TX7/AzusA Fortran

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　　山本　秀喜　左近　彰一

概要

TX7/AzusA システムは、EPIC（Explicitly Parallel Instruction Computing）アーキテクチャを採用した Intel の次世代最新の Itanium™プロセッサを搭載しており、ソフトウェア・パイプライニング支援機能、レジスタ・ローテート、データ・プリフェッチなどのハードウェア機能を有している。TX7/AzusA Fortran コンパイラはそれらのハードウェアの性能を充分に引き出すための機能を備えている。本文では、それらの機能について説明する。

1. はじめに

TX7/AzusA Fortran コンパイラは、Itanium™プロセッサのもつハードウェアの機能をいかんなく引き出すため、ハードウェアに密着した最適化、ループ最適化、ソフトウェアパイプライニング、キャッシュ利用の最適化、OpenMP による並列化機能、プロファイル利用による最適化を有したコンパイラであり、Intel 社との技術提携をベースに NEC が開発を行っています。本稿では、これらの言語仕様とコンパイラ技術を中心に解説します。

2. Fortran コンパイラ

TX7/AzusA の Fortran は、ISO/IEC 139-1:1997 規格（JIS X 3001-1:1998)、通称 Fortran95 に準拠し、さらに拡張機能を追加したものです。以下のような一般的な Fortran の拡張機能を備えています。

- 255 字まで利用可能な変数名
- 固定形式、自由形式、拡張固定形式プログラムの記述
- 1/2/4/8 バイト整数、単精度/倍精度/四倍精度の実数と複素数
- CRAY 形式ポインタ
- automatic/static/volatile 属性
- C 言語互換文字列
- %VAL、%REF 関数
- コンパイラ指示行

以下ではコンパイラの最適化機能を中心に説明します。

2.1 デフォルトオプションによる最適化機能

デフォルトオプションでは、最適化レベル-O2(-O1,-O も-O2 と同じ)の最適化が有効になります。この最適化レベルでは、主に、

- 命令の並列スケジュール機能
- ソフトウェアパイプライニング
- レジスタ・ローテーション

を行います。以下で、これら最適化機能について説明します。

2.1.1 命令の並列スケジューリング機能

Itanium™の EPIC アーキテクチャに対応して、CPU 内の複数の演算器が可能な限り同時並列に実行できるように命令をコンパイル時にスケジューリングする機能です。EPIC による性能を最大限に引き出します。

2.1.2 ソフトウェア・パイプライニング機能

ソフトウェア・パイプライニング機能とは、各ループの繰り返しを一つずつ行うのではなく、ループの繰り返しをオーバーラップして行うような最適化です。ループ内の命令が並列にスケジューリングされるようになるため、大きな高速化効果が得られます。コンパイラは Itanium™のレジスタ・ローテーション、プレディケーションやカウント指定ループ分岐機能を活用して、モジュロースケジューリングを用いた効率の良いコンパクトなコードを生成し、命令キャッシュの有効利用に役立てます。

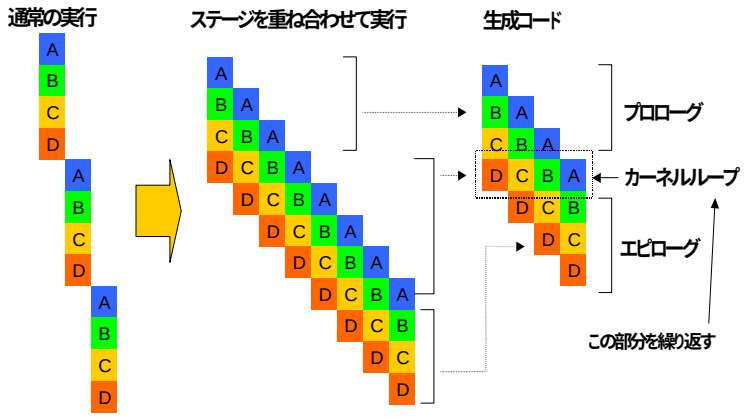

図１ ソフトウェアパイプライニング

(1) コンパイラによるソフトウェア・パイプラインの実際

ソフトウェアによるパイプライン化とは、ハードウェアによるパイプライン化と同じような形で、ループの繰り返しをオーバーラップさせることによって、依存関係のない命令を並列に実行するテクニックです。コンパイラは、命令の実行時間を考慮して、ハードウェアのパイプライン化と同じようにステージを分解します。例えば、次のループは４つのステージに分解されます。

例１　ソフトウェアによるパイプライン化のステージ分割の概念

```
do i=1,n
    a(i) = b + c(i)
end do
  ↓
stage1: ld4  r4 = [r5],4    // c(i)
stage2:                     // empty stage
stage3: add  r7 = r4,r9     // b + c(i)
stage4: st4   [r6] = r7,4   // a(i)
```

次に、パイプライン化された５回の繰り返しの概念図を示します。

例２　各繰り返しの実行概念図

```
    1     2     3     4     5          実行サイクル数
-------------------------------------------------
 ld4                                    X
      ld4                               X+1
 add        ld4                         X+2
 st4  add         ld4                   X+3
      st4   add         ld4             X+4
            st4   add                    X+5
                  st4   add              X+6
                        st4              X+7
```

このソフトウェアパイプライン化されたループは、ソフトウェア・パイプライン・ループと呼ばれ、次の例のように、プロローグ、カーネルおよびエピローグの３つのフェーズから構成されます。

例３　ソフトウェア・パイプライン・ループ概念図

```
    1     2     3     4     5           フェーズ
-------------------------------------------------
 ld4
      ld4                               プロローグ
 add        ld4
-------------------------------------------------
 st4  add         ld4                   カーネル
      st4   add         ld4
-------------------------------------------------
            st4   add
                  st4   add             エピローグ
                        st4
```

このようにして、ソフトウェア・パイプライン化を行うことで、ソフトウェアパイプライン化をしなかった場合、ループの実行に 4*n サイクル必要であった実行時間が、

6+n の実行時間に短縮されます（キャッシュ・ヒット・ミスによるペナルティ時間を除く）。

(2) レジスタ・ローテーション

レジスタ・ローテーションとは、ソフトウェア・パイプライニングされたループ内のハードウェアによるレジスタのリネーム機能です。この機能を活用することで、ソフトウェア・パイプライニングを効率よく実行することが可能になります。ソフトウェア・パイプライン・ループの最後で、ソフトウェア・パイプライン分岐命令を実行すると、レジスタがローテートされます。レジスタがローテートされるとレジスタ X の値がレジスタ X+1 に移動したように見えます。レジスタ・ローテーションは、プリディケート・レジスタ（P16～P63）、浮動小数点レジスタ（f32～f127）、および汎用レジスタ(r32～r127)の 3 つのローテート・レジスタ・ファイルを対象に行われます。　次に、レジスタ・ローテーションの例を示します。

例４レジスタ・ローテーションの概念

```
L1:   (p16) ld4    r35 = [r5],4    // C(I)
      (p18) add    r38 = r37,r9    // B+ C(I)
      (p19) st4    [r6] = r39,4    // A(I)
            br.ctop.stpk L1;;
```

ld4 命令で r35 に書き込む値は、2 回のカーネル繰り返し（および 2 回のローテーション）の後 add 命令によって、r37 として読み出されます。同様に、add 命令で書き込まれた r38 の値は、1 回のカーネル繰り返しの後 st4 命令によって、 r39 として読み出されます。

また、レジスタ・ローテーションと同様の効果は、ループをアンローリングし、レジスタのリネームを明示的に行うことでも実現可能ですが、次の例のように、各ループの繰り返しの間で待ち時間が発生してしまいます。そのため、ソフトウェア・パイプライニングの方が、より高い効果を得ることができます。

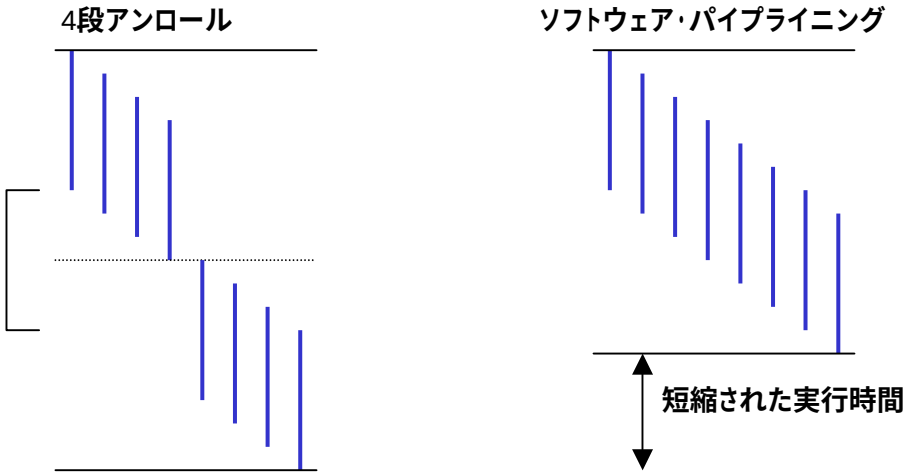

図 2　アンロールとソフトウェア・パイプライニングの比較

2.2 高度最適化機能

高度最適化機能は、オプション-03 を指定することにより、最適化レベル-02 の最適化に加え、以下の最適化が有効になります。

- ループ最適化
- キャッシュ利用の最適化
- ループ変換
- データプリフェッチ
- スカラーリプレースメント

以下で、これら最適化機能について説明します。

2.2.1 ループ最適化機能

プログラムの実行時間の主要な中心となるループに対して、多重ループの一重化や入れ替え、隣接するループの融合、アンローリングなどの種々のハイレベル最適化を行っています。なお、これらの最適化は DO ループに対してだけでなく、Fortran90 で追加された配列式や配列代入文にも適用し、効果を高めています。

2.2.2 キャッシュ利用の最適化機能

アプリケーションに対してこれらのループ変換を行うことによって、ループ制御のオーバーヘッドの削減だけでなくデータ参照の局所性を大きく向上させることができます。データ参照の局所性が向上することにより、キャッシュ・ヒットの確率が高くなり、メモリアクセスのオーバーヘッドを減らすことができます。また、ループ内の各変数へのアクセスに対しては、データ・プリフェッチが重要な役割を果たします。

2.2.3 ループ変換の例

ここで、いくつかの例を紹介します。

次の例では、外側ループと内側ループを入れ替えることで、配列 a が連続アドレスで参照されキャッシュ利用の効率が高まります。また、配列ｂは内側ループで一度のみのアクセスになるため、このループ変換によりデータ参照の局所性を向上させています。

例５　ループ変換１

```
do i = 1, 1000                          do j=1, 1000
    do j = 1, 1000                          do i = 1, 1000
        c(j) = c(j) + a(i,j) * b(j)   →         c(j) = c(j) + a(i,j) * b(j)
    end do                                  end do
end do                                  end do
```

次の例では、ループを融合させることにより、配列 a へのアクセスを１つのループ

内で行うようになり、キャッシュを効率よく利用することが可能になります。

例６　ループ融合

```
do i = 1, 1000
    a(i) = x
end do
do i = 1, 1000
    c(i) = a(i-1) + d(i-1)
    d(i) = c(i)
end do
```

→

```
do i = 1, 1000
   a(i) = x
      c(i) = a(i-1) + d(i-1)
     d(i) = c(i)
end do
```

2.2.4　データ・プリフェッチ

データ・プリフェッチは、メモリ・アクセス・レイテンシを隠す効果的な手法です。これは、メモリへのアクセス時間と、計算時間や他のメモリへのアクセス時間とをオーバーラップさせます。データ・プリフェッチでは、データ参照用のプリフェッチ命令を挿入することにより、参照すべきデータをデータが実際に必要になるときまでに、可能な限りキャッシュへ移動させます。以下に、データ・プリフェッチを挿入した場合の例を示します。挿入後のソースにおいて、ｋは実際のデータ使用時にキャッシュ・ヒット・ミスが生じないようにするためのメモリアクセス時間をコンパイラが考慮して計算した値です。また、ソフトウェア・パイプライン・ループ内のデータ･プリフェッチ命令の場合は、レジスタのローテート機能を利用し、複数のデータ・プリフェッチを１つの命令で実現しています。

例７　データ・プリフェッチ命令を挿入した場合の概念

```
do j = 1, n
    do i = 1, m
        a(i,j) = a(i,j) + b(i) + x
    end do
end do
```

↓

```
do j = 1, n
    do i = 1, m
        a(i,j) = a(i,j) + b(i) + x
        prefetch( a(i+k,j) )
        prefetch( b(i+k) )
    end do
end do
```

2.3　手続き間最適化機能

手続き間の最適化機能はオプション（IP0）を指定することにより、手続きのインライン展開や、手続き間解析を行うことにより、定数伝播や、不要なコードの削除など、最適化の一層の促進を行います。

2.4　プロファイル利用の最適化

プログラムを一度実行させて、各ブロックの実行頻度や IF 文の分岐情報などのプロファイル情報を採取し、その情報を利用してもう一度コンパイルを行うことにより、実際のプログラムの動きを考慮したインライン展開、命令の並べかえや、ソフトウェアパイプライニングの促進の最適化を行うことができます。類似のデータでプログラムを繰り返し走行する場合に効果があります。プロファイル利用の最適化は、一回目のコンパイルでオプション‘ prof_gen ’、二回目のコンパイルでオプション‘ prof_use ’を指定することにより利用が可能です。

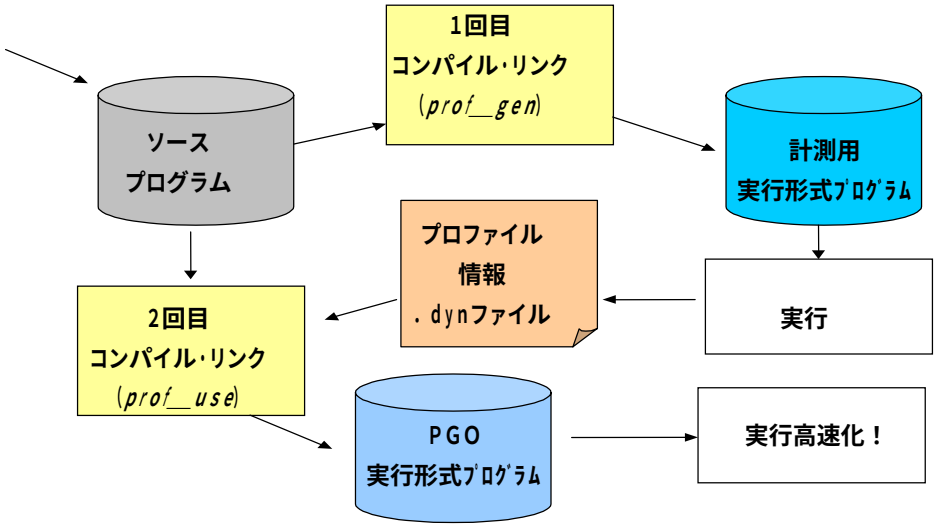

図３　プロファイル利用最適化

例えば、次のループは、IF 文がループ内に存在するため、ソフトウェア・パイプライニングを適用できないループですが、プロファイルを利用した最適化を使うと、IF 文の偏りがコンパイル時に分かるため、ソフトウェア・パイプライニングが可能になります。

例 8　プロファイル利用最適化の例

```
do i = 1, n
    if(b(i).eq.0.0) then
        a(i) = a(i)*c(i)
    else if( a(i) .ne. 0.0 ) then
        a(i) = c(i) + c(i)/d(i)
    end if
end do
```

3　並列化機能

共有メモリ並列処理向けの Fortran API として設計された OpenMP 仕様に準拠した並列処理機能と自動並列化機能を提供しています。OpenMP を用いることにより、異なるベンダ間の並列アプリケーションの移植が容易に行えます。本コンパイラでは OpenMP の各種指示行、ライブラリおよび環境変数をサポートしており、16CPU をフルに用いた並列処理を行うことができます。OpenMP の実装は、Linux のスレッドおよび Pthread ライブラリを用いて実現しています。なお、並列化機能の詳細は次回ご説明する予定です。

4.　MPI

MPI は Fortran、C/C++から呼び出し可能な分散メモリプログラミング並列処理インタフェースライブラリです。

ユーザが分散メモリを意識し、タスク構造の定義から同期処理、分散メモリ間のデータの転送まで、すべてを明示的に行う「メッセージ交換」の考え方に基づいています。TX7/AzusA では、共有メモリ内でのデータ転送のオーバーヘッドが非常に小さいことを利用し、最適な処理を行うように考慮し、スケーラビリティの高い並列処理を実現しています。

5. おわりに

NEC では、並列化技術を注入し、今後もコンパイラの機能、最適化および性能強化を行っていく予定です。本稿が TX7/AzusA Fortran コンパイラの理解の一助になれば幸いです。