サイエンティフィック・システム研究会
科学技術計算分科会第2回会合資料

FUJITSU

「スカラ並列機におけるアプリの高速化事例」

2000年11月15日

富士通株式会社

計算科学技術センター　HPCシステム部

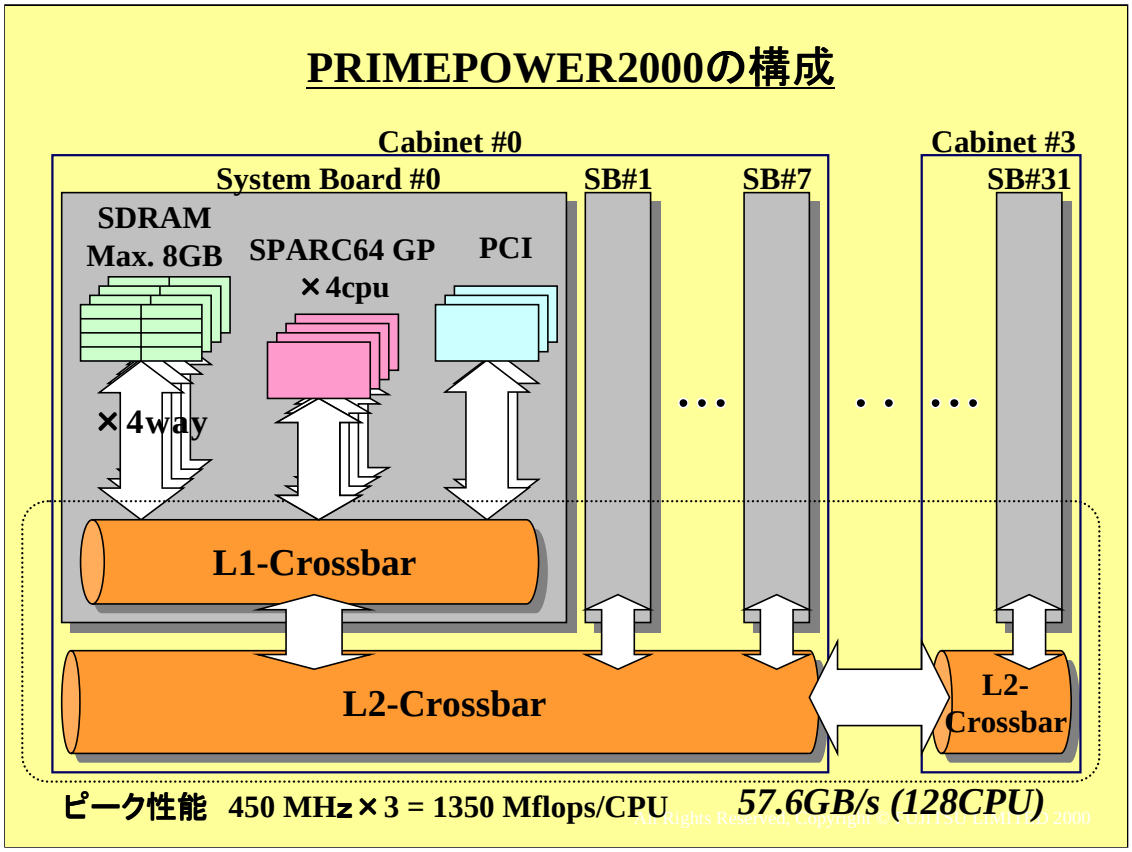
市　川　眞　一

e-mail : ichikawa@strad.se.fujitsu.co.jp

All Rights Reserved, Copyright © FUJITSU LIMITED 2000

内容構成　：

1）富士通のスカラ並列機PRIMEPOWER2000の紹介
2）スカラ処理の特徴
3）スカラ処理性能の考え方
4）プログラミング例
5）スカラ並列処理の特徴、並列処理性能向上の考え方と注意点
6）プログラミング例
7）NAS　Parallel　Benchmark（NPB）　BT、LUのプログラム構造とスカラチューニング方法、並列化方法
8）PRIMEPOWER2000におけるNPB　BT、LUのスカラチューニング効果と並列処理性能



富士通のスカラ並列機PRIMEPOWER2000は2階層クロスバーによってCPUとメモリが接続された共有メモリ型並列計算機である。システムボード当たり4CPU、最大メモリ8GBを搭載し、システム全体では128CPU、最大メモリ256GBを搭載する。

プロセッサの構成

CPU	**:**	**SPARC64 GP** **Superscalar,** **Register Renaming,** **Out of Order Instruction Execution** **450MHz**
浮動小数点演算器	**:**	**M&A×1, A×1**
メモリアクセスパイプ	**:**	**2 Load/Store**
一次Cache	**:**	**data 128KB(4way)** **instruction 128KB(4way)**
二次Cache	**:**	**8MB**

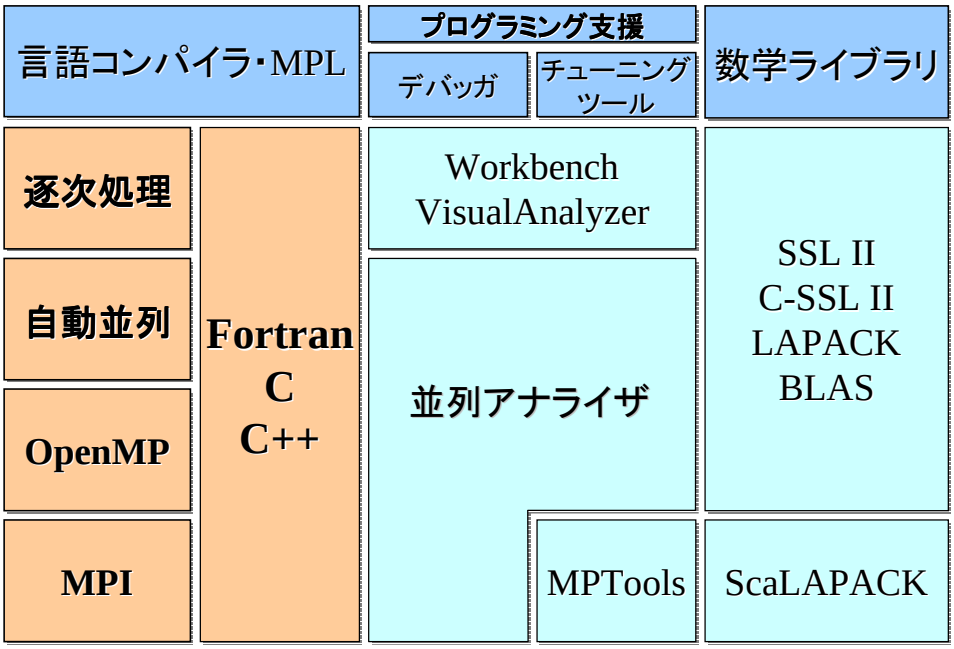
All Rights Reserved, Copyright © FUJITSU LIMITED 2000

各プロセッサはSPARC64 GPが採用されており、ハードウェア検出による命令の並列実行を行うSuperscalar、コンパイラが管理するレジスタをハードウェアが一時的に名称変更することによって管理し実効的なレジスタ数を増加させるRegister Renaming、ベクトル計算機でも取り入れられてきた方法である依存性の無い命令を先行実行させることによって性能向上をはかるOut of Order Instruction Execution などの機能があり、クロックは450MHzである。4命令の同時実行が可能である。

浮動小数点演算器は、M&AとAddから構成され、ピーク性能はCPU当たり1350Mflopsである。また2つのLoadとStore兼用のパイプラインがある。

キャッシュサイズは、L1がデータ用と命令用各々128KB、L2がデータと命令共用で8MBである。

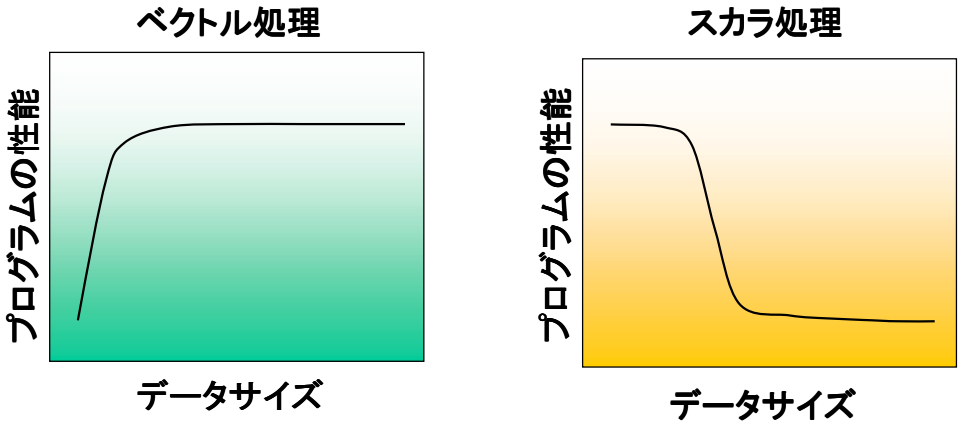
PRIMEPOWERの言語環境



All Rights Reserved, Copyright © FUJITSU LIMITED 2000

PRIMEPOWER2000では、Fortran, C, C++に対して各々、自動並列、OpenMP、MPIといった並列処理機能・環境が用意されている。また、デバッガ、チューニングツールや、数学ライブラリが用意されている。

スカラ処理の特徴



◇スカラ処理ではCache経由によるメモリアクセスを行う。

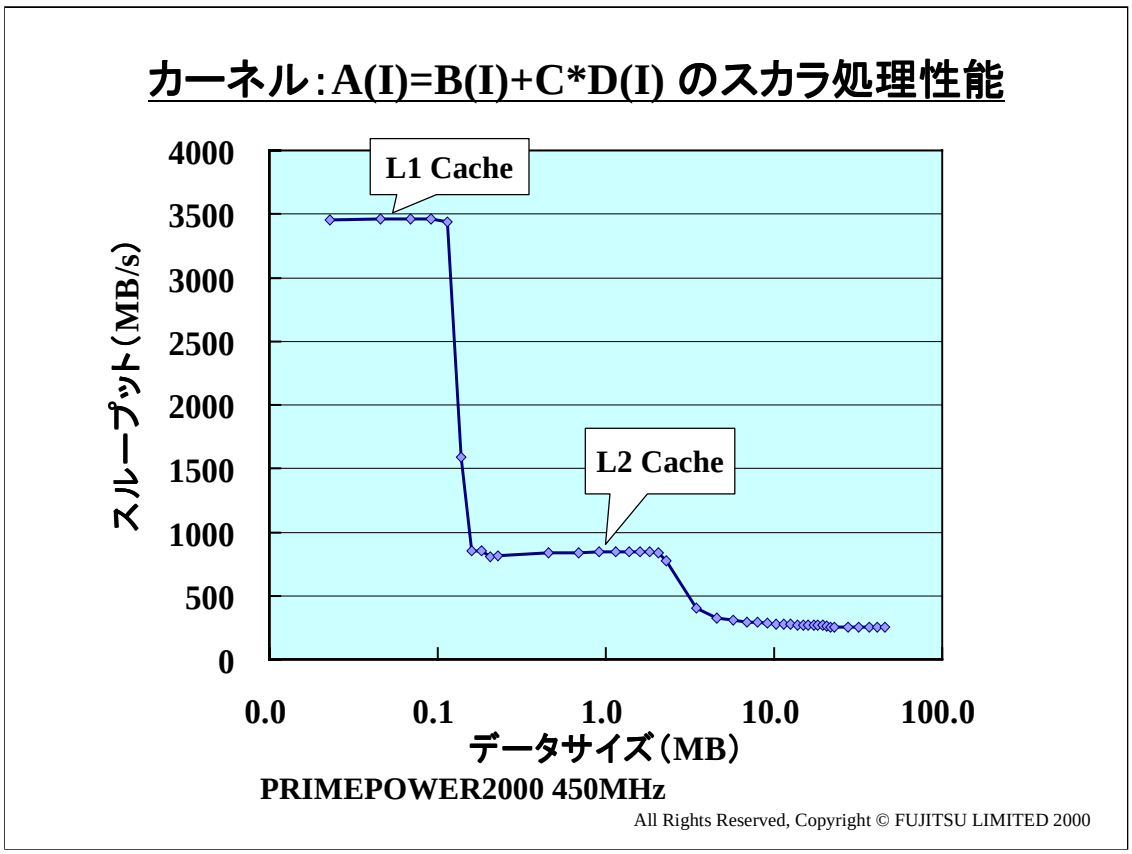
◇データサイズの拡大とともに性能が低下するが、小さいときは高性能。

All Rights Reserved, Copyright © FUJITSU LIMITED 2000

ベクトル処理と対比したときのスカラ処理の最大の特徴は、ベクトル処理ではインタリーブ構造によって高速化されたメモリに直接アクセスするのに対して、スカラ処理では、キャッシュを経由したメモリアクセスを行う点にある。

このため、ベクトル処理ではループ長が短くデータサイズが小さいときには立ち上がり時間のために性能が低いが、データサイズが大きくなると非常に高い性能を発揮できるのに対して、スカラ処理では、データサイズがキャッシュより大きくなると性能が大きく低下する。

しかし、スカラ処理ではデータサイズが小さい範囲では高い性能を発揮することが可能である。



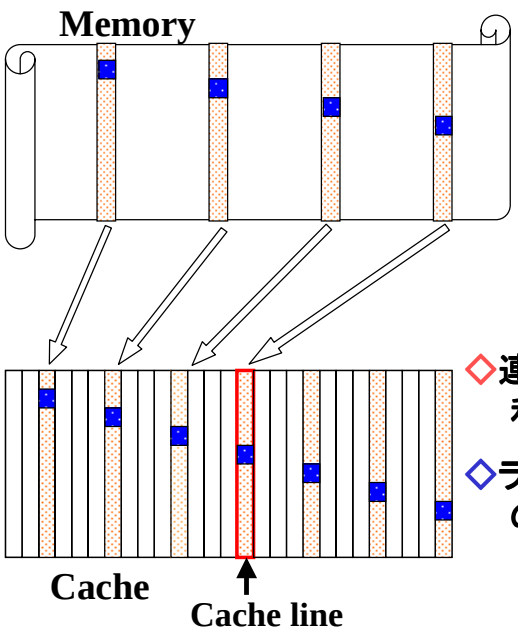
キャッシュシステムの性能がデータサイズが小さいときには高く、データサイズが大きいときには低いことの具体例として、PRIMEPOWER2000上で測定した、STREAMベンチマークのTRIADと同一のカーネルの性能を示す。

測定は、各データサイズにおいてカーネルを繰り返し実行したときのものである。

ちょうどL1キャッシュのサイズでは非常に高い性能であるが、データがL1キャッシュよりも大きくなると性能が低下し、更にL2キャッシュよりも大きくなるとかなり低下する。最大、約14倍の性能差がある。

これより、高いスカラ処理性能を得るためには、できるだけ、キャッシュ上に必要なデータが存在し、メモリまでアクセスせずに処理できるようにする必要があることがわかる。

Cache経由によるメモリアクセスの特徴(1)



Cache line 長単位で連続域を転送

64Bytes/Cache-line (PRIMEPOWER2000)

◇連続アクセスではCache lineを有効利用

◇ランダムアクセスではメモリアクセスの効率低下 ：

1／(line)

line = cache line上要素数

All Rights Reserved, Copyright © FUJITSU LIMITED 2000

キャッシュ経由によるメモリアクセスの特徴は、キャッシュラインサイズを単位としてアクセスが行われる点にある。このため、キャッシュ上にデータが存在しないときには、たとえ1要素のみのアクセスであっても、キャッシュラインサイズのデータがメモリから転送される。PRIMEPOWER2000ではキャッシュラインサイズは64Byteであり、ベクトル計算機のアクセス単位である4ないし8Byteに比べてかなり大きい。他のスカラ計算機でも同様の傾向にある。

従って、連続アクセスではメモリから転送されたキャッシュライン上のデータが有効に使われるが、ランダムアクセスではキャッシュラインデータが有効に使われず、メモリの転送性能が活かせないことになる。

例示の場合では、キャッシュライン上の1要素だけが使用され、実効的なメモリアクセス性能はライン上要素数分の1となる。

キャッシュラインサイズを単位とするメモリアクセスであることから、キャッシュシステムでは、アプリケーションのメモリアクセスにおける連続性がとりわけ重要であると言える。

Cache経由によるメモリアクセスの特徴(2)

	ベクトル処理	スカラ処理
メモリレイテンシ隠蔽	Pipeline処理 “高速ハードウェアに依存したスケジューリング”	Prefetch処理 “コンパイラによるスケジューリング”
一時保管データの再利用	Vector Register長単位：コンパイラによる管理 “確実な再利用”	Cache line単位：ハード任せ、Cache line競合 “再利用に不確実さ” Register単位：コンパイラ管理“再利用範囲小”
	Vector Registerサイズ＝128KB （VPP)	Cacheサイズ=8MBと巨大化（PRIMEPOWER) -Cache line競合の緩和 -再利用範囲の拡大

All Rights Reserved, Copyright © FUJITSU LIMITED 2000

その他のキャッシュ経由によるメモリアクセスの特徴を次にまとめる。

1）一般にメモリは遠くアクセスの際にレイテンシを持つが、このレイテンシを隠蔽することが重要となる。ベクトル処理ではインタリーブ構造によるパイプライン処理によってレイテンシの隠蔽を行っており、これにコンパイラのスケジューリングが加わるが、高速ハードウェア主体のスケジューリングによってレイテンシが隠蔽されていると言える。

スカラ処理ではハードウェアによるメモリアクセスのパイプライン処理があったとしてもその範囲は狭く限られ、依然としして残るレイテンシを隠蔽できるのは、プリフェッチ処理である。プリフェッチ処理はコンパイラがスケジューリングし、オブジェクト上に命令を挿入することによって実現されるため、その効果はコンパイラだけでなくアプリケーションプログラムにも依存する。

2）演算器近辺にデータを一時的に置き、再度参照する際にはメモリからではなくこの一時的な保管先のデータを再利用することもメモリアクセスの観点から重要である。ベクトル処理では、コンパイラがVector Register長を単位として管理するため、管理の範囲はループレベルに限られるが、多量のデータの確実な再利用が可能である。一方、スカラ処理では、キャッシュライン単位でハードウェアがコードの内容に関係なく固定的な方法で管理する。このため、同一のキャッシュラインではあるが別のメモリアドレスにあるデータをアクセスすることによるキャッシュラインの競合が頻繁に発生し、再利用を確実に行うことができない。また、スカラ処理では、コンパイラによるレジスタ単位の管理は行っているが、再利用範囲は狭く再利用規模は極めて小さい。ただし、キャッシュサイズの大容量化により、キャッシュライン競合の緩和と再利用範囲の拡大が計られている。更に、キャッシュラインの多重化（set-associative cache)によりキャッシュライン競合の緩和が計られている。

プログラミングによるスカラ性能向上の考え方

(1) キャッシュラインデータの利用率向上

⟶ メモリアクセスの“連続性”を高める。

(2) キャッシュ上データの再利用度向上

⟶ メモリアクセスの“局所性”を高め、

Cache line競合を可能な限り回避することによって、

繰り返し参照による再利用を促進。

(3) レイテンシの隠蔽

⟶ ループアンロール、ループ融合、等によるPrefetch

最適化の促進や命令実行の並列度の向上。

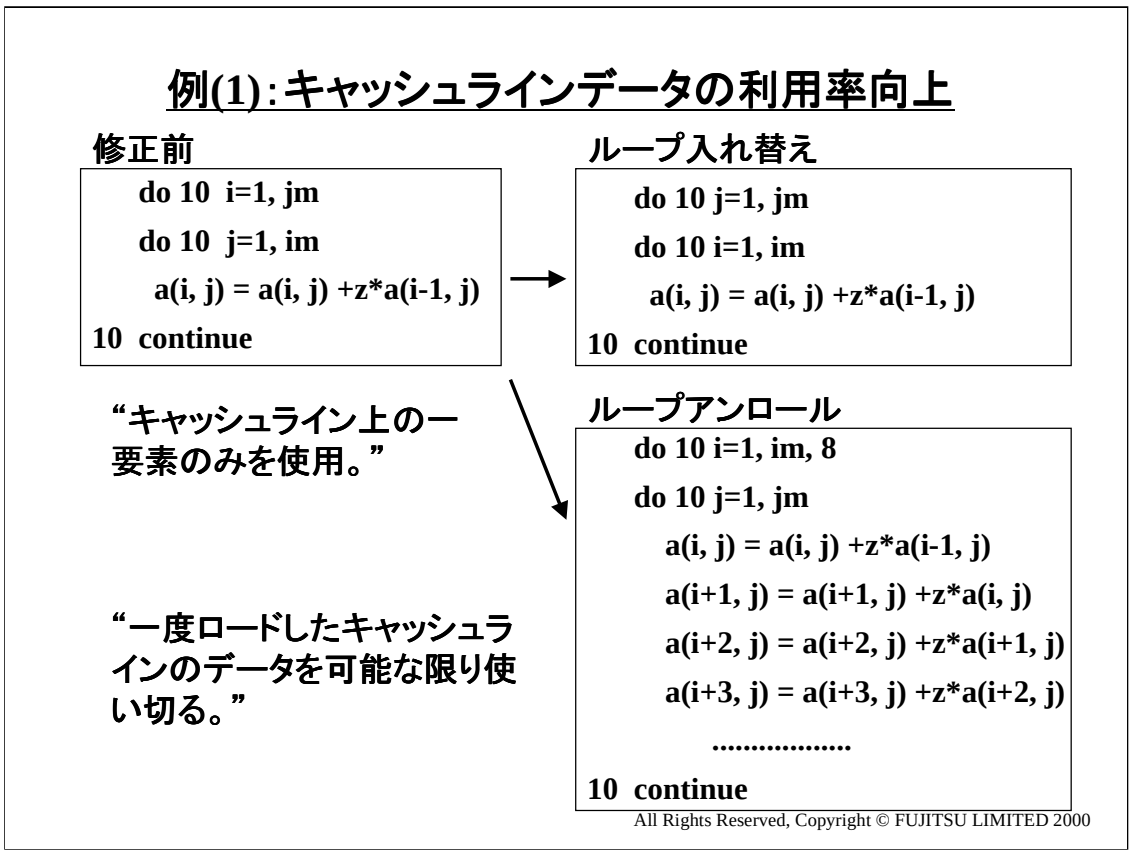
All Rights Reserved, Copyright © FUJITSU LIMITED 2000

以上からプログラミングによるスカラ性能向上の考え方を整理すると以下となる。

1）まずメモリアクセスの連続性を可能な限り高め、一度ロードしたキャッシュライン上のデータを使い切るようにする。

2）メモリに頻繁にアクセスすると高い性能は期待できない。しかし、アプリケーションによって程度の差は存在するが、科学技術計算では同一のデータを繰り返し参照するため、これを利用してキャッシュ上のデータの再利用度を高めることが可能である。このために、アプリの並列性を利用して、コードのメモリアクセスの局所性を高めることによって、キャッシュライン競合をできるだけ回避し、これによって繰り返し参照の際のキャッシュからの再利用を促進させる。

3）メモリだけでなくその他の演算器も一定のレイテンシを持つ。ループアンロールやループ融合などによって、コンパイラのプリフェッチ最適化を促進させ、また命令実行の並列度を向上させ、レイテンシを隠蔽できる可能性を高める。



キャッシュライン上のデータの利用率向上の例である。

修正前はメモリアクセスが連続ではなく飛び飛びのアクセスとなっているため、配列が大きいときキャッシュライン上の一要素のみを使用していて、利用率が低い。連続性を高めるためにはループの外側と内側を入れ替え連続アクセスとすることができる。もうひとつの案としては、ループの入れ替えは行わず外側ループをアンロールすることによって、連続性を持たせることができる。

例(2):キャッシュラインデータの利用率向上

修正前

```
    dimension a(im, jm, 3)
    dimension b(im, jm, 3)
    do  m=1,3
    do 10 j=1, jm
    do 10 i=1, im
       a(i, j, m) = a(i, j, m) *z1
       b(i, j, m) = b(i, j, m) *z2
         .............
10  continue
    end do
```

“配列A、B、、、、の間のキャッシュライン競合が起きやすい。”

配列次元入れ替え&ループアンロール

```
    dimension a(3, im, jm)
    dimension b(3, im, jm)
    do 10 j=1, jm
    do 10 i=1, im
       a(1, i, j) = a(1, i, j) *z1
       a(2, i, j) = a(2, i, j) *z1
       a(3, i, j) = a(3, i, j) *z1
       b(1, i, j) = b(1, i, j) *z2
       b(2, i, j) = b(2, i, j) *z2
       b(3, i, j) = b(3, i, j) *z2
         ..............
10  continue
```

“同左　起きにくい。”

All Rights Reserved, Copyright © FUJITSU LIMITED 2000

キャッシュライン上のデータの利用率向上の例である。

修正前は、ループ中に配列が多数出現し、また、ベクトル計算機向けのコードでよく見られるように配列宣言においてベクトル化に利用できない次元については最後の次元としている。ベクトル処理とは対照的にスカラ処理では最初に手続き方向にアクセスが進むため、ループ中に多数の配列が出現するとメモリアクセスの連続性が低くなり、配列アクセス(配列A,B, , , ,)の間でキャッシュライン競合が起きやすい。

これを、まず配列の次元を入れ替え、大きさが3の次元を1次元目としかつ、この次元のループをアンロールすることによって連続要素へのアクセスをループ中に出現させ、メモリアクセスの連続性を高めることができる。これによって、キャッシュラインデータの利用率を向上させることができる。

例(3):キャッシュデータの再利用度向上

修正前

```
    do 10 j=1, jm
    do 10 i=0, im
       work(i, j) = a(i, j) *z
10  continue
    do 20 j=1, jm
    do 20 i=1, im
     c(i, j) = c(i, i)+y1*work(i, j)
   &                  - y2*work(i-1, j)
20  continue
```

“問題規模が大きいとき、IとJの二次元分の回転によって配列workはキャッシュからあふれ、do20で再利用できない。”

ループ融合 & 配列次元縮小

```
    do 30 j=1, jm
    do 10 i=0, im
       work(i) = a(i, j) *z
10  continue
    do 20 i=1, im
      c(i, j) = c(i, i)+y1*work(i)
   &                  - y2*work(i-1)
20  continue
30  continue
```

“一次元分のサブセットを単位に処理する。

do20で配列workをキャッシュ上から再利用できる可能性が高まる。”

All Rights Reserved, Copyright © FUJITSU LIMITED 2000

キャッシュデータの再利用度を高める例である。

修正前のコードはベクトル計算機向けのコードでよく見られるように、作業配列に全ての次元分の計算結果を格納したのち、これを全次元分回転する別のループで参照するものである。作業配列が大きくループの回転範囲が大きいとき、キャッシュからデータがあふれ、作業配列を参照するループではキャッシュ上からデータを再利用することができない、あるいは再利用できる可能性が低い。

計算内容はIとJ各次元について独立であり、次元Jの特定値について定義し、その結果を参照したのち、次のJについて繰り返す手順が可能である。そこで、作業配列定義部と参照部分のJループを融合し、I次元だけのサブセットを単位に処理を行うことによって、キャッシュからのデータあふれを回避し、再利用を促進させることができる。このとき、作業配列は二次元である必要はなく、また二次元のままでは、J次元の推移によってキャッシュ上を全てスウィープし、キャッシュミスヒットを引き起こすため、次元縮小によってこれを回避することができる。

例(4):キャッシュデータの再利用度向上

修正前

```
do j = 1, jm
  do i = 1, im
    do k = 1, km
      c(i, j) = c(i, j)+a(i, k)*b(k, j)
    enddo
  enddo
enddo
```

→

ブロック化(ストリップマインニング)

```
do is = 1, im, nblock
do ks = 1, km, nblock
  do j = 1, jm
    do i = is, min(is+nblock-1, im)
      do k = ks, min(ks+nblock-1, km)
        c(i, j) = c(i, j)+a(i, k) * b(k, j)
      enddo
    enddo
  enddo
enddo
enddo
```

“配列が大きくキャッシュに乗り切らない時、ブロック化によって各、Iループによる配列Bの、Jループによる配列Aの、キャッシュ上データの再利用度が高まる。”

All Rights Reserved, Copyright © FUJITSU LIMITED 2000

キャッシュデータの再利用度を高める例である。例(3)ではモデルの構造を自然に利用してサブセット化したが、ここでは機械的にサブセット化する例を取り上げる。

行列AとBの行列積の計算であり、元の行列AとBは繰り返し参照されるが行列が大きいとき、キャッシュからあふれ、キャッシュからの再利用ができない。そこで行列積を部分的に行うようブロック化(ストリップマイン)することによって、キャッシュあふれの可能性を低下させ、再利用を促進することができる。

スカラ並列処理の特徴

	ベクトル処理	スカラ並列処理
利用する並列性	ループレベルに限定	広い：ループ、ループ群、サブルーチン
起動オーバーヘッド	小 （pipeline立上り)	やや大 (thread起動)
負荷バランス	対処不要（不可能）	対処必要（可能）
並列化の注意点	回帰演算	-回帰演算 -共有データ／プライベートデータの区分け

All Rights Reserved, Copyright © FUJITSU LIMITED 2000

スカラ並列処理の特徴についてまとめる。

スカラ並列処理では、利用できる並列性がベクトル処理よりも遥かに広く、サブルーチンレベルにまで広げられるという特徴を持つ。

共有メモリ上のスカラ並列処理では、プロセス起動やスレッド起動が必要であり、その起動オーバーヘッドはベクトル処理の場合のパイプライン立ち上がり時間に比べてやや大きい。スカラ並列処理で高い並列性能を得るにはこの点に留意する必要がある。

また、ベクトル処理では、負荷バランスそのものが定義されず、対処不要かつ不可能であるのに対して、スカラ並列処理では高速化のためには対処が必要であり可能である。

スカラ並列処理化の際に特に注意しなくてはならないのは、共有データとプライベートデータの区分けが必要である点である。逐次処理の場合には同一の作業領域がアクセスされて問題がなくても、共有メモリ上で並列処理する場合には、同一領域がアクセスされることによって逐次処理とは異なる計算となる。プライベートデータはプログラム上は同一領域に見えても実際は各プロセッサ固有の作業領域にあり、こうした現象を回避できる。

回帰演算についてはベクトル処理同様に単純な並列化ができない。

例(5):スカラ並列処理性能向上の考え方と注意点

```
      do 10 k = 1, km
      do 10 j = 1, jm
      do 20 i = 1, im
20    b(i) = c * d(i  ,j  ,k  )
      do 30 i = 2, im
30    a(i  , j  ,k  ) = a(i  , j  ,k  )
     &                +b(i-1)
     &                +b(i )
10    continue
```

(1) 性能向上：粒度拡大

利用できる並列性の広さを最大限に利用　：

I, Jループではなく Kループで並列化。

(2) 注意点：プライベート化

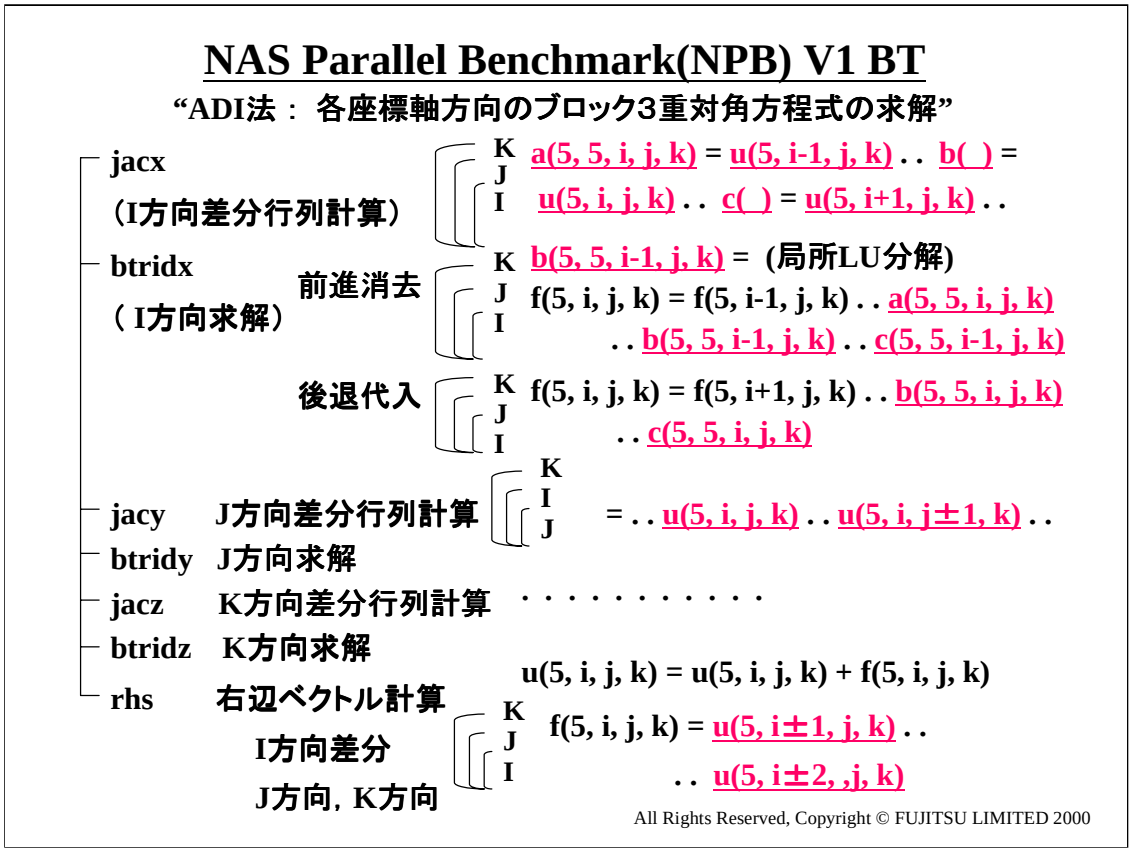
配列 b はCPU間で共有できない。

（ Kループで並列化するとき。共有すると、意図した計算ではなくなり、結果の再現性が無くなる。）

All Rights Reserved, Copyright © FUJITSU LIMITED 2000

スカラ並列処理性能向上のためには、負荷バランスに留意するのはもちろんであるが、まず大きい起動オーバーヘッドが実効的に並列処理性能に影響しにくいよう、利用可能な並列性の範囲の広さを最大限に利用して並列処理の粒度を拡大する必要がある。多重ループであればできるだけ外側のループを並列化する。また、手続き呼び出しを含むようなコストの大きいループを並列化できる。

プログラム例ではKループが並列化され作業配列bがCPU間で共有されると、他CPUが定義した値が再現性無く参照されることとなり、意図した計算ではなくなる。このため、プライベート化する必要がある。



アプリケーションプログラムのチューニング例としてNAS Parallel Benchmark（NPB）のBTを取り上げる。図はVersion-1のサンプルコードの構造をスカラ処理および、スカラ並列処理の観点から纏めている。BTでは解法としてADI法が採用されており、各座標軸方向毎に順次、ブロック三重対角行列の生成とこれによる連立方程式を解いている。ブロック三重対角方程式の求解部では各格子点での局所的なLU分解とこれによる前進消去後退代入を行いながら、隣接格子点間差分関係に基づく三重対角行列としてのLU分解と前進消去と後退代入が行われ、この部分は対応する座標軸方向に各々回帰的な演算となる。各座標軸終了後、最後に右辺ベクトルの計算が行われ、以上が繰り返される。

ここでブロック三重対角行列は座標空間3次元と変数次元（5,5）の次元を持つ作業配列a,b,cに格納され、Class-Bの問題規模ではメモリサイズが合計600MB程度にもなる。これらは、全ての格子点について計算された後すぐに前進消去後退代入計算に使われるが、メモリサイズが大きくこのままではキャッシュ上から再利用することができない。しかし、行列生成と求解はこれに対応する座標軸（例えばI軸）と直交する面内（JK面内）の格子点について独立であるため、1列（特定の（J,K）点）のみ行列生成と求解を行い、このあと他の列（（J,K）点）を計算することができる。この場合作業配列は座標空間については1次元分の領域しか使せず、メモリサイズは60KB程度となり、キャッシュの8MB よりも十分に小さくなる。このとき、もはや作業配列は座標空間について3次元である必要はない。同様に、各方向の行列生成と右辺ベクトル計算の各方向差分で参照される漸近解Uは40MB程度であるが、I方向計算とJ方向計算はK点について独立であることを利用しIJ面の2次元分の約400KBをI方向計算のあとJ方向計算の際にキャッシュ上から再利用することができる。

NPB V1 BTのスカラチューニング

1) 配列 a, b, cのキャッシュデータ再利用化 （ J方向、K方向も同様）

- ループ融合　Kループ，Jループ　：　jacx, btridx全体
- 配列次元縮小　：　作業配列 a, b, c (5, 5, i, j, k)　➡　(5, 5, i)

2) 配列 u のキャッシュデータ再利用化

ループ融合　Kループ　：　jacx, btridx, jacy, btridy全体

：　rhs内 I方向 、J方向計算

NPB V1 BTの並列化

1) スカラチューニング後の最外側ループをOpenMPで並列化
2) 次元縮小した配列 a, b, cをプライベート化

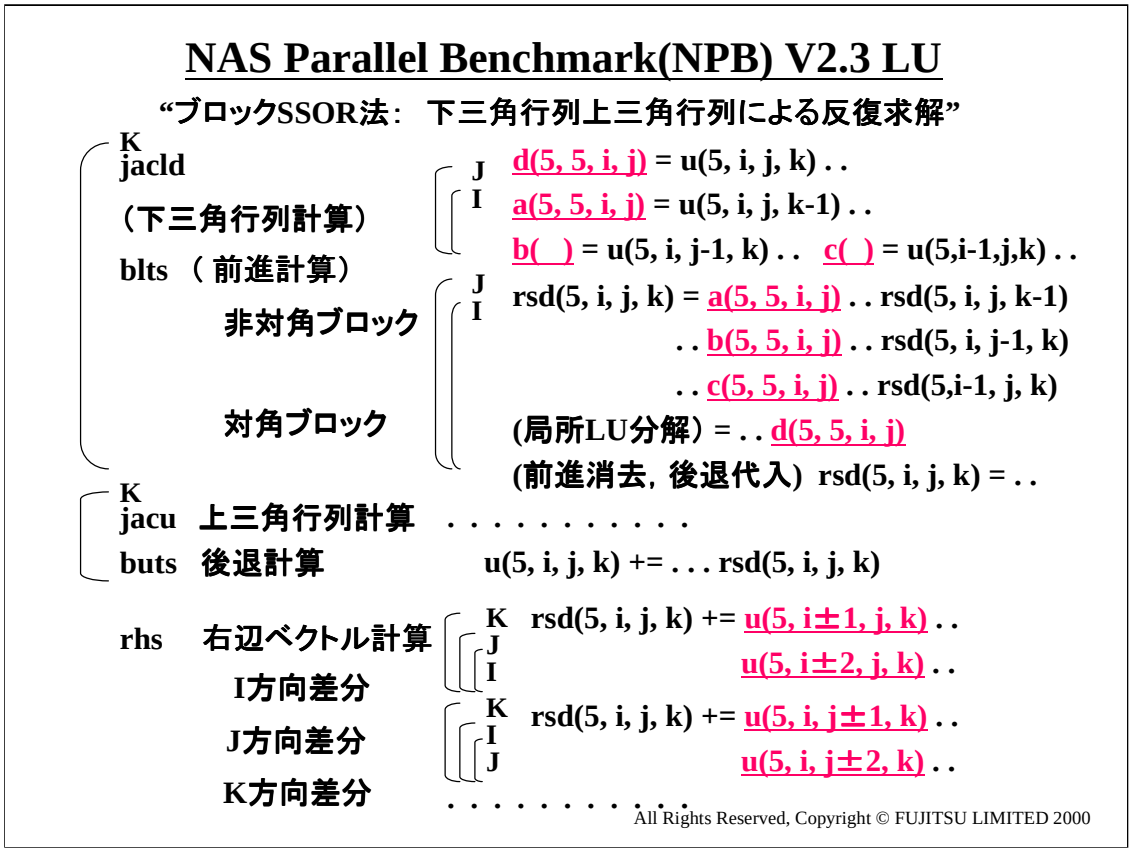
All Rights Reserved, Copyright © FUJITSU LIMITED 2000

前項のようにキャッシュデータ再利用化によるスカラチューニングが可能であり、作業配列a.b.cの再利用化のためには、I方向計算ではKループとJループについて、J方向計算ではKループとIループについて、またK方向計算ではJループとIルーについて、行列生成と求解計算の間のループ融合を行う。また、漸近解Uについては I方向とJ方向計算全体のKループの融合と、右辺ベクトル計算でも同様のループ融合を行う。

更に、漸近解Uの再利用がIJ面2次元分の計算を行った後であり、上記変更だけでは作業配列a,b,cの座標空間2次元分の領域がアクセスされ、依然6MB程度の領域であるため、漸近解Uとのキャッシュライン競合を生じ、性能劣化の原因となる。このため、不要となっている空間次元2次元分を無くす配列の次元縮小を行う。

以上のチューニングにより、Class-B規模ではプログラムのメモリサイズは約760 MBから約140MBに縮小された。

並列化は、スカラチューニング後の最外側ループをOpenMPで並列化することによって行い、その際に、各CPUが次元縮小した同一の作業配列a,b,cの領域を使用しないよう、プライベート化する。



次にNPB Verison2.3のLUのチューニング事例を取り上げる。

LU はSSOR法による反復計算を行っており、格子点インデクスの増加方向に、下三角行列に相当する格子点間の差分法による行列を生成し、3次元の各方向に回帰的な計算を最後の格子点まで計算したのち、今度は上三角行列を計算し逆の減少方向に計算する。この時各格子点には5つの変数があり、差分関係はブロック行列によって表されるため、対角ブロックの局所的なLU分解と前進消去と後退代入が行われる。インデクス増大の前進計算と減少方向の後退計算が終わると、右辺ベクトルの計算を行い、以上を繰り返す。図は逐次版ASISコードの構造を纏めたものである。

ASISコードでは、前進計算全体が一つのKループの中にあり、下三角行列は、座標空間IJの2次元と変数次元5×5の次元を持つ作業配列a,b,c,dに部分的に格納され、座標空間2次元までの次元縮小ができている。Class-Bの問題規模では作業配列のメモリサイズが合計8MB程度になり配列間のキャッシュライン競合がおきやすく、座標空間2次元分まとめて計算した後すぐに参照されるにもかかわらず、キャッシュ上から再利用ができない。しかし、1格子点について対応する下三角行列要素を計算しすぐにこの格子点についての前進計算をした後次の格子点に移るよう行列生成と前進計算の計算順序を変更することが可能である。このとき作業配列は1KB以下の領域しか使用せず、また、もはや座標空間については次元を持つ必要が無くなる。後退計算についても同様なことが言える。

右辺ベクトル計算ではBT同様に、I方向とJ方向差分計算部分の計算順序の変更により漸近解UをJ方向計算の際にキャッシュ上から再利用することができる。

NPB V2.3 LUのスカラチューニング

1) 配列 a, b, c, dのキャッシュデータ再利用化

ループ融合　Jループ，Iループ ： jacld, blts全体

jacu, buts全体

配列次元縮小 ：作業配列 a, b, c ,d(5, 5, i, j) ➡ (5, 5)

2) 配列 u のキャッシュデータ再利用化

ループ融合　Kループ ： rhs内 I方向 、J方向計算

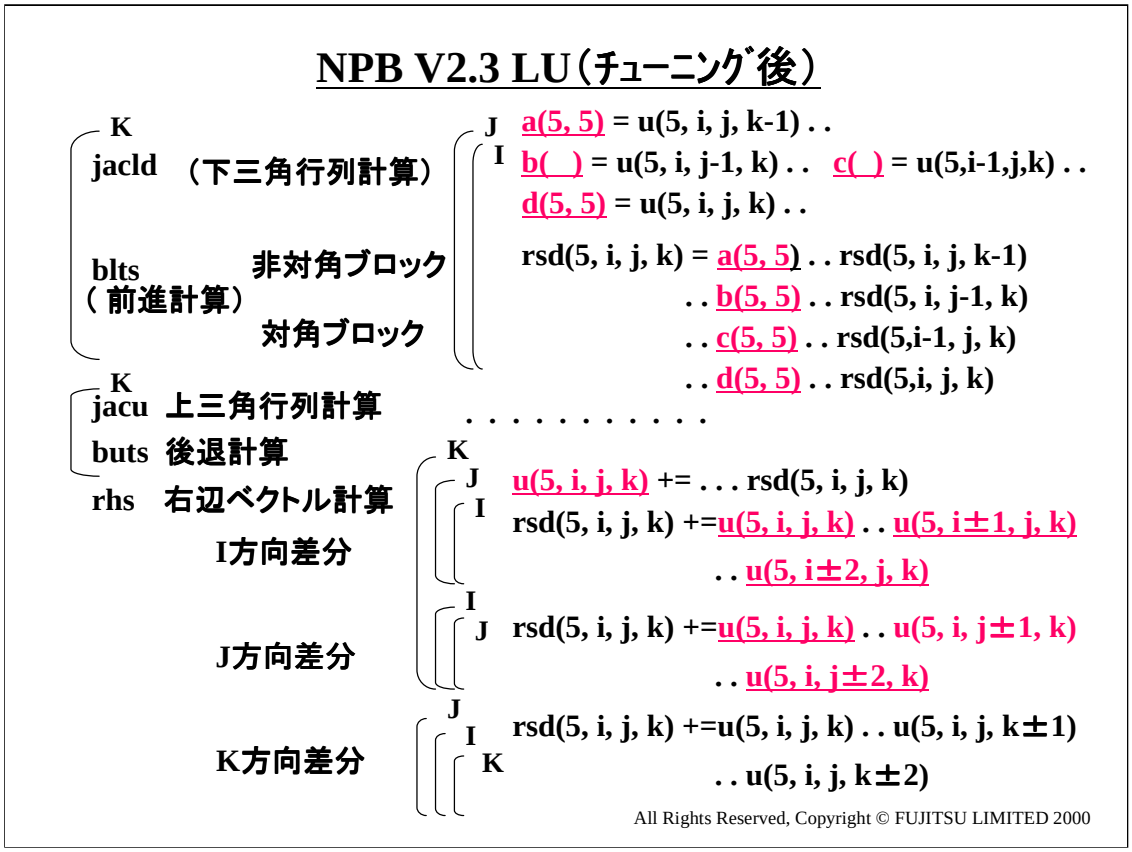
All Rights Reserved, Copyright © FUJITSU LIMITED 2000

スカラチューニングを纏めると、以下となる。

作業配列a.b.c.dのキャッシュ上からの再利用化が可能であり、このためにJループとIループについて、行列生成部と前進計算の間のループ融合を行う。また、作業配列の座標空間次元は不要であり、次元縮小しないことによるキャッシュライン競合を回避するために、次元縮小を行う。後退計算についても同様な変更を行う。

また、漸近解Uの再利用化のために右辺ベクトル計算のI方向計算とJ方向計算の間でKループの融合を行う。

もともと作業配列には座標空間について2次元分しか使われていなかったことが起因して、以上のチューニングではプログラム全体のメモリサイズはClass-B規模では約180MBから約140MBになるだけであり、あまり変わらない。

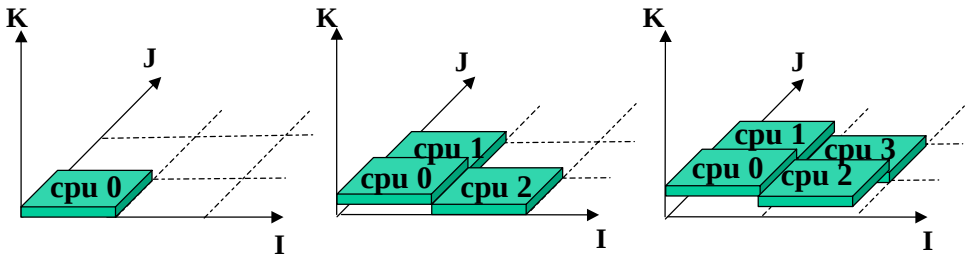


スカラチューニング後のLUの構造を示す。

NPB V2.3 LUの並列化(1)

下三角行列の前進計算・上三角行列の後退計算
各方向について回帰参照
→ I,Jの2次元方向で計算領域を分割、パイプライン方式により並列化

(1) **(2)** **(3)**



最初にcpu 0がK=1の面を計算し、終了フラグをセットする。

cpu 0はK=2の面を計算。cpu1, 2は cpu 0の終了フラグを確認し、K=1の面を計算する。

cpu 3はcpu 1とcpu 2の終了フラグを確認し、K=1の面を計算する。以下同様。

All Rights Reserved, Copyright © FUJITSU LIMITED 2000

LUの並列化について

SSOR法では各座標軸方向に回帰演算であるため、ベクトル計算機上では差分関係からI+J+K=定数となる平面（ハイパープレーン）上の格子点が独立に計算できる性質を利用して、この面内の格子点についてベクトル化を行っている。しかし、このときメモリアクセスは連続ではなくなり、スカラ処理には向かない。そこで、ハイパープレーン的な並列性はCPU間のみで利用し、CPU内では連続アクセスとなる回帰演算を残す、パイプライン方式と呼んでいる方法を使う。これは前項のスカラチューニングを生かす方法でもある。

図のように、IJ面内の二次元分割により、隣接するCPU がKインデクスについては1つずつずれた位相で計算を行うことによって並列処理を行うことができる。1次元分割であっても同様の並列化が可能であるが、2次元分割のほうが負荷バランスが良く高い並列性能が期待できる。これは、各分割軸のCPU数が1次元分割より少ないためである。

NPB V2.3 LUの並列化(2)

“OpenMPを用いたパイプライン方式の並列化”

```
!$OMP PARALLEL
                  . . . . . .
            終了フラグの初期設定
      do k = 2, nz - 1

        do while (lock(k, mype_x - 1, mype_y). eq. 0. or.
     &            lock(k, mype_x, mype_y - 1). eq. 0)
!$OMP      FLUSH ( lock )
        enddo

            IJ平面1面の計算

!$OMP      FLUSH ( rsd )
        lock(k, mype_x, mype_y) = 1
!$OMP      FLUSH ( lock )

      end do
!$OMP BARRIER
```

X方向隣とY方向隣の終了フラグが設定されるまで待つ

終了フラグの設定

All Rights Reserved, Copyright © FUJITSU LIMITED 2000

並列化はOpenMPで記述する。パイプライン処理を行うために、Kインデクス毎に、I方向J方向の各方向の隣接CPUから参照する計算終了フラグを採用する。各CPU の各Kインデクスでの計算前にこのフラグが隣接CPUによって設定されるまで待ち、自身の計算が終わると終了フラグを設定する。これをOpenMPとFortranで記述したものがコーディング例である。各部にあるOpenMPのFLUSH操作は、CPU間で共有されるデータが参照される際に常に最新の値となっていることを保証するために行っている。

スカラチューニング結果

NPB問題規模 ： Class-B

	オリジナル	チューニング	性能向上
V1 BT	**8279.4 sec.**	**1555.4 sec.**	**5.3倍**
V2.3 LU	**3923.6 sec.**	**833.8 sec.**	**4.7倍**

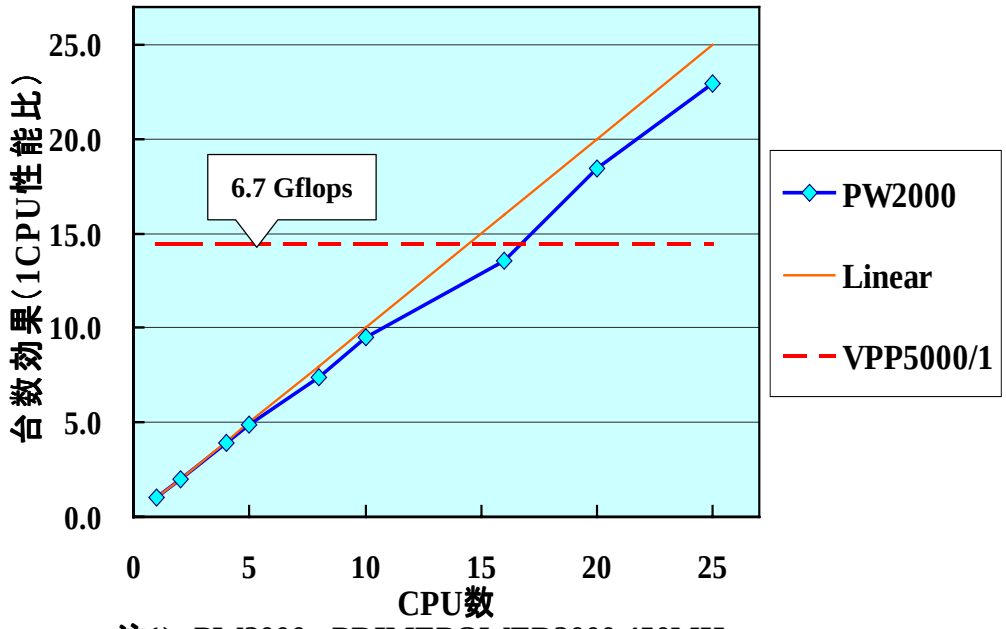
マシン: PRIMEPOWER2000 450MHz 1CPU

All Rights Reserved, Copyright © FUJITSU LIMITED 2000

スカラチューニングの効果をClass-Bの問題規模についてPRIMEPOWER2000 450MHz 1CPU上で測定した。

オリジナルコードに対してBTが5.3倍、LUが4.7倍の性能向上を達成することができた。スカラチューニング後のBTはNPB Version1の演算量定義によると463Mflopsに相当し、同LUはNPB Version2.3の演算量定義によれば 598MOPSに相当し、Version1の演算量定義によれば383Mflopsに相当する。

NPB V1 BT Class-B並列処理性能(チューニング)



注1) PW2000 : PRIMEPOWER2000 450MHz
2) 8CPU,16CPUでは格子を各100/8, 100/16と分割

All Rights Reserved, Copyright © FUJITSU LIMITED 2000

並列処理性能をPRIMEPOWER2000 450MHz上で測定し、1CPU性能比としてグラフ化した。かなり良好なスケーラビリティを示しており、25CPUまで殆ど飽和せずに性能が向上している。

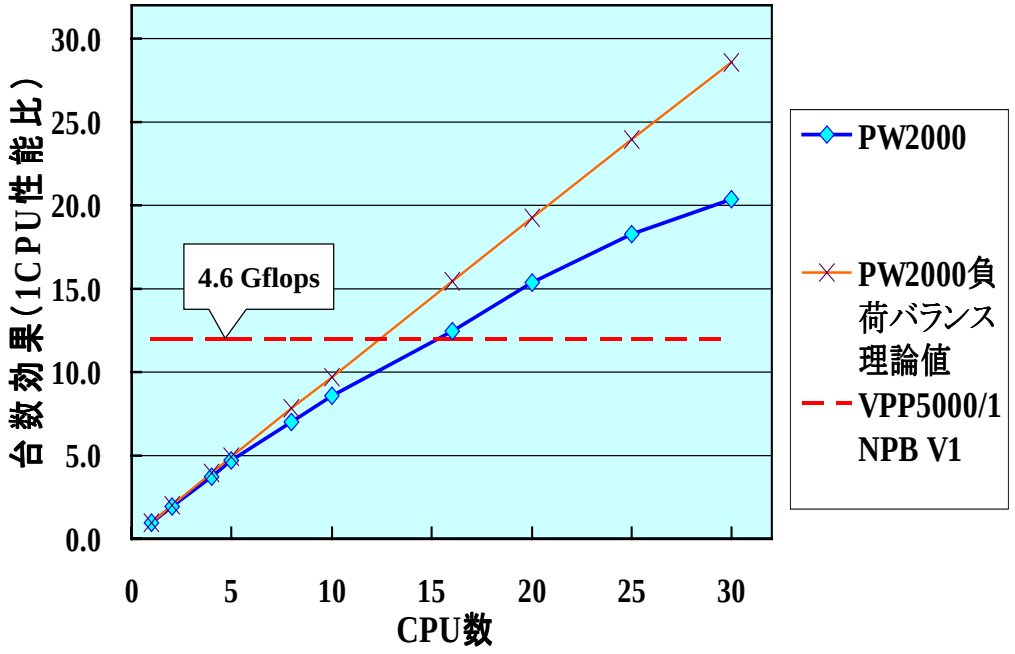
参考のため、VPP5000/1PE上でのベクトルチューニング版の性能(時間を使用)をPRIMEPOWER2000 1CPU性能に対する性能比の形で表示した。

およそ16CPUでVPP5000性能を超える結果となった。

なお、8CPU、16CPUでややスケーラビリティが落ちているのは、格子分割が割り切れず、負荷バランスが劣化しているためである。

また、スカラチューニングにより1CPUでも既にキャッシュデータ再利用化が十分にできており、一方、逐次処理で再利用化ができなかった部分は、前後の部分と並列化の際の分割軸が異なるため、CPU数を増加したとき参照データのサイズが縮小することによるオンキャッシュ化効果は殆ど期待できない。

NPB V2.3 LU Class-B並列性能（チューニング）



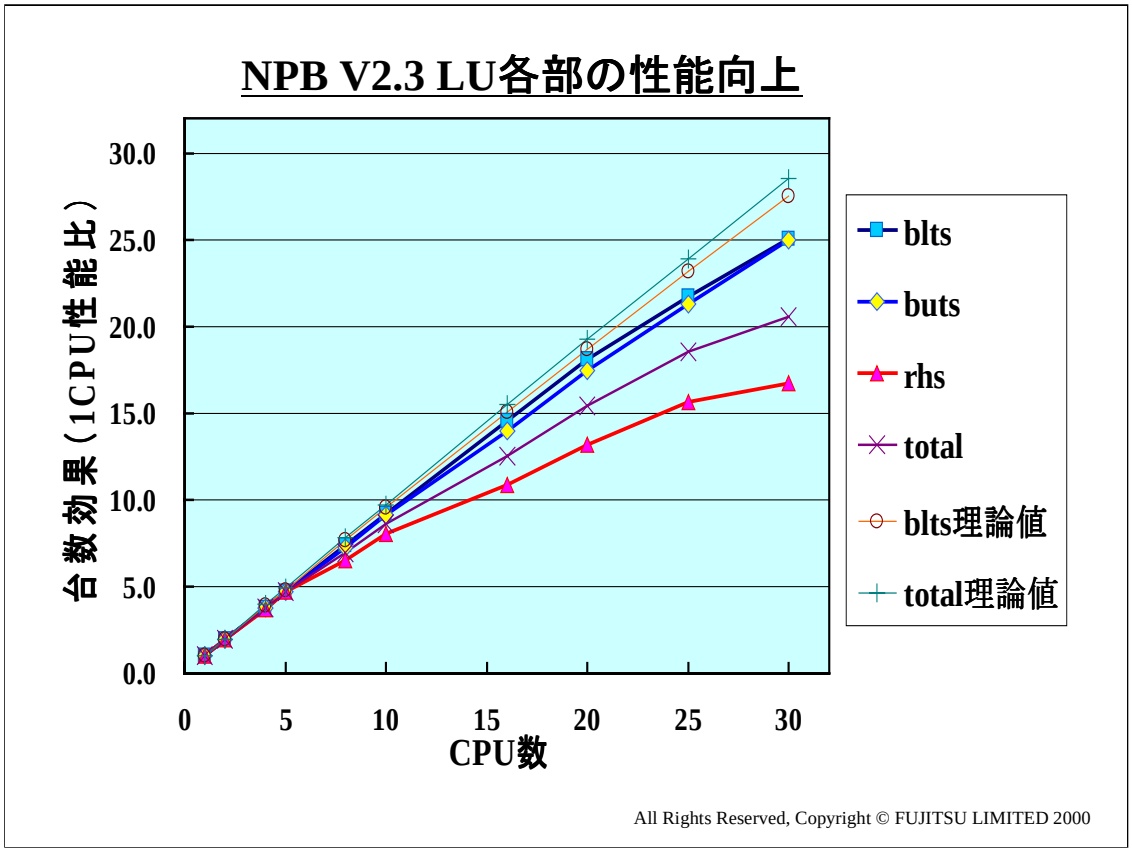
注) PW2000: PRIMEPOWER2000

All Rights Reserved, Copyright © FUJITSU LIMITED 2000

並列処理性能をPRIMEPOWER2000 450MHz上で測定し、1CPU性能比としてグラフ化した。パイプライン方式による負荷バランスだけを考慮し1CPU性能から算出した理論値と比べると、若干の伸びの悪さはあるが、かなり良好なスケーラビリティを示しており、30CPUまで性能が向上している。

参考のため、VPP5000/1PE上でのVersion1ベクトルチューニング版の性能（時間を使用）をPRIMEPOWER2000 1CPU性能に対する性能比の形で表示した。16CPUでVPP5000性能を超える結果となった。

なお、NPBVersion1とVersion2.3は全く同じ計算を行っており、比較対象となりうる。



LUのCPU数による性能向上が負荷バランス理論値（前項参照）よりも低い原因を知るため、各部の台数効果を測定した。

台数効果劣化の主原因は右辺ベクトル計算部分（rhs）にある。ここでは、他に比べてメモリアクセスが多いため、このことがメモリ競合を増加させ、rhsの台数効果を劣化させる原因となっていると考えられる。BTでは求解部で局所LU分解とブロック三重対角のLU分解が行われ右辺ベクトル計算の計算コストが相対的に低いのに対して、LUでは求解部の高コスト計算がほぼ局所LU分解のみであるため、右辺ベクトル計算の占める割合が高い。このため、全体性能への影響が目立つものと考えられる。

この結果から、今回は測定しなかったが、メモリアクセスの多いスカラチューニング前のプログラムを並列化した場合には、スケーラビリティが更に低下することが予想される。

パイプライン方式の並列化をした部分（blts, buts)の負荷バランス理論値は、CPU台数倍の直線に極めて近く、また実機測定値は負荷バランス理論値に比較的近い性能向上を示す。このことから、パイプライン方式の並列化が有効な方法であることが分かる。

なお、スカラチューニングにより1CPUでも既にキャッシュデータ再利用化ができており、また、逐次処理で再利用化ができなかった部分は、前後の部分と並列化の際の分割軸が異なるため、CPU数を増加したとき参照データのサイズが縮小することによるオンキャッシュ化効果は殆ど期待できない。

まとめ

1) メモリアクセスの局所性を高めることによって、キャッシュデータの再利用度を高め、スカラ性能とスカラ並列性能を向上させることが可能。

（ NPB BT、LUのスカラ性能向上比　各約5.3倍、4.7倍 ）

- **ループ融合によるデータセットのサブセット化**
- **作業配列の次元縮小**

2) SSORのパイプライン処理による並列化が効果的。

3) 課題 ： コンパイラの最適化能力の向上に期待。
手続き間にわたるループ融合、ループ再構成、による
メモリアクセスの局所性向上、他

All Rights Reserved, Copyright © FUJITSU LIMITED 2000

メモリアクセスの局所性を高めることがキャッシュデータの再利用を促進させ、スカラ性能向上にとって効果的であった。その際の手法として、アプリの並列性を利用するなどしたループ融合によるデータセットのサブセット化や、作業配列の次元縮小を行い、効果があった。

ただし、どのようなアプリケーションでも今回の手法で高い性能向上が得られるわけではなく、性能向上が難しい場合もあると考えられ、性能向上はアプリケーションに依存する点に留意する必要がある。

なお、LUの結果から、BTとLUともに、スカラチューニングは並列処理のスケーラビリティの良好さにも寄与していると推測される。

各座標軸方向に回帰演算となるSSOR法はスカラチューニングを生かすパイプライン方式による並列化が効果的であった。

SSOR法やSOR法は、ベクトル化のためにハイパープレーンの計算順序とするよう大幅なループ再構成やアクセスリストの初期化が必要であったように、パイプライン方式によるスカラ並列化のためには手続き分割と同期処理のプログラミングが必要となった。両者を比べて、プログラミングの負担という意味では大差がないと感じている。

コンパイラの課題として、より容易に性能向上が得られるよう、手続き間にわたるループ融合やループ再構成によってメモリアクセスの局所化を計る改善を行うことが期待される。