社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report

変数の生存期間を考慮してヒープメモリ使用量削減を行うマルチタスクスケジューリング手法の検討

船瀬　広岐　　中田　明夫

広島市立大学　大学院情報科学研究科 システム工学専攻
E-mail:　{ funase@sos.info. , nakata@ } hiroshima-cu.ac.jp

あらまし　マルチタスクシステムにおいて，他のタスクへの処理の切り替え（コンテキストスイッチ）が発生すると，実行中のタスクは作業用に確保したメモリを保持したまま停止するので，その分多くのメモリを消費してしまう．複数のタスクが同時には使用しないメモリを共有化することによりメモリ使用量削減を行うことができるが，タスク実行時に動的に確保・解放が行われるヒープメモリに関しては，同時に使用するか否かの静的な解析は一般に困難であった．

本研究では，分岐や繰り返し構造を持たない独立のタスク群から構成されるマルチタスクシステムを対象とし，変数の生存期間解析により得られる各タスクのヒープメモリ使用量の時系列変化の情報を用いて，マルチタスクシステムのメモリ使用量を抑えるスケジューリング手法について検討する．

キーワード マルチタスク，コンテキストスイッチ，ヒープメモリ，スケジューリング，変数生存期間解析

On Multi-Task Scheduling for Reducing Heap Memory Consumption Using Live Variable Analysis

Hiroki FUNASE　and　Akio NAKATA
Department of Systems Engineering, Graduate School of Information Sciences, Hiroshima City University

E-mail:　{ funase@sos.info. , nakata@ } hiroshima-cu.ac.jp

Abstract Multitasking system often consume more memory than singletasking system, because when context switching occurs, the running task stops executing while preserving its working memory. Although such memory consumption can be reduced by sharing the working memory that is not used by multiple tasks at the same time, it is challenging to analyse statically which memory can be used at the same time by multiple tasks, especially for heap memory that is allocated dynamically during task execution. In this paper, we consider a scheduling technique to reduce heap memory consumption of multitasking systems which consists of mutually independent tasks with no branches nor loops by using the information of the time sequence of heap memory usage extracted by live variable analysis.

Keyword　multitasking, context switch, heap memory, scheduling, live variable analysis

1. まえがき

携帯電話や家電などに代表される組込みシステムは，特に大量生産されるものについては，製造コストの削減のために，しばしば必要最低限のメモリでの動作が要求される[1]．そのため，組込みソフトウェア開発では，メモリ使用量を抑えることも 1 つの開発目標である．メモリ搭載量を抑えることで，消費電力削減，ハードウェア機器の小型化，製造コスト削減を行うことができる．

一方で，多くの組込みソフトウェアは，複数のタスクを切り替えて並行処理を行うマルチタスクシステムで構成されている．実際に市場に流通しているマルチタスクシステムで動作する製品には，パソコンや携帯電話など，10～100 個のタスクを扱う製品も存在する．しかし，そのようなマルチタスクシステムにおいて，他のタスクへの切り替え（コンテキストスイッチ）が発生すると，実行中のタスクは作業用に確保したメモリを保持したまま停止するので，その分多くのメモリを消費する．例えば，図 1 の例において，最初に Task3 の処理を中断し Task2 に処理が切り替わることにより，

Task3 の処理が再開されるまでメモリを確保し続ける．同様に，Task2 の処理を中断し Task1 に切り替わることで，Task2 の処理を中断し，処理が再開されるまで Task2 はメモリを確保し続ける．このとき，Task2 と Task3 の 2 つのタスクで使用された分のメモリ量を消費することになる．一般に，マルチタスクシステムは，メモリ消費量を考慮せず様々なスケジューリング方針でタスクの処理を切り替えており，システム全体では，メモリ消費量の大きい状態を訪れる可能性を持つ．

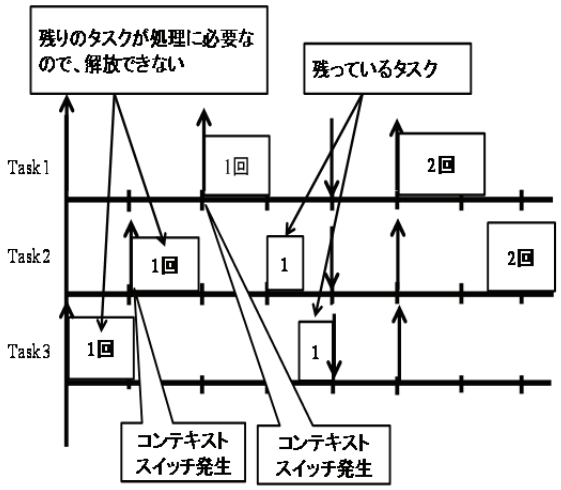

図 1：コンテキストスイッチ発生時における
各タスクのメモリ消費状況
Fig.1 Memory Consumption Situation when Context Switch Occurs in Multitasking

従来のマルチタスクシステムのメモリ使用量削減を目的とした研究には[2]，[3]などがある．しかし，作業用に各タスクが動的に確保したヒープメモリの使用量削減に焦点を当てた研究は我々が知る限り存在しない．メモリ使用量削減のみを目的とする場合，複数のタスクをマルチタスクシステムで処理せずにシングルタスクシステムで処理することにより達成可能である．しかし，シングルタスクシステムでは待ち時間が長くなるタスクが発生するという問題があるので，タスクの応答性を保つためにはマルチタスク実行が必要である．タスク間で同時に使用しないメモリ領域を見つけ共有化することにより，メモリ使用量削減が可能となる．そこで本研究では，マルチタスクシステムを対象としたヒープメモリ使用量削減手法を提案し，その手法を用いた場合のメモリ使用量削減効果と各タスクの応答性の評価検討を行う．

提案手法の概要は以下の通りである．最初に各タスク内で用いている各変数について生存期間解析[5]を行う．ここで変数の生存期間とは，タスク内の全ての変数が持つ変数の値が定義されてから使用が終わるまでのプログラムコードの範囲を意味する．次に，求めた変数の生存期間から，タスクが消費するメモリ使用量の時系列変化を求める．その情報をマルチタスクシステム実行時に用いて，複数のタスクをマルチタスクシステムで並行して実行する．このとき，提案手法では，メモリ使用量の増分が最も小さい次状態を持つタスクをスケジュールする．このことにより，メモリ使用量の大きい状態を避けたマルチタスクスケジューリングが実現できる．提案手法を用いた場合と用いなかった場合とで最大のメモリ使用量(最悪メモリ使用量)とタスクの最大の処理待ち時間(最悪 CPU 獲得待ち時間）を比較し，提案手法の評価を行う．

2. 関連研究

本研究に関連するマルチタスクシステムに対するメモリ使用量削減手法として，文献[2]と[3]が存在する．文献[2]では，並行処理を行うプログラミング言語 SHIM [4] に対するメモリ削減手法を提案している．この研究で用いられているタスクは SHIM によって記述されており，そのタスク内の変数はタスク間で通信を行うチャンネルとして用いられている．このチャンネルを介して通信するたびに各タスクは処理動作の同期を行う．この論文では，並行処理を行うタスク間で通信を行うために変数を使用し，その変数を通信チャンネルとして用いている．そのチャンネルから同時に使用しない組合せを見つける．その組合せをバッファ共有可能なものとみなし，メモリ削減を行う．なお，この解析はタスク群のコンパイル時に静的に行っている．

それに対し，本研究では，タスクのプログラムコードとして用いる変数からタスクが各状態で消費するメモリ使用量を求める．そのメモリ使用量から，マルチタスクシステム実行時に，システム内の各状態が消費するメモリ使用量を求め，その情報をもとにメモリ削減が行えるようなスケジューリング制御を動的に行う．

文献[3]はマルチタスクシステムに対する別のメモリ削減手法を提案している．文献[3]の手法においては，対象とするタスク群を，実行開始時にそのタスクで利用する共有変数の全ての値を読込み（フェッチ），処理を実行し（実行），最後に更新した全ての共有変数の値を書き込む（ライトバック）という 3 つの段階から構成されるものに制限することにより，タスク間共有変数の生存期間解析を容易化している．解析により，共有化によりできるだけ共有変数領域を削減できるように，なるべく同時に利用しないような静的スケジュールを線形計画法に帰着して導出している．

本研究と関連研究[2]の論文との違いとしては，まず，メモリ削減に用いる変数の生存期間の長さが挙げられる．通信に用いる変数は，送受信を行うときのみに使

用するので，生存期間は短く，またその期間は自明に求められる．しかし，作業用に用いる変数は，いつ作業が終わり,変数の使用が終了するかわからないため，生存期間解析が必要となる．また，一般に変数の生存期間は通信に用いるときよりも長く使用される．よって，本研究は関連研究[2]と比較し，変数の生存期間解析が必要な分,より難しい問題を扱っているといえる．

一方で，関連研究[2]と[3]は静的に解析を行い，メモリ削減を行っているが，提案手法はマルチタスクシステム実行時のスケジューリング方法を工夫することにより動的にメモリ削減を行う．このことにより，メモリ削減量の最適性は必ずしも保証されないが，関連研究[2]で生じていた状態爆発問題を回避でき，かつ，関連研究[3]で課していた各タスクの変数定義・使用の一に対する制限も必要ない．

3. 提案手法

本研究では，分岐や繰り返し構造を持たない独立のタスク群から構成されるマルチタスクシステムを対象とし，マルチタスクシステム実行時における最悪メモリ使用量を持つ状態を避けるようにスケジューリングを制御することで，メモリ使用量を抑える手法を提案する．

この手法は，プログラムコードから生存期間解析を行うことで求めた「各タスクの現在の状態」と「各タスクの現在滞在中の状態と次に遷移する可能性のある状態それぞれのメモリ使用量」を入力データとしている．これらの入力データから，どのタスクに遷移すればどれ程メモリ使用量が増減するかを調べる．これにより，出力データとして次に遷移させてもいいタスク群を出力する．

まず実行前の静的解析として，各タスクのプログラムに対して事前に変数の生存期間解析[5]を行う．各タスクは分岐や繰り返し構造を持たず，かつ，独立，すなわちタスク間同期・通信がないという前提により，この解析は容易に行える．これにより，当該タスクのメモリ使用量の時系列変化をあらかじめ求めておく．次に，マルチタスクシステム実行時に，各タスクの現在の状態から，各タスクの状態の組合せを構成する．この現在の状態からどのタスクを次の状態に遷移させればどの程度メモリ使用量が増加するかを比較し，最もメモリ使用量の増分が小さいタスクを選択しスケジュールを行う．もし増分が等しいタスクが複数存在する場合，次にスケジュールするタスクの決定は OS のスケジューラに委ねる．初期状態から最終状態まで全ての状態を一通り見てから，メモリ使用量の少ない経路をスケジュールする最適化方法では，処理に時間がかかる．そのため，現在探索している状態に隣接する範囲内で，メモリ使用量の少ない状態を選択するスケジューリング手法を用いる．

今回の提案手法では，各タスクの各状態におけるメモリ使用量と各タスクの現在の状態の組から構成された現在のマルチタスクシステム全体の状態を入力としている．これらの入力データを読込んで，次にスケジュールするタスクを出力として求める．

これらの入力データから出力を求めるためのアルゴリズムを図 2 に示す．

入力：各タスクの現在の状態の組(s1,...,sn)
　　各状態 si と si の次の状態 next(si)それぞれにおけるメモリ使用量
　　mem(si), mem(next(si))　(i = 1...,,n)
出力：次にスケジュールさせても良いタスクの集合 Sched_Set ⊆{1,...,n}

1. (s1,...,sn)を入力する
2. 各 si (i = 1,...,n)に対して, mem(si),mem(next(si))を入力する
3. 変数 minevp に値+∞を代入
4. i∈{1,...,n}に対して 4.1.から 4.2.までの処理を行う
   4.1. もし si がタスク i の最終状態でないならば,
      4.1.1. evp(i):=mem(next(si))-mem(si)
      さもなければ
      4.1.2. evp(i):=+∞
   4.2. もし evp(i)<minevp ならば minevp:=evp(i)
5. もし，任意の i∈{1,...,n}に対して evp(i) == +∞ となるならば,
   5.1. 全タスクが最終状態に到達したものとみなし，Sched_Set:= φ を出力
6. Sched_Set:={i | evp(i) == minevp}として，Sched_Set を出力する

図 2：提案手法によるマルチタスクスケジューリングのアルゴリズム

Fig.2 Proposed Memory-aware Multitask Scheduling Algorithm

図 2 のアルゴリズムにおいて，各タスク i においての現在の状態 si から次の状態 next(si)に進めた場合のメモリ使用量の増分 mem(next(si))-mem(si)をタスク i の評価値 evp(i)とする．各 i∈{1,...,n}に対して evp(i)の最小値を minevp とする．また，処理が終了したタスクの場合，そのタスク内で次に遷移する状態が存在し

ないので，評価値は求められない．従って，最終状態に到達したことを意味する評価値として+∞を代入する．

最小の増分を持つタスクに遷移するこのアルゴリズムでは, evp(i)=+∞であるようなタスク i は決してスケジュールされない．図 2 の 4.の処理は，各タスク i の評価値 evp(i)を求め，それらの最小値を minevp に求めている．ただし，最終状態に到達したタスク j については evp(j)=+∞として，決してタスク j がスケジュールされないようにしている．また，5.の処理では，全てのタスクの評価値 evp(i)の値が+∞になったことは，全てのタスクが最終状態に達したことを意味するため，スケジュールすべきタスク集合 Sched_Set を空集合として出力する．さもなければ， 6.の処理で最小の評価値 minevp と等しい評価値 evp(i)を持つタスク i の集合をスケジュールすべきタスク集合 Sched_Set として出力する．

図 2 のアルゴリズムをタスクスケジューラの起動周期毎に実行されるようにし, OS のタスクスケジューラは Sched_Set の集合からのみタスクを選択してスケジュールを行うことにより，ヒープメモリ使用量をできるだけ抑えたマルチタスクスケジューリングを実現できる．

することで, 各タスクの処理を逐次的に実行させることができる．これにより，初期状態から最終状態に到達するまでのマルチタスクスケジューリングを行うことができる．

この提案手法の適用例として図 3 のタスク集合を考える．図 3 は，図中の Task1 と Task2 の 2 つのタスクを用いて，マルチタスクシステム実行時の総メモリ使用量の変化を示した図である．図中のマルチタスクシステムの矢印は，左向きは Task1，右向きは Task2 がスケジュールされた場合の状態遷移を表す．図中の数字は，その状態のメモリ使用量の合計を示す．

まず，提案手法を導入しない場合を考える．任意のスケジューリングが行われるとの仮定の下で，図 3 のマルチタスクシステムを実行すると，初期状態から最終状態に到達するまでの状態遷移の経路には任意の可能性があり，その中には最悪メモリ使用量 9 を持つ状態に遷移する経路が存在する．その場合のスケジュール例を図 4 に示す．なお，図 4 では，初期状態から最終状態に到達するまでに最悪メモリ使用量を持つ状態を経由する経路を，実線で示している．

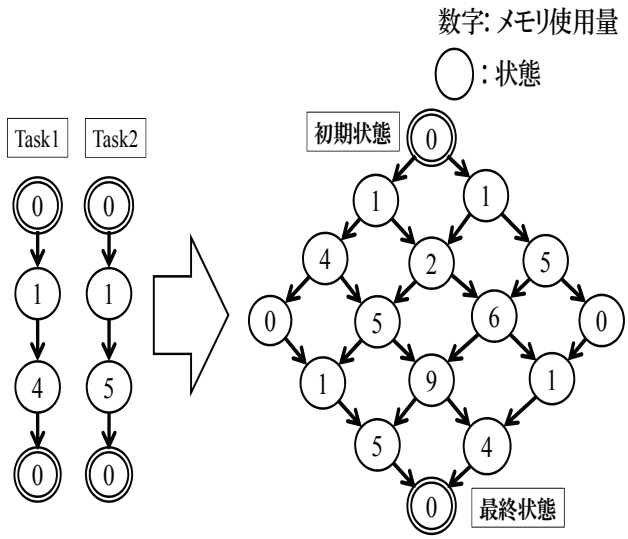

図 3：2 つのタスクの並行実行例

Fig.3 Example of Concurrent Execution of Two Tasks

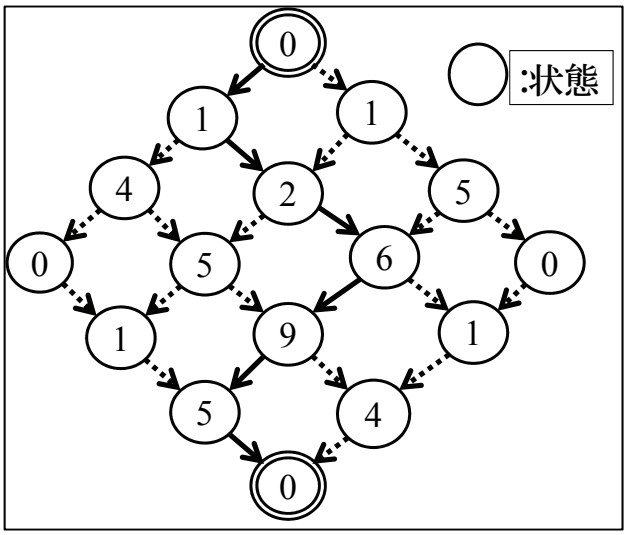

図 4：提案手法導入前の並行実行例

Fig.4 Example of Concurrent Execution in Arbitrary Scheduling

図 4 で示すとおり，提案手法を導入していない場合は, (0,1,2,6,9,5,0)と最大のメモリ使用量 9 を通る可能性を持つ．

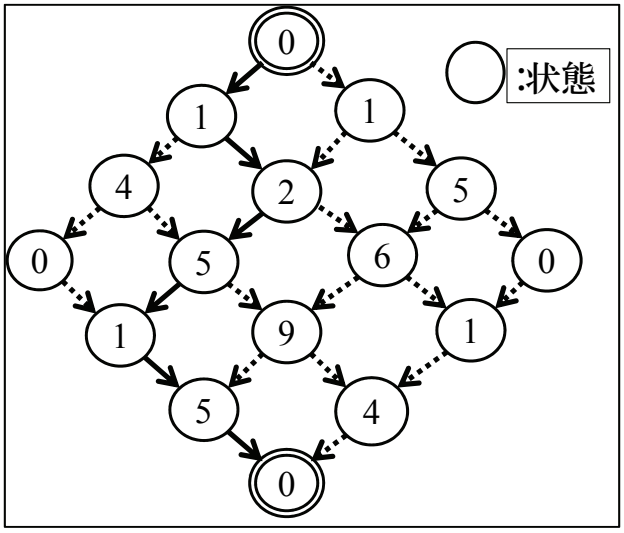

図 5：提案手法導入後の並行実行例

Fig.5 Example of Concurrent Execution in Proposed Memory-aware Scheduling

次に，提案手法を導入して，スケジューリングを行った場合の状態遷移図を図 5 に示す.

提案手法を導入することで，初期状態から最終状態に到達するまでの間に，常にメモリ使用量の増分がより小さい次の状態を選択する．提案手法を導入した図 5 を見ると，このマルチタスクシステムで最大のメモリ使用量が 9 である状態は通らず, (0,1,2,5,1,5,0)の経路を通る．これにより，メモリ使用量 5 を持つ状態が最もメモリを使用する状態となることが分かる.

4. 評価実験

4.1 実験目的

今回の実験では，入力データであるタスクからのメモリ使用量の時系列変化を読込んで，各タスクの並行動作をランダムでシミュレーションし，提案手法を用いた場合と用いない場合のそれぞれについて，最悪メモリ使用量を求め，メモリ削減率の評価を行うことを目的とする．また，各タスクの最悪 CPU 獲得待ち時間の測定も行い，応答性の評価も行う．なお，最悪 CPU 獲得待ち時間とは，各タスクにおいて連続で他のタスクが状態遷移した回数(すなわち, 当該タスクが連続でスケジュールされなかった回数)の最大値を表す.

4.2 実験方法

この評価実験では，入力データとして，実際にプログラムコードが書かれたタスクを用いる代わりに各タスクに変数の定義・使用のタイミングをランダムに生成したデータを入力として用いた．このデータをマルチタスクシステムとして扱うことで，複数のタスクの状態を組合せる. このとき発生する複数の状態の中で，最悪メモリ使用量を持つ状態を選択しないようにすることで，メモリ使用量を抑える.

マルチタスクシステム上で，初期状態から最終状態に到達するまでの状態をランダムに選択し，1 本の経路を作成する．その経路内の最悪メモリ使用量を記憶する．この方法を 1 つの入力データに対し，一定回繰り返し，その回数の中で最悪メモリ使用量を測定結果として用いる．今回の評価実験では，1 つのデータに対する繰り返し回数を 1 万回に設定する. 同時に, CPU 獲得待ち時間の測定を行い，各タスクの応答時間の評価も行う.

今回の実験では，タスクを 3 つ用いた場合，4 つ用いた場合, 5 つ用いた場合で測定し, 1 タスクあたりの状態数を変化させた場合で測定を行う. この測定では, 入力データをそれぞれの状態数に対し 10 個用意し, その平均値を求める．その平均値を用いて，提案手法の評価を行う．入力したデータに対する提案手法導入前と導入後の最悪メモリ使用量と各タスクの CPU 獲得待ち時間の最悪値を測定する.

この提案手法の評価実験を行うために用いた CPU・OS などの実験環境は下記の通りである：

- ハードウェア機器：HP xw8400 Workstation
- CPU：Intel(R) Xeon(R) CPU X5355 2.66GHz
- OS: Windows XP Professional x64
- メモリ: 8GB
- コンパイラ: Cygwin gcc version 3.4.4

4.3 実験結果

実験結果である提案手法導入前後の最悪メモリ使用量の測定値から，提案手法によるメモリ削減率を求める. また, 提案手法導入後の CPU 獲得待ち時間の最悪値は，提案手法導入前のものよりも長くなっているので, 提案手法による CPU 獲得待ち時間増加率として結果をまとめる. メモリ削減率と CPU 獲得待ち時間増加率の評価結果をグラフとしてまとめたものを図 6 に示す.

図 6 より，1 タスクあたりの状態数を増やすとメモリ削減率は下がり，タスクの数を増やすと削減率は上がる一方，1 タスクあたりの状態数が小さい場合は CPU 獲得待ち時間増加率はさほど上昇しないという結果が得られた．1 タスクあたりの状態数を増やすとメモリ使用量が増したのは，各タスクの変数の生存期間が一般に長くなるため，メモリ使用量を長く確保したためと考えられる．また，タスクを増やすとメモリ削減率が増したのは，遷移する状態の経路が増えるため，メモリ消費量の少ない状態を選択する可能性が増したためと考えられる.

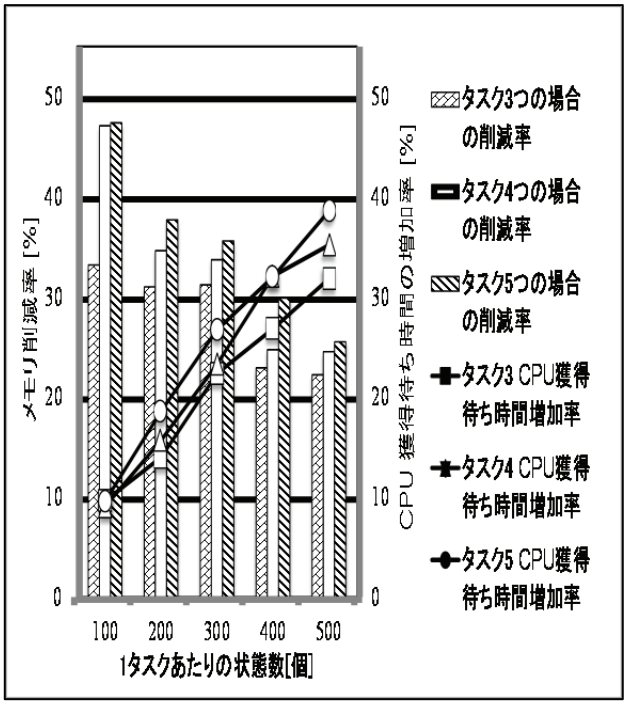

図 6 : 提案手法によるメモリ削減率及び CPU 獲得待ち時間増加率

Fig.6 Memory Reduction Rate and Increase Rate of Wait Time for CPU Allocation by the Proposed Method

1 タスク辺りの状態数が増えると CPU 獲得待ち時間が増したのは，提案手法により，他のタスクが使用している変数の生存期間が長くなるため，CPU 獲得待ち時間が長くなってしまったためと考えられる．

従って，この提案手法は，状態数の小さいタスクを多数並行動作させた場合は，応答性をそれほど犠牲にせず大きくメモリ使用量を削減率できるという結果となった．

5. あとがき

本研究では，入力データであるタスクから各変数の生存期間解析を行い，その変数にランダムでメモリ使用量を割り当てた．これにより，各状態で消費するメモリ使用量を求め，そのデータを使いマルチタスクシステムを実行した．そのマルチタスクシステム実行時に発生する複数の状態のうち，メモリ消費を抑えられる状態を選択する．それにより，最悪メモリ使用量を持つ状態を避けるようにスケジューリングを行うことでメモリ使用量を抑える手法を提案した．

評価実験の結果により，1 タスク当たりの状態数を増やすと，メモリ削減率は低下し，タスクの数を多く使用すると，メモリ削減率が向上するという結果となった．しかし，各タスクの CPU 獲得待ち時間は，状態数が増加するにつれ，長くなり応答性が悪化するという結果となった．これにより，本研究で提案した手法は，状態数の小さいタスクを多数並行動作させるシステムに対し有効であることが言える．

今後の課題としては，CPU の獲得待ち時間を短くし，悪化したタスクの応答性をある程度改善させるように手法を改良することが考えられる．また，今回の研究では，変数の“定義”や“使用”のデータのみをランダムに記載した擬似的なタスクを入力データとして用いたため，定義，使用のパターンが実際のタスクを反映していない可能性がある．なので，実際にプログラムのソースコードで記載されたタスクから変数の生存期間解析を行うことで，メモリ使用量の変化のパターンがどのようなものかを調べ，この提案手法が，実際のメモリ使用量の変化パターンにおいて有効なのかどうかを調べる必要がある．本研究では，分岐や繰り返し構造がないタスクのみを対象として，変数の生存期間から各タスクが消費するメモリ使用量の時系列変化を求めたが，一般に分岐や繰り返し構造を持つプログラムの変数生存期間解析は困難であるため，malloc 関数や free 関数などによるヒープメモリの動的確保・解放を実行時に観測し，予測するなど，別の方法によるメモリ使用量解析の可能性も検討する必要がある．

文　献

[1] 阪田史郎，高田広章,”組込みシステム”，株式会社オーム社，2006.

[2] N. Vasudevan, S.Edwards: ”Buffer Sharing in CSP-like Programs”, Proc. of the 7th IEEE/ACM Int. Conf. on Formal Methods and Models for Codesign (MEMOCODE 2009),pp.151-160,2009.

[3] B. Ristau and G. Fettweis, “An Optimization Methodology for Memory Allocation and Task Scheduling in SoCs via Linear Programming” “, Proc. oOf 6th Int. Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS 2006), Lecture Notes in Computer Science 4017, Springer-Verlag, pp.89-98, 2006.”

[4] S.A.Edwards and O.Tardieu. SHIM: A deterministic model for heterogeneous embeded systems. In Proc. of the Int. Conf. on Embedded Software (EMSOFT 2005),pp. 37-44,2005.

[5] L.D.Fosdick and L.J.Osterweil, “Data Flow Analysis In Software Reliability”, ACM Computing Surveys, Vol. 8, No. 3, pp.305-330, 1976.