5Q-07 SR8000向けコンパイラへの配列リージョン解析の実装

家坂 聡　中島 恵　布広 永示
(株)日立製作所　ソフトウェア開発本部

1　はじめに

スーパーテクニカルサーバ HITACHI SR8000 は複数のノードから構成される並列型スーパーコンピュータであり、各ノードは複数の CPU から構成されている。SR8000 向けコンパイラはノード内自動並列化機能を持ち、ループを自動的に並列化する。並列化適用範囲の拡大のため、SR8000 コンパイラの配列データフロー解析に配列リージョン解析を実装した。本報告では実装した配列リージョン解析の解析手法、その解析結果を使った並列化例、およびその効果を報告する。

2　配列リージョン解析

配列リージョン解析は、ループ中の配列要素参照点が参照する部分配列を求める解析で、この部分配列を配列リージョンと呼んでいる。SR8000 コンパイラの配列リージョン解析では、配列リージョンを整数線形不等式系で表現する。例えば以下のようなループを考える。

```
DO I=1,N
    DO J=1,M
        A(J+1,2*I)=...
```

図 1:サンプルプログラム1

最内側Jループ一回の繰り返しで定義される配列リージョンは以下のように表現される。

$$R1=\left\{(r1,r2)\,\middle|\,\begin{matrix}1\le j\le M, r1=j+1\\ 1\le i\le N, r2=2*i\end{matrix}\right\}$$

このリージョンからjを消去することで、Iループ一回の繰り返しで定義される配列リージョンが得られる。

$$R1=\left\{(r1,r2)\,\middle|\,\begin{matrix}2\le r1\le M+1\\ 1\le i\le N, r2=2*i\end{matrix}\right\}$$

さらにiを消去することでループネストの全繰り返しで定義される配列リージョンが求められる。

$$R1=\left\{(r1,r2)\,\middle|\,\begin{matrix}2\le r1\le M+1\\ 2\le r2\le 2*N, r2=2*\alpha\end{matrix}\right\}$$

ここでαはiの消去の際に新たに導入された変数である。

配列リージョン解析には、このような変数の消去の他に、配列リージョン間の和や積、配列リージョンの補集合を求める操作関数を用意している。変数の消去と積、補集合の計算には、線形計画法の一種であるオメガテスト[1]を用いている。配列リージョンの和はヒューリスティクスを用いて求めている。

3　配列プライベート化

配列プライベート化は、配列のコピーを各 CPU に用意し、各 CPU が同一配列を参照しない様にすることによって並列化適用範囲を拡大する変換である。配列リージョン解析を実装する目的は、配列プライベート化を実現し、より外側のループでの並列化を目指すことである。

アルゴリズム：

(1) ループ中で使用される可能性のある USE リージョンを求める。これは配列要素使用点に対して求める。

(2) ループ中で定義される可能性のある MOD リージョンを求める。これはループ中の配列要素定義点に対する配列リージョンの和を計算することで求められる。

(3) ループ中で必ず定義される KILL リージョンを求める。これはループ中の配列要素定義点に対する配列リージョンの積を計算することで求められる。

(4) ループ中で露出使用されるEUSEリージョンを求める。USE−(USE∩KILL)を計算することで求められる。

(5) MOD∩EUSE を計算する。結果φであればプライベート化可能。この時 EUSE が初期値保証の必要なリージョンとなる。

Implementation of Array Region Analysis for SR8000 Compiler
Satoshi Iesaka, Kei Nakajima, Eiji Nunohiro
Software Development Center, Hitachi Ltd.,

4　ループイタレーション分割

ループのどの繰り返しでも同一配列要素を使用するような場合、その要素を定義するイタレーションをループ外に追い出すことにより並列化が可能になる場合がある。

```
DO 126 I=1,NX1
   DNX(I)=DNX(I)/DNX(NX1)
END DO
```

図 2:SPEC95 FP wave5

上記はSPECfp95のwave5の一部である。I=NX1の時にDNX(NX1)が定義されるため、このままでは並列化ができない。これを図3のように変換すればDO 126で並列化が可能となる。この例では、ループピーリングを適用するのと同じであるが、配列リージョンを使うと、ループ外に追い出すイタレーションがイタレーションの最初や最後に限定されなくなる。

```
DO 126 I=1,NX1-1
   DNX(I)=DNX(I)/DNX(NX1)
END DO
DNX(NX1)=DNX(NX1)/DNX(NX1)
```

図 3:イタレーション分割後のループ

アルゴリズム：

(1) 依存関係にある配列要素参照点の配列リージョンを求める。図3の例では以下となる。

$$DEF=\{(r)\,|\,1\le r\le NX1\}\quad USE=\{(r)\,|\,r=NX1\}$$

(2) DEF リージョン、USE リージョンの積を求めて共通にアクセスされる配列リージョンを求める。図3の例では以下のようになる。

$$DEF\cap USE=$$
$$\{(r)\,|\,1\le r\le NX1\}\cap\{(r)\,|\,r=NX1\}=$$
$$\{(r)\,|\,r=NX1\}$$

(3) (2)で求めた配列リージョンに定義点の配列添字、ループ上下限からなる制約を追加して、rを消去する。ここで求められたiの制約は、依存のあるループイタレーションを表している。このイタレーションをループ外に追い出すことで並列化が可能となる。

$$\{(r)\,|\,r=NX1, r=i, 1\le i\le NX1\}\rightarrow\{(i)\,|\,i=NX1\}$$

5　評価

NAS Parallelベンチマークを用いてSR8000 1ノードで性能を測定した。並列化しない場合を1とし、配列リージョン解析を使う場合と使わない場合の並列化による性能向上率を示す。

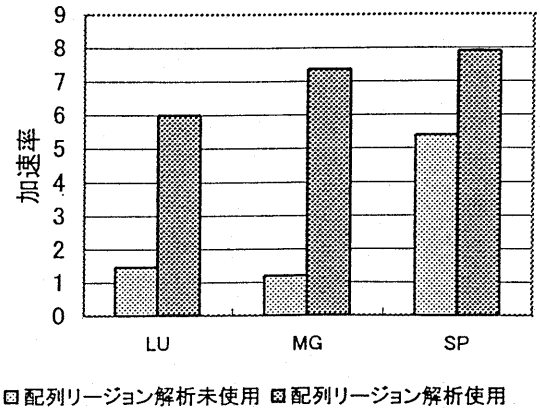

図 4:NAS Parallel ベンチマークでの評価

図5を使ってMGについて調べてみる。配列リージョン解析を使わない場合、do i2、do i3ループでは、r1,r2に対して毎回同じ配列要素に定義を行なうため並列化できず、 do i1ループでの並列化しか行なうことができない。配列リージョン解析によりr1,r2に配列プライベート化が適用されると、do i3ループで並列化が可能となる。

```
do i3=2,n3-1
   do i2=2,n2-1
      do i1=1,n1
         r1(i1) = r(i1,i2-1,i3)+...
         r2(i1) = r(i1,i2-1,i3-1)+...
      enddo
      do i1=2,n1-1
         ...
```

図 5:MG ベンチマークの一部

6　おわりに

SR8000コンパイラの配列リージョン解析の実装方式、それを使った並列化例について述べ、NAS Parallel Benchmarkによる測定結果を示した。

参考文献

[1]William Pugh. The Omega test: a fast and practical integer programming algorithm for dependence analysis. Supercomputing '91