CPUオーバコミット時における並列アプリケーション実行の最適化

高山 都旬子[1]　光来 健一[1]

概要：クラウドコンピューティングの普及により，仮想マシン（VM）内で並列アプリケーションを動かすことも増えてきた．クラウドにおいて CPU オーバコミットを行っている場合，このような VM が複数実行されると物理 CPU が不足することがある．その際に，VM が利用可能な物理 CPU の減少分以上に性能が低下することが報告されており，VM の仮想 CPU 数を減らすという最適化手法が提案されている．しかし，先行研究では限定的な状況での効果しか確認されていない．物理 CPU 不足時の様々な対処について調査を行った結果，先行研究で提案されているように仮想 CPU 数を減らしても最適性能を達成できない場合があることが分かった．そこで，本稿では実行時情報に基づいて VM の仮想 CPU 数を動的に最適化することで並列アプリケーションの性能を改善する pCPU-Est を提案する．この手法に加えて，pCPU-Est はアプリケーションスレッド数を変更することで間接的に仮想 CPU 数を減らす手法も提供する．これらの最適化のために，pCPU-Est は VM に仮想 CPU アフィニティが設定されている場合でも VM が利用可能な物理 CPU 数を見積もることを可能にする．先行研究と比較して，仮想 CPU 数の動的最適化は最大 30%，スレッド数の最適化は最大 99%性能を向上させられることが確認できた．

1. はじめに

近年，CPU のコア数の増加により，並列アプリケーションの利用が進んでいる．並列アプリケーションは処理をスレッドに分割して CPU の各コアに割り当て，同時に処理を行うことで高速化を図る．クラウドコンピューティングの普及により，クラウドが提供する仮想マシン（VM）の中で並列アプリケーションを動作させることも増えてきた．仮想化環境では，CPU のコア（物理 CPU）は仮想化され，複数の仮想的な CPU（仮想 CPU）として VM に提供される．物理 CPU が十分にある場合，一つの物理 CPU が一つの仮想 CPU に割り当てられる．

一方，CPU のオーバコミットが行われると物理 CPU 数よりも仮想 CPU の総数のほうが大きくなるため、物理 CPU が足りなくなる場合がある．例えば，マイグレーションによって VM が他のサーバに移動された場合などが考えられる．このように十分な物理 CPU が確保できない時には，VM が利用可能な物理 CPU の減少分以上にアプリケーションの性能が低下することが報告されている [1], [2], [3]．そこで，VM の仮想 CPU 数を VM が利用可能な物理 CPU 数に合わせて減らすことでアプリケーションの性能低下を抑える最適化手法が提案されている [1], [2], [3]．VM の仮想 CPU 数を減らすことで仮想 CPU と物理 CPU を 1 対 1 に対応づけ，性能低下の原因となるロックホルダ・プリエンプション [5] などの問題を回避することができる．

しかし，これらの先行研究では，限定的な状況のみで物理 CPU 不足時の性能低下および提案手法による性能改善を示していた．そこで，我々はまず，物理 CPU 不足時の様々な対処についてアプリケーションの性能を測定し，先行研究の効果を検証した．その結果，いずれの場合でもアプリケーションの性能低下が起きるが，性能低下の度合いは対処によって異なることが分かった．また，先行研究で用いられている仮想 CPU 数が最適ではない場合があることも分かった．

本稿では，実行時情報に基づいて VM の仮想 CPU 数を動的に最適化することで並列アプリケーションの性能を改善する pCPU-Est を提案する．pCPU-Est は VM の CPU 使用率に基づいて最適な仮想 CPU 数を決定する．この手法に加えて，pCPU-Est は，アプリケーションスレッド数を減らすことで間接的に VM が利用する仮想 CPU 数を減らす手法も提供する．そのために，pCPU-Est は VM 内のアプリケーションが最適な仮想 CPU 数を取得することを可能にする．いずれの手法でも VM が利用可能な物理 CPU 数が基準となるため，pCPU-Est は仮想 CPU アフィニティを考慮して利用可能な物理 CPU 数を見積もる．

[1] 九州工業大学
Kyushu Institute of Technology

我々は pCPU-Est を Xen 4.4 に実装し，物理 CPU の不足時にこれらの最適化手法を用いて並列アプリケーションの性能が改善できるかを調べた．その結果，仮想 CPU 数の動的最適化では先行研究と同等かそれ以上の性能改善を行うことができ，先行研究と比べて性能が最大で 30%向上した．アプリケーションスレッド数の最適化ではどのベンチマークでも先行研究より高い効果が得られ，先行研究からの性能向上は最大で 99%となった．

以下，2 章で物理 CPU 不足時に VM の性能を改善するための従来手法について述べ，3 章で並列アプリケーションの性能低下および先行研究の効果に関する予備実験の結果を示す．4 章で pCPU-Est の提案を行い，5 章で pCPU-Est を用いて行った実験結果について述べる．6 章で関連研究に触れ，7 章で本稿をまとめる．

2. 物理 CPU 不足時の VM の性能改善

仮想化環境では物理的な CPU コア（物理 CPU）は仮想化され，仮想的な CPU（仮想 CPU）として VM に提供される．物理 CPU が十分にある場合は，1 つの物理 CPU は 1 つの仮想 CPU にだけ割り当てられる．割り当て先の仮想 CPU はスケジューリングによって変わる可能性があるが，仮想 CPU アフィニティを設定することによって割り当てを固定することもできる．VM 上で並列アプリケーションが動いているとき，アプリケーションのスレッドはプロセススケジューラによって仮想 CPU に割り当てられて動作する．

仮想化環境では CPU のオーバコミットを行い，物理 CPU より多くの仮想 CPU を動作させることが多い．これはすべての仮想 CPU が物理 CPU を 100%使うとは限らないためである．一般に，サーバの CPU 使用率は 10〜20%程度と言われている．しかし，並列アプリケーションは CPU を占有することが多いため，オーバコミットを行うと物理 CPU が不足する場合がある．特に，VM がマイグレーションによって他のホストに移動されると，ハードウェア構成やまわりの VM のワークロードが大きく変化する．そのため，マイグレーション前は十分な物理 CPU を使えていても，マイグレーション後には物理 CPU が不足するという事態が起こる可能性がある．

このような場合に VM が利用可能な物理 CPU の減少分以上にアプリケーションの性能低下が起きることが報告されている [1], [2], [3]．性能低下の原因はロックホルダ・プリエンプション [5] や vCPU スタッキング [6] などである．ロックホルダ・プリエンプションは，ロックを保持している仮想 CPU に割り当てられた物理 CPU が他の仮想 CPU に奪われることで処理が遅延する現象である．vCPU スタッキングは，ロックを待っている仮想 CPU がロックを保持している仮想 CPU よりも先に物理 CPU を割り当てられたために処理が進められない現象である．これらの現象は VM が利用可能な物理 CPU 数が VM の仮想 CPU 数を下回ると発生しやすくなる．

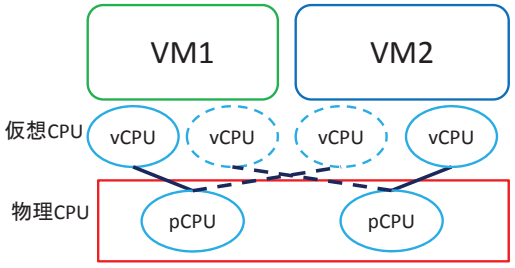


図 1 先行研究の最適化手法

この問題を解決するために，図 1 のように VM が利用可能な物理 CPU 数に合わせて VM の仮想 CPU 数を減らす最適化手法が提案されている [1], [2], [3]．しかし，これらの先行研究では物理 CPU の不足時に 2 台の VM 間で物理 CPU を共有した場合の性能低下と提案手法による性能改善を調べているだけである．そのため，多くの VM で物理 CPU を共有した場合や，VM の CPU 使用率を制限した場合，VM への物理 CPU の割り当てを削減した場合における性能は不明である．また，先行研究では VM の変更後の仮想 CPU 数は

$$\text{物理 CPU の総数} \times \frac{\text{VM の重み}}{\text{VM の重みの合計}}$$

で算出される．VM の重みが一定の場合は物理 CPU の総数を VM 数で割った値となる．これは VM が利用可能な物理 CPU 数である．しかし，この算出方法が最適かどうかについては議論されていない．少なくとも，VM に仮想 CPU アフィニティが設定されている場合には最適でない場合がある．

3. 予備実験

物理 CPU 不足時の 3 つの対処について，VM 上で動く並列アプリケーションの性能低下と先行研究による性能改善を調べる実験を行った．また，最も性能が改善する仮想 CPU 数の調査を行った．

実験に用いた対処は以下の三つである。

- **複数の VM 間で物理 CPU を共有**
  1 つの物理 CPU を複数の VM の仮想 CPU に割り当てる．各仮想 CPU が利用可能な物理 CPU の割合は VM に設定された重みによって決まる．ただし，割り当てられた物理 CPU を使い切らない仮想 CPU がある場合はその分を他の仮想 CPU が利用できる．
- **VM が利用できる CPU 使用率を制限**
  VM に CPU 使用率の上限を設定することによって，複数の VM 間で物理 CPU を分け合う．初期状態では VM は利用可能な物理 CPU 数× 100%の CPU を利用できる．何個の物理 CPU を何%ずつ利用するかは仮

想 CPU スケジューラによって決まる．

- **VM への物理 CPU 割り当てを削減**
  仮想 CPU アフィニティを利用して，1 つの物理 CPU を 1 つの VM にだけ割り当てる．VM が物理 CPU を占有できるが，1 つの物理 CPU が複数の仮想 CPU に共有される．

この実験には，AMD Opteron 6376（16 コア）の CPU を 2 基，320GB のメモリを搭載したマシンを用いた．仮想化ソフトウェアには Xen 4.4.0 を使用し，16 個の仮想 CPU と 4GB のメモリを持つ VM を用いた．対象とした並列アプリケーションは NAS Parallel Benchmarks[4] である．

3.1 物理 CPU 不足時の性能低下と先行研究の効果

まず，VM 間で物理 CPU を共有した場合に VM の台数を 1 台から 8 台まで増やしていき，各 VM でのベンチマークの実行時間を測定した．この実験では，VM の台数が 1 台の時の実行時間に VM の台数を掛けた時間を目標実行時間とした．例えば，VM を 2 台にした時は 2 倍の実行時間が目標となる．そして，測定した実行時間を目標実行時間で割ることにより相対実行時間を算出した．ベンチマークごとの相対実行時間の最悪値を図 2 に示す．最も性能低下の小さい EP でも 20%の性能低下となっており，LU に関しては 426 倍も実行時間がかかっている．これらのベンチマークに対して先行研究を適用した結果を図 3 に示す．BT では先行研究によって性能が改善され，目標実行時間に近くなっていることが分かる．LU でも大幅な性能の改善が見られたが，まだ最大で 61 倍の実行時間がかかった．いずれの場合も，VM が 2 台の時よりも 2 台を超えた時のほうが大きく性能を改善できている．

次に，VM の CPU 使用率を制限した場合に VM の CPU 使用率を 1600%から 100%まで減らしていき，1 台の VM に対してベンチマークの実行時間を測定した．この実験では，CPU 使用率が 1600%の時の実行時間に $\frac{1600}{\text{VM に設定した CPU 使用率}}$ を掛けた時間を目標実行時間とした．例えば，CPU 使用率を 800%に減らした時には 2 倍の実行時間が目標となる．ベンチマークごとの相対実行時間の最悪値を図 4 に示す．LU を除いて物理 CPU の共有時よりも性能低下が大きくなった．先行研究を適用すると図 5 のように CPU 使用率を低く設定した時に大幅に性能が改善された．しかし，LU においては CPU 使用率が 1200%の時に仮想 CPU 数を減らすことにより逆に 11 倍の実行時間がかかった．

最後に，物理 CPU 割り当てを削減した場合に 1 台の VM に割り当てる物理 CPU 数を 16 個から 1 個まで減らしていき，1 台の VM に対してベンチマークの実行時間を測定した。この実験では、物理 CPU 数が 16 個の時の実行時間に $\frac{16}{\text{VM に割り当てた物理 CPU 数}}$ を掛けた時間を目標実行時間とした．例えば，物理 CPU 数を 8 個に減らした時には 2 倍の実行時間が目標となる．ベンチマークごとの相対実行時間の最悪値を図 6 に示す．他の二つの対処ほど性能が低下していないことが分かる．目標実行時間とほとんど変わらない実行時間のベンチマークがある一方で，LU や UA のように性能が大幅に低下する場合もあった．先行研究を適用した結果を図 7 に示す．BT では最適化を行わない場合でもそれほど性能が低下しなかったため先行研究との差が小さい．しかし，物理 CPU 数が 12 個の場合には最適化しない場合より 34%実行時間が長くなった．物理 CPU が不足すると性能が極端に低下する LU のようなベンチマークでは先行研究の効果が大きかった．

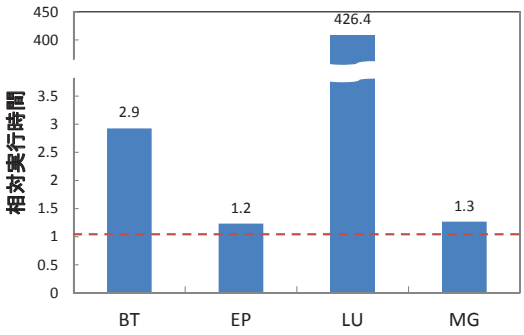


図 2　物理 CPU 共有時の相対実行時間の最悪値

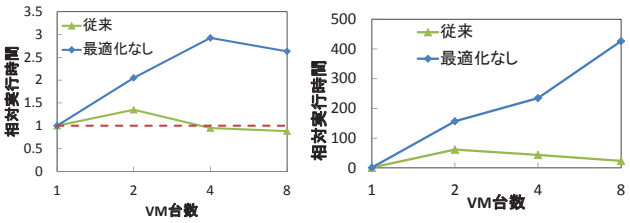


図 3　物理 CPU 共有時の先行研究による最適化

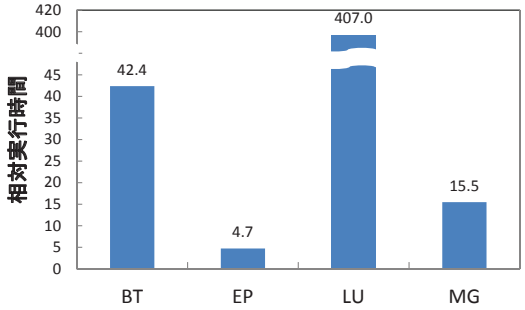
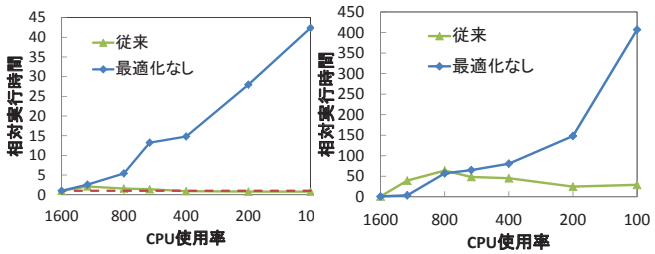


図 4　CPU 使用率制限時の相対実行時間の最悪値

3.2 最適な仮想 CPU 数

3.1 節の実験より，先行研究を適用するとむしろ性能が低下する場合があることが分かった．そこで各ベンチマークについて様々な仮想 CPU 数で性能測定を行い，最適な仮想 CPU 数の調査を行った．VM に割り当てる物理 CPU 数を 8 個に減らした時に，先行研究のように仮想 CPU も 8 個に減らした時の性能に対する改善率を図 8 に示す．BT



BT　　LU

図 5　CPU 使用率制限時の先行研究による最適化

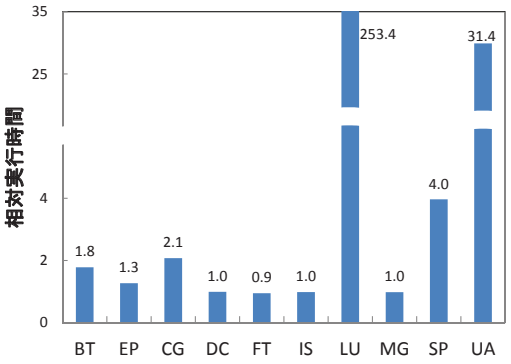
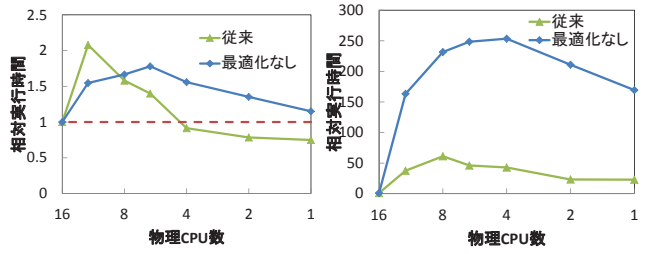


図 6　CPU 削減時の相対実行時間の最悪値



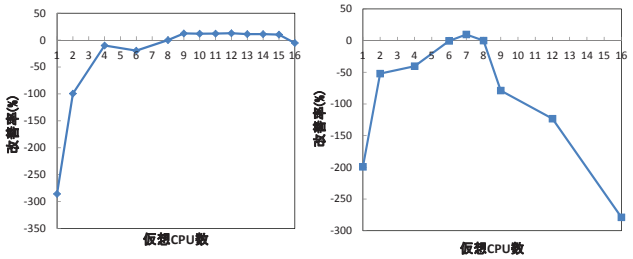
BT　　LU

図 7　物理 CPU 削減時の先行研究による最適化

では仮想 CPU が 9〜15 個の時に先行研究よりも性能を改善でき，12 個の時に最も改善率が大きいことが分かる．LU では仮想 CPU が 7 個の時に先行研究よりも性能が改善する．様々な物理 CPU 数について，最適な仮想 CPU 数を調査した結果を図 9 に示す．BT では割り当てられる物理 CPU が多い時，最適な仮想 CPU 数は利用可能な物理 CPU 数よりも大きくなっている．LU では物理 CPU 数が 8 個と 4 個の時に最適な仮想 CPU 数は利用可能な物理 CPU 数よりも小さいという結果になった．このようにベンチマークによって最適な仮想 CPU 数は必ずしも VM が利用可能な物理 CPU 数とならないことが分かった．この調査によって判明した最適な仮想 CPU 数を用いた場合に，先行研究と比べて性能が最大でどのくらい改善するかを図 10 に示す．ベンチマークによって改善率は異なるが，最大で約 90%改善できることが分かった．

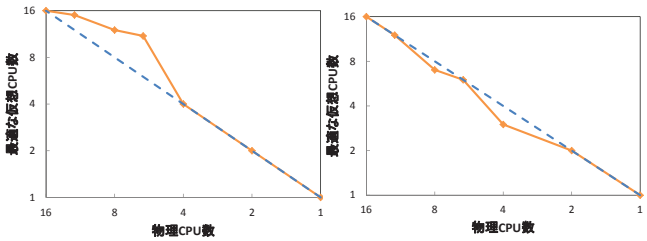
VM の CPU 使用率を制限した場合の最適な仮想 CPU 数は図 11 のようになった．BT では利用可能な物理 CPU 数と同数で最適となったが，LU では CPU 使用率が 1200%と 800%の時に最適な仮想 CPU 数は利用可能な物理 CPU 数よりも大きくなった．最適な仮想 CPU 数を用いた場合の先行研究の性能に対する最大改善率を図 12 に示す．改善率はどのベンチマークでも大きく，改善率が最小の EP でも 79%となっている．ほかのベンチマークでは 90%以上の改善が見られた．

VM 間で物理 CPU を共有する場合には，BT，LU ともに最適な仮想 CPU 数は利用可能な物理 CPU 数と同数となった．このように同じベンチマークでも物理 CPU 不足時の対処によって最適な仮想 CPU 数は異なる．先行研究で提案されているように利用可能な物理 CPU 数と同数の仮想 CPU 数にするという手法では性能が十分に向上しない場合があることが分かった．



BT　　LU

図 8　物理 CPU8 個の時の先行研究に対する性能改善率



BT　　LU

図 9　物理 CPU 削減時の最適な仮想 CPU 数

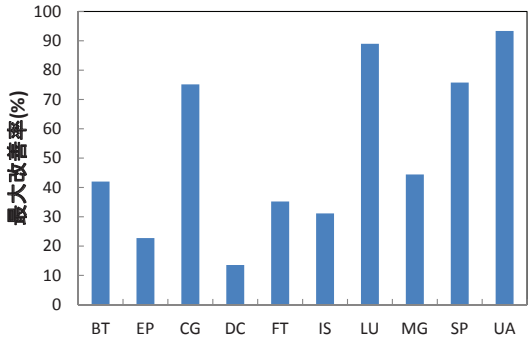


図 10　物理 CPU 削減時の最大改善率

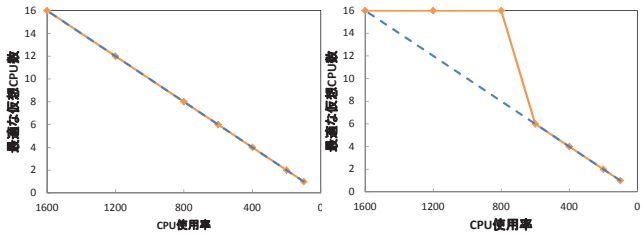


図 11　CPU 使用率制限時の最適な仮想 CPU 数

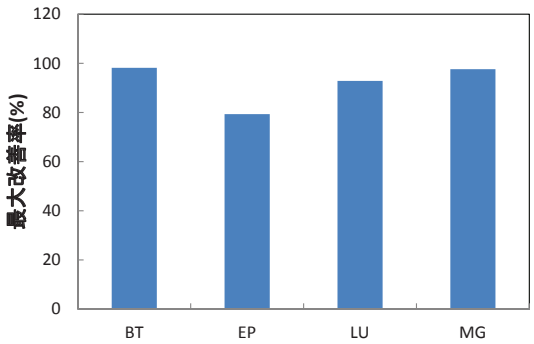


図 12　CPU 使用率制限時の性能の最大改善率

4. pCPU-Est

本稿では，実行時情報に基づいて仮想 CPU 数を動的に最適化することで物理 CPU 不足時の並列アプリケーションの性能を改善する pCPU-Est を提案する．この手法に加えて，pCPU-Est はアプリケーションスレッド数を最適化する手法も提供する．また，これらの最適化のために，VM に仮想 CPU アフィニティが設定されているときでも利用可能な物理 CPU 数を見積もることを可能にする．

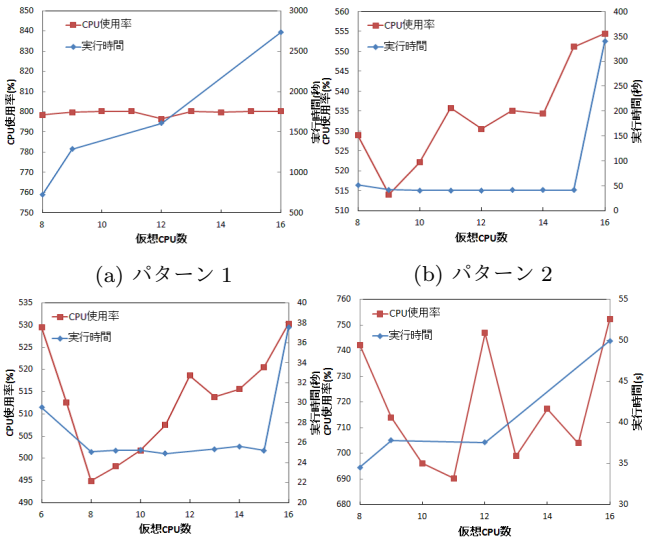
4.1 仮想 CPU 数の動的最適化

実行時情報を用いて VM の仮想 CPU 数を最適化するために，我々は VM の CPU 使用率に着目した．VM の仮想 CPU 数を変えながらベンチマーク実行中の CPU 使用率を測定し，実行時間との相関関係を調べたところ，図 13 のように大きく 4 つのパターンに分類できることが分かった．VM が利用可能な物理 CPU 数より仮想 CPU 数が小さい場合については，それが最適になることはほぼないため省略している．

- CPU 使用率が一定（図 13(a)）
  この場合には仮想 CPU を増やすほど性能は低下する．利用可能な物理 CPU を使い切っており，仮想 CPU を増やしても物理 CPU の利用効率を上げることはできない．また，仮想 CPU を増やすとロックホルダ・プリエンプション等の影響が大きくなる．
- CPU 使用率が徐々に増加して一定になる（図 13(b)）
  この場合には性能も徐々に向上し，CPU 使用率が一定になったあたりで性能が向上しなくなるか低下する．CPU 使用率が増加している間は，仮想 CPU の増加による並列性向上の効果があると考えられる．
- CPU 使用率が大きく低下した後，増加するか一定になる（図 13(c)）
  この場合には CPU 使用率が低下している間，性能は向上する．CPU 使用率が低下しなくなると性能が向上しなくなるか低下する．
- CPU 使用率が大きく増加・減少を繰り返す（図 13(d)）
  この場合には仮想 CPU を増やすほど性能は低下する．

この分類に基づいて，pCPU-Est は VM の CPU 使用率を測定しながら最適な仮想 CPU 数を動的に決定する．pCPU-Est は利用可能な物理 CPU 数と同数の仮想 CPU 数から始めて，CPU 使用率が増加する間，仮想 CPU 数を増やしていく．そして，CPU 使用率が一定になった時点での仮想 CPU 数を最適とする．CPU 使用率が最初から大きく低下した場合は，増加に転じるか一定になった時点での仮想 CPU 数を最適とする．また，CPU 使用率が大きく増加・低下を繰り返す場合には，利用可能な物理 CPU 数を最適な仮想 CPU 数とする．

Xen 4.4 の xl vcpu-set コマンドでは VM に対する仮想 CPU 数の変更がうまく行えなかったため，ドメイン 0 から VM の仮想 CPU 数を変更する機構を開発した．ドメイン 0 から XenStore に VM の新しい仮想 CPU 数を書き込むと，VM 内にインストールされた Linux カーネルモジュールが XenStore への書き込みを検知する．カーネルモジュールは XenStore から新しい仮想 CPU 数を読み込み，必要に応じて CPU のホットプラグまたはホットアンプラグを行う．具体的には，各 CPU の状態をオンラインまたはオフラインに変更することで，指定された仮想 CPU 数の CPU だけを有効にする．



(a) パターン 1　(b) パターン 2

(c) パターン 3　(d) パターン 4

図 13　CPU 使用率と実行時間の関係

4.2 アプリケーションスレッド数の最適化

pCPU-Est は VM 内で使われるアプリケーションスレッド数を減らすことで間接的に仮想 CPU 数を減らす．これはスレッドが割り当てられていない仮想 CPU がスケジューリングされないことを利用している．これによりカーネルレベルだけでなく，アプリケーションレベルでのロックホルダ・プリエンプションなども解消可能である．アプリケーションのスレッド数の調整は VM 内でユーザが行う必要があるため，pCPU-Est は VM が利用可能な物理 CPU 数を返すハイパーコールを提供する．我々の実験によると，最適なスレッド数は VM が利用可能な物理 CPU 数と常に同数であったため，このハイパーコールが返す値をスレッド数として用いる．ただし、アプリケーションのスレッド数を変更できるかどうか，また，実行中に変更できるかどうかはアプリケーション依存である．

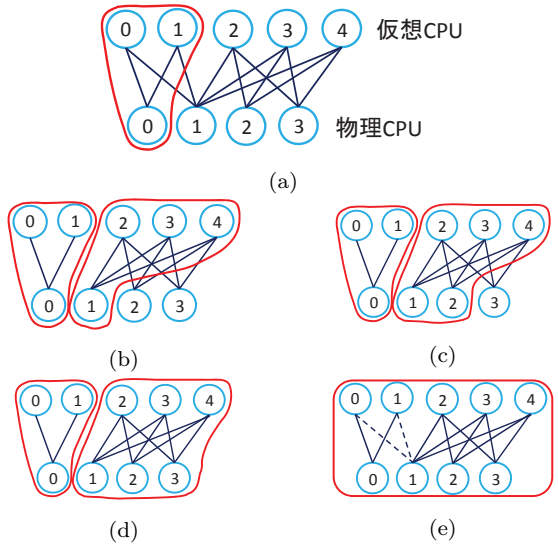


図 14 物理 CPU と仮想 CPU の分割例

4.3 利用可能な物理 CPU 数の見積もり

pCPU-Est は各仮想 CPU が利用できる物理 CPU の割合に基づいて，物理 CPU と仮想 CPU をいくつかのグループに分ける．そして，それを基に各 VM が利用できる物理 CPU 数を見積もる．このとき，同じグループ内のすべての仮想 CPU は，VM に設定された重みが同じであれば同じ割合の物理 CPU を利用することができるようにする．物理 CPU の割り当てに制約がなければ物理 CPU と仮想 CPU は全体で 1 つのグループを形成するが，仮想 CPU アフィニティが設定されていると複数のグループに分かれる可能性がある．明示的にアフィニティを設定していない場合でも，仮想化システムによって暗黙のうちに NUMA アフィニティが設定されることもある．NUMA アフィニティは NUMA を考慮した CPU 割り当てである．

pCPU-Est は図 14 のような物理 CPU と仮想 CPU のグラフを分割することでグループを作成する．物理 CPU と仮想 CPU は頂点となり，物理 CPU が仮想 CPU に割り当てられている場合にそれらは辺でつながれる．いくつかの物理 CPU からいくつかの仮想 CPU へのアフィニティが設定されている場合，各物理 CPU はそれらの仮想 CPU すべてにつながれる．図 14 は，物理 CPU 0, 1 から仮想 CPU 0, 1 へのアフィニティおよび，物理 CPU 1, 2, 3 から仮想 CPU 2, 3, 4 へのアフィニティが設定された場合の例である．

pCPU-Est のグラフ分割アルゴリズムは以下のようになる．まず，グラフ上でつながっている仮想 CPU 数が最小の物理 CPU を選択し，その物理 CPU とそれからつながっている仮想 CPU からなるグループを作成する．これは，グループの $\frac{\text{物理 CPU 数}}{\text{仮想 CPU 数}}$（物理 CPU の割合）がなるべく大きくなるようにするためである．図 14(a) では，物理 CPU 0 につながっている仮想 CPU 数が最小の 2 であるので，これらでグループを作成している．

次に，そのグループからつながっている物理 CPU のうち，つながっている仮想 CPU 数が最小の物理 CPU を選択し，その物理 CPU とそれからつながっている仮想 CPU を含めたより大きなグループを作成する．このグループの物理 CPU の割合が元のグループより大きいか等しければ新しいグループを採用する．これを繰り返してグループをできるだけ大きくする．新しいグループの物理 CPU の割合のほうが小さくなった場合は，グループからその外側につながっている辺を削除する．図 14(a) で作成されたグループの物理 CPU の割合が $\frac{1}{2}$ であるのに対し，物理 CPU 1 と仮想 CPU 2, 3, 4 を追加した新しいグループの物理 CPU の割合は $\frac{2}{5}$ と小さくとなるため，物理 CPU 1 と仮想 CPU 0, 1 をつなぐ辺を削除する．

この作業をグラフの残りの部分に対して繰り返す．図 14(a) で残っている物理 CPU の中でつながっている仮想 CPU 数が最小であるのは物理 CPU 1 なので，物理 CPU 1 と仮想 CPU 2, 3, 4 で図 14(b) のようにグループを作成する．このグループからつながっている物理 CPU の中で，つながっている仮想 CPU 数が最小であるのは物理 CPU 2 なので，図 14(c) のように物理 CPU 2 を追加したグループを作成する．元のグループの物理 CPU の割合が $\frac{1}{3}$ であるのに対し，新しいグループは $\frac{2}{3}$ と大きくなるため，新しいグループを採用する．

グラフ全体がいくつかのグループに分割されたら，作成されたグループをできるだけマージする．これは，局所的にグループ分けを行うだけでは正しいグラフ分割ができないためである．まず，削除された辺でつながれていた 2 つのグループを選び，その辺の仮想 CPU 側をグループ 1，物理 CPU 側をグループ 2 とする．グループ 2 の物理 CPU の割合のほうが大きいか等しければ，2 つのグループをマージする．これを削除された辺でつながれていたすべてのグループについて繰り返す．図 14(d) で作成された 2 つのグ

ループは削除された辺（図 14(e) の点線）でつながれていたため，マージできないかをチェックする．最初に作成されたグループ 1 の物理 CPU の割合は $\frac{1}{2}$ であるのに対し，後で作成されたグループ 2 は 1 と大きいため，これらのグループをマージして 1 つのグループを作成する．

グラフ分割アルゴリズムによって作成されたグループ内で，VM に設定された重みに応じて物理 CPU を仮想 CPU に均等に分配する．図 14 の場合，VM の重みが同じであれば各仮想 CPU に分配される物理 CPU の割合は $\frac{4}{5}$ となる. 次に，それぞれの VM の持つ仮想 CPU に分配される物理 CPU をすべて足し合わせる. この値が小数となる場合は小数点以下を切 り上げ，その値を VM が利用可能な物理 CPU 数とする. 例えば，図 14 の仮想 CPU 0, 1 が 1 つの VM に属する場合，その VM が利用可能な物理 CPU 数は 2 となる．

5. 実験

pCPU-Est による最適化の効果を調べる実験を行った．実験環境は 3 章の予備実験と同じものを用いた．

5.1 仮想 CPU 数の動的最適化の効果

VM への物理 CPU 割り当てを削減した場合に，BT，UA，LU の性能を先行研究と比較した．BT，UA，LU はそれぞれ 4.1 節のパターン 3，パターン 2，パターン 1 の代表的なベンチマークである．パターン 4 については未対応であるため，今回は実験の対象としていない．図 15 に pCPU-Est と先行研究を適用した際のそれぞれの相対実行時間を示す．BT では物理 CPU が 6〜12 個のときに pCPU-Est のほうが性能がよくなり，最大で 30%性能が向上した．UA では物理 CPU が 4〜8 個のときに pCPU-Est のほうが性能がよくなり，最大で 23%の性能向上が見られた．LU は先行研究と同等の性能となった．このように仮想 CPU 数の動的最適化を行うことで先行研究より性能が向上することが確認できた．

次に，pCPU-Est の動的最適化を適用した際の性能を 3.2 節で調査した際の最適性能と比較した．図 16 から BT では pCPU-Est が最適性能とほぼ同等の性能を達成できていることが分かる．一方，UA では物理 CPU が 12 個の時には pCPU-Est のほうが 19%性能が低かった．LU では物理 CPU が 4 個と 8 個の時に pCPU-Est のほうが性能が低かった．これらより，pCPU-Est の動的最適化はある程度，最適性能を実現できるが，一部，最適とならない場合があることが分かった．

その原因を調べるために，pCPU-Est の動的最適化によって決定された仮想 CPU 数を 3.2 節で調べた最適な仮想 CPU 数と比較した．その結果を図 17 に示す．BT では動的最適化により決定された仮想 CPU 数が最適な仮想 CPU 数より少し小さい場合があったが，性能にはほぼ影響しなかった．UA の場合は物理 CPU が 2 個と 6 個の時に最適な仮想 CPU 数より大きくなったが，これらの時にはほぼ最適性能と差がなかった．一方，物理 CPU が 12 個の時には動的最適化によって決定された仮想 CPU 数が最適な仮想 CPU 数よりも少しだけ小さくなったが，最適性能と大きな差となった．LU では物理 CPU が 4 個と 8 個の時に最適な仮想 CPU 数が同数よりも少ないため現在の手法では最適化が行えなかった．

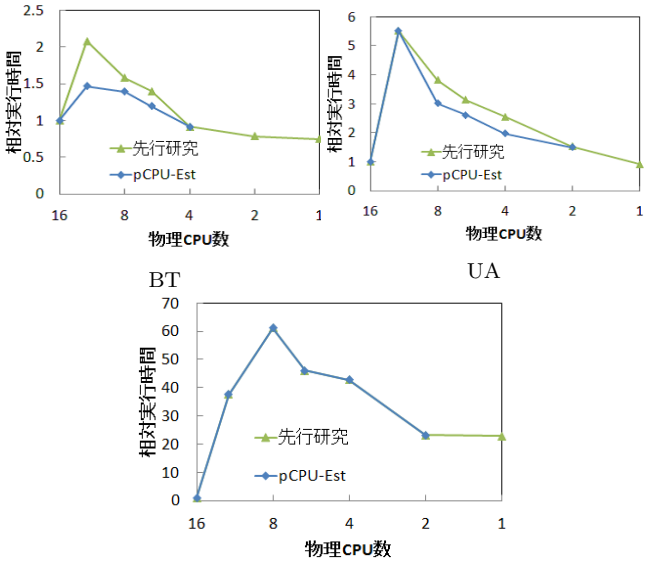


図 15 動的最適化による性能向上

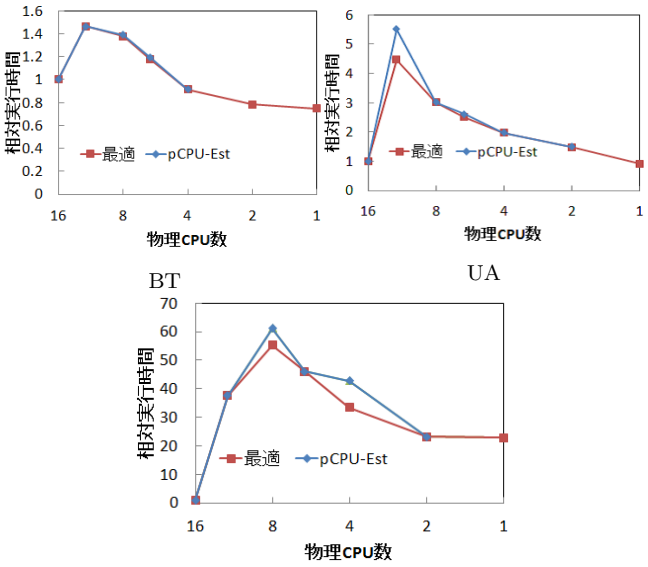


図 16 動的最適化後の性能と最適性能の比較

5.2 スレッド数の最適化の効果

物理 CPU 不足時の三つの対処に対して，pCPU-Est によるアプリケーションスレッド数の最適化を行った．その性能を先行研究と比較したものが図 18，図 19，図 20 である．図 18 は VM への物理 CPU 割り当てを削減した場合

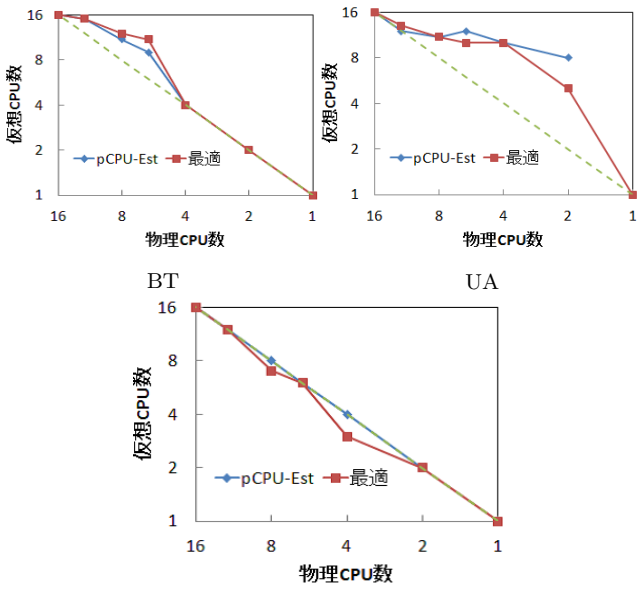


図 17　動的最適化によって決定された仮想 CPU 数

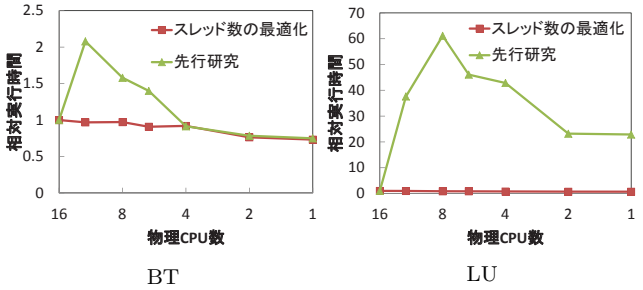


図 18　物理 CPU 削減時のスレッド数の最適化

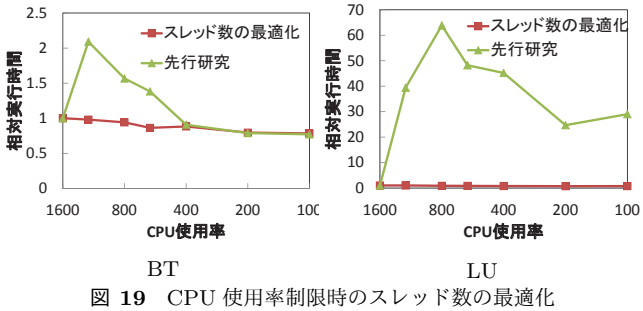


図 19　CPU 使用率制限時のスレッド数の最適化

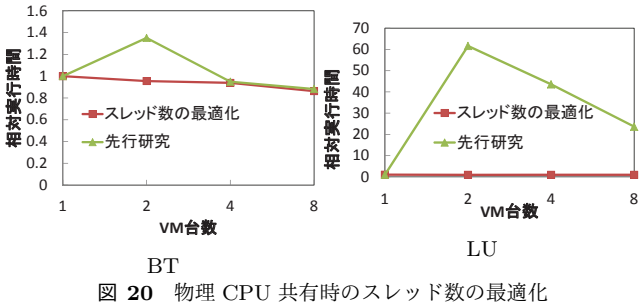


図 20　物理 CPU 共有時のスレッド数の最適化

であるが，BT ではほとんどの場合でスレッド数の最適化の方が効果があり，先行研究に比べて最大で 53%性能が向上した．特に，割り当てられる物理 CPU が多いときに効果が大きかった．LU でもスレッド数の最適化の方が効果が大きく，すべての場合で目標実行時間を下回った．先行研究と比べると最大で 99%の性能向上が見られた．CPU 使用率を制限した場合や物理 CPU を共有した場合でも，BT，LU ともにスレッド数の最適化の方が効果が大きいという結果になった．このようにスレッド数の最適化は非常に効果的であるが，アプリケーション依存であるため常に適用可能とは限らない．

6. 関連研究

VCPU-Bal[1] は VM の仮想 CPU 数を増減させる vCPU バルーニングを用いて，ロックホルダ・プリエンプションや vCPU スタッキングを防ぐ．これらの問題は，仮想 CPU とアプリケーションスレッドの二重のスケジューリングによって生じるため，VCPU-Bal では仮想 CPU が物理 CPU に一対一に割り当てられるように VM の仮想 CPU 数を静的に決定する．一方，pCPU-Est では仮想 CPU アフィニティを考慮して VM が利用可能な物理 CPU 数を算出し，最適な仮想 CPU 数を動的に決定する．FlexCore[3] は VCPU-Bal を実装したシステムであり，KVM 上で仮想 CPU の競合の検出，VM とハイパーバイザ間の効率のよい通信，仮想 CPU のホットプラグを実現している．

vScale[2] は VM の仮想 CPU 数を VCPU-Bal よりもきめ細かく調整することを可能にしている．VCPU-Bal と同様に，vScale も基本的には VM の重みに応じて仮想 CPU 数を決定する．しかし，VM が割り当てられた物理 CPU を使い切らなかった場合，その VM の仮想 CPU 数を減らし，他の VM の仮想 CPU 数を増やす．このような頻繁な仮想 CPU 数の変更を可能にするために，vScale では OS が迅速に仮想 CPU を再構成する機構を提供する．VCPU-Bal や vScale では複数の VM が物理 CPU を共有することを想定しているが，pCPU-Est では CPU 使用率を制限したり物理 CPU 割り当てを削減したりすることも想定している．

Intel の Pause Loop Exiting や AMD の Pause Filter はスピンウェイト中に連続して PAUSE 命令が実行されるとハイパーバイザに制御を移すことができる機能である．スピンウェイトが検出された際に，ハイパーバイザが別の仮想 CPU をスケジューリングすることでロックホルダ・プリエンプション等の問題を緩和することができる．しかし，これらの機能を用いることで頻繁にハイパーバイザに制御が移されると，その影響が小さくないことが報告されている [8]．また，NAS Parallel Benchmarks の LU のように，PAUSE 命令が実行されないスピンウェイトには効果がない．

コスケジューリング [9] は VM のすべての仮想 CPU を同時に物理 CPU にスケジュールすることにより，ロックホルダ・プリエンプションなどの問題を解決する．しかし，必要な数の物理 CPU がそろうまでスケジューリングを行

うことができず，高い優先度を持った仮想 CPU が低い優先度の仮想 CPU より後にスケジュールされる可能性もある．バランススケジューリング [6] は VM の各仮想 CPU を異なる物理 CPU にスケジュールすることで vCPU スタッキングを防ぐが、ロックホルダ・プリエンプションを防ぐことはできない．

7. まとめ

本稿ではまず，物理 CPU 不足時の様々な対処について並列アプリケーションの性能低下および先行研究の効果を調べた．その結果，先行研究はで用いられている仮想 CPU 数は必ずしも最適ではないことが分かった．そこで，本稿では実行時情報に基づいて VM の仮想 CPU 数を動的に最適化することで並列アプリケーションの性能を改善する pCPU-Est を提案した．pCPU-Est はアプリケーションスレッド数の最適化も提供する．また，これらの最適化のために，VM に仮想 CPU アフィニティが設定されている場合でも VM が利用可能な物理 CPU 数を見積もることができる．最適化の効果を確認する実験を行ったところ，仮想 CPU 数の動的最適化では先行研究と比べて最大 30%性能を向上させることができ，アプリケーションスレッド数の最適化では最大 99%性能を向上させられることが確認できた．

今後の課題は，4.1 節のパターン 4 のベンチマークに対して仮想 CPU 数の動的最適化が行えるようにすることである．CPU 使用率が増加すると性能が悪くなるベンチマークもあったため，４つのパターンに当てはまらない場合の最適化手法も考える必要がある．また，今回の実験では物理 CPU 削減時についてのみ動的最適化を適用したため，それ以外の対処での効果を確認する予定である．将来的には，NAS Parallel Benchmarks 以外のベンチマークでも効果が得られるか実験したいと考えている．

参考文献

[1] X. Song, J. Shi, H. Chen, and B. Zang ”Schedule processes, not VCPUs.” Proceedings of the 4th Asia-Pacific Workshop on Systems. 2013.
[2] L. Cheng, J. Rao, and F. Lau ”vScale: automatic and efficient processor scaling for SMP virtual machines.” Proceedings of the Eleventh European Conference on Computer Systems. 2016.
[3] M. Tianxiang and H. Chen ”FlexCore: Dynamic virtual machine scheduling using VCPU ballooning.” Tsinghua Science and Technology, Vol.20, No.1, pp.7–16, 2015.
[4] NASA Advanced Supercomputing Division : NAS Parallel Benchmarks，入手先 ⟨http://www.nas.nasa.gov/publications/npb.html⟩
[5] V. Uhlig, J. LeVasseur, E. Skoglund, and U. Dannowski ”Towards Scalable Multiprocessor Virtual Machines.” Virtual Machine Research and Technology Symposium. 2004.
[6] O. Sukwong and H. Kim ”Is co-scheduling too expensive for SMP VMs?.” Proceedings of the sixth conference on Computer systems. 2011.
[7] C. Xu, S. Gamage, H. Lu, R. Kompella, and D. Xu ”vTurbo: Accelerating Virtual Machine I/O Processing Using Designated Turbo-Sliced Core.” USENIX Annual Technical Conference. 2013.
[8] 山崎修平, 河野健二. ”Lock-Holder Preemption に対する Pause Loop Exiting の限界に関する調査” Vol.2016-OS-138, 2016.
[9] VMware, Inc.: *Co-scheduling SMP VMs in VMware ESX Server*, (2008).