5Q-06

SR8000におけるOpenMPの実装と評価

西谷 康仁†, 根岸 清†, 太田 寛‡, 布広 永示†
†(株)日立製作所ソフトウェア開発本部, ‡(株)日立製作所システム開発研究所

1 はじめに

従来共有メモリ向けの並列プログラムを作成するためには、各プラットフォームベンダー固有の並列化指示仕様やライブラリを用いなければならず、移植性のある並列プログラムを作成することが困難であった。この問題を解決するために、業界標準を目指した共有メモリ向け並列化仕様であるOpenMP [1]が策定された。現在多くのプラットフォーム上で商用・非商用のコンパイラ [2][3]がサポートされつつある。

我々は、スーパーテクニカルサーバ日立SR8000用コンパイラにOpenMPを実装した。本実装では、SR8000のハードウェア機構を用いてOpenMPの並列実行を効率よく実現すること、及び診断メッセージによりユーザの並列化作業を支援することを可能とした。

本稿では、SR8000におけるOpenMPの実装方式と評価について述べる。

2 コンパイラの概要

本コンパイラは、SR8000用のネイティブコードを出力する。SR8000は，ノードと呼ばれる、プロセッサを複数搭載した計算ユニットを基本単位として構成される。各ノードは、複数のInstruction Processor(IP)から構成され、共有メモリ型の並列処理システムを構成する。本コンパイラにおけるOpenMPの目的は、ノード内での並列化を制御することである。

OpenMPは、Fortran及びC, C++向けのAPIが定められている。本コンパイラでは、それぞれの言語に対応したフロントエンドが、OpenMPの指示文を取り込み、中間言語に変換した上で、共通のバックエンドに渡す。

バックエンドは、中間後に埋め込まれたOpenMPの指示文を解析して、次のような並列化変換を行なう。

- OpenMPの並列実行構造であるPARALLELリージョンを各スレッドに並列実行させるためのコードを出力する
- DO, SECTIONS等の処理分割指示に対応して、ループやセクションの実行が各スレッドで分担されるようにコードを変換する
- PRIVATE、REDUCTION等の指示に従い、変数のスレッド固有領域への割り付けや、リダクションコードの出力を行なう
- その他の同期指示文等に対して必要なコード変換を行なう

本コンパイラでは、OpenMP Fortran API 1.0仕様のフルセットをサポートした。ただし、ネスト並列化やdynamic thread adjustmentは実装していない。

3 OpenMP指示文の実装方式

3.1 PARALLEL指示文

PARALLEL指示は、並列実行範囲を指定する。コンパイラは、PARALLEL指示の開始位置でスレッドを起動(fork)し、終了位置で終結(join)するコードを生成する。SR8000用OpenMPでは、SR8000の持つ協調型マイクロプロセッサ機構を用いることにより、スレッドの起動・終結を高速に行なうことを可能とした。本機構を用いたPARALLELリージョンの実行の様子を図1に示す。

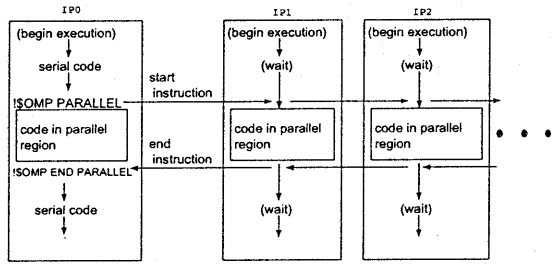

図1: SR8000におけるPARALLELリージョンの実行

協調型マイクロプロセッサ機構では、プログラムの開始時点でスレッドが各IPに固定的に割り付けられる。マスタースレッドがPARALLELリージョンの先頭に到達すると、マスターIP(IP0)は、他のIPに対して「スタート命令」を発行する。「スタート命令」を受けた各IPは、同時にPARALLELリージョンの開始アドレスを受け取り、すぐに実行を開始する。

PARALLELリージョンが終了すると、各IPは「エンド命令」を発行し、同期をとる。その後、マスターIPのみが実行を継続し、他のIPは再び起動待ち状態に入る。

Implementation and Evaluation of OpenMP for Hitachi SR8000.
Yasunori NISHITANI†, Kiyoshi NEGISHI†, Hiroshi OHTA‡, Eiji NUNOHIRO†
†Software Development Division, Hitachi Ltd.,‡Systems Development Laboratory, Hitachi Ltd.

このように本実装では、OpenMP のスレッドの生成が、既に起動している IP に対するハードウェア命令の発行で実現できるため、スレッド生成のオーバーヘッドを低くすることが出来る。

3.2 DO directive

OpenMP では、DO 指示文によりループを並列化する。コンパイラは、基本的にはユーザの指示通りに並列化を行なうが、あるループが並列化可能かどうか判定することや、ループ内の全ての変数について PRIVATE 化が必要かどうか判定するといった作業は、ユーザにとって楽なものではない。

そのため、本コンパイラでは、ユーザの指示通りに並列化を行なう一方で、コンパイラがループ並列化や、PRIVATE 化解析を行ない、その結果を診断メッセージとして出力する機能を備えた。診断メッセージの例を図 2 に示す。

```
1:       subroutine sub(a,c,n,m)
2:       real a(n,m)
3: !$OMP PARALLEL DO PRIVATE(tmp1)
4:       do j=2,m
5:         do i=1,n
6:           jm1=j-1
7:           tmp1=a(i,jm1)
8:           a(i,j)=tmp1/c
9:         enddo
10:      enddo
11:      end

(diagnosis for loop structure)
   KCHF2015K
             the do loop is parallelized by
             "omp do" directive. line=4
   KCHF2200K
             the parallelized do loop contains
             data dependencies across loop
             iterations. name=A line=4
   KCHF2201K
             the variable or array in do loop is
             not privatized. name=JM1 line=4
```

図 2: 診断メッセージの例

この機能により、ユーザが間違った並列化指示をすることを防ぐことが出来る。また、ユーザ指示が無いループに対して、ループが並列化可能かどうか、あるいは各変数について PRIVATE 化する必要があるかどうかを出力する機能も持つため、逐次プログラムを OpenMP により並列化する作業において役立つ。

4 性能評価

実装した OpenMP コンパイラの評価を、NAS Parallel Benchmarks(NPB) を用いて行なった。使用した NPB のバージョンは、NPB2.3-serial で、実行文は変更せずに、OMP 指示文を追加するのみで並列化を行なった。ただし、OpenMP で規定されている仕様だけでは並列化出来ない部分については、ソースの書き換えも行なった。問題サイズは、class A を用いた。

測定は、日立 SR8000 の 1 ノードを用い、逐次実行と 8 プロセッサでの並列実行時の実行時間を比較した。

測定結果を図 3 に示す。

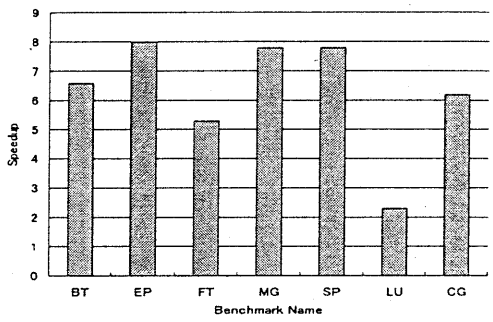

図 3: Speedup of NPB2.3-serial

図より、LU を除く 6 題で、8 プロセッサでおよそ 5.3〜8.0 倍の加速率が得られていることが分かる。LU について加速率が悪くなっているのは、OpenMP の指示文だけでは並列化出来ないウェーブフロント型のループ (ループネストのどのレベルにもループ運搬依存があるようなループ) があり、そのループが並列化出来ていないためである。そこで、並列化出来なかったループを含むサブルーチンのみを本コンパイラの自動並列化機能により並列化し、その他のサブルーチンは OpenMP で並列化したところ、8 プロセッサでの加速率は 6.3 倍となった。

5 おわりに

本稿では、Hitachi SR8000 のノード内での並列化向けに、OpenMP を実装したコンパイラについて、その実装方式と性能評価について述べた。本コンパイラにより NAS Parallel Benchmarks を OpenMP を用いて並列化したところ、8 プロセッサでおよそ 5.3〜8.0 倍の加速率を得ることが出来た。

今回の OpenMP の評価を通して、OpenMP に不足している機能があるため並列化出来ないループがあった。これらのループは、実プログラムにおいてよく現れるパターンのループであるため、指示文のみで並列化出来ることが望ましい。今後、これらのループが OpenMP の指示文のみで並列化出来るように OpenMP の拡張仕様を提案していきたい。

参考文献

[1] OpenMP Architecture Review Board. OpenMP Fortran Application Program Interface, Oct 1997 1.0.

[2] Eduard Ayguadé, Marc Gonzàlez, Jesús Labarta, Xavier Martorell, Nacho Navarro, and José Oliver. NanosCompiler. A Research Platform for OpenMP extensions. In *EWOMP'99*.

[3] Mitsuhisa Sato, Shigehisa Satoh, Kazuhiro Kusano, and Yoshio Tanaka. Design of OpenMP Compiler for an SMP Cluster. In *EWOMP'99*.