マイクロアーキテクチャを考慮したスピンロックの PAUSE 命令最適化

森崎 光[1,a)]　辻本 有紀[1,b)]　河野 健二[1,c)]

概要：同じ命令セットを実装する CPU であっても，マイクロアーキテクチャによって命令の実行特性は異なる．スピン待機に用いられる PAUSE 命令もその一例であり，その実行レイテンシの違いがスピンロックの挙動やアプリケーションの性能に影響を及ぼすことが知られている．本研究では，PAUSE 命令の実行レイテンシの違いがスピンロックの挙動に及ぼす影響を分析し，マイクロアーキテクチャに応じた最適化の効果を評価する．具体的には，PAUSE 命令を用いたスピン待機を導入したキーバリューストア memcached を用いて，4 種類のマイクロアーキテクチャでスピンロック内部の PAUSE 挿入回数とスループットの関係を測定した．その結果，最大スループットを達成する PAUSE 命令の挿入回数はマイクロアーキテクチャ間で約 4 倍異なり，あるマイクロアーキテクチャで最適な設定を別のマイクロアーキテクチャに適用すると，適用先の最適設定と比較して最大 13%のスループット損失が生じることを確認した．さらに，PAUSE 挿入回数に伴うスピンロックの挙動の変化を，ロック獲得遅延などの観点から分析した．これらの結果を踏まえ，スピンロックの挙動とアプリケーションの性能を考慮した，マイクロアーキテクチャごとのスピンロック最適化指針を示す．

1. はじめに

ハードウェア資源の有効活用という観点から，ハードウェアの特性に応じてソフトウェアを最適化することが求められる．特にデータセンタをはじめとする大規模な計算環境では，マシンを段階的かつ定期的に更新する都合上，導入時期の異なる複数世代のマシンが大量に混在して稼働しているため [1], [2], [3], [4], [5], [6]，上位で動くワークロードをハードウェアに合わせて最適化することは，それによる性能向上が計算環境における大幅な運用費削減に寄与し得る [4], [7].

ハードウェアの特性は多岐にわたるが，なかでも CPU の違いはワークロードの性能に大きく影響する要因の一つである．複数世代の CPU が混在する環境においては，同じ命令セットを実装する CPU 同士であっても，マイクロアーキテクチャの違いによって命令の実行特性が異なり，ワークロードの性能に影響を及ぼす可能性がある (§2.1)．実際に，スレッドライブラリやデータベースなどの広く利用されるソフトウェアにおいても，CPU のアーキテクチャによる性能特性の違いを考慮し，実行時の挙動を調整するためのパラメータが提供されている [8], [9], [10]．しかし，こうしたパラメータが調整可能であっても，その設定がワークロードの性能にどの程度影響するのか，また個々の実行環境に対してどのような値が適切なのかを判断することは容易ではない [11].

本研究ではマイクロアーキテクチャの違いによる PAUSE 命令の実行特性に着目し，その違いが，幅広いソフトウェアで利用される基本的な同期機構の一つであるスピンロックの挙動に与える影響を分析する．スピンロックは，メモリ上の共有資源へのアクセスを排他制御するロックの中でも，スピンループによって継続的に共有資源の利用可否を示すロック変数を確認するものである．スピンループ内部には，プロセッサの実行資源の消費を抑制し，電力消費を抑えるという観点から，PAUSE 命令を挿入することが推奨されている [12].

スピンロックでの PAUSE 命令挿入の方式は多岐にわたるが，本研究では特にソースコード 1 のようなスピンロックを対象とする．具体的には，一回のスピンループにつき一つの PAUSE 命令を挿入する [13], [14] のではなく，特にユーザ空間でのスピンロック実装で見られる，ロック獲得の試行間に複数の PAUSE 命令を挿入するという方式である [8], [15], [16], [17]．このような使い方には，複数の PAUSE 命令の挿入によりロック変数を確認する頻度

[1] 慶應義塾大学
Keio University, Yokohama, Kanagawa 223–8522, Japan
[a)] hikaru.morisaki@sslab.ics.keio.ac.jp
[b)] tsujimoto@sslab.ics.keio.ac.jp
[c)] kono@sslab.ics.keio.ac.jp

ⓒ2026 Information Processing Society of Japan

```c
pthread_mutex_t mutex
  = PTHREAD_MUTEX_INITIALIZER;
// ...

// Spin phase
for (i = 0; i < N_SPIN_ROUNDS; i++) {
  if (!pthread_mutex_trylock(&mutex))
    goto acquired;
  for (j = 0; j < N /* μarch-dependent */; j++)
    asm volatile("pause");
}
// Blocking phase
pthread_mutex_lock(&mutex);

acquired:
// Critical section

pthread_mutex_unlock(&mutex);
```

ソースコード 1 本研究で対象とするスピンロックと PAUSE 命令の用法.

が減ることでロックの競合を減らせる利点がある一方で，PAUSE 命令の実行レイテンシによってロック獲得の再試行間隔が変化するため，同じコードを動かしていてもマイクロアーキテクチャの違いによってスピンロックを利用するアプリケーションの性能が変わるという難点がある．図 1 は，ソースコード 1 の形式のスピンロックを実装したインメモリキーバリューストア memcached [18] を異なる四世代のマシン上で動作させた時のスループットを表している．棒グラフの値は，各マシンにおいて性能が最大となるような PAUSE 挿入回数を探索した上で，その時のスループットを 1 とし，同じ値をすべてのマシンに横断して適用した場合の memcached の正規化スループットを表している．この結果から，すべてのマイクロアーキテクチャで最良の性能を発揮するような PAUSE の挿入回数はないことがわかる．

本研究の貢献． インメモリキーバリューストア memcached [18] を題材として，マイクロアーキテクチャごとの PAUSE 命令の実行レイテンシの違いを考慮したスピンロックの最適化を行うとともに，その最適化過程で得られた知見に基づき，他のアプリケーションにも同様の最適化を適用するための指針を示す．

マイクロアーキテクチャごとのスピンロック最適化． Intel CPU の四種類の世代 (Broadwell，Skylake，Ice Lake，Emerald Rapids) のそれぞれにおいてスピンロックへの PAUSE 命令の挿入回数を調整することで，最適な設定がマイクロアーキテクチャによって実際に異なることを示す (§3.2)．分析の結果，memcached の性能が最大となるような PAUSE 命令の挿入回数はマイクロアーキテクチャ間で

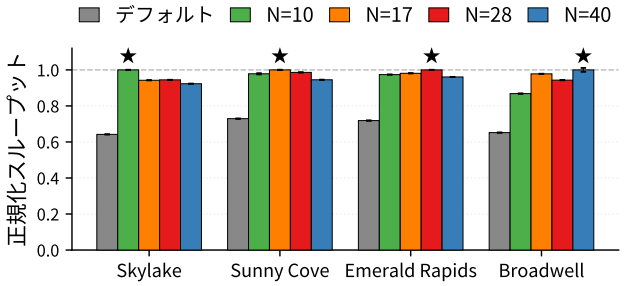

図 1 各マイクロアーキテクチャに最適な PAUSE 挿入回数を適用した場合の memcached の正規化スループット．詳細な実験環境は §3.1 に示す．

約 4 倍異なり，あるマイクロアーキテクチャで最適な設定を別のマイクロアーキテクチャに適用すると，適用先の最適設定と比較して最大 13%のスループット損失が生じることを確認した．

スピンロックの最適化指針． マイクロアーキテクチャ間で最適な PAUSE 挿入回数が異なる要因を明らかにし，ソースコード 1 の形式のスピンロックを用いるアプリケーションの性能を最大化するための最適化指針を示す (§3.3)．PAUSE 命令の実行レイテンシの差を吸収する素朴な対処法として，PAUSE 命令の実行時間を事前に計測したうえで一定の待機時間に相当する挿入回数を求める方法 [19], [20] が考えられるが，スピンロックの最適化においては，この方法だけではアプリケーションの性能を最大化できない (§3.3.1)．§3.3.2 では，PAUSE 命令の挿入回数を増やすことでロック競合が緩和される一方で，ロック獲得までの待ち時間が増加するというトレードオフが存在することを示す．この分析に基づき，§3.3.3 では，スピンロックの最適化に有用な指標として，ロック解放処理に要する時間である unlock latency と，あるスレッドがロックを解放してから次のスレッドがロックを獲得するまでの時間である handoff latency の二つに着目する．前者はロック競合の程度を，後者はロックの引き渡しに伴う遅延を評価するための指標であり，両者を考慮して PAUSE 命令の挿入回数を調整することが，アプリケーションの性能を最大化するための指針になると考えている．

2. 背景と研究動機

2.1 CPU のマイクロアーキテクチャの多様性

データセンタをはじめとする大規模計算環境では，大量のマシンのうち一部の古いマシンを新しいものに定期的に置き換えるというメンテナンスを行う都合上，複数世代のハードウェアが混在して稼働している [1], [2], [3], [4], [5], [6]．特に CPU に着目すると，命令セットに互換性があっても，CPU のマイクロアーキテクチャなどの違いによって同じワークロードの性能は変化する．

性能最適化の必要性． マイクロアーキテクチャのような

表 1 各マイクロアーキテクチャにおける PAUSE 命令の実行レイテンシ．

マイクロアーキテクチャ	レイテンシ [サイクル]
Broadwell	10
Skylake	142
Sunny Cove	38
Emerald Rapids	37

ハードウェア特性を考慮したソフトウェアの最適化は，大きな性能改善の余地があるうえ [4]，大規模な計算環境ではたとえ数%程度の性能改善であっても数百万ドル規模のコスト削減につながる可能性がある [21] ことから，経済的な観点で有望なアプローチである．

2.2 PAUSE 命令

PAUSE は，実行中のプログラムが短いスピンループを実行しているというヒントをプロセッサに伝える命令であり，スピンループ中のプロセッサ資源の消費の抑制や [22]，電力消費の抑制 [22], [23] を目的としている．

実行レイテンシの違い．PAUSE 命令は名目では共通の役割を持つ一方，その実行に要する時間 (実行レイテンシ) は異なるマイクロアーキテクチャの間で一定ではないことが知られている [12]．表 1 は四種類のマイクロアーキテクチャにおける PAUSE 命令の実行レイテンシを示している．PAUSE 命令の実行レイテンシは Broadwell では 10 サイクルであるのに対し，Skylake では 142 サイクルと実行レイテンシに約 14 倍の差がある．また，Skylake より新しい Sunny Cove と Emerald Rapids ではそれぞれ 38 サイクル，37 サイクルと実行レイテンシが短縮されており，傾向としてマイクロアーキテクチャの世代が新しくなるにつれて単調に変化するわけではない．

2.3 スピンロック

スピンロックは，複数のスレッドによるメモリ上の共有資源へのアクセスを排他制御する同期機構の一つである．ロックの獲得を試みた際に他のスレッドがロックを保持している場合，スピンループ内でロックの状態を示すロック変数を繰り返し確認し，ロックが解放されるまで待機する．スレッドをスリープさせずにロックの解放を待つため，短時間でロックが解放される場合には，スレッドのスリープと復帰に伴うオーバーヘッドを回避できる利点がある一方で，スピンループでの待機中にプロセッサの実行資源を消費し，消費電力が増大するという欠点がある．

PAUSE 命令の活用．スピンロックの利点を保ちつつ欠点を軽減するため，スピンループ内に PAUSE 命令を挿入することが推奨されている [12]．PAUSE 命令は，スピン待機中であることをプロセッサに伝えるヒントとして機能する．このヒントを利用するプロセッサでは，待機中の消費電力を抑制でき，またスピンループからの脱出時のパイプラインフラッシュを回避することで性能低下を抑制する効果もある．

PAUSE 命令の利用方式．スピンロック中の PAUSE 命令の利用には様々な方法があるが，本研究では特にユーザレベルでのスピンロック実装 [8], [15], [16], [17] によく見られるソースコード 1 のような利用方式を対象とする．この方式はスピンロックにスレッドのブロックを伴うロックを組み合わせることで，短時間でロックを獲得できる場合はスピンロックの恩恵を受けながら，ロック獲得までに時間がかかる場合にはスピンロックの欠点である CPU 資源の消費を抑制しようとするものであり，以下の手順でロックの獲得を試みる．

- **Spin phase．** スピンループを定数回だけ繰り返しながらロックの獲得を試みる．各スピンループでロックの獲得に失敗した場合は，PAUSE 命令を特定の回数分実行してから再試行する．Linux カーネルのスピンロックなどに見られる，ロック獲得の試行 1 回ごとに PAUSE 命令を 1 回実行する方式 [13], [14] とは異なり，複数回の PAUSE 命令を連続して実行できる．これによりロック獲得の試行頻度が下がり，複数のスレッドが同一のロック変数にアクセスすることによるキャッシュラインの競合を緩和できる．一方，連続した PAUSE 命令の実行中はロックの状態を確認できないため，挿入回数を増やしすぎると他のスレッドがロックを解放したことを検出するのが遅れる可能性がある．
- **Blocking phase．** スピンループでロックを獲得できなかった場合にこのフェーズに移行し，ロックが解放されるのを待つ．ロックの競合状況よってはスレッドがスリープ状態となるため，スピンループとは異なり待機中の CPU 資源の消費を抑制できる．

3. マイクロアーキテクチャを考慮したスピンロック最適化

3.1 実験環境

ハードウェア．CloudLab [24] 上で提供されている，マイクロアーキテクチャの異なる Intel CPU を搭載した 4 種類のマシン，xl170 (Broadwell)，c220g5 (Skylake)，sm110p (Sunny Cove)，c6620 (Emerald Rapids) のいずれかを，各ベンチマークにおいて 1 台ずつ使用した．すべてのマシンで Simultaneous Multi-Threading (SMT) および Turbo Boost を無効化し，CPU の動作周波数をベースクロックに固定した．ベンチマークでは単一ソケット内の CPU コアのみを使用する．

memcached へのスピンロックの組み込み．インメモリキーバリューストア memcached に対し，ソースコード 1 に示した方式のスピンロックを実装した．スピンループを実行する回数 `N_SPIN_ROUNDS` は 30 回に固定し，各スピン

ⓒ2026 Information Processing Society of Japan

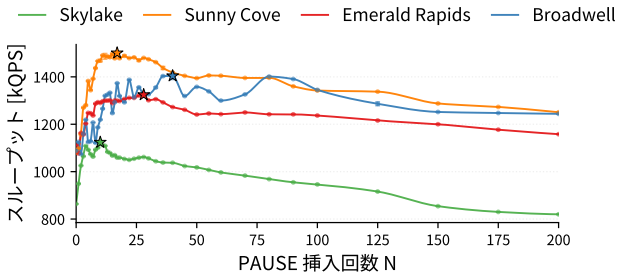

図 2 4 種のマイクロアーキテクチャにおける PAUSE 挿入回数とスループットの関係.

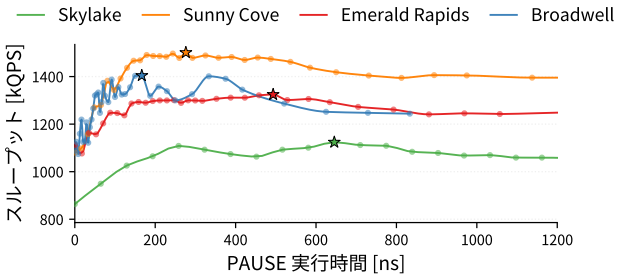

図 3 1 スピンループあたりの全ての PAUSE 命令の実行にかかる時間に対する memcached スループット.

ループでの PAUSE 挿入回数 `N` を変更する．また，このスピンロックを memcached 内部のアイテムの排他制御に用いられる `item_locks` と，スラブアロケータの排他制御に用いられる `slabs_lock` で使用するように変更を加えた．

ベンチマーク． 1 台の計測用マシン上で，memcached サーバと memcached に対して負荷を生成するプログラムである mutilate [25] を動作させる．memcached と mutilate のスレッド数はそれぞれ 4 に設定し，実験中の CPU 干渉を防ぐために memcached は CPU コア 0〜3 のみ，mutilate は CPU コア 4〜7 のみを使用するよう，CPU affinity を固定する．mutilate は 1 スレッドあたり 1 つのコネクションを使用し，パイプライン深度は 32 で，memcached に対して単一のキーへの GET と SET (1:1) の負荷を生成する．memcached 側の PAUSE 命令の挿入回数 N を 0 から 200 までの 40 通りに変化させ，各条件について 120 秒間のウォームアップ後，60 秒間の測定を 30 回実施し，その平均値をプロットする．

3.2 スピンロック最適化によるアプリケーションの性能改善

本章では，PAUSE 命令の挿入回数が memcached の性能に与える影響を評価し，本研究で取り扱うマイクロアーキテクチャごとに性能を最大化する挿入回数を明らかにする．

図 2 に，4 種類のマシンで PAUSE 命令の挿入回数 N を変化させた場合のスループットを示す．いずれのマイクロアーキテクチャにおいても，N が増加するにつれてスループットが向上し，最大値に達した後にスループットが低下する傾向が見られる．スループットを最大化する N は，Broadwell で 40，Skylake で 10，Sunny Cove で 17，Emerald Rapids で 28 であり，マイクロアーキテクチャ間で最大 4 倍の差がある．また，PAUSE 命令を全く挿入しない $N = 0$ の場合と比較すると，各マイクロアーキテクチャで最適な N を適用することで，スループットが 1.37〜1.56 倍に向上した．これらの結果から，PAUSE 命令の挿入回数は memcached の性能に大きく影響し，その最適値はマイクロアーキテクチャによって異なることがわかる．

3.3 スピンロックの最適化指針

本章では，まず § 3.2 の結果に見られるようなマイクロアーキテクチャごとの性能特性を吸収する素朴な手法について議論し，その限界を示す．次に，Skylake マイクロアーキテクチャを題材に，§ 3.2 で見られた山なりのスループットの傾向が現れる理由について分析し (§ 3.3.2)，その分析を基に本研究の対象とするスピンロック内部の PAUSE 命令の挿入回数に関する最適化指針を示す (§ 3.3.3).

3.3.1 素朴な方法：PAUSE レイテンシの正規化

PAUSE 命令の実行レイテンシの差による memcached の性能差を吸収するために，PAUSE 命令の実行時間を事前に計測したうえで，ある一定の待機時間に相当する挿入回数を求める方法が考えられる．この方法は主に，複数回の PAUSE 命令の実行によってスレッドをスリープさせずに一定の時間を経過させるための待機プリミティブの実装に使われているものの [19], [20]，スピンロックを用いるアプリケーションの性能を最大化することはできないことを示す．

図 3 に § 3.2 で得られた各マイクロアーキテクチャのスループットを，1 スピンループあたりの PAUSE 命令の実行回数ではなく，その実行にかかる時間に対してプロットした結果を示す．実行時間は，各マイクロアーキテクチャの PAUSE 命令の実行レイテンシ (表 1) に挿入回数を乗じて算出した．4 種類のマイクロアーキテクチャのいずれにおいても，PAUSE 命令の実行時間に対するスループットは山なりの傾向を示した．一方，スループットを最大化する PAUSE 命令の実行時間は，Broadwell では約 167 ns，Sunny Cove では約 276 ns，Emerald Rapids では約 493 ns，Skylake では約 645 ns となり，最大で約 3.9 倍異なった．この結果から，PAUSE 命令の実行時間が一定となるように挿入回数を正規化する既存の手法だけでは，異なるマイクロアーキテクチャ上で memcached のスループットを最大化できないといえる．したがって，スピンロックの最適化には，PAUSE 命令の実行レイテンシだけでなく，その挿入回数がスピンロックの挙動に与える影響を考慮する必要がある．

ⓒ2026 Information Processing Society of Japan

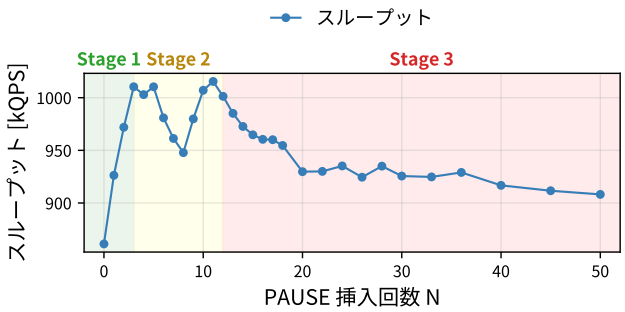

図 4 Skylake マイクロアーキテクチャにおける PAUSE 挿入回数によるスループットの変化の傾向.

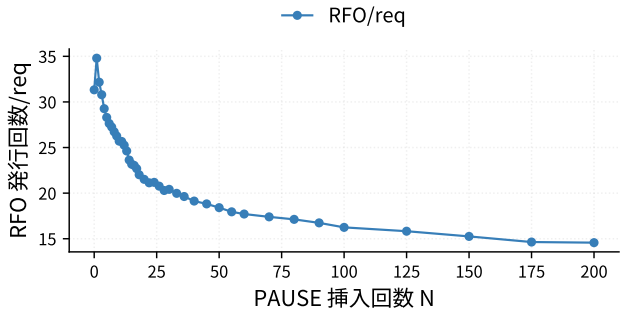

図 5 Skylake における PAUSE 挿入回数とリクエスト当たりの RFO 発行回数の関係.

3.3.2 PAUSE 命令の増減によるスピンロックの挙動への影響

本節では，Skylake マイクロアーキテクチャを題材に，PAUSE 命令の挿入回数がスピンロックの挙動に与える影響を分析する．特に，§ 3.2 で観察された山なりのスループットの傾向について，図 4 のように，PAUSE 挿入回数の増加に伴ってスループットが向上する領域 (Stage 1)，改善が飽和する領域 (Stage 2)，および低下する領域 (Stage 3) に分け，それぞれの要因を分析する．これらの領域では，ロック変数をめぐる競合の緩和と，ロック獲得に要する時間の増加という，異なる効果が現れると考えられる．

左側：ロック競合の減少．PAUSE 命令の挿入回数を増やすと，ロック獲得を試行する頻度が低下し，複数の CPU コアによるロック変数への競合が緩和されると考えられる．この影響を調べるため，Read For Ownership (RFO) の発行回数を測定した．RFO は，CPU コアがキャッシュラインの書き込み権限を取得する際に発行する要求であり，その発行頻度からロック変数を含むキャッシュラインの競合を推定することができる．図 5 に示すように，PAUSE 挿入回数が増えるにつれて，リクエスト当たりの RFO 発行回数は減少する傾向にある．この結果は，PAUSE 命令の挿入によってロック変数へのアクセス頻度が抑えられ，キャッシュラインの競合が緩和されるという説明と整合する．

右側：ロック獲得待ち時間の増加．一方，PAUSE 挿入回数が一定以上になると，スループットは緩やかに低下する．これは，1 スピンループ当たりの PAUSE 挿入回数が増えることで，ロック獲得を再試行するまでの間隔が長くなるためと考えられる．スピンループ内で PAUSE 命令を実行している間に他のスレッドがロックを解放しても，待機スレッドは次のロック獲得試行までその解放を検出できない．そのため，PAUSE 挿入回数が過剰になると，ロック解放から次のスレッドによる獲得までの時間が増加し，スループットの低下につながると考えられる．

以上より，PAUSE 挿入回数の増加には，ロック変数への競合を緩和する一方で，ロック解放の検出を遅らせるというトレードオフが存在する．スループットが最大となる挿入回数はこれらの効果の均衡によって決まると考えており，次節 (§ 3.3.3) ではこのトレードオフを踏まえ，適切な PAUSE 挿入回数を決定するための指針を検討する．

3.3.3 PAUSE 挿入回数の最適化指針

これまでの評価では，PAUSE 挿入回数を変化させながらアプリケーションのスループットを測定し，最適な挿入回数を特定した．しかし，この方法ではアプリケーションごとに複数の設定を試して性能を比較する必要がある．本節では，アプリケーション全体の性能測定に依存せず，スピンロック内部で観測できる情報から適切な PAUSE 挿入回数を推定する方法を考える．前節で述べたように，PAUSE 挿入回数はロック変数への競合とロック解放後の獲得待ち時間の双方に影響する．本節では，これらの影響を評価する指標として，unlock latency と handoff latency に着目する．

Unlock latency．Unlock latency は，ロックを保持するスレッドがロック解放処理を開始してから完了するまでの時間と定義する．ロック変数への書き込みには，そのキャッシュラインの書き込み権限が必要となる．複数の CPU コアが同じロック変数に対して獲得や解放を試みると，キャッシュラインの所有権移動が発生し，ロック解放処理にも遅延が生じ得る．したがって，unlock latency はロック変数をめぐる競合の程度を推定する指標として利用できる．また，ロック獲得時間には，ロックが他のスレッドに保持されている間の待機時間も含まれるのに対し，unlock latency はロック解放処理自体に要する時間を測定できるという利点がある．

図 6 は，図 2 と同じ実験条件で測定した unlock latency を示している．PAUSE 挿入回数が少ない領域では unlock latency が大きく，挿入回数を増やすにつれて減少する傾向が見られる．この傾向は中央値だけでなく，上位パーセンタイルでも確認でき，PAUSE 挿入回数の増加によってロック変数への競合が緩和されるという前節の分析と整合する．一方，挿入回数が一定以上になると改善は小さくなり，それ以上 PAUSE を増やしても競合緩和による効果は限定的になると考えられる．

ⓒ2026 Information Processing Society of Japan

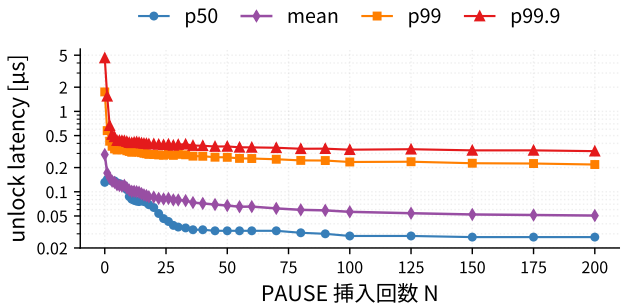

図 6 Skylake における PAUSE 挿入回数と unlock latency の関係.

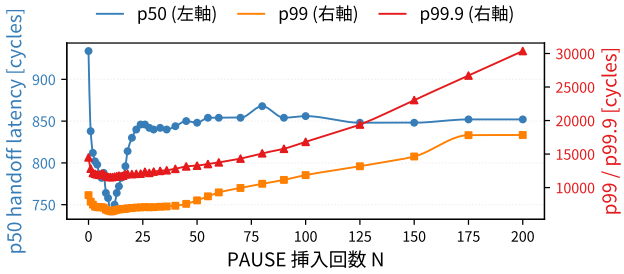

図 7 Skylake における PAUSE 挿入回数と handoff latency の関係.

Handoff latency. Handoff latency は，あるスレッドによるロック解放が完了してから，次のスレッドがそのロックを獲得するまでの時間と定義する．PAUSE 挿入回数が多い場合，ロックが解放されても待機スレッドは PAUSE 命令の実行を終えるまで次の獲得試行を行えない．そのため，handoff latency は PAUSE 挿入回数の増加に伴うロック解放の検出遅延を評価する指標となる．

図 7 は，図 2 と同じ実験条件で測定した handoff latency を示している．中央値は PAUSE 挿入回数が少ない領域で減少した後，挿入回数の増加に伴って緩やかに増加する．一方，99 パーセンタイルおよび 99.9 パーセンタイルでは，挿入回数が多い領域でより顕著な増加が見られる．これは，複数の待機スレッドが PAUSE 命令を実行している間にロックが解放され，いずれのスレッドも直ちに獲得を試行できない状況が生じている可能性を示唆する．

以上の結果から，unlock latency と handoff latency は，PAUSE 挿入回数を調整する際に考慮すべき二つの異なる効果を捉える指標となる．すなわち，unlock latency の減少はロック変数への競合緩和を，handoff latency の増加はロック解放後の獲得遅延をそれぞれ示す．したがって，unlock latency の改善が飽和し，handoff latency の悪化が顕著になる領域を避けることが，PAUSE 挿入回数を決定する際の指針となる．

4. 関連研究

異種 CPU に応じた最適化. 異なる CPU 上でのソフトウェアの効率的な実行に向けて，ハードウェアの違いを考慮した最適化が研究されてきた．MELF は，異なる ISA やマイクロアーキテクチャに対応する複数のコードバリアントを一つの実行ファイルに格納し，実行環境に適したバリアントを選択する仕組みを提案している [26]．また，Chimera は，ISA 拡張のサポートが異なるコア間でのプログラム実行を可能にするため，バイナリ書き換えを用いた命令の変換を実現している [27]．一方，本研究では同じ ISA を実装する CPU 間でもマイクロアーキテクチャによって命令の実行特性が異なる点に着目している．uops.info は x86 命令の実行レイテンシ，スループット，実行ポートの使用特性をマイクロアーキテクチャごとに測定する手法を提案している [28]．さらに，PMEvo や PALMED は命令の実行資源に関する性能モデルを構築し，uiCA や Facile は基本ブロックのスループットを予測する手法を提案している [7], [29], [30], [31]．これらの研究と同様に，本研究もマイクロアーキテクチャによる命令実行特性の違いに着目するが，命令やベーシックブロックの性能解析ではなく，その違いがスピンロックの挙動を介してアプリケーション全体の性能に及ぼす影響を分析し，マイクロアーキテクチャに応じたスピン待機のパラメータ設定について指針を示す点で異なる．

PAUSE の実行特性と待機の調整. PAUSE 命令はスピン待機ループで利用されるが，その実行時間はマイクロアーキテクチャによって大きく異なる [12]．特に Skylake 世代で PAUSE 命令の実行時間が増大したことを受け，既存ソフトウェアでも待機処理の調整が行われてきた．MySQL InnoDB では，CPU 世代による PAUSE 命令の実行時間の違いを考慮し，スピン待機中の PAUSE 実行回数を調整するパラメータが導入されている [9]．また，.NET ランタイムでは，PAUSE 命令の実行時間の違いを吸収するため，プロセッサに応じて待機処理を調整する変更が提案されている [11], [20]．ユーザ空間パケット処理フレームワークである Data Plane Development Kit [32] にも，PAUSE 命令の実行時間を測定し，一定時間の待機に必要な実行回数を求める実装が存在する [19]．これらは，マイクロアーキテクチャごとの PAUSE 実行時間の違いを考慮した待機処理の調整例である．しかし，スピンロックでは，PAUSE の実行回数を増やすことでロック変数へのアクセス頻度を抑えられる一方，ロック解放の検出が遅れる可能性もある．したがって，PAUSE 命令の実行時間に基づいて待機時間を一定にするだけでは，必ずしもアプリケーション性能を最大化できない．本研究では，異なるマイクロアーキテクチャ上で PAUSE 実行回数とアプリケーション性能の関係を測定し，スピン待機のパラメータを最適化する際に考慮すべき要因を明らかにしている．なお，ロック獲得の失敗に応じて待機時間を指数関数的に増加させる指数バックオフも既存のスピン待機手法として用いられているが，その

ⓒ2026 Information Processing Society of Japan

方式と本研究で扱う固定回数の PAUSE 待機との性能比較は今後の課題とする．

5. おわりに

本研究では，マイクロアーキテクチャごとの PAUSE 命令の実行レイテンシの違いを考慮したスピンロックの最適化指針を示した．具体的には，ロック獲得の試行１回あたりに複数の PAUSE 命令を挿入する形式のスピンロックにおける最適な PAUSE 挿入回数を，インメモリキーバリューストアの memcached を題材として調査した．さらに，アプリケーションの性能を繰り返し測定することなく PAUSE 挿入回数を調整できるようにするため，ロックの獲得・解放処理の範囲内で計測できる２つの指標，すなわち，1) ロック解放処理に要する時間である unlock latency と，2) ロック解放から次のスレッドによる獲得までの時間である handoff latency に着目した．マイクロアーキテクチャの違いを考慮したスピンロック最適化を行う上では，PAUSE 挿入回数を，unlock latency の改善が飽和し，handoff latency の悪化が顕著になる前の範囲に調整することが望ましいと考えている．本研究で示した分析と指針が，同様のスピン待機方式を採用する他のアプリケーションや同期ライブラリにおいても，マイクロアーキテクチャに応じたロックの最適化に活用されることを期待する．

謝辞 この成果は，ＮＥＤＯ（国立研究開発法人新エネルギー・産業技術総合開発機構）の委託事業「ポスト５Ｇ情報通信システム基盤強化研究開発事業」（JPNP20017）の結果得られたものです．

参考文献

[1] Google: The Production Environment at Google, from the Viewpoint of an SRE, https://sre.google/sre-book/production-environment/ (2026). Accessed: 2026-10-16.

[2] Microsoft: VM lifecycle overview, https://learn.microsoft.com/en-us/azure/virtual-machines/sizes/lifecycle/lifecycle-overview (2026). Accessed: 2026-10-16.

[3] Amazon Web Services: Amazon EC2 instance types, https://docs.aws.amazon.com/ec2/latest/instancetypes/instance-types.html (2026). Accessed: 2026-10-16.

[4] Mars, J. and Tang, L.: Whare-map: heterogeneity in "homogeneous" warehouse-scale computers, *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, New York, NY, USA, Association for Computing Machinery, p. 619–630 (online), DOI: 10.1145/2485922.2485975 (2013).

[5] Metri, G., Srinivasaraghavan, S., Shi, W. and Brockmeyer, M.: Experimental Analysis of Application Specific Energy Efficiency of Data Centers with Heterogeneous Servers, *2012 IEEE Fifth International Conference on Cloud Computing*, pp. 786–793 (online), DOI: 10.1109/CLOUD.2012.89 (2012).

[6] Ishiguro, K., Fonyuy-Asheri, C., Barraud, E., Lachaize, R., Bromberg, Y.-D. and Tchana, A.: Everything You Need to Know About Virtual Machine Live Migration Between Heterogeneous Processors, *Proceedings of the 21st European Conference on Computer Systems*, EUROSYS '26, New York, NY, USA, Association for Computing Machinery, p. 1399–1413 (online), DOI: 10.1145/3767295.3803609 (2026).

[7] Ritter, F. and Hack, S.: PMEvo: portable inference of port mappings for out-of-order processors by evolutionary optimization, *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2020, New York, NY, USA, Association for Computing Machinery, p. 608–622 (online), DOI: 10.1145/3385412.3385995 (2020).

[8] GNU Project: GNU C Library: POSIX Threads, https://www.gnu.org/software/libc/. Accessed: 2026-10-26.

[9] Oracle: MySQL 8.0 Reference Manual: Configuring Spin Lock Polling, https://dev.mysql.com/doc/refman/8.0/en/innodb-performance-spin_lock_polling.html. Accessed: 2026-10-16.

[10] MariaDB: MDEV-19845: Adaptive spin loops, https://jira.mariadb.org/browse/MDEV-19845 (2019). Accessed: 2026-10-16.

[11] .NET developers: GitHub Issue in the .NET Repository: Spin wait tuning, https://github.com/dotnet/runtime/issues/8744. Accessed: 2026-10-26.

[12] Intel Corporation: *Intel 64 and IA-32 Architectures Optimization Reference Manual: Volume 1* (2024). https://cdrdv2-public.intel.com/821612/248966-Optimization-Reference-Manual-V1-050.pdf.

[13] Linux Kernel Developers: Linux Kernel: Queued Spinlock Implementation, https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/kernel/locking/qspinlock.c. Accessed: 2026-10-26.

[14] Linux Kernel Developers: Linux Kernel: Optimistic Spin Queue Lock Implementation, https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/kernel/locking/osq_lock.c. Accessed: 2026-10-26.

[15] NGINX: NGINX Shared Memory Mutex Implementation, https://github.com/nginx/nginx/blob/9a7439fd79076de03c408de136ba1772347030c2/src/core/ngx_shmtx.c. Accessed: 2026-10-26.

[16] UXL Foundation: oneTBB: oneAPI Threading Building Blocks, https://github.com/uxlfoundation/oneTBB. Accessed: 2026-10-26.

[17] Crossbeam Developers: Crossbeam: Tools for concurrent programming., https://docs.rs/crossbeam/latest/crossbeam/. Accessed: 2026-10-26.

[18] Memcached Developers: Memcached: Free & Open Source, High-Performance, Distributed Memory Object Caching System, https://memcached.org. Accessed: 2026-10-29.

[19] Anatoly Burakov: [dpdk-dev] [PATCH v20 3/4] power: add PMD power management API and callback, https://mails.dpdk.org/archives/dev/2021-January/197997.html. Accessed: 2026-10-26.

[20] .NET developers: GitHub Pull Request in the .NET Repository: Add normalized equivalent of YieldProcessor, retune some spin loops, https://github.com/dotnet/coreclr/pull/13670. Accessed: 2026-10-26.

[21] Nagendra, N. P., Ayers, G., August, D. I., Cho, H. K., Kanev, S., Kozyrakis, C., Krishnamurthy, T., Litz, H., Moseley, T. and Ranganathan, P.: AsmDB: Understanding and Mitigating Front-End Stalls in Warehouse-Scale

ⓒ2026 Information Processing Society of Japan

Computers, *IEEE Micro*, Vol. 40, No. 3, pp. 56–63 (online), DOI: 10.1109/MM.2020.2986212 (2020).

[22] Intel Corporation: *Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 2: Instruction Set Reference, A-Z*. Accessed: 2026-10-26.

[23] Advanced Micro Devices, Inc.: *AMD64 Architecture Programmer's Manual, Volumes 1–5* (2026).

[24] Duplyakin, D., Ricci, R., Maricq, A., Wong, G., Duerig, J., Eide, E., Stoller, L., Hibler, M., Johnson, D., Webb, K., Akella, A., Wang, K., Ricart, G., Landweber, L., Elliott, C., Zink, M., Cecchet, E., Kar, S. and Mishra, P.: The Design and Operation of CloudLab, *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, USENIX Association, pp. 1–14 (online), available from ⟨https://www.usenix.org/conference/atc19/presentation/duplyakin⟩ (2019).

[25] Leverich, J.: Mutilate: high-performance memcached load generator, `https://github.com/leverich/mutilate`. Accessed: 2026-10-26.

[26] Töllner, D., Dietrich, C., Ostapyshyn, I., Rommel, F. and Lohmann, D.: MELF: Multivariant Executables for a Heterogeneous World, *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, Boston, MA, USENIX Association, pp. 257–273 (online), available from ⟨https://www.usenix.org/conference/atc23/presentation/tollner⟩ (2023).

[27] He, J., Pan, Q., Zhao, R., Qi, J., Liang, K., Xu, J., Li, Z., Wang, Y., Yu, J. and Wu, Y.: Chimera: Transparent and High-Performance ISAX Heterogeneous Computing via Binary Rewriting, *Proceedings of the 21st European Conference on Computer Systems*, EUROSYS '26, New York, NY, USA, Association for Computing Machinery, p. 279–295 (online), DOI: 10.1145/3767295.3769323 (2026).

[28] Abel, A. and Reineke, J.: uops.info: Characterizing Latency, Throughput, and Port Usage of Instructions on Intel Microarchitectures, *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, New York, NY, USA, Association for Computing Machinery, p. 673–686 (online), DOI: 10.1145/3297858.3304062 (2019).

[29] Derumigny, N., Bastian, T., Gruber, F., Iooss, G., Guillon, C., Pouchet, L.-N. and Rastello, F.: PALMED: Throughput Characterization for Superscalar Architectures, *2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pp. 106–117 (online), DOI: 10.1109/CGO53902.2022.9741289 (2022).

[30] Abel, A. and Reineke, J.: uiCA: accurate throughput prediction of basic blocks on recent intel microarchitectures, *Proceedings of the 36th ACM International Conference on Supercomputing*, ICS '22, New York, NY, USA, Association for Computing Machinery, (online), DOI: 10.1145/3524059.3532396 (2022).

[31] Abel, A., Sharma, S. and Reineke, J.: Facile: Fast, Accurate, and Interpretable Basic-Block Throughput Prediction, *2023 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 87–99 (online), DOI: 10.1109/IISWC59245.2023.00023 (2023).

[32] DPDK Project: DPDK: an open source framework for fast packet processing on standard hardware, `https://www.dpdk.org`. Accessed: 2026-10-29.

ⓒ2026 Information Processing Society of Japan