Vol.2021-ARC-244 No.11
Vol.2021-SLDM-194 No.11
Vol.2021-EMB-56 No.11
2021/3/25

H/W リソース枯渇要因分析自動化手法の提案

森田清隆[†1]　桐村昌行[†1]　水口武尚[†1]

概要：組込み製品の性能を阻害する要因として，H/W リソースの枯渇による S/W 処理の遅延が挙げられる．この解決に H/W リソース枯渇の要因となる処理の分析が必要となるが，分析に必要なデータの測定は，その処理自体の処理負荷による測定結果への影響や測定したデータの記録に伴う記憶容量の圧迫が起こり得る課題がある．また，測定すべきデータの判断や測定したデータの分析に労力が必要であり，特に H/W リソース枯渇の要因をアプリケーションの関数やソースコード箇所まで分析する場合の労力は大きい．これに対して測定結果への影響や記憶容量消費を抑えた上で，H/W リソース（CPU，メモリ，ストレージ I/O）の枯渇を検知し，その枯渇要因となるアプリケーションの関数およびソースコード箇所を自動的に分析する手法を提案する．

提案手法は，H/W リソースが枯渇する範囲における H/W リソース使用処理に限定して測定を行うことで，測定処理負荷による測定結果への影響や測定データによる記憶容量消費を抑える．また，H/W リソースが枯渇する範囲において対象 H/W リソースを使用する処理をアプリケーションの関数およびソースコード箇所まで分析し，H/W リソースの枯渇要因として示すことができる．評価において，提案手法により CPU，メモリ，ストレージ I/O に関して H/W リソース枯渇を生じさせるアプリケーションの関数およびソースコード箇所を，H/W リソースの枯渇要因として提示できることを確認した．

キーワード：性能分析，H/W リソース，パフォーマンスモニタリング，プロファイリング，トレーシング，CPU，メモリ，ストレージ I/O，Linux

1. はじめに

組込み製品の性能を阻害する要因として，S/W の処理負荷による H/W リソースの枯渇が挙げられる．H/W リソースの枯渇とは，CPU などの H/W リソースに対する使用要求が処理可能な量よりも多く，H/W リソースの空きを待つ処理が生じている状態を指す．H/W リソースが枯渇している状態では，H/W リソースの空きを待つ時間分だけ，処理時間が長くなり，製品の性能阻害につながる．

このような H/W リソース枯渇による性能阻害の解決には，より高い性能の H/W を用いる対策が考えられるが，製品コストの増大につながるため安易にこの対策はとれない．そのため，S/W の改善による解決が求められ，H/W リソース枯渇の要因となる処理の分析が必要となる．

一方，この要因分析に必要なデータを取得する測定処理は，その処理自体の処理負荷による測定結果への影響や測定データの記録に伴う記憶容量の圧迫が課題となる．特に，H/W リソースが制約される組込み機器では，これらの課題が顕著となる．また，H/W リソース枯渇の要因分析には，測定すべきデータの判断や測定したデータの分析に労力が必要となる．特に，アプリケーションの関数やソースコード箇所まで要因を分析する場合，プロセス単位や S/W の低レイヤに閉じた分析に比べて労力は大きくなる．

これに対して本研究では，測定結果への影響や記憶容量消費を抑えた上で，H/W リソース枯渇要因となるアプリケーションの関数およびソースコード箇所を自動的に分析する手法を提案する．

2. 関連技術

本研究の関連技術について示す．

2.1 パフォーマンスモニタリング

パフォーマンスモニタリングは，H/W リソースの使用状態を監視するものであり，H/W リソースが枯渇しているかを分析するための情報を得られる．Linux では top[1]などのツールの実行が該当する．

ただし，H/W リソース使用の内訳として得られる情報は一般的にプロセス単位であり，この情報だけでは要因となっている処理を詳細に分析するのには不十分である．

2.2 プロファイリング・トレーシング

プロファイリング・トレーシングは，サンプリング周期やイベント毎にプログラム実行時の各種情報を収集することができる．例えば，サンプリング周期毎の CPU を使用した関数情報や特定のシステムコール実行時の引数・戻り値情報，関数コールスタック情報，などを収集できる．これらの情報は，H/W リソースを使用する処理の詳細な分析に役立てることができる．Linux では perf[2]などのツールの実行が該当する．

一方，これらの測定処理はパフォーマンスモニタリングに比べ，多くの H/W リソースを使用する．このため，測定処理自体の H/W リソース使用が測定結果に影響を与える．特に広い範囲を対象とした測定の場合は，その影響が大きくなることに加えて，測定データの出力が記憶容量を圧迫する．これに対して，測定の対象範囲を限定的にすることが検討できるが，その対象範囲の絞り込みに労力が必要で

†1 三菱電機株式会社　情報技術総合研究所
Information Technology R&D Center, Mitsubishi Electric Corporation

Vol.2021-ARC-244 No.11
Vol.2021-SLDM-194 No.11
Vol.2021-EMB-56 No.11
2021/3/25

ある．

また，対象の H/W リソースを CPU，メモリ，ストレージ I/O として，各 H/W リソースの枯渇要因となる H/W リソース使用をアプリケーションの関数やソースコード箇所まで分析する場合，既存のツールだけでは人手による分析作業が必要となる．既存のツールとして，各関数の実行時間を関数の呼び出し関係とともにグラフに出力できるもの[3][4]はあり，この出力と CPU 使用率などのパフォーマンスモニタリング情報とを比較することで H/W リソースの枯渇要因となる関数の見当を付けることはできる．しかし，要因をソースコードレベルで詳細に分析するためには別の分析が必要となる．他のツールとして，メモリリークを起こすソースコード箇所を分析できるもの[5]はあり，このようなツールによりメモリリークによるメモリの枯渇要因となる箇所が分かる．しかし，メモリリークに起因しないメモリの枯渇や他の H/W リソースの枯渇に関してはその要因を分析できない．

3. 提案手法

測定結果への影響や記憶容量消費を抑えた上で，H/W リソース枯渇要因となるアプリケーションの関数およびソースコード箇所を自動的に分析する手法を提案する．なお，提案手法の対象とする H/W リソースは CPU，メモリ，ストレージ I/O とする．また，本提案手法は OS が Linux であり，実行するプログラムの記述言語が C/C++である環境を対象に検討したものである．

3.1 概要

提案手法は，H/W リソースの枯渇を検知してその範囲（タイミング）の抽出を行う**(A)H/W リソース枯渇検知・範囲抽出**と，抽出した範囲における H/W リソース使用処理をアプリケーションの関数およびソースコード箇所まで分析する**(B)H/W リソース使用処理分析**の 2 段階の処理で構成する（図 1）．上記(A)(B)において必要なデータを取得する測定処理は独立して実施し，(A)の処理結果から(B)の測定の対象範囲を限定する．各測定は同条件で再現されたターゲットの実行に対して行うものとする．そのため，ターゲットに対して同条件の実行を繰り返し行えるテストの用意は前提となる．

提案手法では，H/W リソースが枯渇する範囲における H/W リソース使用処理をアプリケーションの関数およびソースコード箇所単位で分析し，その単位ごとに H/W リソース使用量を示す．これにより，H/W リソース枯渇の要因を提示する．また，H/W リソースの枯渇範囲における H/W リソース使用に限定して測定することにより，測定処理負荷による測定結果への影響や測定データによる記憶容量消費を抑える．

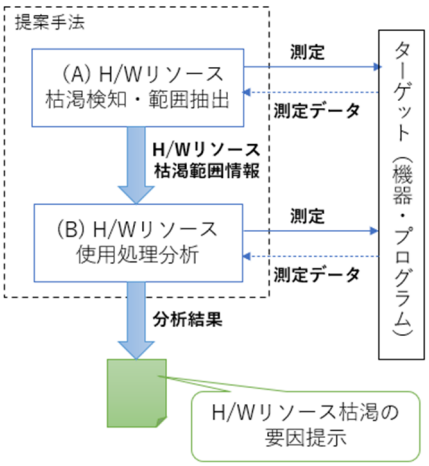

図 1　提案手法の構成

以降の 3.2，3.3 には，(A)H/W リソース枯渇検知・範囲抽出，(B)H/W リソース使用処理分析，のそれぞれの処理の詳細を示す．

3.2 （A）H/W リソース枯渇検知・範囲抽出

H/W リソース枯渇の検知および範囲の抽出を行う処理手順を次に示す．

① **ターゲットの実行開始**
② **基準とするイベントの実行時刻と各 H/W リソースの枯渇の判定に用いるメトリクス情報を取得時刻とともに測定**
③ **手順②で得た測定データとメトリクスに対する閾値から H/W リソース枯渇の検知・範囲抽出**

手順①において測定の対象とするターゲットのテスト実行を開始し，その実行に対して手順②の測定処理を行う．手順②では，CPU，メモリ，ストレージ I/O の各 H/W リソースの枯渇を判定するためのメトリクス情報をパフォーマンスモニタリングにより取得する．手順③では，手順②で取得するメトリクス情報が閾値を超える場合に H/W リソース枯渇が生じたと判定する．この閾値については一概に値を定義できないが，各メトリクスに応じてあらかじめ設定するものとする．また，手順③では H/W リソース枯渇が生じたと判定した場合，手順②で取得する時刻情報から，H/W リソースの枯渇が生じる開始時刻から終了時刻までの時間帯（以降，**H/W リソース枯渇範囲**）を抽出する．この H/W リソース枯渇範囲は，基準とするイベント実行からの相対時刻として抽出する．基準とするイベントは，ターゲットに対するテスト実行に含まれるものであり，OS の起動やプログラムの起動，ターゲットに対する操作などを対象とする．

本手法で用いるメトリクスの例を対象 H/W リソースごとに表 1 に示す．本稿では，使用するメトリクスは限定しないが，単一のメトリクスよりも表 1 に示すような複数の

Vol.2021-ARC-244 No.11
Vol.2021-SLDM-194 No.11
Vol.2021-EMB-56 No.11
2021/3/25

表 1　H/W リソース枯渇判断に用いるメトリクス

対象 H/W リソース	メトリクス
CPU	CPU 使用率，ランキュー数
メモリ	メモリ空き容量，スワップアウト発生
ストレージ I/O	デバイス使用率，I/O キュー数，I/O 応答待ち時間

メトリクスを用いて H/W リソース枯渇の判定精度を高めることが検討できる．例えば，CPU リソースの枯渇を判定する場合に CPU 使用率だけを用いると，CPU 使用率が高ければ待ちプロセスがなくても CPU リソースの枯渇状態と判定してしまう．しかし，このような状態は枯渇状態ではなく CPU を有効に活用している状態だといえる．これに対して，待ちプロセス数を示すランキュー数と合わせて判定することにより判定精度を高められる．

複数のメトリクスを用いる場合は，各メトリクスに対して閾値を超える範囲の AND 条件や OR 条件で，H/W リソース枯渇範囲を抽出する．図 2 に対象 H/W リソースが CPU の場合に，CPU 使用率とランキュー数の閾値を超える範囲の AND 条件により，H/W リソース枯渇範囲を抽出している例を示す．この例では，抽出した H/W リソース枯渇範囲が[6.5 - 8.1]（秒）の範囲であることを示す．

抽出した H/W リソース枯渇範囲の情報は次段階の(B)H/W リソース使用処理分析の測定における対象範囲の限定に用いる．このため，H/W リソース枯渇範囲は毎回のターゲットの実行で再現性がある範囲に限定した方が要因分析の精度を高められる．これに対して，上記の手順を複数回行って，毎回 H/W リソース枯渇範囲となる範囲を再現性のある H/W リソース枯渇範囲として抽出する．

3.3　（B）H/W リソース使用処理分析

3.2 の手法において枯渇が検知された H/W リソースに対して，その要因となる H/W リソース使用処理を分析する処理について示す．H/W リソースの種別によって処理が異なる部分があるため，3.3.1 に本手法の全体的な処理内容について示し，3.3.2～3.3.4 に CPU，メモリ，ストレージ I/O の H/W リソース種別ごとに異なる処理内容を示す．

3.3.1 H/W リソース使用処理の分析

H/W リソース枯渇範囲における H/W リソース使用処理を分析する処理の手順を以下の(1)～(5)に示す．この手順により，H/W リソース枯渇要因となるアプリケーションの関数およびソースコード箇所を出力する．

(1)　リソース使用データ測定

H/W リソース枯渇範囲における H/W リソース使用に限定したプロファイリング・トレーシングの測定処理を行う．

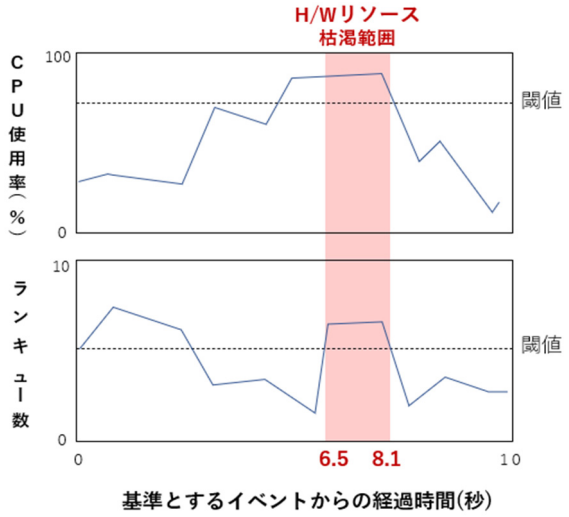

図 2　H/W リソース枯渇範囲抽出

この測定は H/W リソース枯渇範囲情報を用いて，次に示す処理手順で行う．

①　ターゲットの実行開始

②　H/W リソース枯渇範囲に基づいたタイミングで測定処理を実行

手順①は 3.2 に示した手順におけるターゲットの実行と同条件の実行を行う．手順②では，H/W リソース枯渇範囲情報をもとに測定処理の開始と終了を行う．ただし，H/W リソース枯渇範囲に対して，どのタイミングで測定を開始・終了するかは対象の H/W リソース種別によって異なるため，このタイミングについては 3.3.2～3.3.4 に H/W リソース種別ごとに示す．

測定処理では，対象の H/W リソース枯渇に関わる H/W リソース使用処理毎の関数コールスタック情報と，H/W リソース使用量の算出に用いる情報を取得する．関数コールスタック情報には，コールスタック内の関数ごとに関数名，実行アドレス，実行ファイルパス情報が少なくとも含まれるものとする．図 3 に例として，ベンチマークソフトである UnixBench[6]の実行に対して取得した関数コールスタックを示す．なお，以降でもこのプログラムを例に手順を示す．

関数コールスタック情報の取得対象や H/W リソース使用量の算出に用いる情報については対象 H/W リソースによって異なるため，この詳細は 3.3.2～3.3.4 に H/W リソース種別ごとに示す．

(2)　リソース使用関数情報抽出

(1)で取得した H/W リソース使用処理毎の関数コールスタック情報から H/W リソースを使用した処理の関数（以降，**実処理関数**）とその関数を呼び出したアプリケーション内の関数（以降，**アプリケーション関数**）を抽出する（図 4）．実処理関数については，関数コールスタックの先頭から抽出する．アプリケーション関数については，関数コー

Vol.2021-ARC-244 No.11
Vol.2021-SLDM-194 No.11
Vol.2021-EMB-56 No.11
2021/3/25

図 3　関数コールスタック例

図 4　アプリケーション関数抽出例

ルスタック情報に含まれる実行ファイルパス情報からアプリケーションの関数かどうかを判別して抽出する．このため，対象とするアプリケーションの格納フォルダのパスをあらかじめ指定することとする．

なお，関数コールスタック内に対象となるアプリケーション関数が複数ある場合，どの関数を抽出対象にするかは検討の余地があるが，本稿では関数コールスタック内で一番浅い階層のものを対象とする．

(3) 関数単位リソース使用量算出

(2)で抽出した実処理関数とアプリケーション関数について，各関数単位で対象 H/W リソースの使用量を算出する．アプリケーション関数単位での算出は，どのアプリケーション関数が H/W リソース枯渇範囲において H/W リソースを多く使用しているかを示すためのものである．実処理関数単位での算出は，(4)で分析する実処理関数を呼び出すアプリケーション関数のソースコード箇所と結び付けることにより，どのアプリケーションのソースコード箇所が H/W リソース枯渇範囲において H/W リソースを多く使用しているかを示すためのものである．実処理関数単位の H/W リソース使用量については，アプリケーションのソースコード箇所ごとに対応させる必要があるため，実処理関数ごとに分けることに加えて，呼び出し元のアプリケーション関数の実行アドレス別に分けて算出するものとする．

H/W リソース使用量の算出方法に関しては，3.3.2～3.3.4 に H/W リソース種別ごとに示す．

(4) 処理ソースコード分析

H/W リソースを使用するアプリケーション関数内のソースコード箇所を抽出するため，実処理関数を呼び出すアプリケーション関数内のソースコード箇所を分析する．この分析の処理手順を次に示す．この手順では，実行アドレスとソースコード箇所の対応関係を参照できる対象アプリケーションの逆アセンブリコードを生成し，その逆アセンブリコードを用いて実処理関数を呼び出すアプリケーション関数の実行アドレスに該当するソースコード箇所を抽出する．

① 関数コールスタック情報から対象アプリケーション関数に対する実行アドレスと実行ファイルパスを抽出する（図 5）

② 手順①で抽出した実行ファイルパス情報に対応する実行ファイルを，該当ソースファイルパス・行番号情報付きで逆アセンブルする

③ 手順②で生成した逆アセンブリコードから，手順①で抽出した実行アドレスにマッチするアセンブリコード行を検索し，そのアセンブリコード行に該当するソースファイルパスと行番号情報を抽出する（図 6）

④ 手順③で抽出したソースファイルパスと行番号情報から，該当ソースコード箇所を抽出する（図 7）

(5) 結果出力

分析結果として，H/W リソース枯渇範囲において H/W リソースを使用したアプリケーションの関数およびソースコード箇所を H/W ソース使用量とともに示す．(1)~(4)で得た情報をもとに，以下の情報を少なくとも出力する．

- アプリケーション関数単位の H/W リソース使用量
- 実処理関数単位の H/W リソース使用量
- 実処理関数とその関数を呼び出したアプリケーション関数ソースコードとの対応

例えば，図 8 のような表形式で出力する．このような出力から，H/W リソース枯渇範囲におけるアプリケーション関数ごとの H/W リソース使用量が分かるので，どのアプリケーション関数が H/W リソース枯渇の要因となっているかが分かる．また，その内訳として，アプリケーション関数内で呼び出した H/W リソースを使用する実処理関数のリソース使用量と対応するソースコード箇所が分かり，アプリケーション内のどのソースコードが H/W リソース枯渇の要因となっているかが分かる．

3.3.2 CPU 使用処理の分析

CPU については，H/W リソース枯渇範囲において CPU を使用しているアプリケーション関数を分析すればよい．3.3.1 に示した処理の内で対象 H/W リソースが CPU の場合に個別の処理となる，(1)リソース使用データ測定，(3)関数単位リソース使用量算出，について示す．

(1) リソース使用データ測定

H/W リソース枯渇範囲に限定して測定処理（プロファイリング）を行い，サンプリング毎に CPU を使用している処理の関数コールスタック情報を取得する．この関数コールスタック情報が 3.3.1 (2)以降の処理で対象となる H/W リソース使用処理毎の関数コールスタック情報となる．

Vol.2021-ARC-244 No.11
Vol.2021-SLDM-194 No.11
Vol.2021-EMB-56 No.11
2021/3/25

```
7fbf3eb7829c __GI___strcmp_ssse3 (/lib/x86_64-linux-gnu/libc-2.19.so)
400d22 Func_2 (/home/melco/app/UnixBench/pgms/dhry2reg)
400830 main (/home/melco/app/UnixBench/pgms/dhry2reg)
7fbf3ea59ec5 __libc_start_main (/lib/x86_64-linux-gnu/libc-2.19.so)
40096c _start (/home/melco/app/UnixBench/pgms/dhry2reg)
 0 [unknown] ([unknown])
```

図 5　実行アドレス・実行ファイルパス抽出

```
/home/melco/app/UnixBench/src/dhry_2.c:181

 400d1d: e8 ce f8 ff ff        callq  4005f0 <strcmp@plt>
 400d22: 85 c0                 test   %eax,%eax
 400d24: 7e 22                 jle    400d48 <Func_2+0x38>
/home/melco/app/UnixBench/src/dhry_2.c:185
・・・
```

図 6　ソースファイルパス・行番号情報抽出例

```
181       if (strcmp (Str_1_Par_Ref, Str_2_Par_Ref) > 0)
182         /* then, not executed */
183       {
184         Int_Loc += 7;
185         Int_Glob = Int_Loc;
```

図 7　該当ソースコード箇所抽出例

アプリケーション関数	H/W リソース使用量		実処理関数	ソースコード箇所
	全体	実処理関数内訳		
app_func1	50			
		30	lib_funcA	app.c:13
		20	lib_funcB	app.c:47
app_func2	25			
…	…	…	…	…

図 8　結果出力イメージ

(3) **関数単位リソース使用量算出**

サンプリング毎の関数コールスタック情報における実処理関数とアプリケーション関数の出現数をそれぞれ集計し，この出現数を全体のサンプリング数で割る．これにより，実処理関数単位とアプリケーション関数単位の H/W リソース枯渇範囲における CPU 使用割合を H/W リソース使用量として算出する．

3.3.3 メモリ使用処理の分析

H/W リソース枯渇範囲におけるメモリの枯渇要因を分析するためには，メモリ枯渇時点（H/W リソース枯渇範囲の開始時刻）で使用されているメモリ領域を確保したアプリケーション関数について調べる必要がある．これを調べるためには，H/W リソース枯渇範囲に至るまでの以下の情報を時系列に収集する必要がある．

(a) メモリ領域の確保を要求したアプリケーション関数
(b) (a)の要求により確保されたメモリ領域
(c) (b)のメモリ領域に対する解放処理の有無

上記(a)の情報は，メモリ領域を確保するシステムコール実行に対する関数コールスタック情報から得られる．上記(b)の情報は，メモリ領域を確保するシステムコールの引数・戻り値から得られる．上記(c)の情報は，メモリ領域を解放するシステムコールの引数・戻り値から得られる．これらの情報を分析することで，メモリ枯渇時点において確保後未開放のメモリ領域に対して，その領域のサイズと確保したアプリケーション関数を分析できる．Linux においては，ユーザプログラムからの動的なメモリ領域の確保，解放には brk, mmap, munmap システムコールが関わる．brk はヒープ領域の拡張・縮小を行い，mmap, munmap はそれぞれメモリマップ領域の確保・解放を行うものである（図 9）．

以上を踏まえて，3.3.1 に示した処理の内で対象 H/W リソースがメモリの場合に個別の処理となる，(1)リソース使用データ測定，(2)リソース使用関数情報抽出，(3)関数単位リソース使用量算出，について示す．

(1) **リソース使用データ測定**

H/W リソース枯渇範囲の開始時刻に至るまでのメモリ領域の確保・解放を行うシステムコール（brk, mmap, munmap）に対する測定処理（トレーシング）を行い，対象システムコール実行毎の引数・戻り値の情報を取得する．メモリ領域の確保を行うシステムコールに関しては，システムコール実行時の関数コールスタック情報も取得する．

(2) **リソース使用関数情報抽出**

(1)で得られたデータに対して次の処理を行うことにより，H/W リソース枯渇範囲の開始時刻で使用されているメモリ領域（確保後未開放のメモリ領域）とその領域を確保したシステムコール実行時の関数コールスタックを抽出する．

① メモリ領域の確保・解放を行うシステムコール（brk, mmap, munmap）の引数・戻り値を時系列に分析し，確保後未解放のメモリ領域（アドレス，サイズ）を分析する
② 手順①で分析したメモリ領域を直近で確保したシステムコール（brk, mmap）実行に対する関数コールスタック情報を抽出する

上記で抽出した関数コールスタック情報を H/W リソース使用処理毎の関数コールスタック情報として，3.3.1(2)の処理を行う．

(3) **関数単位リソース使用量算出**

上記(2)の手順①で分析した確保後未解放のメモリ領域のサイズ情報と，上記(2)の手順②で抽出した関数コールスタック情報から，各メモリ領域のサイズをそのメモリ領域を確保した実処理関数とその関数を呼び出したアプリケーション関数ごとに集計する．

Vol.2021-ARC-244 No.11
Vol.2021-SLDM-194 No.11
Vol.2021-EMB-56 No.11
2021/3/25

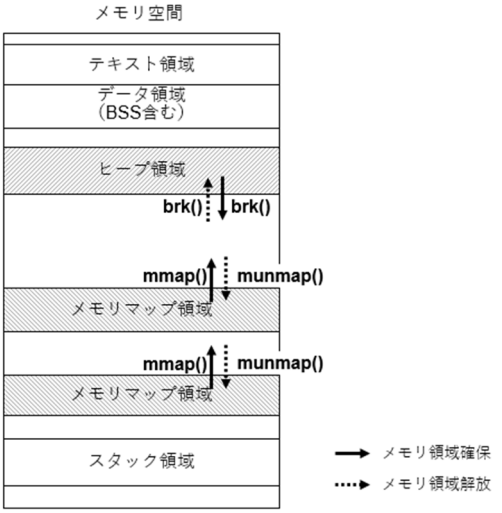

図 9　メモリ確保・解放に関わるシステムコール

3.3.4 ストレージ I/O 使用処理の分析

ストレージ I/O については，H/W リソース枯渇範囲およびその直前において，書き込み・読み出しのアクセス要求を行ったアプリケーション関数とそのアクセスデータサイズを分析する．ストレージのデータがメモリにキャッシュされることを考えると，アプリケーションからのアクセス要求がストレージ I/O の使用に直結するとは限らないが，H/W リソース枯渇範囲やその直前に行われたアクセス要求がストレージ I/O の枯渇に起因する可能性は高い．このため，H/W リソース枯渇範囲およびその直前に行われる書き込み・読み出しのアクセス要求を行うシステムコールに対して測定（トレーシング）を行い，その要求元のアプリケーション関数とアクセスデータサイズを分析する．

なお，アプリケーションからの書き込み・読み出しのアクセス要求以外に，メモリ枯渇によるスワップアウト・スワップインによりストレージへアクセスされる場合もあり，このアクセスがストレージ I/O の枯渇に起因する場合も考えられる．しかし，このような枯渇に対する分析は 3.3.3 に示したメモリの枯渇に対する分析の対象とし，本分析の対象外とする．

以上から，3.3.1 に示した処理の内で対象 H/W リソースがストレージ I/O の場合に個別の処理となる，(1)リソース使用データ測定，(3)関数単位リソース使用量算出，について示す．

(1) リソース使用データ測定

H/W リソース枯渇範囲の開始直前（H/W リソース枯渇範囲開始時刻の何秒前かをパラメータで指定）から H/W リソース枯渇範囲終了時刻までに限定して，書き込み・読み出しのアクセス要求を行うシステムコール（write, read）実行に対する測定処理（トレーシング）を行い，対象システムコール実行毎の各システムコールの引数・戻り値と関数コーススタック情報を取得する．この関数コールスタック情報が 3.3.1 (2)以降の処理で対象となる H/W リソース使用処理毎の関数コールスタック情報となる．

(3) 関数単位リソース使用量算出

対象システムコールの引数・戻り値からアクセスデータサイズを分析し，対応する関数コールスタック情報から，実処理関数およびアプリケーション関数ごとにアクセスデータサイズを集計する．

3.4 制約条件

本提案手法を用いるための制約としては，対象プログラムのデバッグ情報が必要なことがある．デバッグ情報は，本手法における関数コールスタック情報の取得やソースコード箇所の分析に必要である．また，測定処理（プロファイリング・トレーシング）を行うための対象環境におけるカーネルコンフィグ設定などは，必要に応じて行う必要がある．加えて，測定の対象とするターゲットの実行を同条件で再現できるテストが必要である．

4. 評価

提案手法をツールとして実装し，評価を行った．開発したツールは，ターゲット機器において測定処理を行い，開発 PC（またはサーバ）において分析処理を行う構成である．本評価におけるツールを実行したターゲット機器環境と開発 PC 環境を表 2 に示す．

本評価では，意図的に対象 H/W リソースに負荷を掛けるプログラムを実行して H/W リソース枯渇を生じさせ，その負荷を掛けた関数およびソースコード箇所を H/W リソースの枯渇要因として提示できるかを確認した．各 H/W リソースに対する負荷プログラムの概要は表 3 に示すもので，負荷プログラムを複数プロセス実行して検証した．評価はそれぞれのプログラム毎に行い，対象の H/W リソースの枯渇要因を提示できるかを確認した．

CPU に関して負荷を掛けた評価における，H/W リソース枯渇範囲の抽出とその範囲の CPU 使用処理の分析結果を図 10，図 11 に示す．図 10 はターゲットの実行における CPU 使用率（図上部）とランキュー数（図下部）の複数回の測定結果から CPU リソースが枯渇する範囲を抽出したものである．図 11 は図 10 の H/W リソース枯渇範囲に対して，その要因を分析した結果を示したもので，出力の各項目の内容は表 4 に示す．なお，この分析結果は H/W リソース使用量が大きい順に上から表示しており，図 11 は分析結果の最上部を抜粋したものである．また，今回は同名のアプリケーション関数でもプロセスが異なれば H/W リソース使用量の集計を分けて分析している．

図 11 から，プロセス pow1 の関数 calculate_pow が一番

Vol.2021-ARC-244 No.11
Vol.2021-SLDM-194 No.11
Vol.2021-EMB-56 No.11
2021/3/25

表 2　評価環境

	ターゲット機器 (R-Car Starter Kit Pro(M3)[7]を使用)	開発 PC
プロセッサ	ARM Cortex-A57 Dual	Intel Core i5-6500 3.20GHz
メモリ	2.0GB	16.0GB
OS	Linux（4.14.75）	Linux (4.4.0)

表 3　負荷プログラム概要

対象 H/W リソース	プログラム概要
CPU	標準ライブラリ関数 pow()でべき乗計算を繰り返し行い CPU に負荷を掛けるプログラム（ソースファイル pow.c）
メモリ	標準ライブラリ関数 malloc()でメモリ確保を繰り返し行いメモリに負荷を掛けるプログラム（ソースファイル mem_stress.c）
ストレージ I/O	標準ライブラリ関数 fprintf()で書き込みを繰り返し行いストレージ I/O に負荷を掛けるプログラム（ソースファイル io_stress.c）

多く CPU を使用しており，その CPU 使用割合が 27.4%であることが読み取れる．つまり，関数 calculate_pow が CPU リソースの枯渇要因であることを示している．関数 calculate_pow は，負荷プログラムにおいて意図的に負荷を掛けるよう作成した関数である．よって，CPU リソースの枯渇要因となる意図的に負荷を掛けたアプリケーション関数を枯渇要因として提示できている．

また，図 11 の詳細情報を示す項目から，対象アプリケーション関数内のリソース使用の内訳が分かり，calculate_pow が使用している CPU 使用割合 27.4%の内の 20.7%は，ライブラリ libm-2.26.so の関数__pow_finite であり，cpu_stress_pow.c の 14 行目で対象のアプリケーション関数から呼び出していることが分かる．該当ソースコードは，負荷プログラムにおいて関数 pow を呼び出して負荷を掛けている部分である．よって，CPU リソースの枯渇要因となる意図的に負荷を掛けたソースコード箇所を提示できている．

以上から，CPU リソースの枯渇要因となる意図的に負荷を掛けたアプリケーションの関数およびソースコード箇所を提示できることを確認した．

メモリ，ストレージ I/O を対象にした評価についても，CPU と同様に H/W リソースの枯渇要因となる意図的に負荷を掛けたアプリケーション関数やそのソースコード箇所を提示できることを確認できた．詳細の掲載は割愛するが，H/W リソース使用処理の分析結果に関して一部に絞ったものを図 12，図 13 に示す．それぞれの結果から，関数 malloc，fprintf の呼び出しによりメモリ，ストレージ I/O に負荷を掛けているソースコード箇所を示せていることが分かる．

以上のように，CPU，メモリ，ストレージ I/O の H/W リソース枯渇に対して，その要因となる負荷を掛けたアプリケーションの関数およびソースコード箇所を提示できることを確認できた．

5. おわりに

本稿では，H/W リソース枯渇要因となるアプリケーションの関数およびソースコード箇所を自動的に分析する手法を提案した．提案手法は，H/W リソース枯渇範囲の H/W リソース使用処理に限定して測定を行うことで，測定処理負荷による測定結果への影響や測定データによる記憶容量消費を抑えることができる．また，H/W リソース枯渇範囲において対象 H/W リソースを使用する処理を，アプリケーションの関数およびソースコード箇所まで分析し，H/W リソースの枯渇要因として示すことができる．

提案手法をもとに実装したツールによって，CPU，メモリ，ストレージ I/O に関して H/W リソース枯渇を生じさせるアプリケーションの関数およびソースコード箇所を，H/W リソースの枯渇要因として提示できることを確認した．今後は実用化に向けた詳細な評価・改善を行う．

参考文献

[1] “procps-ng / procps” https://gitlab.com/procps-ng/procps, (参照 2021-1-12).
[2] “Linux perf Examples”. http://www.brendangregg.com/perf.html, (参照 2021-1-12).
[3] “LTTng: an open source tracing framework for Linux”. https://lttng.org/, (参照 2021-1-12).
[4] “Trace Compass User Guide - LTTng-UST Analyses”. https://archive.eclipse.org/tracecompass/doc/stable/org.eclipse.tracecompass.doc.user/LTTng-UST-Analyses.html, (参照 2021-1-12).
[5] “Valgrind Home”. https://valgrind.org/, (参照 2021-1-12).
[6] “UnixBench”. https://code.google.com/archive/p/byte-unixbench/, (参照 2021-1-12).
[7] “R-Car H3 and M3 Starter Kit | Renesas Electronics”. https://www.renesas.com/us/en/solutions/automotive/adas/solution-kits/r-car-starter-kit.html, (参照 2021-1-12).

Vol.2021-ARC-244 No.11
Vol.2021-SLDM-194 No.11
Vol.2021-EMB-56 No.11
2021/3/25

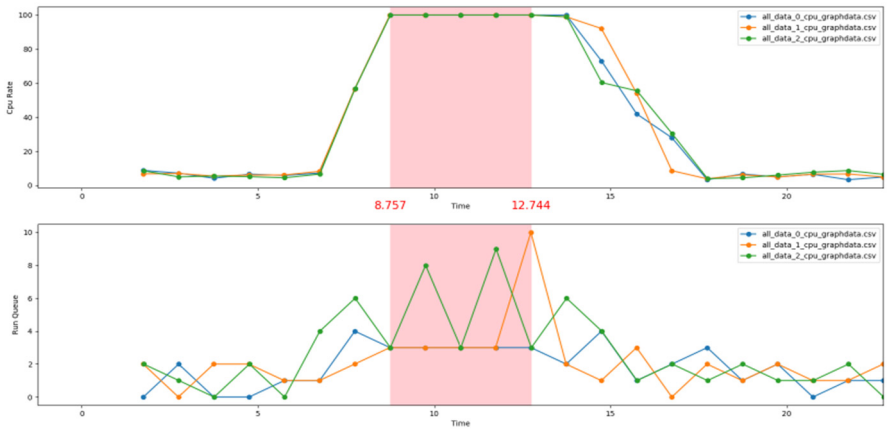

図 10　H/W リソース枯渇範囲抽出結果（CPU 負荷）

	A	B	C	D	E	F	G
1	1.OUTLINE			2.DETAIL			
2	[PROCESS]	[APP_FUNC]	[USAGE]	[BREAKDOWN]	[ACTUAL_FUNC]	[APP_SOURCE_CODE]	
3	pow1	calculate_pow	27.4				
4				20.7	_pow_finite (/lib/libm-2.26.so)	/home/melco/stress_CPU/pow.c:14	14　pow(count,2); 15 } 16}
5				3.1	_pow (/lib/libm-2.26.so)	/home/melco/stress_CPU/pow.c:14	14　pow(count,2); 15 } 16}
6				1.1	calculate_pow (/home/root/stress_CPU/pow)	/home/melco/stress_CPU/pow.c:13	13　count++; 14　pow(count,2); 15 }
7	pow3	calculate_pow	23.5				
8				17.5	_pow_finite (/lib/libm-2.26.so)	/home/melco/stress_CPU/pow.c:14	14　pow(count,2); 15 } 16}

図 11　　H/W リソース使用処理分析結果（CPU 負荷）

表 4　H/W リソース使用処理分析結果の項目

項目	内容
1.OUTLINE	（概要情報）
[PROCESS]	プロセス名
[APP_FUNC]	対象 H/W リソースを使用したアプリケーション関数
[USAGE]	H/W リソース使用量 - CPU：使用割合[%] - メモリ：メモリ使用量[Byte] - ストレージ I/O：アクセスデータ量[Byte]
2.DETAIL	（詳細情報）
[BREAKDOWN]	H/W リソース使用量の内訳
[ACTUAL_FUNC]	対象 H/W リソースを使用した実処理関数
[APP_SOURCE CODE]	アプリケーション関数内の該当ソースコード箇所

D	E	F	G
2.DETAIL			
[BREAKDOWN]	[ACTUAL_FUNC]	[APP_SOURCE_CODE]	
1702973440	__mmap (/lib/libc-2.26.so)	/home/melco/stress_MEM/mem_stress.c:15	15　str = (char *)malloc(sizeof(char) * LARGE_SIZE); 16　if(str == NULL) { 17　printf("error:malloc");
3379200	__brk (/lib/libc-2.26.so)	/home/melco/stress_MEM/mem_stress.c:29	29　str = (char *)malloc(sizeof(char) * SMALL_SIZE); 30　if(str == NULL) { 31　printf("error:malloc");

図 12　　H/W リソース使用処理分析結果（メモリ負荷）

D	E	F	G
2.DETAIL			
[BREAKDOWN]	[ACTUAL_FUNC]	[APP_SOURCE_CODE]	
8507392	__GI__libc_write (/lib/libc-2.26.so)	/home/melco/stress_IO/io_stress.c:20	20　fprintf(fp, "0123456789¥n"); 21　i++; 22 }
15904	__GI__libc_write (/lib/libc-2.26.so)	/home/melco/stress_IO/io_stress.c:23	23　fclose(fp); 24} 25

図 13　　H/W リソース使用処理分析結果（ストレージ I/O 負荷）