システムソフトウェアと
オペレーティング・システム 89－3
システム評価 2－3
（２００２．２．１４）

SPEC CFP2000 ベンチマークの縮小プログラム開発手法とその評価

稲田 由江† 河場 基行† 安里 彰†

サイズの大きいベンチマークプログラムの性能評価を論理シミュレータ等を用いて現実的な時間で行うことを可能にするため、元プログラムの特性を保持したままでプログラムを縮小する手法を提案する。本手法は、高頻度処理部分の抽出、繰り返し回数の削減、データ量削減の3段階からなる。この手法を用いて CFP2000 プログラム中の 172.mgrid と 183.equake の縮小を試みたところ、命令コンビネーションやキャッシュアクセスパターンを維持したままで、オリジナルプログラムのステップ数を数千分の一から数万分の一にまで縮小することができた。

Program Shrinking Technique and its Evaluation with SPEC CFP2000

Yoshie Inada†, Motoyuki Kawaba†, Akira Asato†

To enable the performance evaluation of large benchmarks using logic simulators with reasonable time period, we propose a methodology to develop smaller programs having almost same characteristics with their originals. Our approach consists of three steps, that is, the extraction of frequently executed parts, the reduction of repetitions and the reduction of data size. We applied this approach to 172.mgrid and 183.equake in CFP2000 and succeeded in shrinking these programs. Though these programs have only about 1/10,000 times as many cycles as original programs, they keep their characteristics such as instruction combinations and memory access patterns similar to original programs.

1. はじめに

一般に、CPU の開発においては、論理シミュレータを用いて設計データの検証や性能評価が行われる。性能評価を行うベンチマークプログラムとして CPU2000 ベンチマークプログラム群[1]が用いられることが多い。しかし、このプログラム群は命令数、データ量共に莫大であり、論理シミュレータで全てのベンチマークを実行し性能評価することは現実的ではない。そこで、我々は現実的な時間で性能評価を行うために CPU2000 プログラム群の特性を保持した小さなプログラムを開発した。

ベンチマークの特性を保持しつつ命令数を縮小する研究として、サンプリング抽出や類似する命令列の抽出などの研究が行われている[2][3][4][5][6]。我々の研究は、これを論理シミュレータ上での性能評価に応用したものと位置付けられる。

本論文では、CPU2000 プログラム群のうち浮動小数点演算系プログラム群 CFP2000 に焦点を絞り、縮小プログラムの作成方法について述べ、これを評価した結果を述べる。

本論文の構成は以下の通りである。まず2章ではプログラム縮小の基本方針について述べる。次に3章4章で CFP2000 プログラム群の中の 172.mgrid と 183.equake の縮小方法について述べ、最後に 5 章でまとめを行う。

2. 基本方針

2.1. CFP2000 プログラムの特徴

SPEC CPU2000 ベンチマークは、CPU 性能測定ベンチマークプログラム群である。整数演算系 12 本のプログラム群 CINT2000 と、浮動小数点演算系 14 本のプログラム群 CFP2000 に分類される。今回対象としたＣＦＰ 2000 プログラムは、科学技術計算プログラムであり以下の特徴がある。

① 支配的な処理が限定的
② 同様な処理の繰り返し

†(株) 富士通研究所
FUJITSU LABORATORIES LTD.

③　静的なデータ構造

従って、縮小プログラムの作成方法は上記の特徴に対応し次の手順で実現できると考えられる。

①　高頻度処理部分抽出
②　繰り返し回数削減
③　データ量削減

一方、CINT2000ﾌﾟﾛｸﾞﾗﾑは高頻度処理部がプログラム中に広範囲に点在し、またデータ構造も動的に変化するため、同様な方法での縮小は難しい[7]。

2.2. 縮小方針

縮小方針として、まず実機を用いて全プログラムの挙動を概略的に調査する。プロファイル情報を元に高頻度処理部分を抽出し、繰り返し回数を削減する。実機のＣＰＵは、実行命令数やサイクル数等の内部カウンタを持っており、この値を用いて CPI やキャッシュミス率の評価を行う。実機として Fujitsu PRIME-POWER GP7000F(CPU:SPARC64 GP,2 次キャッシュ:2MB direct mapped)を使用する。

次に、データ量削減については、キャッシュアクセスに影響するため、ターゲットとなる CPU の構造を再現できるキャッシュシミュレータを用いてキャッシュアクセスの詳細な分析を行う。ここでは評価対象のキャッシュ構成を、2 次キャッシュ:2MB,4way とした。

2.2.1. 高頻度処理部分抽出

プログラムコードは一様に実行されるのではなく、処理に偏りがあることが多い。よって、プロファイル情報を採取することにより高頻度処理部分を把握し、抽出箇所を絞り込む。

2.2.2. 繰り返し回数削減

主要処理部分が繰り返し処理で、かつ、繰り返し回数によって処理の挙動が変化しない場合、すなわち、1 ループの挙動が全ループの挙動と等しいと仮定できる場合は、1ループのみを抽出することが可能である。よって、ループの各繰り返し間で、以下の要件を満たせばループ回数を1回に削減可能とする。

- 繰り返し回数と命令数が比例する
- 繰り返し回数によってキャッシュミス率は変化せず一定である(1 ループのキャッシュミス率=全ループのキャッシュミス率)

2.2.3. データ量削減

データ量の削減はプログラム中で参照されるデータ領域を削減することになり、命令数の削減につながる。ここで、考慮すべき点が2点ある。

一点は、「縮小変数の選択」である。データの削減には配列データのサイズ縮小が考えられるが、データのアクセスパターンによって縮小可能な変数と不可能な変数がある。

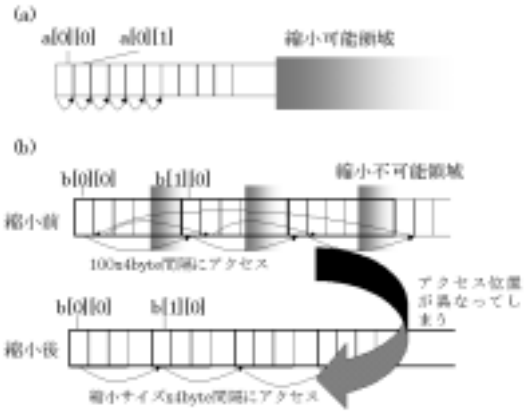


図 1. 縮小可能な変数と不可能な変数のデータアクセス
(a)縮小可能, (b)縮小不可能

- 縮小可能な変数

データサイズを縮小することによりアクセスストライドの幅が変化しないもの。

例えば、double a[100][100]の配列データのアクセスが1次元目から順にアクセスされるものは、処理の後の方でアクセスされる 2 次元目を縮小可能である。これは、参照領域を削減してもキャッシュラインへのアクセスが変わらないことによる（図 1a）。

- 縮小不可能な変数

データサイズを縮小することによりアクセスストライドの幅が変化するもの。

例えば、double b[100][100]の配列データのアクセスが2次元目からストライドアクセスされるものは、処理の後の方でアクセスされる1次元目を削減できない。これは、変数アクセスのストライド幅がオリジナルのアクセスと異なってしまい、アクセスするキャッシュラインの位置も異なってしまうことから、オリジナルプログラムのキャッシュミス率を再現できないためである（図 1b）。

考慮すべきもう一点は、「データ縮小によるキャッシュミス」である。オリジナルのプログラムでは、参照される

データサイズが十分に大きく、キャッシュからのデータの追い出しが頻繁に発生する。しかし、データを削減し参照領域を縮小すると、データの追い出しが減少するため、キャッシュミスが再現しにくくなる。よって、キャッシュミス率が変化しないよう配慮しなければならない。

2.3. 縮小プログラムの評価方法

オリジナルのプログラムを代行する縮小プログラムを用いてCPUの性能評価をするには、オリジナルのプログラムと縮小プログラムの CPU 処理が等価でなければならない。ここでは以下の2点を満たせばプログラムが等価であると定義した。

- キャッシュアクセスパターンの類似

2次キャッシュミスは、メインメモリへのアクセスを生じさせるためレイテンシが大きく、CPU の性能に大きく関与する。そのためキャッシュミス率やミスパターンは、オリジナルプログラムと類似していなければならない。

- 命令コンビネーションの類似

命令コンビネーションはプログラムの特徴を示す。プログラム処理における算術演算、データ転送、制御等の命令コンビネーションの割合は、オリジナルプログラムと同じでなければならない。

2.4. 縮小ターゲット

提案した手順で 172.mgrid と 183.equake のプログラムの縮小プログラムを作成する。オリジナルのソースコードは CPU2000v1.10 を使用し、コンパイラは SUN Forte6 を用いた。

縮小命令サイズの目標は、数万分の一の速度で動作するシミュレータで10分程度で検証しうる命令数である 100 M程度とする。可能であればこれ以上の縮小も試みる。

3. 172.mgrid の縮小

プログラム内容を以下の表に示す。

処理内容	3次元ポテンシャルフィールドの計算プログラム [8]
言語	Fortran 言語
ﾌﾟﾛｸﾞﾗﾑｻｲｽﾞ	500 行
実行命令数	約 464G
データサイズ	58MB
主要変数	Realx8 U(2530944)
	Realx8 V(2197000)
	Realx8 R(2530944)

3.1. 高頻度処理部分抽出による縮小

172.mgrid は大きな2重ループ（外側のループ回数:25 回、内側のループ回数:40 回）から構成される(図2)。プロファイル情報から、高頻度処理部分は内側ループ中の関数 mg3p が 99.3%を占める。外側のループはデータの入力と初期化であるため内側のループのみを抽出を行った。また、抽出部分の処理をオリジナルプログラムと等しくするため、抽出コード実行の直前のデータを保持しする必要がある。172.mgrid は Fortran 言語であるため、Data 文を用いて直前のデータを展開した (図 2)。

高頻度処理部分を抽出したプログラムを GP7000F で評価した(表 1)。オリジナルプログラムと高頻度処理部分抽出によるプログラムとはCPI、2次キャッシュミス率共にほぼ同じであり、内側のループを抽出したことは妥当であることがいえる。

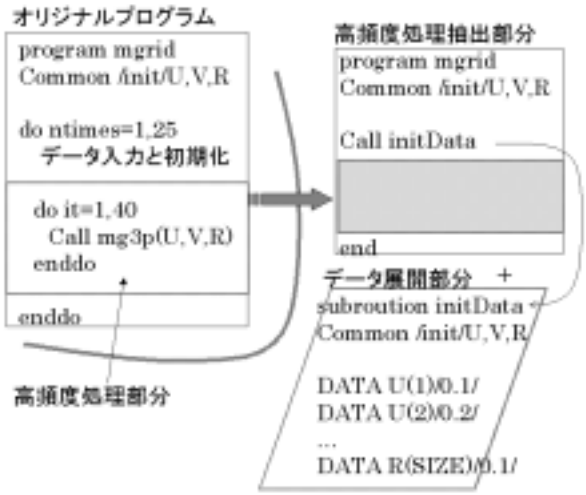


図 2.172.mgrid の高頻度処理部分プログラム抽出

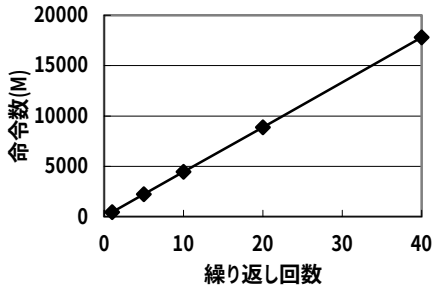
表 1. オリジナルプログラムと高頻度処理部分抽出の比較

	実行命令数(M)	CPI	2次ｷｬｯｼｭﾐｽ率(%)
オリジナル	464604	1.038	0.8696
高頻度抽出	17820	1.042	0.90

3.2. 繰り返し回数削減による縮小

ループ回数による命令数やキャッシュミス率の変化を GP7000F で評価した（図 3a, 3b）。その結果、172.mgrid は 2.2.2 で述べた条件を満たし、1ループの抽出が可能であることがわかった。これにより、命令数を高頻度処理部分抽出時の 1/ループ回数(40)となる 445M 命令に縮小できた。

(a)



(b)

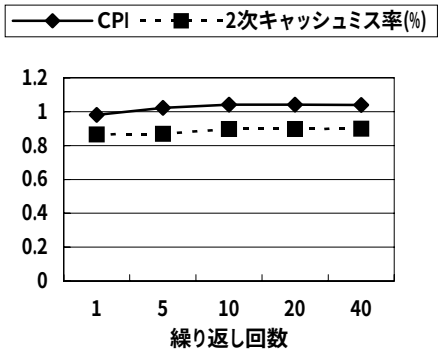


図 3. 172.mgrid の繰り返し回数と命令数(a)と
キャッシュミス率(b)

3.3. データ量削減による縮小

一次元配列の主要変数 U,V,R はプログラム内部で3次元に変更され使用される。この配列データはすべて低次元から順にアクセスされるため縮小可能な変数である。

これら変数の処理領域の一部を抜粋したしたものが図 4 である。mgrid プログラムは図中①から順に処理空間を変えながら3次元立方格子の処理を行う。従って、各次元のインデックスを等しく削減した。この際、オリジナルのﾌﾟﾛｸﾞﾗﾑの 2 次キャッシュミス率をキャッシュシミュレータで調査し、縮小サイズを検討した。結果、図中①②③④は 2 次キャッシュミス率が 0.1%であるのに対し、⑤⑥は、0.5%であった。これは、⑤⑥の処理はそれぞれ 2.3MB、17.5MB と 2 次キャッシュのサイズよりも大きいデータ領域を参照するため、キャッシュからのデータの追い出しが頻繁に発生することが原因であった。従って、縮小の際にも⑤⑥のデータサイズをキャッシュ容量よりも大きいサイズにする必要がある。⑤の処理は命令数が少なく全体への影響が少ないと考え、⑥に着目し各次元のインデックスを 2/3 に縮小し⑥の処理領域を5Mとした。

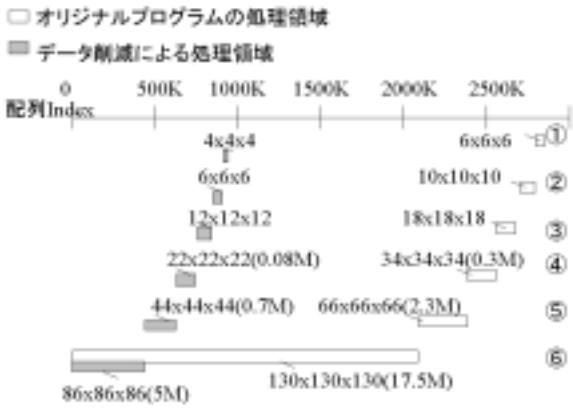
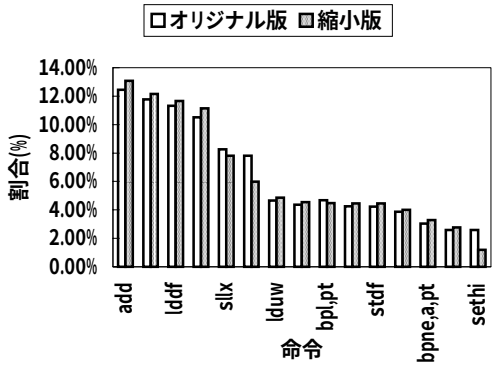


図 4.オリジナルプログラムと縮小プログラムの処理領域

3.4. 172.mgrid の評価

高頻度処理部分の抽出、繰り返し回数の削減、データ量の削減を行い、命令数を約 1/3000 の 138M 命令とする縮小プログラムを開発し評価した。命令コンビネーションについては、各命令はオリジナルプログラムと割合がほぼ同じであり、プログラムの特徴を保持することができた(図 5a)。また、キャッシュミス率についても、オリジナルプログラムのキャッシュミス率をほぼ再現することができた(図 5b)。以上2点から、オリジナルプログラムと縮小プログラムは等価であるといえる。

(a)



(b)

	実行命令数(M)	データサイズ(MB)	2 次ｷｬｯｼｭﾐｽ率(%)
ｵﾘｼﾞﾅﾙ	464000	58	0.6933
縮小	138	17	0.6597

図 5.172.mgrid の評価
(a)命令コンビネーション,(b)命令数と 2 次キャッシュミス率

4. 183.equake の縮小

プログラム内容を以下の表に示す。

処理内容	盆地地形における地震弾性波の伝搬をシミュレートするプログラム[9]	
言語	C 言語	
ﾌﾟﾛｸﾞﾗﾑｻｲｽﾞ	1500 行	
実行命令数	約 162G	
データサイズ	31MB	
主要変数	double **M	30169x3
	double **C	30169x3
	double **M23	30169x3
	double **C23	30169x3
	double **V23	30169x3
	double **vel	30169x3
	double ***disp	3x30169x3
	double ***K	220546x3x3

4.1. 高頻度処理部分抽出による縮小

183.equake は大きな 1 重ループ (ループ回数:3855)内にループ回数の等しい 6 種類のループ(処理 1～6)を含む(図 6)。プロファイル情報から、外側の大きな1重ループが 94.5%であったためこれを高頻度処理部分として抽出した。また、抽出部分の処理を再現するために、172.mgrid の場合と同様、抽出コード実行の直前のデータを保持し展開した。183.equake は C 言語で記述されており、変数はポインタ宣言であるためこれを再現した(図 6)。

高頻度処理部分を抽出したプログラムを GP7000F で評価した(表 2)。CPI と 2 次キャッシュミス率は同等であった。命令数の比較から高頻度処理部分はほぼプログラム全体であることが言えるので、命令コンビネーションについてもオリジナルプログラムと同等であるといえる。

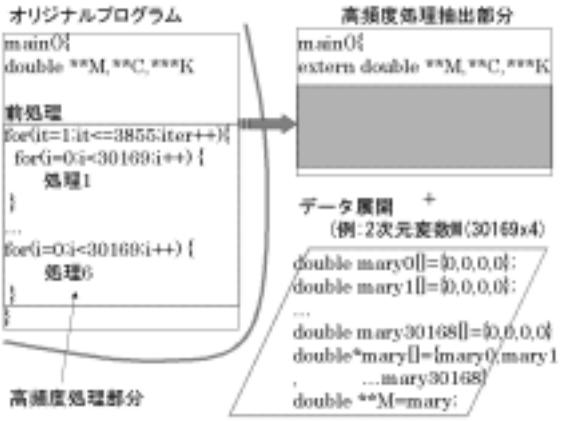


図 6.183.equake の高頻度処理部分プログラム抽出

表 2. オリジナルプログラムと高頻度処理部分の比較

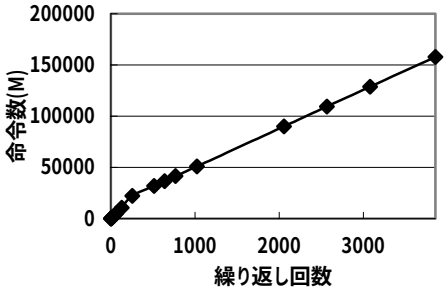
	実行命令数(M)	CPI	2 次ｷｬｯｼｭﾐｽ率(%)
オリジナル	162974	2.523	2.8865
高頻度抽出	157940	2.573	2.8849

4.2. 繰り返し回数削減による縮小

ループ回転数を変化させ、命令数やキャッシュミス率を GP7000F で調査した。その結果、183.equake は

172.mgrid と異なり、繰り返し回数と命令数は比例せず、また繰り返し回数によって、CPIやキャッシュミス率が一定せず処理が異なることが分かった(図 7a,7b)。この原因は、処理 1,2,3,5,6 については繰り返し中処理内容が変化しないが、処理 4 は繰り返しの 250 回目以前とそれ以降とで処理内容が変化するからである。そこで、オリジナルプログラムと命令のバランスやキャッシュミス率を維持するため、前半と後半の処理内容を250:(3855-250)の比率で含むように処理 4 のプログラムコードを処理 4-a,4-b に分けた上で外側1ループを抽出した(図 8)。これにより命令フローが大きく変わってしまうが、縮小プログラムの等価性は保存されるので問題はない。プログラムコードの修正は、コンパイラによってオリジナルプログラムとアセンブラコードが異なってしまう可能性があるために注意しなければならないが、今回は修正の規模が小さく、アセンブラコードへの影響は無かった。

(a)



(b)

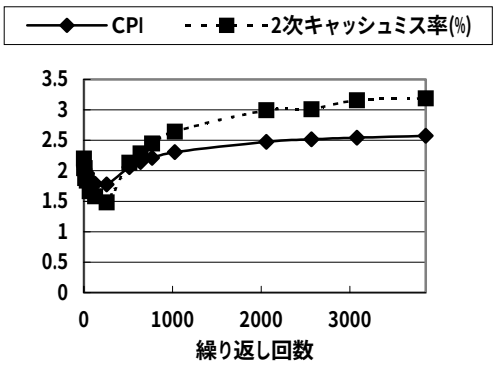


図 7. 183.equake の繰り返し回数と命令数(a), キャッシュミス率(b)

図 8.処理 4 の変更

図 9 は、キャッシュシミュレータにより、それぞれ 1-249 ループの 1 ループの 2 次データキャッシュミス率、250-3855 ループの 1 ループの 2 次データキャッシュミス率、処理内容を混在して抽出したときのミス率を調査したグラフである。処理内容を混在して1ループを抽出したキャッシュミス率は、オリジナルのﾌﾟﾛｸﾞﾗﾑのミス率とほぼ一致した。

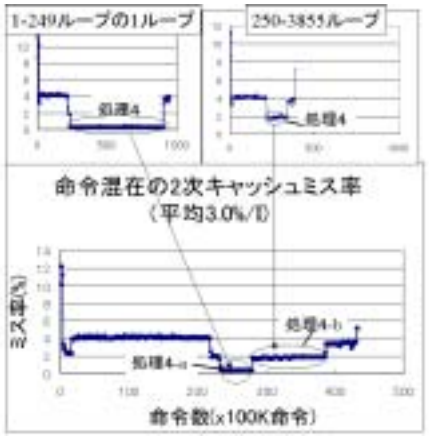


図 9.混在して1ループ抽出したときのキャッシュミス率

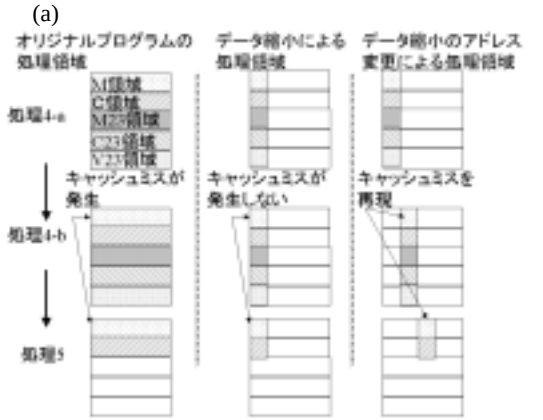
4.3. データ量削減による縮小

データを削減する変数として、サイズが大きく、縮小可能な変数であるKを選択した。

変数Kは処理 2 でのみ参照される。オリジナルのプログラム処理 2 のキャッシュミス率をシミュレータで調査した結果、処理 2 の 2 次キャッシュミス率は 4.15%でありキャッシュからのデータの追い出しが頻繁に発生していた。従って、縮小するサイズはキャッシュ容量を越えるサイズになるよう3次元目のサイズ(220546)を 1/10 に縮

小し、処理 2 の参照領域を支配するループ数を 30169 から 2410 とした。ここで、全体の処理のバランスを保つためには、処理 2 以外の処理も処理 2 と同様にループの回転数を減らす必要がある。しかし、単純に回転数を減らすと、それに連動して処理する領域も小さくなり、キャッシュミス率が小さくなってしまう。なぜなら、オリジナルプログラムでは、処理 4-a の実行中にキャッシュから追い出される領域が、データ縮小によって処理 4-b 開始時にキャッシュに残ってしまい、本来発生するキャッシュミスが発生しなくなるからである(図 10a)。そこで、キャッシュミスを発生させるため、処理 4-b のアクセスする変数のインデックスを書き換えた(図 10b)。処理 5 についても同様の変更を加えた。

(a)



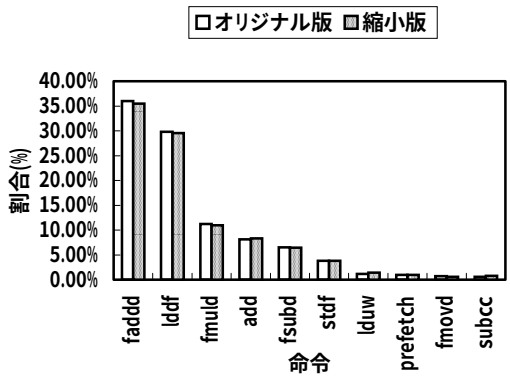
(b)

図 10.オリジナルプログラムと縮小プログラムのデータアクセス(a)データ縮小によるアクセス変数のアドレス変更方法(b)

4.4. 183.equake の評価

高頻度処理部分の抽出、繰り返し回数の削減、データ量の削減を行い、命令数を約 1/46000 の 3.5M 命令とする縮小プログラムを開発し評価した。命令コンビネーションについては、各命令はオリジナルプログラムと割合がほぼ同じであり、プログラムの特徴を保持することができた(図 11a)。また、キャッシュミス率については、データ縮小によるキャッシュアクセスへの影響が大きかったにもにもかかわらず、オリジナルプログラムのキャッシュミス率をほぼ再現することができた(図 11b)。以上2点から、オリジナルプログラムと縮小プログラムは等価であるといえる。

(a)



(b)

	実行命令数(M)	データサイズ(MB)	2次ｷｬｯｼｭﾐｽ率(%)
ｵﾘｼﾞﾅﾙ	162975	31	2.8865
縮小	3.5	22	3.00

図 11.183.equake の評価
(a)命令コンビネーション,(b)命令数と 2 次キャッシュミス率

5. おわりに

ＣＦＰ2000 プログラムのうち、172.mgrid と 183.equake の縮小プログラムを開発した。縮小方法として、高頻度処理部分の抽出、繰り返し回数削減、データ量削減を試み、命令数を大幅に削減する縮小プログラムを開発することができた。

172.mgrid は、提案方法にそってオリジナルプログラムと等価な縮小プログラムを開発することができた。183.equake では、繰り返し数削減において、繰り返しの途中で処理内容が変化することから、変化の前後の処理が 1 ループ内に含まれるようにプログラムを書き換えて抽出した。またデータ量削減では、削減によるキャッシュミスへの影響が大きかったが、変数のアドレス操作によるキャッシュミス率再現方法を考案し、オリジナルと等価な縮小プログラムを開発することができた。

CFP2000 プログラムは、浮動小数点演算の単一ジョブを処理する科学技術計算プログラムであり、繰り返し処理中のジョブ間でＣＰＵ性能が変化しないプログラムが多い。従って、今回縮小プログラムを開発した 172.mgrid,183.equake 以外のＣＦＰ2000 プログラムについても縮小手順に当てはめ縮小バイナリが作成できると考えられる。

今回、縮小プログラムの開発にはプログラムの挙動を把握しながら人手により作成した。CFP2000 プログラム群は縮小プログラムの作成の手順が統一的で、かつ、プログラム構造も比較的単純であることから自動化も考えられ、今後はその手法を考えていきたい。

参考文献

[1] J.L.Henning. SPEC CPU2000: Measuring CPU Performance in the New Millennium, In *IEEE Computer*, 33(7):28-35, July 2000. SPEC CPU2000 Press Rlease FAQ, availale at http://www.spec.org/osg/cpu2000/press/faq.html

[2] T.M.Conte, M.A.Hirsche, and K.N.Menezes. Reducing state loss for effective trace sampling of superscalar processors. In IEEE *International Conference on Computer Design: VLSI in Computers and Processors*, pages 468-477, October 1996. IEEE COmuter Soiety.

[3] P.K.Dubey and R. Nair. Profile-driven sampled trace generation. Technical Report RC 20041, IBM Research Division, April 1995.

[4] S.Laha, J.Patel, and R.Iyer. Accurate low-cost methods for performance evaluation of cache memory systems. *IEEE Transactions on Computers*, 37(11):1325-1336, 1988.

[5] D.A.Wood, M.D.Hill, and R.E. Kessler. A model for estimating trace-sample miss ratios. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 1991.

[6] AJ. KleinOsowski, J. Flynn, N. Meares and D.J.Lilja. Adapting the SPEC 2000 Benchmark Suite for Simulation-Based Computer Architecture Research. *Workshop on Workload Characterization, International Conference on Computer Design*, Austin, TX, Sept 18-20, 2000.

[7] 小野寺　聡, 上田　晴康, 安里　彰,"SPEC CINT2000(181.mcf)の縮小プログラム開発手法とその評価,Feb 14-15,2002

[8] http://www.nas.nasa.gov/Pubs/TechReports/NASreports/NAS-95-020/

[9] http://www.cs.cmu.edu/~quake/