3.2. 言語比較

3.2.1. Fortran と C 言語，C++の速度比較

山口大学大学院医学系研究科　　平野　靖
(前・名古屋大学情報基盤センター)

1. はじめに

スパコンやPCクラスタなどを用いた科学計算においては, Fortranが用いられることが多い. しかし，とくに大学の情報基盤センターなどではユーザ層の拡大のためにこれまで Fortran を使ってこなかった研究者の取り込みを行なう必要がある. Fortran の文法は他の高級言語と比較して単純であり制約が強い反面，他の言語と比較して最適化が簡単であるため，高速な実行プログラムを生成可能である．また，高速な実行プログラムを生成可能であることから科学計算を行なう研究者が好んで Fortran を用い，さらに利用者数が多いことからコンパイラ開発者がさらなる高速化を行なうという図式があると考えられる．一方，C 言語や C++などの言語は，Fortran と比較すると抽象化レベルが高いため直感的なプログラムを作成できるという長所があるが，文法に大きな自由度がある上に，ポインタ操作が多く用いられるため，あまり高度な最適化ができない．その結果として高速な実行プログラムが生成できない，という問題がある.

そこで，本文では，Fortran，C 言語および C++で同様の処理を行なう際の計算速度を評価する.

2. 測定に使用する処理とプログラムの概要

速度比較を行なう処理は，逐次処理による「行列同士の積」と「計算結果のファイルへの書き出し」とし，それぞれに要する CPU 時間を測定した．なお，行列のサイズは 1000×1000 とし，ファイルへの書き出しの際の通信の影響を避けるために, 計算結果は計算を行なった計算機のローカルディスク(/tmp)に書き出した.

また，言語，メモリの確保方法，およびファイルへの書き出し方法の違いによる性能の差異を比較するため, 下記のような同じ処理を行なう 24 個のプログラムを作成した. 表 1 にプログラムの概要を示す. また，作成したプログラムのうちのいくつかを文末の【補足資料】に例示する．表 1 で，例えばプログラム番号 17 は「2 次元配列」が×であり,「動的確保」と「構造体」,「書式付」が○となっている．これは，プログラム番号 17 では，構造体のメンバーとなっている行列を 1 次元的に動的確保し，行列積の計算結果を書式付きで書き出すことを意味する.

[C 言語]　8 個
- 行列を 1 次元配列で確保するか 2 次元配列で確保するか
- 配列(1 次元あるいは 2 次元)を静的に確保するか動的に確保するか
- 配列を構造体のメンバーにするか否か

[C++]　4 個
- 行列を 1 次元配列で確保するか 2 次元配列で確保するか
- 配列(1 次元あるいは 2 次元)を静的に確保するか動的に確保するか

なお，C++においては，すべてのプログラムで配列をクラスのメンバーとした.

[Fortran]　12 個
- 行列を 1 次元配列で確保するか 2 次元配列で確保するか
- 配列(1 次元あるいは 2 次元)を静的に確保するか動的に確保するか
- 配列を構造体のメンバーにするか否か
- 配列をファイルに書き出す際の書式の有無

なお，行列を 1 次元配列あるいは 2 次元配列で確保したときに，静的確保＆構造体＆書式無および動的確保＆構造体＆書式無のプログラムは作成していない．この理由として，たとえば静的確保＆構造体＆書式無の行列積に要する時間は静的確保＆構造体＆書式付と同様であり，書式の有無によるファイル

書き出しに要する時間の違いは静的確保＆非構造体＆書式付と静的確保＆非構造体＆書式無を比較することで推測可能であるからである．

表 1．測定に使用したプログラムの条件

	C 言語								C++				Fortran											
プログラム番号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
2 次元配列	×	×	×	×	○	○	○	○	×	×	○	○	×	×	×	×	×	×	○	○	○	○	○	○
動的確保	○	×	○	×	○	×	○	×	○	×	○	×	○	○	×	×	○	×	○	○	×	×	○	×
構造体	×	×	○	○	×	○	○	×	○	○	○	○	×	×	×	×	○	○	×	×	×	×	○	○
書式付													○	×	○	×	○	○	○	×	○	×	○	○

3．測定に用いた計算機，コンパイラおよびコンパイラオプション

用いた計算機は富士通製 SPARC Enterprise M9000，FX1 および HX600 である．各計算機の諸元を表 2 に示す．

表 2．各計算機の諸元

	M9000	FX1	HX600
CPU	SPARC64VII		AMD Opteron (Shanghai)
1CPU あたりのコア数	4		
クロック周波数	2.5GHz		
コアあたりの理論演算性能	10GFlops		
L1 キャッシュ	64KB(データキャッシュ，コア毎)		
L2 キャッシュ	6MB(コア共通)		512KB(コア毎)
L3 キャッシュ	なし		6MB(コア共通)
OS	Solaris10	OpenSolaris	RHEL4.7

また，用いたコンパイラとコンパイラオプションは下記の通りである．これらのオプションを指定することにより，いずれのコンパイラでも最高レベルの最適化が行なわれる．

[C 言語]

富士通コンパイラ

M9000(Version 5.8)	fcc	-Kfast,V9 -O5
HX600(Version 3.2)	fcc	-Kfast -O5
FX1(Version 5.8)	fcc	-Kfast,V9 -O5
SunStudio12	cc	-fast
GNU コンパイラ(Version 3.4.6)	gcc	-O3

[C++]

富士通コンパイラ

M9000(Version 5.8)	fcc	-Kfast,V9 -O5
HX600(Version 3.2)	fcc	-Kfast -O5
FX1(Version 5.8)	fcc	-Kfast,V9 -O5
SunStudio12	CC	-fast
GNU コンパイラ(Version 3.4.6)	g++	-O3

[Fortran]

富士通コンパイラ

M9000(Version 8.1)	fcc	-Kfast,V9,tl_trt -X9 -NRtrap -O5
HX600(Version 3.2)	fcc	-Kfast -X9 -NRtrap -O5
FX1(Version 8.1)	fcc	-Kfast,V9,tl_trt -X9 -NRtrap -O5
SunStudio12	f95	-fast
GNU コンパイラ(Version 3.4.6)	g77	-O3

なお，富士通コンパイラ，あるいは SunStudio12 では，-Kfast あるいは-fast オプションの指定により，コンパイルを行なう計算機のアーキテクチャに最適なオプションが自動的に設定される．富士通コンパイラの-Kfast オプションおよび SunStudio12 の-fast オプションは各計算機で下記のように展開された．

富士通コンパイラの-Kfast オプションの展開結果

[M9000]
-O5 -Kfsimple -Kdalign -Kns -Kfuse -Kmfunc -Kprefetch -VIS2 -FMADD -Keval -KSPARC64VII

[HX600]
-O3 -Komitfp -Keval -Kmfunc -Kprefetch -KSSE2 -KSSE3 –KOPTERON

[FX1]
-O5 -Kfsimple -Kdalign -Kns -Kfuse -Kmfunc -Kprefetch -VIS2 -FMADD -Keval -KSPARC64VII

SunStudio12 の-fast オプションの展開結果

[M9000]
-xO5 -xarch=sparcfmaf -xcache=64/64/2:6144/256/12 -xchip=sparc64vi -xdepend=yes -xmemalign=8s -fsimple=2 -fns=yes -ftrap=%none -xlibmil -xlibmopt -xbuiltin=%all -dryrun

[HX600]
-xO5 -xarch=amdsse4a -xcache=64/64/2:512/64/16 -xchip=amdfam10 -xdepend=yes -fsimple=2 -fns=yes -ftrap=%none -xlibmil -xbuiltin=%all -nofstore -xregs=frameptr -Qoption CC -iropt -Qoption CC -xcallee64 -dryrun -Qoption ube -xcallee=yes

[FX1]
-xO5 -xarch=sparcfmaf -xcache=64/64/2:6144/256/12 -xchip=sparc64vi -xdepend=yes -xm emalign=8s -fsimple=2 -fns=yes -ftrap=%none -xlibmil -xlibmopt -xbuiltin=%all -dryrun

なお，それぞれのコンパイラオプションの詳細は下記の URL を参照されたい．
富士通：http://www2.itc.nagoya-u.ac.jp/riyou/tuning.pdf
SunStudio12：http://jp.sun.com/products/software/tools/studio12/documentation/ss12/mr/man1/cc.1.html
GNU：http://gcc.gnu.org/onlinedocs/gcc-3.4.6/gcc/Optimize-Options.html#Optimize-Options

4. 測定結果

測定結果を図 1～6 に示す．このうち図 1～3 は計算機ごとの CPU 時間を，図 4～6 はコンパイラごとの CPU 時間を示す．また，各図(a)は行列積の計算に要した CPU 時間を，各図(b)は計算結果の書き出しに要した CPU 時間を示す．プログラムの実行は各 10 回行い，CPU 時間の平均値を求めた．

なお，g77 は，Fortrun90 から導入された配列の動的確保に対応していないため，いずれの計算機でもプログラム番号 13,14,17～20,23,24 については測定していない．また，FX1 においては g77 でコンパイルした Fortran プログラムの実行が不可能であったため，測定していない(プログラム番号 13～24)．

図 1～図 6 から観察される事項を下記に示す．

M9000 における計算時間の比較(図 1(a))：

- 富士通コンパイラと SunStudio12 が GNU コンパイラと比較して性能が高い．
- GNU コンパイラで C 言語プログラムにおいて配列を動的に確保した場合には静的に確保した場合(プログラム番号 1,3,5,7)に比べて計算時間が 15%程度増加している．
- C 言語プログラムおよび C++プログラムにおいて配列を 2 次元的に動的確保する(プログラム番号 5,7,11)と，富士通コンパイラおよび SunStudio12 においては，3～4 倍程度，GNU コンパイラにおいては 2 倍程度計算時間が増大する．
- SunStudio12 においては Fortran プログラムで配列を 1 次元配列として確保する(プログラム番号

13～18)と 2 次元的に確保した場合(プログラム番号 19～24)に比べて計算時間が 2 倍程度増大する.

- 富士通コンパイラは配列を 2 次元的に動的確保した場合(プログラム番号 19～24)を除き，安定して計算時間が短い.

M9000 における書き出し時間の比較(図 1(b))：

- いずれのコンパイラでも C 言語プログラムの場合(プログラム番号 1～8)が最も書き出し時間が短く，安定している.
- 書き出し時間の長さは C++プログラム，Fortran プログラム，C 言語プログラムの順番である.
- いずれのコンパイラでも C++プログラム(プログラム番号 9～12)では他の言語に比べて書き出し時間が 5～15 倍増大する.

HX600 における計算時間の比較(図 2(a))：

- 富士通コンパイラと GNU コンパイラにおいては, いずれのプログラムでも同程度の計算時間となった.
- 富士通コンパイラにおいて，配列を構造体のメンバーにして 1 次元的に動的確保した場合(プログラム番号 17), 計算時間が 3 秒程度になる. これは他のプログラムでの計算時間の 1/5 程度である.
- 配列を 2 次元的に動的確保した場合(プログラム番号 5,7)および C++プログラムの場合(プログラム番号 9～12)を除き, SunStudio12 が富士通コンパイラおよび GNU コンパイラに比べて 1/2～1/10 程度計算時間が短い.
- SunStudio12 では，C 言語プログラムにおいて，配列を 2 次元的に動的確保した場合(プログラム番号 5,7)は，それ以外の方法で配列を確保した場合に比べて計算時間が 2～3 倍増大する.
- SunStudio12 では, C++プログラムの場合(プログラム番号 9～12)は他言語のプログラムに比べて計算時間が増大する.
- SunStudio12 では，Fortran プログラムにおいて配列を構造体のメンバーとして動的確保する(プログラム番号 17,23)と他の方法で配列を確保した場合に比べて計算時間が 5～10 倍程度増大する.

HX600 における書き出し時間の比較(図 2(b))：

- C 言語プログラムと Fortran プログラムでは C++プログラムに比べて書き出し時間が 1/2～1/3 程度である.
- C 言語プログラムではいずれの場合においても書き出し時間が同程度である.
- Fortran プログラムでは, 書式付きの書き出しの方が書式無しの書き出しに比べて書き出し時間が短い.

FX1 における計算時間の比較(図 3(a))：

- C 言語プログラムにおいて配列を 2 次元的に動的確保した場合(プログラム番号 5,7)，C++プログラムで配列を構造体(クラス)のメンバーとして 2 次元的に動的確保した場合(プログラム番号 11)および GNU コンパイラで C++プログラムをコンパイルした場合(プログラム番号 9～12)を除き, いずれにコンパイラおよびプログラムでの比較的計算時間が短い.

FX1 における書き出し時間の比較(図 3(b))：

- C 言語プログラムではいずれの場合においても書き出し時間が同程度である.
- Fortran プログラムでは，書式付きの書き出しの方が書式無しに比べて書き出し時間が短い.

富士通コンパイラにおける計算時間の比較(図 4(a))：

- 全体的な傾向として M9000 と FX1 は HX600 に比べて計算時間が 1/3～1/5 程度短い.
- C 言語プログラムにおいては，ほぼ HX600，M9000，FX1 の順番に計算時間が長いが，配列を 2 次元的に動的確保した場合(プログラム番号 5,7)では M9000 の計算時間が増大する.

富士通コンパイラにおける書き出し時間の比較(図 4(b))：

- いずれの計算機でも C 言語プログラムでの書き出し時間は同程度である.

- HX600 では C 言語プログラムおよび Fortran プログラムでの書き出し時間が同程度である.
- M9000 と FX1 では C 言語プログラムでの書き出し時間は Fortran プログラムに比べて 1/3～1/2 程度である.

SunStudio12 における計算時間の比較(図 5(a)):

- M9000 と FX1 において，C 言語プログラムで配列を 2 次元的に動的確保した場合(プログラム番号5,7)および C++プログラムで配列を構造体(クラス)のメンバーとして 2 次元的に動的確保した場合(プログラム番号 11)は計算時間が増大する.

SunStudio12 における書き出し時間の比較(図 5(b)):

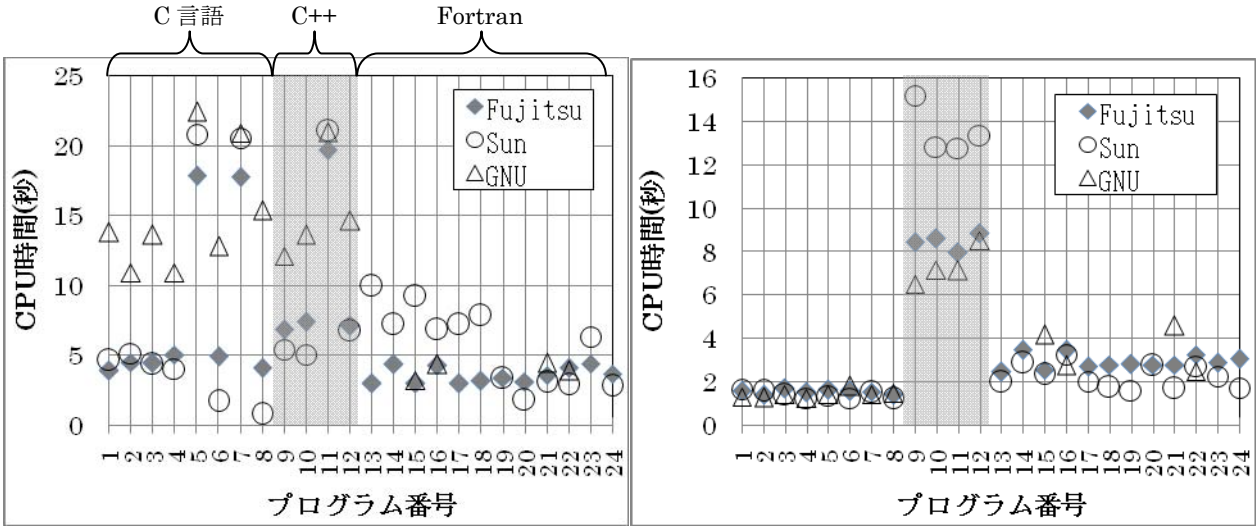
- C 言語プログラムおよび Fortran プログラムは C++プログラムに比べて書き出し時間が短い.
- HX600 では C 言語プログラムおよび Fortran プログラムでの書き出し時間が同程度である.
- いずれの計算機でも, Fortran プログラムで配列を構造体のメンバーとして書式付きで書き出した場合(プログラム番号 14,16,20,22)は，他の Fortran プログラムに比べて書き出し時間が 2 倍程度増大する.

GNU コンパイラにおける計算時間の比較(図 6(a)):

- C 言語プログラムにおいては，ほぼ HX600，M9000，FX1 の順番に計算時間が長いが，配列を 2 次元的に動的確保した場合(プログラム番号 5,7)では M9000 の計算時間が増大する.
- C++プログラム(プログラム番号 9～12)では，いずれの計算機でも同程度の計算時間である.
- C++プログラムで配列を 2 次元的に動的確保した場合(プログラム番号 11)では M9000 の計算時間が増大する.

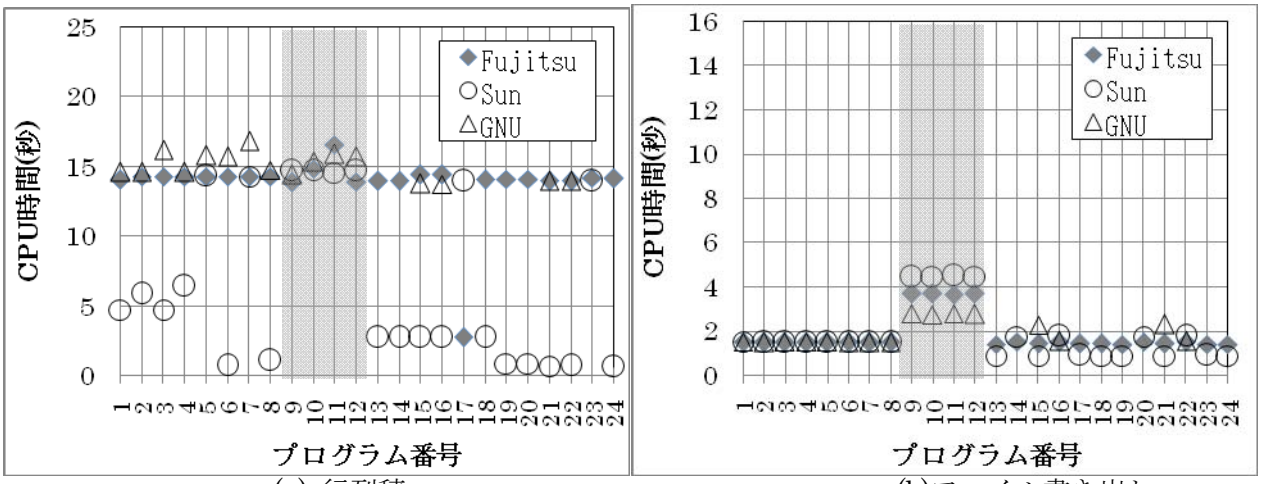
GNU コンパイラにおける書き出し時間の比較(図 6(b)):

- 書き出し時間の長さは C++プログラム，Fortran プログラム，C 言語プログラムの順番である.
- C 言語プログラムでは, いずれの計算機でも書き出し時間はほぼ同程度であるが, C++プログラムと Fortran プログラムでは M9000 の書き出し時間が増大する傾向にある.



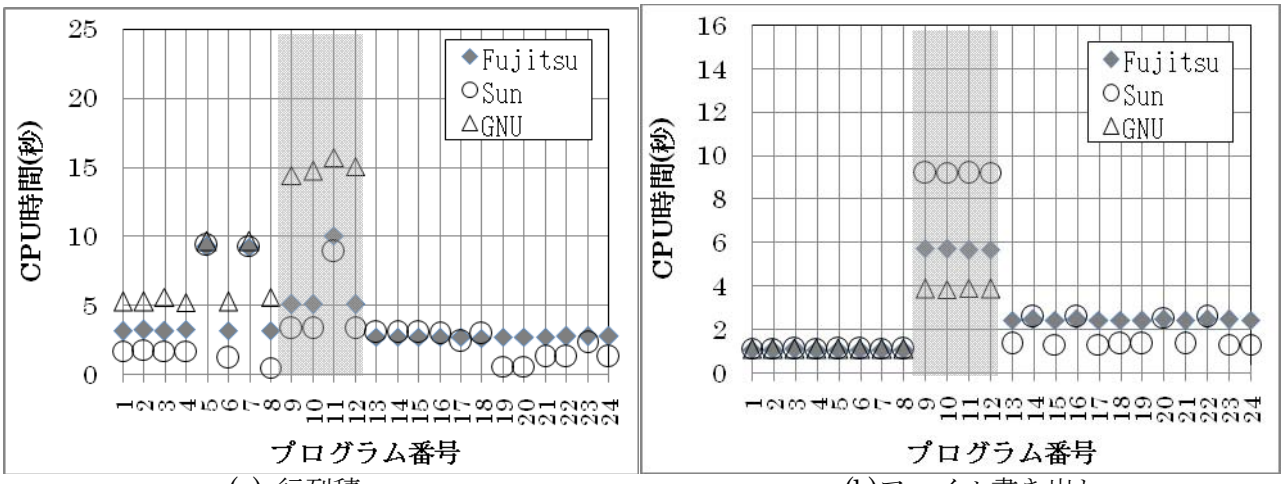
(a) 行列積　　(b)ファイル書き出し

図 1. M9000



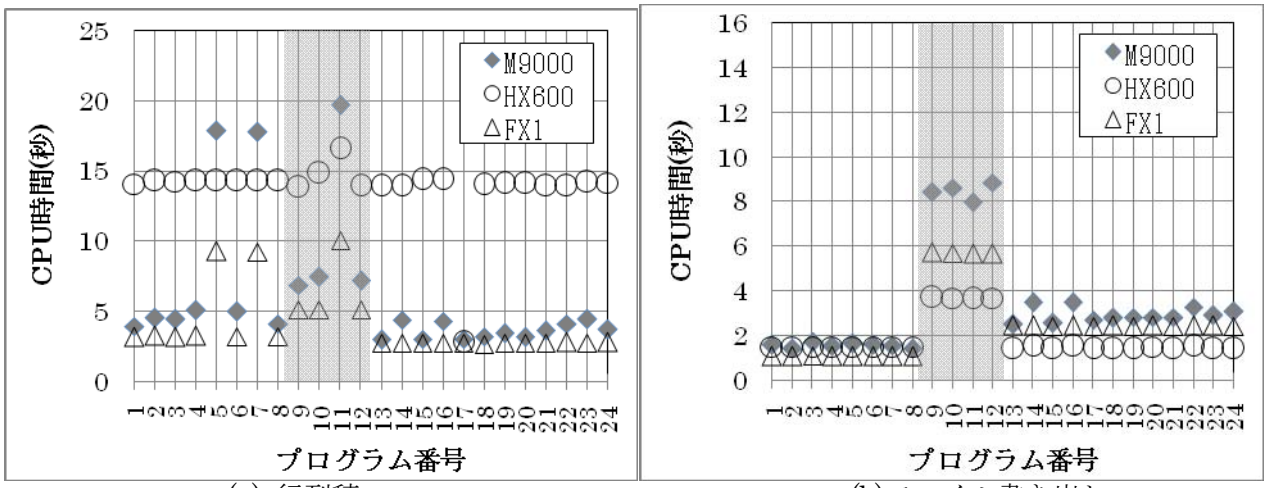
(a) 行列積　　(b)ファイル書き出し

図 2. HX600



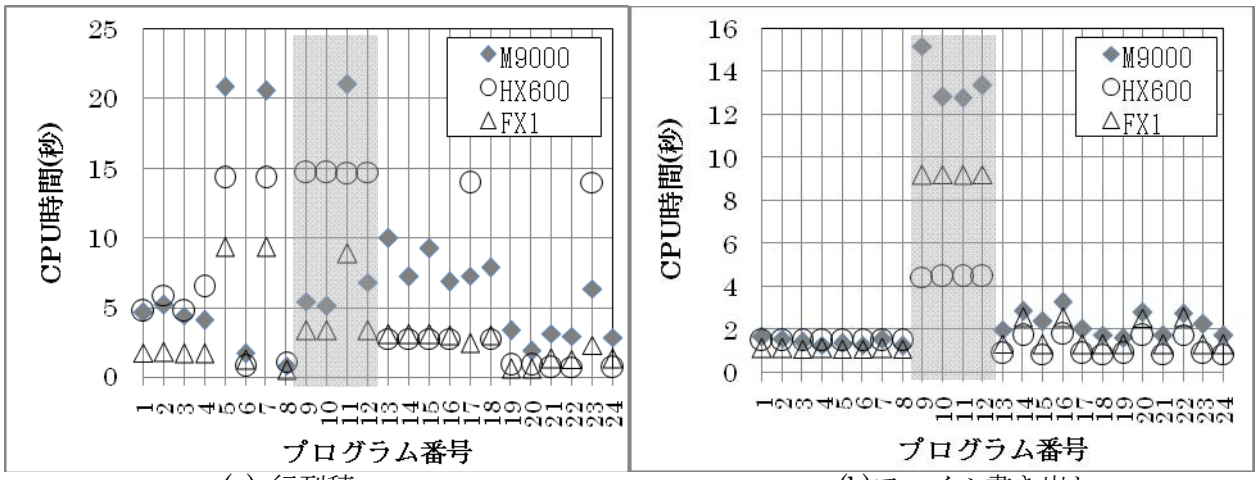
(a) 行列積　　(b)ファイル書き出し

図 3. FX1



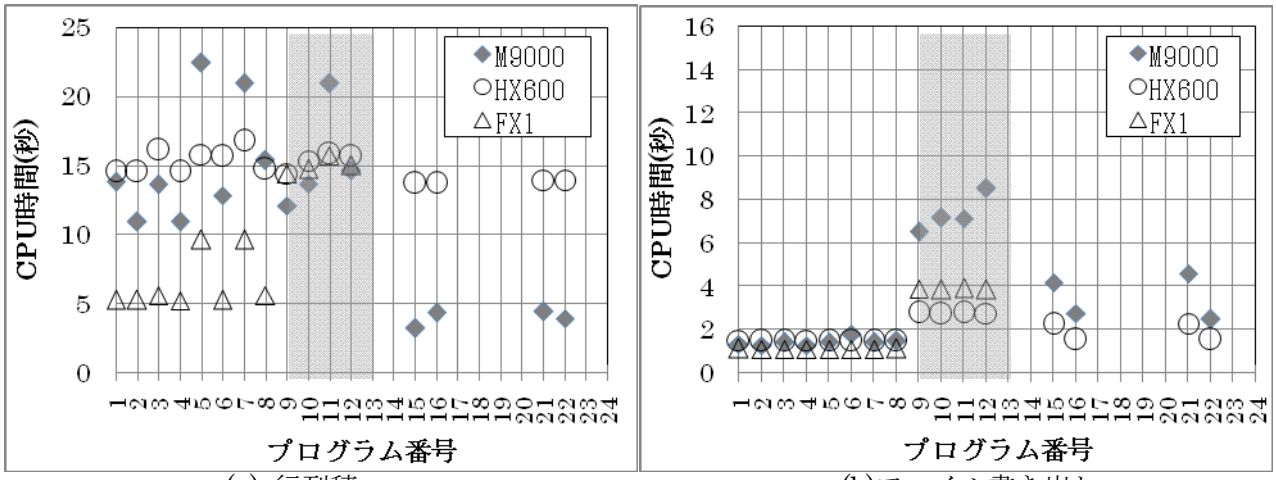
(a) 行列積　　(b)ファイル書き出し

図 4. 富士通コンパイラ



(a) 行列積　　(b)ファイル書き出し

図 5. SunStudio12



(a) 行列積　　(b)ファイル書き出し

図 6. GNU コンパイラ

5. 考察とまとめ

今回の測定において，SunStudio12 が最も高い計算性能を示す場合が多かった．これは，SunStudio12 のインストール時に CPU のアーキテクチャ，キャッシュの特性などを推定し，最適なコンパイルオプションを指定する機構が搭載されているために，いずれの計算機においても高い演算性能を引き出していることが可能であったと推測される．なお，SunStudio12 では，CPU を-xchip=sparc64vi のように誤って推定している．しかし，キャッシュの特性については，-xcache=64/64/2:6144/256/12 のように推定しており，少なくともキャッシュサイズについては SPARC64 VII のキャッシュの特性が正しく推定されている(SPARC64 VI は L1$:128KB および L2$:6144KB, SPARC64 VII は L1$:64KB および L2$:6144KB).

多くの場合において，FX1 がもっとも高い演算性能を示したことの原因として，名古屋大学情報基盤センターの運用方針では M9000 とは異なり FX1 はデフォルトで逐次プログラムであってもノードを占有して使用できること，および HX600 に比べてメモリバンド幅が広いことが考えられる．また，ノードを占有的に利用することが可能な HX600 と FX1 では，平均値に対する標準偏差の大きさ(=標準偏差/平均値)が高々0.02 程度であるのに対して，ノードを占有できない M9000 では 0.1～2.5 であった．これは M9000 ではメインメモリやローカルディスクへの読み書き時に他のユーザのプログラムの影響によるものと考えられる．

いずれのコンパイラおよび計算機を使った場合でも，C 言語プログラムや C++プログラムで配列を 2 次元的に動的確保した際に計算性能が極端に低下することがある．この原因としては，よく知られている事象ではあるが，実際の値が格納されているメモリ領域に到達するまでにアドレス参照が複数回発生することが原因であると考えられる．一方で C++プログラムを除けば，いずれの言語であっても配列を 1 次元的に確保したり，静的に確保したりすることにより，同程度の計算性能を得られることが分かった．したがって，言語の選択に関しては，2 次元的に動的確保を行なわないという点に注意すれば，C 言語プログラムであっても Fortran プログラムと同程度の演算性能が得られる可能性があることが分かる．

SunStudio12 で Fortran プログラムをコンパイルした場合に，配列を 1 次元的に確保した場合の方が 2 次元的に確保した場合よりも計算時間が長くなる傾向があった．これは富士通コンパイラおよび GNU コンパイラと異なる挙動であるとともに，一般常識とも異なる挙動であり，再検証が必要とされる．

I/O 性能についても，C 言語プログラムと Fortran プログラムに関しては大きな性能の差は見られなかったが，C++プログラムでは大幅に低下した．また，今回の測定では多くの場合において Fortran プログラムでの書式の有無に関して性能の差は見られなかったが，書式付の方が書き出し時間が短い場合も観測された．一般的には書式付の方が I/O 性能が低下すると認識されており，今後の再検証が必要とされる．

以上

【補足資料】
作成したプログラムの例

プログラム番号 7　(C 言語・2 次元的に配列確保・配列の動的確保・構造体の使用)

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define N 1000

typedef struct matrix{
double **mat;
int       x, y;
} Matrix;

int main( void )
{
Matrix     a, b, c;
double tmp;
int       n=N;
int       i, j, k;
time_t tv1, tv2, tv3;
FILE     *output;

output = fopen("mm-2D-struct-dynamic-c.out", "w" );
a.mat = (double **)malloc( sizeof(double *)*N );
b.mat = (double **)malloc( sizeof(double *)*N );
c.mat = (double **)malloc( sizeof(double *)*N );

for( i=0; i<N; i++ ){
a.mat[i] = (double *)malloc( sizeof(double)*N );
b.mat[i] = (double *)malloc( sizeof(double)*N );
c.mat[i] = (double *)malloc( sizeof(double)*N );
   }

for( i=0; i<N; i++ ){
for( j=0; j<N; j++ ){
a.mat[i][j] = (N-i) * j;
b.mat[i][j] = (N-i) * (N-j);
      }
   }

a.x = b.x = c.x = N;
a.y = b.y = c.y = N;
tv1    = clock();

for( i=0; i<a.x; i++ ){
for( j=0; j<a.y; j++ ){
tmp = 0.0;
for( k=0; k<a.y; k++ ){
tmp += a.mat[i][k] * b.mat[k][j];
         }
c.mat[i][j] = tmp;
      }
   }

tv2 = clock();
for( i=0; i<a.x; i++ ){
fprintf( output, "[%d]¥n", i);
for( j=0; j<a.y; j++ ){
fprintf( output, "%d¥t%9.6f¥n", j, c.mat[i][j]);
      }
fprintf( output, "¥n");
   }

tv3 = clock();
fprintf( output, "¥n%9.6f¥t%9.6f¥n", (double)(tv2 - tv1)
/CLOCKS_PER_SEC,                (double)(tv3                -
tv2)/CLOCKS_PER_SEC );

return(0);
}
```

プログラム番号 11　(C++・2 次元的に配列確保・配列の動的確保・クラスの使用)

```
#include<iostream>
#include<fstream>
#include<iomanip>
#include<time.h>
#define N 1000

using namespace std;

class Matrix
{
public:
Matrix(int size);
~Matrix();
void product(Matrix a, Matrix b);
double **matrix;
int x, y;
};

Matrix::Matrix(int size)
{
int i;

matrix = new double*[size];
for( i=0; i<size; i++ ) matrix[i] = new double[size];
x = size;
y = size;
}

void Matrix::product(Matrix a, Matrix b)
{
   double tmp;
   int i, j, k;

   for( i=0; i<x; i++ ){
      for( j=0; j<y; j++ ){
tmp = 0.0;
for( k=0; k<a.x; k++ ){
tmp += a.matrix[i][k] * b.matrix[k][j];
         }
matrix[i][j] = tmp;
      }
   }
}

int main( void )
{
Matrix a(N), b(N), c(N);
int      i, j;
time_t tv1, tv2, tv3;
ofstream output;

output.open("mm-2D-dynamic-c++.out", ios::out);

for( i=0; i<a.x; i++ ){
for( j=0; j<a.y; j++ ){
a.matrix[i][j] = (N-i) * j;
b.matrix[i][j] = (N-i) * (N-j);
      }
   }

tv1 = clock();
c.product( a, b );
tv2 = clock();

output.width(15);
output << setprecision(6);
output << setiosflags(ios::fixed);

for( i=0; i<c.x; i++ ){
      output << "["<< i << "]" << endl;
      for( j=0; j<c.y; j++ ){
         output << j << "¥t" << c.matrix[i][j] << endl;
       }
      output << endl;
   }
   tv3 = clock();

output << endl << (double)(tv2 - tv1)/CLOCKS_PER_SEC << "¥t" << (double)(tv3 - tv2)/CLOCKS_PER_SEC << endl;

   return(0);
}
```

プログラム番号 23　(Fortran・2 次元的に配列確保・配列の動的確保・構造体の使用・書式付き出力)

```
program main
type data
   real*8, allocatable ::   matrix(:,:)
   integer   n
end type data
type(data) a, b, c
real*8   tmp
integer i, j, k
real      tv1, tv2, tv3

open(17, file='mm-2D-struct-dynamic-withFormat-f.out',
& status='replace')

a%n = 1000
b%n = 1000
c%n = 1000

allocate(a%matrix(a%n, a%n))
allocate(b%matrix(b%n, b%n))
allocate(c%matrix(c%n, c%n))

do j=1, a%n
   do i=1, a%n
      a%matrix(i, j) = (a%n-i+1) * (j-1);
      b%matrix(i, j) = (b%n-i+1) * (b%n-j+1);
   enddo
enddo

call cpu_time( tv1 );

 do i=1, a%n
   do j=1, b%n
      tmp = 0.0;
      do k=1, c%n
        tmp = tmp + a%matrix(i, k) * b%matrix(k, j)
        enddo
     c%matrix(i, j) = tmp
  enddo
enddo

call cpu_time( tv2 );

do j=1, c%n
   write(17, "(i4)") j
   do i=1, c%n
      write(17, "(i4,6x,f20.6)") i,   c%matrix(i, j)
   enddo
enddo

call cpu_time( tv3 )

write(17, *) tv2-tv1, tv3-tv2

stop
end
```