SX-7プログラムチューニング

2003年3月12日

日本電気株式会社
第一コンピュータソフトウェア事業部

NEC

内容

- FORTRAN90/SX
  - FORTRAN90/SX**の特長**
  - **ベクトル化チューニング**
  - **並列化チューニング**
  - OpenMP
- C++/SX
- MPI/SX

FORTRAN90/SX

FORTRAN90/SXの特長

言語の特長

- **Fortran95規格完全準拠**
  ISO (JIS) Fortran最新規格に完全準拠
- **拡張仕様**
  FORTRAN77/SX完全上位互換
  8ﾊﾞｲﾄ整数型ｻﾎﾟｰﾄ
- **他社互換機能**
  デファクト機能のサポート
  他社ﾊﾞｲﾅﾘﾃﾞｰﾀ形式の入出力機能

チューニング支援機能

- **手軽に性能情報の採取可能**
  ﾌﾟﾛｸﾞﾗﾑ特性情報出力
  簡易性能解析機能
  入出力解析情報出力

最適化機能

- 手続きのインライン展開
- 分岐を越えた命令並べ換え
- Fortran95向け最適化
  (配列構文に対する最適化等)

強力な自動ベクトル化/自動並列化機能

- ループ最適化
  (入れ換え、一重化、ｱﾝﾛｰﾘﾝｸﾞ等)
- 条件ベクトル化／条件並列化等

OpenMP

- 共有メモリマシン向け並列処理の標準 API をサポート

FORTRAN90/SX:ベクトル化/並列化 (1)

- 既存のプログラムをコンパイルするだけで、SXのHW性能を引き出す実行プログラムを生成可能
  - DOループ、配列式を自動的にベクトル＋並列コードに変換
  - 高度なループ最適化機能により、ベクトル化、並列化の効果を高めるようにループの形を自動的に変形
    - ループアンローリング
    - ループの一重化、入れ換え
    - 条件ベクトル化、条件並列化
    - マクロ演算の認識（総和、内積、漸化式、最大･最小、行列積など）
    - ループ融合
    - etc.
  - 豊富なｺﾝﾊﾟｲﾗ指示行により、わずかな修正でﾁｭｰﾆﾝｸﾞ可能

FORTRAN90/SX:ベクトル化/並列化 (2)

条件並列化

```
DO J=1,M
  DO I=1,N
    A(I,J)=A(I,J)+B(I,J)*C(I,J)
  ENDDO
ENDDO
```

並列化効果が得られる粒度があるかどうか不明

最適化/並列化/ベクトル化

```
IF(M*N.GT.1000) THEN
  DO J=1,M                          並列
    DO I=1,N
      A(I,J)=A(I,J)+B(I,J)*C(I,J)   ベクトル
    ENDDO
  ENDDO
ELSE
  DO J=1,M
    DO I=1,N
      A(I,J)=A(I,J)+B(I,J)*C(I,J)   ベクトル
    ENDDO
  ENDDO
END IF
```

粒度により最適なコードを選択して実行

配列構文のﾍﾞｸﾄﾙ化＋並列化

```
a(1:N,1:1000) = b(1:N,1:1000) * c(1:N,1:1000)
b(1:N,1:1000) = sin(c(1:N,1:1000))
```

ループ展開イメージ

```
do j = 1, 1000
  do i = 1, N
    a(i,j) = b(i,j) * c(i,j)
    b(i,j) = sin(c(i,j))
  end do
end do
```

複数の配列代入文を1個のループにまとめる

最適化/並列化/ベクトル化

並列

ﾍﾞｸﾄﾙ

```
do i1=1, 500      ！外側ループアンローリング
  do i2 = 1, N
    a(i2,i1*2-1) = b(i2,i1*2-1)*c(i2,i1*2-1)
    a(i2,i1*2)   = b(i2,i1*2)*c(i2,i1*2)
    b(i2,i1*2-1) = sin(c(i2,i1*2-1))
    b(i2,i1*2)   = sin(c(i2,i1*2))
  end do
end do
```

f90コマンド,sxf90コマンドの指定形式

- **形式:**

  f90 *[オプション|ファイル名]* …　　(セルフコンパイラ)

  sxf90 *[オプション|ファイル名]* …　　(クロスコンパイラ)

- **ファイル名:以下のサフィックスに従う**
  - .f　　:Fortranソースファイル、固定形式
  - .f90　　:Fortranソースファイル、自由形式

  } オプションで変更可能

  - .o　　:オブジェクトファイル
  - .a　　:アーカイブファイル(ライブラリ)

主なコンパイル時オプション

-C ：**ベクトル化･最適化レベル等翻訳モードを指定**

例：f90 -C hopt sample.f　　*高レベルの最適化/ベクトル化を行う*

-pi ：**手続きのインライン展開機能を利用することを指定**

例: f90 -pi sample.f　　*自動インライン展開機能を使用*

-P :**並列化機能の使用に関する指定**

例：f90 -P auto sample.f　　*自動並列化機能を使用*

-R[012345]：**リスト出力制御を指定**

例： f90 -R5 sample.f　　*編集リストを出力することを指定*

-Wf：**詳細な機能を制御するオプションを指定**

例： f90 -Wf”-pvctl fullmsg” sample.f　　*最適化･ベクトル化の詳細な診断メッセージを表示*

ベクトル化チューニング

ベクトル化による高速化の観点

- **ベクトル化**
  ループ中で繰り返される規則的に並んだ配列データ（ベクトルデータ）の演算を高速なベクトル命令で行う実行プログラムを生成

(1) **ベクトル化率を上げる**
  ⇨ **ベクトル化の阻害要因を取り除き、ベクトル化を促進する**

(2) **ベクトル命令の効率を高める**
  ⇨ **ベクトル長（ループ長）を大きくする**

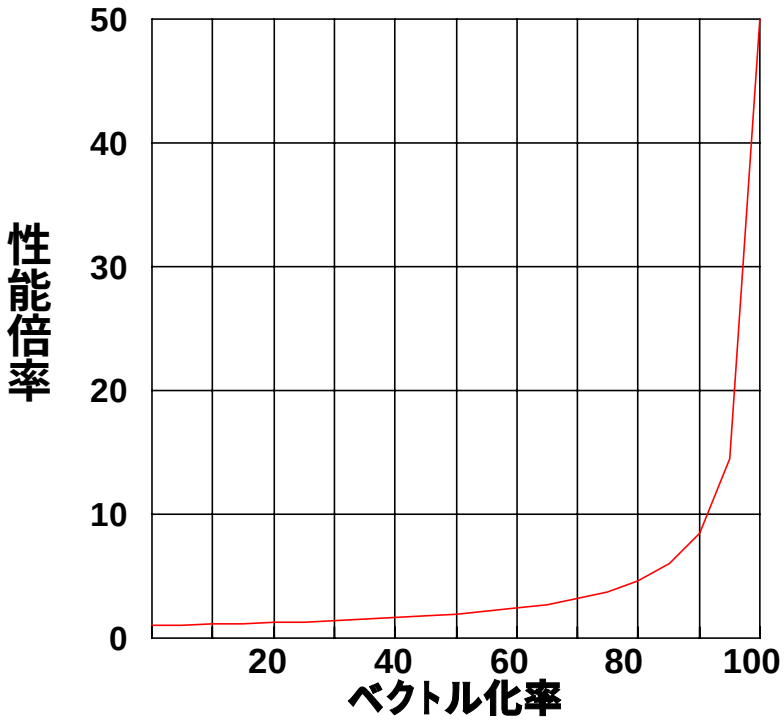
(3) **メモリアクセスを効率化する**
  ⇨ **バンクコンフリクトを削減する**

ベクトル化率と性能(アムダールの法則)

■ ベクトル化率

プログラム中のベクトル命令で実行可能な部分の割合

一般にベクトル化率を正確に求めることは困難であるため、SXではプログラム特性情報(proginf)に表示される、**ベクトル演算率**（ベクトル演算が実行された割合）で代用

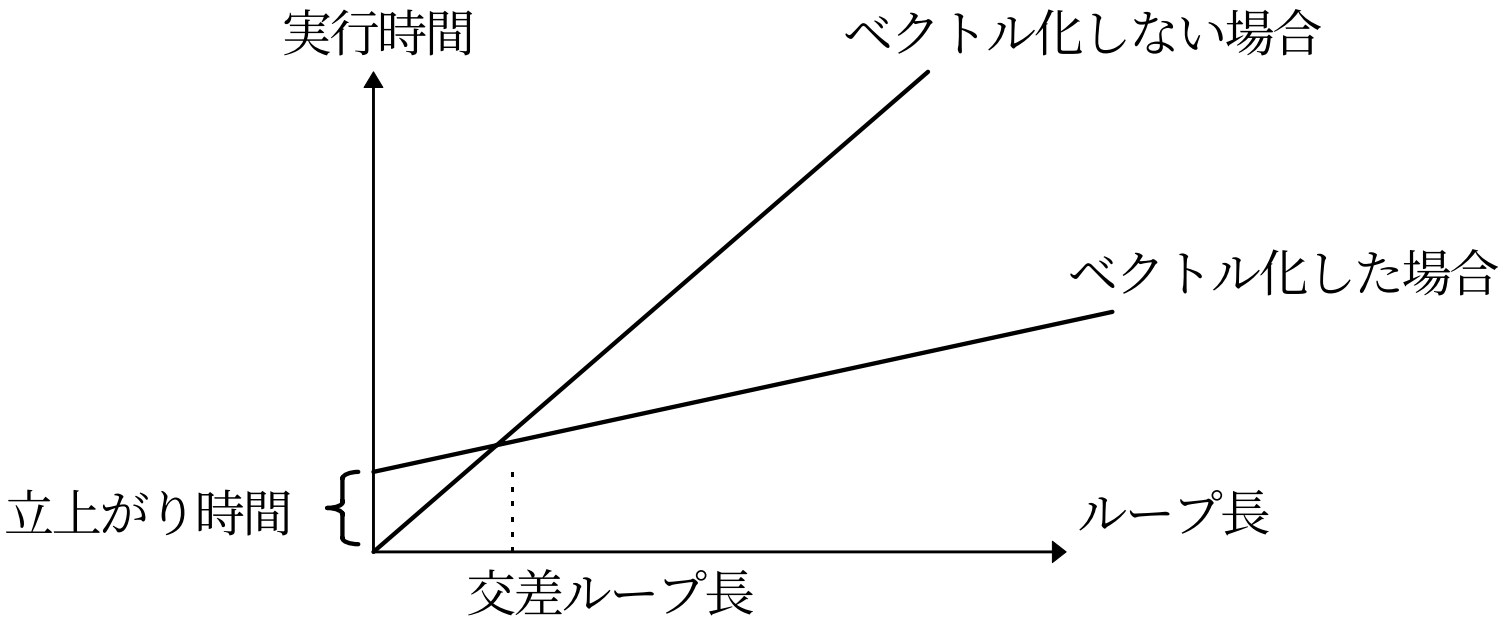


ベクトル化による性能向上比を50倍と仮定

ベクトル化率が50% → 全体での性能は2倍にしかならない。

ベクトル性能を十分引き出すにはベクトル化率を100%に近づける必要

ループ長 (ベクトル長)



ループ長をできるだけ長くした方が、ベクトル化による高速化の効果が大きい (交差ループ長 5 程度)

チューニングの手順

proginf情報からプログラム全体の性能を分析

↓

プロファイラ情報、ftrace情報から、チューニングすべき手続き(サブルーチン・関数)を選択

↓

ベクトル化診断メッセージ、編集リストから、チューニングすべきループ及び配列式を選択

↓

チューニングの実施(ベクトル化指示行の挿入、ソースプログラムの変更)

プログラム特性情報(PROGINF)の利用

setenv F_PROGINF YES **または** DETAIL**で表示**

```
******    プログラム   情報    ******
  経過時間（秒）                 :         6.774000
  ユーザ時間（秒）               :         4.911325
  システム時間（秒）             :         0.230647
  ベクトル命令実行時間（秒）     :         4.846089
  全命令実行数                   :       380150140.
  ベクトル命令実行数             :       205713526.
  ベクトル命令実行要素数         :     52468073111.
  浮動小数点データ実行要素数     :     14758745137.
  MOPS 値                        :     10718.595612
  MFLOPS 値                      :      3005.043295
  平均ベクトル長                 :       255.054075
  ベクトル演算率（%）            :        99.668639
  メモリ使用量（MB）             :      1820.031250
  MIPS 値                        :        77.402761
  命令キャッシュミス（秒）       :         0.014333
  オペランドキャッシュミス（秒） :         0.010467
  バンクコンフリクト時間（秒）   :         0.244283
```

→ ループ長は十分か？

→ ベクトル演算率は十分か？

→ バンクコンフリクト回避は必要か?

平均ベクトル長、ベクトル演算率、バンクコンフリクト時間に着目

簡易性能解析機能：ftrace

■ プログラム単位ごとに性能情報を採取

使用方法

```
% f90 -ftrace test.f90
% a.out
% ftrace
```

```
*--------------------------*
  FLOW TRACE ANALYSIS LIST
*--------------------------*

Execution : Mon Dec 16 15:36:40 2002
Total CPU : 0:00'07"732
```

呼び出したプログラムを含まない実行時間とその割合

呼び出し1回当たりのEXCLUSIVE TIME

呼び出し回数

proginf と同じ

PROG.UNIT	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONF
sub01	26	6.688(86.5)	257.212	6602.2	2593.9	98.64	216.3	6.661	0.0000	0.0116	0.0323
main_	1	1.006(13.0)	1005.635	682.1	83.8	44.44	250.1	0.043	0.0004	0.0003	0.0000
sub02	26	0.039(0.5)	1.500	3414.6	1332.6	97.52	203.0	0.026	0.0000	0.0099	0.0000
total	53	7.732(100.0)	145.890	5816.2	2261.1	97.81	216.5	6.730	0.0005	0.0218	0.0323

ベクトル演算率　平均ベクトル長　バンクコンフリクト時間

コンパイラ指示行

■ コンパイラ指示行

!CDIR **オプション［, オプション**]

配列の定義･引用関係に矛盾が無いことなど、コンパイラが知り得ない情報を、指示行の形式で与えることにより、ベクトル化/並列化の効果を促進させる

■ 主なベクトル化指示オプション

- VECTOR/NOVECTOR　：**ベクトル化の対象とする／しない**
- NODEP　：**参照する配列要素に重なりがない**
- LOOPCHG/NOLOOPCHG　：**ループ入れ換えを行う／行わない**

指示行の挿入によるベクトル化

ソースプログラム

```
DO I=1, N
   A( IX(I) ) = A( IX(I) ) + B(I)
END DO
```

-Wf“-pvctl fullmsg” **を指定することにより、以下のメッセージが出力される**

f90: vec(1): test.f, line 5: **ベクトル化できないループである。**
f90: opt(1036): test.f, line 6: **異なる繰り返しで定義された値を参照している可能性がある。**(nosync/nodepを指定すれば最適化を行う)
F90:vec(22): test.f, line 6: a**は依存関係の解析ができないためベクトル化不可の依存関係が存在すると仮定する。**

```
!CDIR NODEP
  DO I=1, N
     A( IX(I) ) = A( IX(I) ) + B(I)
  END DO
```

配列IX(I)**の値に、重複した要素のないことがわかっているならば指示行**NODEP**を挿入することでベクトル化できる**
(例: IX(I)**が、**9,3,2,4,1,5,7,10,8,.....)

インライン展開によるループのベクトル化

- **ユーザ手続き呼び出しを含むループはベクトル化されない**
  コンパイル時オプション -pi を指定すると、インライン展開可能なユーザ手続きを、呼び出し元にインライン展開する。
  ループ中に手続きの呼び出しがあれば、展開後にベクトル化を行う。

ソースプログラム

```
DO I=1, N
  A(I) = FUN(B(I), C(I)) + D(I)
END DO
```

```
FUNCTION FUN(X, Y)
FUN = SQRT(X) * Y
END FUNCTION FUN
```

```
f90: vec(3): test.f, line 3: ベクトル化できないループである。
f90: opt(1025): test.f, line 4: 最適化を阻害する関数呼出しがある。
f90: vec(10): test.f, line 4: ループまたは配列式全体をベクトル化不可とする手続funが指定された
```

-pi を指定してFUNをインライン展開

```
DO I=1, N
  A(I) = SQRT(B(I)) * C(I) + D(I)
END DO
```

ベクトル化される

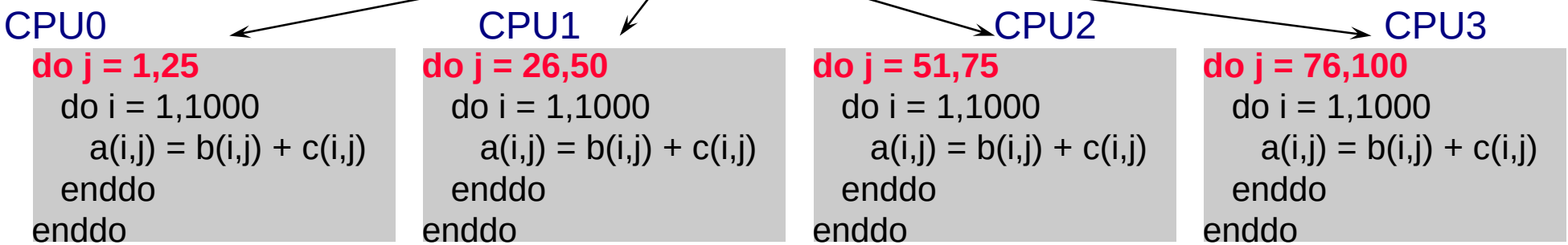
並列化チューニング

自動並列化

- コンパイラが、並列実行可能なループや文の集まりを抽出し、ループの繰り返しなどを複数のタスクに割り当てて実行する機能

```
do j = 1,100
  do i = 1,1000
    a(i,j) = b(i,j) + c(i,j)
  enddo
enddo
```

-Pauto オプション
を指定してコンパイル



並列化における高速化の観点

- **並列化率**

  シリアル実行した場合の実行時間に対する、並列実行可能部分の実行時間の割合

  - プログラムの主要なループが並列化されているか？
- **並列化効率**
  - 並列化されているループの実行時間は十分大きいか？
  - 並列処理のオーバーヘッドが大きくないか？
  - 各タスクの負荷バランスは均一か？

並列化のチューニング手順

0. ベクトル性能チューニング
   - 並列化の前に、ベクトル化のチューニングを完了させておく
1. 並列対象ループ／サブルーチンの抽出
   - プロファイラ/ftrace情報から、コストの大きいループを含むサブルーチンを見つけ出す
2. 並列化率の向上
   - 1. で選択した手続中の並列化されていないループを並列化できないか調査し、必要なら指示行の指定、ソースの修正を行う
3. 並列化オーバーヘッドの削減
   - 並列化のオーバーヘッドが大きくなっていないか調査し、必要ならオプションの指定を変更する
4. 負荷バランスの改善
   - 各タスクに処理が均等に割り当てられるよう、バランスを調整する

並列化時のPROGINF情報

```
******  Program Information  ******
 Real Time (sec)       :          49.384518
 User Time (sec)       :         186.987137
 Sys  Time (sec)       :           1.182086
 Vector Time (sec)     :         150.553135
 Inst. Count           :        5309765101.
 V. Inst. Count        :        3377473534.
 V. Element Count      :     1418449397463.
 FLOP Count            :      575659440969.
 MOPS                  :        7596.146515
 MFLOPS                :        3078.604498
 MOPS   (concurrent)   :       29327.554103
 MFLOPS (concurrent)   :       11886.018758
 VLEN                  :         419.973505
 V. Op. Ratio (%)      :          99.863960
 Memory Size (MB)      :         464.000000
 Max Concurrent Proc.  :                  4.
  Conc. Time(>= 1)(sec):          48.431645
  Conc. Time(>= 2)(sec):          48.346999
  Conc. Time(>= 3)(sec):          47.825707
  Conc. Time(>= 4)(sec):          42.387946
 Event Busy Count      :                  0.
 Event Wait (sec)      :           0.000000
 Lock Busy Count       :                  0.
 Lock Wait (sec)       :           0.000000
 Barrier Busy Count    :                  0.
 Barrier Wait (sec)    :           0.000000
 MIPS                  :          28.396419
 MIPS (concurrent)     :         109.634209
 I-Cache (sec)         :           0.162070
 O-Cache (sec)         :           0.744901
 Bank (sec)            :           0.170449
```

経過時間（秒）
ユーザ時間（秒）
システム時間（秒）
ベクトル命令実行時間（秒）
全命令実行数
ベクトル命令実行数
ベクトル命令実行要素数
浮動小数点データ実行要素数
MOPS 値
MFLOPS 値
MOPS 値　（実行時間換算）
MFLOPS 値（実行時間換算）
平均ベクトル長
ベクトル演算率（%）
メモリ使用量（MB）
最大同時実行可能プロセッサ数
　1台以上で実行した時間（秒）
　2台以上で実行した時間（秒）
　3台以上で実行した時間（秒）
　4台以上で実行した時間（秒）
イベントビジー回数
イベント待ち時間（秒）
ロックビジー回数
ロック待ち時間（秒）
バリアビジー回数
バリア待ち時間（秒）
MIPS 値
MIPS 値（実行時間換算）
命令キャッシュミス（秒）
オペランドキャッシュミス（秒）
バンクコンフリクト時間（秒）

PROGINFを用いた性能分析

- Conc. Time(>=1)と比べ、Conc. Time(>=2)が小さい

```
Conc. Time(>= 1)(sec):          74.154168
Conc. Time(>= 2)(sec):           8.549322
Conc. Time(>= 3)(sec):           8.292376
Conc. Time(>= 4)(sec):           8.071275
```

  - 並列化率が低い⇨ 並列化されていないループを並列化

- Conc. Timeの値に偏りがある

```
Conc. Time(>= 1)(sec):          69.503482
Conc. Time(>= 2)(sec):          58.271920
Conc. Time(>= 3)(sec):          33.497481
Conc. Time(>= 4)(sec):          12.927761
```

  - 負荷バランスが悪い ⇨ ループ並列実行方法の変更

並列化率向上のための技法

- **並列化阻害要因の除去**
  - **診断メッセージから並列化阻害要因を知る**
    ｵﾌﾟｼｮﾝ -Wf"-pvctl fullmsg"
- **依存関係が不明で並列化しない場合のメッセージ**

ﾒｯｾｰｼﾞ No.	メッセージ
1033	同一の配列要素に対して定義が複数回行われる可能性がある
1036	異なる繰り返しで定義された値を参照している可能性がある (nodep/nosyncを指定すれば最適化を行う)

➔ 依存関係が並列化可能かどうかコンパイラが判定できない
- 依存関係がない ⇨ NOSYNC 指示行を指定
- 並列化不可の依存関係がある⇨ プログラム修正

指示行による並列化促進

- NOSYNC **指示行**

  ループ中の配列要素が繰り返しにまたがって定義･参照されない（＝並列化しても問題ない）ことを指定する

  例：a(I,k1,j+1)**と**a(I,k2,j) **の依存関係が不明**
  K1≠K2**であることが保証できるなら**a(I,k1,j+1)**と**a(I,k2,j) **が同じ要素となることはない**
  → nosync**を指定して並列化可能**

```
!cdir nosync
 do j = 1, ny
      do i = 1, nx
           a(i,k1,j+1) = a(i,k2,j) + b(i)
      enddo
 enddo
```

並列化

OpenMPによる並列化

OpenMP

- **共有メモリマシン向け並列処理の標準API**
  - **異なる共有メモリアーキテクチャを持つベンダ間で 可搬なプログラミングモデルを提供**
  - **並列化は利用者がすべて明示的に記述**
    - **ディレクティブで動作を指示**
    - **実行時ライブラリ、環境変数も用意されている**

参考：OpenMP**に関する**Web**情報** http://www.openmp.org/

- FORTRAN90/SX**での使用方法**
  - **オプション** -Popenmp **を指定してコンパイル･リンク**

例： `% f90 -Popenmp prog.f90`

注意： **自動並列化関連のコンパイルオプションは同時に指定できない**
ソース中の自動並列化関連のコンパイル指示行は無効となる

OpenMPの仕様

- **ディレクティブ**

  !$OMP *ディレクティブ名 [clause[[,]clause]…]*

  - “!$OMP” **は1カラム目から空白なしに記述**
  - **固定形式の場合は**“*$OMP” **または** “C$OMP” **も使用可**

  - **主なディレクティブ**

パラレルリージョン構造

```
PARALLEL/END PARALLEL
```

同期構造

```
CRITICAL/END CRITICAL
BARRIER
ATMIC
```

Work-Sharing構造

```
DO/END DO
SECTIONS/SECTION/END SECTIONS
SINGLE/END SINGLE
```

データスコープ

```
THREADPRIVATE
```

OpenMPの例

```
!$OMP PARALLEL DEFAULT(PRIVATE) SHARED(field, ispectrum)
        call initialize_field(field, ispectrum)
        call compute_field(field, ispectrum)
        call compute_spectrum(field, ispectrum)
!$OMP END PARALLEL
            ……
        subroutine initialize_field(field, ispectrum)
            ……
!$OMP DO
        do i = 1, nzone
          ispectrum(i) = 0
        enddo
!$OMP END DO NOWAIT
!$OMP DO
        do j = 1, npoints
          field(j) = 0
        enddo
!$OMP END DO
!$OMP SINGLE
        field(npoints/4) = 1.0
!$OMP END SINGLE
        return
        end
```

並列実行を行う範囲を指定
※共有変数に関する排他
制御等は利用者が制御

DOループを並列実行

1スレッドのみ実行

C++/SX

C++/SXの特長

言語の特長

- ISO/ANSI C++規格準拠
  - STL (Standard Template Library)
  - 例外処理 (try catch, throw)
  - RTTI (Run-Time Type Identification)
  - “inline”指定子によるインライン展開
- ISO/ANSI C 規格準拠
- 64bit整数、64bitポインタのサポート

チューニング支援機能

- 手軽に性能情報の採取可能
  ﾌﾟﾛｸﾞﾗﾑ特性情報出力
  簡易性能解析機能

最適化機能

- 手続きのインライン展開
- 分岐を越えた命令並べ換え

強力な自動ベクトル化/自動並列化機能

- ループ最適化
  (入れ換え、一重化、ｱﾝﾛｰﾘﾝｸﾞ等)
- 条件ベクトル化／条件並列化

OpenMP C/C++

- 共有メモリマシン向け並列処理の標準 API をサポート

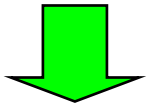
C++/SXのベクトル化/並列化

■ ベクトル化/並列化の例

多重ループの一重化

ソースプログラム

```
float a[100][100][100], b[100][100][100]
for( i=1; i < n-2; i++ ){
  for( j=0; j < 100; j++ ){
    for( k=0; k < 100; k++ ){
      a[i][j][k] = b[i][j][k];
    }
  }
}
```



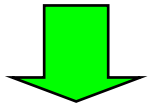
コンパイラによる変形のイメージ

```
for( i=0; i < n*10000-30000; i++){
  a[1][0][i] = b[1][0][i];
}
```

ベクトル＋並列化

ソースプログラム

```
for( j=0; j < 1000; j++ ){
  for( i=0; i < 10; i++ ){
    b[i][j] = 2.0 + a[i][j]*3;
  }
}
```



コンパイラによる変形のイメージ

```
for( i=0; i < 10; i++ ){        並列
  for( j=0; j < 1000; j++ ){    ﾍﾞｸﾄﾙ
    b[i][j] = 2.0 + a[i][j]*3;
  }
}
```

ﾙｰﾌﾟを入れ換え、内側をﾍﾞｸﾄﾙ化、
外側を並列化

C++/SX: コマンドの形式

- **形式**

  c++　[**オプション|ファイル名**] …　(**セルフコンパイラ**)

  sxc++　[**オプション|ファイル名**] …　(**クロスコンパイラ**)

  - **ファイルのサフィックス**

    .c　C**ソースとしてコンパイル**

    .C, .cpp, .cc, .cxx, .CXX　C++**ソースとしてコンパイル**

    } **オプションで変更可能**

- **代表的なコンパイルオプション**

  - **ベクトル化レベルの指定**

    sxc++ -Chopt a.c b.c

  - **自動インライン展開**

    sxc++ -pi,auto a.c b.c

  - **自動並列化**

    sxc++ -Pauto a.c b.c

ベクトル化レベル

-Caopt　**高**

-Chopt

-Cvopt

-Cvsafe　**低**

MPI/SX

MPI/SXの特長

- MPI-2 仕様フルサポート
  - 分散並列処理向けメッセージ通信操作の標準仕様
  - 特定プラットフォームに依存しない共通API仕様
  - C, C++, Fortran プログラムから呼び出し可能なライブラリ手続きインタフェース
  - MPI-2 は MPI-1 に以下の機能を追加
    - 片側通信
    - 動的ﾌﾟﾛｾｽ生成
    - MPI-IO
- 世界最高レベルのレイテンシー、スループット性能
  - HW命令による高速同期
  - シングルコピー通信によるスループット性能強化

MPIプログラミング例

```
include ”mpif.h”
real A(10)
integer st(MPI_STATUS_SIZE)
integer rank
integer err
call MPI_Init(err)          !初期化
call MPI_Comm_rank(MPI_COMM_WORLD,rank,err) !ランク取得
if (rank.eq.0) then
   call MPI_Send(A,4,MPI_REAL,1,123,MPI_COMM_WORLD,err)
else if (rank.eq.1) then
   call MPI_Recv(A,4,MPI_REAL,0,123,MPI_COMM_WORLD,st,err)
endif
call MPI_Finalize(err)  !終了
end
```

ランク0のMPIプロセスからランク1のプロセスへ
実数型データ４要素を転送

MPI プログラムのコンパイルとリンク

- MPI/SXの提供するシェルスクリプトを利用
  - mpif90, mpic++ (セルフ環境)
  - sxmpif90, sxmpic++ (クロス環境)
  - MPIインクルードファイル、MPIライブラリの明示的指定不要
  - クロス環境での利用例

    sxmpif90 fsrc.f90
- コンパイラを直接利用
  - MPIインクルードファイルおよびMPIライブラリの指定必要

    -I/usr/include … -lmpi (セルフ環境)

    -I/SX/usr/include … -lmpi (クロス環境)
  - クロス環境での例

    sxf90 -I/SX/usr/include fsrc.f90 -lmpi

MPI プログラムの実行

■ mpirun コマンドを利用

◆ 直接指定形式

- SPMD (Single-Program Multiple-Data) モデルの実行

mpirun [*オプション*] *実行プログラム名* [*引数*]

例：mpirun -np 8 a.out　　*a.out を 8 プロセスで実行*

◆ 間接指定形式

- MPMD (Multiple-Program Multiple-Data)モデルの実行
- SPMD モデルの MPIプログラムで，プロセス毎に異なる引数を与える場合

mpirun [*オプション*] -f *実行定義ファイル名*

例：mpirun -f run.conf

run.conf

```
-p 4  -e /home/progA arg1
-p 2  -e /home/progB arg2 arg3
```

progA を4プロセス
progB を2プロセス

E N D