情報処理学会第37回(昭和63年後期)全国大会

並列プロセッサによる並列化Cコンパイラ

6Y-1

小池　稔　　大川　善邦

大阪大学

1．　緒言

プログラムの開発は、エディット・コンパイル・実行の繰り返しによって行なわれる。その繰り返し数は、実際にプログラムがエラーなく実行されるまでに、数回もしくは数十回以上に及ぶ。コンパイル以外の２つの作業については、ソースプログラムの入力、初期値のキーボード入力、ディスプレイからの結果の読み取り等の対話的な作業が多い。これに対しコンパイル作業においては、開始時に指示を与えた後はコンパイルが終了するまで、プログラマは待ち状態になる。コンパイルの回数が多くなれば、この無駄時間は、無視できないほど大きくなり、当然プログラムの開発能率は落ちることになる。

本研究では、C言語を対象として並列プロセッサによる並列化コンパイルによって処理時間の短縮化を試みた。本稿では並列化コンパイラの設計における方法論と実際に得られた短縮化の効果について述べる。

2．　並列プロセッサの構成と機能

本研究に使用した並列プロセッサの構成を図１に示す。これは当研究室で設計・製作されたもので、マスタ・スレイブ方式に結合されている。各プロセッサは32kbyte のTwo Port Ram(以下TPRと略す)を持つ。マスタ側CPU(以下マスタと略す)・スレイブ側CPU(以下スレイブと略す）間のメッセージ交換は、割り込み信号により行なわれる。更にマスタはブロードキャスト(Broadcast) 機能を有し全てのTPR の同一アドレス領域に同時に書き込むことが可能である。

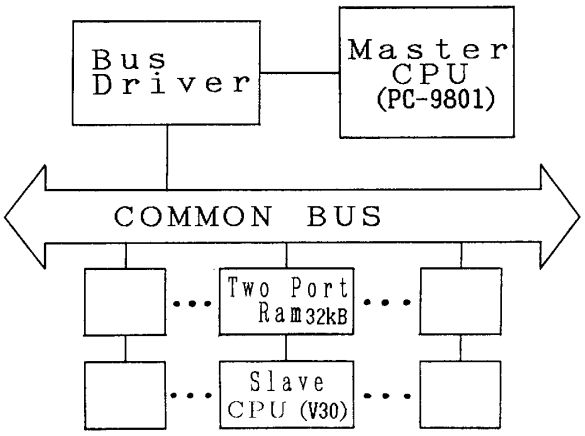

図１　並列プロセッサのシステム構成

Parallel C Compiler by Multi Processor
Minoru Koike , Yoshikuni Okawa
Osaka University

3．　並列化コンパイラ

3．1　並列化コンパイラの設計方針

並列化については次の方針にしたがって設計した。

(1) 各スレイブが担当するソースプログラムの大きさは可能な限り均等にする。

(2) ソースプログラムを分割するときは、可能ならブロックが閉じた所、最悪でも文の終端子で切れるようにする。これは文単位で分割すれば構文解析・コード生成において不都合が生じにくいからである。

3．2　並列化コンパイラの構成

今回作成したコンパイラの構成はおよそ次の通りである。下記の７個の作業の内③、⑤は並列化に伴う付加的な作業である。また②は従来単語分けの一部分を成す作業である。

① ソースファイルのロード
② 単語分けの準備
③ 文法的情報の伝達
④ 単語分け
⑤ 変数表の転送及び調整
⑥ 構文解析
⑦ コード生成
⑧ オブジェクトファイルのセーブ

3．3　単語分けの準備

ここでは④（単語分け）以後で必要とする各スレイブ上の文法的情報を調査する。この段階ではプログラム上の文の区切りは不明であるのでバイト数が均等になるように各スレイブに担当させる。調査する情報は次の３種類である。

(1) 担当範囲の先頭・末尾での文の状態（実行文、非実行文（引用文、注釈文））

(2) 担当範囲の先頭・末尾でのブロックの深さ

(3) 担当範囲内の末尾に一番近い終端子の位置

(2)は制御文に対して生成されるジャンプラベルに利用される情報である。(3)は各スレイブが担当する範囲を決定するための情報である。

3．4　文法的情報の伝達

ここでは②（単語分けの準備）によって各スレイブで得られた情報をマスタが他の各スレイブに伝達する。なお伝達時間を短縮させるためにブロードキャスト機能を利用している。

３．５　変数表の転送及び調整

ここでは④（単語分け）で得られた各スレイブ上の変数情報を他のスレイブに転送する。これを行なう理由は次の通りである。④はスレイブ毎に独立して処理される。そのため、ソース・プログラムの分割によってあるスレイブ(Ｉ)上で変数宣言された変数が、別のスレイブ(Ｊ)上で使用される場合が起こる。このときコンパイラはこの変数をスレイブ(Ｊ)上の新たな未宣言変数として扱っている。これをこのまま各々のスレイブ上に残して置くと、⑦(コード生成)の変数割り付けの段階で同じ変数に対して、二つもしくはそれ以上の領域を確保することになり、誤ったコードを生成することになる。

これを避けるために⑥（構文解析）・⑦に先立って次に述べる作業を行なう。まず各スレイブ上で作成された変数情報を、他の全部（大域的変数の場合）もしくは一部（局所的変数の場合）のスレイブへ転送する。次に各スレイブ上において転送された変数情報を発生した未宣言変数と照らし合わせる。一致したときは変数番号・属性等の転送を行なう。

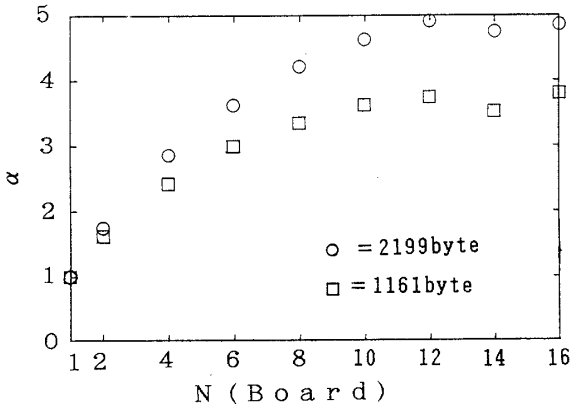

図２　並列プロセッサ台数と加速指数との関係

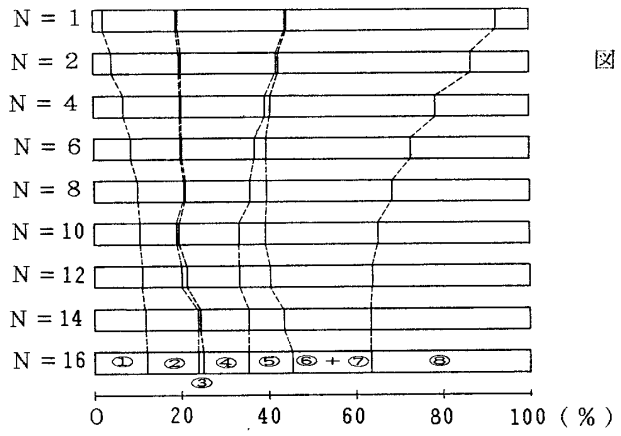

図３　各作業が全作業に占める処理時間の割合
（2199byteの場合）

４．　結果及び考察

$$\text{加速指数} = \frac{\text{単一プロセッサによる処理時間}}{\text{並列プロセッサによる処理時間}}$$

上式の定義にもとづく加速指数αを縦軸に、分割数（スレイブ台数）Ｎを横軸に取った結果を図２に示す。また、図３に各作業の処理時間が全体の処理時間に占める割合を各スレイブ毎に示す。図２によるとＮ≧１２では加速指数はほぼ横這いである。この理由として図２、図３より次の２つが考えられる。

第１の理由は次の通りである。まず分割数が大きくなると各スレイブが担当するソース・プログラムのバイト数が小さくなる。すなわちスレイブの占有時間が小さくなる。逆にマスタの占有時間は、分割数の増加に伴う通信時間の増大により大きくなる。したがって、マスタ・スレイブの両占有時間の合計である全処理時間の変化が鈍くなり、結果として加速指数の上昇も鈍くなる。通信時間の増加については、図４より明らかである。

第２の理由は以下の通りである。ロード・セーブ時間は分割数の大小に関わらずほぼ一定になっている。その上図３によると、ロード・セーブ時間はスレイブ台数(分割数)Ｎが大きくなるに従って大きな割合を占めていく。つまり分割数以外のパラメータが一定のとき、分割数が増加してスレイブ占有時間が小さくなると、全処理時間は次第にロード・セーブ時間という一定値に漸近していく。加速指数の定義式における分母の変化が小さくなるために、結果として加速指数の上昇が鈍くなる。

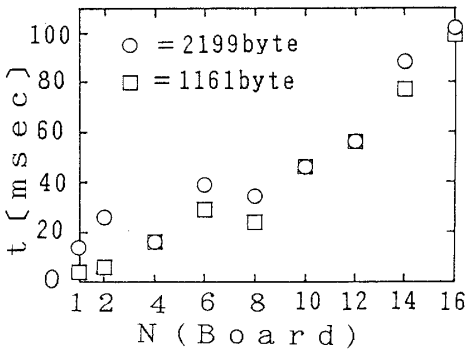

図４　分割数(並列プロセッサ台数)と通信時間との関係

５．　結言

本研究では、元来逐次的であるコンパイル作業を並列プロセッサを用いて並列化を行なった。並列化に伴う付加的な作業の全体の処理時間に対する影響は少なく、その結果、ソース・プログラムの大きさが約２キロバイトのとき最高４．９１倍のスピードアップが得られた。

〔参考文献〕

(1) 中田；コンパイラの技法，竹内書店新社
(2) 黄　鎧，Faye A.Briggs；Computer Architecture and Parallel Processing，McGrawHill