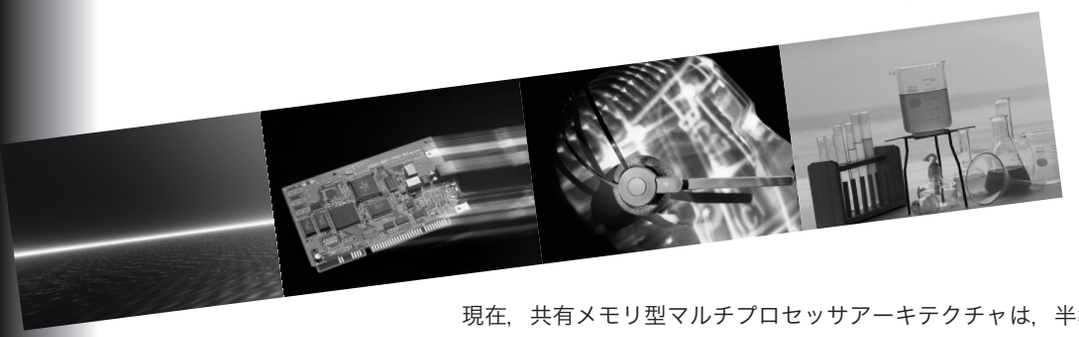
解説

最先端の自動並列化コンパイラ技術

笠原 博徳
早稲田大学／アドバンスト並列化コンパイラ研究体
kasahara@waseda.jp



現在，共有メモリ型マルチプロセッサアーキテクチャは，半導体チップ上に複数のプロセッサを集積したチップマルチプロセッサから，デスクトップワークステーション，エントリレベルサーバ，スーパーコンピュータに至るまで，多くのコンピュータで採用されている．また，今後チップマルチプロセッサは，携帯電話，ゲームなどの SoC（System on Chip）分野で多く採用されることが予測されており，Multiprocessor Everywhere の時代が到来すると考えられる．このようにマルチプロセッサシステムが多く使われるようになると，複数のプロセッサを効率よく動作させることができる並列プログラムを短期間で開発することが重要となる．本稿では，これを実現しマルチプロセッサを簡単かつ効率よく使えるようにする自動並列化コンパイラ技術について紹介する．

自動並列化コンパイラとは

自動並列化コンパイラ [1)〜3)] とは，通常の逐次型プログラムを，対象とする並列コンピュータのハードウェアを効率よく動作させる並列化プログラムに翻訳するプログラムである．具体的には，逐次プログラム中より，並列に動作可能な部分を抽出し，その並列動作可能部分を複数のプロセッサにスケジューリングし，並列化されたプログラムを生成する．

このような並列化コンパイラが必要なのは，マルチプロセッサシステムの使用にあたって，次のような問題を軽減するためである．

(1) プロセッサ台数とともに向上するハードウェアピーク性能に比べ実際にプログラムを実行したときの実効性能が低い．開発側の立場から見ると，プロセッサ数の増加とともに，価格性能比が劣化し市場競争力の確保が難しい．
(2) ユーザは，望ましい性能を得ようとすると，ハードウェアを効果的に動作させる並列プログラムを自ら作成する，あるいは実行結果を見ながら並列化チューニングを繰り返す必要があり，本来費やすべき研究開発ではなく，並列処理のために多くの時間をとられてしまう．

すなわち，自動並列化コンパイラが，ユーザのプログラムを自動的に並列化し高い実効性能を与えることができれば，

- ユーザは簡単に所望の性能を得ることができる．
- 同じ性能を得るのであれば，よりプロセッサ台数の少ない低価格のシステムを購入すればよい．
- 並列化プログラムの作成・チューニングにかけていた時間を本来の研究開発に割り当てることができ並列コンピュータを利用する各種の研究開発を推進することができる．

などのメリットが生じる．

このような自動並列化コンパイラの開発を目指した研究開発は，1970 年代初頭の ILLIAC Ⅳの時代に開始され，30 年近い長期にわたり続けられている．従来の自

動並列化コンパイラでは，プログラムの実行時間の多くはループで消費されるということから，ループの並列化を中心に研究開発が行われていた[2]．このようなループを対象とした並列処理では，1980 年代に我が国のコンパイラが世界をリードした自動ベクトル化コンパイラ[1]が思い出される．この国産ベクトル化コンパイラとベクトルスーパーコンピュータハードウェアの優位性が高かったことから，マルチプロセッサ用の自動並列化コンパイラ[1)～3)]では米国が 1980 年代中旬から製品開発を行っていたのに対し，我が国では 1990 年代中旬以降と開発が大幅に遅れた．

現在，スーパーコンピュータのような超高性能コンピュータでは，地球シミュレータに代表されるベクトルプロセッサを並列接続したベクトル型マルチプロセッサシステム，あるいは**図-1** のPower4 のような汎用プロセッサを並列接続した**図-2**（b）のIBM pSeries690 のようなスカラ型マルチプロセッサ，またスカラ型マルチプロセッサをノードとしそれを多数接続した米国ASCI（Accelerated Strategic Computing Initiative）マシンのように，すべてがマルチプロセッサ化されている．

また，高性能コンピュータだけでなく，**図-2**（a）に示すような 4 プロセッサ程度のデスクトップワークステーション，大学研究室単位で購入可能なエントリレベルサーバなど多くのコンピュータがマルチプロセッサ方式に移行している．さらには，図-1 のPower4 は，1 チップ上に 2 つのプロセッサコアと共有L2 キャッシュを集積したチップマルチプロセッサで，今後の高性能プロセッサの多くはこのようなチップマルチプロセッサになると考えられている．

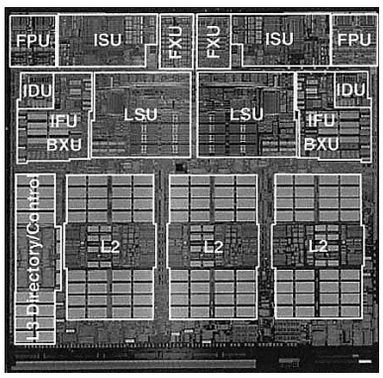
このようなチップマルチプロセッサ方式が採用されると，

- 異なる能力のプロセッサが，要求される性能に合わせ集積するプロセッサ数を変えるだけで，短期間で開発できる．
- クロック周波数もチップ内プロセッサ単位でローカルに上げやすい．
- アプリケーションにより，一時的に利用されないプロセッサには電力を供給しないなど低消費電力化が可能．

という利点もあり，今後の携帯電話，ゲーム，PDA，ディジタルテレビ等多くのSoC（System on Chip）分野でも採用されていくと考えられる．

このためマルチプロセッサ用の自動並列化コンパイラの開発は，SoC 分野からスーパーコンピュータ分野，さらにはそれらの情報システムを利用する多くの科学技術分野の研究開発競争力強化のために有用である．

このような背景の下，マルチプロセッサシステム用自動並列化コンパイラ技術を世界最先端に高めるため，2000 年 9 月より政府ミレニアムプロジェクトIT21 の一環としてアドバンスト並列化コンパイラプロジェクトが 3 年度計画で開始された．このプロジェクトは，経済産業省ミレニアムプロジェクト官民共同研究開発プロジェクトとして，新規産業創出型産業科学技術研究開発制度（産技制度）に基づき，新エネルギー・産業技術総合開発機構（NEDO）からアドバンスト並列化コンパイラ技術研究体が委託を受け行われた．なお，この研究体は（財）日本情報処理開発協会（JIPDEC），早稲田大学，産業技術総合研究所で構成され，参加企業の日立，富士通からの研究者はJIPDEC に出向するかたちで共同体に加わり，再委託の電気通信大学，東京工業大学，東邦大



Tendler, J. M., Dodson, J. S., Fields, J. S. Jr., Le, H. and Sinharoy, B. : POWER4 System Microarchitecture, IBM Journal of Research and Development, Vol46, No. 1 (2002).

図-1　チップマルチプロセッサ IBM Power4

(a)　(b)

(a) 4 プロセッサ・ワークステーション Sun Ultra 80
(b) 32 プロセッサ・ハイエンドサーバ IBM pSeries690

図-2　共有メモリ型マルチプロセッサ・システム

学のメンバとともに研究開発を行っている．

次章では，このアドバンスト並列化コンパイラプロジェクトで研究開発された最先端の自動並列化コンパイラ技術を，従来の基本的な技術も織り交ぜながら解説する．

最先端の自動並列化技術

自動並列化コンパイラは，ユーザが記述した通常の逐次型プログラム中から並列性を抽出するため，最初に，

(1) データの定義・使用関係による実行順序制約を解析するデータ依存解析 [1)～3)]
(2) 条件分岐などの制御の流れを解析し，どの条件分岐がどの命令集合の実行を確定するかという制御依存解析 [1)]
(3) 依存によりそのままでは並列化できないプログラム部分を，同じ計算を行う並列性のある表現へと変換するプログラムリストラクチャリング [1)～3)]

を行う．次に，並列実行可能な部分のプロセッサへのスケジューリングすなわち割当ておよび実行順序の決定，プロセッサ間データ転送の最小化あるいはキャッシュ・分散共有メモリの有効利用を目指したデータ分割配置（データ分散）を行う．最後にそのプログラムのスケジューリング結果およびデータ分散結果を実現する並列プログラムの生成を行う．生成されたプログラムは，OSの管理の下，ターゲットマシン上で実行される．

通常のコンパイラでは，生成される並列プログラムは機械語で記述されるが，本アドバンスト並列化コンパイラ（以下APC）プロジェクトでは生成される並列プログラムがどの共有メモリ型マルチプロセッサシステム上でも動作するように，共有メモリ型マルチプロセッサシステム用の標準並列化言語拡張OpenMP（http://www.openmp.org）で記述されたプログラムを生成する．OpenMPは，Fortran，C，C++等に対する拡張で，並列実行可能部分の指定および同期等をコンパイラに対する指示文の形（この指示文は，対応コンパイラ以外ではコメント文として認識される）で記述する．すなわち，APCプロジェクトで開発するコンパイラは参加企業開発のマルチプロセッサ以外でも処理を高速化できることを目指し，**図-3** のように逐次FORTRANプログラムから，OpenMP並列化指示文を持つFORTRANプロ

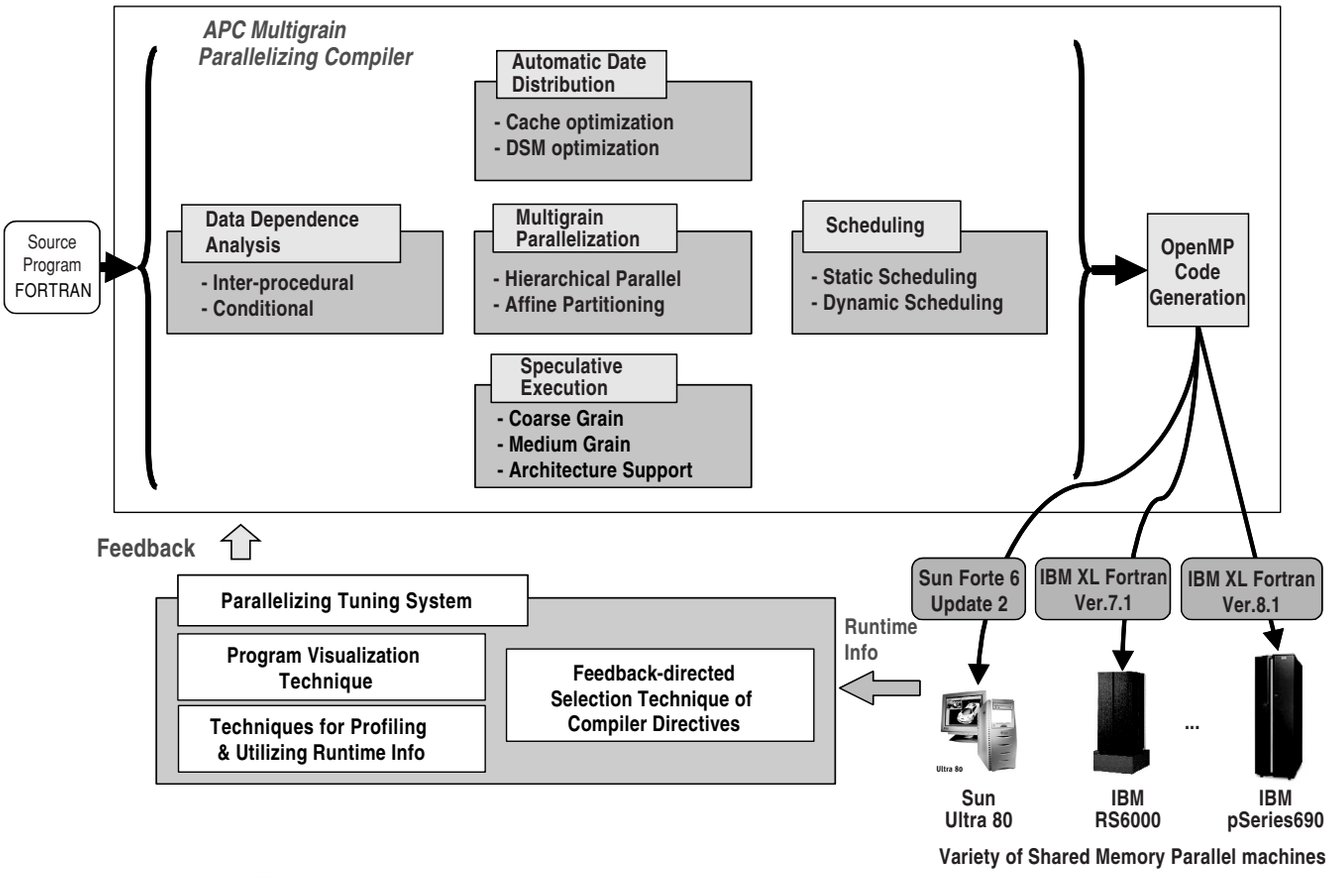


図-3　APC コンパイラの構成

グラムを生成し，ターゲットマルチプロセッサ用のコンパイラで機械語プログラムに翻訳し実行するという方式をとっている．また，一部のエキスパートユーザ向けに，実行プロファイルを利用したプログラムチューニングツールも用意している．

このAPCコンパイラは，現在のループ並列化の限界を超えるため，早稲田大学で提案されたマルチグレイン並列化[1), 6), 7)]の実現を目指している．

ループ並列化

ここで，ループ並列化とは，現在市販されているマルチプロセッサシステム用のコンパイラが採用している並列処理方式で，ループのイタレーション（繰り返し）間のデータ依存を解析して各イタレーションを独立に処理してよいと判断できる場合には，各イタレーションを別々のプロセッサに割り当てて並列処理する方式である[1)～5)]．このループ並列化を効果的に行うために，現在各社から販売されている商用並列化コンパイラ，および米国イリノイ大学で開発されているPolaris[4)]，Parafrase2，あるいはスタンフォード大学で開発されているSUIF[5)]のような研究コンパイラは，

(a) ステートメントの実行順序の変更
(b) ループディストリビューション（単一ループを少数のステートメントを含む複数のループへ分割し，ベクトル化したり，ループ並列性を高める方式）
(c) ノードスプリッティング／スカラエクスパンション（データのコピーあるいはスカラ変数の配列変数への拡張を行うことにより逆依存，出力依存と呼ばれる見た目上のデータ依存を削除し，並列化を可能とする方式）
(d) ループインターチェンジ（ネストされたループの外側と内側を入れ替え，より大きな並列性を引き出したり，キャッシュアクセスを最適化する方式）
(e) ループアンローリング（ループの複数イタレーションを展開し，より大きな並列性を抽出したり，ループ制御あるいはキャッシュプリフェッチオーバーヘッドを削減する方式）
(f) ループフュージョン（複数の別々なループを1つのループへ融合し，ループ制御オーバーヘッドを軽減したり，データローカリティを高める方式）
(g) ストリップマイニング（1重ループをネストされた多重ループへと変換し，キャッシュ等限られた容量のメモリの有効利用，階層的なループ並列化を可能とする方式）
(h) アレイプライベタゼーション（ループ内テンポラリー配列による見た目上のデータ依存を削除するために，配列データをプライベート化し並列化可能とする方式）
(i) ユニモジュラー変換（ループリバーサル，パーミュテーション，スキューイング等を線形変換で統一的に扱う方式）

など各種のプログラムリストラクチャリングを行い並列性を高めようとしている．しかし，このループ並列化は長期にわたるさまざまな研究のためすでに成熟期に達しており，今後大幅な性能向上は難しいといわれている．そこで，マルチグレイン並列化ではこのループ並列化に加え，より大きい並列化単位であるサブルーティンおよびループ間の粗粒度並列性，あるいは基本ブロックと呼ばれる算術代入を行うステートメントが連続して並んでいるプログラム部分内のステートメント間近細粒度並列性を階層的に引き出し，プログラム全域の並列性を使用しようとする方式である．

マルチグレイン並列化

単一プログラム中のサブルーティン，ループ，基本ブロック間の並列性を利用する粗粒度並列処理は，マクロデータフロー処理とも呼ばれ[1), 6), 7)]，APCコンパイラのコアの1つとなる早稲田大学OSCARマルチグレインFortran並列化コンパイラで初めて実現された[1)]．

OSCARコンパイラでは，粗粒度タスク（マクロタスク）として，単一のFORTRANプログラムよりRB（Repetition Block），SB（Subroutine Block），BPA（Block of Pseudo Assignment Statements）の3種類のマクロタスクを生成する．RBは各階層での最外側ナチュラルループであり，SBはサブルーティン，BPAはスケジューリングオーバーヘッドあるいは並列性を考慮し融合あるいは分割された基本ブロックである．

次にマクロタスク間の制御フロー[1)]とデータ依存を解析し，**図-4**（a）のようなマクロフローグラフ（MFG）を生成する．MFGでは，各ノードがマクロタスク（MT），点線のエッジが制御フロー，実線のエッジがデータ依存，ノード内の小円が条件分岐文を表している．また，MT7のループ（RB）は，内部で階層的にMTおよびMFGを定義できることを示している．

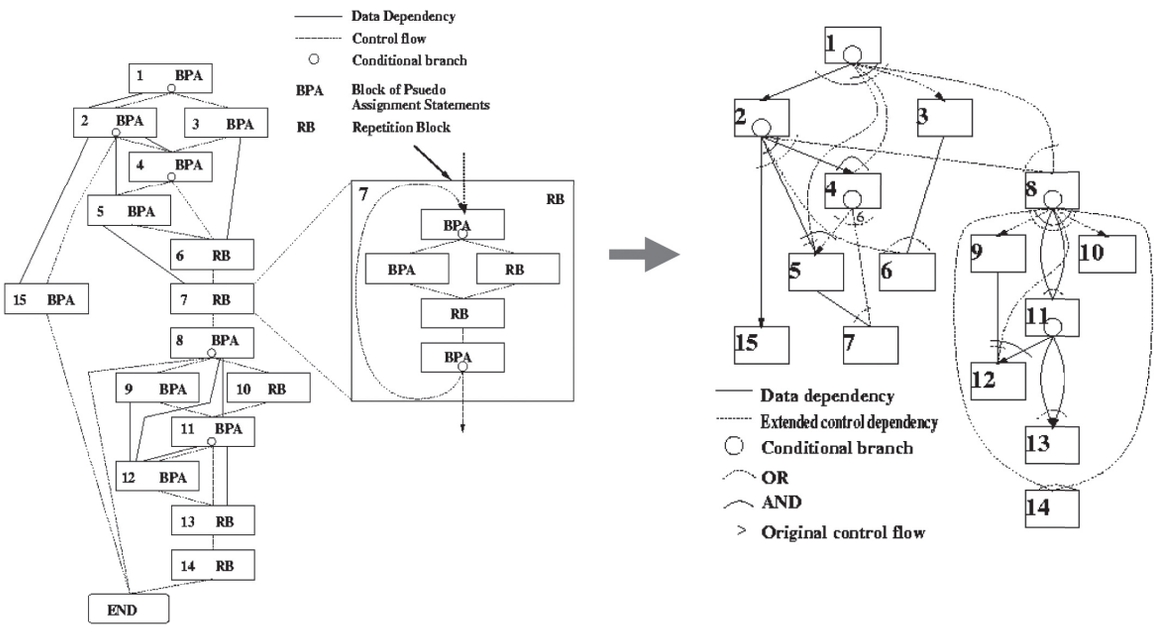
次に，マクロタスク間制御依存[1), 6)]およびデータ依存を統合的に解析し各マクロタスクが最も早く実行できる条件（最早実行可能条件）[1)]すなわちマクロタスク間の並列性を検出する．この並列性をグラフ表現したのが**図-4**（b）に示すマクロタスクグラフ（MTG）である．MTGでも，ノードはMT，実線のエッジがデータ依存，ノード内の小円が条件分岐文を表す．ただし点線のエッジは拡張された制御依存を表し，矢印のついたエッジは

元のMFGにおける分岐先，実線の円弧はAND関係，点線の円弧はOR関係を表している．たとえば，MT6 へのエッジは，MT2 中の条件分岐がMT4 の方向に分岐するか，MT3 の実行が終了したとき，MT6 の実行が可能になることを示している．

次にコンパイラは，MTG上のMTをプロセッサグループへスタティックに割当てを行う（スタティックスケジューリング）か，実行時に割当てを行うためのダイナミックスケジューリングプログラムをDynamic CPアルゴリズムを用いて生成し，これを並列化プログラム中に埋め込む．これは，従来のマルチプロセッサのようにOSあるいはランタイムライブラリに粗粒度タスクの生成，スケジューリングを依頼すると数千から数万クロックという大きなオーバーヘッドが生じてしまう可能性があり，これを避けるためである．このような並列化プログラムをOpenMPを用いて生成するために，APCコンパイラでは**図-5** に示すワンタイムシングルレベルスレッド生成法[6),7)]を開発した．図-5 は，サブルーティン，逐次ループ等ネスト構造を持つプログラムを8 スレッドで階層的に並列処理する場合に，ワンタイムシングルレベルスレッド生成法を用いて生成される並列化プログラムのイメージを表している．この例では，第 1 階層のマクロタスクグラフがデータ依存エッジのみを持っているため，第 1 階層の 4 つのマクロタスクをコンパイラがスタティックスケジューリングを適用し，各 4 スレッドからなる 2 つのスレッドグループに割り当てている．また，サブルーティンであるマクロタスクMT1_3 および並列化できないループであるマクロタスクMT1_4 内では，階層的にマクロタスクグラフMTG1_3，MTG1_4 を生成し，各グラフの並列度を推定後，その並列度に合ったスレッドグループを生成し階層的に並列処理している．この例の場合，これらのマクロタスクグラフは条件分岐を含んでいるため，ダイナミックスケジューリングが適用される．図中では，マクロタスクグラフMTG1_3 に対しては，1 つのスレッドにスケジューリングを任せる集中型ダイナミックスケジューリングを適用する場合が示されており，マクロタスクグラフMTG1_4 に対しては 4 スレッドを 2 スレッドずつの 2 グループに分け，各スレッドグループがスケジューリングを行う分散型ダイナミックスケジューリングの場合が示されている．

APC コンパイラは，この例のように多重にネストされたプログラムに対しても，各階層のマクロタスクグラフの並列度をループ並列性も含め推定し，各階層ごとにその並列度に見合ったスレッドグループ数を自動的に設定し，フレキシブルな階層的並列処理を可能とする[7)]．

さらに，このような粗粒度並列性を抽出する際には，サブルーティン間にわたるデータ依存を正確に解析することが重要となる．APC プロジェクトではこのインタープロシージャ解析の高度化に関する研究開発も行った．APC コンパイラは，**図-6** に示すようにサブルーティンにわたる定数の伝搬を行い，必要に応じクローンと呼ぶサブルーティンのコピーを作成し，その結果不要となる条件分岐文等を削除しプロシージャ間のデータ依存



(a) A Macro Flow Graph　　(b) A Macro Task Graph

図-4　マクロタスクフローグラフからの最早実行可能条件解析によるマクロタスクグラフの生成

を正しく解析できるようにしている[7]．これにより粗粒度タスク間の並列性をより多く引き出せるとともに，ループボディ中にサブルーティンコールを含むループの並列化もより高精度に行うことができるようになった[7]．

またAPCでは，不完全ネストループより並列性を引き出すとともに，データローカリティを高め，さらにイタレーション間同期オーバーヘッドを減らす手法としてスタンフォード大SUIFグループにより提案されたがコンパイラには実装できなかったAffine Partitioning[8]を実現する実用的な手法を開発し，**図-7**に示すようなパイプライン化された並列コードをOpenMPを用いて生成することに成功した[7]．

メモリ利用最適化技術

実際のマルチプロセッサシステム上で効果的な並列処理を実現するためには，並列性の抽出のみならず，メモリアクセスオーバーヘッドをいかに抑えるかが重要な課題となる．これはプロセッサの動作速度向上に対しメモ

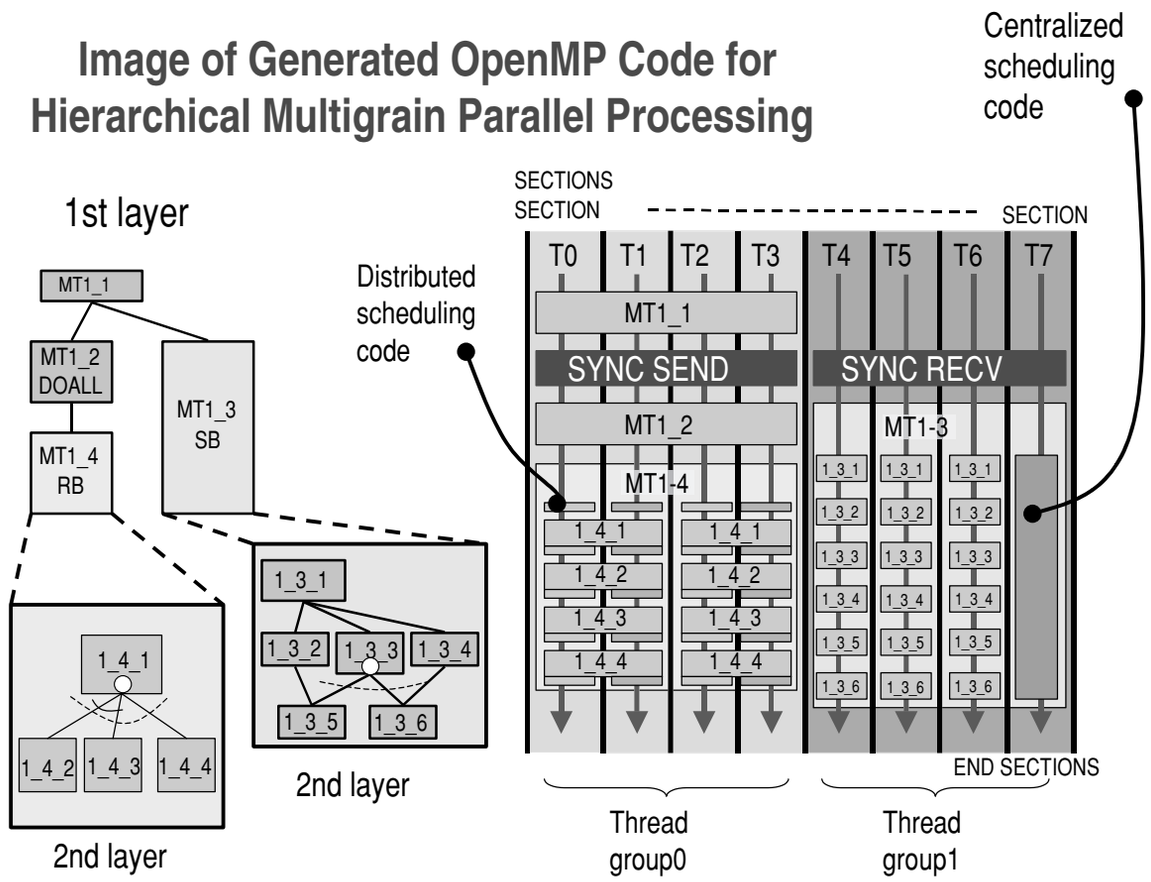


図 -5　OpenMP を用いた階層的マルチグレイン並列化のための並列コード生成方式

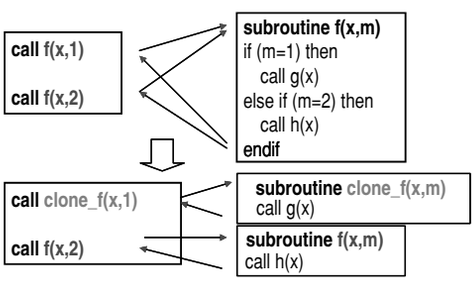


図 6　定数伝搬・クローニングを用いたインタープロシージャ解析

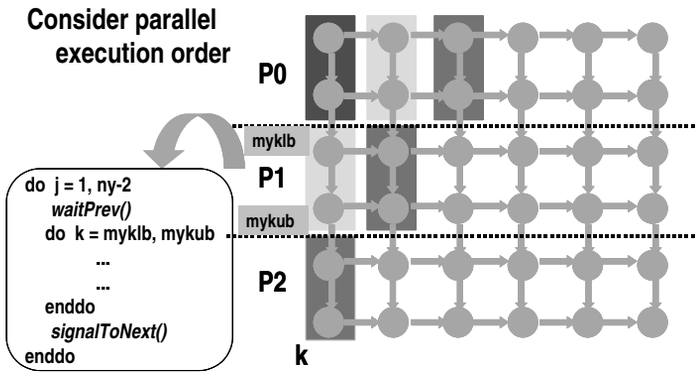


図 7　Affine Partitioning を用いたパイプライン並列化

リのアクセス速度向上が追いついていないために生じるもので，主記憶共有型マルチプロセッサではキャッシュの有効利用が，また分散共有メモリ型マルチプロセッサでは近接分散共有メモリの有効利用が性能に大きく影響する．

このためAPCコンパイラでは，主記憶共有型マルチプロセッサ用の分散キャッシュメモリの有効利用を目指したデータローカライゼーション技術と，分散共有メモリ型マルチプロセッサのためのファーストタッチ制御方式を開発した．データローカライゼーションは，マクロタスクグラフ上でデータ依存が存在する複数のループにわたりイタレーション間のデータ依存を解析し（インターループデータ依存解析），プロセッサ間データ転送が最小になり，さらにデータがキャッシュに収まるようにループとデータを分割（ループ整合分割）する[6]．さらに一度キャッシュ上に載せたデータを複数のループにわたり使用できるように，前のループが終了してから次のループを実行するという通常の実行順序ではなく，前のループの一部（整合分割により生成された小ループ）を実行すると同一のデータにアクセスする他のループの一部，またそれにデータ依存する他のループの一部を実行するという方式で，キャッシュ上のデータを長期間有効利用できるようにしている．さらに，プログラム中の配列サイズがキャッシュサイズの整数倍あるいは整数分の1の場合，配列間でラインコンフリクトが頻発することがある．これを避けるため，本データローカライゼーション手法では配列サイズを拡張する配列間パディングを行い，ローカライゼーション後のラインコンフリクトを最小化する方式も開発している[7]．

また，分散共有メモリ型マルチプロセッサ用の自動データ分散では，SGI社Origin2000のOSが，最初に当該データを触ったプロセッサ上のメモリにデータを割り付けるファーストタッチ方式を採用していることから，これを利用しコンパイラが望むデータ分散を実現する方式を開発した．これは，プログラム中で最も処理時間を消費するループ（メインループ）で要求されるデータ分散に合わせ，コンパイラがその分散を実現するダミーのループ（ファーストタッチ制御ループ）を生成し，メインループ内での遠隔分散共有メモリへのアクセスを最小化する．この方式を用いると，配列へのアクセスパターンが実行時まで分からないため人手でもデータ分散が難しかった配列の間接参照（配列添え字中に他の配列が現れるようなメモリ参照）を伴うプログラムに対しても最適なデータ分散を実現できる．これにより，SGI社コンパイラに対し，32プロセッサOrigin2000上でNAS Parallel Benchmarks CGプログラムを6.6倍高速に実行できることを確かめている．

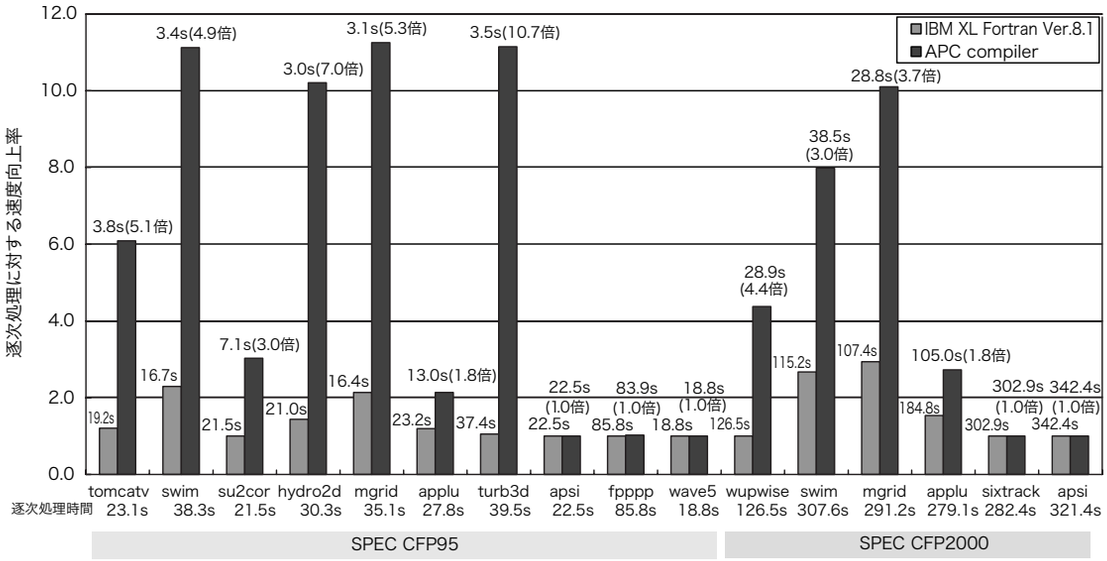
APCコンパイラの性能

ここでは，前章までで述べたコンパイラ技術を統合したAPCコンパイラを市販の主記憶共有メモリ型マルチプロセッサシステム上で性能評価した結果について述べる．

APCコンパイラは図-3に示したように，逐次型FORTRANプログラムを入力するとOpenMP指示文を用いて並列化されたFORTRANプログラムを生成する．今回評価に使用したマシンは，Power4チップを搭載したIBM pSeries690 16プロセッサ・ハイエンドサーバ，IBM RS6000 604e high-node 8プロセッサ・ミッドレンジサーバ，Sun Ultra 80 4プロセッサデスクトップワークステーションである．これらのマシンを使用するにあたって，使用するコンパイラは，pSeries690では最新のIBMコンパイラXL Fortran Ver.8.1，RS6000ではAPCプロジェクト開始時のコンパイラXL Fortran Ver.7.1，Ultra 80ではSun Forte 6 update 2を用いた．評価においては，SPEC CFP95およびSPEC CFP2000のうちの全FORTRAN77プログラム16本に対しAPCコンパイラを適用し，生成されたOpenMPコードを上述の各社コンパイラを用いてコンパイルして実行した．本アドバンスト並列化コンパイラプロジェクトにおいてはプロジェクトの基本計画としてプロジェクト開始当時のループ並列化コンパイラの性能を概ね2倍上回るという数値目標を与えられており，この性能評価を客観的に行うために性能評価を担当するグループがプロジェクト内に設置されている．コンパイラ開発プロジェクトで数値目標を持った研究開発は世界でも例がなく，2倍をどのように評価するかが問題になるが，本プロジェクトにおいては使用可能なプロセッサ数までのプロセッサを用いた際の商用コンパイラの最高性能とAPCコンパイラの最高性能を比較する方式をとることとした．これはループ並列性のみを使っている商用コンパイラに比べ，APCコンパイラはより大きな並列性およびメモリアクセスオーバーヘッドの軽減が可能であるためプロセッサ数とともに性能が向上する場合が多いのに対して，商用コンパイラでは，プロセッサを増やすと逆に性能が低下してしまう場合があり，たとえば同じ16プロセッサ同士の性能を比較するとAPCコンパイラの性能が非常によく見えてしまうためである．

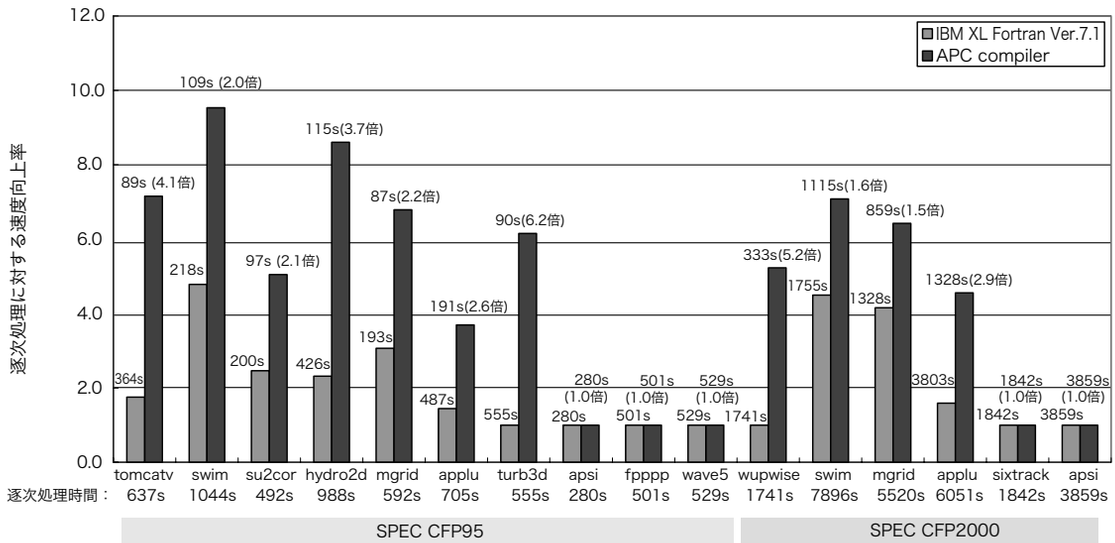
このように各マシンが持つ最大プロセッサ数までの最高性能を，逐次処理に対するスピードアップ率として表示したのが，**図-8**から**図-10**である．

図-8は，16プロセッサpSeries690上での性能であり，各バー2本の組のうち，左側がXL Fortran Ver.8.1



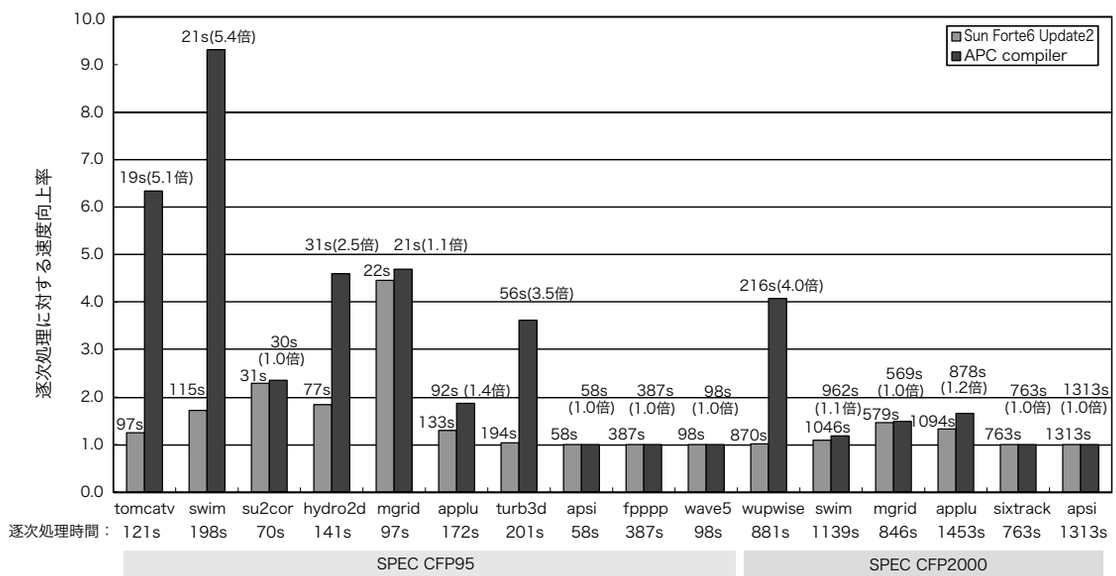
*各バーの数字はIBM社製最新コンパイラXL Fortran Ver.8.1と，開発したAPCコンパイラによる各プログラムの処理時間[s:秒]を示す．
APCコンパイラによる処理時間横のカッコ内は，IBMコンパイラに比べAPCコンパイラが何倍処理速度を向上できるかを示している．

図 -8　IBM ハイエンドサーバ pSeries690 16 プロセッサ上での APC コンパイラの性能



*各バーの数字はIBM社製最新コンパイラXL Fortran Ver.7.1と，開発したAPCコンパイラによる各プログラムの処理時間[s:秒]を示す．
APCコンパイラによる処理時間横のカッコ内は，IBMコンパイラに比べAPCコンパイラが何倍処理速度を向上できるかを示している．

図 -9　IBM RS6000 604e high node 8 プロセッサ上での APC コンパイラの性能



*各バーの数字はSun社製最新コンパイラ Forte 6 Update 2と，開発したAPCコンパイラによる各プログラムの処理時間[s:秒]を示す．
APCコンパイラによる処理時間横のカッコ内は，IBMコンパイラに比べAPCコンパイラが何倍処理速度を向上できるかを示している．

図 -10　4 プロセッサ Sun Ultra 80 ワークステーション上での APC コンパイラの性能

ループ並列化コンパイラを用いたときのスピードアップ率であり，右側がAPCコンパイラを用いたときのスピードアップ率である．図中プログラム名の下に書かれている数値は逐次処理時間[s]を表しており，各バー上の数字は並列処理時の処理時間[s]を，またカッコ内の数字はAPCコンパイラによるXL Fortranコンパイラに対する速度向上率を示している．この図を見ると分かるように，APCコンパイラはSPEC CFP95 turb3dプログラムに対し，pSeries690上で10.7倍の高速化を達成できることが分かる．また16プログラムの平均速度向上率を計算すると算術平均で3.5倍という数値が得られており，最新のマシンおよび最新の市販コンパイラに対して，プロジェクトの目標を大幅に上回る性能を得ることができた．

また，**図-9**は8プロセッサIBM RS6000上での性能を表している．プロセッサ数が少ないこともあり，pSeries690と比べ性能向上率は小さいが，最大で6.2倍，平均で2.4倍の性能を得ることができた．これらのIBMマシンで最大性能が得られたturb3dでは，クローニングを伴うインタープロシージャ解析，ワンタイムシングルレベルスレッド生成を用いたマルチグレイン並列化が有効に働き上記のような性能が得られている．また，tomcatv，swim等のプログラムではデータローカライゼーションを用いたキャッシュ最適化が有効に働き，良好な速度向上が得られた．

図-10は上記2マシンと比べ価格の低い4プロセッサ主記憶共有型ワークステーション上での性能を表している．このマシンは，上記2マシンに比べメモリ性能が低いため，キャッシュ最適化が性能に大きく影響する．このマシン上では，レベル2キャッシュがダイレクトマップ方式であるためパディングによるラインコンフリクト除去を併用したキャッシュ最適化がきわめて有効で，SPEC CFP95 swimでは4プロセッサで逐次処理に比べ9.3倍の速度向上，Forteコンパイラに比べ5.4倍の速度向上を得ることができた．また全16プログラムに対する速度向上の平均でも2.0倍という数値を得られている．

以上のように，APCコンパイラは複数の機種の主記憶共有メモリ型マルチプロセッサに対してきわめて高い性能を示すことが確かめられた．

自動並列化コンパイラの現状および今後

アドバンスト並列化コンパイラプロジェクトは2003年3月に実質2年半の研究開発を終了する．このような短い研究開発期間を選んだのは，競争の激しいこの分野でマルチグレイン並列化という新しい技術の優位性を示すには，参加メンバが所有しているコンパイラ技術をフル活用した短期間での開発実証が必須と考えたためである．

プロジェクトを通じ，従来例を見ない数値目標は大きなプレッシャーであったが，最新の16プロセッササーバ上で，そのマシン用の最新コンパイラと比べ，平均3.5倍の速度向上を達成できたことは，開発者としても予想しなかった好成績である．これは，同一のハードウェア上で，APCコンパイラを一度通すだけで自動的に性能向上を得られることを意味しており，この技術をさらに磨けばマルチプロセッサを使用する各種アプリケーションユーザの方の計算速度向上およびプログラム開発期間の短縮が可能になると考えている．

また，このコンパイラは高性能コンピュータの利用者のみならず，今後のチップマルチプロセッサの性能向上，価格性能比向上，アプリケーション開発期間の短縮にも役立つものと期待できる．特に，ゲーム，携帯電話，PDAなどのSoC分野ではハードウェア・ソフトウェアの開発期間短縮，価格性能比の向上，アプリケーションソフトウェアの質が市場を分ける重要な要素となるため，自動並列化コンパイラが今後この分野の競争力強化のために役立つものと考えている．

謝辞　本稿執筆にあたり，種々ご協力いただきました経済産業省，新エネルギー・産業技術総合開発機構（NEDO），IT21評価助言委員会，アドバンスト並列化コンパイラプロジェクトの皆様に感謝いたします．

参考文献

1) 笠原博徳：並列処理技術，コロナ社 (1991).
2) Banerjee, U.: Loop Parallelization, Kluwer Academic Pub. (1994).
3) Wolfe, M.: High Performance Compilers for Parallel Computing, Addison Wesley (1996).
4) Eigenmann, R., Hoeflinger, J. and Padua, D.: On the Automatic Parallelization of the Perfect Benchmarks, IEEE Trans. on Parallel and Distributed Systems, Vol.9, No.1 (1998).
5) Hall, M. W., Anderson, J. M., Amarasinghe, S. P., Murphy, B. R., Liaos.-W., Bugnion, E. and Lam, M. S.: Maximizing Multiprocessor Performance with the SUIF Compiler, IEEE Computer (1996).
6) Kasahara, H., Obata, M., Ishizaka, K., Kimura, K., Kaminaga, H., Nakano, H., Nagasawa, K., Murai, A., Itagaki, H. and Shirako, J.: Multigrain Automatic Parallelization in Japanese Millenium Project IT21 Advanced Parallelizing Compiler, Proc. of IEEE PARELEC, Warsaw, Poland (Sep. 23, 2002).
7) APC2003: アドバンスト並列化コンパイラ国際シンポジウム資料，（財）日本情報処理開発協会(JIPDEC)，早稲田大学理工学部 (Mar. 20, 2003).
8) Lim, A. W. and Lam., M. S.: Cache Optimizations with Affine Partitioning, Proceedings of the Tenth SIAM Conference on Parallel Processing for Scientific Computing (2001).

（平成15年3月5日受付）