[添付]

PAイベント情報活用チュートリアル
-サイクルアカウンティング　FX1編-

富士通株式会社　　青木正樹

目次

- ◆ PAイベント情報概略
- ◆ 代表的なPAイベント情報
  - ■ 演算コミット数
  - ■ メモリアクセス待ち
  - ■ 浮動少数点演算待ち
  - ■ キャッシュアクセス待ち
- ◆ チューニング事例
- ◆ PAイベント情報採取ツール

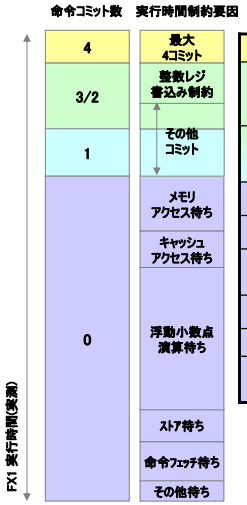
サイクルアカウンティングとは

性能ボトルネック要因分析の手法である。SPARC64VIIでは豊富なPAイベント情報を備えており、アプリケーションプログラム実行時のCPU動作状態の情報が取得できる。あるアプリケーションプログラムを実行するためにかかった総時間（CPUサイクル数）をCPUの動作状態で分類(*)し、CPU内のどの部分にボトルネックがあるかを把握することで、詳細な性能分析やチューニングを行うことができる。

(*)命令実行中、メモリアクセス待ち、演算完了待ちである、など

予告：次世代スパコン向けにも＋α充実させていきます。目標2010年末

PAイベント情報概略



最大4コミット	完了命令数が4である時間
整数レジスタ書き込み制約	整数レジスタを2個更新しているため、4命令完了できなかった時間、命令完了数は2もしくは3
その他コミット	整数レジスタ書き込み制約を伴わない、完了命令数が1，2，3のいずれかである時間
メモリアクセス待ち	主記憶からのデータ待ちのため命令完了数が0である時間
キャッシュアクセス待ち	キャッシュからのデータ待ちのため命令完了数が0である時間
浮動小数点演算待ち	実行中の命令の中で一番古い命令が、浮動小数点の演算実行中で、命令完了数が0である時間
ストア待ち	ストアポートフル(*1)要因でCSE(*2)が空となっている時間
命令フェッチ待ち(*3)	ストアポートフル要因以外でCSEが空となっている時間
その他待ち	メモリアクセス待ち、キャッシュアクセス待ち、浮動小数点演算待ち、ストア待ち、命令フェッチ待ち以外の要因で、命令完了数が0である時間

(*1) ストアポートフル
ストアーポートとはストアデータを格納するキューです。SPARC64™ VIIIfxはストアポートが満杯となった時点でストアポートに空きができるまで、以降の命令デコードを停止します。ストアポートフルの典型的なトリガはストア命令のデータキャッシュミスです。

(*2) CSE(コミットスタックエントリ)
CSEとは実行中の（発行されてまだ完了していない）命令の情報を保持するためのバッファです。

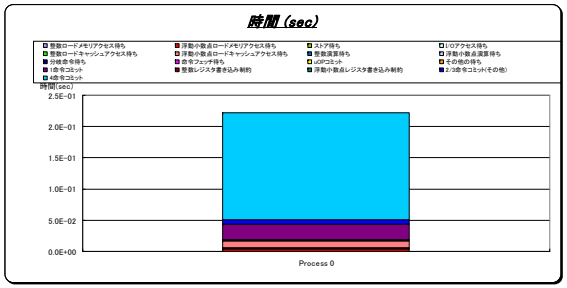
(*3) 命令フェッチ待ち
典型的なケースは以下の2つです。
- 命令キャッシュミスによりCPU内で実行すべき命令を供給できない
- スレッド並列実行において、ソフトから他コアとの同期を指示され同期成立を待っている

代表的なPAイベント情報
ーどのようなコード？ どのような情報？ー

代表的なPA情報：演算コミット数

プログラム例：行列積

```
A=MATMUL(B,C)
```



0 Commit
1 Commit
2-3 Commit
4 Commit

1マシンサイクルでn命令実行したことを示す。

0Commit部分は、何らかの要因で命令が動作していない時間。

※モノクロ印刷ではグラフの色識別が不鮮明です。SS研Webサイト「WG成果報告書」のPDFでカラー版をご覧ください。 以降同様。

代表的なPA情報：メモリアクセス待ち

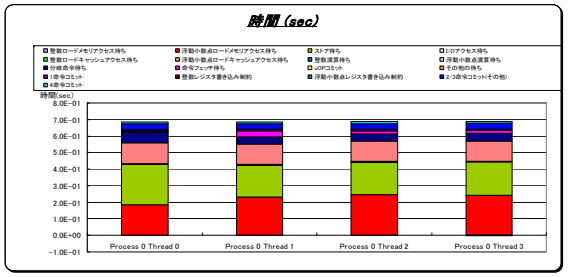
プログラム例：STREAM COPY

```
DO I=1,L2$に乗らない大きさ
  A(I)=B(I)
ENDDO
```

浮動小数点ロードメモリアクセス待ち

整数ロードメモリアクセス待ち

ストア待ち

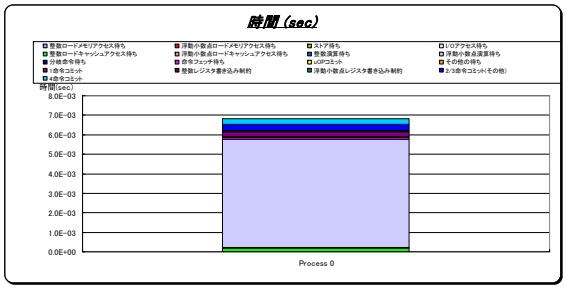


浮動小数点ロードメモリアクセス待ち、整数ロードメモリアクセス待ち、ストア待ち部分は、それぞれのメモリアクセス命令が引き起こすL2$ミス要因で他の命令が動作していない時間。

代表的なPA情報：浮動少数点演算待ち

プログラム例：除算

```
DO I=1,M    配列A,B,CはL1$上
  A(I)=B(I)/C(I)
ENDDO
```



浮動小数点演算待ち

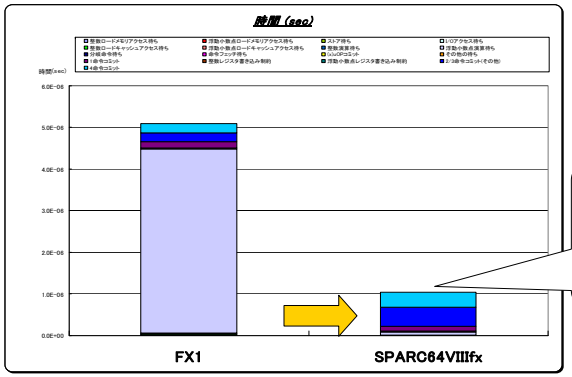
浮動小数点演算待ち部分は、浮動小数点演算命令が要因で他の命令が動作していない時間。

除算はレイテンシの長い、かつ、パイプライン実行できないハード命令で実装されているため、浮動小数点演算待ちが発生。

参考）ただしSPARC64Viiifxでの除算は

プログラム例：除算

```
DO I=1,M    配列A,B,CはL1$上
  A(I)=B(I)/C(I)
ENDDO
```

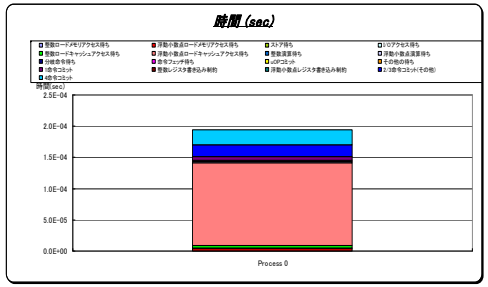


FX1性能比
約5倍向上
除算影響の待ちは
無くなる

代表的なPA情報：キャッシュアクセス待ち

プログラム例

```
DO J=1,M
DO I=1,N    配列A,BはL2$上
  A(I,J)=B(I,J)+1.0
ENDDO
ENDDO
```



浮動小数点ロードキャッシュアクセス待ち

整数ロードキャッシュアクセス待ち

浮動小数点ロードキャッシュアクセス待ちは、整数ロードキャッシュアクセス待ちは、それぞれのメモリアクセス命令がキャッシュアクセスの待ち要因で他の命令が動作していない時間。

チューニング事例

- 1次キャッシュ・スラッシング回避
- リストアクセスに対するプリフェッチ
- キャッシュ・チューニング（多重ループ融合）
- スレッド並列のロードバランス不均等
- リストアクセスのチューニング（配列マージ）
- メモリアクセスの局所化（タイリング）

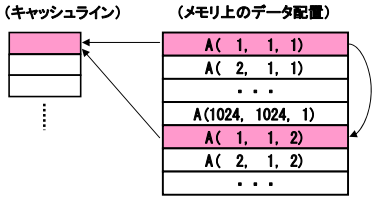
1次キャッシュ・スラッシング回避（前）

SPARC64ViiのL1D$キャッシュは、64KB/2Wayです。

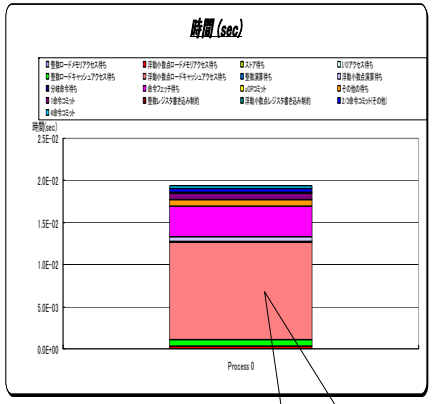
配列の形状が2のべき乗になっていると、キャッシュ競合で性能低下する場合があります。

オリジナルソース

```
REAL*8 A(1024, 1024, 4)
DO I=1,N
  A(I,1,1)=A(I,1,2)+A(I,1,3)+A(I,1,4)
ENDDO
```



同じキャッシュラインへの競合



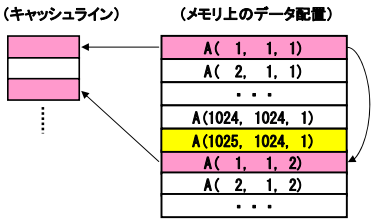
キャッシュ待ち時間

1次キャッシュ・スラッシング回避（後）

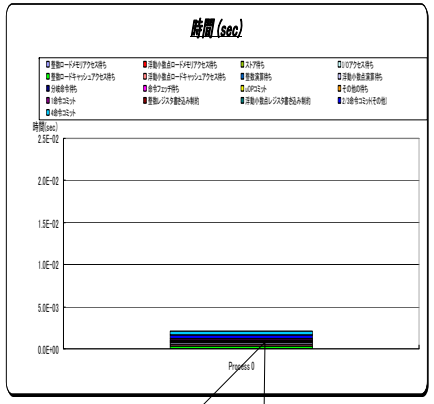
パディング（内側にすき間を空ける）を行うことにより、キャッシュ競合を削減します。

コンパイラ最適化のイメージ

```
REAL*8 A(1025, 1024, 4)
DO I=1,N
  A(I,1,1)=A(I,1,2)+A(I,1,3)+A(I,1,4)
ENDDO
```



異なるキャッシュラインで競合回避



キャッシュ待ち時間激減

リストアクセスに対するプリフェッチ（前）

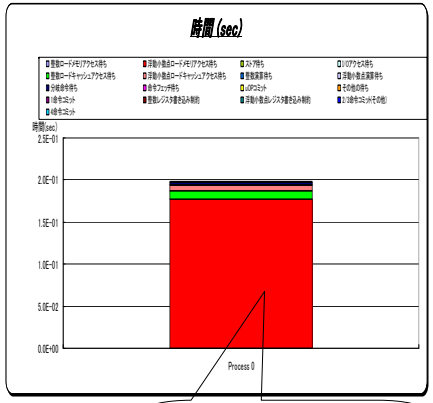
配列のリスト参照のようなインダイレクト（リスト）アクセスについては、-Kfastオプションでは、プリフェッチ命令を生成しません。

オリジナルソース

```
REAL*8 A(N),B(N),C(N)
REAL*8 D(N),E(N),F(N)
DO I=1,N

A(D(I))=B(E(I))+SCALAR*C(F(I))
ENDDO
```

配列D,E,Fに対して
プリフェッチ命令を生成



リストアクセスのメモリアクセス待ち時間

リストアクセスに対するプリフェッチ（後）

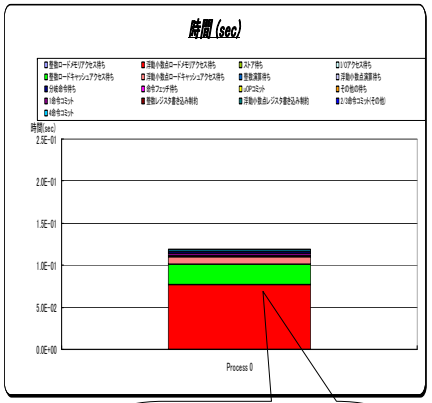
コンパイラオプション-Kprefetch_indirectもしくは最適化制御行による直接指示でリストアクセスについてもプリフェッチ命令を生成することができます。

コンパイラ最適化のイメージ

```
REAL*8 A(N),B(N),C(N)
REAL*8 D(N),E(N),F(N)
DO I=1,N

A(D(I))=B(E(I))+SCALAR*C(F(I))
ENDDO
```

配列A,B,C,D,E,Fに対して
プリフェッチ命令を生成



プリフェッチにより
待ち時間減少

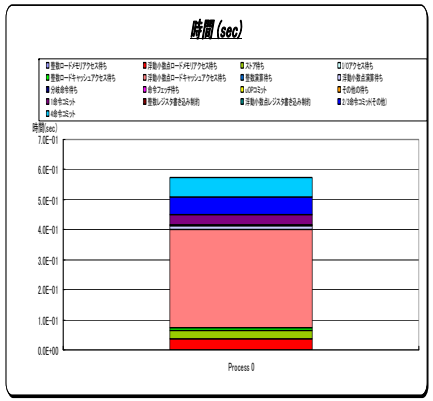
キャッシュ・チューニング：多重ループ融合（前）

キャッシュのチューニングとは、時間的/空間的に近くをアクセスすることです。チューニングの際は、メモリアクセスの状況を大域的に見てチューニングすることを薦めます。

オリジナルソース

```
REAL*8 A(N,N),B(N,N),C(N,N)
DO J=1,N
  DO I=1,N
    A(I,J)=B(I,J)+C(I,J)
  ENDDO
ENDDO
DO J=1,N-1
  DO I=1,N
    C(I,J)=A(I,J)+B(I,J)+1.0
  ENDDO
ENDDO
```

N=1000



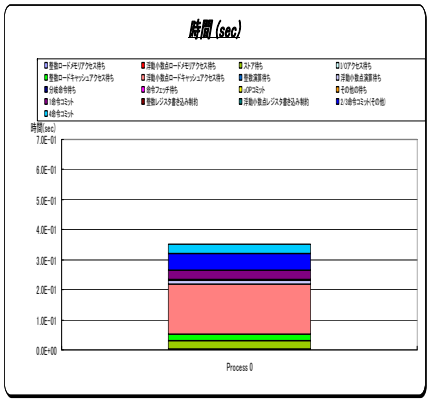
配列のアクセス時間が全体の2/3

キャッシュ・チューニング：多重ループ融合（後）

LOOP-PEELINGを行い、2つの2重ループを融合することにより、キャッシュアクセスのチューニングを行った例です。

チューニング後ソース

```
REAL*8 A(N,N),B(N,N),C(N,N)
A(N,N)=B(N,N)+C(N,N)
DO J=1,N-1
  DO I=1,N
    A(I,J)=B(I,J)+C(I,J)
    C(I,J)=A(I,J)+B(I,J)+1.0
  ENDDO
ENDDO
J=N
DO I=1,N
  A(I,J)=B(I,J)+C(I,J)
ENDDO
```



配列のアクセス時間が半減

スレッド並列のロードバランス不均等（前）

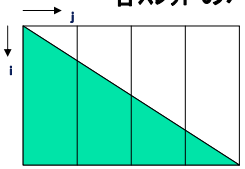
並列チューニングの観点から、解決しなければならない課題にロードバランスの均等化があります。スレッド並列の三角ループを外側ループで均等に割り付ける場合は、注意が必要です。

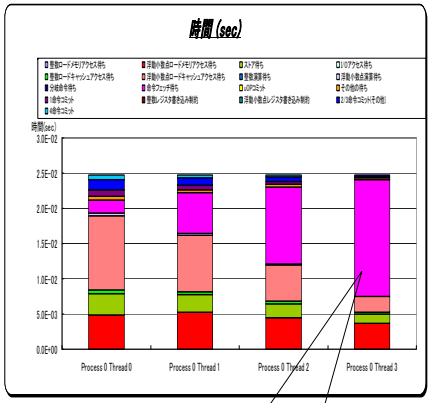
オリジナルソース

```
COMMON A,B,C,D
REAL*8 A(4097,4096),B(4097,4096),C(4097,4096)
!$OMP PARALLEL DO
DO J=1,4096
  DO I=J,4096
    A(I,J)=B(I,J)+C(I,J)
  ENDDO
ENDDO
```

三角ループ

各ｽﾚｯﾄﾞのバランス悪い！





バリア待ち時間

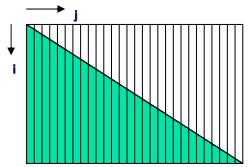
スレッド並列のロードバランス不均等（後）

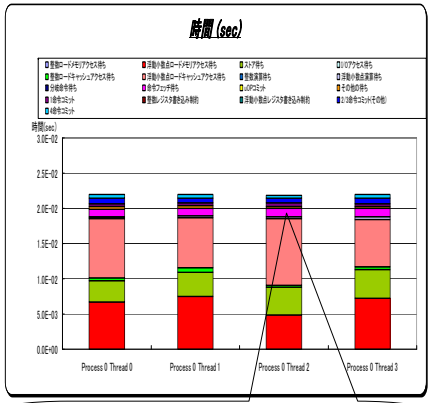
三角ループは、サイクリック割付を行うことにより、ロードバランスの均等化が可能となります。

チューニング後ソース

```
COMMON A,B,C,D
REAL*8 A(4097,4096),B(4097,4096),C(4097,4096)
!$OMP PARALLEL DO SCHEDULE(STATIC,1)
DO J=1,4096
  DO I=J,4096
    A(I,J)=B(I,J)+C(I,J)
  ENDDO
ENDDO
```

三角ループ





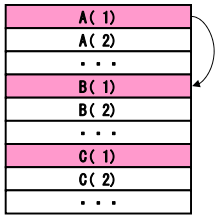
バリア待ち時間均一。ただし、1刻みだと L1$のFalse sharing影響見える

リストアクセスのチューニング：配列マージ（前）

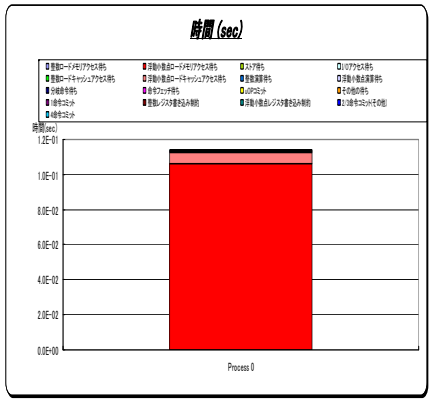
リストアクセス時、リストの値が不連続＆遠い場合、キャッシュの使用効率が悪い場合があります。

オリジナルソース

```
REAL*8 A(N),B(N),C(N)
DO I=1,N
  A(L(I))=B(L(I))+C(L(I))
ENDDO
```



ランダムアクセス



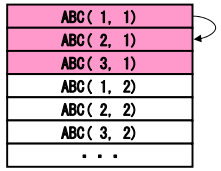
⇒キャッシュの使用効率が良くない

リストアクセスのチューニング：配列マージ（後）

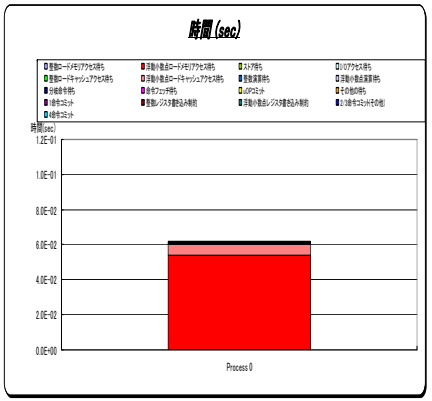
複数の配列を1個にマージすることでデータアクセスが改善される可能性があります。
最適化制御行！ocl array_mergeも用意されています。

融合後イメージ

```
REAL*8 ABC(3,N)
DO I=1,100
  ABC(1,L(I))=ABC(2,L(I))+ABC(3,L(I))
ENDDO
```



連続アクセス



⇒キャッシュ使用効率向上

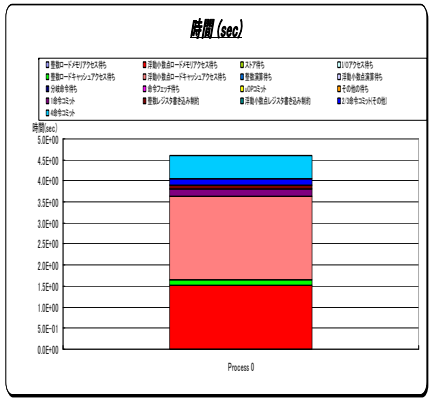
メモリアクセスの局所化：タイリング（前）

大きな配列の転置は、キャッシュの使用効率が良くありません。

オリジナルソース

```
REAL*8 A(1000,1000),B(1000,1000)
DO I=1,N
   DO J=1,N
      A(I,J)=B(J,I)
   ENDDO
ENDDO
```

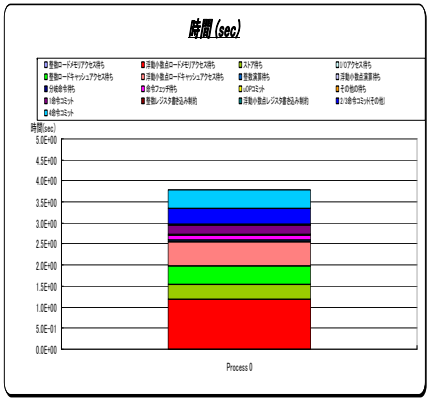
配列Bは1000要素飛びアクセス



メモリアクセスの局所化：タイリング（後）

メモリアクセスを局所化するために,配列をM×Mサイズに分割します。これをタイル化といいます。(Mのサイズはアクセス状況/キャッシュサイズに依存します) 今回は、タイルサイズ40での実測結果。

チューニング後ソース

```
REAL*8 A(1000,1000),B(1000,1000)
M=40
DO II=1,N,M
   DO JJ=1,N,M
      DO I=II,MIN(II+M,N)
         DO J=JJ=MIN(JJ+M,N)
            A(I,J)=B(J,I)
         ENDDO
      ENDDO
   ENDDO
ENDDO
```

配列BはM要素飛びアクセス