2006－ARC－166（7）
2006／1／24

自動学習による高性能プロセッサ向け命令スケジュール規則の抽出とその実装

川上　肇[†]　梅谷征雄[††]

高性能プロセッサ用のスケジューラとして遺伝アルゴリズムを命令スケジューリングに応用した遺伝的命令スケジューラでは，実行時間の観点からは効果が得られたが計算コストがかかりすぎる問題があった．そこで遺伝的スケジューラの結果を検討し，実行時間に影響を与えると思われる性質を自動学習を使って抽出し，直接コンパイラに組み込むことによって実行時間，計算コストの両面でより実用的なスケジューラを実装した．

Extraction and the implementation of efficient instruction schedule rule for pipeline processor using machine learning method

Hajime Kawakami[†] and Yukio Umetani[††]

In the genetic instruction scheduler that applied the genetic algorithm to the instruction scheduling for an pipeline processor, there was a problem that it requires high calculation cost though if achieves an excellent schedule. Therefore, more practicable scheduler in both the execution time and the scheduling cost was studied by applying machine learning methods to the results obtained by the genetic schedular,and it was incorporated in the existing compiler to test its capability.

1. はじめに

現在，プロセッサのパイプライン制御方式が複雑化しておりプロセッサの性能を十分に引き出せていない可能性がある．これはプロセッサがどの時点をとらえてみても複数の命令をパイプラインの異なるステージで実行しているためである．そのためパイプライン処理の遅延を減らす命令スケジューリングが重要である．そこで遺伝的アルゴリズム[9),10)]を命令スケジュールに応用した遺伝的命令スケジューラが提案された．遺伝的命令スケジューラは，遺伝的アルゴリズムを用いた命令スケジューリング[3),4)]である．静的な評価関数を用いる従来のコンパイル手法とは異なり，対象としたプログラムの実行時間のみに依存したスケジューリングをするため，そのプロセッサの最適化方法が分からなくとも最適化できる．この遺伝的命令スケジューラを Pentium プロセッサに適用した結果，実行時間の観点からは十分な効果が得られた．しかし，計算コストがかかりすぎるという問題があった．

そこで本研究では遺伝的命令スケジューラを LFK の FORTRAN で書かれた 24 個のカーネルプログラムを C に書き直したものに対して実行した結果を使用し，計算コストを減らしより実用的な新たな命令スケジューラを検討し実装した．遺伝的命令スケジューラでは実行時間においては十分な効果が得られたことが報告されているので，効果のある命令列には何かしらの規則性があると思われる．その規則性を自動学習の手法により抽出し，得られた規則性を直接コンパイラに組み込むことにより，遺伝的命令スケジューラにおいて問題であった計算コストを減らすことができた．

† 静岡大学大学院　情報学研究科　情報学専攻
Department of Information, Graduate School of Information, Shizuoka University
†† 静岡大学　情報学部　情報科学科
Department of Information Science, Faculty of Information, Shizuoka University

2. 属性の決定

規則を生成するのに，本研究では自動学習（データマイニング）の手法を用いる．データマイニングの中の決定木学習によって規則を生成するのだが，この規則に使われるのが属性である．例えば if （属性）＝（属性値）ならば結果（class）のような規則が生成される．このような規則を生成するために属性と属性値からなるデータセットの作成を行わなければならない．本研究ではデータセット作成のために遺伝的命令スケジューラの結果を利用する．遺伝的命令スケジューラを実行すると多くの命令列と実行時間を出力するのでこの結果を使って属性を決定する．属性を決定するために，まず命令列の一部分だけが違っており，実行時間が違う二つの命令列を取り出す．この命令列の並びが違っている部分には性能に影響を与える重要な性質があると考えられる．表 1 と表 2 に例を示す．表 1 ではロード系命令である lea 命令と xor 命令で構成されている．ここで二つの命令列を比較すると最速命令列では xor 命令が間に実行されていることが分かる．この例からは，lea 命令のロード命令の連続性が性能に

表 1 命令列の比較 1:ロードの連続性

time 2153（ms:ミリ秒）	最速命令列	time 2163（ms:ミリ秒）	比較命令列
lea	esi,dword ptr [esp+8084]	lea	eax,dword ptr [esp+76]
lea	ebx,dword ptr [esp+16092]	lea	esi,dword ptr [esp+8084]
xor	edx,edx	lea	ebx,dword ptr [esp+16092]
lea	eax,dword ptr [esp+76]	lea	ecx,dword ptr [esp+24100]
lea	ecx,dword ptr [esp+24100]	xor	edx,edx

表 2 命令列の比較 2：浮動小数点命令の連続性

time 406（ms:ミリ秒）	最速命令列	time 453（ms:ミリ秒）	比較命令列
add	edx,40	add	edx,40
add	eax,8	fsubr	qword ptr [esp+8]
add	ecx,5	add	eax,8
cmp	ecx,1001	add	ecx,5
fsubr	qword ptr [esp+8]	cmp	ecx,1001
fstp	qword ptr [esp+8]	fstp	qword ptr [esp+8]

影響を与えるのではないかと考えられる．次に表 2 では最速命令列で浮動小数点命令が連続して実行されているかどうかが二つの命令の違っている部分になっている．この例からは，浮動小数点命令の連続性が性能に影響を与えるのではないかと考えられる．このような手法で属性を決定していく．以下，属性として決定したものと属性値の説明を列挙する．

- 平均依存距離
  - 属性値は各命令の依存する命令との命令数差をそれぞれ計算し平均をとったもの
- オペランドの依存
  - 属性値はオペランドの部分に依存があるものが連続して実行されている場合 1，2，… と加算していく
- ストア命令の連続性
  - 属性値はストア命令が連続して実行されていた場合 1，2，… と加算していく
- 浮動小数点命令のあとに整数命令
  - 属性値は浮動小数点命令のあとに整数命令が実行されている場合 1 を加算していく
- 浮動小数点命令のあとの整数命令の連続性
  - 属性値は浮動小数点命令のあとに整数命令がどの程度連続して続いてるかを調べたもので，整数命令が続けば加算して最後に平均をとる
- ロード命令の連続性
  - 属性値はロード命令が連続して実行されていた場合 1，2，… と加算していく
- 浮動小数点命令の連続性
  - 属性値は浮動小数点命令が連続して実行されていた場合 1，2，… と加算していく
- 浮動小数点命令の平均連続性
  - 属性値は浮動小数点命令が連続してどの程度連続して実行されているかを調べたもので，浮動小数点命令が続けば加算して最後に平均をとる
- 連続ストア命令間のアドレスの距離
  - 属性値はストア命令の中で連続するアドレスがある場合，どの程度離れて実行されているかを計測し最後に平均をとる
- 実行時間
  - 属性値は実行時間が早いものから順に 1 から 5 に分類する

以上の属性を使い次章で規則の生成を行う．

3. 規則の生成

3.1 データマイニング

規則の生成にはデータマイニングを用いる．LFK のカーネルプログラム 24 個に対し，遺伝的命令スケジューラを適用し得られた結果から属性値を計算しデータセットを作成する．データセットを作成したあと，データマイニングツール WEKA[12] を使って決定木学習から規則の生成を行う．表 3 は得られた規則の一部である．この結果を見れば分かるように，各カーネルによって規則に現れる属性値に幅があることが分かる．直接コンパイラに組み込んでやるためには，規則を一般化してやる必要がある．そのために，各カーネルごとに規則を生成するのではなくすべてのカーネルのデータセットをまとめて規則を生成する．

3.2 規則の一般化

カーネルに依存しない一般的な規則を生成するためにすべてのカーネルのデータセットをまとめて同様に決定木学習から規則の生成を行う．表 4 はすべてのカーネルのデータセットをまとめて得られた規則から，実行時間（class）が 1 から 5 の中で早いほうから順に二つまで規則を抽出したものである．事例の割合の/の前の数字は規則にマッチした事例の総数，/の後ろの数字は規則にマッチしたが class がマッチしなかった事例の数になっている．

3.3 属性選択によるルールの簡単化

規則を一般化することはできたが，規則の条件部が

表 3 得られた規則の一部

kernel 番号	class	依存距離の逆数の和	オペランドの依存	ストアの連続	float 後の整数	整数の平均	ロードの連続	float の連続
1	1	≤ 13						
1	2	> 15						> 2
1	2	≤ 15						
2	2				≤ 11	> 0.384	> 1	
3	1	≤ 11						
5	1	≤ 11						
5	2							> 1
5	2				> 2			
5	2						> 1	≤ 1
6	1	≤ 18						
6	2		> 6				> 1	
7	1	> 22			> 2			> 5
9	1	≤ 18						
9	1	> 19	≤ 6					

表 4 一般化して得られた規則の一部：属性選択なし

class	平均依存距離	オペランドの依存	ストアの連続	float 後の整数	整数の平均	ロードの連続	float の連続	float の平均	連続アドレスのストアの距離	事例の割合
1	≤ 1.65								≤ 4	70/11
1		> 0.95					$> 1 and \leq 11$			68/7
2	≤ 1.85		> 1			≤ 0.185	$> 1 and \leq 18$			104/9
2	≤ 3.16	> 0.47	> 1		≤ 2.85 $and > 2.33$	≤ 0.518	> 1			58/10
2		≤ 0.506 $and > 0.5$	> 3	> 4		> 0.5	> 1			50/11
2	≤ 2.44	≤ 0.5	< 0		> 6		$> 1 and \leq 12$			64/11
2		> 0.5	≤ 3						≤ 3.33	72/33
2	> 1.86	≤ 0.5	≤ 4	> 1		≤ 0.22		≤ 0.71		54/19

長く非常に分かりづらい規則であるものが多い．そこでデータマイニングで用いられる手法である属性選択を行う．属性選択とは，性能劣化を引き起こす無関係あるいは冗長な属性を排除する作業である．属性選択の方法は，まず空集合から出発し一つづつ属性を加えて最も分類精度が高い属性を空集合に加える．次にまた集合に一つづつ属性を加えて最も分類精度が高い属性を集合に加える．この操作をデータセットの分類制度が上がるまで繰り返す．この属性選択を行うことの利点としてはデータセットの分類精度を上げることができることと，属性数を絞り込むのでよりわかりやすい規則にすることができる．表 5 は，属性選択して規則を生成したものの中で実行時間が 1 と 2 のものを取り出したものである．表 4 よりも属性数が減っているので，分かりやすい規則になっていることが分かる．

4. 実　　装

スケジューラの処理手順は以下のようになっている．

(1) 対象とする命令列の生成.
遺伝的命令スケジューラで遺伝的枠組みである DAG を参照し作り出される初期命令列を使用する．

(2) 3 章で得られた規則を使って命令列を評価：属性選択ありとなしの二通りで評価する．

(3) 命令列の選択
- (a) 規則にマッチした命令列を実行して評価する方法.
  - (i) 命令列を得られた規則を使い評価し，マッチした命令列のみ選択し実行時間を計測.
  - (ii) 最も実行時間が早いものを最適命令列とする．
- (b) 規則にマッチした命令列を実行しないで評価する方法.
  - (i) 命令列を実行しないで規則を使って命令列を選択．選択する基準として，規則の事例数が多いものを優先する方法と，規則の分類精度が高いものを優先する方法の二通りがある．
  - (ii) 実際に選択された命令列の実行時間が知りたいのでスケジュールが終了した後に実行時間を計測する．

(4) スケジュール時間を計測

表 5 一般化して得られた規則の一部：属性選択あり

class	平均依存 距離	オペランド の依存	ロード の連続	float の平均	事例 の割合
1	$\leq 1.65 and > 1.42$	≤ 0.4		> 1.14	84/21
1		> 0.49		$\leq 1.63 and > 1.5$	68/7
2		> 0.46	≤ 0.5	≤ 0.88	127/30
2		> 0.46			158/63
2	≤ 2.4	> 0.46		$\leq 0.85 and > 0.5$	81/35
2		> 0.46	> 0.43	≤ 0.33	116/27
2	≤ 1.85	> 0.45	$> 0.1 and \leq 0.18$	≤ 1.66	105/10
2	> 3.11				207/115
2	> 2.21	$\leq 0.5 and > 0.45$		$> 1.33 and \leq 1.85$	70/12
2	≤ 2.03	≤ 0.37	$0.1 and \leq 0.36$	≤ 1.25	75/37
2		$\leq 0.22 and > 0.18$	≤ 0.20		56/17
2			$\leq 0.46 and > 0.3$		86/43

5. 性能評価

実装したスケジューラの性能を評価する．評価する方法は，最適化を行っていない命令列（オリジナル），遺伝的命令スケジューラで選択された最適命令列，今回実装したスケジューラで選択された最適命令列の三つを比較する．また初期命令列の数を 10 個と 30 個の二通り，測定値は測定時の誤差を小さくするためにそれぞれ 10 回実行して得た平均値を使って評価する．実験で使用したマシンはメモリ 1024MB であり，Pentium4 3.0GHz を搭載している．OS は WindowXPHE である．

5.1 規則にマッチした命令列を実行する方法

表 6 と表 7 に，それぞれ初期命令列 10 個と 30 個での最速命令列の実行時間とスケジュール時間を示す．この結果からは，初期命令列の数が 10 個よりも 30 個の場合のほうが実行時間の観点からは優れていることが分かる．これはある程度妥当な結果である．またオリジナルよりも劣っている命令列もないことが分かる．次に遺伝的命令スケジューラとの比較だが，多くのカーネルで遺伝的命令スケジューラの最適命令列よりも劣っていることが分かる．しかし，いくつかは遺伝的命令スケジューラと同じ性能のカーネルもあることが分かる．最後にスケジュール時間を見てみると，遺伝的スケジューラに比べて大幅にコストを削減できていることが分かる．

5.2 規則にマッチした命令列を実行しないで選択する方法

表 8 は，規則にマッチした命令列を実行しないで二つの指標を使って命令列を選択する方法で実行した結果である．二つの指標とは，規則の事例数が多いものを優先する方法と，規則の分類精度が高いものを優先する方法である．その中でさらに属性選択を行うものと属性選択を行わないものに分類する．表 8 では事例数が多いものを優先する方法の結果が示されている．この結果からは，スケジュール時間が遺伝的スケジューラは当然として，表 6 や表 7 と見比べて分かるようにさらに計算コストが削減できていることが分かる．これはマッチした命令列を実行しないで実行していないことが大きな原因である．しかし，実際に選択された命令列の実行時間を計測してみると遺伝的スケジューラの最適命令列と比べて大きく劣っており，オリジナル命令列に対しても多くのケースで悪いので，更なる改善が必要である．また分類精度が高いものを優先する方法でも同様の結果が得られた．

6. まとめ

遺伝的スケジューラの問題点であった計算コストを減らすために，本研究では遺伝的スケジューラの規則性を抽出し直接コンパイラに組み込むことによって問題の解決を試みた．表 6 や表 7 からも分かるように，最適命令列の実行時間においてはやや劣っているが計算コストにおいては遺伝的スケジューラに比べて大幅な削減に成功している．しかし計算コストを大幅に削減することはできたが，まだ実用的なレベルに達していない．

そこでさらに計算コストを削減するために，命令列を実行する時間を削る方法に取り組んだ．命令列の選択方法として，規則の事例数と分類精度を使って選択する方法を採用した．結果として表 8 に示されているように計算コストをさらに削減することができたが，実行時間において遺伝的スケジューラに比べて大きく劣っていることになった．

今後の課題として，次の点が挙げられる．

- マッチした命令列の選択方法の改良
- 今回得られた規則を命令列作成部分に組み込む
- より大規模な問題を対象とした評価
- 他の PC プロセッサへの適用実験

参考文献

1) Intel Corporation:“ インテル・アーキテクチャ・ソフトウェア・ディベロッパーズ・マニュアル 上巻 ”, 資料番号 243190J, (1999)
2) Umetani,Y.:“ Application of genetic algorithm to instrucion sequence optimization for risc

表 6 規則にマッチした命令列を実行する方法：初期命令列 10 個
*はマッチした命令列なし

実行時間					スケジュール時間	
kernel 番号	オリジナル (ms)	属性選択あり (ms)	属性選択なし (ms)	遺伝的 (ms)	スケジュール時間 (ms)	遺伝的スケジュール時間 (ms)
1	405	334	375	301	6862	91604
2	188	166	*	156	4164	61993
3	2967	2763	2778	2734	31354	365748
4	411	385	387	375	6689	92293
5	248	228	234	218	3775	68071
6	198	187	*	187	4329	73326
7	311	291	*	279	6421	78439
9	223	209	*	190	5148	74870
10	1122	993	1017	968	15651	193859
11	284	265	267	263	5803	62564
12	84	68	70	62	4840	35484
13	128	109	110	109	6700	63203
14	459	421	425	412	9877	117451
15	359	328	326	318	12375	125740
16	123	109	109	105	6626	60337
17	894	796	796	781	22171	152500
18	78	63	74	62	4712	73656
19	620	412	*	406	7829	110109
20	530	466	527	427	9159	130993
21	375	338	*	328	4445	110517
22	328	301	*	296	6242	92615
23	275	251	254	250	10507	89639
24	173	140	140	125	8198	49068

表 7 規則にマッチした命令列を実行する方法：初期命令列 30 個
*はマッチした命令列なし

実行時間					スケジュール時間	
kernel 番号	オリジナル (ms)	属性選択あり (ms)	属性選択なし (ms)	遺伝的 (ms)	スケジュール時間 (ms)	遺伝的スケジュール時間 (ms)
1	405	309	383	281	19587	304687
2	188	157	*	156	11778	208387
3	2967	2756	2779	2660	87787	1378628
4	411	381	382	375	20214	379840
5	248	226	230	218	10682	277896
6	198	187	187	187	12237	288953
7	311	285	*	268	17984	292532
9	223	203	*	191	14221	317931
10	1122	984	992	968	42696	868870
11	284	265	265	260	15706	214253
12	84	65	63	63	13475	96215
13	128	109	109	109	18981	216410
14	459	421	421	406	30431	561042
15	359	326	325	312	35346	497479
16	123	109	109	101	18974	239264
17	894	793	790	788	64004	673446
18	78	62	76	62	12165	299920
19	620	409	*	398	21682	435631
20	530	453	486	418	27350	505581
21	375	332	*	328	11918	444190
22	328	296	*	296	17128	407212
23	275	251	251	248	30078	409823
24	173	140	140	125	23575	154731

processor. ”Technical Report838,GMD,(1995).
3) 伊東勇，梅谷征雄:“ 遺伝的アルゴリズムを用いる命令スケジューリング方式とその効果 ”, 情報処理学会論文誌,Vol. 41,No. 4,pp. 1073–1085 (2000).
4) 長倉裕叔，梅谷征雄 :“ PC アーキテクチャにおける遺伝的命令スケジューリングの適用実験 ”, 情報処理学会研究報告 Vol. 2000,No. 57,pp. 31–36(2000)

表 8 規則にマッチした命令列を実行しないで選択する方法：初期命令列 10 個
*はマッチした命令列なし，属性選択あり：Y，属性選択なし：N

実行時間					スケジュール時間
kernel 番号	オリジナル (ms)	事例数・Y(ms)	事例数・N(ms)	遺伝的	事例数 (ms)
1	405	439	442	301	820
2	188	214	*	156	573
3	2967	2941	*	2734	3333
4	411	428	422	375	800
5	248	253	*	218	597
6	198	213	*	187	581
7	311	356	*	279	719
9	223	248	*	190	627
10	1122	1138	1138	968	1637
11	284	295	294	263	645
12	84	92	92	62	436
13	128	128	136	109	622
14	459	497	461	412	1117
15	359	366	348	318	1250
16	123	127	137	105	650
17	894	877	859	781	1400
18	78	97	86	62	1122
19	620	639	*	406	1035
20	530	541	536	427	1070
21	375	402	*	328	799
22	328	350	*	296	739
23	275	300	281	250	767
24	173	186	173	125	542

5) 中田育男: “ コンパイラの構成と最適化 ”, 朝倉書店 (1999)
6) Steven S. Muchnick:“ *Advanced* COMPILER DESIGN IMPLEMENTATION ”, MORGAN KAUFMANN, (1997)
7) McMahon, F.H.: “ The livermore fortran kernels: A computer test of numerical performance range ”, Technical report, Lawrence Livermore National Laboratory (Dec. 1986).
8) Intel Corporation:“ インテル・アーキテクチャ・ソフトウェア・ディベロッパーズ・マニュアル 中巻 ”, 資料番号 243191J, (1999)
9) Steven J. Beaty: “ Genetic Algorithms and Instruction Scheduling. ”Proceedings of the 24th Annual International Symposium on Microarchitecture, (Nov. 1991)
10) 坂和正敏, 田中雅博: “ 遺伝的アルゴリズム ”, 朝倉書店 (1996)
11) M.Bekerman, A.Yoaz, F. Gabbay , S.Jourdan ,M.Kalaev , R.Ronen: “ Early Load Address Resolution Via Register Tracking ”, Proc. of the 27th ISCA, pp. 306–315(2000)
12) Weka Machine Learning Project, http://www.cs.waikato.ac.nz/ml/