B-034

マルチプロセッサ向け先読み同期プロトコル
A Look-Ahead Synchronization Protocol for Multiprocessors

石川 洋輔† 山崎 信行†
Yousuke Ishikawa Nobuyuki Yamasaki

1. 概要

リアルタイムシステムにおいて複数のタスクが資源を共有する場合，時間制約と資源の同期の両方を考える必要がある．本論文では，マルチプロセッサにおける固定優先度スケジューリングを対象として，資源共有を行う高優先度タスクの時間制約を最大限に維持することを目的とする．そのために，各タスクの資源アクセスを先読みし，高優先度タスクの時間制約を破綻させる可能性のある低優先度タスクの資源アクセスを遅延させる同期プロトコルを提案する．シミュレーションによる評価の結果，従来の同期プロトコルに比べて，本提案手法が高優先度タスクに対してスケジューリング可能性の向上及びジッタの変動の抑制を実現できたことを示す．

2. はじめに

リアルタイムシステムでは，タスクの時間制約を保証するために，スケジューリングアルゴリズムが重要となる．スケジューリングアルゴリズムは，主に動的優先度方式と固定優先度方式に分類される．多くの商用 OS では，優先度による予測性が高く，高優先度タスクのジッタが小さいという利点をもつ固定優先度方式が利用されている．たとえば，ヒューマノイドロボットでは，センサ/アクチュエータ制御といった時間を厳密に守る必要のある高優先度タスクのハードリアルタイムタスクと，画像処理といった時間制約を多少破ることを許容するソフトリアルタイムタスクが混在している．こういったシステムにおいては，高負荷時に低優先度タスクから必ずデッドラインミスをする予測性と高優先度の制御タスクが一定のタイミングで実行されるジッタが小さいことが求められる．

資源共有によりタスク間に依存関係が存在するタスクに対しては時間制約を保証するだけでなく，資源の整合性を保つ必要がある．シングルプロセッサにおける固定優先度方式では Priority Ceiling Protocol(PCP)[6]がしばしば利用される．PCP では共有資源について競合するタスクの中での最高優先度を優先度上限値として設定する．現在アクセスされている資源の優先度上限値の中の最高値をシステム優先度上限値として設定し，システム優先度上限値以下の優先度であるタスクの資源アクセスをブロックすることでデッドロックや多重ブロックを回避し，優先度逆転に対処している．しかしながら，低優先度タスクの資源アクセスによりシステム上限値が引き上げられた場合，高優先度タスクの資源アクセスは低優先度タスクの資源アクセスによってブロックされる可能性がある．

マルチプロセッサにおいてはシングルプロセッサと比較してスケジューリングが異なるだけでなく，異なるプロセッサで実行されるタスクで資源共有されるため，シングルプロセッサにおける同期プロトコルをそのまま利用することは難しい．また，マルチプロセッサにおいてはシングルプロセッサでは起きないリモートブロッキング[9]が起こる可能性がある．リモートブロッキングとは，あるタスクの資源アクセスが異なるプロセッサで実行されるタスクによってブロックされることを指す．マルチプロセッサにおいて，あるタスクは同一プロセッサ内のタスクだけでなく他プロセッサ内のタスクによってもブロックされる可能性があるため，固定優先度方式が有している優先度による予測性や高優先度タスクのジッタが小さいという利点を損なう原因となる．

ブロック時間を軽減する技術としてタスクの資源アクセスを先読みする技術が知られている[7]．しかしながら，これまでの先読み技術を適用した提案はタスクのデッドロックや多重ブロックの対処がされていなく，スケジューリング解析もされていないため，タスクの時間制約を事前に保証することが難しい．また，既存の提案はシングルプロセッサ向けの同期プロトコルであり，既存の提案をマルチプロセッサにそのまま適用することは難しい．

本論文では，資源共有を行う高優先度タスクの時間制約を最大限に維持することを目標として，各タスクの資源アクセスを先読みし，高優先度タスクの時間制約を破綻させる可能性のある低優先度タスクの資源アクセスを遅延させる同期プロトコルを提案する．本提案手法はシングルプロセッサだけでなく，マルチプロセッサにおいても有効であり，スケジューリングアルゴリズムには動的に生成されるタスクや非周期タスクが存在する可能性を考慮して RateMonotonic(RM)[3]を用いる．RM の優先度に基づいて資源アクセスを先読みすることで，実行中に新規に起動するタスクセットにも対応できる．また，資源アクセスの最悪実行時間とタスクがリリースしてから資源アクセスするまでの最短時間を定式化することで，先読みによる高優先度タスクの時間制約を維持する効果を高め，かつブロック時間と遅延時間の上限値からスケジュール可能性を保証できる．評価により，本提案手法に適用する先読み処理が資源共有を行う高優先度タスクの時間制約を最大限に維持でき，既存の固定優先度スケジューリングに対して有効であることを示す．

本論文の構成を以下に示す．第 3 章では，既存の同期プロトコルに関する関連研究，及び問題点について述べる．第 4 章では，システムモデルと用語の定義を述べる．第 5 章では，提案する同期プロトコルを述べ，第 6 章では，そのスケジュール可能性解析を行う．第 7 章では，シミュレーションによる評価結果を述べ，第 8 章では，先読み処理のオーバーヘッドの評価結果を述べる．第 9 章で結論と今後の課題について述べる．

3. 関連研究

先読み同期プロトコルとしては，文献[7]によるプロトコルが知られている．非リアルタイムタスクの資源アク

†慶應義塾大学 Keio University

セスが長い場合はリアルタイムタスクの資源アクセスを先読みし，リアルタイムタスクの実行の妨げになる資源アクセスを遅延させる．非リアルタイムタスクとリアルタイムタスク間において有効であるが，デッドロックや優先度逆転の対処が必要である．リアルタイムタスクのスケジュール可能性解析や，マルチプロセッサへの拡張も求められる．

固定優先度スケジューリングにおけるマルチプロセッサ向け同期プロトコルは，想定している仮定が異なる MPCP[9], DPCP[10], FMLP[1, 2]の 3 つの同期プロトコルが知られている．

MPCP はプロセッサ間に共有メモリがある状況を想定している．MPCP は PCP に基づいたプロトコルであるため，プロセッサ間で共有されない資源のアクセス方法は PCP と同じである．プロセッサ間で共有される資源の同期にはセマフォを利用する．資源アクセスがブロックされた場合は通常のセマフォと同様にタスクをサスペンドさせるが，プロセッサ間で共有される資源のアクセスを待つタスクは優先度順に起床する．また，プロセッサ間で共有される資源の優先度上限値と資源アクセスの優先度をプロセッサ間で共有されない資源の優先度上限値と資源アクセスの優先度よりも高く設定することでプロセッサ間で共有される資源のアクセスを優先し，リモートブロッキングの時間を軽減している．

DPCP はプロセッサ間に共有メモリがない状況を想定しており，MPCP とはプロセッサ間で共有される資源のアクセス方法が異なる．DPCP では，あるタスクが他プロセッサにある資源にアクセスしたい場合，そのプロセッサに配置された代理タスクが代わりに資源にアクセスすることで，プロセッサ間で共有される資源にアクセスする．

FMLP はタスクによって異なる資源のアクセス時間を資源ごとに短いアクセスか長いアクセスかの 2 つに分類でき，資源アクセスのネスト状況を事前に把握できることを想定している．この仮定により，FMLP はこれまでの提案では制限していたプロセッサ間で共有される資源アクセスのネストを許容できるプロトコルである．

固定優先度スケジューリングにおけるマルチプロセッサ向け同期プロトコルにおいても，低優先度タスクの資源アクセスにより高優先度タスクの資源アクセスは一度以上ブロックされる可能性がある．そのため，固定優先度方式が有している優先度による予測性や高優先度タスクのジッタが小さいという利点を損なう原因となる．

4. システムモデル

n 個のタスクはタスクセットにより構成され，m 個のプロセッサ，または $P=\{P_1,P_2,...,P_m\}$ コアにより実行される．本論文で用いるマルチプロセッサは，マルチスレッドプロセッサとマルチコアプロセッサに限定していることとし，プロセッサ間に共有メモリがあることとする．各タスク τ_i は周期タスクを想定しており，$\tau_i=(C_i,T_i)$ というタプルで表される．C_i は最悪実行時間を示し，T_i はタスクの周期を示す．τ_i は優先度 p_i の高い順に整列されているものとする．τ_i の利用率を $U_i=C_i/T_i$ で表す．タスクセット T に含まれるタスクの合計利用率を $U(T)=\sum_{\tau_i\in T}U_i$ で表す．すなわち，$U(T)$ はシステム全体の負荷を表す．ここで，$U(T)/m$ をシステム利用率と呼ぶことにする．タスクは一番負荷の低いプロセッサに割り当てるものとする．$\tau_{i,a}$ はタスク τ_i の a 番目のジョブを表し，リリース時刻 $r_{i,a}$，相対デッドライン $d_{i,a}$ で定義し，相対デッドライン $d_{i,a}$ は周期 T_i に等しいものとする．また，優先度継承などにより一時的に変更された優先度は p_i' により定義する．ある 2 つのタスク τ_i と τ_j が同一プロセッサに割り当てられている場合，2 つのタスクはローカルであると呼び，そうでなければリモートであると呼ぶこととする．

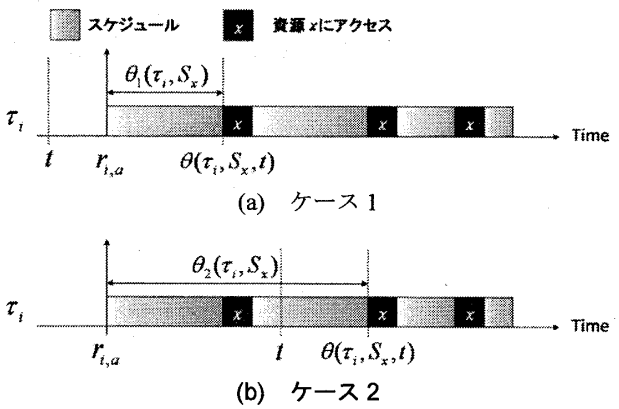


図 1 $\theta(\tau_i,S_x,t)$ の例

本論文では共有資源のアクセスに関して以下を仮定する．

- ある資源に対するアクセス時間はタスクによって異なる．
- タスクの資源アクセスの最悪実行時間が既知である．
- タスクがリリースされてから資源にアクセスするまでの最短時間が既知である．

共有資源 S_x は，排他的にアクセスされるものとする．S_x のロックにはセマフォを利用する．S_x の優先度上限値 $C(S_x)$ は，競合するタスクの中での最高優先度とし，あるプロセッサのシステム優先度上限値 $C(S^*)$ は，そのプロセッサ内で現在アクセスされている資源の優先度上限値の中での最高値で定義し，S^* はその資源を表わす．タスク τ_i が S_x に p 番目にアクセスするクリティカルセクションを $\varepsilon_p(\tau_i,S_x)$，その最悪実行時間を $\omega_p(\tau_i,S_x)$ により定義する．また，タスクがリリースしてから $\varepsilon_p(\tau_i,S_x)$ の実行を始めるまでの最短時間を $\theta_p(\tau_i,S_x)$ とする．$\theta(\tau_i,S_x,t)$ は下式で表わす．

$$\theta(\tau_i,S_x,t)=\min\{r_{i,a}+\theta_p(\tau_i,S_x)\geq t\}$$

つまり，$\theta(\tau_i,S_x,t)$ は τ_i がある時刻 t より後に S_x アクセスする時刻の中で最も早い時刻を表わす．$\theta(\tau_i,S_x,t)$ で表わす時刻は，τ_i が他タスクに影響を受けずに実行する場合の時刻とする．図 1 は 2 つのケースにおける $\theta(\tau_i,S_x,t)$ の例である．ケース 1 の場合，時刻 t において τ_i はスケジュールされておらず，次に τ_i が S_x にアクセスするのは $r_{i,a}+\theta_1(\tau_i,S_x)$ となる．ケース 2 の場合，時刻 t において τ_i はスケジュールされており，次に τ_i が S_x にアクセスするのは $r_{i,a}+\theta_2(\tau_i,S_x)$ となる．あるプロセッサに割り当てられたタスクからのみアクセスされる資源を内部資源 L_x，あるプロセッサに割り当てられたタスクだけでなく，違うプロセッサに割り当てられたタスクからもアクセスされる資源を外部資源 G_x として区別する．式の簡略化のため

Algorithm: MLA-PCP for local resource

$\tau_{i,a}$ **tries to enter** $\epsilon_p(\tau_i, L_x)$ **at time t:**
1. **if** $\{\tau_{j,b} \mid (\tau_j \in \mathcal{H}_i \cap \mathcal{R}(L_x)) \wedge (t \le r_{j,b} < t + \omega_p(\tau_i, L_x))\}$ exists **then**
2. suspend $\tau_{i,a}$;
3. **else**
4. **if** $p_i' > c(S*)$ **then**
5. let $\tau_{i,a}$ lock L_x and execute $\epsilon_p(\tau_i, L_x)$;
6. **else**
7. $\tau_{k,c} = S^*$ is currently locked by;
8. suspended $\tau_{i,a}$;
9. let $\tau_{k,c}$ inherit p_i';
10. **fi**
11. **fi**

$\tau_{i,a}$ **exits** $\epsilon_p(\tau_i, L_x)$ **at time t:**
12. let $\tau_{i,a}$ unlock L_x and wake up the highest-priority job that blocked on L_x;
13. update p_i';

図 2　内部資源に対する MLA-PCP のアルゴリズム

いくつかのタスクの集合を定義する．H_i は τ_i より優先度の高いタスクの集合，R_i は τ_i と同じ資源にアクセスするタスクの集合，$R(S_x)$ は S_x にアクセスするタスクの集合，P_i は τ_i とローカルであるタスクの集合を表す．

5. 提案手法

本論文では，マルチプロセッサにおける固定優先度スケジューリングを対象として，資源共有を行う高優先度タスクの時間制約を最大限に維持するための MultiprocessorLook-Ahead Priority Ceiling Protocol(MLA-PCP)を提案する．マルチプロセッサでは，内部資源と外部資源の 2 つの資源に対して同期を行わなければならない．そのため，本章では内部資源に対する MLA-PCP の同期方法と外部資源に対する MLA-PCP の同期方法をそれぞれ 5.1 節と 5.2 節で述べる．MLA-PCP は PCP をマルチプロセッサ向けに拡張し，それに先読みによる遅延のルールを適用した同期プロトコルである．また，タスクの資源アクセスの最悪実行時間とタスクがリリースされてから資源アクセスするまでの最短時間を定式化することで，先読み処理による高優先度タスクの時間制約を維持する効果を高めている．よって，文献[7]に比べて以下の優位性を有している．

- マルチプロセッサに対しても有効
- 優先度継承による多重ブロックの回避
- 優先度上限値によるデッドロックの回避
- 高優先度タスクのスケジューラビリティ

ここで，先読み処理によりタスクを遅延させない場合はすべての時間制約を保証することが可能なタスクセットであっても，先読み処理によりタスクを遅延させることで時間制約を保証できなくなることがある点に注意されたい．しかしながら，本提案手法では時間制約が破綻するタスクは低優先度タスクだけであり，高優先度タスクの時間制約は最大限に維持することが保証される．

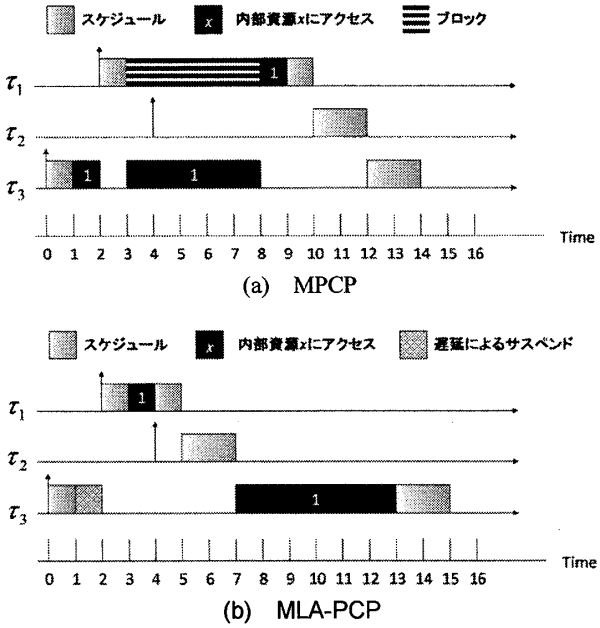


図 3　内部資源に対する 3 つのタスクの資源共有の例

5.1 内部資源に対する MLA-PCP の同期

図 2 に内部資源に対する MLA-PCP のアルゴリズムを示す．タスクが内部資源にアクセスを試みる際に各タスクの資源アクセスを先読みする．あるタスクが資源アクセスを遅延する条件は，資源アクセスの間にその資源を利用する高優先度タスクがリリースされることである(1 行目)．タスクを遅延させる場合，そのタスクをサスペンドさせる(2 行目)．遅延を行わない場合，PCP と同様にシステム優先度上限値をそのタスク自身の優先度が上回るならば，そのタスクは資源アクセスできる(4-5 行目)．タスクが資源アクセスできなければタスクをサスペンドさせ，優先度継承を行う(7-9 行目)．資源アクセスはプリエンプティブに実行し，高優先度タスクのプリエンプションを許容する．タスクは資源アクセスを終えると資源を解放し，その資源のアクセスを待つタスクの中の最高優先度タスクが起床し，優先度継承していた優先度が更新される(12-13 行目)．図 3 は内部資源に対する 3 つのタスクの資源共有の例を示しており，τ_i と τ_j は共に L_x を共有している．内部資源はあるプロセッサに割り当てられたタスクによってのみアクセスされる資源であるので，例はシングルプロセッサの例を示している．図 3(a)では，MPCP を利用した内部資源の共有の例を示している．MPCP の内部資源に対するアルゴリズムは PCP のアルゴリズムと同じである．$t=1$ において $\tau_{3,1}$ は資源 L_1 にアクセスし，$\varepsilon_1(\tau_3, S_1)$ が実行される．$t=2$ において $\tau_{1,1}$ がリリースされ，$\tau_{3,1}$ をプリエンプションして実行される．しかしながら，$t=3$ において $\tau_{3,1}$ により $C(S^*)=p_1$ となっているため，$\tau_{1,1}$ は L_1 にアクセスできず，$\tau_{3,1}$ に優先度 p_1 を継承させ $t=8$ までブロックされる．途中 $t=4$ において $\tau_{2,1}$ がリリースされるが，$\tau_{3,1}$ の優先度は τ_1 の優先度を継承しているため p_1 であるので実行されない．多重ブロックを防ぐことはできているが，$\tau_{1,1}$ は $p_1 > p_3$ にも関わらず $t=[3,8)$ の間ブロ

Algorithm: MLA-PCP for global resource

$\tau_{i,a}$ **tries to enter** $\epsilon_p(\tau_i, G_x)$ **at time t:**

1. **if** $\{\tau_{j,b} \mid (\tau_j \in \mathcal{H}_i \cap \mathcal{P}_i \cap \mathcal{R}(G_x)) \wedge (t \le r_{j,b} < t + \omega_p(\tau_i, G_x))\}$ exists **then**
2. suspend $\tau_{i,a}$;
3. **else if** $\{\tau_{j,b} \mid (\tau_j \in \mathcal{H}_i \cap \bar{\mathcal{P}}_i \cap \mathcal{R}(G_x)) \wedge (\theta(\tau_j, G_x, t) < t + \omega_p(\tau_i, G_x))\}$ exists **then**
4. suspend $\tau_{i,a}$;
5. **else**
6. **if** $(p'_i > c(S^*)) \wedge (G_x$ is not currently locked) **then**
7. let $\tau_{i,a}$ lock G_x and execute $\epsilon_p(\tau_i, G_x)$ non-preemptively;
8. **else**
9. suspend $\tau_{i,a}$;
10. **if** S^* is local resource **then**
11. $\tau_{k,c} = S^*$ is currently locked by;
12. let $\tau_{k,c}$ inherit p'_i;
13. **fi**
14. **fi**
15. **fi**

$\tau_{i,a}$ **exits** $\epsilon_p(\tau_i, G_x)$ **at time t:**

16. let $\tau_{i,a}$ unlock G_x and wake up the highest-priority job that blocked on G_x;
17. schedule $\tau_{i,a}$ preemptively;

図 4　外部資源に対する MLA-PCP のアルゴリズム

ックされてしまい，かつ $\tau_{2,1}$ の実行も遅れることになるため，τ_1 や τ_2 のジッタの増大の原因となる．図 3(b)では，MLA-PCP を利用した内部資源の共有の例を示しており，図 3(a)と同様のタスクセットである．$t=1$ において $\tau_{3,1}$ は先読み処理を行う．$\tau_{1,1}$ は L_1 にアクセスし，$p_1 > p_3$ である．また，$r_{1,1}=2$，かつ $\omega_1(\tau_3, L_1)=6$ であり，$t \le r_{1,1} < t+\omega_1(\tau_3, L_1)$ となるため，$\tau_{3,1}$ はサスペンドする．$t=2$ において $\tau_{1,1}$ がリリースされ，L_1 のアクセスをブロックされることなく $t=5$ には実行を終える．$t=4$ においてリリースされていた $\tau_{2,1}$ も，$t=5$ には実行を始め，$t=7$ にはジョブの実行を終える．図 3(a)で見られた MPCP による同期では，$\tau_{1,1}$ は $t=[3,8)$ の間ブロックされたが，MLA-PCP では一度もブロックされることなく実行を終えることができる．$\tau_{1,1}$ は $t=10$ から $t=5$ へ，$\tau_{2,1}$ は $t=12$ から $t=7$ へ終了時刻を早めることができる．

5.2 外部資源に対する MLA-PCP の同期

図 4 に外部資源に対する MLA-PCP のアルゴリズムを示す．タスクが外部資源にアクセスを試みる際に各タスクの資源アクセスを先読みする．あるタスクは次の 2 つの場合に資源アクセスを遅延する(1,3 行目)．

(1) 資源アクセスを試みるタスクとローカルである高優先度タスクが外部資源のアクセスの終了までにリリースされ，かつその資源を利用する場合
(2) 資源アクセスを試みるタスクとリモートである高優先度タスクが外部資源のアクセスの終了までにその資源を利用する場合

(1)の遅延判定は内部資源と同様の判定方法である．(2)の遅延判定は高優先度タスクが資源アクセスする時刻を知

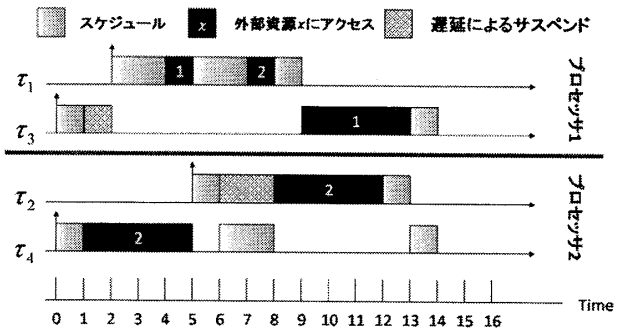


図 5　MLA-PCP を利用した 2 つのプロセッサにおける 4 タスクの外部資源の共有の例

る必要があるが，正確にその時刻を知ることは難しい．そのため，この場合の資源アクセスする時刻は，高優先度タスクが他タスクに影響を受けずに実行する場合に資源にアクセスする時刻とする．タスクを遅延させる場合，そのタスクをサスペンドさせる(2,4 行目)．遅延を行わない場合，PCP と同様にシステム優先度上限値をそのタスク自身の優先度が上回るならば，そのタスクは資源アクセスできる(6 行目)．また，外部資源アクセスはノンプリエンプティブに実行する(7 行目)．外部資源アクセスをプリエンプティブに実行することでタスクセットによっては高優先度タスクのジッタを小さくできる場合もある．しかしながら，外部資源アクセスはボトルネックになりやすく速やかに実行を終わらせる必要があるため，本提案手法では外部資源アクセスはノンプリエンプティブに実行する．外部資源アクセスはノンプリエンプティブに実行されるため，外部資源にアクセス中のタスクに対して優先度継承をする必要はなくなる．タスクが資源アクセスできなければタスクをサスペンドさせ，ブロックしたタスクのアクセス資源が内部資源の場合は優先度継承を行う(9-12 行目)．タスクは資源アクセスを終えると資源を解放し，その資源のアクセスを待つタスクの中の最高優先度タスクが起床する(16 行目)．解放したタスクはプリエンプティブに実行を戻す(17 行目)．また，ある外部資源アクセスをネストするタスクを同プロセッサに割り当てることで外部資源アクセスをネストすることもされることも可能である．

図 5 は，MLA-PCP を利用した 2 つのプロセッサにおける 4 タスクの外部資源の共有の例を示しており，τ_1 と τ_3 はプロセッサ 1，τ_2 と τ_4 はプロセッサ 2 に割当てられ，G_1 は τ_1 と τ_3，G_2 は τ_1 と τ_2 と τ_4 に共有されている．$t=1$ においてプロセッサ 1 では $\tau_{3,1}$ が先読み処理を行う．$p_1 > p_3$ である $\tau_{1,1}$ は，G_1 にアクセスし，かつ $\tau_{3,1}$ とローカルである．また，$r_{1,1}=2$，かつ $\omega_1(\tau_3, G_1)=4$ であり，$t \le r_{1,1} < t+\omega_1(\tau_3, G_1)$ となるため $\tau_{3,1}$ は遅延する．プロセッサ 2 では $\tau_{4,1}$ が先読み処理を行う．$\tau_{1,1}$ と $\tau_{2,1}$ は G_2 にアクセスし，かつ $\tau_{4,1}$ よりも高優先度である．$\tau_{4,1}$ とローカルである $\tau_{2,1}$ において $r_{2,1}=5$，かつ $\omega_1(\tau_4, G_2)=4$ であり，$t \le r_{2,1} < t+\omega_1(\tau_4, G_2)$ とはならない．$\tau_{4,1}$ とリモートである $\tau_{1,1}$ においても $\theta(\tau_1, G_2, 1)=7$ であり，$\theta(\tau_1, G_2, 1) < t+\omega_1(\tau_4, G_2)$ とはならない．そのため，$\tau_{4,1}$ は遅延せず $\varepsilon_1(\tau_4, G_2)$ を実行する．また，その実行はノンプリ

エンプティブであるため $t=5$ まで実行される．$t=2$ において $\tau_{1,1}$ がリリースされ，$\tau_{3,1}$ をプリエンプションして実行される．$t=4$ において $\varepsilon_1(\tau_1,G_2)$ は $\tau_{3,1}$ が遅延したことにより，ブロックされることなく実行することができる．また，$t=6$ において $\tau_{2,1}$ が先読み処理を行う．$p_1>p_2$ である $\tau_{1,1}$ は，G_2 にアクセスし，かつ $\tau_{2,1}$ とリモートである．また，$\theta(\tau_1,G_2,6)=7$ であり，$\theta(\tau_1,G_2,6)<t+\omega_1(\tau_2,G_2)$ となるため $\tau_{2,1}$ は遅延し，代わりに $\tau_{4,1}$ が実行される．$t=7$ において $\varepsilon_1(\tau_1,G_2)$ は $\tau_{2,1}$ が遅延したことにより，ブロックされることなく実行することができる．$t=[6,7)$ の間，遅延による優先度逆転が起きてしまっているが，その分 $\tau_{1,1}$ の優先度逆転の時間を短くすることできる．

6. スケジューリング解析

本論文の主要な目的は先読み処理が資源共有を行う高優先度タスクの時間制約を最大限に維持でき，既存の固定優先度スケジューリングに対して有効であることを示すことである．そのため，システム全体の時間検証を議論することは本論文の範囲外とし，スケジューラビリティ保証のためにタスクに起こる遅延時間とブロック時間の上限を解析することに焦点を当てる．あるタスク τi に起こる遅延時間とブロック時間は，それぞれ D_i，B_i により上限を定めることができる．本章では D_i，B_i を求めるにあたり，単純化のためいくつかの用語を定義する．あるタスク τ_i は内部資源と外部資源にアクセスすることができる．τ_i がアクセス可能な内部資源の集合を L^i とし，L^i と L_x の積集合の要素である内部資源にアクセスする最も長いクリティカルセクションを $\omega_i^*(L_x)$ で示す．τ_i と τ_j が共にアクセス可能な内部資源の集合を $L^{i,j}$ とする．τ_i がアクセス可能な外部資源の集合を G^i とし，G^i と G_x の積集合の要素である外部資源にアクセスする最も長いクリティカルセクションを $\omega_i^*(G_x)$ で示す．τ_i が外部資源 G_x にアクセスする回数を $N_i(G_x)$ とする．τ_i と τ_j が共にアクセス可能な外部資源の集合を $G_{i,j}$ とする．

6.1 遅延時間

遅延時間の上限は下式により表すことができ，

$$D_i = D_{i,1} + D_{i,2} \tag{1}$$

それぞれの要素は次のように表わされる．

$$D_1 = \sum_{\tau_j \in P_i \cap H_i \cap R_i} \max \left\{ \begin{array}{l} \left(\left\lceil \frac{T_i + \omega_i^*(L_x)}{T_j} \right\rceil \times \omega_i^*(L_x) \right), \\ \left(\left\lceil \frac{T_i + \omega_i^*(G_y)}{T_j} \right\rceil \times \omega_i^*(G_y) \right) \mid L_x \in L^{i,j}, G_y \in G^{i,j} \end{array} \right\}$$

$$D_2 = \sum_{\tau_j \in P_i \cap H_i \cap R_i} \max \left\{ \left\lceil \frac{T_i + \omega_i^*(G_x)}{T_j} \right\rceil N_j(G_x) \times \omega_i^*(G_x) \mid G_x \in G^{i,j} \right\}$$

$D_{i,1}$ は τ_i とローカルであるタスクによる遅延時間の上限である．τ_j による遅延は τ_i の周期と先読みの最大時間の間に τ_j がリリースされる回数だけ起こる可能性がある．$D_{i,2}$ は τ_i とリモートであるタスクによる遅延時間の上限である．τ_j により遅延は τ_i の周期と先読みの最大時間の間に τ_j がリリースされる回数と，τ_j が外部資源にアクセスする回数の積だけ起こる可能性がある．

6.2 ブロック時間

ブロック時間の上限は下式により表わすことができ，

$$B_i = B_{i,1} + B_{i,2} + B_{i,3} + B_{i,4} + B_{i,5} \tag{2}$$

ここで，

$$\alpha_i = \max \left\{ \omega_j^*(L_x) \cup \omega_j^*(G_y) \mid \tau_j \in \overline{H}_i \cap P_i \cap R_i, \; L_x \in L^j, G_y \in G^j \right\}$$

$$\beta_i(G_x) = \max \left\{ \omega_j^*(G_x) \mid \tau_j \in \overline{H}_i \cap \overline{P}_i, \forall G_x \right\}$$

とすると，それぞれの要素は次のように表わされる．

$$B_1 = \alpha_i$$

$$B_2 = \sum_{G_x \in G^i} N_i(G_x) \times (\alpha_i + \beta_i(G_x))$$

$$B_3 = \sum_{\tau_j \in P_i \cap H_i} \left\lceil \frac{T_i}{T_j} \right\rceil C_j$$

$$B_4 = \sum_{\tau_j \in P_i \cap H_i \cap R_i} \max \left\{ \begin{array}{l} \left(\left\lceil \frac{T_i + \omega_i^*(L_x)}{T_j} \right\rceil \times \alpha_i \right) \cup \\ \left\{ \left\lceil \frac{T_i + \omega_i^*(G_y)}{T_j} \right\rceil \times \left(\alpha_i + \beta_i(G_y) \right) \right\} \\ \qquad \mid L_x \in L^{i,j}, G_y \in G^{i,j} \end{array} \right\}$$

$$B_5 = \sum_{\tau_j \in P_i \cap H_i \cap R_i} \max \left\{ \begin{array}{l} \left\lceil \frac{T_i + \omega_i^*(G_x)}{T_j} \right\rceil N_j(G_x) \times \\ \qquad \left(\alpha_i + \beta_i(G_x) \right) \mid G_x \in G^{i,j} \end{array} \right\}$$

$B_{i,1}$ は τ_i とローカルである低優先度タスクによって，τ_i のリリース時，または最初の資源のアクセス時に起こるブロック時間の上限である．$B_{i,2}$ はある外部資源にアクセスを試みる際，その外部資源が τ_i とリモートである低優先度タスクにアクセスされている場合に起こるブロック時間の上限である．τ_i が外部資源の解放を待つ間，τ_i とはローカルな低優先度タスクが資源にアクセスすることで τ_i をブロックする可能性がある．このブロックは τ_i の外部資源のアクセス回数だけ起こる可能性がある．$B_{i,3}$ は τ_i がある外部資源の解放を待つ間，τ_i とリモートである高優先度タスクの実行により起こるブロック時間の上限である．外部資源のアクセスはノンプリエンプティブに行われるが，解放されたタスクの優先度は変動しないため，解放されたタスクよりも高優先度なタスクが実行され続ける可能性がある．$B_{i,4}$ は τ_i とローカルである高優先度タスクにより遅延することで起こるブロック時間の上限である．遅延している間，τ_i とローカルである低優先度タスクが資源アクセスすることで τ_i はブロックされる可能性がある．また，外部資源アクセスにおいて遅延する場

合，外部資源の解放待ちによるブロックが遅延した回数だけ起こる可能性がある．$B_{i,S}$ は τ_i とリモートである高優先度タスクにより遅延することで起こるブロック時間の上限である．遅延している間，τ_i とローカルである低優先度タスクが資源アクセスすることで τ_i はブロックされる可能性がある．また，外部資源アクセスを遅延することで外部資源の解放待ちによるブロックが遅延した回数だけ起こる可能性がある．

6.3 スケジュール可能性解析

あるタスク τ_i の遅延時間の上限 D_i とブロック時間の上限 B_i により，MLA-PCP を利用した RM のスケジュール可能性判定式を求めることができる．解析は，文献[5]を基にしている．定理に必要な補題を以下に示す．

補題 1. タスク間に依存関係が存在しないタスクにおいて，すべてのプロセッサに対し，個々に割り当てられた n' 個のタスクが下式を満たすならば RM でスケジュール可能である．

$$\sum_{i=1}^{n'} \frac{C_i}{T_i} \le n'(2^{1/n'} - 1) \tag{3}$$

証明. 文献 11) により証明されている． □

定理 1. タスク間に依存関係が存在するタスクにおいて，すべてのプロセッサに対し，個々に割り当てられた n' 個のタスクが下式を満たすならば MLA-PCP を利用した RM でスケジュール可能である．

$$\sum_{i=1}^{i} \frac{C_i}{T_i} + \frac{D_i + B_i}{T_i} \le i(2^{1/i} - 1) \quad \forall i, 1 \le i \le n' \tag{4}$$

証明. すべての τ_i に対して式(4)が満たされていると仮定する．$n' = i$，$C_i^* = (C_i + D_i + B_i)$ と置くことで，式(3)を満たす．たとえ $(C_i + D_i + B_i)$ を実行したとしても，τ_i はデッドラインを守ることができる．つまり，τ_i はたとえ C_i だけ実行したとしても D_i や B_i によって遅れることが可能であり，デッドラインを守ることができる．よって，定理 1 が証明された． □

7. シミュレーション評価

本章では，MLA-PCP のシミュレーションによる評価を行う．文献 [7]によるプロトコルはノンリアルタイムタスクとリアルタイムタスク間を対象としており，優先度タスクを対象としている本提案手法と比較することは難しい．また，本提案手法はプロセッサ間に共有メモリがあり，ある資源に対するアクセス時間がタスクによって異なることを許容していることから，DPCP や FMLP とは想定している仮定が異なるため，評価対象は MPCP とする．あるタスクセットにおいて高優先度タスクからある割合のタスクを見たとき，そのタスクすべてがデッドラインミスを起こさずにスケジュール可能であったならばスケジュール成功であったとする．スケジュールしたタスクセット数に対するタスクのスケジュール成功数の割合をスケジュール成功率とする．優先度による予測性を示す評価指標にはスケジュール成功率を用いる．高優先度タスクのジッタが小さいことを示す評価指標には，タスクの周期に対するタスクの絶対終了ジッタの割合を用いる．最高優先度タスクを用いて高優先度タスクのジッタを評価する．実機ではシミュレーションのように多様なタスクセットを生成してスケジュール成功率やジッタを評価することは難しいため，シミュレーションにより評価を行う．

7.1 評価環境

プロセッサ数 m は，$m = \{1,2,4\}$ とする．タスクセットに含まれるタスクの利用率の最大値 $U\max$ は，$U\max = \{0.5,1.0\}$ とする．タスクの利用率は $\{0.01, U\max\}$ の範囲とし，タスクを生成する．タスクセットに含まれるタスクの利用率の合計は $Utotal$ で表わし，システム利用率は $Utotalm$ で定義する．資源共有の時間を含むタスクの最悪実行時間は，文献 [14]を参考に [50.0,500.0]μ sec の範囲から生成する．

各プロセッサにはそれぞれ 6 個の内部資源を用意する．$m = \{2,4\}$ の場合，各プロセッサに与えられた内部資源に加えて 6 個の外部資源を用意する．$m = 1$ の場合，タスクは 6 個の内部資源にアクセスすることができる．また，$m = \{2,4\}$ の場合，タスクは 6 個の内部資源と 6 個の外部資源にアクセスすることができる．資源アクセスに関する環境は次のように決定する．まず，アクセス時間 [1.0,5.0]μ sec の範囲の資源アクセスを各タスクに割り当てる．資源アクセスを割り当てる回数は，各タスクが一回以上の資源アクセスを行うように [1,3] 回の範囲から無作為に選択される．次にアクセス時間 [30.0,40.0]μ sec の範囲の資源アクセスをあるタスクセットの中から無作為に選択したタスクに割り当てていく．資源アクセスを割り当てる回数は，[1, 10] 回の範囲から無作為に選択され，選択した回数になるまでタスクに資源アクセスを割りあてていく．アクセス資源は，タスクがアクセス可能な資源から無作為に決定する．文献[13]で紹介されているヒューマノイドロボット HRP のアプリケーションでは，資源アクセスをネストするような複雑なプログラムはあまり見られない．そのため，評価には資源アクセスのネストは含めないこととする．

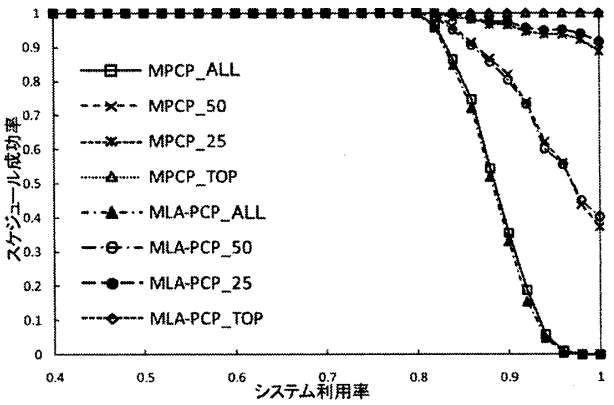
各アルゴリズムに対するスケジュール成功率は以下の 4 段階に分けて評価する．

- タスクセットに含まれるすべてのタスクに対するスケジュール成功率
- タスクセットに含まれるタスクのうち優先度順に上位 50% のタスクに対するスケジュール成功率
- タスクセットに含まれるタスクのうち優先度順に上位 25% のタスクに対するスケジュール成功率
- タスクセットに含まれる最高優先度タスクに対するスケジュール成功率

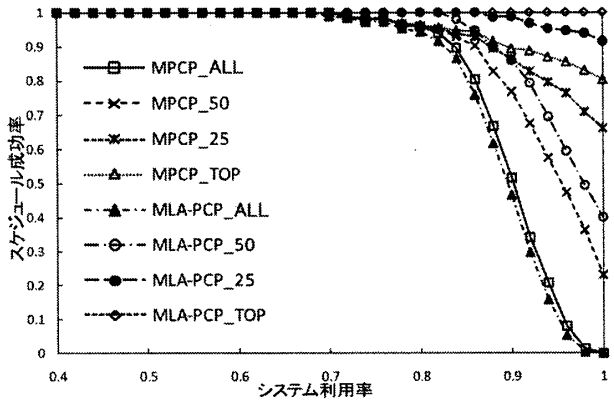
実験結果の表記では，MPCP{ALL,50,25,TOP} と MLA-PCP{ALL,50,25,TOP} は，それぞれ MPCP と MLA-PCP に対する上記 4 段階のスケジュール成功率を指すこととする．

7.2 優先度による予測性に関する評価結果

図 6 は，$m = 1$，$U\max = \{0.5,1.0\}$ における MPCP と MLA-PCP のスケジュール成功率を示している．図 6(a)では両プロトコル共に若干の差は見られるものの，最高優先度タスクのスケジュール成功率は，システム利用率が高い場合でも 1.0 を維持しており，MPCP と MLA-PCP は

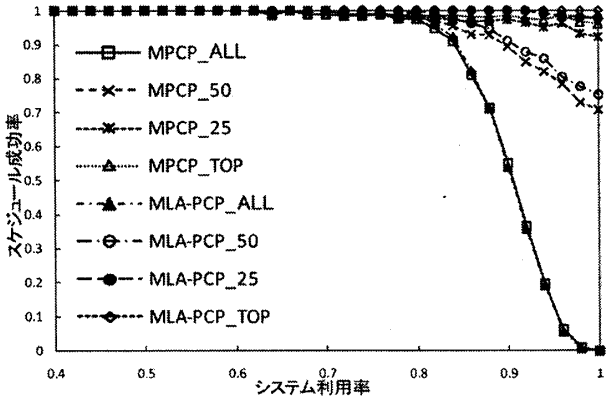


(a) Umax=0.5

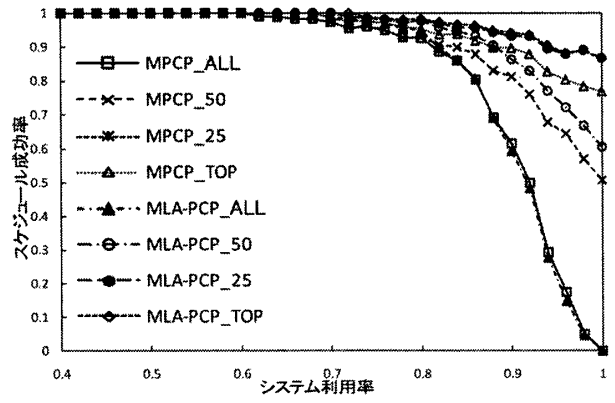


(b) Umax=1.0

図 6 m=1 におけるスケジュール成功率



(a) Umax=0.5



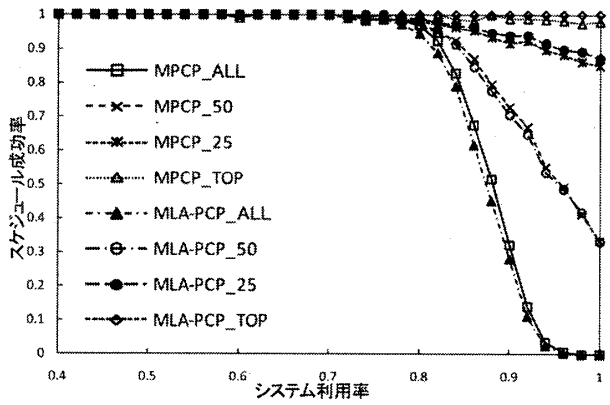
(b) Umax=1.0

図 7 m=2 におけるスケジュール成功率

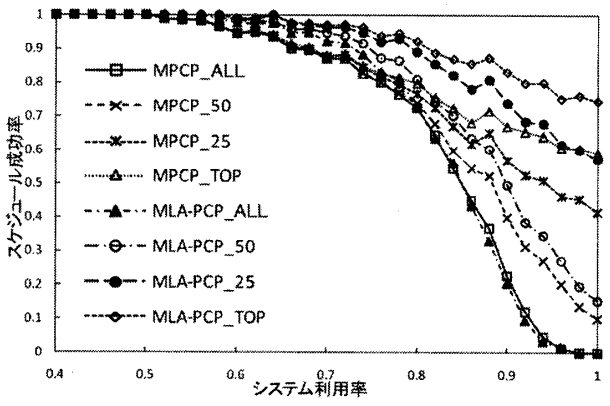
近いスケジュール成功率を示している．しかしながら，図 6(b)では MLA-PCP は MPCP よりもタスクセットすべてのタスクのスケジュール成功率は若干下がるものの，他の割合のタスクのスケジュール成功率は高い数値を示している．特に MLA-PCP の最高優先度タスクのスケジュール成功率は，システム利用率が高い場合でも 1.0 を維持している．そのため，内部資源に対して MLA-PCP は MPCP と比べて高優先度タスクの時間制約を維持できているといえる．

図 7 と図 8 は，$m=\{2,4\}$，$U\max=\{0.5,1.0\}$ における MPCP と MLA-PCP のスケジュール成功率を示している．図 7(a)と図 8(a)では，MLA-PCP は MPCP よりもタスクセットすべてのタスクのスケジュール成功率は若干下がるものの，上位 25%のタスク，最高優先度タスクのスケジュール成功率は高い数値を示している．特に MLA-PCP の最高優先度タスクのスケジュール成功率はシステム利用率が高い場合でも 1.0 を維持している．また，図 7(b)と図 8(b)では，MLA-PCP と MPCP 共にシステム利用率が高くなるにつれて最高優先度タスクのスケジュール成功率が減少してしまっている．しかしながら，MLA-PCP は MPCP よりも上 50%，25%のタスク，最高優先度タスクのスケジュール成功率が高い数値を示している．さらに，タスクセットすべてのタスクのスケジュール成功率においても MLA-PCP より MPCP が高い数値を示している部分がみられる．主な要因は，外部資源のアクセスをノンプリエンプティブに実行したことで外部資源アクセスの負荷を軽減したことが考えられる．内部資源だけでなく外部資源に対しても MLA-PCP は MPCP と比べて高優先度タスクの時間制約を維持できているといえる．

本論文の目標は，資源共有を行う高優先度タスクの時間制約を最大限に維持することである．評価結果では，タスクセットすべてのタスクのスケジュール成功率を下げることなく，高優先度タスクのスケジュール成功率を高めることができている．タスク間の資源共有は固定優先度スケジューリングにおける利点である優先度による予測性を損なう原因であるが，本提案手法の適用する先



(a) Umax=0.5



(b) Umax=1.0

図 8 m=4 におけるスケジュール成功率

読み処理によって優先度による予測性を維持し，資源共有を行う高優先度タスクの時間制約を最大限に維持できているといえる．

7.3 最高優先度タスクのジッタに関する評価結果

優先度による予測性に関する評価結果により，最高優先度タスクのスケジュール成功率が MLA-PCP では 1.0 を維持できたが，MPCP では維持できてなかった状況に着目する．内部資源と外部資源それぞれのアクセスに対して先読みの有効性を示すため，$\{m=1, U\max=1.0\}$，$\{m=4, U\max=0.5\}$ に着目しジッタを評価する．

図 9 は，$m=1$ における (a) MPCP，(b) MLA-PCP の最高優先度タスクの絶対終了ジッタ/周期を示しており，環境は図 6(b)と同じである．図 9(a)と図 9(b)を比較すると，MPCP でのシステム利用率の増加に伴い，1.2 強まで散乱していたジッタが MLA-PCP では 0.4 程度まで改善している．

図 10 は，$m=4$ における(a) MPCP，(b) MLA-PCP の最高優先度タスクの絶対終了ジッタ/周期を示しており，環境は図 8(a)と同じである．図 10(a)と図 10(b)を比較すると，MPCP でのシステム利用率の増加に伴い，1.4 強まで散乱していたジッタが MLA-PCP では 0.8 程度まで改善している．

8. オーバーヘッド評価

本章では，MLA-PCP においてタスク実行の先読み処理にかかるオーバーヘッドの評価を行う．タスク実行の先読み処理のオーバーヘッドは，あるタスクが資源にアクセスを試みる際に各タスクの資源アクセスを先読みし，資源アクセスを遅延するかどうかを判断するまでに要する時間である．評価では，n 個未満のタスクの先読みによって遅延するかどうかを判断できた場合も n 個すべてのタスクを先読みし，タスク実行の先読み処理に要する最悪の場合の実行時間を測定する．

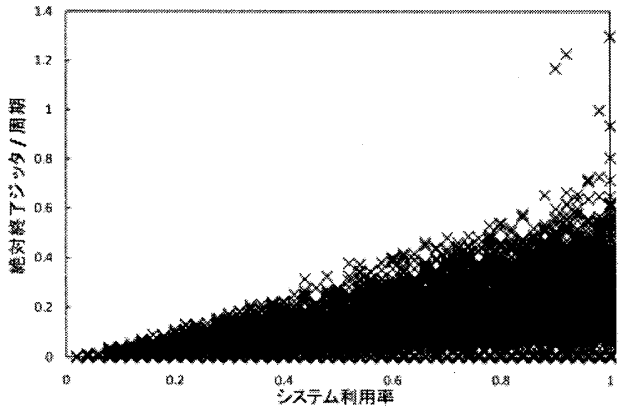
評価対象は固定優先度スケジューラ実行にかかるオーバーヘッドとする．スケジューラ実行は以下の 4 つの実行時間に大別できる．

- タイマ割込みサービスルーチンにかかる時間
- 次に実行すべきジョブを選択するのにかかる時間
- 実際のタスク切り替えにかかる時間
- タスクを正しい時刻にリリースさせるために行われる処理の時間

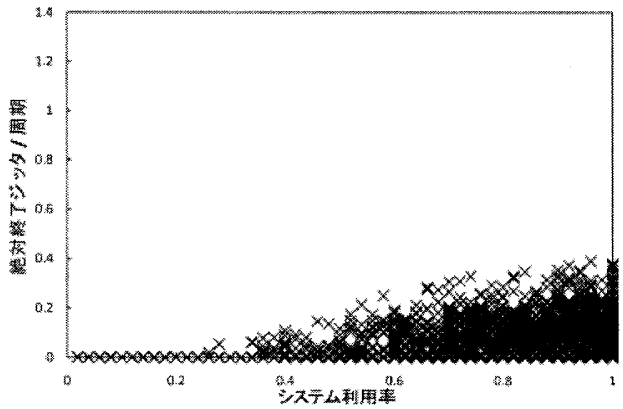
スケジューラはタイマ割込み時，タスクの実行終了時に実行される．また，通常のセマフォによる同期では，ある高優先度タスクが資源アクセスをブロックされた場合にタスクをサスペンドさせる必要があるため，スケジューラ実行によって次に実行すべきジョブを選択し，実際にタスクを切り替える必要がある．先読み処理によりブロックされずに実行できた場合，この 2 つの時間を削減できると考えられるため，評価対象のスケジューラ実行にかかるオーバーヘッドは，次に実行すべきジョブを選択し，実際にタスクを切り替えるのにかかる時間とする．評価指標はサイクル数を用いる．

8.1 評価環境

評価には Responsive Multithreaded Processor(RMTP)[8]を用いる．RMTP は Simultaneous Multithreading(SMT)[4]において各スレッドに優先度を付与し，高優先度スレッドには優先的に CPU 資源を割り当てる機能を有しているが，本論文では各スレッドは公平に CPU 資源が割り当てられることを仮定しているため，各スレッドの優先度は同一とした．また，同時実行可能なスレッド数は 4 とした (4 way SMT)．OS は我々の研究室で開発しているヒューマノイドロボットに利用することを目的とした軽量 OS を利用する．評価に用いるタスクの周期は，$T=10$[msec]で実行されるタイマ割込みの周期に対して {10T,20T,30T}msec から選択し，タスクの最悪実行時間は {50,100,200}μsec から選択する．タスク数 n は {1, 5, 10, 15, 20, 25,30} に設定し，選択可能タスクから組み合わせ可能なすべてのタスクセットに対して評価する．実験評価時間は各タスクセットについて 30 秒とし，すべてのタスクは時間制約を破綻させることなくスケジュール可能である．

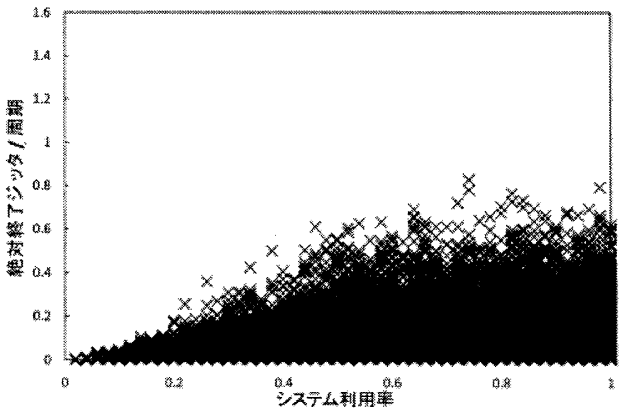


(a) m=1，Umax=1.0 における MPCP

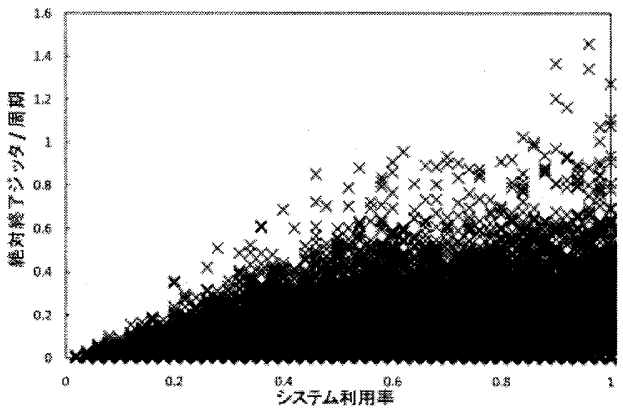


(b) m=1，Umax=1.0 における MLA-PCP

図 9 m=1 における最高優先度タスクのジッタ/周期



(a) m=4，Umax=0.5 における MPCP



(b) m=4，Umax=0.5 における MLA-PCP

図 10 m=4 における最高優先度タスクのジッタ/周期

8.2 オーバーヘッド評価結果

図 11 は，タスク数に応じた固定優先度スケジューラ実行とタスク実行の先読み処理のオーバーヘッドの平均値を示している．タスク数の増加に伴い，先読みしなければならないタスクが増えるため先読み処理にかかる時間が増加している．また，タスク数が増えることでタスクキュー内のタスク数も増えるため，タスクキューの検索にかかる計算量が増加する．そのため，タスク数の増加に伴い，スケジューラ実行時間も増加している．評価に用いた実験環境では，タスク数が 20 までの場合，先読み処理の時間はスケジューラ実行時間と比較して小さい値を示している．また，タスク数が 20 を越えると，先読み処理の時間はスケジューラ実行時間と比較して大きい値を示している．

今回の測定ではスケジューラ実行時間にタイマ割込みサービスルーチンにかかる時間とタスクをリリースさせるための処理の時間を含めていない．そのため，タイマ割込みによって再スケジュールする場合，スケジューラ実行時間は図 11 に示す時間よりも単純に増加する．また，図 11 に示すタスク実行の先読み処理の時間は n 個すべてのタスクを先読みした時間であり，あるタスクが先読み処理によって遅延する場合，先読みするタスクは実際は n 個以下のタスクで済む．そのため，スケジューラ実行と先読み処理のオーバーヘッドの大小の分岐点は，20 よりも大きくなることが予測させる．

通常のセマフォによる同期では，タスクが資源アクセスをブロックされた場合に，タスクをサスペンドさせる必要があるため，資源アクセスがブロックされる度にタスクがスケジューラを呼び出す必要がある．タスク実行の先読み処理により，高優先度タスクの資源アクセスのブロックを減らすことで，タスクをサスペンドさせるためのスケジューラ実行が削減できるため，高優先度タスクに対するスケジューラ実行の影響を削減することができる．

また，周期タスクの場合は資源アクセスのブロック時だけでなく，タスクのリリース時やタスクの終了時にもスケジューラ実行が必要である．資源の整合性を保ちつつタスクの時間制約を保証するためにはスケジューラが頻繁に呼び出される必要がある．ヒューマノイドロボットが固定優先度スケジューラを用いて資源の整合性を保ちつつ，時間制約を保証していることから，固定優先度スケジューラ実行の時間はタスクの時間制約の保証に影響しない程度の実行時間であると考えられる．そのため，20 という比較的多くのタスクが存在しても，先読み処理にかかるオーバーヘッドはスケジューラ実行のオーバーヘッドと同程度であり，実験から得られた知見の範囲では実用的に利用できると考えられる．タスク数の増加に

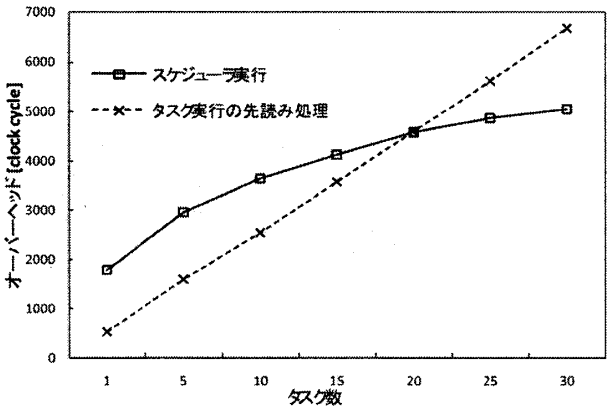


図 11　タスク実行の先読み処理とスケジューラ実行の平均オーバーヘッド時間

対する対応として，先読みする対象タスクの数を限定することも可能である．

9. 結論

本論文では，マルチプロセッサにおける固定優先度スケジューリングを対象として，資源共有を行う高優先度タスクの時間制約を最大限に維持する同期プロトコル MLA-PCP を提案した．本提案手法は，ヒューマノイドロボットなどに代表される正確なタイミングが求められるハードリアルタイムタスクと，時間制約を多少破ることを許容するソフトリアルタイムタスクが混在するシステムに有効であるといえる．また，あるタスクに起こるブロック時間と遅延時間の上限値によりスケジュール可能性解析を行うことで，タスクの時間制約を保証することができた．シミュレーション評価では，従来手法である MPCP と比較し，内部資源と外部資源の両方に対してタスクセット全体のスケジュール成功率の著しい減少を回避しつつ高優先度タスクのスケジュール成功率を向上でき，かつジッタの最大値を小さくすることができた．オーバーヘッド評価では，先読み処理の時間が大きなオーバーヘッドにはならないことを示した．

以下に今後の課題を示す．本論文で提案した同期プロトコルでは，タスクを遅延させるためにサスペンドさせていた．これは優先度逆転やアイドルする時間を許すことになるため，この時間の良否の解析は重要であると考えられる．また，スケジューラビリティ解析において本論文では，システム全体の時間検証を議論することは範囲外とした．そのため，充足可能性判定[15]やモデル検査[12]のような技術を応用することでスケジューラビリティ解析の正確性を向上させる予定である．先読み処理においては資源アクセスがネストされる状況を想定していないため，その状況においても先読み処理が有効であることを示す予定である．また，本論文ではパーティション方式に基づく固定優先度スケジューリングを対象としたが，今後は動的優先度スケジューリングにおいても先読み処理が有効であることを示す予定である．

謝辞

本研究は科学技術振興機構 CREST の支援によるものであることを記し，謝意を表す．また，本研究の一部は文部科学省グローバル COE プログラム「環境共生・安全システムデザインの先導拠点」に依るものであることを記し、謝意を表す。

参考文献

[1]B.Brandenburg and J.Anderson.: A Comparison ofthe M-PCP, D-PCP, and FMLP on LITMUS RT , the12th International Conference on Principles of Distributed Systems, pp. 105-124 (2008).

[2]B.Brandenburg and J.Anderson.: An Implementation of the PCP, SRP, D-PCP, M-PCP, and FMLP Real-Time Synchronization Protocols in LITMUS RT , Real-Time Computing Systems and Applications, pp. 185-194 (2008)

[3]C.Liu and J.Layland.: Scheduling algorithms for multiprogramming in a hard real-time environment, Journal of the ACM, Vol. 20, pp. 46-61 (1973).

[4]D.M.Tullsen, S.J.Eggers, H.M.Levy, J.L.Lo andR.L.Stamm: Exploiting Choice: Instruction Fetchand Issue on an Implementable Simultaneous Multithreading Processor, Annual International Symposium on Computer Architecture, pp.191-202 (1996).

[5]L.Sha, R.Rajkumar and J.P.Lehoczky.: Priority Inheritance Protocols: An Approach to Real-Time Synchronization., IEEE Transactions on Computers, pp. 39(9):1175-1185 (1990).

[6]M.Chen and K.Lin.: Dynamic Priority Ceiling: A Concurrency Control Protocol for Real-Time Systems., The Journal of Real-Time Systems, pp. 2:325-346 (1990).

[7]M.Dai, T.Matsui and Y.Ishikawa.: A Light Lock Management Mechanism for Optimizing Real-Time and Non-Real-Time Performance in Embedded Linux, Embedded and Ubiquitous Computing, pp.162-168 (2008).

[8]N.Yamasaki.: Responsive Multithreaded Processor for Distributed Real-Time Systems, Journal of Robotics and Mechatronics, Vol. 17, pp. 130-141(2005).

[9]R.Rajkumar.: Real-time Synchronization Protocols for Shared Memory Multiprocessors., Distributed Computing Systems, pp.116-123 (1990).

[10] R.Rajkumar, L.Sha and J.P.Lehoczky.: Real-time synchronization protocols for multiprocessors., Real-Time Systems Symposium, pp. 259-269 (1988).

[11] S.K.Dhall and C.L.Liu.: On a Real-Time Scheduling Problem, Operations Research, Vol. 26, pp. 127-140 (1978).

[12] T.Amnell, E.Fersman, L.Mokrushin, P.Pettersson and W.Yi: TIMES: a Tool for Schedulability Analysis and Code Generation of Real-Time Systems, Formal Modeling and Analysis of Timed Systems: First International Workshop, FORMATS 2003, LNCS,Vol. 2791, pp. 60-72 (2003).

[13] T.Matsui, H.Hirukawa, N.Yamasaki, H.ishikawa,S.Kagami, F.Kanehiro, H.Saito and T.Inamura: Distributed Real-Time Processing for HumanoidRobots, 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp. 205-210 (2005).

[14] U.Dev, H.Leontyev and J.Anderson.: Efficient Synchronization under Global EDF Scheduling on Multiprocessors., 18th Euromicro Conf. on Real-Time Systems, pp. 75-84 (July 2006).

[15] 足立 正和，末次 亮，結縁 祥治，手嶋 茂晴，佐野 範佳: 充足可能性判定に基づくリアルタイムシステムのスケジューリング解析, 組込みシステムシンポジウム論文集, Vol. 9 号, pp. 41-49 (2008).