TX7 プログラム　チューニング

2003.3.12

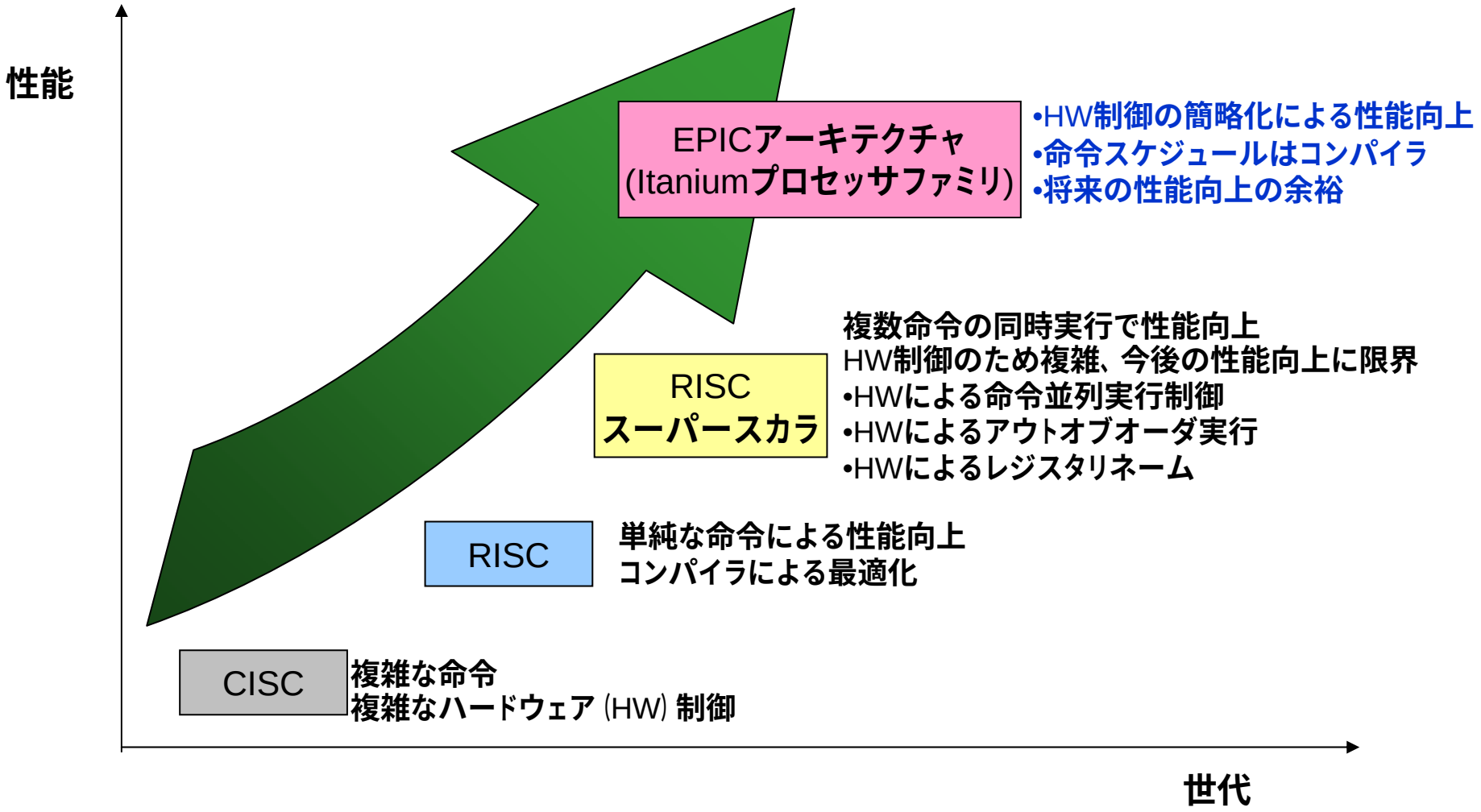
第一コンピュータソフトウェア事業部

NEC

内容

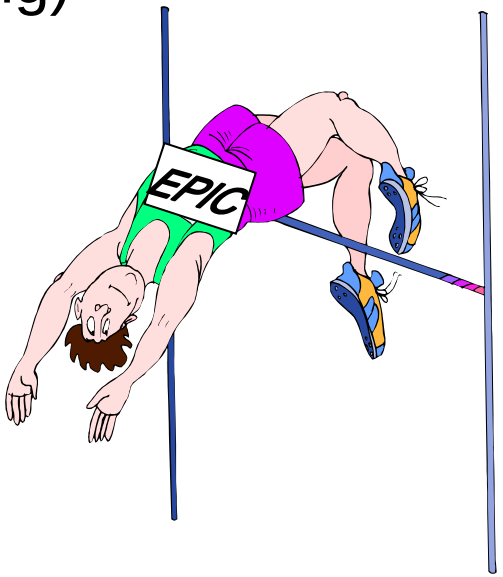
- Itanium2ハードウェアアーキテクチャ
- Fortranの言語仕様と使い方
- 自動並列化機能
- 性能チューニング

プロセッサアーキテクチャの変遷

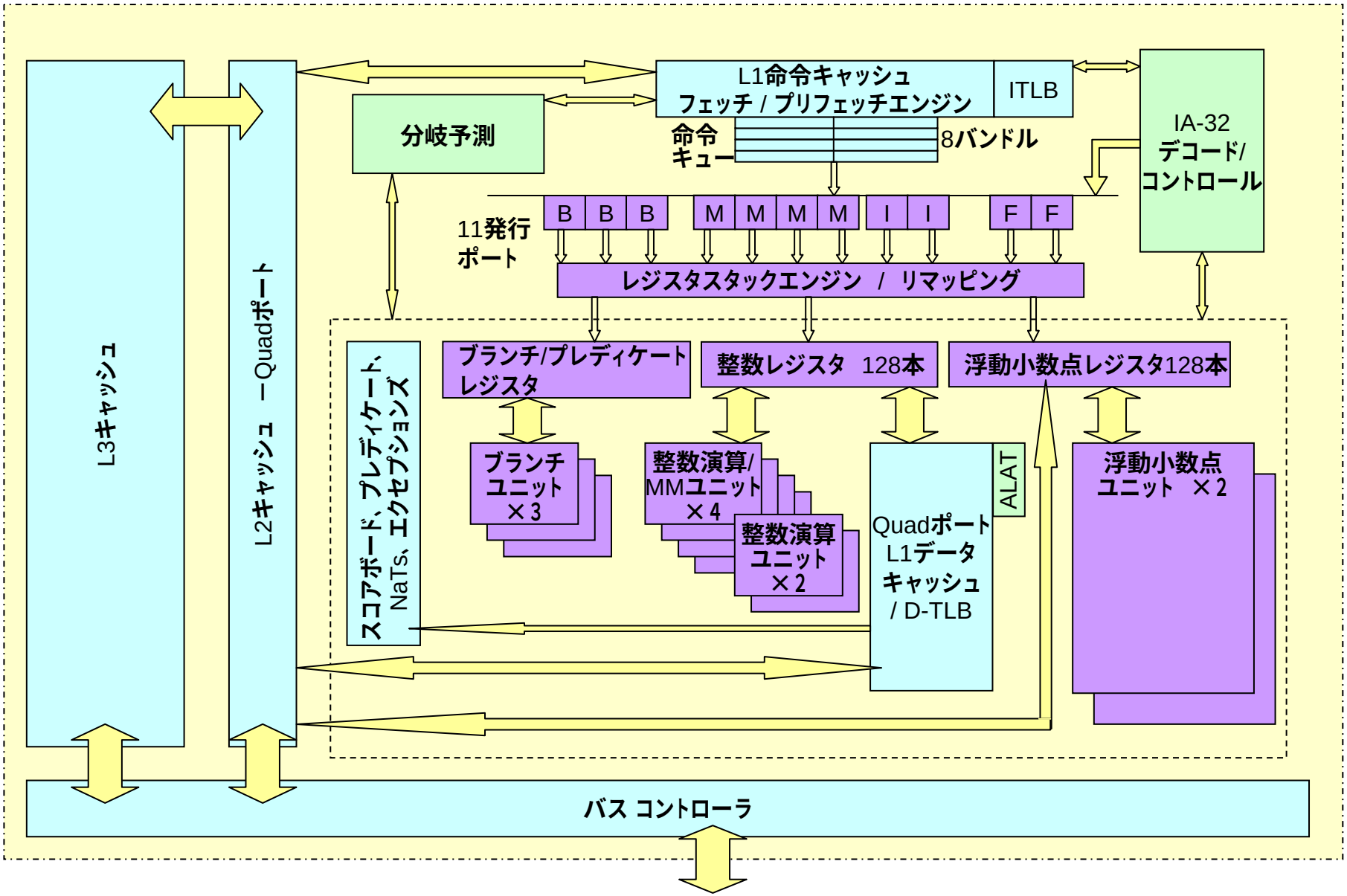


Itanium Processor Family (IPF) ～RISCを越えて

- EPIC(Explicitly Parallel Instruction Computing)
  - **明示的並列：コンパイラが命令をスケジュールする**
  - HW**が簡略化される。オーバーヘッドも小さい。**
- **ブランチへの対応**
  - **ブランチ命令を削除できる**Predicate**命令の採用**
- CPU**とメモリの速度差への対応**
  - **レイテンシーを隠蔽するための、プリフェッチ命令と** Control Speculation、Data Speculation**命令**
  - **キャッシュ階層の制御（指定階層にデータを格納）**
- **レジスタ**
  - **手続き呼び出しを高速に行うレジスタスタック**
  - **レジスタローテーション機能：ソフトウェアパイプライニングの容易な適用**



Itanium2　ブロック図



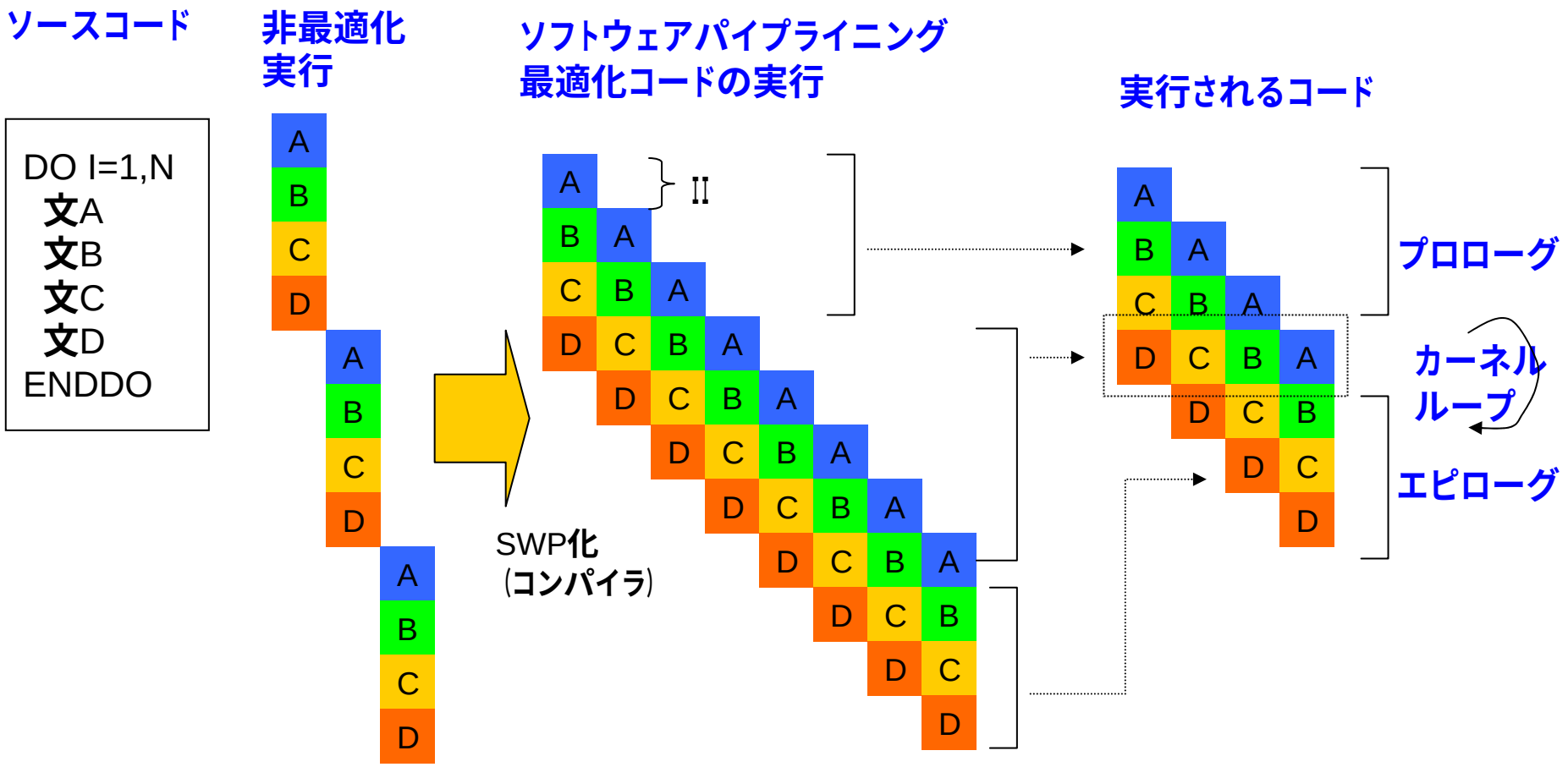
Itanium2　キャッシュ

	L1命令 キャッシュ	L1データ キャッシュ	L2 キャッシュ	L3 キャッシュ
容量	16KB	16KB	256KB	3MB
ラインサイズ	64B	64B	128B	128B
レーテンシー (**サイクル数**)	1	**整数**:1	**整数**:5 **浮動小数**:6	12〜
バンド幅(fill)	32GB/s	32GB/s	32GB/s	32GB/s

Itanium: 浮動小数演算

- IEEE規格準拠
  - 単精度(32ビット)、倍精度(64ビット)、拡張倍精度(80ビット)をHWでサポート [拡張倍精度はコンパイラではサポートしていない]。
  - レジスタ上は82bit高精度計算
  - 他システムとの互換性のため四倍精度 (128ビット) をソフトウェアでサポート (Fortranのみ) 。
- Multiply&Add命令
  - 積和命令。この命令は1クロックで処理されるため、1クロックで2バンドル*2演算=4浮動小数演算同時実行可能。
  - クロックを1GHzとして、理論最大性能は4GFLOPS (倍精度) となる。
- 多数の浮動小数レジスタ
  - 128個のレジスタ=32個固定、96個rotating
- 除算命令・平方根命令
  - 除算や平方根 (の近似値) を算出する命令がある。

ソフトウェア パイプライニング (SWP)

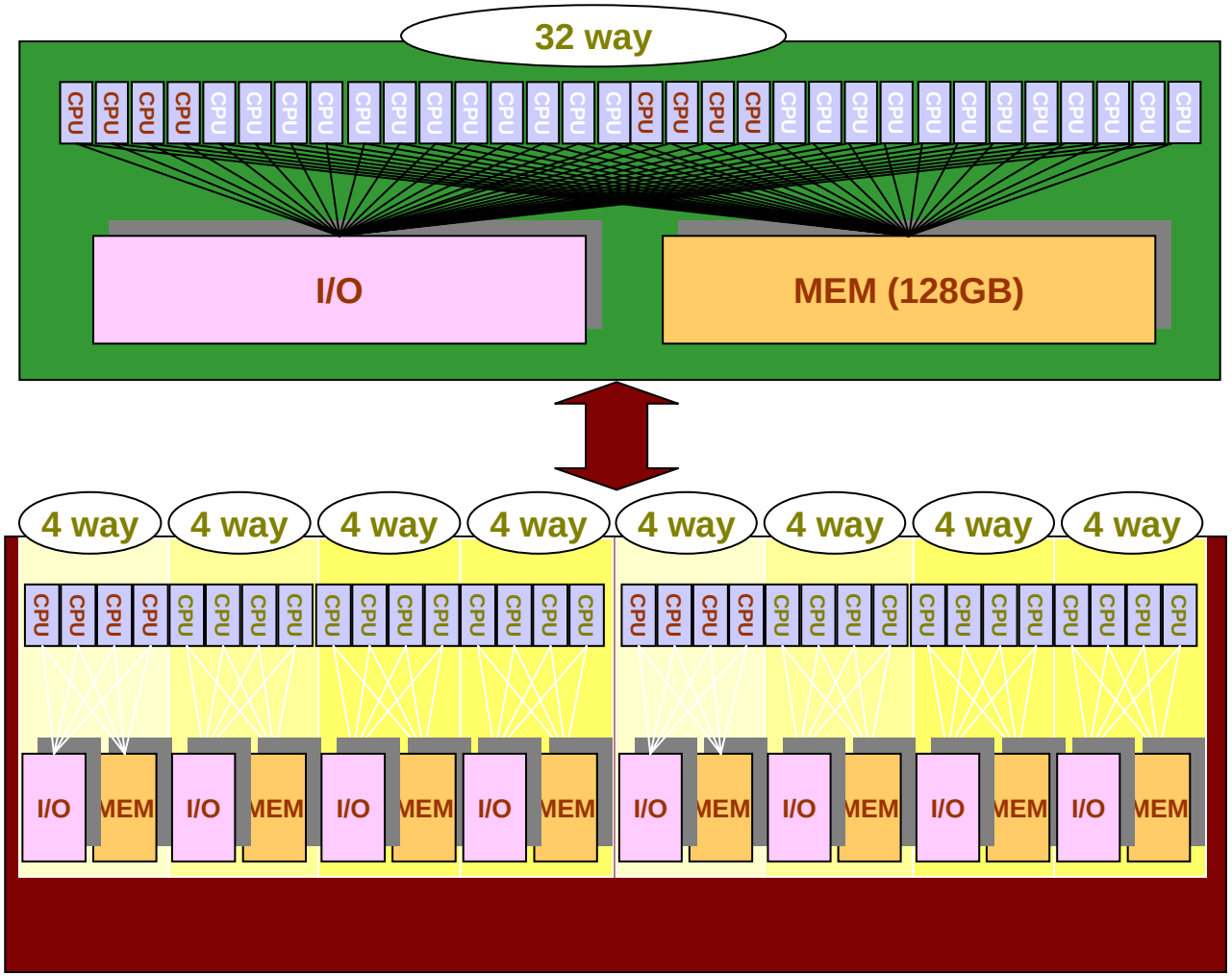


ループの1つの繰り返し内のA,B,C,Dには依存関係があることが多く、命令の並列実行が出来ない

しかし他の繰り返しの文と組み合わせれば並列に実行できることが多い。これを利用したもの。(ベクトル化に似ている)

TX7 i9000シリーズ

32 Itanium2 (**最大**)
128GB**メモリ**(**最大**)
112　I/O**スロット**(**最大**)



4way SMP (**セル**) *8**の**
ccNUMA

ただし、ローカル/リモート
メモリアクセスレーテンシー
の差は小さい。

Itanium/Linux　Fortran

言語仕様とコンパイラ利用方法

NEC Fortran言語仕様

Fortran95標準(ISO/IEC 1539:1995) ＋ 以下の拡張

- #で始まる注釈行、d/Dで始まるデバッグ行
- 名前に$を使用可能（先頭を除く）
- 名前は255文字まで使用可能。（標準Fortranは31文字まで）
- 文の追加
  - 記憶域指定：automatic, static,
  - 最適化抑止：volatile
  - 入出力：encode, decode, define file
  - VAX Fortran互換：record, structure, union, map
  - CRAY*互換：pointer
  - byte文
- 固定形式、自由形式に加え、拡張固定形式(132カラム有効)
- 手続き呼び出しの拡張
  - %VAL関数と%REF関数
    - %VAL関数による値渡し(call by value)、Cの関数を呼ぶときに使える。
    - %REF関数による参照渡し(call by reference)。通常の呼び出しとほぼ同じ、ただし文字型では隠れ引数（文字長）が渡されない。

```
call Croutine(%VAL(N))
```

NEC Fortran: **指示行**

- IVDEP

  !DIR$ IVDEP : **ループの直前に指定。依存関係が無いことを示す。**

  本来ベクトル化の指示行（Ignore Vector DEPendency)**であるが、各種最適化にも利用される。**

```
!DIR$ IVDEP
      DO I=1,N
       A(I)=A(I+K) + B(I)
      ENDDO
```

  IVDEP **指示行を入れることによって、最適化（ソフトウェアパイプライニング）が促進される。**

- **その他の指示行**（!DIR$ **の後に書けるもの**）
  - unroll [**数値**] / nounroll : **ループのアンローリングを指示**
  - prefetch [**配列名**,..] / noprefetch : **プリフェッチを制御**
  - swp / noswp :: swp**を制御**
  - distribute point : **ループ分割点を指定**
  - loop count (**数値**) : **ループ長を指定**

NEC Fortran, C/C++コンパイラ

- Intel社のItaniumコンパイラ技術をベースにNECが開発、保守
- Itaniumの性能をフルに引き出す最適化技術の集積
  - 命令並列スケジューリング
  - ソフトウェアパイプライニング
  - 各種ループ変形 (ループ入れ換え、一重化、融合、アンロール)
  - キャッシュの有効利用 (プリフェッチ)
  - 手続き間最適化、手続きのインライン展開
  - 実行時プロファイルに基づく最適化
- 使いやすさを追求
  - ISO標準準拠、拡張機能・他社互換機能
  - 64ビットアドレッシング、大規模プログラムへの対応
  - OpenMP2.0並列処理
  - 自動並列化
  - Big-Little エンディアン変換 (Fortranバイナリファイル)
    - » F_UFMTENDIAN=u[,u]
  - ハードウェアパフォーマンスモニタ表示機能(ftrace)

Fortranコンパイラ(efc)最適化オプション

最適化レベル:
- -O0　最適化を行わない
- -O1　-O2の最適化の内、コードサイズが増大する最適化は行わない。
- -O2　最適化を行う (DEFAULT)
- -O3　-O2 に加えて、さらに高度の最適化を行う

浮動小数関連:
- -mp　浮動小数計算の精度を完全に保つ (精度が変わる最適化を抑止)
- -mp1　浮動小数計算の精度をほぼ保つ (-mpに比べて、性能低下小)
- -ftz　デノーマル数を0として処理する。(-O3の場合の既定値)

手続き間最適化関連:
- -ip　単一ファイル内での手続き間最適化を行う
- -ipo　複数ファイル (ファイル間) の手続き間解析、インライン展開を行う。

プロファイル利用最適化 (PGO) **関連**:
- -prof_gen　PGOのための計測実行コードを挿入する
- -prof_use　計測されたプロファイルを用いてPGOを行う

並列処理 (SMP):
- -openmp　OpenMP指示行を有効にし、並列化されたコードを生成する。
- -parallel　自動並列化を行なう

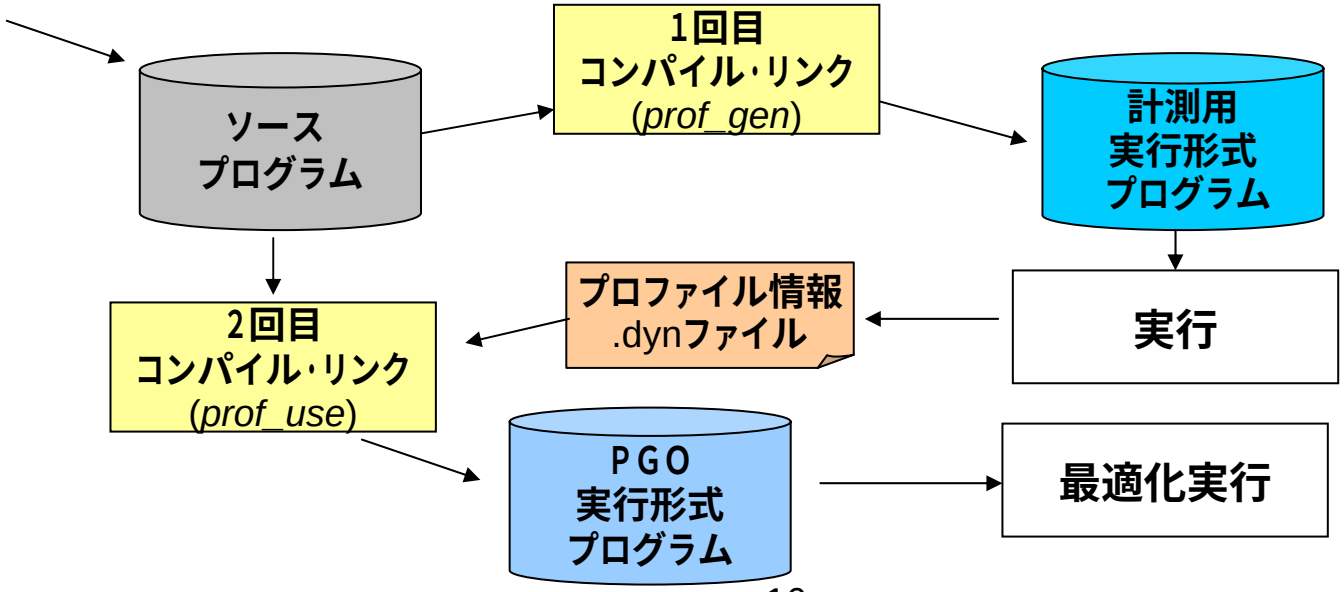
その他:
- -opt_report　最適化レポートの出力
- -opt_report_phase 最適化レポートのうち特定の最適化情報のみ出力
- -unroll0　ループアンローリングを行わない

最適化レベルの詳細

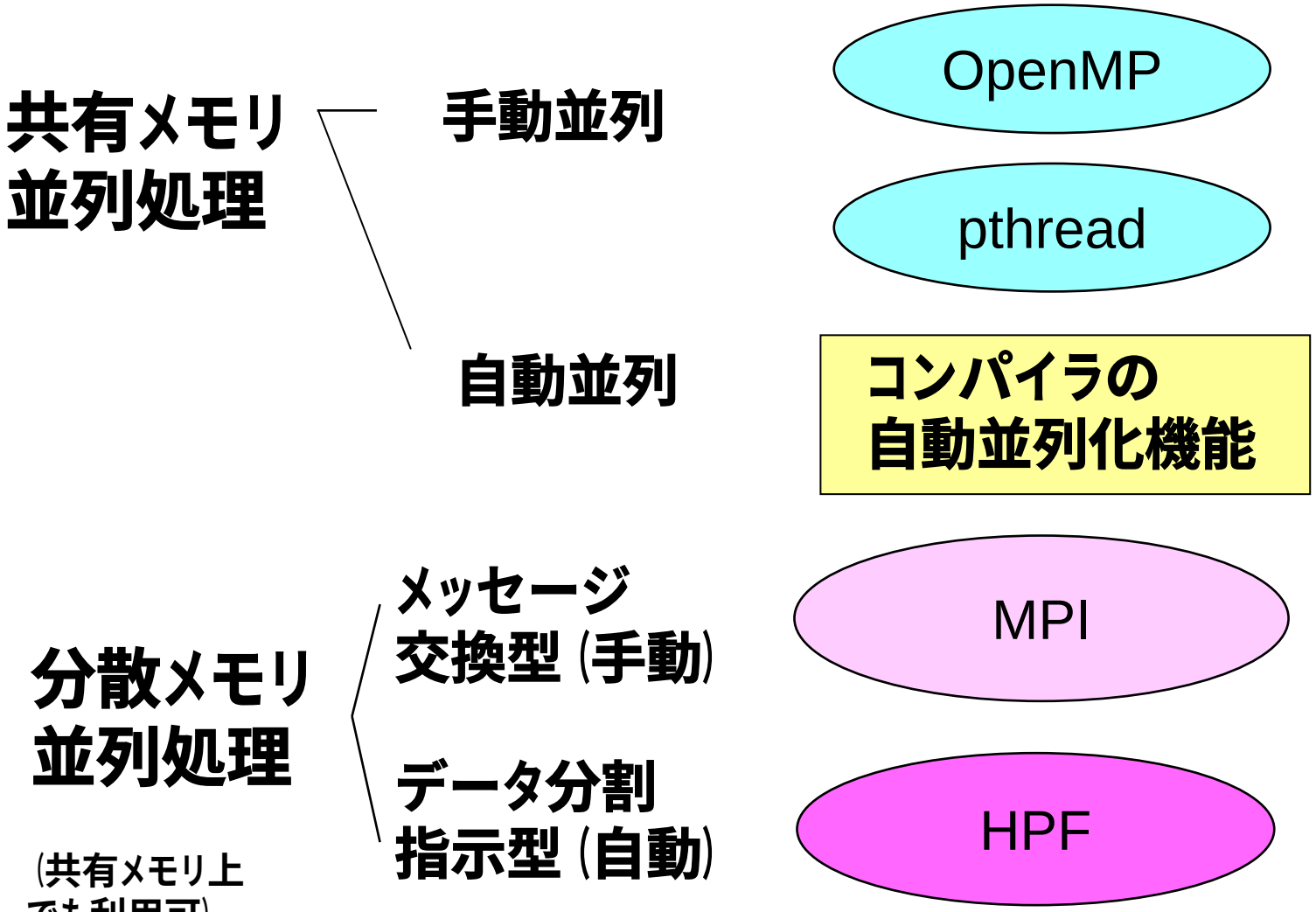
- -O0: **最適化しない**
- -O1: **最適化を行う。コードサイズが増大するものは行わない。**
  - **コンパイル時の定数計算**
  - **共通部分式削除**
  - **不要コードの排除**
  - **広域レジスタ割り当て**
  - **広域命令スケジュール**
  - **コントロール･スペキュレーションの利用**
  - **プレディケーションの利用**
- -O/-O2:**一般的な最適化を行う。(既定値)**
  - **ループのアンロール**
  - **ソフトウェアパイプライニング**
- -O3: **高度の最適化を行う**
  - **ループレベル最適化**
    - **ループ分割、ループ融合、ループアンローリング、ループ入れ換え**
  - **スカラーリプレースメント**
  - **データプリフェッチ**
  - **ペアロードの使用**

プロファイルに基づく最適化(PGO)

- -prof_gen:
  - **最初に、コンパイラがプログラムのどのパスが実行されやすいかを計測するコードを埋め込んでコンパイル、リンクし、計測実行**
- -prof_use:
  - **再度コンパイルする際、計測結果を利用して、最適化の精度を高める。**
    - **実行頻度の高い手続きを展開**
    - **SWPを行う判断基準の精密化**



並列処理プログラミングモデル



自動並列化の基本方針

- 多重ループの場合は、可能な最も外側ループが並列化、内側ループはSWP化
- 一重ループの場合は、基本的には、ループを分割し並列化＋SWP化を行う。

これにより最大限の性能を引き出す

```
        subroutine c2sub(a,h)
        real a(600,100,100)
        integer h(100)
P       do i = 1,100
          do j = 1,100
SWP         do k=1,599
              a(k,j,i) =(a(k+1,j,i) +a(k,j,i))*0.5
            enddo
          enddo
          h(i)=h(i)+1
        enddo
        end
```

```
P+swp   do i=1,n
          a(i)=sqrt(b(i)**2+c(i)**2)
        enddo
```

P：パラレル実行，
SWP：ソフトウェアパイプライン実行

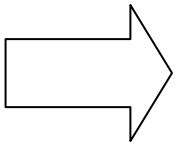
自動並列化コンパイルオプションと指示行

- -parallel
  - 自動並列化を行う
- -par_report[0|1|2|3]
  - 自動並列化メッセージを出力
  - -par_report0: メッセージを出力しない
  - -par_report1(既定値)：並列化されたループを表示
  - -par_report2:並列化されたループとされていないループを表示
  - -par_report3: report2に加えて並列化不可要因(依存関係)を表示
- -par_threshold[n]
  - 自動並列化のしきい値(確信度)を設定(nは0から100まで)
  - 0: 常に並列化する、100：性能向上が確実なループのみ並列化する。
  - 既定値は75。
- **指示行**
  - !dir$ parallel
    - 多重ループ中の指定のループが並列化可能なら、並列化する。
    - データ依存関係がコンパイラに不明な場合、並列化可能とする。
  - !dir$ noparallel
    - ループを並列化しない。
    - 多重ループの外側ループにnoparallelを指定すると内側ループも並列化対象外となる

自動並列化指示行の例

- parallel**指示行**

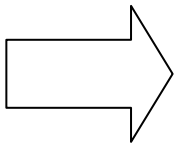
```
subroutine sub2(a,n,m)
real*8 a(n,m)
do j=1,m
 do i=1,n
  a(i,j)=sin(a(i,j))
 enddo
enddo
end
```



```
subroutine sub2(a,n,m)
real*8 a(n,m)
do j=1,m
!dir$ parallel
 do i=1,n
  a(i,j)=sin(a(i,j))
 enddo
enddo
end
```

do j**ではなく、**
do i**のループで**
並列化したい時

```
subroutine sub(a,m,n)
dimension a(n),m(n)
do i=1,n
  a(m(i))=a(m(i))+1
enddo
end
```



```
subroutine sub(a,m,n)
dimension a(n),m(n)
!dir$ parallel
do i=1,n
  a(m(i))=a(m(i))+1
enddo
end
```

配列a**の依存関係が**
不明だが、
指示行により
並列化される

自動並列化の環境変数

- **以下の環境変数が使用できる**（OpenMP**と共通**）
  - OMP_NUM_THREADS : **スレッド数を指定**（**既定値**：**構成**CPU**数**）
  - OMP_SCHEDULE：**ループのスケジューリング**（**既定値**：static）
  - KMP_STACKSIZE：**スレッドのスタックサイズを指定**（**既定値**:4M）
  - KMP_LIBRARY: serial | turnaround | throughput
    - serial: **シリアル実行用ライブラリ**
    - turnaround: **占有環境でのターンアラウンド優先。待ち状態で**sleep**せず** busy loop**しつづける。**
    - throughput:**共用環境でのスループット優先。一定時間待つとタスクが**sleep **する。（既定値）**

Itanium/Linux**向け**
プログラムチューニング

性能チューニングのポイント

1. 時間のかかっているサブルーチン、ループを見つける
2. ループのソフトウェアパイプライニングを可能な限り行うようにする
   - 指示行の利用
   - 文が多いループの分割＊
   - IF文の排除またはPGO(-prof_use, -prof_gen)の利用＊
   - 手続きのインライン展開(-ipo)＊
3. メモリアクセスの効率化
   - キャッシュの有効利用
     - » キャッシュブロッキング、配列宣言で2のべき乗を避ける
   - メモリアクセスそのものを減らす
     - » ループアンロール＊、外側ループアンロール＊、ループ融合＊、ループ入れ換え＊
   - 配列の連続アクセス化
     - » ループ入れ換え＊、配列を連続アクセスになるように詰め替え

＊コンパイラも行なうもの

性能表示ツール gprof

- -p オプションでコンパイル・リンク。
- 実行すると、gmon.outというファイルが出来る。
- $gprof [--no-graph] 実行ファイル [gmon.out]でルーチンごとの実行時間と実行回数が表示される。
- システムのルーチンも表示される。
- 一定時間ごとの割込みを用いて測定しているので、時間に多少誤差がある。

% time	cumulative seconds	self seconds	calls	self us/call	total us/call	name
37.42	0.06	0.06	133874	0.42	0.42	daxpy_
29.68	0.10	0.04				__mcount
17.42	0.13	0.03	270000	0.10	0.10	ran_
7.74	0.14	0.01	27	434.03	1410.59	matgen_
1.94	0.14	0.00	2574	1.14	1.14	idamax_
1.94	0.15	0.00	26	112.68	2357.21	dgefa_
1.29	0.15	0.00				_mcount_ret_helper
1.29	0.15	0.00				mcount
0.65	0.15	0.00	2574	0.38	0.38	dscal_
0.65	0.15	0.00	1	976.56	102539.06	main
0.00	0.15	0.00	72	0.00	0.00	second_
0.00	0.15	0.00	26	0.00	84.19	dgesl_
0.00	0.15	0.00	1	0.00	0.00	dmxpy_
0.00	0.15	0.00	1	0.00	0.00	epslon_

性能表示機能 ftrace

- コンパイルオプション -ftraceの指定で、サブルーチンごとの実行回数とHWパフォーマンスモニタの情報を実行終了時に表示する。
- 100以上のパフォーマンスイベントから4種類を選んで表示できる。既定値は、FTRACE_EVENT="CPU_CYCLES FP_OPS_RETIRED IA64_INST_RETIRED INST_DISPERSED"

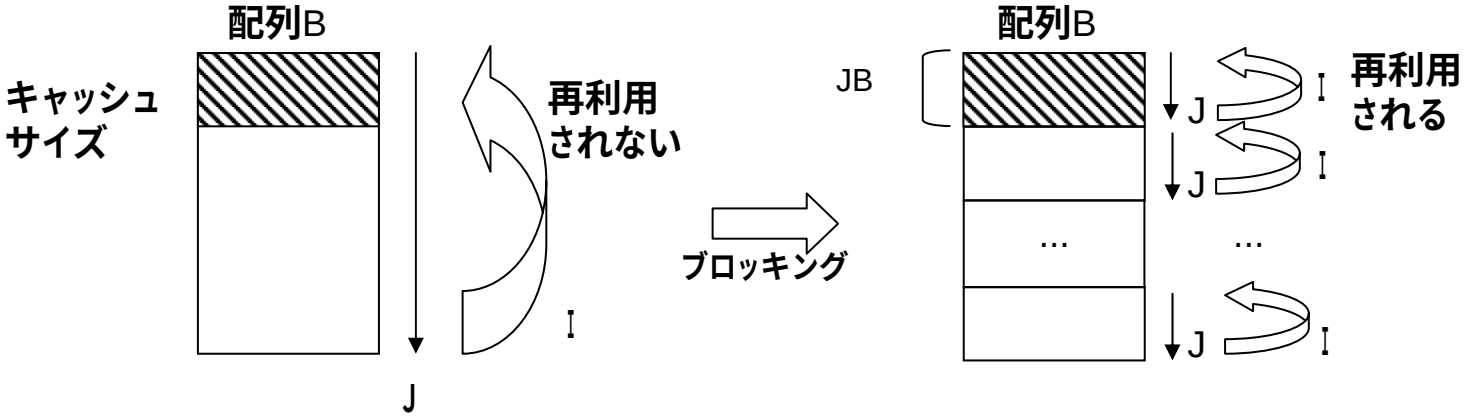
```
- Total -
Frequency       CPU_CYCLES   FP_OPS_RETIRED IA64_INST_RETIRE   INST_DISPERSED Routine
--------- ---------------- ---------------- ---------------- ---------------- ---------------
        1           180351             8640           167754           189060 main
       27          8873191          2700000         36782625         37940003 matgen_
       26         49309586         17520418         83248961         92884461 dgefa_
       26          1785156           577174          2853269          3202644 dgesl_
        1            10372            20000            34278            36658 dmxpy_
--------- ---------------- ---------------- ---------------- ---------------- ---------------
                  60158656         20826232        123086887        134252826 Total
- Averages -
Frequency       CPU_CYCLES   FP_OPS_RETIRED IA64_INST_RETIRE   INST_DISPERSED Routine
--------- ---------------- ---------------- ---------------- ---------------- ---------------
        1        180351.00          8640.00        167754.00        189060.00 main
       27        328636.70        100000.00       1362319.44       1405185.30 matgen_
       26       1896522.54        673862.23       3201883.12       3572479.27 dgefa_
       26         68659.85         22199.00        109741.12        123178.62 dgesl_
        1         10372.00         20000.00         34278.00         36658.00 dmxpy_
- Ratios -
Frequency       CPU_CYCLES   FP_OPS_RETIRED IA64_INST_RETIRE   INST_DISPERSED Routine
--------- ---------------- ---------------- ---------------- ---------------- ---------------
        1            0.300%           0.041%           0.136%           0.141% main
       27           14.750%          12.964%          29.883%          28.260% matgen_
       26           81.966%          84.127%          67.634%          69.186% dgefa_
       26            2.967%           2.771%           2.318%           2.386% dgesl_
        1            0.017%           0.096%           0.028%           0.027% dmxpy_
```

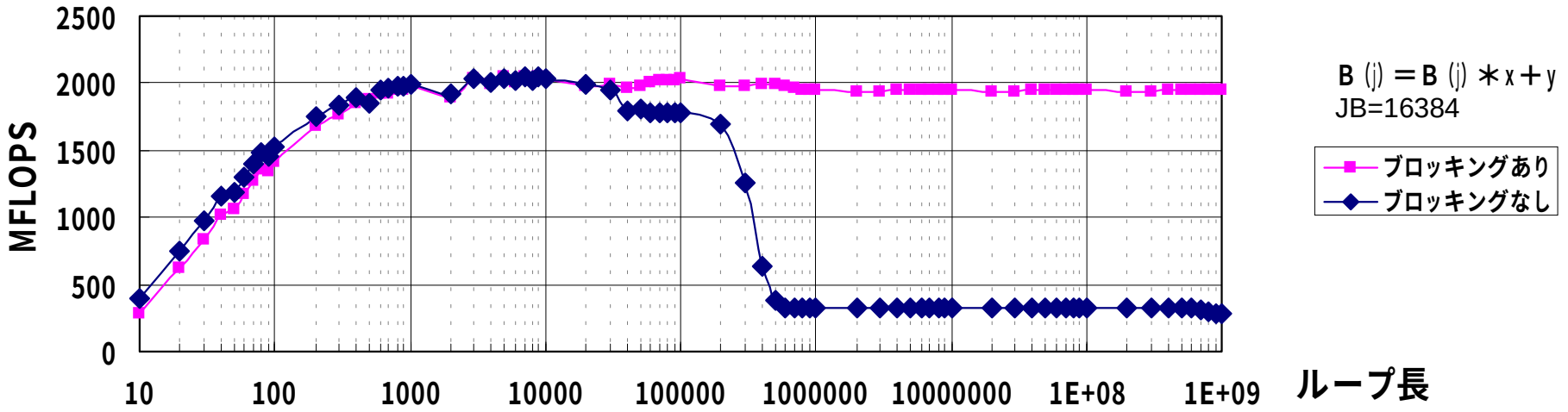
キャッシュブロッキング技法

```
DO I=1,N
 DO J=1,M
  ... = ... *B(J)
```

⇒

```
DO J2=1,M,JB
 DO I=1,N
  DO J=J2,min(J2+JB-1,M)
   ... = ... *B(J)
```



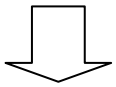


ソフトウェア　パイプライニングの促進 (効率化)

-opt_report [-opt_report_phase hlo -opt_report_phase ecg_swp]

```
19      do i=1,k
20       a(m(i))= a(m(i)) + 1.0
21      enddo
```

依存関係のため、
あまりうまくSWPされない。
II(**ループ**1**回**)=18**サイクル**
でスケジュール



指示行

```
19 !dir$ ivdep
20      do i=1,k
21       a(m(i))= a(m(i)) + 1.0
22      enddo
```

うまくSWPされるようになった。
自動的にアンローリングもされた。
実質のII=13/6 = 2.16**サイクル**

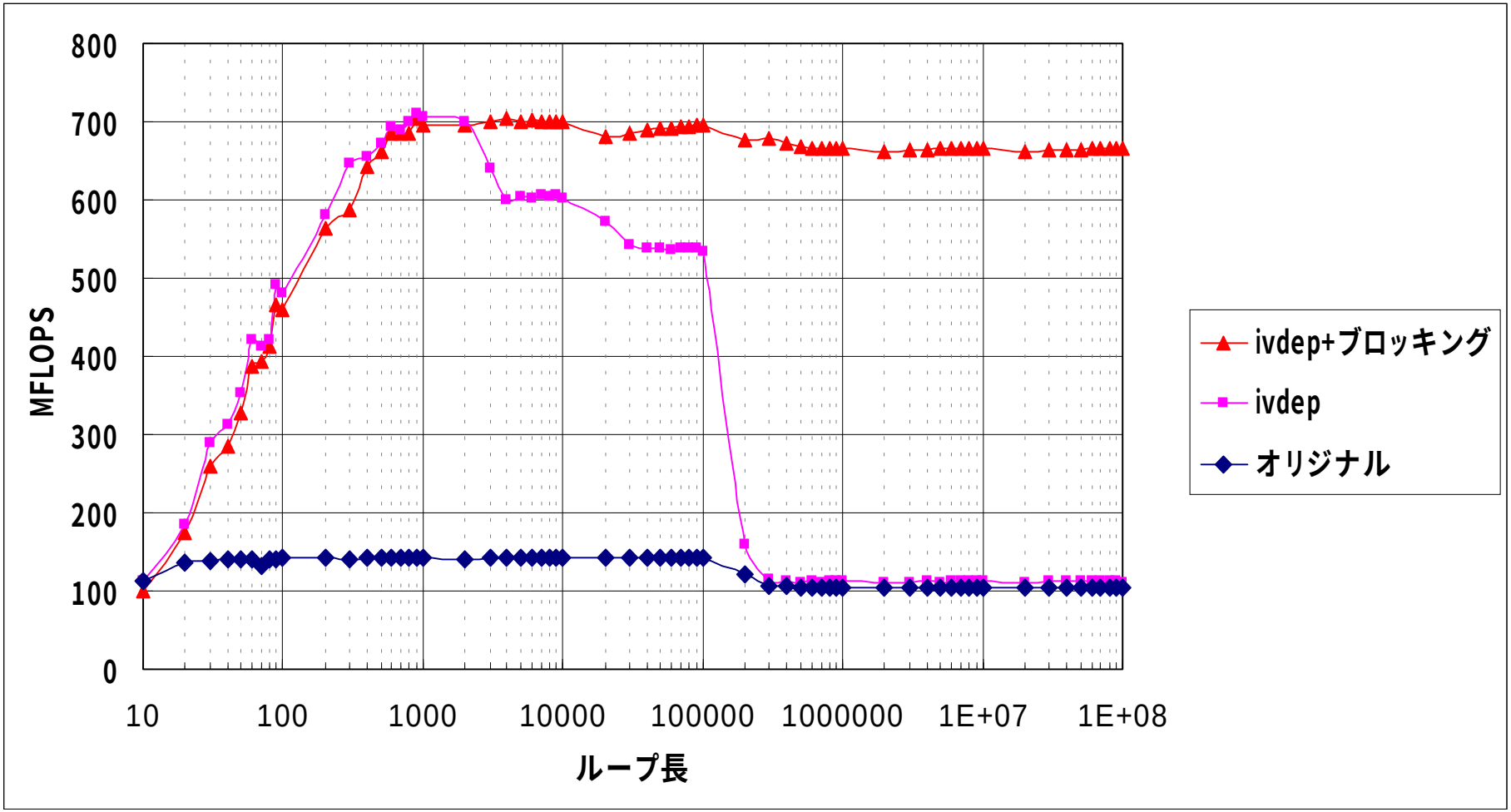
```
Swp report for loop at line 19 in main in file d0.f

        Resource II  =       3
        Recurrence II =      17
        Minimum II   =       17
        Scheduled II  =      18

        Percent of Resource II needed by arithmetic ops     = 100%
        Percent of Resource II needed by memory ops         = 100%
        Percent of Resource II needed by floating point ops =  33%

        Number of stages in the software pipeline =    2

        Following are the loop-carried memory dependence edges:
        Store  at line    20 --> Load   at line    20
```

```
Block, Unroll, Jam Report:
(loop line numbers, unroll factors and type of transformation)
Loop at line 12 unrolled with remainder by 8
Loop at line 20 unrolled with remainder by 6

Swp report for loop at line 20 in main in file d.f

        Resource II  =       12
        Recurrence II =      0
        Minimum II   =       12
        Scheduled II  =      13

        Percent of Resource II needed by arithmetic ops     = 100%
        Percent of Resource II needed by memory ops         = 100%
        Percent of Resource II needed by floating point ops =  25%

        Number of stages in the software pipeline =    9
```

ソフトウェア　パイプライニングの促進 (効率化) 例

```
do i=1,k
  a(m(i))= a(m(i)) + 1.0
enddo
```

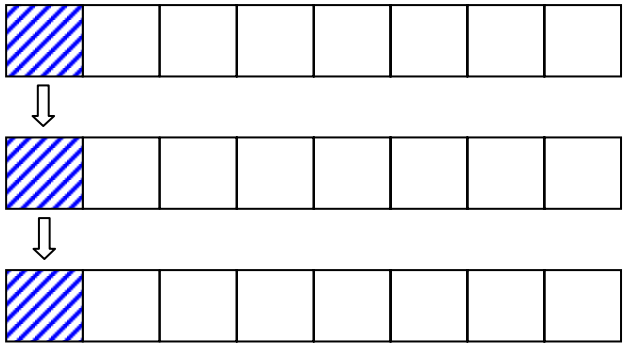
⇨ ivdep指示行 (⇨ キャッシュブロッキング)



配列の連続アクセス化

```
do i=1,n
  do j=1,n
   a(i,j) ...
```

とびアクセス

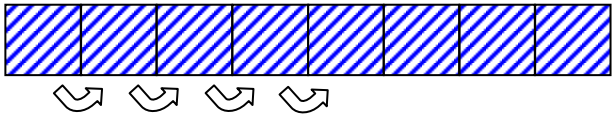


キャッシュにロードされた
データの一部しか利用しない

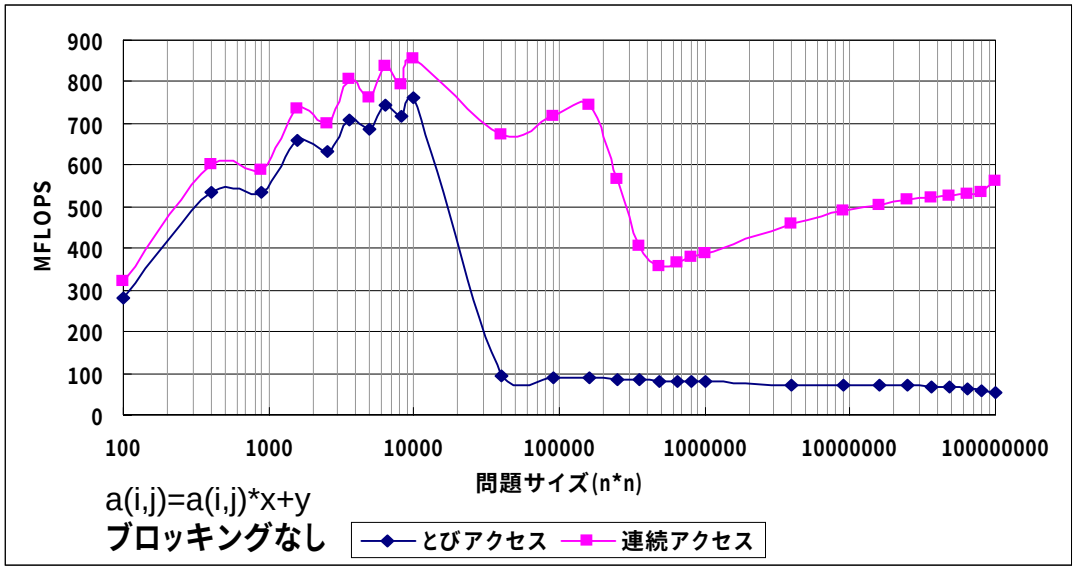
特に、2のべき乗とびのアクセス
は避ける。

```
do j=1,n
  do i=1,n
   a(i,j) ...
```

連続アクセス

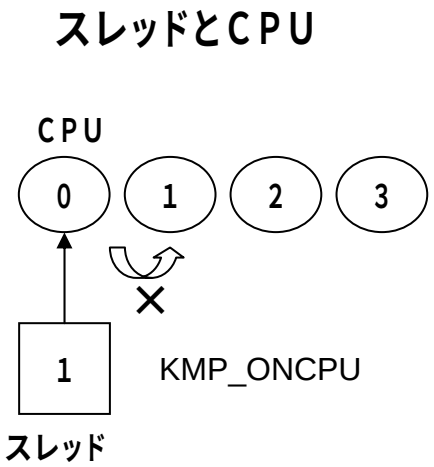


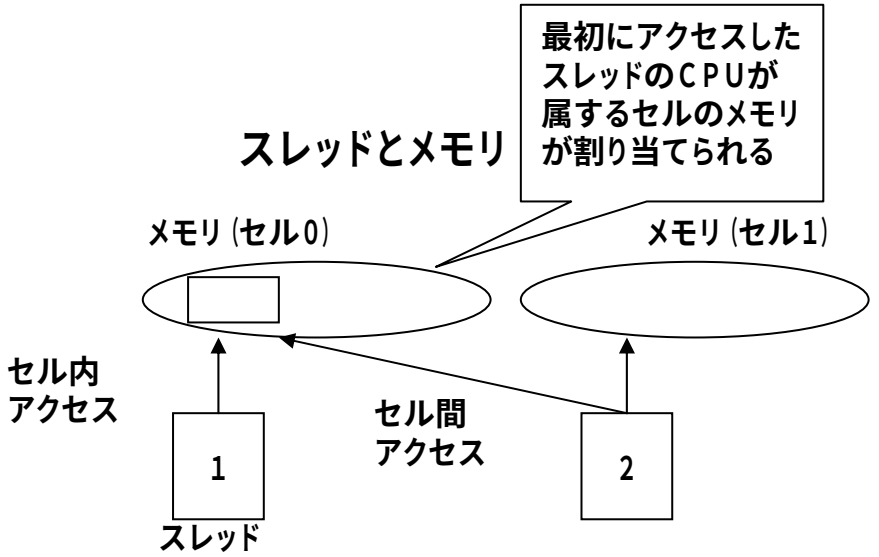
キャッシュにロードされた
データをすべて利用



並列プログラム　性能チューニングのポイント

1. 並列化率を高める
2. タスク間で負荷のバランスをとる
3. メモリアクセスの削減、効率化（シリアルプログラムの技法と同じ）
4. アフィニティ機能の利用
   - スレッドとCPUのアフィニティ（スレッドのcpuへの固定）
     - » OpenMP、自動並列化ではKMP_ONCPU環境変数を利用
   - スレッドとメモリのアフィニティ（セル内のメモリの使用）
     - » OpenMP/自動並列化：初期化ループの並列化、プログラム全体で同次元・同分割で並列化

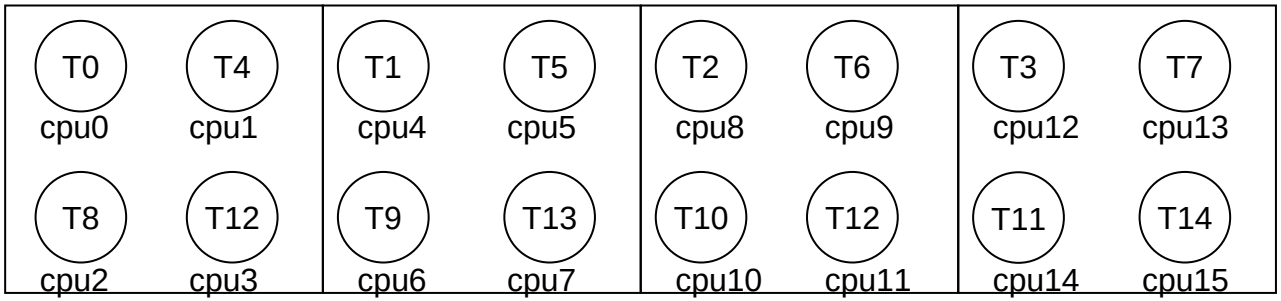




並列処理スレッドのCPUへの固定

- KMP_ONCPU環境変数
  - OpenMPまたは自動並列化で有効
  - スレッドを順に割り当てるＣＰＵ番号を指定する。
  - KMP_ONCPU=""　　(空文字を指定)
    - » スレッドが全セルにできるだけバラバラになるようにCPUの固定割当てを行なう。
  - KMP_ONCPU="0 2 4 6 8 10 12 14 1 3 5 7 9 11 13 15"
    - » 各スレッドを指定したCPUに順次固定割当てを行なう
  - KMP_ONCPUなし：OSが他のプロセスと共に総合的に割当てを行なう。

KMP_ONCPU="" の時のスレッドTnの割り当て順序 (4セル16CPUの場合)



OpenMPチューニング技法の例

オリジナル

```
do k=...
 do j=...
   do i=...
     a(i,j,k)=0

!$omp parallel do
do k=...
  do j=...
    do i=...
     a(i,j,k)=...

!$omp parallel do
do i=..
  do j = ...
    do k=...
      a(i,j,k)= ...
```

特定のセルのメモリにデータが集中してしまう

初期化ループの並列化

```
!$omp parallel do
do k=...
 do j=...
   do i=...
     a(i,j,k)=0

!$omp parallel do
do k=...
  do j=...
    do i=...
     a(i,j,k)=...

!$omp parallel do
do i=..
  do j = ...
    do k=...
      a(i,j,k)= ...
```

セル間アクセスになる

並列化次元の統一
(配列連続アクセス)

```
!$omp parallel do
do k=...
 do j=...
   do i=...
     a(i,j,k)=0

!$omp parallel do
do k=...
  do j=...
    do i=...
     a(i,j,k)=...

!$omp parallel do
do k=..
  do j = ...
    do i=...
      a(i,j,k)= ...
```