Expanding Antenna Farm in NEC2++

Yoshiyuki Takeyasu / JA6XKQ

NEC2++ アンテナファームの拡張

武安義幸 / JA6XKQ

NEC2++ [1] の数値演算ライブラリを ATLAS [2] から OpenBLAS [3] に置換して、マルチコア・プロセッサ対応による高速化を図った。[4] また、プロセッサのコア数を 4 コアから 8 コアへ増やして計算速度の高速化を図った。[5] 道具である NEC2++ のこれらの手入れを一旦離れ、目的であるアンテナ最適化設計に着手したが、道具に気になる不備が発見された。シミュレーションとしては、大規模モデルを高速に計算したい訳であるが、NEC2++ が規模と速度の観点でコンピュータの能力を最大限に生かしきれていないので改善を図った。その改善について備忘録として本稿にまとめる。

モデル規模の制限

NEC2++ を使い始めた当初 [6] 、「C++ 言語の特長を生かして、計算モデルのセグメント数の制限を取払っている」と記した。これは、NEC2++ のマニュアルには明記されていないが、プログラムの作者 Timothy Molteno が NEC-LIST メーリングリストで

> *The ports of nec-2 to C and C++ both use dynamic memory allocation and so have no built-in limit on segment number (OK, on nec2++ it is 4 billion segments).*

との発言 [7] を受けてのことであった。ホーン・アンテナのシミュレーションの過程で Surface Patch のセグメントの細かさが計算結果に与える影響を評価した際、セグメントを細かくしてモデル規模が大きくなると NEC2++ プログラムがエラー（ Memory Segmentation Fault ） で停止する不具合に遭遇した。作者の意図に反して、モデル規模に制限を生じるバグを含んでいるようである。

3 セクション・ホーン・アンテナ [6] でのセグメントの細かさと Surface Patch 総数、そして、エラー発生の関係は下表のようになる。

1/8 波長以上のセグメントの細かさでエラーを生じている。コンピュータに実装しているメモリーは 144 GB であり、1/7 波長までの状況から類推すると、メモリー不足とは考えられない。なお、エラーを生じているケースでの必要メモリー・サイズはバグを解消した後に判明した値である。当初はセグメントの細かさとは無関係に、計算過程のメモリー使用量が約 28.4 GB に達するとエラーを生じていたため、計算に必要とする最終的なメモリー・サイズは不明であった。

Size of Segment (λ)	No. of Surface-patch	Memory Segmentation Fault	Required Memory Size (GB)
1/4	6,319	None	2.5
1/5	9,968	None	6.0
1/6	13,992	None	11.8
1/7	19,096	None	21.9
1/8	24,957	Error	37.3
1/9	31,641	Error	59.9
1/10	39,117	Error	91.5

デバッグ

NEC2++ と OpneBLAS について、デバッグ・オプション -g を付けて gcc でコンパイルし、gdb + Eclipse でデバッグを行なった。

まず判明したことは、OpenBLAS による演算を開始する以前のマトリクスを準備する段階でエラーを生じていることであった。NEC2++ の matrix_algebra.cpp においてマトリクスを準備する際、NEC2++ は、メモリー操作に関するエラーを自前の safe_array でチェックしており、そこでエラー（ Memory Segmentation Fault ）を検出して、プログラムを NEC2++ 自身で終了していた。プログラムがクラッシュしたのではなく、想定される異常に対して行儀良く「正常」終了している訳である。

さらにデバッグを進めると、マトリクスのために必要なメモリ領域を計算した結果、領域サイズが「負」という矛盾を生じていることが判明した。領域サイズはマトリクスの「行」数と「列」数に依存し、それらの積に依存することは容易に理解できる。プログラムの計算式を見て電卓で検算しても、掛け算の結果が「負」になることはない。あるサイズのマトリクスまでは正常であるので、計算は論理的には正しいと判断される。では、数値の大きさに依存する観点は何か？と問えば、変数の「型」が問題になることに考えが至る。件の計算式の変数の型を確認すると int 型であり、int 型同士の掛け算結果が int 型を越えるものとなっていた。

関連する変数の型を int から long long int に変更することで、変数の掛け算結果が型の範囲を（実用領域で）越えないようにした。関連する変数を含むプログラムは matrix_algebra.cpp の他に、c_geomery.cpp、nec_context.cpp とそれらのヘッダー・ファイルである matrix_algebra.h、nex_context.h である。（詳細は付録）

デバッグの結果、1/8 波長以上のセグメントの細かさでも Memory Segmentation Fault のエラーを生じることなく、144 GB のメモリ領域を最大で約 92 GB まで使用して計算を実行できるようになった。

OpenBLAS の制限

NEC2++ のメモリ操作のバグが解消されてメモリ・リソースを最大限に利用できるようになり、従来は計算できなかった 1/8 波長刻みのモデル、1/9 波長刻みのモデルとシミュレーションを進めると、1/10 波長刻みのモデルにおいて再び Segmentation Fault を生じた。この Segmentation Fault は、OpenBLAS によるマルチコアでの演算が始まった時点で発生している。

gdb + Eclipse でトレースすると、8 個のマルチコアへ演算を割振る過程でエラーを生じていることが判明した。マルチスレッドに関するデバッグであり、高度に最適化された OpenBLAS のデバッグは自分の能力を越えているので自力でのデバッグは早々に諦めた。OpenBLAS の開発リポジトリである Github [8] でバグ情報を検索したが、該当する情報は得られなかった。しかし、Segmentation Fault の問題は各所のルーチンで発生しては対策が講じられているようである。

OpenBLAS では使用するコア数を環境変数 OPENBLAS_NUM_THREADS=N （N はコア数） にて指定することができ、8 コアから使用コア数を順次減らして Segmentation Fault の発生具合を試してみると、シングル・コア（N = 1）ではエラーを生じないことが確認できた。しかし、シングル・コアで 1/10 波長刻みのモデルを計算したのでは 30 時間以上を要すると予想されるので実用的ではない。

OpenBLAS では 8 コアのリソースを活かせないので、別の数値演算ライブラリへ移行することとした。

Intel Math Kernel Library (MKL) への移行

ATLAS から OpenBLAS と来たら、残るは本家 Intel の Math Kernel Library (MKL) [9] しかないだろう。Intel MKL を使うにあたって確認した事項は、

- NEC2++ が使用する zgetrf は行列の因子分解用のルーチン、zgetrs は連立一次方程式を解くためのルーチンで、LAPACK に含まれる
- LAPACK は FORTRAN77 で記述されている
- C/C++ から LAPACK へのインターフェースは mkl_lapacke.h と mkl_cblas.h を通じて行なう
- 関数名は、LAPACKE_zgetrf と LAPACKE_zgetrs
- インクルードすべきライブラリ名とコンパイラへのオプション指定は、Intel Math Kernel Librar Link Line Advisor で得ることができる [10]

確認事項に従って、まず NEC2++ の matrix_algebra.cpp を修整する。修正内容は、

- インクルード・ファイルを指定する
  ```
  #include </opt/intel/composer_xe_201.1.117/mkl/include/mkl_lapacke.h>
  #include </opt/intel/composer_xe_201.1.117/mkl/include/mkl_cblas.h>
  ```
- 関数を指定する
  ```
  int info = LAPACKE_zgetrf (CblasColMajor, n, n, (MKL_Complex16*) a_in.data(), ndim, (int*) ip.data());
  int info = LAPACKE_zgetrs (CblasColMajor, 'N', n, 1, (const MKL_Complex16*) a.data(), ndim, (const int*) ip.data(), (MKL_Complex16*) b.data(), n);
  ```


(C) 2014, Yoshiyuki Takeyasu / JA6XKQ

次に、Intel Math Kernel Librar Link Line Advisor でインクルードすべきライブラリとコンパイラへのオプション指定を決定した。 Advisor への入力事項は、

- Intel (R) product : Intel(R) MKL 11.1
- OS : Linux
- Usage model of Intel (R) Xeon Phi (TM) Coprocessor : None
- Complier : Intel (R) C/C++
- Architecture : Intel (R) 64
- Dynamic or static linking : Static
- Interface layer : LP64 (3-bit integer)
- Sequential or multi-threaded layer : Multi-threaded
- OpenMP library : Intel (R) (libiomp5)

これらを指定した結果、コンパイル時のリンクとオプションは、

- -wl, ––start-group $(MKLROOT)/lib/intel64/libmkl_intel_lp64.a $(MKLROOT)/lib/intel64/libmkl_core.a $(MKLROOT)/lib/intel64/libmkl_intel_thread.a -wl, ––end-group lpthread -lm
- -openmp -I$(MKLROOT)/include

とのアドバイスを得た。アドバイスの他に、コンパイルには最適化オプションの -fast を付与した。

Intel Math Kernel Library (MKL) の威力

コンパイルで得られた実行プログラムで 1/10 波長刻みのモデルを試すと、期待どおりに 8 コア全てを使ってシミュレーションを完了することができた。これまでの NEC2++ の計算可能なモデル規模と計算速度の改善過程を下表にまとめる。

Size of Segment (λ)	No. of Surface-patch	3.16 GHz Core2 Duo ATLAS (sec)	3.0 GHz Core2 Quad OpenBLAS (sec)	2.93 GHz Dual X5570 OpenBLAS (sec)	2.93 GHz Dual X5570 Intel MKL (sec)
1/4	6,319	360	135	68	67
1/5	9,958	1,380	510	253	247
1/6	13,992	3,600	1,360	669	663
1/7	19,096	—	—	1,671	1,663
1/8	24,957	—	—	3,760	3,659
1/9	31,641	—	—	7,633	7,417
1/10	39,117	—	—	—	14,004

セグメント・サイズとシミュレーション結果

シミュレーション・モデルのセグメント・サイズを 1/4 から 1/10 波長刻みまで計算できるようになったので、セグメント・サイズがシミュレーションに与える影響 [6] を改めて確認してみる。

3 セクション・コニカル・ホーン・アンテナ [6] の 12.5 GHz での輻射パターンを 図-1 に示す。メイン・ローブとバック・ローブに対するセグメント・サイズの影響は小さいが、サイド・ローブについてはその影響は大きい。

これと同様な論点は、W1GHZ が 1998～1999 年に NEC2 でホーン・アンテナをシミュレーションした際に言及している。 [11] W2IMU＋Potter ホーンである VK2ALU の設計例（3 セクション・コニカル・ホーン・アンテナに相当）を NEC2 でシミュレーションするのは、当時の PC のメモリ容量では限界であった。解析解を用いる（のでメモリ使用量が少なくて済む）PO (Physical Optics) [12] で計算してみると、実測との相異点がみられた。そこで、メモリを 128 Mbyte (!!) まで増やして NEC2 で計算してみると、サイドとバック・ローブが実測よりも高い値となった、とある。理由として、

Even with the larger memory, I was unable to make a fine enough mode, so

(C) 2014, Yoshiyuki Takeyasu / JA6XKQ

the calculated radiation patterns have excessive side and back lobes.

と記述している。ホーン内の電磁界のモードを十分に高次までシミュレーションできないことが、サイドとバック・ローブを（見かけ上）高くしている理由である。

「ああ、私は 144 GB のメモリを突っ込んで、ようやくそのレベルに到達したのか」と先達に追いついて嬉しいのか、今更の話で悲しいのか、道具（ CPU 速度とメモリ容量）に頼ってしまい情けないのか判らない心境である。

セグメント・サイズの細かさと「ホーン内の電磁界のモードを高精度にシミュレーションする」という観点を今一度考えてみると、セグメントの刻みが大きいということは、ホーンの断面を円形ではなく多角形で表現しているのであって、シミュレーション・モデルとして高精度か否かという観点とは異なるのではないだろうか？ 例えば、1/4 波長刻みのモデルと同じ多角形の断面を二倍の細かさ、つまり 1/8 波長刻みでシミュレーションすると **図-1** と同じ結果となるだろうか？ シミュレーション・モデルの精度の観点では、NEC2 のマニュアルに記載されているように、1/5 波長刻みが一つの目安であって [12] 、それ以上の刻みは多角形と円形での電磁界モードの違いを精度良くシミュレーションしていると言えるかもしれない。

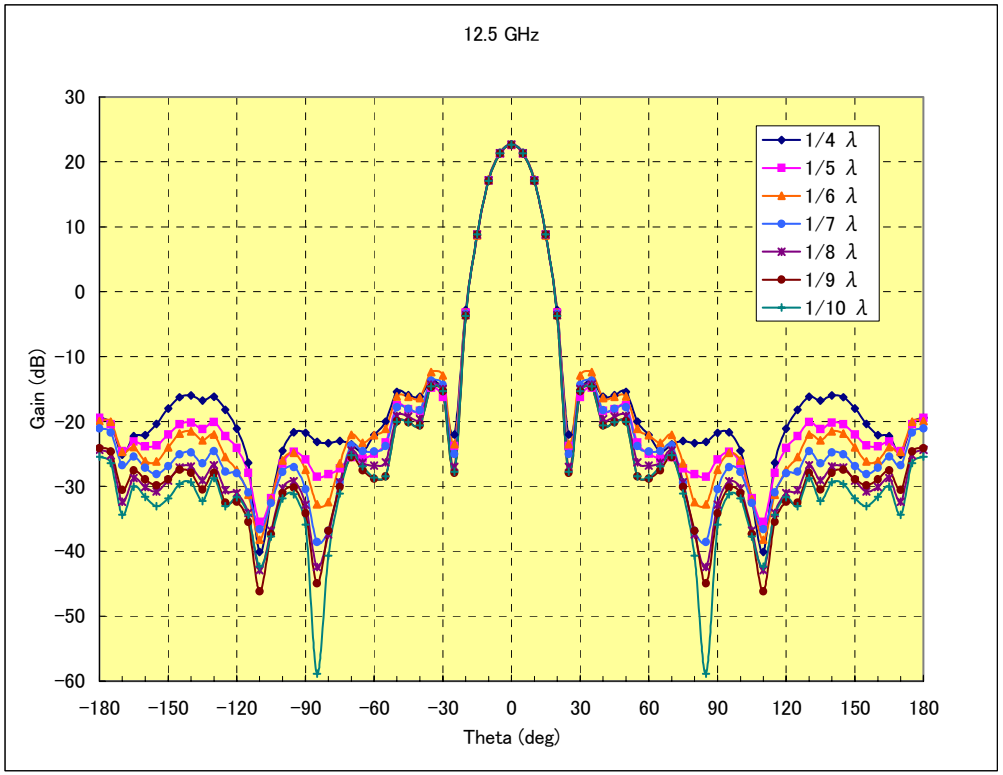

図-1 ： セグメント・サイズと輻射パターンの関係

むしろ、ここで着目すべきは、多角形の断面で発生する電磁界モードであって、それを設計に積極的に取り入れることを示唆しているように思う。マルチ・セクションのコニカル・ホーンは、円錐形の広がり具合と長さにより電磁界モードを制御することで所望のアンテナ特性を実現している。円錐形を多角錐とすることで設計パラメータの自由度を増やすことができ、より小型で高性能なホーン・アンテナが可能とならないだろうか？

まとめ

NEC2++ のメモリ領域操作に関わるバグを修正することでシミュレーションのモデル規模の制限を取除くことができた。モデル規模の制限が取除かれた一方で、数値演算ライブラリ OpenBLAS のマルチ・コア動作に、モデル規模に依存した動作制限が発現した。このマルチ・コア動作での制限を取除くために、数値演算ライブラリを Intel MKL へ置換した。

これらの改善の結果、コンピュータの CPU とメモリのリソースを有効活用して、NEC2++ の初期の目的である大規模モデルを高速にシミュレーションすることが可能となり、NEC2++ の中に広がるアンテナファームを拡張することができた。

大規模モデルを高速にシミュレーションが可能となった現在、円錐ホーン以外に多角錐ホーン、あるいは円形の一部に突起を設けた（＝調整個所）ホーンも設計テーマとして興味深い。

／／
☆

2014年5月26日 初版
2014年5月27日 第二版 一覧表に単位を追記

(C) 2014, Yoshiyuki Takeyasu / JA6XKQ

参考文献

[1] NEC2++
Timothy Molteno
http://elec.otago.ac.nz/w/index.php/Necpp

[2] Automatically Tuned Linear Algebra Software (ATLAS)
http://math-atlas.sourceforge.net/

[3] OpenBLAS
http://xianyi.github.com/OpenBLAS/

[4] Speeding Up NEC2++ by OpenBLAS
OpenBLAS による NEC2++ の高速化
武安義幸、JA6XKQ
http://www.terra.dti.ne.jp/~takeyasu/Nec2ppOpenBlas_1.pdf

[5] Speeding Up NEC2++ by Multi-core Processor
マルチコア・プロセッサによる NEC2++ の高速化
武安義幸、JA6XKQ
http://www.terra.dti.ne.jp/~takeyasu/Nec2ppMultiCore_1.pdf

[6] Simulation of a horn antenna using NEC2++
NEC2++ によるホーン・アンテナのシミュレーション
武安義幸、JA6XKQ
http://www.terra.dti.ne.jp/~takeyasu/Nec2pp3SecHorn.pdf

[7] Re: NEC-LIST: Fortran execution speed
Tim Molteno
http://www.devilsfoot.com/Nec/Archive2008/0044.html

[8] GitHub
xianyi / OpenBLAS
https://github.com/xianyi/OpenBLAS

[9] Intel Math Kernel Library
Intel Corporation
https://software.intel.com/en-us/intel-mkl

[10] Intel Math Kernel Librar Link Line Advisor
Intel Corporation
https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor

[11] Chapter 6
Feeds for Parabolic Dish Antennas
Paul Wade, W1GHZ
1998, 1999
http://www.w1ghz.org/antbook/ch6_5-2.pdf

[12] Antenna Engineering Using Phsical Optics : Practical CAD Techniques and Software
Leo Diaz, Thomas Milligan
Artech House Publishers, 01 October 1996
ISBN: 0890067325

付録

関連する変数の型を int から long long int に変更することで、変数の掛け算結果が型の範囲を（実用領域で）越えないようにした。関連する変数を含むプログラムは matrix_algebra.cpp の他に、c_geomery.cpp、nec_context.cpp とそれらのヘッダー・ファイルである matrix_algebra.h、nex_context.h である。

型の変更を要する部分を列挙する。

matrix_algebra.cpp

```
long long int row_offset = row * ndim;
void lu_decompose(nec_output_file& s_outpu, long long int n,
        complex_arra& a, int_array& ip, long long int ndim)
long long int i_offset = i * ndim;
long long int j_offset = 0;
long long int r_offset = r * ndim;
long long int j_offset = j * ndim;
void solve(int n, complex_array& a, int_array& ip, complex_array& b,
```

```
            long long int ndim)
        void lu_decompose(nec_output_file& s_output, long long int n,
            complex_arrray& a_in, int_array& ip, long long int ndim)
        void factrs(nec_output_file& s_output, long long int np,
            long long int nrow, complex_array& a, int_array& ip)
        long long int mode_offset = mode * np;
        void solves(complex_array& a, int_array& ip, complex_array& b,
            long long int neq, long long int nrh, long long int np,
            long long int n, long long int mp, long long int m,
            long long int nop, complex_array& symmetr_array)
        long long int column_offset = ic * neq;
        long long int ia = i + k * npeq;

c_geomery.cpp
        long long int jco1 = n_plus_2m;
        long long int jco2 = jco1 + m;



nec_context.cpp
        long long int iresrv = 0;
        void nec_context::cmset (long long int nrow, complex_arra& in_cm,
            nec_float rkhx)
        void nec_context::compute_marix_ss (int 1, int j2, int im1, int im2,
            complex_array& in_cm, long long int nrow, int itrp)
        void nec_context::cmsw (int j1, int 2, int i1, int i2,
            complex_array& in_cm, complex_array& cw, long long int ncw,
            long long int nrow, int itrp)
        void nec_context::cmws (int j, int i1, int i2, complex_array& in_cm,
            long long int nr, complex_array& cw, long long int nw, int itrp)
        long long int ipr,
        void nec_context::cmww (int j, int i1, int i2, complex_array& in_cm,
            long long int nr, complex_array& cw, long long int nw, int itrp)

matrix_algebra.h
        void lu_decompose (nec_output_file& s_output, long long int n,
            complex_array& a, int_array& ip, long long int ndim);
```



```
        void factrs (nec_output_file& s_output,  long long int np,
            long long int nrow, complex_array& a, int_array& ip );
        void solve ( int n, complex_array& a, int_array& ip, complex_array& b,
            long long int ndim );
        void solves (complex_array& a, int_array& ip, complex_array& b,
            long long int neq, long long int nrh, long long int np,
            long long int n, long long int mp, long long int m,
            long long int nop, complex_array& symmetry_array);

nec_context.h
        void cmset (long long int nrow, complex_array& in_cm,
            nec_float rkhx);
        void compute_matrix_ss (int j1, int j2, int im1, int im2,
            complex_array& in_cm, long long int nrow, int itrp);
        void cmsw (int j1, int j2, int i1, int i2, complex_array& in_cm,
            complex_array& cw, long long int ncw, long long int nrow,
            int itrp);
        void cmws (int j, int i1, int i2, complex_array& in_cm,
            long long int nr, complex_array& cw, long long int nw, int itrp);
        void cmww (int j, int i1, int i2, complex_array& in_cm, long long int nr,
            complex_array& cw, long long int nw, int itrp);
                                                                    //
                                                                    ☆
```