プログラム記述とプロセッサの高性能化機構の関係

古川友樹[†] 山内利宏[†] 谷口秀夫[†]

オペレーティングシステムにおいて，プログラム記述の工夫により，処理オーバヘッドを削減することは重要である．このためには，プロセッサの高性能化機構を考慮したプログラム記述が必要である．ここでは，周期実行制御のプログラムを事例として挙げ，プログラム記述と処理性能の関係について述べる．また，アーキテクチャの差異による性能への影響について述べる．

The relation between program description and high performance mechanism of processor

Yuuki Furukawa,[†] Toshihiro Yamauchi[†] and Hideo Taniguchi[†]

In operating systems, reduction of processing overheads by optimizing the program description is important. The optimization requires the program description in considering the high performance mechanism of processor. In this paper, we discuss the relation between program description and high performance mechanism of processor using the program for the periodic execution control. In addition, we discribe the influence on the performance by the difference in the architecture.

1. はじめに

オペレーティングシステム（以降，OS）において，プログラム記述の工夫により，処理オーバヘッドを削減することは重要である．このためには，プロセッサの高性能化機構を考慮したプログラム記述が必要である．

ロボットのモータ制御やセンサ制御を行う処理は，周期的に実行される．このように周期的に実行される処理は，次の周期までに 1 周期の実行が終了しない場合，予定されていた処理の実行が行われず，問題となる．また，周期実行制御にかかる制御オーバヘッドが大きいと，周期的に実行される処理の実行可能な時間が短くなってしまう．このため，周期実行制御にかかる制御オーバヘッドは小さく，処理時間の変動は小さいことが望まれる．そこで，周期実行制御に関し，ロボットに利用される OS[1] である ART-Linux[2)3)4)] より制御オーバヘッドの小さい周期実行制御の実現を目標とし，ART-Linux の問題点を解決する周期実行制御法[5)] を提案した．

ここでは，周期実行制御のプログラムを事例として挙げ，プログラム記述と処理性能の関係について述べる．また，アーキテクチャの差異による性能への影響について述べる．

† 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University

2. 周期実行制御

2.1 ART-Linux の問題点と提案制御法の期待される効果

ART-Linux の問題点と提案制御法の期待される効果について，**図 1** にまとめ，以下に説明する．

ART-Linux は，1 つのキューにより待機状態の実時間プロセスを管理する．ART-Linux では，実時間プロセスの起動処理において，WAIT キューの先頭に存在する実時間プロセスの起動待ち時間を計算し，起動を判定する．このため， 実時間プロセスの起動において，制御オーバヘッドは同時に起動する実時間プロセスの数に比例し，処理時間の変動も大きい（問題点 1）．また，待機処理において，WAIT キューを先頭から探索し，起動待ち時間を計算し，待機を要求した実時間プロセスを接続する位置を決定する．このため，実時間プロセスの待機における WAIT キューへの接続処理の制御オーバヘッドは実時間プロセスの数に比例し，処理時間の変動も大きい（問題点 2）．

これに対して，提案制御法では，タイマ割り込みの発生時刻毎に起動する実時間プロセスをまとめて管理する（対処 1）．また，起動要素にキューエントリを 2 つ持たせ，実行待ち管理表への接続と周期制御表への

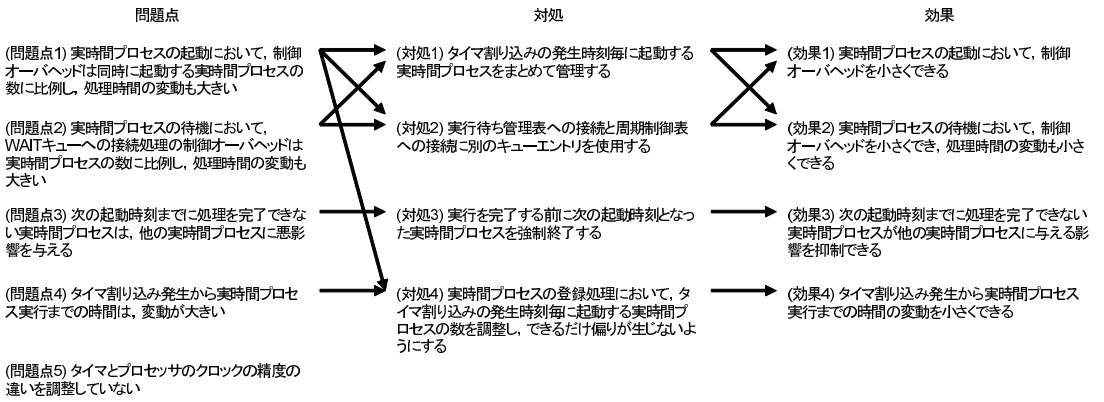

図 1 ART-Linux の問題点と提案制御法の期待される効果

接続に別のキューエントリを使用する（対処 2）．このため，提案制御法では，実時間プロセスの起動と待機において，起動待ち時間の計算，および WAIT キューの接続と削除の処理がなくなり，制御オーバヘッドを小さくでき，処理時間の変動も小さくできる．

（問題点 3）として，次の起動時刻までに処理を完了できない実時間プロセスは，他の実時間プロセスに悪影響を与えることがある．提案制御法では，実行を完了する前に次の起動時刻となった実時間プロセスを強制終了する（対処 3）．

（問題点 4）として，タイマ割り込み発生から実時間プロセス実行までの時間の変動が大きいことがある．この要因の 1 つとして，タイマ割り込み毎に起動する実時間プロセスの数に偏りがあると，実時間プロセスの起動における処理時間の変動は大きいことがある．そこで，提案制御法では，実時間プロセスの登録処理において，タイマ割り込みの発生時刻毎に起動する実時間プロセスの数を調整し，できるだけ偏りが生じないようにする（対処 4）．

2.2 評価（予想と実測）

提案制御法を ***AnT*** オペレーティングシステム（以降，***AnT***）[6] に実現し，評価する．評価項目を以下にまとめる．

(1) 同時に起動する実時間プロセスの数を増加させた場合の起動処理時間

実時間プロセスを N 個登録し，タイマ割り込み発生時において，実時間プロセスの起動処理の開始から N 個の実時間プロセスを全て実行待ちにするまでの時間を測定した．これにより，(効果 1) を明確にする．

(2) 待機状態の実時間プロセスの数を増加させた場合の待機処理時間

実時間プロセスを N 個登録し，待機を要求した実時間プロセスより先に起動する実時間プロセスが $N-1$ 個存在する場合において，実時間プロセスが待機を要求してから，待機状態となるまでの時間を測定する．これにより，(効果 2) を明確にする．

(3) 起動する実時間プロセス数

タイマ割り込み周期 1ms のとき，周期 100ms の実時間プロセスを不規則なタイミングで 100 個登録し，各タイマ割り込みの発生時刻において，起動する実時間プロセスの数を測定する．これにより，(効果 4) を明確にする．

提案制御法の効果から予想される測定結果を**図 2** に示し，Celeron D（2.8GHz）プロセッサにおける測定結果を**図 3** に示す．また，測定に使用した実時間プロセスの処理内容は，待機のシステムコールを繰り返し発行するのみである．なお，***AnT*** と ART-Linux 上で動作する実時間プロセスの数は同じである．図 2 と図 3 より，待機処理時間と起動する実時間プロセスの数は，予想通りの結果を得られたことがわかる．しかし，予想に反して，起動処理時間は，ART-Linux より ***AnT*** の方が大きい．

3. プログラム記述の問題点と対処

3.1 問 題 点

予想に反して，起動処理時間が大きくなった原因として，キャッシュミスの増大が考えられる．***AnT*** のプログラム記述を**図 4** に示し，プログラム記述の問題点を**図 5** に示す．図 5 の rflag は実時間プロセスの周期や状態を保持し，rpri は優先度を保持する．***AnT*** において，キャッシュミスが増大する要因として，以下の 2 つがある．

(1) 起動要素の参照

AnT の起動処理において，周期制御表のカレントエ

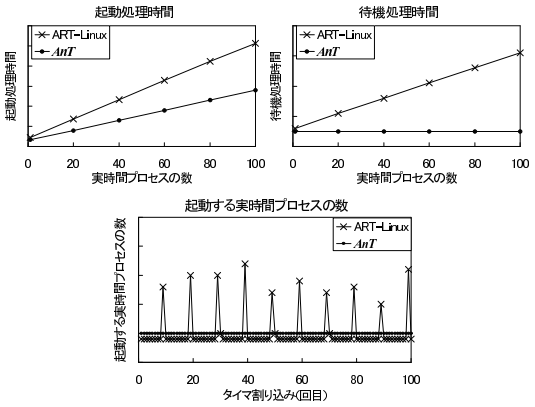

図 2 測定結果（予想）

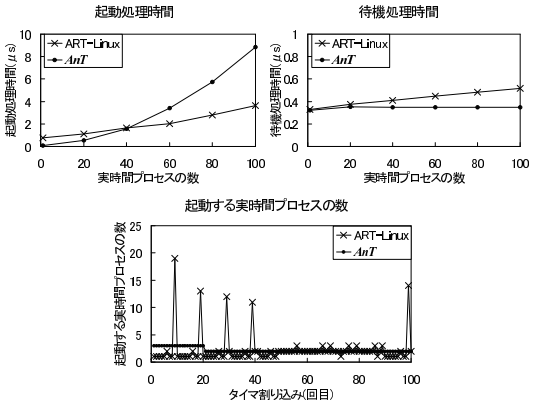

図 3 測定結果（実測）

ントリに存在する起動要素を参照し，実行待ち管理表に接続する．また，図 4 のプログラム記述において，OS の初期化処理時に確保した起動要素を先頭から順番に使用している．このため，起動要素はプロセス単位でメモリ上にまとまって存在することになる．したがって，プロセスの数が増えると，1 回の起動処理において，参照する起動要素がメモリ上で不連続に存在することになり，キャッシュミスを増大させる要因となる．

(2) 実時間プロセスの情報の参照

起動処理において，起動する実時間プロセスの情報を参照する必要がある．図 4 のプログラム記述において，優先度，周期や状態といった実時間プロセスの情報はプロセス管理表が保持する．プロセス管理表は多くの情報を持つため，プロセス管理表が実時間プロセスの情報を保持すると，プロセス管理表の参照により，キャッシュから次に参照するデータが削除される可能性がある．このため，プロセスの数が増えると，キャッ

```
1.  for (tsk = curpointp->head; tsk;) {
2.
3.    if (tsk->procp->rflag & RT_READY) {  /* 周期を超過する場合 */
4.      …                                  /* 強制終了 */
5.      continue;
6.    }
7.
8.    /* 実行待ち管理表の末尾に起動要素を追加 */
9.    tsk->prinext = NULL;
10.   rpri = tsk->procp->rpri;
11.   *rtreadyq[rpri].last = tsk;
12.   rtreadyq[rpri].last = &tsk->prinext;
13.
14.   tsk->procp->rflag |= RT_READY;       /* 実行待ちフラグを設定 */
15.
16.   *rtqbits |= 1 << rpri;
17.
18.   tsk = tsk->next;
19. }
```

(2) プロセス管理表の参照

(1) メモリ上で不連続に存在

図 4 プログラム記述

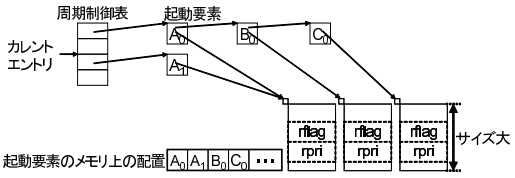

図 5 プログラム記述の問題点

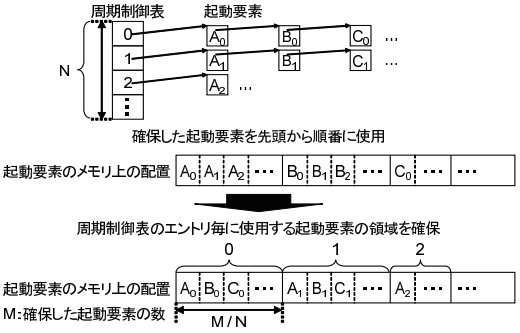

図 6 参照順を意識した情報格納

シュミスが増大する要因となる．なお，起動処理において，実時間プロセスの情報以外のデータをプロセス管理表から参照することはない．

3.2 対　　処

3.2.1 参照順を意識した情報格納

起動要素の参照順を意識し，周期制御表のエントリ毎に存在する起動要素がメモリ上で連続に存在するように情報格納を行う．参照順を意識した情報格納の様子を**図 6** に示し，以下に説明する．

OS の初期化処理時に確保した起動要素を先頭から順番に使用するのではなく，周期制御表のエントリ毎に使用する起動要素の領域を確保する．具体的には，OS の初期化時において，起動要素を M 個の配列として確保する．M（確保した起動要素の総数）を N（周

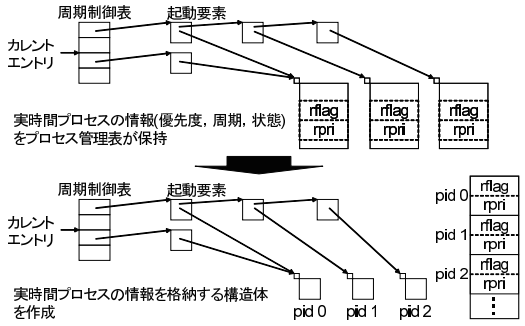

図 7 情報格納域の集中化

```
1.  for (tsk = curpointp->head; tsk;) {
2.      pid = tsk->procp - proctbl;
3.
4.      if (rttaskinfo[pid].rflag & RT_READY) {/* 周期を超過する場合 */
5.          ...                                /* 強制終了 */
6.          continue;
7.      }
8.
9.      rttaskinfo[pid].rflag |= RT_READY;     /* 実行待ちフラグを設定 */
10.
11.     /* 実行待ち管理表の末尾に起動要素を追加 */
12.     tsk->prinext = NULL;
13.     rpri = rttaskinfo[pid].rpri;
14.     *rtreadyq[rpri].last = tsk;
15.     rtreadyq[rpri].last = &tsk->prinext;
16.
17.     rtqbits |= 1 << rpri;
18.
19.     tsk = tsk->next;
20. }
```

(2) 実時間プロセスの情報格納域の参照

(1) メモリ上で連続に存在

図 8 プログラム記述（改善後）

期制御表のサイズ）で割った値を x とする．周期制御表の i 番目のエントリに起動要素を接続する場合，確保した起動要素の $x \times i$ 番目の位置から探索し，未使用の起動要素を用いる．これにより，周期制御表の各エントリの起動要素はメモリ上で連続して存在することになる．したがって，データの局所性が向上し，実時間プロセスの起動処理におけるデータのアクセス時間は短くなる．

3.2.2 情報格納域の集中化

情報格納域の集中化を行い，周期実行制御と関係しないデータを読み込まないようにする．情報格納域の集中化の様子を**図 7** に示し，以下に説明する．

実時間プロセスの情報をプロセス管理表が保持するのではなく，実時間プロセスの情報を格納する構造体を作成し，この構造体に実時間プロセスの情報を格納する．また，作成した構造体の各エントリの番号がプロセス識別子に対応する．これにより，起動処理において，周期実行制御と関係しないデータの読み込みがなくなる．したがって，データの局所性が向上し，実時間プロセスの起動処理におけるデータのアクセス時間は短くなる．

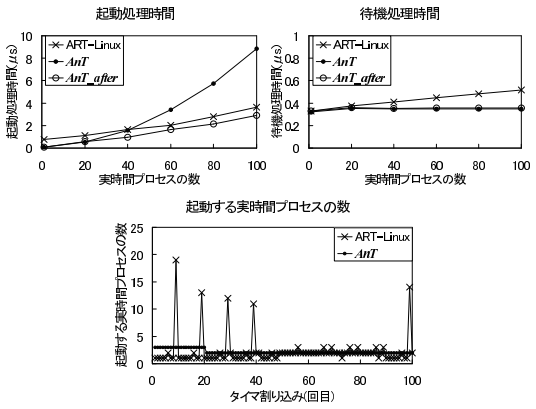

図 9 測定結果（改善後）

3.3 評価（改善後）

改善後のプログラム記述を**図 8** に示し，プログラム記述を改善した後の測定結果を**図 9** に示す．以降では，プログラム記述後の ***AnT*** を ***AnT_after*** とする．図 9 より，以下のことがわかる．

(1) ***AnT_after*** の起動処理時間は，ART-Linux より短い．

AnT_after の起動処理時間は ART-Linux の約 81%である．これは，提案制御法の起動処理において，起動待ち時間の計算とキューの削除処理がないためである．また，***AnT*** と ***AnT_after*** の比較により，プログラム記述の改善により，起動処理時間を約 67%短縮できたことがわかる．

(2) プロセス数が少ない場合，***AnT*** の起動処理時間は，***AnT_after*** より短い．

実時間プロセスの情報を参照する処理において，プロセス管理表とは別の構造体を参照するため，プロセス識別子の取得，内部変数への代入などの処理が増える．このため，プロセス数が少ない場合，***AnT*** の起動処理時間は，***AnT_after*** より短いと考えられる．

(3) ***AnT*** の待機処理時間は短く，一定である．

ART-Linux では，待機処理において，WAIT キューの探索と起動待ち時間の計算，READY キューからの削除，および WAIT キューへの接続を行う．このため，ART-Linux の待機処理時間は，待機を要求した実時間プロセスより先に起動する実時間プロセスの数に比例し，増加する．一方，提案制御法では，実行待ち管理表から起動要素を削除するのみである．このため，***AnT*** では，待機状態の実時間プロセスの数に関わらず，待機処理時間は一定である．

(4) ***AnT*** は，起動する実時間プロセス数の偏りが小さい．

表 1 対象プロセッサ

プロセッサ	周波数	キャッシュ			パイプライン
		L1	L2	ラインサイズ	
SH-4	240MHz	命令：16KB(2way) データ：32KB(2way)		32Bytes/Line	5 段
Pentium II	400MHz	命令：16KB(4way) データ：16KB(4way) (Write Allocate)	512KB(4way) 速度は L1 の 1/2	32Bytes/Line	14 段
Celeron D	2.8GHz	命令：12KμOP(8way) データ：16KB(8way)	256KB(4way)	6μOPs/Line 64Bytes/Line	31 段

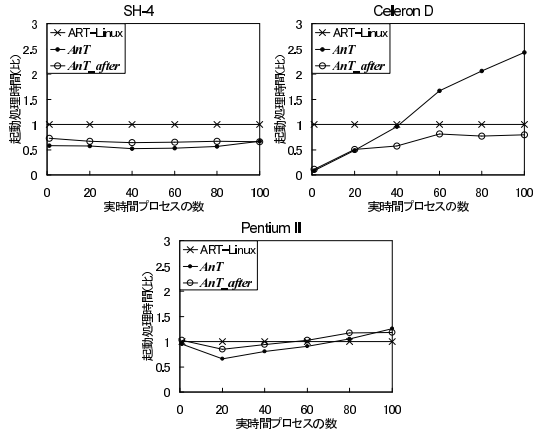

図 10 起動処理時間

提案制御法において，周期制御表に起動要素を接続する際，管理する起動要素の少ないエントリを探索し，発見したエントリに起動要素を接続する．このため，周期制御表の各エントリに接続する起動要素の数について，偏りが生じることが少なくなる．

4. プロセッサの高速化機構と処理性能

4.1 対象プロセッサ

アーキテクチャの差異による処理性能への影響を明確化するため，SH-4，Pentium II，および Celeron D において，提案制御法の評価を行った．**表 1** に対象としたプロセッサの情報をまとめる．3 つのプロセッサにおける測定結果を比較する．なお，***AnT*** は，プログラム記述の改善前と改善後のそれぞれの場合において測定を行った．

4.2 基 本 評 価

4.2.1 起動処理時間

3 つのプロセッサ上で測定した起動処理時間の結果を**図 10** に示す．図 10 の起動処理時間は，ART-Linux の値を 1 としたときの ***AnT*** と ***AnT_after*** の値の比である．図 10 より，以下のことがわかる．

(1) ***AnT*** の起動処理時間において，SH-4 は ART-Linux より短く，Pentium II は ART-Linux と同等であり，Celeron D は ART-Linux より長い．

AnT の起動処理時間は，実時間プロセスの数の増加に伴い，長大化する．これは，***AnT*** のデータの局所性が低く，実時間プロセスの数が増えると，キャッシュミスの確率が増大するためである．また，起動処理時間の増加する時期や増加量は，3 つのプロセッサにおいて異なる．これは，3 つのプロセッサにおいて，データキャッシュの容量やデータの更新方式，およびラインサイズが異なるためであると考えられる．Celeron D は，データキャッシュのラインサイズが 64 バイトであり，他の 2 つのプロセッサより大きいため，データの局所性が低い ***AnT*** において，キャッシュミスが増大する時期が最も早く，増加量も大きいと推察される．

(2) ***AnT*** と ***AnT_after*** より，プログラム記述の改善により，キャッシュミスの確率が低くなり，起動処理時間が短くなる．

3 つのプロセッサにおいて，実時間プロセスの数が少ない場合，***AnT_after*** の起動処理時間は ***AnT*** より短い．これは，***AnT_after*** において，情報格納域の集中化に伴い，代入などの処理が増加したためである．しかし，実時間プロセスの数が増加すると，***AnT_after*** の起動処理時間は ***AnT*** より短くなる．これは，プログラム記述の改善により，データの局所性が向上したため，実時間プロセスの数の増加に伴う，キャッシュミスの確率が低くなったためである．

(3) ***AnT_after*** の起動処理時間において，SH-4 と Celeron D は ART-Linux より常に短く，Pentium II は ART-Linux と同等である．

Pentium II において，起動処理時間が ART-Linux と同等である要因として，***AnT_after*** のデータの局所性が ART-Linux より低いこととがある．例えば，***AnT_after*** は，ART-Linux と異なり，1 個の実時間プロセスに対し，複数の起動要素を用い，実時間プロセスの情報を参照するために，起動要素やプロセス管理表とは別の構造体を参照する．また，Pentium II において，書き込み時のミスヒットにおけるキャッシュ

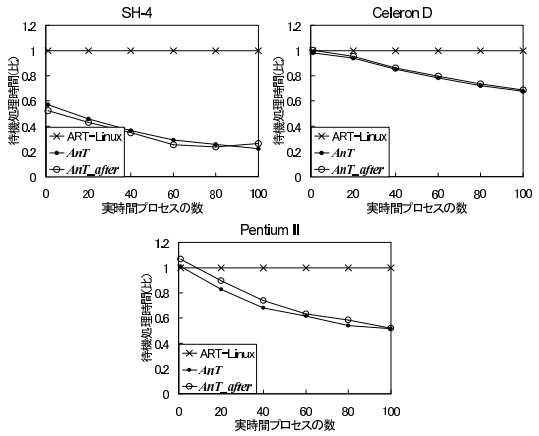

図 11 待機処理時間

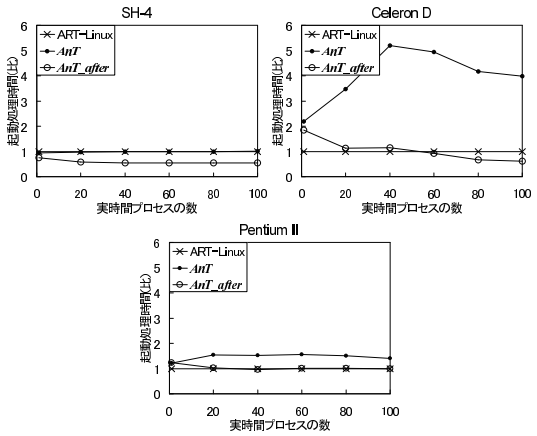

図 12 他処理の影響（起動処理時間）

の更新方式が Write Allocate であり，かつ L2 キャッシュへのアクセス速度が L1 の 1/2 である．これらの要因により，キャッシュミスによるペナルティが ART-Linux より大きくなり，***AnT_after*** の起動処理時間と同等になったと推察する．

4.2.2 待機処理時間

3 つのプロセッサ上で測定した待機処理時間の結果を**図 11** に示す．図 11 の待機処理時間は，ART-Linux の値を 1 としたときの ***AnT*** と ***AnT_after*** の値の比である．図 11 より，以下のことがわかる．

(1) 3つのプロセッサにおいて，***AnT*** と ***AnT_after*** は ART-Linux より短く，一定である．

これは提案制御法の効果によるものである．

(2) ***AnT*** と ***AnT_after*** より，待機処理時間はプログラム記述の改善による影響が小さい．

提案制御法における待機処理は，実時間プロセスの数に関わらず，1 回の WAIT キューからの削除のみである．このため，プログラム記述の改善による影響が小さいと推察される．

(3) 実時間プロセスの数が増加すると，***AnT_after*** の待機処理時間が増加する．

これは，実時間プロセス数の増加に伴うキャッシュミスの増加と考えられる．

4.3 他処理の影響

4.2 節における評価では，測定に関するプロセス以外の処理は実行されていない．このため，周期実行制御において，参照する多くのデータがキャッシュに存在する状態で，測定を行っていた．そこで，実際の環境を想定し，キャッシュサイズを超えるデータを読み込む非実時間プロセス（他処理）を実行させた場合の起動処理時間と待機処理時間を測定する．

図 12 に起動処理時間の測定結果を示す．図 12 より，以下のことがわかる．

(1) 他処理が存在する場合，***AnT*** の起動処理時間は ART-Linux より長い．

他処理が存在する場合，起動処理において参照するデータがキャッシュにほとんど存在しない状態となる．また，***AnT*** は，データの局所性が低いため，メモリアクセスの回数が多くなる．このため，他処理が存在する場合，***AnT*** の起動処理時間は ART-Linux より長い．

(2) 他処理が存在する場合，実時間プロセスの数が少ないと，Celeron D における ***AnT_after*** の起動処理時間は ART-Linux より長い．また，実時間プロセスの数が多いと，***AnT_after*** の起動処理時間は ART-Linux より短い．

この原因として，***AnT_after*** のメモリアクセスの回数が ART-Linux より多いことが考えられる．起動処理において，1 個の実時間プロセスに対し，ART-Linux は 1 つの構造体を参照するが，***AnT_after*** は複数の構造体を参照する．また，Celeron D はラインサイズが大きいため，一度のメモリアクセスで，複数の実時間プロセスの情報を読み込むことができる．このため，実時間プロセスの数が少ない場合，***AnT_after*** のメモリアクセスの回数が ART-Linux より多くなっていると推察する．

(3) 図 10 と図 12 の比較により，***AnT*** は，***AnT_after*** と ART-Linux に比べ，他処理の影響による起動処理時間の増加が大きい．

これは，***AnT*** のデータの局所性が低く，起動要素や実時間プロセスの情報を参照するたびにメモリアクセスが発生するためと考えられる．一方，***AnT_after***

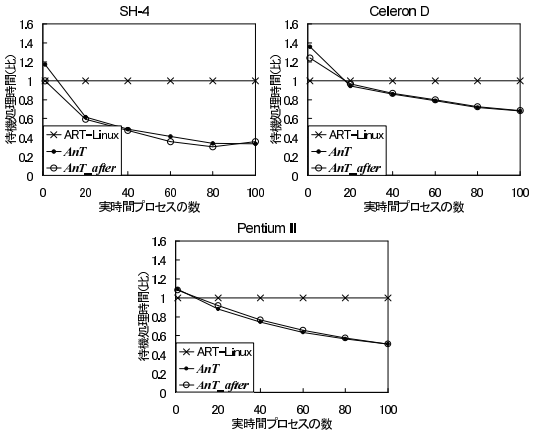

図 13 他処理の影響（待機処理時間）

や ART-Linux では，1 回のメモリアクセスで複数の実時間プロセスの情報を読み込むことができる．Celeron D は，ラインサイズが大きいため，上記の影響が最も大きい．

(4) 他処理が存在する場合，プログラム記述の改善による効果が大きい．

図 10 と図 12 の比較により，他処理が存在する場合，***AnT*** と ***AnT_after*** の起動処理時間の差が大きくなることがわかる．したがって，他処理の影響を考慮した場合，プログラム記述の改善による効果は大きい．

(5) SH-4 は提案制御法の効果が大きい．

図 10 と図 12 の結果より，SH-4 は提案制御法の効果が大きいことがわかる．SH-4 において，***AnT*** と ***AnT_after*** の起動処理時間は，常に ART-Linux と同等か，それより短い．一方，Pentium II や Celeron D では，ART-Linux より起動処理時間が長い場合がある．このため，SH-4 における提案制御法の効果は大きい．この原因として，SH-4 はキャッシュやパイプラインといったプロセッサの高性能化機構による影響が小さいことが挙げられる．

図 13 に待機処理時間の測定結果を示す．図 13 より，以下のことがわかる．

(1) 他処理がある場合でも，3 つのプロセッサにおいて，***AnT*** と ***AnT_after*** の待機処理時間は ART-Linux より短く，一定である．

待機処理において，ART-Linux では，READY キューからの削除，WAIT キューの探索と起動待ち時間の計算，および WAIT キューへの接続を行う．一方，提案制御法では，実行待ち管理表からの削除のみである．このため，他処理が存在する場合でも，***AnT*** と ***AnT_after*** の待機処理時間は短く，一定である．

(2) 他処理が存在する場合，実時間プロセスの数が 1 のとき，***AnT*** と ***AnT_after*** の待機処理時間は ART-Linux より長い．

実時間プロセスの数が 1 の場合，タイマ割り込みから 1 番目に実行された実時間プロセスの待機処理となり，参照するデータがキャッシュに存在しないことがある．一方，実時間プロセスの数が 2 以上の場合，タイマ割り込みから 20 番目に実行された実時間プロセスの待機処理となり，待機処理で共通して参照するデータはキャッシュに存在する状態となる．このため，実時間プロセスの数が 1 の場合，***AnT*** と ***AnT_after*** の待機処理時間は長い．

(3) 待機処理時間は他処理の影響が小さい．

図 11 と図 13 の比較により，待機処理時間は他処理の影響が小さいことがわかる．待機処理時間の測定において，タイマ割り込み発生後，起動処理，1 番目の実時間プロセスの実行，待機処理の順に処理が実行される．このため，実時間プロセスの数が 1 の場合でも，起動処理と待機処理で共通に参照されるデータはキャッシュに存在する．また，実時間プロセスの数が 20 から 100 の場合，待機処理で共通して参照するデータがキャッシュに存在する状態となる．このため，今回の評価において，他処理（非実時間プロセス）による影響は小さかったと考える．

5. おわりに

周期実行制御のプログラムを事例として挙げ，プログラム記述と処理性能の関係について考察した．キャッシュを意識しない場合とした場合のプログラムを比較した結果，キャッシュを意識したプログラム記述により，起動処理時間を約 67%短縮できた．また，アーキテクチャの差異による性能への影響を明確にするため，SH-4，Pentium II，および Celeron D において，提案制御法の評価を行った．評価結果から，「SH-4 は，高性能化機構の影響が小さいため，提案制御法による効果が大きい」，「Pentium II は，キャッシュミス時の更新方式が Write Allocate，かつ L1 キャッシュミスによる影響が大きいため，データの局所性による影響が大きい」，「Celeron D は，ラインサイズが大きいため，データの局所性の影響が Pentium II より大きい場合がある」ことを示した．更に，実際の環境を想定し，キャッシュサイズを超えるデータを読み込む他処理を実行させた状態で測定を行った．この結果から，キャッシュに参照するデータが存在しない場合，プログラム記述の改善による処理性能の向上が大きいことを示した．

質 疑 応 答

Q1 どのようにしてキャッシュミスが原因と判断したのか.（島根大 鈴木）

A1 実際にキャッシュミスの回数を測定したのではなく，起動処理時間の増加から判断した.

Q2 プログラム記述の改善による起動処理時間の短縮は，キャッシュミスの削減による効果よりもキャッシュの先読みによる効果が得られることの影響が大きいのではないか.（同上）

A2 キャッシュの先読みによる効果はあると考えているが，どちらの影響が大きいかはわからない.

Q3 異なる OS で比較しているが，評価として妥当か.（電通大 岩崎）

A3 妥当である．評価において測定した時間は，カーネル内部の周期実行制御に関わる部分の処理時間であり，OS の違いによる影響が出る箇所ではない.

Q4 キャッシュの影響ではなく，プログラム記述を改善した部分以外にバグがあり，起動処理時間が二次曲線的に増加したのではないか.（高知工科大 山崎）

A4 キャッシュミスが原因である．プログラム記述の改善前の ***AnT*** では，データの局所性が低いため，実時間プロセスの数が増えると，キャッシュミスの回数が指数的に増加し，起動処理時間が増加する.

Q5 キャッシュミスが原因だとすれば，実時間プロセスの数を更に増やした際に，起動処理時間のグラフは二次曲線から直線に変わるはずである.（同上）

A5 図 10 の起動処理時間は，実時間プロセスの数が更に増えると，図 12 と同じ傾きの直線になると考えられる．図 12 は，起動処理において参照するデータがキャッシュにほとんど存在しない状態である.

Q6 起動する実時間プロセスの数を調整しているが，問題にならないのか．起動要素の位置をスケジューラが勝手に変更していることになるのではないか.（電通大 寺田）

A6 実時間プロセスの登録処理において実時間プロセスの数を調整しているが，実際に周期実行されているときに起動要素の位置が変更されることはないため，問題にならない.

Q7 周期的に動いている複数の実時間プロセスを同時に起動させたいという要求はないのか.（同上）

A7 ないと考えている．必ず同じタイミングで起動させたい複数の実時間プロセスがある場合，1 つのプロセスにまとめるか，特定のプロセスと同じ時刻に起動するように調整できる機能を追加する必要がある.

Q8 マルチコア化に伴い，キャッシュが複雑化すると考えられるが，マルチコア対応への見通しはどうなっているか.（高知工科大 山崎）

A8 グローバルスケジューラの場合，キャッシュに関しては，今回の結果と似た傾向の結果が得られると考える．しかし，パーティショニング方式では，コアごとにスケジューラを持つため，キャッシュの影響は予測できない.

謝　　辞

本研究の一部は，科学研究費補助金（課題番号 21500055）による.

参 考 文 献

1) K. Yokoi, F. Kanehiro, K. Kaneko, S. Kajita, K. Fujiwara, H. Hirukawa: Experimental Study of Humanoid Robot HRP-1S, International Journal Robotics Research, Vol.23, No.4-5, pp. 351–362, (2004).

2) 石綿陽一: SMP カーネルに基づく ART-Linux の安定化と実時間処理性能の測定, 第 3 回計測自動制御学会システムインテグレーション部門講演会 論文集, (2002).

3) 石綿陽一: SH-4 プロセッサ上の ART-Linux の開発とその品質管理への応用, 第 3 回計測自動制御学会システムインテグレーション部門講演会 論文集, (2002).

4) 石綿陽一, 加賀美聡, 西脇光一, 松井俊浩: シングル CPU 用 ART-Linux 2.6 の設計と開発, 日本ロボット学会誌, Vol. 26, No. 6, pp. 546–552, (2008).

5) Y. Furukawa, T. Yamauchi, H. Taniguchi: Implementation and Evaluation for Sophisticated Periodic Execution Control in Embedded Systems, International Journal of Control and Automation, Vol. 4, No. 2, pp. 59-78, (2011).

6) 岡本幸大, 谷口秀夫: ***AnT*** オペレーティングシステムにおける高速なサーバプログラム間通信機構の実現と評価, 電子情報通信学会論文誌 (D), Vol. J93-D, No. 10, pp. 1977-1989, (2010).