2004－HPC－98　(7)
2004／4／13

粗粒度並列化コンパイラ CoCo の開発

池田倫久[†]　Ngo Tau Van[†]　田中雅俊[††]
福岡岳穂[†††]　片桐孝洋[†]
本多弘樹[†]　弓場敏嗣[†]

本稿では，COINS と呼ばれる並列化コンパイラ開発のための共通インフラストラクチャを用いて，逐次プログラムを粗粒度並列化し，OpenMP プログラムを自動的に生成するコンパイラ CoCo の開発について述べる．SPEC95 の swim および tomcatv を用いて CoCo の性能評価を行ったところ，4 プロセッサ並列実行時において swim については約 1.9 倍，tomcatv については約 1.7 倍の速度向上が得られた．さらに，一般的に困難とされているコンパイラ開発者のデバッグ情報の理解を助けるために，逐次プログラムを構成しているマクロタスク等を視覚化できるツールの開発も合わせて行った．

Development of coarse grain parallelizing compiler“ CoCo ”

Norihisa Ikeda ,[†] Ngo Tau Van ,[†] Masatoshi Tanaka ,[††]
Takeaki Fukuoka ,[†††] Takahiro Katagiri ,[†] Hiroki Honda [†]
and Toshitugu Yuba [†]

This paper describes the development of coarse grain parallelizing compiler“ CoCo ”, which automatically parallelizes a sequential program into an OpenMP program by using a infrastructure for developing parallelizing compiler called COINS. By performance evaluation using four processors in parallel, CoCo achieves 1.9 times speed-up in SPEC95 swim, and 1.7 times in SPEC95 tomcatv.Moreover, we developed a tool which visualizes the macrotasks which compose a sequential program, for helping a developer of a coarse grain parallelizing compiler understand the debug information.

1. はじめに

逐次プログラムを並列実行する際，プログラムを複数の粗粒度タスク（以下マクロタスク）に分割して並列実行させる粗粒度並列処理（以下マクロデータフロー処理）が広く研究されている．SMP 型並列計算機上でのマクロデータフロー処理の研究は従来から盛んに行われており，[1)2)] では Fortran プログラムに対する方式が提案された．これらはいずれもコンパイル時にマクロタスク間の並列性を抽出し，実行時にスケジューラによって，マクロタスクを順次プロセッサに割り当てる．同方式を利用して，[3)4)] では OpenMP を用いたマクロタスク並列実行方式が提案された．本論文では，OpenMP を用いるマクロタスク並列実行方式を利用し，COINS（A Compiler Infra Structure）[5)] を用いた粗粒度並列化コンパイラ CoCo（Coins based Coarse grain parallelizing Compiler）の開発および性能評価について述べる．

粗粒度並列化コンパイラのデバッグ情報の理解が開発者にとって大きな負担である．それを軽減するために CoCo のデバッグ情報を GUI を用いて視覚的に表現する支援ツールの開発も合わせて行った．

以下 2 章でマクロデータフロー処理の概要を述べ，3 章では CoCo について，4 章では支援ツールについて述べる．5 章では CoCo 及び支援ツールの評価結果を示し，6 章で今後の課題について述べ，7 章でまとめる．

2. マクロデータフロー処理

マクロデータフロー処理では，プログラムをマクロタスクの集合としてとらえ，各マクロタスクをプロ

† 電気通信大学大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications
†† 電気通信大学 電気通信学部 情報工学科
Department of Computer Science, The University of Electro-Communications
††† 株式会社 管理工学研究所
Kanrikogaku Kenkyusho, Ltd.

セッサへの割当ての単位として並列に処理する．マクロタスクとは，プログラム中の一部分であり，その部分の実行を開始する文がその部分の先頭の行だけであるもの[1] と定義される．例として，基本ブロックやサブルーチン，最外殻ナチュラルループなどがマクロタスクとなりうる．

コンパイル時（逐次プログラムから並列プログラムへの変換時）には，プログラムをマクロタスクに分割し，マクロタスク間の制御フローおよびデータ依存をマクロフローグラフと呼ばれる非循環の有向グラフとして表現する．また，マクロフローグラフにもとづいて，マクロタスク間の並列性を実行開始条件[1] として検出する．

並列プログラム実行時には，実行開始条件が成立したマクロタスクを順次プロセッサに割り当てることで処理を進める．

3. 粗粒度並列化コンパイラ CoCo

3.1 COINS

COINS[5] とは「並列化コンパイラ向け共通インフラストラクチャの総合研究」として平成 12 年度より進められている研究プロジェクトである．COINS は，コンパイラで利用される基本的な解析や最適化などの機能をモジュール化して提供することを目的としている．COINS で用いられている高水準中間表現は HIR(High-level Intermediate Representation) と呼ぶ．HIR はソースプログラムの情報を保持した木構造表現である．

3.2 CoCo

CoCo は COINS を用いて構築された粗粒度並列化コンパイラである．C 言語の逐次プログラムを入力とし，SMP 型並列計算機上で粗粒度並列処理を行う並列プログラムを自動生成する．CoCo の設計方針は下記の通りである．

(1) COINS を並列化コンパイラ開発基盤として利用する．
(2) 並列プログラムの可搬性を考慮し，OpenMP による並列化を行う．
(3) HIR 水準でマクロデータフロー解析を行う．
(4) OpenMP 指示文をコメントとして含む HIR プログラムを C プログラムに変換する．
(5) C プログラムへの変換過程で動的スケジューリングルーチンを付加する．
(6) SMP 型並列計算機を対象とする OpenMP コンパイラにより並列実行コードを生成する．

図 1 に CoCo の処理フローを示す．

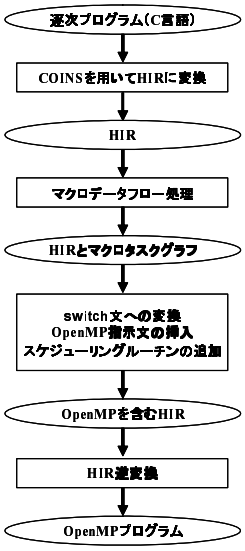

図 1 CoCo の処理フロー

3.2.1 switch 文への構造変換

CoCo では HIR を用いて与えられた逐次プログラムの構造を解析し，マクロタスクを作成し並列性を抽出する．その後，プログラムの構造を組み替えて C 言語表現に逆変換する．プログラムの構造は，入力された逐次のものから，1 つのマクロタスクを 1 つの case ブロックで制御するように C 言語の switch 文に組み換えられる．

以下では，図 2 に示すサンプルプログラムを用いて，switch 文への変換について述べる．

```
int main() {
  int i, a1[100], a2[100];

  for (i = 0; i < 100; i++) {
    a1[i] = ⋯;
  }

  for (i = 0; i < 100; i++) {
    a2[i] = ⋯;
  }

  printf("a1=%d,a2=%d",a1[1],a2[2]);
  return(0);
}
```

図 2 サンプルプログラム

図 2 にはループ文のマクロタスク 2 つと，サブプログラム文のマクロタスク 1 つが存在する．図 3 は，サンプルプログラムをマクロタスクに分割したものである．図 3 をマクロタスクフローグラフ化したものを図 4 に示す．図中の点線はデータフローを，実線は制

```
int main() {
  int i, a1[100], a2[100];

  /** MT1 **/
  for (i = 0; i < 100; i++) {
    a1[i] = ⋯ ;
  }

  /** MT2 **/
  for (i = 0; i < 100; i++) {
    a2[i] = ⋯ ;
  }

  /** MT3 **/
  printf("a1=%d,a2=%d",a1[1],a2[2]);
  return(0);
}
```

図 3 マクロタスク分割したサンプルプログラム

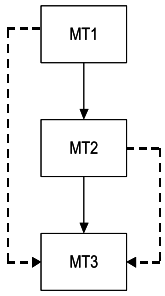

図 4 例題プログラムのマクロフローグラフ

御フローを示す．マクロタスク 3 は，マクロタスク 1 と 2 にデータ依存している．またマクロタスク 1 と 2 はプログラム実行開始時に既に実行可能であることがわかる．これらの情報を元にプログラムの構造変換を行う．プログラムの構造は 1 つのマクロタスクを 1 つの case ブロックとして switch 文に変換される．また，動的スケジューリングを行うためには，プログラム実行時にスケジューラに対して実行開始条件の変更を伝えなければならない．そのため以下のコードをプログラムに挿入する．

(1) 変数宣言部分を除いたプログラム全体を覆うループ文
(2) スケジューラに対する関数呼び出し
(3) スケジューラに通知する実行開始条件の変更を示すコード
(4) マクロデータフロー処理の終了を示すコード

図 3 のプログラムを switch 文に構造変換したプログラムを図 5 に示す．

3.2.2 OpenMP 指示文の挿入

プログラムを並列実行するために，OpenMP 指示文をプログラムに挿入する．使用する指示文は下記の

```
int main() {
  int i, a1[100], a2[100];
  int taskNum = -1:  // MT 番号

  // スケジューラの初期化
  initScheduler();

  while(1) {
    if (出口 MT の処理が終了) break;

    // スケジューラへのアクセス
    taskNum = getTask();

    switch(taskNum) {
      /** MT1 **/
      case 1:
        for (i = 0; i < 100; i++) {
          a1[i] = ⋯ ;
        }
        (実行開始条件の更新)
        break;

      /** MT2 **/
      case 2:
        for (i = 0; i < 100; i++) {
          a2[i] = ⋯ ;
        }
        (実行開始条件の更新)
        break;

      /** MT3 **/
      case 3:
        printf("a1=%d,a2=%d",a1[1],a2[2]);
    }
  }
  return(0);
}
```

図 5 switch 文に変換したサンプルプログラム

通りである．

- pragma omp parallel
  プログラム全体を覆うループの外側に挿入することにより，マクロタスクの並列処理を実現する．
- OpenMP の lock 関数
  スケジューラへのアクセス部分の前後に挿入する事により，スケジューラが利用するキューデータへの排他アクセスを行う．

また，#ifdef を OpenMP ライブラリ関数の前後に挿入することにより，OpenMP コンパイラ以外のコンパイラ上での動作を可能にしている．図 5 のプログラムを粗粒度並列化したプログラムを図 6 に示す．

4. 支援ツール

粗粒度並列化コンパイラの開発では，デバッグ情報の理解が開発者にとって大きな負担となっている．そこで CoCo のデバッグ情報を GUI を用いて表示する

```
int main() {
  int i, a1[100], a2[100];
  int taskNum = -1:

  // スケジューラの初期化
  initScheduler();

#pragma omp parallel private(i)
firstprivate(task)

  while(1) {
    if (出口 MT の処理が終了) break;

#ifdef_OPENMP
    // 並列実行時のみロックを獲得
    omp_set_lock(&lock);
#endif

    // スケジューラへのアクセス
    taskNum = getTask();

#ifdef_OPENMP
    // 並列実行時のみロックを開放
    omp_unset_lock(&lock);
#endif

    switch(taskNum) {
      /** MT1 **/
      case 1:
        for (i = 0; i < 100; i++) {
          a1[i] = ··· ;
        }
        (実行開始条件の更新)
        break;

      /** MT2 **/
      case 2:
        for (i = 0; i < 100; i++) {
          a2[i] = ··· ;
        }
        (実行開始条件の更新)
        break;

      /** MT3 **/
      case 3:
        printf("a1=%d,a2=%d",a1[1],a2[2]);
    }
  }
  return(0);
}
```

図 6 粗粒度並列化したサンプルプログラム

以下のツールを作成した．

(1) マクロタスクとソースプログラムの対応関係を表示する機能
ユーザが選択したマクロタスクに対応するソースプログラム中の部分を強調表示する．

(2) マクロフローグラフ視覚化機能
マクロタスク間の制御フロー情報・データ依存情報を解析し，マクロフローグラフを自動的に生成する．マクロフローグラフの表示には AT & T 研究所で開発された graphviz[7] を用いた．

(3) 実行開始条件表示機能
ユーザが選択したマクロタスクの実行開始条件及び各条件（実行確定条件，非実行確定条件，データアクセス可能条件）を表示する．

図 7 に支援ツールを実行した場合のウィンドウ例を示す．各ウインドウの詳細は以下の通りである．

(A) マクロフローグラフ表示ウインドウ
マクロフローグラフを表示する．

(B) プログラムソース表示ウインドウ
プログラムソースを表示する．また，マクロフローグラフ表示ウインドウでマクロタスクを選択すると，対応するプログラムソース部分が強調表示される．

(C) 実行開始条件表示ウインドウ
選択されたマクロタスクの実行開始条件，実行確定条件，非実行確定条件，データアクセス可能条件を表示する．

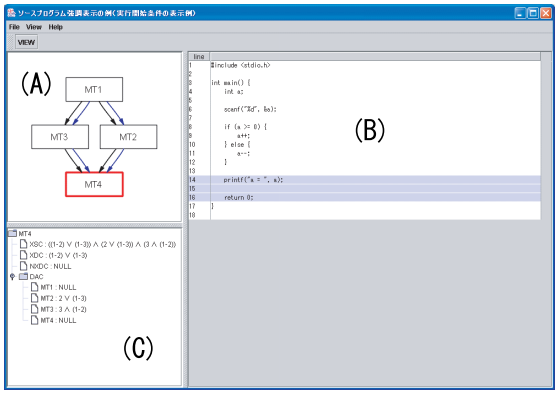

図 7 支援ツールのウィンドウ例

5. 評　　価

5.1 CoCo の評価

5.1.1 評 価 環 境

表 1 に示す環境において，SPEC95 ベンチマークプログラムの swim および tomcatv を用いて CoCo の性能評価を行った．

5.1.2 swim による評価

swim は図 8（左）のようなマクロフローグラフをしており，マクロタスク間がほぼ逐次処理になっているため，そのままでは並列効果が得られない．また，swim の実行時間の 97 %はマクロタスク 8 の実行で

表 1 評価環境

OS	Solaris 8
プロセッサ	Sun UltraSPARC2(450MHz) x 4
メモリ	1GB
Java コンパイラ	JDK 1.4.2
C コンパイラ	gcc 2.95.3
OpenMP コンパイラ	Omni OpenMP Compiler 1.4

占められている．マクロタスク 8 はループブロックであり，ループの 1 イタレーションはさらに複数のループブロックから構成される．そこでマクロタスク 8 を図 8（右）のように 4 スレッドに手動で分割したうえで CoCo でコンパイルを行い，その実行時間を測定した．結果を表 2 に示す．

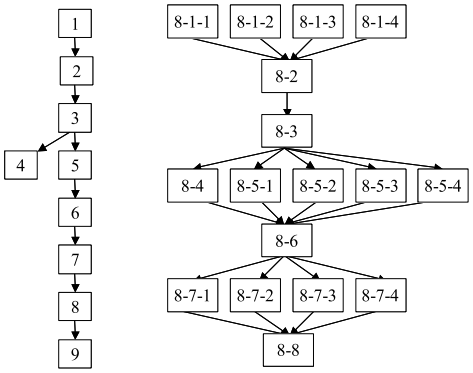

図 8 （左）swim のマクロフローグラフ
（右）マクロタスク 8 を分割した場合のマクロフローグラフ

表 2 swim の測定結果

	プロセッサ数	実行時間 [sec]	台数効果
変換前	1	2.52	1.0
変換後	1	2.53	1.0
	2	1.61	1.5
	3	1.60	1.5
	4	1.28	1.9

表 2 に，2 並列時には逐次実行の約 1.5 倍，4 並列時には約 1.9 倍の速度向上が得られたことを示す．

5.1.3 tomcatv による評価

tomcatv は図 9（左）のようなマクロタスクフローグラフをしており，swim と同様にそのままでは並列性を抽出できない．実行時間が他のマクロタスクに比べて非常に大きいマクロタスク 5 を，図 9（右）のように 4 スレッドに手動で分割後，CoCo でコンパイルを行い実行時間を測定した．図 9 の点線はデータフロー，実線が制御フローを表す．測定結果を表 3 に示す．

表 3 に，2 並列時には逐次実行の約 1.4 倍，4 並列

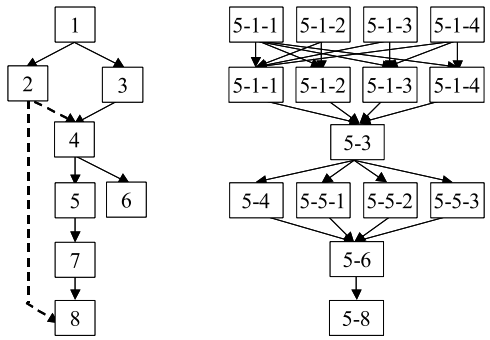

図 9 （左）tomcatv のマクロフローグラフ
（右）マクロタスク 5 を分割した場合のマクロフローグラフ

表 3 tomcatv の測定結果

	プロセッサ数	実行時間 [sec]	台数効果
変換前	1	0.67	1.0
変換後	1	0.67	1.0
	2	0.47	1.4
	3	0.45	1.5
	4	0.39	1.7

時には約 1.7 倍の速度向上が得られたことを示す．

5.2 支援ツールの評価

本支援ツールを用いて，以下に述べる 3 つの作業を効率よくできたことが確認できた．

(1) マクロタスクとソースプログラムの対応関係の把握
(2) マクロタスク間の依存関係を視覚的に把握
(3) 任意のマクロタスクの実行開始条件及び各条件の取得

6. 今後の課題

現在の CoCo の制約を以下に示す．

(1) 粗粒度並列化モジュールはプログラムの main 関数のみを並列化する．
(2) 並列プログラムを効率よく実行するために「ループアンローリング」等の粒度調整手法を取り入れるべきであるが，現在の CoCo ではそれを行えない．
(3) 現在の CoCo では，HIR の予約語を用いたループは 1 つのマクロタスクに変換できるものの，それ以外のループ構造は 1 つのマクロタスクに変換できない．
(4) CoCo では，後続タスクがない，または return 文を含むマクロタスクのみ出口と判断される．例えば，もしあるマクロタスクが exit() 関数等を含んでいても，そのマクロタスクは出口と判断されない．

(5) 依存関係がないマクロタスクが複数あった場合，逐次実行の場合とはマクロタスクの実行順序が異なることがある．

(1)，(2) の制約により CoCo では，プログラム中に実行時間の長いループやサブルーチンを含む場合は並列効果を得ることができない．今回用いた tomcatv 等はループ部を手動で展開し，CoCo で速度向上が見込めるプログラム構造に変換した．

今後の課題としては，本稿は手動で行った main ループ部分以外のプログラム構造のマクロタスク分割を自動で行う必要があることが挙げられる．そのために階層型マクロタスクグラフのための異階層タスクの統合実行制御手法[6] を用いることを検討している．同手法を用いて CoCo のマクロタスク生成クラスに階層マクロタスクの概念を取り込み，階層統合型動的スケジューリングを行うことで，問題が解決できると考える．

7. ま と め

本論文では逐次プログラムを自動的に OpenMP プログラムに変換する粗粒度並列化コンパイラ，及び，デバッグ情報の視覚化を目的とした支援ツールについて述べた．CoCo を用いることにより，逐次プログラムを自動的に OpenMP プログラムに変換することができた．また，SPEC95 tomcatv にようなマクロタスクの並列性が表れていないようなプログラムに関しては，サブルーチンの展開，ループの分割後に CoCo でコンパイルを行うことにより，並列性のある OpenMP に変換することができた．

粗粒度並列化コンパイラのデバッグ情報については，支援ツールを用いることにより，理解が困難とされているテキストベースのデバッグ情報を視覚的に確認でき，作業の効率化を図ることができた．

謝　　辞

本研究は，科学技術振興調査費「並列化コンパイラ向け共通インフラストラクチャの研究」による．

参 考 文 献

1) 本多弘樹, 岩田雅彦, 笠原博徳,“ Fortran プログラム粗粒度タスク間の並列性検出手法, ” 電子情報通信学会 D-I, Vol.J73-D-I, No.12, pp.951-960, Dec. 1990.
2) 本多弘樹, 合田憲人, 岡本雅巳, 笠原博徳,“ Fortran プログラム粗粒度タスクの OSCAR における並列実行方式, ” 電子情報通信学会, D-I, Vol.J75-D-I, No.8, pp.526-535, Aug. 1992.
3) 福岡岳穂, 本多弘樹, 弓場敏嗣,“ OpenMP による粗粒度タスク並列実行方式, ” 情報処理学会ハイパフォーマンスコンピューティング研究会, Vol.2000, No.82, pp.65-70, Aug. 2000.
4) 石坂一久, 小幡元樹, 笠原博徳,“ OpenMP を用いた粗粒度並列処理, ” 情報処理学会計算機アーキテクチャ研究会, Vol.2000, No.139, pp.187-192, Aug. 2000.
5) COINS project, http://www.coins-project.org
6) 吉田 明正, “ 階層型マクロタスクグラフのための異階層タスクの統合実行制御手法, ” 情報処理学会計算機アーキテクチャ研究会，Vol.2003, No.154, pp.73-78, Aug. 2003.
7) Graphviz - Open source graph drawing software
http://www.research.att.com/sw/tools/graphviz/