Vol.2014-OS-130 No.16
2014/7/29

組込みシステム向け メモリアクセス志向によるタスクの CPU コア割当て機構

松岡 武史[†1] 岩崎 学[†1] 福井 英理[†1]

近年，ネットワーク装置に代表される組込み装置にマルチコアプロセッサの搭載が進んでいる．組込み装置に搭載するソフトウェアに強く要求されるリアルタイム応答性を損なうことなく，スケーラブルな性能をマルチコアプロセッサ環境の下で引き出すためには，タスクの処理遅延の要因を排除する必要がある．本稿では，メモリアクセス志向に基づくタスクの動的な CPU コア割当て機構とその効果の予測について考察する．

Memory Access Oriented CPU Binding Method for Embedded System

TAKESHI MATSUOKA[†1] MANABU IWASAKI[†1] HIDEMICHI FUKUI[†1]

In recent years multi-core processors are used for embedded system represented by network equipment. To improve scalability on multi-core embedded systems without spoiling real-time performance, it's necessary to remove the factor of the task delay. We examine memory access oriented dynamic CPU binding method and expect of effect in this paper.

1. はじめに

ネットワーク装置に代表される組込み装置においては，タスクの動作に高いリアルタイム応答性が要求される．そのため，従来のリアルタイム OS（RTOS）のみならず，ネットワーク装置への適用が拡がっている Linux のような汎用 OS においても，リアルタイム制御技術の研究・開発が企業やコミュニティで続けられている．リアルタイム応答性を必要とするサービスの最小単位であるタスクには，極めて短い規定時間内に確実に応答する制御が要求される．そのため，タスクの処理遅延の要因を極力排除し，利用者が要求する応答性を確保するための制御が必要となる．

本稿では，近年，組込み装置への採用が進むマルチコアプロセッサの環境において，リアルタイム応答性を損なうことなくスケーラブルな性能を実現するために，仮想メモリシステムを使用する OS のメモリアクセス方法に着目し，キャッシュ同期によるオーバーヘッドを削減するための，タスクの動的な CPU コア割当て機構，及び，その評価方法について考察する．

2. マルチコアプロセッサの課題

2.1 マルチコアプロセッサへの期待と従来の取組み

スマートフォンに代表されるように，多機能化が進む組込み装置においては，プロセッサ性能の向上が必須である．シングルコアプロセッサの性能を向上するためのクロックアップでは発熱の問題があるため，CPU クロックを低く落としてコアを並列化させるマルチコアプロセッサの採用が主流となっている．マルチコアプロセッサには，コア数の増加に伴った単位時間内における各コアで並列実行される処理量の増加による性能向上と，CPU クロックの低下に伴う低消費電力化が期待されている．そのため，今後の装置開発を支える有力な技術ではあるが，その活用に向けた課題を十分に踏まえた対策が求められる．

マルチコアプロセッサには，コア数に対してリニアに性能が向上しないスケーラビリティの阻害問題がある．その要因としては，シングルコアプロセッサからそのまま移植されたタスクによるコア間の逐次処理待ち合わせや，複数のタスクがアクセスするメモリ領域のコア間でのロック等によるアプリケーションタスク並列化の阻害が挙げられる．この問題の対策として，OpenMP に代表される並列化技法により，アプリケーションタスクの並列度を向上させる取組みが行われている．

2.2 マルチコアプロセッサにおける問題

マルチコアプロセッサには，ソフトウェア要因とは別のハードウェア要因のオーバーヘッドが存在する．メモリやペリフェラルアクセスのためのバス調停や，キャッシュコヒーレンシを保つためのキャッシュ同期がそれであり，アプリケーションの並列化度に対して，リニアに性能が向上しない遅延要因の 1 つとなっている．

†1 富士通九州ネットワークテクノロジーズ(株)
Fujitsu Kyushu Network Technologies Ltd.

本稿では，並列化できない部分をアムダールの法則における性能向上の阻害（オーバヘッド）として考察する．図1に，8 コアのマルチコアプロセッサを利用した場合の，アムダールの法則による性能向上率と，遅延要因の位置付けを示す．

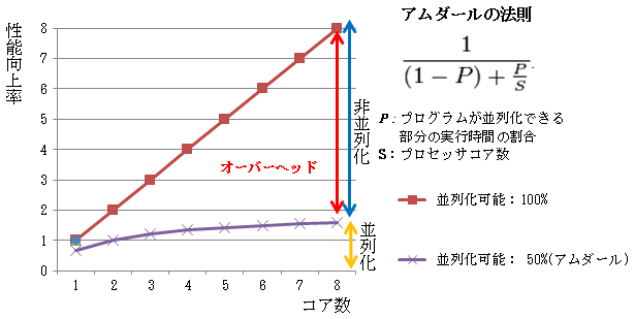

図 1：アムダールの法則における遅延要因の位置付け

8 コアのマルチコアプロセッサにおいて，アプリケーションタスクの並列化可能な処理の割合が 50%である場合，アムダールの法則により示される性能向上率は 1.6 である．ソフトウェア・ハードウェアによるコア間の干渉が全くなく，8 コアであれば性能向上率が 8 であることが理想的な並列化であり，これに対してギャップとして示される遅延要因（6.4）が，スケーラビリティを阻害している要因である．

2.3 マルチコアプロセッサにおける課題

マルチコアプロセッサを搭載する組込み機器において性能を引き出すためには，アプリケーションタスクの並列度を高めるアプローチに加え，このハードウェアによる遅延要因（オーバーヘッド）を解消する必要がある．この中で，キャッシュ同期については，タスクのメモリアクセス方法に依存しているため，ソフトウェアによる対策が求められている．

Linux を含め，多くの OS はマルチコアプロセッサの制御方法として均一に負荷を分散する SMP 方式を実装している．SMP において，複数コアに実装される各アプリケーションタスクに改造を加えることで，複数コアからアプリケーションが意識する同一メモリリソースをアクセスしないようにすることは可能である．事実，アプリケーションレベルにおいて，メモリアクセスの排他を極力排除するために，タスクがアクセスするメモリはコア内にローカライズすることを重要視して設計される．

ところが，Linux を含む多くの OS では，実際に搭載されているメモリ容量以上にアプリケーションがメモリ空間を利用可能とするように，仮想メモリシステムを採用している．この仮想メモリシステムにおいては，アプリケーションタスクが利用している仮想メモリと物理メモリの対応管理はカーネルとメモリ管理ユニット（MMU）が行っているため，アプリケーションタスクからブラックボックス化されている．

従って，アプリケーションタスクは，カーネルのシステムコールを通じて間接的にアクセスしているメモリリソース（リソース X と表現）がコア間で衝突し，キャッシュ同期が行われていることを意識することができない．図 2 に，カーネルのメモリリソースへのアクセスに伴うキャッシュ同期処理を示す．

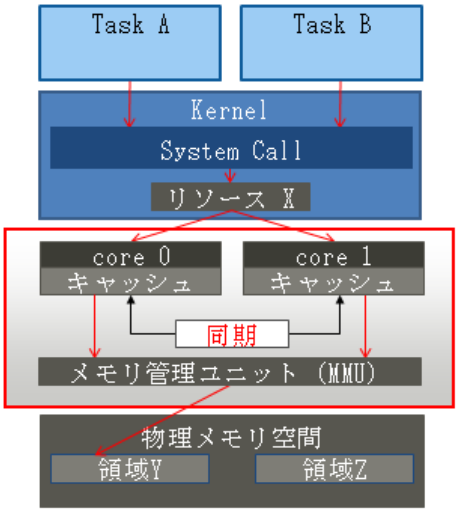

図 2：カーネルのメモリリソースへのアクセス

このアプリケーションが意図しないキャッシュ同期によるオーバーヘッドはコア数に対して指数的であり，マルチコアプロセッサのスケーラビリティを損なう大きな要因となる．

図 2 はシンプルなキャッシュ構造を示した例であり，最新の CPU アーキテクチャにおいては，複数コアでキャッシュを共有する例もある．図 3 に 4 コア単位で共有キャッシュを配備している CPU アーキテクチャの例を示す．

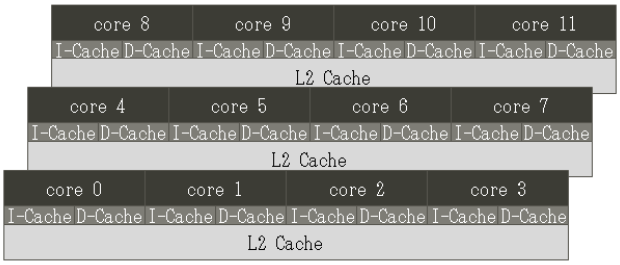

図 3：共有キャッシュを配備した CPU アーキテクチャの例

コア数の増加が進む将来的なメニーコアプロセッサに

おいて全てのキャッシュを共有することは，複数コアからの共有キャッシュアクセス時の調停（FIFO 制御）による待ち合わせ，即ちアクセス遅延が増加することになる．

そのため，図 3 に示すような CPU アーキテクチャの性能を引き出すためにも，キャッシュ同期によるオーバーヘッドを想定して対策を講じる必要がある．その解決すべき課題の 1 つは，アプリケーションが意図しないカーネル空間メモリリソースのキャッシュ同期に伴うオーバーヘッドの削減である．

3. 対策の考え方と具体的な施策

3.1 キャッシュ同期削減に向けた従来技術

キャッシュ同期の課題に対して，コア間に分散したタスクに実質的に物理メモリを共有させないことにより同期の回数を削減するための考え方は従来から存在している．参考文献 10 においては，複数コアからのメモリアクセスに用いられるロック変数と当メモリ領域へのアクセス処理を紐付け，タスク単位ではなく，ロックが必要なメモリアクセス単位で動作コアを集約する機構が提案されている．

3.2 キャッシュ同期削減に向けた本稿での考察

タスク単位ではなく，特定メモリアクセス処理のみを別のコアに移動させる機構では，当処理が移動先コアで実行されている間に移動元コアで処理完了を待ち合わせる場合には待ち合わせによる遅延が，移動元コアで他のタスクを動作させる場合にはコンテキストスイッチによるオーバーヘッドが発生する懸念がある．本稿では，リアルタイム応答性を損なわずに，タスクの処理遅延の要因を極力排除することを目的としているため，メモリアクセスをタスク単位で紐付ける機構について，以降に考察する．

仮想メモリ方式を採用している OS においては，アプリケーションがキャッシュ同期の対象となる物理メモリ領域の衝突を意識することができない．アプリケーションのタスクは，上位にタスクグループ（プロセス）という概念を持ち，タスクグループ内のタスクは，仮想メモリ空間を共有する．仮想メモリ方式を利用する OS では同一仮想メモリ空間内のタスクを，タスク生成時に同一コアに割当てるアルゴリズムを持っている．また，アプリケーションが利用するカーネル空間のメモリリソースは，全タスクグループに共有されている．

図 4 に仮想メモリシステムにおける，タスクグループ A 及びその空間上のタスク A/B，タスクグループ B 及びその空間上のタスク C/D と，カーネル空間の関係を示す．

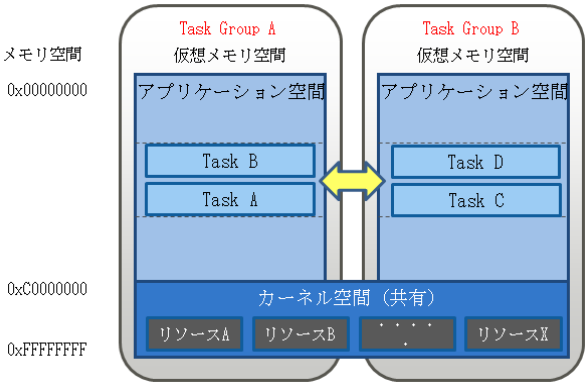

図 4：仮想メモリシステムのメモリ空間

この仮想メモリ空間において，カーネル空間のメモリリソースアクセスに伴うキャッシュ同期処理の削減を行うための制御は，カーネル側で実現する必要がある．

3.3 カーネル空間のメモリリソースへの仕掛け

タスクを集約する目的は，コア毎に使用するメモリ領域を限定させることである．そのために，各タスクが動作するコアがそれぞれ定まるように制御する．本稿では，そのコアを「タスク対応コア」と呼ぶ．カーネル空間で使用されるメモリの割当てはカーネルが行うため，その目的の達成には特別な仕掛けが必要である．メモリの割当てはカーネルに任せるが，以降の動作ではタスクを適切なコアに移動させることで，目的の達成を図る．

「タスク対応コア」の情報はカーネルのメモリリソースの構造体に追加する．即ち，メモリリソース単位にそのメモリが使用されるコアの印をつけておくものである．この情報によりカーネルでメモリリソースが割当てられているコアを判別する．また，本施策によりコアを移動したタスクを判別可能とするために，「タスク対応コアへの移動フラグ」を，スケジューラが参照可能なタスクの構造体に追加する．

3.4 具体的な基本制御

① メモリ獲得時のタスク動作コア設定

システムコールの延長で動作するカーネル空間の処理において，メモリ獲得が成功した場合，呼び出し元であるアプリケーションタスクに割り当てられているコア番号を取得し，獲得したメモリに割り付けるメモリリソース構造体の「タスク対応コア」に取得したコア番号を設定する．

図 5 にメモリ獲得時に実施する処理のフローを示す．

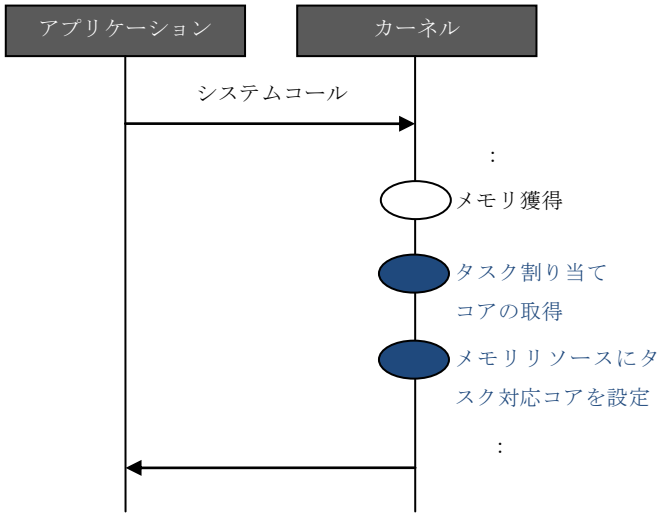

図 5：メモリ獲得時の処理

② メモリ参照・更新時のタスク動作コア移動

システムコールの延長で動作するカーネル空間の処理において，メモリにアクセスしている呼び出し元アプリケーションタスクに割り当てられているコア番号を取得する．取得したコア番号が,「タスク対応コア」と異なる場合には，アプリケーションタスクを「タスク対応コア」に移動し，「タスク対応コアへの移動フラグ」を ON に設定する．

図 6 にメモリ参照・更新時に実施する追加処理のフローを示す．

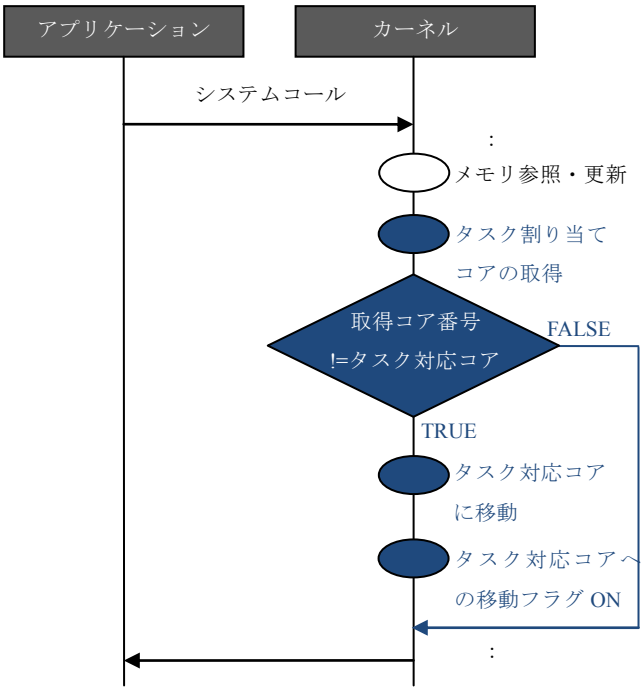

図 6：メモリ参照・更新時の処理

③ ロードバランス時のタスク動作コア移動

汎用 OS の例として，Linux のスケジューラには，特定のコアのロードアベレージが他コアと比較して高い場合，ロードアベレージの高いコアのタスクを他コアに移動するロードバランスの仕組みを搭載している．この機能はマルチコアプロセッサを均等に活用する SMP の目的に沿ったものであるが，施策により集約したタスクを，ロードバランス機能によって移動されてしまう可能性がある．

そのため，カーネルのロードバランス処理により，アプリケーションタスクの他コアへの移動が実施される場合には，②により動作コアを決定したタスク，即ち,「タスク対応コアへの移動フラグ」が ON に設定されているタスクは移動しない．

その際に他の候補として，「タスク対応コアへの移動フラグ」が ON に設定されていないアプリケーションタスクを決定し，コア移動を実施する．

図 7 にタスクスケジューラのロードバランス時に実施する追加処理のフローを示す．

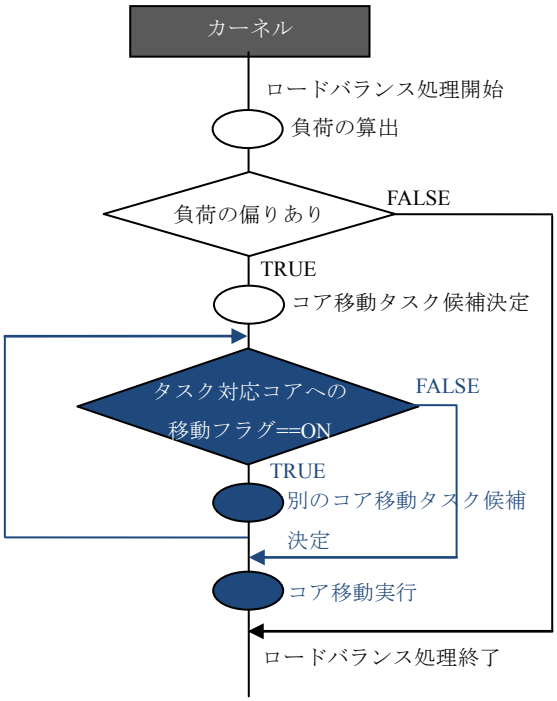

図 7：ロードバランス時の処理

以上の基本制御①②③について整理したものを，表 1 に示す．

表 1：動作コア決定の基本制御

契機となる処理	処理時の動作コア	制御
メモリ獲得	メモリ獲得時の動作コア	獲得したメモリに，「タスク対応コア」を設定する．
メモリ参照・更新	メモリ参照・更新時の動作コア	動作コアと「タスク対応コア」が異なっていたら，「タスク対応コア」に移動する．
ロードバランス	―	「タスク対応コア」が設定されているタスクは移動しない．

3.5 施策の実施例（共有メモリアクセス）

タスクグループが異なるアプリケーションタスク同士が，同一メモリ領域の参照・更新を行う必要がある場合，タスクは共有メモリの獲得をカーネルに要求し，カーネルは獲得した物理メモリ領域を要求元タスクの仮想アドレス空間にマッピングして返す．アプリケーションタスクはその仮想アドレスにアタッチすることによりメモリアクセスを実施する．

タスク A が共有メモリを獲得し，タスク C が共有メモリをアタッチした場合，タスク C がアクセスする共有メモリを生成したタスク A がタスク C と同一コアに割当てているか否かを判定し，別コアに割当てられていた場合には，タスク C が属するタスクグループ単位でタスク A と同一コアに移動する．この例を図 8 に示す．

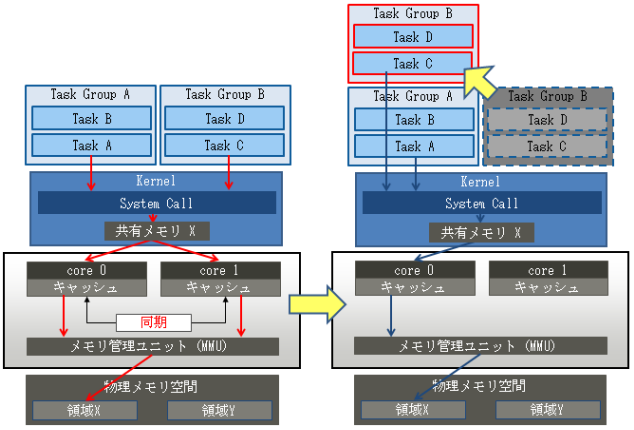

図 8：共有メモリアクセスタスクの集約

また，ロードバランス処理の実行時に，タスクグループ B がロードバランス対象と決定された場合，タスクグループ B はタスクグループ A との関係を所有しているため，ロードバランス対象外とし，他のタスクとメモリリソースの関係を持っていないタスクグループ C を，負荷の低い他コアに追い出す．この例を図 9 に示す．

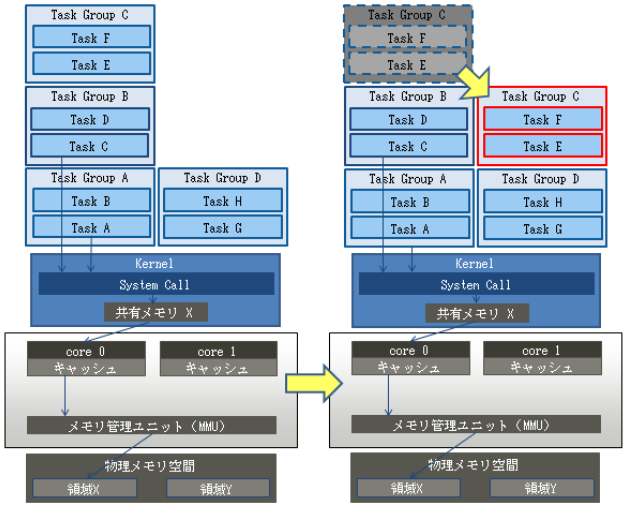

図 9：共有メモリアクセス依存性なしタスクの追い出し

3.6 施策の適用範囲

共有メモリ以外にも，アプリケーションタスクが意識することなく，同一物理メモリ領域にアクセスするカーネル空間のメモリリソースが存在する．メモリリソースに紐ついたタスクを特定のコアに集約するための適用範囲として，表 2 に組込み装置でよく使用されるカーネル空間のメモリリソースを示す．

表 2：カーネル空間のメモリリソース

メモリリソース	メモリリソース管理部	アクセス検出トリガ	「タスク対応コア」追加先
(共有メモリ), メッセージキュー，セマフォ	カーネル（IPC）	IPC リソースのアタッチ	IPC 構造体（カーネル）
メモリマップド IO	カーネル（デバイスドライバ）	ioctl によるアクセス要求	デバイスドライバ個別アクセス構造体（カーネル）
RAMFS	カーネル（ファイルシステム）	ファイルアクセス	ファイルオペレーション情報（カーネル）

上記を含め，カーネル空間でコア間に跨ってアクセスされるメモリリソースに施策の適用範囲を拡げることが，オーバーヘッドの削減に向けて有効な対策となる．

4. 施策効果の予測と考察

アプリケーションタスクが意図しないキャッシュ同期によるオーバーヘッドを削減する本施策の効果として，アムダールの法則における性能向上率の引き上げを見込むことができる．このリニアな性能向上率に対する実効係数を ρ、オーバーヘッド全体を ε とする場合，$\rho=1-\varepsilon$ であり，オーバーヘッドの削減は ρ を 1 に近づけることである．

8 コアのマルチコアプロセッサで並列化可能の割合が 50%である場合，リニアな性能向上率が 8.0、アムダールの法則による性能向上率が 1.6 であるため、施策適用前の ε と ρ は以下の通りである。

- 施策適用前のオーバーヘッド
  $\varepsilon=(8.0-1.6)/8.0=0.8$
- 施策適用前の実効係数
  $\rho=1-0.8=0.2$

図 10 に，アムダールの法則において位置付けたオーバーヘッド ε，オーバーヘッドの中のキャッシュ同期の割合 X，施策（キャッシュ同期削減）に適用するメモリリソースの全体に占める割合 Y の関係を示す．

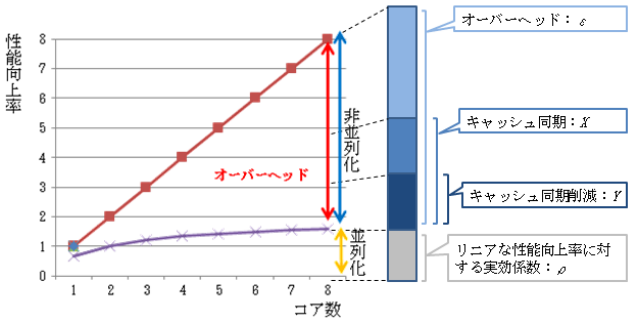

図 10：アムダールの法則における施策（Y）の位置付け

ε は $1-X*Y$ の割合で減少するため，施策適用後のオーバーヘッド $\varepsilon 2$ は，$\varepsilon 2=\varepsilon*(1-X*Y)$となる．例えば，$X,Y$ とも 0.5（50%）である場合，施策適用後の $\varepsilon 2$，ρ は以下の値となる．

- 施策適用後のオーバーヘッド
  $\varepsilon 2=0.8*(1-0.5*0.5)=0.6$
- 施策適用後の実効係数
  $\rho=1-0.6=0.4$（20%の向上）

施策が作用するオーバーヘッド ε は，アムダールの法則よりコア数の増加に伴って増大する．表 3 に 8 コアにおけるオーバーヘッドの中で，キャッシュ同期の割合が $X=0.5$ である場合の施策効果を示す．

表 3：8 コアで $X=0.5$ の場合の施策効果

適用範囲の割合 Y	オーバーヘッド $\varepsilon 2$	実効係数 ρ	性能向上率	備考
0.0	0.80	0.20	1.60	施策適用前
0.2	0.72	0.28	2.24	
0.5	0.60	0.40	3.20	
0.8	0.48	0.52	4.16	
1.0	0.40	0.60	4.80	

また，表 4 に 16 コアにおけるオーバーヘッドの中で，キャッシュ同期の割合が $X=0.5$ である場合の施策効果を示す．

表 4：16 コアで $X=0.5$ の場合の施策効果

適用範囲の割合 Y	オーバーヘッド $\varepsilon 2$	実効係数 ρ	性能向上率	備考
0.0	0.88	0.12	1.92	施策適用前
0.2	0.79	0.21	3.33	
0.5	0.66	0.34	5.44	
0.8	0.53	0.47	7.55	
1.0	0.44	0.56	8.96	

これらの表から，同じ適用条件の下では，8 コアより 16 コアのほうがオーバーヘッド削減の効果があることがわかる．例えば，適用可能の割合が 0.5（50%）である場合，8 コアの 20%オーバーヘッド削減に対して, 16 コアでは 22%のオーバーヘッド削減となる．

5. おわりに

本稿において，アプリケーションの並列化のみでは解決できないハードウェア要因のオーバーヘッドに含まれるキャッシュ同期を削減するために，メモリアクセスを基点としたタスクの動作コアを決定する施策，及び，その効果の予測について考察した．施策の効果に対する考察の妥当性については, 異なる CPU アーキテクチャのマルチコアプロセッサを複数使用し，SMP Linux に適用した上で評価データを収集し，分析を実施する予定である．

また，本稿では，複数のメモリリソースを扱うタスクについて，それぞれのメモリリソースに紐付けられた「タスク対応コア」が異なる場合，どのように振舞うべきかを言及していない．アプリケーションで意識できないカーネル空間のメモリリソースとタスクに 1 対 1 の関係を持たせることにより，マルチコアプロセッサ間のキャッシュ同期を行わせないことが本研究の本質であるため，カーネル空間を含むメモリアクセスを考慮したタスクの分割設計をどう行っていくべきか，実際の評価データからその指針を考案することが今後の課題である．

謝辞

本稿の執筆にあたり適切な助言とご支援を頂きました，岡山大学 大学院自然科学研究科の山内准教授に，感謝の意を表します．

参考文献

1) 稲垣 文二, 山崎 信行, "リアルタイム実行のための優先度付き SMT プロセッサ用 IPC 制御機構",情報処理学会論文誌，Vol.51, No.12，2206-2215 (Dec. 2010)

2) 山田 賢, 日下部 茂, "集約制御機構を持つコア間時間集約スケジューラの実装と評価",情報処理学会論文誌コンピューティングシステム（ACS），Vol.3，No.3，235-247，(Sep. 2010)

3) 大野 有輝, 菅谷 みどり, 秋岡 明香, 中島 達夫, "マルチコア環境でのプロセス動作予測によるリソース配分最適化",情報処理学会研究報告，Vol.2010-ARC-189 No.5，Vol.2010-OS-114 No.5，2010/4/21

4) 佐藤 公紀, 阿部 公輝, "マルチコアプロセッサのコアごとのアクセス局所性を利用した共有キャッシュの消費電力削減",情報処理学会研究報告，Vol.2010-ARC-187 No.3，Vol.2010-EMB-15 No.3，2010/1/28

5) 本橋 剛, 山田 浩史, 吉田 哲也, 河野 健二, "マルチコア CPU 環境における L2 キャッシュの影響を考慮した VM スケジューラ",情報処理学会研究報告，Vol.2009-OS-112 No.1，2009/8/5

6) 脇坂 洋祐, 柴田 直樹, 北道 淳司, 安本 慶一, 伊藤 実, "ターボブースト・ハイパースレッディングを考慮したマルチコアプロセッサ向けタスクスケジューリング",情報処理学会研究報告，Vol.2012-HPC-136 No.23，2012/10/4

7) 谷本 輝夫, 佐々木 広, 三輪 忍, 中村宏, "メニーコアプロセッサにおける競合とスケーラビリティを考慮したスレッドスケジューリング",情報処理学会研究報告，Vol.2011-ARC-197 No.31，Vol.2011-HPC-132 No.31，2011/11/29

8) 下沢 拓, 堀 敦史, 石川 裕, "メニーコア環境におけるキャッシュアウェア・オペレーティングシステムに向けて",情報処理学会研究報告，Vol.2011-ARC-195 No.2，Vol.2011-OS-117 No.2，2011/4/13

9) 海江田 章裕，中山 泰一，田中 淳裕，堀川 隆，蔵杉 俊康，紀一誠，“ロックの比率に着目した並列プログラムの分類”，情報処理学会研究報告, 1999-HPC-77, Vol.1999, No.66, pp.143-148 (1999).

10) 川名部 正純，日本電気株式会社，“マルチコアプロセッサシステム，プロセス制御方法，および，プロセス制御プログラム”，特開 2003-248666，2003-09-05

11) Andi Kleen, Tim Chen , “Scalability of modern Linux kernels” September 2010, LinuxCon Japan