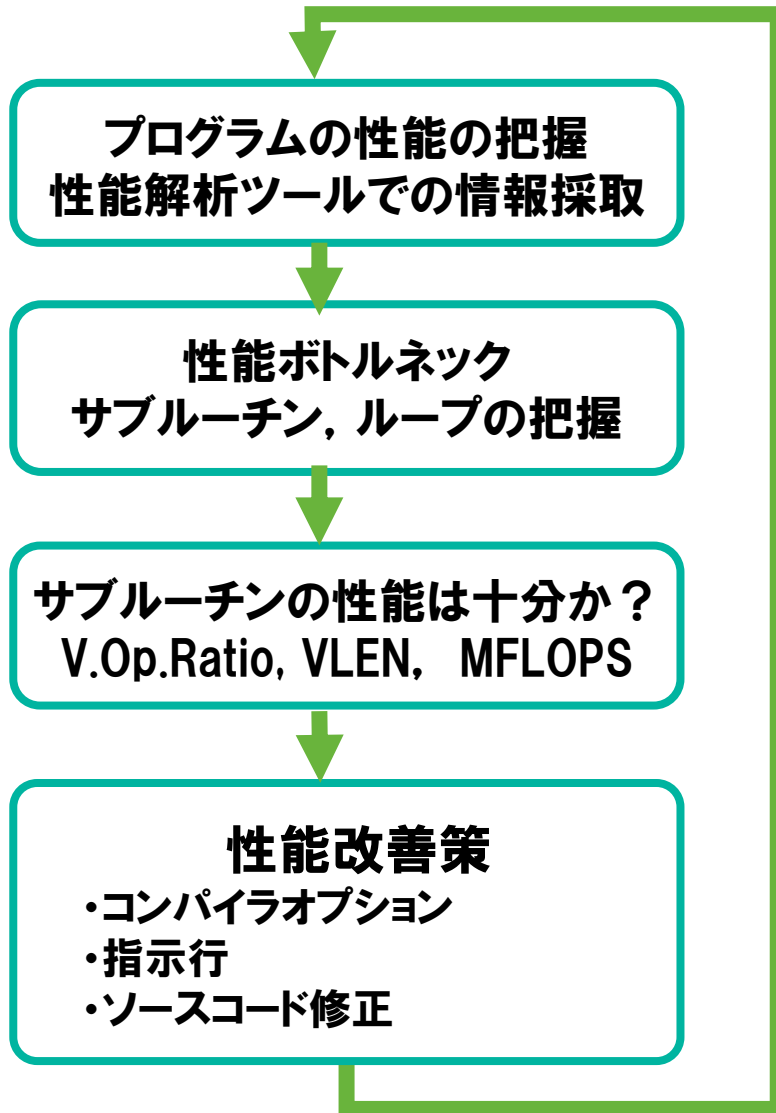


# ベクトルプロセッサ 高速化技法の基礎 (演習用資料)

大阪大学サイバーメディアセンター  
日本電気株式会社

**本資料は、東北大学サイバーサイエンスセンターとNECの共同により作成され、大阪大学サイバーメディアセンターの環境で実行確認を行い、修正を加えたものです。  
無断転載等は、ご遠慮下さい。**

# プログラム最適化の流れ



## 指示行とは...

コンパイラは、最適化を行う上でソースコード上からは判断できない条件があった場合、最適化を抑止します。ユーザーが明示的に指示行で条件を与えてあげることで、最適化を促進させることが可能になります

# 行列積のプログラムを使った課題

---

1. オリジナルコードのコンパイルと実行
2. 性能解析(Ftraceの利用)
3. アンローリング(コンパイラ指示行による最適化)
4. 行列積ライブラリ(コンパイラによる最適化)
5. 自動インライン展開(コンパイラオプションによる最適化)

# 演習問題の構成

## ディレクトリ構成

```
vector/  
|-- practice_1   オリジナルコード実行環境  
|-- practice_2   性能解析(Ftrace)演習問題  
|-- practice_3   コンパイラ指示行演習問題  
|-- practice_4   行列積ライブラリ置換演習問題  
`-- practice_5   自動インライン展開演習問題
```

# 1. 演習問題：オリジナルコードのコンパイルと実行

## 目的

- 現状のプログラムの性能を把握する。

## 手順

- コンパイル(リストの確認)
- 実行(結果, 性能の確認)

## ディレクトリ

- practice\_1

# 1. オリジナルコード:コンパイル(1)

## コンパイラオプション

```
nfort -report-all mat_tune0.f
```

- **-report-all**

コード生成リスト、診断メッセージリスト、編集リスト、インラインリスト、オプションリスト、ベクトルリストを出力する

## コンパイル

```
% ./comp.sx.sh
```

# 1. オリジナルコード:コンパイル(2)

## リストの確認

- 編集リスト(mat\_tune0.L)

```
25: +----->          do j=1, n
26: |+----->          do k=1, n
27: ||V----->        do i=1, n
28: |||               F    a (i, j)=a (i, j)+b (i, k)*c (k, j)
29: ||V-----        end do
30: |+-----          end do
31: +-----          end do
```

V:ベクトル化対象ループ



# 1. オリジナルコード:実行

## ジョブファイル (run.sx.sh)

```
#!/bin/bash

#PBS -q LECTUREV
#PBS --group=kosyuXXX
#PBS --venode=1
#PBS -l cpunum_job=2, elapstim_req=0:05:00
#PBS -j o -N p1-sx-sample

cd $PBS_O_WORKDIR

time ./a.out
```

## NQSVオプション

- q ジョブクラス名を指定
- group 課金グループを指定
- venode 使用するVE数を指定
- l 使用CPU数, 経過時間
- j o 標準エラー出力を標準出力と同じファイルへ出力する
- N ジョブ名を指定

## 実行

run.sx.sh をジョブ投入 (qsub) してください。

投入したジョブのステータスは、qstatコマンドで確認して下さい。

```
$ qsub run.sx.sh
Request *****.sqd submitted to queue: LECTUREV.          *****はジョブ番号
$ qstat
RequestID      ReqName  UserName Queue      Pri STT S  Memory   CPU  Elapse R H M Jobs
-----
*****.sqd    p1-sx-sa kosyuXXX LECTUREV   0  QUE -   0.00B   0.00   0 Y Y Y  1
```

# 1. オリジナルコード:実行結果

結果ファイル(p1-sx-sample.o\*\*\*\*\*) → 約33GFLOPS

```
***** Program Information *****
Real Time (sec) : 0.511899
User Time (sec) : 0.510496
Vector Time (sec) : 0.509786
Inst. Count : 583677718
V. Inst. Count : 134402095
V. Element Count : 34406933536
V. Load Element Count : 17179869585
FLOP Count : 17179869376
MOPS : 85160.400972
MOPS (Real) : 84872.547647
MFLOPS : 33674.900426
MFLOPS (Real) : 33561.074846
A. V. Length : 255.999979
V. Op. Ratio (%) : 98.965902
L1 Cache Miss (sec) : 0.000235
CPU Port Conf. (sec) : 0.000000
V. Arith. Exec. (sec) : 0.222687
V. Load Exec. (sec) : 0.287098
VLD LLC Hit Element Ratio (%) : 49.975928
FMA Element Count : 8589934592
Power Throttling (sec) : 0.000000
Thermal Throttling (sec) : 0.000000
Memory Size Used (MB) : 560.000000
Non Swappable Memory Size Used (MB) : 90.000000
```

PROGINFの出力

## 2. 演習問題：性能解析(Ftraceの利用)

### 目的

- 性能解析ツールFtraceを使い、性能情報を採取する。

### 手順

- ソースコードの修正(Ftrace\_Regionの挿入)
- コンパイラオプションの追加(-ftrace)
- 実行(結果, 性能の確認)

### ディレクトリ

- practice\_2

## 2. 性能解析(Ftraceの利用) : ソースコードの修正

mat\_tune0.f ユーザ指定リージョン (Ftrace\_Region) を挿入.

- プログラムの局所的な部分の性能を知りたい場合に使用する.
  - 通常の Ftrace はサブルーチン単位での情報を表示.  
⇒ Ftrace\_Region はループ単位で細かく情報採取が可能.
- CALL FTRACE\_REGION\_BEGIN/END で測定したい区間をはさむ.

コメントを外す

```
23      t1=etime (cp1)
24 !    CALL FTRACE_REGION_BEGIN (' Main-loop' )
25      do j=1, n
26          do k=1, n
27              do i=1, n
28                  a (i, j) =a (i, j) +b (i, k) *c (k, j)
29              enddo
30          enddo
31      enddo
32 !    CALL FTRACE_REGION_END (' Main-loop' )
```

## 2. 性能解析(Ftraceの利用) :コンパイラオプションの追加

comp.sx.shに-ftraceを追記する.

```
nfort -report-all mat_tune0.f -ftrace
```

- -ftrace  
簡易性能解析機能を利用することを指定する.

### ※注意

-ftraceオプションは測定オーバーヘッドが生じるため、実行回数の多いサブルーチンがある場合には実行時間が延びます。そのため、常に使用することはお勧めしません。

## 2. 性能解析(Ftraceの利用) : コンパイルと実行

### コンパイル

```
% ./comp.sx.sh
```

### 実行

```
$ qsub run.sx.sh  
Request *****.sqd submitted to queue: LECTUREV.
```

\*\*\*\*\*はジョブ番号

## 2. 性能解析(Ftraceの利用) : 実行結果

結果ファイル(ftrace.out) → **約33GFLOPS**

```
% ftrace  
もしくは  
% ftrace -f ftrace.out
```

```
*-----*  
FTRACE ANALYSIS LIST  
*-----*
```

```
Execution Date : Tue Sep 28 22:41:00 2021 JST  
Total CPU Time : 0:00' 00" 510 (0.510 sec.)
```

FREQUENCY	EXCLUSIVE TIME [sec] ( % )	AVER. TIME [msec]	MOPS	MFLOPS	V. OP RATIO	AVER. V. LEN	VECTOR TIME	L1CACHE MISS	CPU PORT	VLD CONF	LLC HIT	PROC. NAME
1	0.510 (100.0)	510.035	85182.4	33683.7	98.97	256.0	0.510	0.000	0.000	49.98	49.98	MAIN_
1	0.510 (100.0)	510.035	85182.4	33683.7	98.97	256.0	0.510	0.000	0.000	49.98	49.98	total
1	0.509 ( 99.9)	509.427	85190.7	33723.9	98.97	256.0	0.509	0.000	0.000	49.98	49.98	Main-loop

Ftrace\_Regionの情報

# 3. 演習問題:コンパイラ指示行

## 目的

- **アウターアンローリング指示行の使い方を理解する。**
- 4段アウターアンロールを行う。

## 手順

- ソースコードの修正
- コンパイル(リストの確認)
- 実行(結果, 性能の確認)

## ディレクトリ

- practice\_3



### 3. コンパイラ指示行:ソースコードの修正

mat\_tune0.f に4段outerloop\_unroll指示行を挿入.

```
% vi mat_tune0.f
```

- outerloop\_unroll指示行の挿入例

段数は2のべき乗の値のみ有効

```
25      do j=1, n
26  !nec$ outerloop_unroll(4)
27          do k=1, n
28              do i=1, n
29                  a(i, j)=a(i, j)+b(i, k)*c(k, j)
30              end do
31          end do
32      end do
```

# 3. コンパイラ指示行:コンパイルと実行

## コンパイル

```
% ./comp.sx.sh
```

## 編集リスト(mat\_tune0.L)の確認

```
25: +----->          do j=1, n
26: |                !nec$ outerloop_unroll (4)
27: |U----->          do k=1, n
28: ||V----->          do i=1, n
29: |||           F      a (i, j)=a (i, j)+b (i, k)*c (k, j)
30: ||V-----          end do
31: |U-----          end do
32: +-----          end do
```

U: 外側ループアンロール

4段アウターアンロールが行われる  
⇒配列aのメモリアクセスの回数が  
1/4になるため高速化される

## 実行

```
$ qsub run.sx.sh
Request *****.sqd submitted to queue: LECTUREV.
```

\*\*\*\*\*はジョブ番号

# 3. コンパイラ指示行: 実行結果

結果ファイル(p3-sx-sample.o\*\*\*\*\*) ⇒ **約60GFLOPS**  
⇒ **オリジナル (演習1) の1.8倍の性能向上**

```
***** Program Information *****
Real Time (sec) : 0.287512
User Time (sec) : 0.285835
Vector Time (sec) : 0.285028
Inst. Count : 268070678
V. Inst. Count : 92459055
V. Element Count : 23669515296
V. Load Element Count : 10737418641
FLOP Count : 17179869376
MOPS : 106119.741308
MOPS (Real) : 105344.342626
MFLOPS : 60193.763335
MFLOPS (Real) : 59753.937868
A. V. Length : 255.999970
V. Op. Ratio (%) : 99.420186
L1 Cache Miss (sec) : 0.000233
CPU Port Conf. (sec) : 0.000000
V. Arith. Exec. (sec) : 0.146272
V. Load Exec. (sec) : 0.138753
VLD LLC Hit Element Ratio (%) : 20.185404
FMA Element Count : 6442450944
Power Throttling (sec) : 0.000000
Thermal Throttling (sec) : 0.000000
Memory Size Used (MB) : 560.000000
Non Swappable Memory Size Used (MB) : 90.000000
```

PROGINFの出力

# 4. 演習問題：行列積ライブラリの利用

## 目的

- 行列積ライブラリの性能を確認する。

## 手順

- ソースコードの修正
- コンパイルスクリプトの修正
- コンパイル(リストの確認)
- 実行(結果, 性能の確認)

## ディレクトリ

- practice\_4

# 4. 行列積ライブラリの利用：プログラム修正

mat\_tune0.f の配列の型宣言を修正する。

```
% vi mat_tune0.f
```

- 配列Cの型を **real (4)** から **real (8)** に変更する。

```
4      implicit real (8) (a-h, o-z)
5      parameter ( n=2048 , moda=0 )
6      real (8) a (n+moda, n), b (n+moda, n)
7      real (4) c (n+moda, n)
```



```
4      implicit real (8) (a-h, o-z)
5      parameter ( n=2048 , moda=0 )
6      real (8) a (n+moda, n), b (n+moda, n)
7      real (8) c (n+moda, n)
```

## 4. 行列積ライブラリの利用:コンパイラオプションの追加

comp.sx.shに-03を追記する.

```
nfort -03 -report-all mat_tune0.f
```

- -03

副作用を伴う最適化・自動ベクトル化、および、多重ループの最適化を適用する.

# 4. 行列積ライブラリの利用:コンパイルと実行

## コンパイル

```
% ./comp.sx.sh
```

## 編集リスト(mat\_tune0.L)の確認

LINE	DIAGNOSTIC MESSAGE
27:	opt (1800): <b>Idiom detected (matrix multiply).</b>
24:	+-----> do j=1, n
25:	+-----> do k=1, n
26:	+-----> do i=1, n
27:	<b>M</b> a(i, j)=a(i, j)+b(i, k)*c(k, j)
28:	+----- end do
29:	+----- end do
30:	+----- end do

コンパイラが認識できる演算パターンでは、ライブラリへの置換が行われる。

**M:ベクトル行列積ライブラリ呼び出しの置換**

## 実行

```
$ qsub run.sx.sh  
Request *****.sqd submitted to queue: LECTUREV.
```

\*\*\*\*\*はジョブ番号

# 4. 行列積ライブラリの利用：実行結果

結果ファイル(p4-sx-sample.o\*\*\*\*\*) ⇒ 約299GFLOPS  
⇒オリジナル(演習1)の9.1倍の性能向上

```
***** Program Information *****
Real Time (sec) : 0.059252
User Time (sec) : 0.057691
Vector Time (sec) : 0.056920
Inst. Count : 86153154
V. Inst. Count : 35187247
V. Element Count : 8976475168
V. Load Element Count : 180355473
FLOP Count : 17179869413
MOPS : 307393.339273
MOPS (Real) : 297338.725232
MFLOPS : 299759.596830
MFLOPS (Real) : 289954.676988
A. V. Length : 255.105924
V. Op. Ratio (%) : 99.710707
L1 Cache Miss (sec) : 0.000241
CPU Port Conf. (sec) : 0.000000
V. Arith. Exec. (sec) : 0.056410
V. Load Exec. (sec) : 0.000510
VLD LLC Hit Element Ratio (%) : 88.598002
FMA Element Count : 8589934592
Power Throttling (sec) : 0.000000
Thermal Throttling (sec) : 0.000000
Memory Size Used (MB) : 560.000000
Non Swappable Memory Size Used (MB) : 90.000000
```

PROGINFの出力



# 5. 演習問題：自動インライン展開

## 目的

- **自動インライン展開のオプションの使い方を理解する。**

## 手順

- **インライン展開前の性能の確認**
  - ・ コンパイル(リストの確認)
  - ・ 実行(結果, 性能の確認)
- **インライン展開後の性能の確認**
  - ・ コンパイルスクリプトへオプション追加, 再コンパイル(リストの確認)
  - ・ 再実行(結果, 性能の確認)

## ディレクトリ

- `practice_5`

# 5. 自動インライン展開:インライン展開前のコンパイル

## コンパイル

```
% ./comp.sx.sh
```

## 編集リスト(mat\_tune1.L)の確認

- サブルーチン呼び出しがあり、ベクトル化ができていない。

LINE	DIAGNOSTIC MESSAGE
26:	vec ( 103): Unvectorized loop.
26:	vec ( 110): <b>Vectorization obstructive procedure reference.: MUL</b>
24:	+-----> do j=1, n
25:	+-----> do k=1, n
26:	+-----> do i=1, n
27:	call mul(n, moda, i, j, k, a, b, c)
28:	+----- end do
29:	+----- end do
30:	+----- end do

# 5. 自動インライン展開:インライン展開前の実行結果

## 実行

```
$ qsub run.sx.sh
```

```
Request *****.sqd submitted to queue: LECTUREV.
```

\*\*\*\*\*はジョブ番号

結果ファイル(p4-sx-sample.o\*\*\*\*\* ) ⇒ 約0.091GFLOPS

```
***** Program Information *****
```

```
Real Time (sec)      :      23.373303
User Time (sec)     :      23.372079
Vector Time (sec)   :       0.000098
Inst. Count         :     71948356346
V. Inst. Count      :       47151
V. Element Count    :     12067872
V. Load Element Count :       401
FLOP Count          :     2147483837
MOPS                 :     3078.963993
MOPS (Real)         :     3078.742341
MFLOPS              :       91.884252
MFLOPS (Real)       :       91.877637
A. V. Length        :     255.940956
V. Op. Ratio (%)    :       0.016770
. . .
```

PROGINFの出力

## 5. 自動インライン展開:コンパイラオプションの追加

comp.sx.sh に `-finline-functions -finline-file=mul.f` を追記する.

```
nfort -report-all mat_tune1.f mul.f -finline-functions -finline-file=mul.f
```

- `-finline-functions`

自動インライン展開を適用する.

- `-finline-file=filename.f`

インライン展開する手続をサーチするとき、指定されたソースファイル(*filename.f*)をサーチする. 複数指定するときにはコロン(:)で区切って指定する. allが指定されたとき、コマンドラインで指定されたコンパイル対象のすべてのソースファイルもサーチする.

# 5. 自動インライン展開:コンパイラオプションの追加

## コンパイル

```
% ./comp.sx.sh
```

⇒ インライン展開され, ベクトル化できた

LINE	DIAGNOSTIC MESSAGE
25:	vec ( 103): Unvectorized loop.
25:	vec ( 113): Overhead of loop division is too large.
25:	vec ( 128): Fused multiply-add operation applied.
26:	vec ( 101): Vectorized loop.
27:	inl (1222): <b>Inlined: MUL</b>
24:	+-----> do j=1, n
25:	+----->F do k=1, n
26:	V-----> do i=1, n
27:	call mul (n, moda, i, j, k, a, b, c)
28:	V----- end do
29:	+----- end do
30:	+----- end do

**I:関数呼び出しをインライン展開**

# 5. 自動インライン展開:インライン展開後の実行結果

## 実行

```
$ qsub run.sx.sh
```

```
Request *****.sqd submitted to queue: LECTUREV.
```

\*\*\*\*\*はジョブ番号

結果ファイル(p4-sx-sample.o\*\*\*\*\* ) ⇒ **約18GFLOPS**

インライン展開により, 0.091GFLOPS から 18GFLOPS に性能向上した.

```
***** Program Information *****
```

```
Real Time (sec)      :          0.118415
User Time (sec)     :          0.117112
Vector Time (sec)   :          0.116351
Inst. Count         :        78947086
V. Inst. Count      :        16824367
V. Element Count    :       4307035168
V. Load Element Count :       2147484049
FLOP Count          :       2147483840
MOPS                :       46626.694677
MOPS (Real)         :       45965.166671
MFLOPS              :       18396.457522
MFLOPS (Real)       :       18135.453135
A. V. Length        :        255.999835
V. Op. Ratio (%)    :        98.858647
```

```
...
```

PROGINFの出力

# ライブラリ

## コンパイラが自動的に置き換えるもの

- 行列積パターン

## ソースコード修正により使用できるもの

- NLC(NEC Numeric Library Collection)
  - ASL(ASL/SX, フーリエ変換, 乱数生成, ソート, FFTW3)
  - BLASライブラリ
  - LAPACKライブラリ など

# おすすめコンパイラオプション

## プログラムを初めてスーパーコンピュータ(SX-Aurora TSUBASA)で実行する場合

```
nfort -02 -report-all “プログラムファイル名”
```

- -02  
副作用を伴う最適化・自動ベクトル化を適用する。(既定値)
- -report-all  
コード生成リスト、診断メッセージリスト、編集リスト、インラインリスト、オプションリスト、ベクトルリストを出力する。

## 正常終了した場合、-03および-04を使用して-02の結果と比較

- -03  
副作用を伴う最適化・自動ベクトル化、および、多重ループの最適化を適用する。
- -04  
C/C++/FORTRANの言語仕様を逸脱した副作用を伴う最大限の最適化・自動ベクトル化を適用する。



# デバッグ用コンパイラオプション

## 正常終了したが、結果がおかしい場合

- -O1

副作用を伴わない最適化・自動ベクトル化を適用する。

## 異常終了 (Segmentation fault) した場合、デバッグ用オプションで分析

- -fcheck=bounds

配列の上下限のチェックを有効にする。

- -traceback [=verbose]

実行時に環境変数VE\_TRACEBACKがセットされているとき、トレースバック情報を出力するオブジェクトファイル、実行ファイルを生成する。

verboseを指定した場合、トレースバック情報を出力する際に、ファイル名や行番号情報を出力するための情報を追加したオブジェクトファイル、実行ファイルを生成する。

## 初期化漏れのチェック

- -minit-stack=SNAN

実行時にスタックに割り付ける領域を倍精度浮動小数点型のSignaling NaN(0x7ff4000000000000)で初期化する。