            c[i + j * (*ldc)] = cij + (*alpha) * sum;
        }
    }
}
```

生成コードの動作検証結果（4/4）

全体的な傾向

- o4-mini は全体的に GPT-4.1 よりも正当性が高い傾向
  - o4-mini は NameToCcode（ルーチン名のみから C コード生成）は dtrsm を除き全ルーチン成功
- Level-3 BLAS ルーチンは成功率が低い
  - パラメータを考慮した実行パターン数が多い
  - ブロッキングを指示すると構造が複雑になる

性能最適化

- OpenMP が適用されているが必ずしも最適な戦略ではない
- SIMD 化は多くの場合 pragma omp simd が使用されるが，AVX2/AVX-512 組み込み関数を使用している場合もある
  - その場合，マクロ（__AVX__，__AVX512F__）でコード分岐されている
- 行列計算のブロックサイズは多くの場合 64 が適当に設定されている（一部 128 や 256）

性能最適化されたコードの性能評価

計算環境

- 名古屋大学「不老」Type II システム
- CPU：Intel Xeon Gold 6230（20 コア，Cascade Lake） × 2
- スレッド数は物理コアに合わせて 40 スレッドを指定
- コンパイルは gcc/gfortran 11.3.0，-march=native のみ指定

評価方法

- Level-1/2 ルーチン（メモリバンド幅律速）：GB/s で評価
- Level-3 ルーチン（演算律速）：GFlops/s で評価
- 問題サイズ：キャッシュサイズより大きく設定
  - level-1：$n = 16777216$，level-2：$m = n = 8192$，level-3：$m = n = k = 2048$
- Fortran の参照実装（並列化・最適化されていない）と性能を比較
- 生成された 10 コード中で最良の結果を示す

Level-1 BLAS ルーチンの性能（GB/s, n=16777216）

		NameToOptCcode				FrtcodeToOptCcode			
Routine	Ref	GPT-4.1		o4-mini		GPT-4.1		o4-mini	
dasum	6.0	68.0	(11.4x)	63.5	(10.7x)	61.5	(10.3x)	63.6	(10.7x)
daxpy	17.2	77.6	(4.5x)	68.8	(4.0x)	71.9	(4.2x)	67.8	(3.9x)
ddot	11.7	65.8	(5.6x)	63.1	(5.4x)	61.0	(5.2x)	60.8	(5.2x)
idamax	7.1	63.8	(9.0x)	65.1	(9.2x)	60.3	(8.6x)	62.8	(8.9x)
dnrm2	5.1	66.5	(12.9x)	64.0	(12.5x)	60.6	(11.8x)	63.5	(12.4x)
drot	10.6	40.6	(3.8x)	34.1	(3.2x)	34.9	(3.3x)	34.4	(3.3x)
drotm	11.1	39.6	(3.6x)	36.6	(3.3x)	34.7	(3.1x)	34.1	(3.1x)

- “Ref” は Fortran 参照実装（並列化・最適化されていない）
- 必ずしも 1 つのコードで実行した結果ではなく，各パラメータ設定で動作した場合の最良値

Level-2 BLAS ルーチンの性能（GB/s, m=n=8129）（1/2）

Routine	Parameters	Ref	NameToOptCcode				FrtcodeToOptCcode			
			GPT-4.1		o4-mini		GPT-4.1		o4-mini	
dgemv	trans=N	9.1	5.4	(0.6x)	30.0	(3.3x)	32.3	(3.5x)	6.6	(0.7x)
	trans=T	6.7	68.9	(10.4x)	66.3	(10.0x)	67.6	(10.2x)	65.9	(9.9x)
dger		18.5	46.3	(2.5x)	64.9	(3.5x)	66.4	(3.6x)	64.2	(3.5x)
dsymv	uplo=L	6.1			4.6	(0.8x)			4.3	(0.7x)
	uplo=U	6.1			5.3	(0.9x)			4.7	(0.8x)
dsyr	uplo=L	17.4	32.4	(1.9x)	64.0	(3.7x)	64.5	(3.7x)	60.9	(3.5x)
	uplo=U	17.5			61.5	(3.5x)	64.4	(3.7x)	58.9	(3.4x)
dsyr2	uplo=L	7.6	26.4	(3.5x)	63.7	(8.3x)	61.0	(8.0x)	60.2	(7.9x)
	uplo=U	15.5	29.0	(1.9x)	59.6	(3.9x)	62.1	(4.0x)	58.6	(3.8x)
dtrmv	uplo=L, trans=N, diag=N	7.6			7.3	(1.0x)	3.0	(0.4x)	3.0	(0.4x)
	uplo=L, trans=N, diag=U	7.5			7.7	(1.0x)	2.9	(0.4x)	2.9	(0.4x)
	uplo=L, trans=T, diag=N	6.5			56.9	(8.8x)	3.3	(0.5x)	3.2	(0.5x)
	uplo=L, trans=T, diag=U	6.5			56.3	(8.7x)	3.3	(0.5x)	3.2	(0.5x)
	uplo=U, trans=N, diag=N	9.0			7.6	(0.8x)	3.1	(0.3x)	3.1	(0.3x)
	uplo=U, trans=N, diag=U	8.8			7.6	(0.9x)	3.1	(0.3x)	3.1	(0.4x)
	uplo=U, trans=T, diag=N	6.1			56.7	(9.3x)	3.2	(0.5x)	3.2	(0.5x)
	uplo=U, trans=T, diag=U	6.1			56.4	(9.3x)	3.2	(0.5x)	3.2	(0.5x)

- “Ref” は Fortran 参照実装（並列化・最適化されていない）
- 必ずしも 1 つのコードで実行した結果ではなく，各パラメータ設定で動作した場合の最良値
- 空欄は動作するコードが生成されなかったことを示す

Level-2 BLAS ルーチンの性能（GB/s, m=n=8129）（2/2）

Routine	Parameters	Ref	NameToOptCcode				FrtcodeToOptCcode			
			GPT-4.1		o4-mini		GPT-4.1		o4-mini	
dtrsv	uplo=L, trans=N, diag=N	8.4	12.0	(1.4x)	17.5	(2.1x)	4.2	(0.5x)	3.5	(0.4x)
	uplo=L, trans=N, diag=U	8.5	12.2	(1.4x)	15.4	(1.8x)	4.4	(0.5x)	3.4	(0.4x)
	uplo=L, trans=T, diag=N	6.2			26.6	(4.3x)	3.1	(0.5x)	3.1	(0.5x)
	uplo=L, trans=T, diag=U	6.2			26.8	(4.3x)	3.1	(0.5x)	3.1	(0.5x)
	uplo=U, trans=N, diag=N	7.7	11.7	(1.5x)	15.1	(2.0x)	4.3	(0.6x)	3.0	(0.4x)
	uplo=U, trans=N, diag=U	7.8	12.4	(1.6x)	15.1	(1.9x)	4.4	(0.6x)	3.1	(0.4x)
	uplo=U, trans=T, diag=N	6.5			27.5	(4.2x)	3.3	(0.5x)	3.3	(0.5x)
	uplo=U, trans=T, diag=U	6.5			27.1	(4.1x)	3.2	(0.5x)	3.3	(0.5x)

- “Ref” は Fortran 参照実装（並列化・最適化されていない）
- 必ずしも 1 つのコードで実行した結果ではなく，各パラメータ設定で動作した場合の最良値
- 空欄は動作するコードが生成されなかったことを示す

Level-3 BLAS ルーチンの性能（GFlops/s, m=n=k=2048）(1/3)

Routine	Parameters	Ref	NameToOptCcode				FrtcodeToOptCcode			
			GPT-4.1		o4-mini		GPT-4.1		o4-mini	
dgemm	transa=N, transb=N	2.7	17.4	(6.5x)	20.5	(7.7x)	18.0	(6.8x)	20.8	(7.8x)
	transa=N, transb=T	2.5	16.5	(6.5x)			16.1	(6.3x)	20.7	(8.1x)
	transa=T, transb=N	1.8	16.8	(9.3x)	0.8	(0.5x)	29.2	(16.1x)	23.8	(13.1x)
	transa=T, transb=T	0.3	15.7	(46.3x)			3.5	(10.4x)	20.3	(59.8x)
dsymm	side=L, uplo=L	3.6	13.6	(3.7x)	17.2	(4.7x)	19.9	(5.5x)	22.3	(6.2x)
	side=L, uplo=U	3.5	13.9	(4.0x)	17.3	(4.9x)	19.9	(5.7x)	22.4	(6.4x)
	side=R, uplo=L	2.7	13.4	(4.9x)	16.4	(6.0x)	19.0	(7.0x)	21.2	(7.8x)
	side=R, uplo=U	2.7	13.4	(4.9x)	21.3	(7.8x)	18.7	(6.8x)	21.1	(7.7x)
dsyrk	uplo=L, trans=N	1.0			2.9	(2.9x)	18.2	(18.2x)	16.6	(16.6x)
	uplo=L, trans=T	1.9	20.1	(10.6x)	21.8	(11.5x)	27.7	(14.7x)	23.0	(12.2x)
	uplo=U, trans=N	2.3			3.8	(1.6x)	16.5	(7.1x)	16.5	(7.1x)
	uplo=U, trans=T	1.9	36.5	(19.2x)	21.0	(11.0x)	34.7	(18.2x)	22.5	(11.9x)
dsyr2k	uplo=L, trans=N	1.8					31.0	(16.8x)	18.4	(10.0x)
	uplo=L, trans=T	3.1	42.9	(13.8x)			29.6	(9.5x)	23.1	(7.4x)
	uplo=U, trans=N	3.1					30.2	(9.9x)	18.1	(5.9x)
	uplo=U, trans=T	3.1	43.2	(13.7x)			27.4	(8.7x)	23.0	(7.3x)

- “Ref” は Fortran 参照実装（並列化・最適化されていない）
- 必ずしも 1 つのコードで実行した結果ではなく，各パラメータ設定で動作した場合の最良値
- 空欄は動作するコードが生成されなかったことを示す

Level-3 BLAS ルーチンの性能（GFlops/s, m=n=k=2048）（2/3）

Routine	Parameters	Ref	NameToOptCcode		FrtcodeToOptCcode			
			GPT-4.1	o4-mini	GPT-4.1		o4-mini	
dtrmm	side=L, uplo=L, trans=N, diag=N	3.1		1.6 (0.5x)	22.4	(7.4x)		
	side=L, uplo=L, trans=N, diag=U	3.1		1.3 (0.4x)	22.9	(7.4x)	23.0	(7.4x)
	side=L, uplo=L, trans=T, diag=N	1.8			28.7	(15.8x)	23.6	(13.0x)
	side=L, uplo=L, trans=T, diag=U	1.8			28.8	(15.7x)	23.7	(12.9x)
	side=L, uplo=U, trans=N, diag=N	3.0		1.7 (0.6x)	22.6	(7.5x)	22.4	(7.5x)
	side=L, uplo=U, trans=N, diag=U	3.0		1.7 (0.6x)	22.6	(7.6x)	22.2	(7.5x)
	side=L, uplo=U, trans=T, diag=N	1.1			27.7	(26.4x)	23.3	(22.2x)
	side=L, uplo=U, trans=T, diag=U	1.1			28.3	(26.7x)	23.6	(22.2x)
	side=R, uplo=L, trans=N, diag=N	3.0			0.3	(0.1x)	0.3	(0.1x)
	side=R, uplo=L, trans=N, diag=U	3.1			0.3	(0.1x)	0.3	(0.1x)
	side=R, uplo=L, trans=T, diag=N	3.4			0.3	(0.1x)	0.3	(0.1x)
	side=R, uplo=L, trans=T, diag=U	3.5			0.3	(0.1x)	0.3	(0.1x)
	side=R, uplo=U, trans=N, diag=N	3.3			0.3	(0.1x)	0.3	(0.1x)
	side=R, uplo=U, trans=N, diag=U	3.2			0.3	(0.1x)	0.3	(0.1x)
	side=R, uplo=U, trans=T, diag=N	3.3			0.3	(0.1x)	0.3	(0.1x)
	side=R, uplo=U, trans=T, diag=U	3.3			0.3	(0.1x)	0.3	(0.1x)

- “Ref” は Fortran 参照実装（並列化・最適化されていない）
- 必ずしも 1 つのコードで実行した結果ではなく，各パラメータ設定で動作した場合の最良値
- 空欄は動作するコードが生成されなかったことを示す

Level-3 BLAS ルーチンの性能（GFlops/s, n=2048）(3/3)

Routine	Parameters	Ref	NameToOptCcode			FrtcodeToOptCcode			
			GPT-4.1	o4-mini		GPT-4.1		o4-mini	
dtrsm	side=L, uplo=L, trans=N, diag=N	1.4		5.0	(3.5x)	20.8	(14.6x)	24.0	(16.9x)
	side=L, uplo=L, trans=N, diag=U	1.4		5.4	(3.7x)	20.9	(14.6x)	24.7	(17.2x)
	side=L, uplo=L, trans=T, diag=N	1.9		16.5	(8.8x)	23.6	(12.6x)	23.5	(12.6x)
	side=L, uplo=L, trans=T, diag=U	1.9		18.0	(9.6x)	23.6	(12.5x)	23.7	(12.6x)
	side=L, uplo=U, trans=N, diag=N	0.9		6.9	(8.0x)	20.7	(23.8x)	24.4	(28.0x)
	side=L, uplo=U, trans=N, diag=U	0.9		4.4	(5.0x)	20.9	(24.0x)	24.5	(28.1x)
	side=L, uplo=U, trans=T, diag=N	1.9		17.1	(9.2x)	23.4	(12.6x)	23.5	(12.7x)
	side=L, uplo=U, trans=T, diag=U	1.9		16.5	(8.9x)	23.4	(12.6x)	23.5	(12.7x)
	side=R, uplo=L, trans=N, diag=N	3.3				9.9	(3.0x)	0.3	(0.1x)
	side=R, uplo=L, trans=N, diag=U	3.2				9.1	(2.8x)	0.3	(0.1x)
	side=R, uplo=L, trans=T, diag=N	2.8				6.3	(2.3x)	20.5	(7.4x)
	side=R, uplo=L, trans=T, diag=U	2.8				5.8	(2.1x)	18.0	(6.5x)
	side=R, uplo=U, trans=N, diag=N	3.3				8.8	(2.7x)	0.3	(0.1x)
	side=R, uplo=U, trans=N, diag=U	3.3				9.9	(3.0x)	0.3	(0.1x)
	side=R, uplo=U, trans=T, diag=N	2.8				6.3	(2.3x)	20.2	(7.3x)
	side=R, uplo=U, trans=T, diag=U	2.8				6.5	(2.4x)	18.9	(6.8x)

- “Ref” は Fortran 参照実装（並列化・最適化されていない）
- 必ずしも 1 つのコードで実行した結果ではなく，各パラメータ設定で動作した場合の最良値
- 空欄は動作するコードが生成されなかったことを示す

まとめ

結論

- ルーチン名のみを与えても正しく動作するコードが生成されている
  - 生成されたコードは Fortran 参照実装と構造が大きく異なる場合がある
  - 仕様書やドキュメントを学習した結果に基づいてコード生成している可能性がある
- 標準的な汎用 LLM でも BLAS レベルのコードは十分生成可能
- やや複雑な計算（level-2/3 ルーチン）の性能最適化には課題が多い

今後の課題

- 生成能力向上のためのアイディア：試行回数の増加，プロンプトの工夫，反復プロンプト，追加学習，RAG，マルチエージェント，etc.
- オープンウェイト LLM の性能評価 – ローカル LLM コード生成システムの開発へ
- 数値計算コード生成向けベンチマークセットの構築（BLAS はドキュメントやコードがネットにあり性能評価にやや不適切）

Anthropic Claude Code

Claude Code による dgemm（倍精度行列積）の実装

- Claude Code v1.0.60, Opus 4（Max プラン課金しており制限はきていない）
- CPU: Intel Core i5-14400F (Raptor Lake) – 225.6 + 112.0 = 337.6 GFLOPS
  - Performance-cores: 4.7 GHz × 4 (AVX2) × 2 (FMA) × 6 コア = 225.6 GFLOPS
  - Efficient-cores: 3.5 GHz × 4 (AVX2) × 2 (FMA) × 4 コア = 112.0 GFLOPS
  - 16 スレッド（P コア 2 スレッド/コア，E コア 1 スレッド/コア）
- OpenBLAS 0.3.30 と性能比較

入力したプロンプト

1. 「C 言語で BLAS の dgemm ルーチンを実装してください．CPU 情報を cat /proc/cpuinfo で確認し，この環境で最大性能が発揮できるように性能最適化を行ってください．OpenBLAS（~/local にインストールされている）と計算結果を比較し正しく動作することを確認してください.」
2. 「さらに最適化してください.」
3. 「問題サイズを 4096 まで，10 回測定して最速値を表示するようにしてください.」

Claude Code – 生成コードの性能

```
# Version 1
N= 128: Optimized:  24.81 GFLOPS, OpenBLAS: 140.74 GFLOPS, Efficiency:  17.6%, Error: 8.881784e-16
N= 256: Optimized:  36.09 GFLOPS, OpenBLAS: 231.48 GFLOPS, Efficiency:  15.6%, Error: 3.552714e-15
N= 512: Optimized:  37.43 GFLOPS, OpenBLAS: 289.88 GFLOPS, Efficiency:  12.9%, Error: 1.065814e-14
N= 768: Optimized:  58.06 GFLOPS, OpenBLAS: 291.67 GFLOPS, Efficiency:  19.9%, Error: 1.398881e-14
N=1024: Optimized:  55.44 GFLOPS, OpenBLAS: 310.42 GFLOPS, Efficiency:  17.9%, Error: 1.598721e-14
N=1536: Optimized:  52.42 GFLOPS, OpenBLAS: 313.46 GFLOPS, Efficiency:  16.7%, Error: 1.776357e-14
N=2048: Optimized:  51.14 GFLOPS, OpenBLAS: 315.77 GFLOPS, Efficiency:  16.2%, Error: 2.309264e-14
N=3072: Optimized:  51.67 GFLOPS, OpenBLAS: 323.50 GFLOPS, Efficiency:  16.0%, Error: 2.553513e-14
N=4096: Optimized:  48.36 GFLOPS, OpenBLAS: 329.53 GFLOPS, Efficiency:  14.7%, Error: 3.108624e-14
```

↓

「さらに最適化してください」

↓

```
# Version 2
N= 128: Optimized:  20.36 GFLOPS, OpenBLAS: 139.62 GFLOPS, Efficiency:  14.6%, Error: 8.881784e-16
N= 256: Optimized:  54.05 GFLOPS, OpenBLAS: 229.96 GFLOPS, Efficiency:  23.5%, Error: 3.552714e-15
N= 512: Optimized:  71.59 GFLOPS, OpenBLAS: 287.73 GFLOPS, Efficiency:  24.9%, Error: 5.329071e-15
N= 768: Optimized: 180.22 GFLOPS, OpenBLAS: 290.67 GFLOPS, Efficiency:  62.0%, Error: 0.000000e+00
N=1024: Optimized: 203.34 GFLOPS, OpenBLAS: 305.95 GFLOPS, Efficiency:  66.5%, Error: 7.771561e-15
N=1536: Optimized: 286.05 GFLOPS, OpenBLAS: 314.14 GFLOPS, Efficiency:  91.1%, Error: 0.000000e+00
N=2048: Optimized: 287.40 GFLOPS, OpenBLAS: 319.22 GFLOPS, Efficiency:  90.0%, Error: 1.154632e-14
N=3072: Optimized: 310.42 GFLOPS, OpenBLAS: 323.78 GFLOPS, Efficiency:  95.9%, Error: 0.000000e+00
N=4096: Optimized: 317.58 GFLOPS, OpenBLAS: 330.58 GFLOPS, Efficiency:  96.1%, Error: 1.509903e-14
```