UMLによる組込みシステムの性能評価シミュレーションのための共有メモリーアクセスの競合の評価手法

河原　亮[†]　小野康一[†]　中田武男[†]

組込みシステムでは性能向上などを目的としてマルチプロセッサーの採用が多くなっている．並列処理による性能向上の阻害要因の 1 つに共有メモリーへのアクセスの競合があり，開発初期段階で評価できることが望ましい．本論文では UML モデルのシミュレーションに適用可能なメモリーアクセスの競合の評価手法を提案する．本手法では, 既存製品からの差分開発を想定し, 共有メモリーのアクセス競合の影響を，計測で得られた粗粒度のパラメーターをもとに統計的に近似してシミュレーションで評価する．今回，従来手法では指数関数オーダーだったこの近似手法のスケーラビリティを向上させた．またベンチマークを使った実験との精度の比較の結果，最大で 12.6%，さらにバースト的アクセスを考慮することにより最大で 4.2%程度まで改善することがわかった．

Method for estimating contention of memory access applicable to UML-based performance simulation

Ryo Kawahara,[†] Kouichi Ono[†] and Takeo Nakada[†]

Multi-processing is now adopted in embedded systems. One of inhibitors of the system performance in such systems is the contention of a shared memory access, which should be assesed at early stages in development. We propose a method for estimating memory access contention applicable to performance estimation method based on UML model. In our method, it is evaluated using statistical approximation with parameters obtained from measurements of an existing system. In this paper, we report an improvement on the scalability of the approximation. We also report the result of comparison with benchmark experiments to find the maximum error of 12.6%. This is reduced to 4.2% if we take into account burstiness of memory accesses.

1. はじめに

組込みシステムが複雑，大規模になるにつれ，性能要件を満たすように適切なアーキテクチャーを選択することが難しくなっている．システムテストの段階になって性能面の問題が判明し，その原因がアーキテクチャー設計にあった場合，大きな手戻りとなる．市場の開発期間の短縮要求に対応するためには開発プロセスの上流工程で性能評価を行うことが重要である．

グラフィック処理や印刷，音声処理など，大規模なデータを扱うソフトリアルタイム，ないしは非リアルタイムのアプリケーションにおいて，実行時間は重要な性能指標のひとつである．このようなシステムでは，アプリケーションの実行時間短縮のためにマルチプロセッサーの採用が増えている．しかし共有メモリーに対するアクセスの競合により予想されるほど性能向上が達成できないことがある．必要とする性能を得るためには，処理の順序やメモリーの構成の見直しなどハードウェアおよびソフトウェアにまたがる変更が必要になることがある[1)]．これを防ぐためにはシステムアーキテクチャーの設計段階で共有メモリーの効果を考慮した性能評価を行っておくことが望ましい．

性能評価には一般に専用のモデリング手法が用いられる．開発プロセスの上流工程でよく用いられるものにキューイングネットワーク (Queuing Network, QN) モデルなどの数理的な手法がある[2)]．モデルが抽象的であるため，アーキテクチャー設計とアプリケーションの性能との直接的な関係を見出しにくい．

メモリーアクセスの競合を評価できる手法として SystemC[3)] をはじめとする電子システムレベル (Electronic System Level, ESL) のシミュレーション手法がある．これらのシミュレーションでは競合を評価するにはメモリーアクセスのタイミングをモデルに記述す

† 日本アイ・ビー・エム株式会社 東京基礎研究所
IBM Research - Tokyo

る必要がある．このようなモデルをシステム全体について作成するのは高コストであり，QN などに比べ計算時間も長くなる．また，システムアーキテクチャーの設計段階ではメモリーへのアクセスタイミングを詳細には見積もれないことが多い．ISS(Instruction Set Simulator)[4] も同様の課題がある．

一方近年，ソフトウェアのみならずシステムアーキテクチャー全体を粗い粒度で記述するために，Object Management Group (OMG) の Unified Modeling Language (UML) が適しているという考えが出てきており，言語の拡張が進められている[5][6]．これにあわせ，拡張された UML を用いて性能評価を行うことにより，性能評価と開発プロセスの一体化が模索されている[7]．

UML を用いた性能評価手法にはいくつかタイプがあるが，その中で Cortellessa らはソフトウェアモデルとプラットフォームモデルという 2 つの分離したモデルからシミュレーションを構成する手法を提案している[8]．プラットフォームモデルはリソースの競合を計算する．ソフトウェアモデルは時間付きの実行可能 UML であるため，QN などと違いシミュレーションの粒度は設計情報の粒度に一致している．またこの関心事項の分離によって，プラットフォームの設計空間をソフトウェアの変更量を抑えながら探索することができる．これらはシステムアーキテクチャー設計の段階での使用に有利な特徴と考えられる．

しかしながらこの研究には以下に挙げる課題がある．

(1) 実行時間の計算に必要な，シミュレーションステップ毎の処理時間などの性能パラメーターを取得する方法が明示されていない
(2) プラットフォームモデルにおいてプロセッサー以外のリソース競合について具体的な計算方法を与えていない

そこで著者らはまず前者について，多くの組込み製品が既存製品からの差分開発であることに着目し，粗粒度のトレースドリブンシミュレーションを行うことを提案した[9][10]．外部プローブで関数やタスクの実行時間などを精度よく計測し，得られた実行トレースから性能パラメーターを抽出する．

本研究は後者の課題について，共有リソースの中でも最も共通性が高く利用頻度が多いと思われる共有メモリーについての具体的な計算方法を提案するものである．すなわち，システムアーキテクチャー設計段階の粗い粒度のモデルに適した，共有メモリへのアクセス競合を考慮したシミュレーション手法を提案する．2 節で関連研究を，3 節で本手法の詳細を述べる．4 節でこれをサイクル精度シミュレーション，および FPGA 評価ボードと組込み向けベンチマークソフトウェアを用いた実験結果と比較し，5 節でそれを議論する．6 節はまとめと今後の課題である．

2. 関 連 研 究

この節では共有メモリーの競合を評価する既存手法を紹介する．

トレースドリブンシミュレーションを使ったメモリーの性能評価手法は多く知られている[11]．しかし多くがキャッシュメモリーの設計を目的としていたため，詳細なメモリーアクセスのトレース情報を用いていた．このようなトレースはサイズが大きく，シミュレーション速度が遅いため今回のような粗い粒度のモデルによる性能評価に向かない．

一方，大きなデータを用いない手法としてメモリーアクセスの統計情報のみをつかって確率モデルで推定する方法がある．その出発点は，Amdahl の法則[12] である．競合のない場合の実行時間を T，競合がある場合を T' とすると，以下のようになる．

$$T' = T((1-P) + AP) = T(1 + (A-1)P) \quad (1)$$

ここで，P は実行時間のうち，メモリー競合の影響を受けている時間の割合であり，A は競合によって伸びる処理時間の平均倍率である．これらは調停器の挙動を表した確率モデルの分析により与えられる．メモリーアクセスの競合の度合いは，どのプロセッサーからアクセスがあるかの組み合わせによって異なる．初期の Hoogendoorn[13] および Mudge ら[14] の理論では，この組み合わせのパターンをすべて数え上げる必要があった．一般的にはこの組み合わせの数はプロセッサー数 N に対して指数関数オーダーで増加するため，多プロセッサーのシステムではシミュレーション時間が増大してしまい，彼らの結果も帯域利用率が全プロセッサーで等しい場合に限られていた．一方組込みシステムはしばしばヘテロジニアスな構成であり，プロセッサー毎の帯域利用率の要求量も大きく異なるのが普通である．また同様にヘテロジニアスな構成でよく用いられる優先度つきの調停への拡張も示されていなかった．

計算量の問題に関して，Smilauer[15] は，調停方法が FCFS (first-come first-serve) の場合にキューイングネットワークモデルの分析で用いられる Mean Value Analysis (MVA) を適用することにより，計算量を N の冪オーダーまで削減した．これは競合中のアクセスの待ち行列の長さをその平均値で置き換えるという近似である．しかしながら，ラウンドロビンや優先度つ

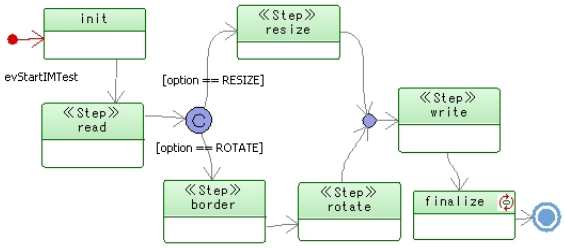

図 1 性能評価シミュレーションのための実行可能な UML モデルの例 (状態マシン図).

きなどその他の調停方法には適用できておらず，また適用範囲が帯域利用率が低い領域に限られていた．

著者らの先行研究[16)17)] では，組込みシステムへの適用を念頭に，様々な調停方法に拡張可能な形で確率モデルを定式化し，具体的にラウンドロビン調停の場合の計算方法を示して，UML ベースの性能評価シミュレーターに応用した．また Shibata らの研究[18)] においては，アクセスパターンの組み合わせの詳細な検討を行い，優先度つきの調停を含む様々な調停方法での計算式を提案している．しかしいずれの場合も，プロセッサー数の増加に対して計算量がスケールしないという問題があった．

一方，Bobrek ら[19)] は式 (1) および確率モデルを用いずに，既存のシステムの様々なアクセスパターンから学習によって T と T' の関係を回帰モデルであらわすという手法を提案している．この場合は対象の調停方式をブラックボックスにできるため様々なデバイスに拡張可能であり，また回帰モデルのパラメーターは少ないためシミュレーション計算も高速である．ただし，事前の学習に多くのコストを要するという問題がある．

本論文の貢献は以下のとおりである．MVA と同様の伸び率 A の平均値での近似を利用して，計算量の削減を実現すると同時に，ラウンドロビン調停など他の調停方法にも拡張可能な形で適用する．またこのラウンドロビン調停において比較的高い帯域利用率でも適用できることを示す．これにより本手法を，組込みシステムの差分開発を念頭に，上流工程において既存システムから低コストで計測可能な粗い粒度のパラメーターからの高速なシミュレーションによる性能評価に適用可能なものとする．

3. 提 案 手 法

3.1 UML ベースの性能評価手法

まず入力となるモデルの例を図 1 に示す．この図は UML モデルの状態マシン図である．

本手法の基礎となるのは実行可能な UML モデルである．実行可能な UML によるモデリングと機能検証用のシミュレーション実行の機能はベンダー各社から提供されているツールですでに実現されているが，計算リソースの使用時間やリソース競合によるその時間の変化など性能評価機能は一般的には実現していない．

UML の実行において，実行の単位となる最小のステップは操作 (operation) やアクティビティ図のアクション，状態マシン図の状態や遷移に割り当てられるアクションである．これらはプログラムコードでは関数やメソッド，ブロックの単位に相当し，プロセッサー命令やメモリーアクセスのトランザクションを大量に含む粗い粒度の振舞いを一塊として記述する．

本手法では，UML モデルの個々のステップ i には負荷パラメーター $s_i = (T_i, U_i)$ が与えられている．ここで T_i, U_i はそれぞれメモリー競合がないときの処理時間と帯域利用率の要求量である．メモリアクセスの帯域利用率 U_i は，以下のように定義する．

$$U_i = \frac{N_{\mathrm{R}i}}{T_i W_{\mathrm{R}}} + \frac{N_{\mathrm{W}i}}{T_i W_{\mathrm{W}}}. \tag{2}$$

ここで W_{R} と W_{W} はそれぞれ，読み出しメモリーアクセスと書き込みメモリーアクセスのスループット (単位時間当たりにメモリーが処理できるメモリーアクセス回数の最大値) である．同様に，$N_{\mathrm{R}i}$ と $N_{\mathrm{W}i}$ はそれぞれ，ステップ内のメモリーアクセスの回数である．これらのパラメーターは多くの場合，リファレンスシステム (既存の類似システム，差分開発ならば前のバージョンの製品など) から計測によって精度良く求めることができる．

このパラメーターは UML のステレオタイプを使ってモデルのステップにアノーテートされる．図 1 では各状態についた <<Step>> ステレオタイプがそれに相当する．モデルには別途外部設定ファイルが付属し，そこでどのオブジェクトがどのプロセッサーで実行され，どの共有メモリーに配置されているかの割り当て情報が定義されている．

本シミュレーション手法では，UML モデルのシミュレーション実行中に共有リソースごとに同時に実行されたステップの帯域利用率の要求量を調べ，リソース競合の度合いを計算する．これを基に競合のない場合のステップの処理時間 T_i から競合下の各ステップの処理時間 T_i' を計算し，ステップのスケジューリングを行う．スケジューリング方法は通常の時間つきイベントドリブンシミュレーションと同じである．システムの全体の実行時間は実行パスに沿って T_i' を合計したものとなる．

実装上は，モデル駆動開発ツールの UML モデルから C++へのソースコード生成機能を利用している．この時，各ステップごとに，ステレオタイプでアノーテートされた負荷パラメーターをシミュレーションフレームワークに受け渡す通信用のコードが挿入される．フレームワークでは上記のリソース競合が計算される．

本論文の目的はメモリーアクセスの競合を考慮した場合のステップ処理時間 T_i' の具体的な計算手法を与えることにある．

3.2 メモリーアクセス競合の評価の方針

メモリアクセスが複数のプロセッサーから同時に発生することをここでは衝突と呼ぶ．式 (1) を拡張して，様々な衝突のケースを数え上げることを明示するとプロセッサー i で実行されているステップの処理時間 T_i は次の式のようになる．

$$T_i'/T_i = \sum_{c}^{\text{num.patterns}} A_i(c)P_i(c_i)P_i(c|c_i), \quad (3)$$

ここで $c = (c_1, \cdots, c_N)$ は各プロセッサーの衝突パターンを表すインデックスである．例えば 1 アクセス 1 サイクルとすると，各プロセッサーの状態 c_j はアクセス中かそうでないかに応じて $c_j = 1, 0$ の 2 値をとり，N 個のプロセッサーの場合には衝突なしの場合を含めて 2^N 個の衝突のパターンがある．現在の多くのシステムではバーストアクセスや DRAM (Dynamic Random Access Memory) のレイテンシーのため，1 アクセスに複数サイクルかかる．この場合はさらに何サイクル目で衝突するかという状態数が掛け算でかかるためパターン数はさらに増える．

$P_i(c_i)$ はプロセッサー i が状態 c_i である確率であり，アクセス中ならば $P_i(c_i = 1) = U_i$，そうでないならば $P_i(c_i = 0) = 1 - P_i(1)$ となる．これは，帯域利用率 U の定義式 (2) より，U_i はメモリーサブシステムがプロセッサー i からのメモリーアクセスの処理に費やしている時間の割合を表すからである．

$P_i(c|c_i)$ はプロセッサー i が状態 c_i のときその他のプロセッサーが衝突パターン c であるという条件付確率である．

$A_i(c)$ はある衝突パターンのとき，プロセッサー i の 1 回のアクセスが何倍に伸びるかを表す．一般に調停方式に依存する．その平均値

$$a_i(c_i) = \sum_{\{c_j\}, \forall j \neq i} A_i(c)P_i(c|c_i) \quad (4)$$

は式 (1) の A と以下の関係にある．

$$a_i(c_i = 1) = A \quad (5)$$

またアクセスがない場合には競合は起きないので自明に以下の式が成立する．

$$a_i(c_i = 0) = 1 \quad (6)$$

これらを代入すると式 (3) は式 (1) に帰着する．

なお，アクセス競合状態におけるステップの帯域利用率を U_i' とすると，処理するべきメモリーアクセスの正味の時間 $U_iT_i = U_i'T_i'$ は競合の有無によらず同じであることから以下の関係式が成立し，帯域利用率の変化も求めることができる．

$$U_i'/U_i = T_i/T_i' \quad (7)$$

よって，競合の計算は式 (4) を特定の調停方式の場合に計算する問題になる．しかし，衝突のパターン数が多いことや詳細な衝突パターンの確率がわからないことから，解析的に実行することは特別な場合を除いて不可能である．

本手法では，個々のメモリーアクセスのタイミング情報を用いずに，性能パラメーターからメモリーアクセスの競合を評価するために，メモリーアクセスについて Mudge らと同様の以下の 2 つの近似を行う[14]．

最初の近似は，メモリーアクセスの競合が無いならば，ステップ内のメモリーアクセスは時間的に一様かつランダムに発生しているということである．

次の近似は，アクセス競合による調停がない限り，同時に共有メモリーにアクセスするステップどうしで，メモリーアクセスのタイミングに相関が無いとすることである．

最初の近似は一般的には正しくないことが知られている．科学技術計算のアプリケーションの場合，メモリーアクセスが集中する時間帯と少ない時間帯が混在するようなパターンが知られており，バースト的アクセス (bursty access, burstiness) と呼ばれている[20]．この点については 5 節で言及する．

後者の近似は，AMP(Asymmetric Multi-Processing) などヘテロジニアスな環境を想定している．プロセッサーごとに性質の異なるタスクを実行していればアクセスに強い相関があることは考えにくいからである．SMP でもタスク並列型の処理ではプロセッサーごとにばらばらのタイミングで処理が開始されるため，同様であると考えられる．一方，この近似が成り立たなくなる要因としてはデータ並列処理がある．大きな配列への繰り返し処理を分割してプロセッサーに割り当てる場合，各プロセッサーのアクセスパターンはきわめて類似しており，相関が大きくなる．またデータ依存性により細かい粒度で同期を繰り返すことも，相関を生じる原因となる．したがって，本手法に適した用途としては，データ並列性をもたず同期が UML のレベルで明示されておりステップ内部に隠れていないシ

ステムとなる．

これらの近似の範囲内で，メモリーアクセスの競合による処理時間の変化を計算する．プロセッサーごとの独立性の仮定からその他のプロセッサーの衝突確率 $P_i(c|c_i)$ は以下のように各プロセッサーの状態の確率の積で書ける．

$$P_i(c|c_i) = \prod_{j \neq i}^{N} p_j(c_j) \tag{8}$$

ここで $p_j(c_j)$ はあるプロセッサー $j \neq i$ が状態 c_j である確率である．先ほどの議論からおおよそ $p_j(c_j) = U_j$ であると考えられるが，この U_j は競合がないときの利用率の要求量であり，新規のメモリーアクセスによる衝突だけを見ている．実際には，他のプロセッサーの以前のアクセスが調停により待ち状態に入っているところに，対象としているプロセッサー i から新たなアクセスがあって衝突が発生することがあり，その分，衝突の確率は大きくなる．

そこで以下の量を考える．

$$U^* = \{(T' - T) + TU\}/T' \tag{9}$$

これは，元々あるステップがメモリーアクセスにかけていた時間 TU と，調停により待ち状態に入っている時間 $T' - T$ の合計が，競合状態でのステップ処理時間 T' に占める割合である．

この影響を取り込む簡単な近似として，ここではその他のプロセッサー $j \neq i$ の確率を U_j^* で表すこととする．

$$p_j(c_j) = \begin{cases} 1 - U_j^*, & (c_j = 0), \\ U_j^*, & (c_j = 1). \end{cases} \tag{10}$$

これを式 (8)，(3) に代入することにより，T_i' を得る．ただし，式 (9) より U_q^* にはすでに T_i' への依存性が入っているため，これは代数方程式である．

本手法の実装においては，逐次代入によりこれを解いている．すなわち，式 (10) の右辺に初期値として $U_j^* := U_j$ を代入する．そして式 (3) から得られた T_i' を式 (9) に代入して U_j^* を更新する．この U_q^* を再び式 (10) の右辺に代入する．これを値が収束するまで繰り返す．

次に式 (4) の $a_i(c_i)$ であるが，一般的に $A_i(c)$ は c の非線形な関数であり平均の解析的な計算は難しい．平均値分析 (MVA) では，この中に現れる待ち行列長を先に何らかの平均値に置き換える近似により c への依存性を消去し，式 (4) に存在する指数関数オーダーのパターン数の和の計算を回避する．具体的な調停方式について同じ戦略をとることとし次節で計算式を与える．

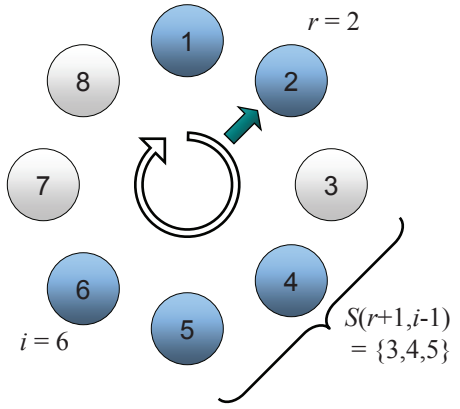

図 2 ラウンドロビン調停の状態の例．丸中の数字はプロセッサー番号を，色つきの丸はそのプロセッサーからメモリーアクセスが到着していることを表す．

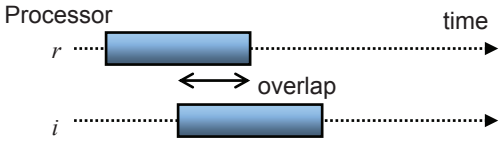

図 3 重なり (overlap) 関数の定義．

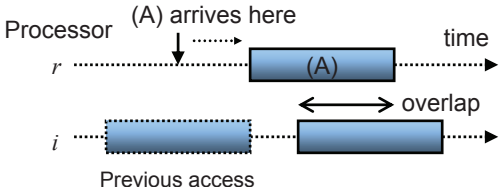

図 4 同一プロセッサーからの直前のアクセスによる重なり (overlap) の増加の例．

3.3 ラウンドロビン調停

ランドロビン調停の場合，プロセッサー i がいつ処理されるかはラウンドロビン調停器がアクセス時に選んでいたプロセッサー r がどれかに依存する．そこで，この選択を平均化して評価する．今，プロセッサーには整数で番号付けがされており，この整数の順に選択されるものとする (図 2)．

$$a_i \ (c_i = 1) \tag{11}$$

$$= \sum_{r}^{N} [U_r \beta_i + l(r+1, i-1)] P_{\mathrm{RR}}(r|i) \tag{12}$$

$$+ 1 \tag{13}$$

ここで

$$l(r+1,i-1)=\sum_{k}^{S(r+1,i-1)} U_k^* \tag{14}$$

はプロセッサー r とプロセッサー i の間にあるプロセッサーから来たアクセスの平均個数である．U^* が用いられているのは平均の評価に式 (10) の確率を用いたからである．そのほかの記号はそれぞれ，プロセッサー a と b の間のプロセッサーのインデックス集合

$$S(a,b)=\{e|\ e=o(a+j), \tag{15}$$
$$j=0,1,\cdots,j_{\max}, \tag{16}$$
$$o\ (a+j_{\max})=o(b)\}, \tag{17}$$

および，ラウンドロビンの番号付けに規格化する関数

$$o(a)=((a-1)\bmod N)+1 \tag{18}$$

である．

また

$$\beta_i=\begin{cases} 1/2, & (U_i<1/2) \\ (1+(1-x))/2, & (U_i\geq 1/2) \end{cases} \tag{19}$$

$$(x=\frac{1}{U_i}-1), \tag{20}$$

$$(x=\frac{1}{U_i}-1), \tag{21}$$

は，アクセス時に調停器に選択されて処理中だったプロセッサー r との処理時間の重なりの平均を表す関数で，通常は任意のタイミングのずれがありうるため $\beta=1/2$ だけ重なっている (図 3) が，$U_i\to 1$ では前のアクセスとの間隔 x がなくなるため重なりは $\beta\to 1$ となる (図 4)．

最後に

$$P_{\mathrm{RR}}\ (r|i) \tag{22}$$
$$=\begin{cases} (1-2U_i)P_{\mathrm{RR}}^{\mathrm{st}}(r) & \\ +2U_i\delta_{r,o(i+1)}, & (U_i\leq 1/2) \\ \delta_{r,o(i+1)}. & (U_i>1/2). \end{cases} \tag{23}$$

これはプロセッサー i がアクセスしたときにラウンドロビン調停がプロセッサー r を選択，処理中である確率であり，厳密には過去の調停の履歴によっているが，ここでも平均的に評価を行う．まずプロセッサー i からのアクセスが多い場合 ($U_i\to 1$)，アクセスの直前に同じプロセッサーから行われたアクセスを処理していたので，このアクセスの時にはその次の番号のプロセッサー $i+1$ が選ばれやすい．

$$P_{\mathrm{RR}}(r|i)=\delta_{r,o(i+1)} \tag{24}$$

一方 $U_i\to 0$ の場合は，直前にどのプロセッサーが選択されていたのかはまったく予測がつかないため，単純にアクセスの多さに比例して選択されているものと推定する．

$$P_{\mathrm{RR}}^{\mathrm{st}}(r)=U_r/C,\quad C=\sum_{r}^{N}U_r \tag{25}$$

これはラウンドロビン調停をマルコフ過程と考えたときの定常状態に相当する (付録 A.1)．式 (22) はこの両者の極限を $U_i=1/2$ で線形につないだものである．

単純な場合の結果を示す．すべてのプロセッサ $\forall j$ について同じ帯域利用率 $U_j=U$ の場合，$U\to 1$ の極限では，$U_j'=1/N$ となり，直観的な結果と一致する．

4. 実　　験

4.1 サイクル精度シミュレーションとの比較

今回，本手法の精度を確認するため，本手法による近似的な評価結果を，商用のサイクル精度シミュレーター (Cadence 社の Incisive シミュレーター) でのメモリーアクセスが競合した場合の帯域利用率と比較した．5 個のバスマスターとひとつの共有バス，およびバススレーブからなり，共有バスにおいて競合が発生する．(よってここではバスマスターをプロセッサー，共有バスを共有メモリーと読み替える)

調停方法はラウンドロビン，メモリーアクセスのバースト長は 8 サイクルであり，この間はバスを占有する．メモリーアクセスはすべて同期的である．すなわち 1 回のトランザクションが開始してから完了するまではバスマスターは次の処理を行わない．メモリーアクセスのパターンはランダムであり，一つのバスマスターはサイクルごとに乱数を引き与えられた一定の確率でメモリーアクセスを発生させる．

シミュレーションの全サイクル数は 10000 サイクルである．

すべてのプロセッサーが同じ帯域利用率 U でアクセスを行った場合の，競合後の帯域利用率 U' のグラフを図 5 に示す．$N=3,5$ の場合とも本手法による見積もりでよく表されている．特に $N=5$ の場合，合計の要求帯域の割合はバスの処理能力の 2 倍程度であり，比較的高い帯域利用率でも本手法とあっているといえる．一方，$N=3$ では $U=0.4$ 付近で，本手法の見積もりがやや低くなっており，競合の影響を過大評価しているところが見られる．

4.2 FPGA 評価ボードでのベンチマークプログラムとの比較

次により実際的なメモリーアクセスで検証するために，組込みシステム向けのベンチマークソフトウェアである MiBench[21] から，consumer というプ

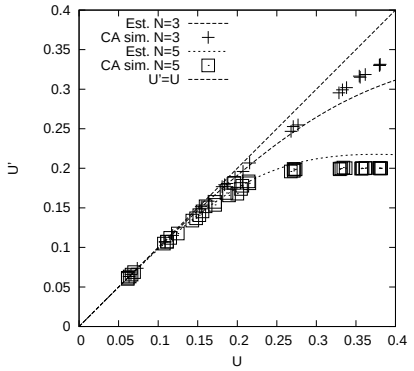

図 5 サイクル精度シミュレーション (CA sim.) と本手法の見積もり (Est.) での帯域利用率の比較

ログラムセットを用いて共有メモリーに負荷を与える．consumer セットは `jpeg`, `lame`, `mad`, `tiff2bw`, `tiff2rgba`, `tiffdither`, `tiffmedian`, `typeset`. のプログラムからなり，それぞれコンシューマー製品の特定の処理を模した負荷を与えることができる．

実行プラットフォームは Xilinx ML510 FPGA board であり，MPU(PowerPC 440)，Direct Memory Access (DMA) コントローラー，および共有メモリーからなる．これらの性能定数は表 1 のとおりである．MPU は L1 キャッシュを持ち，共有メモリーとの間でバースト転送を行う．L2 キャッシュメモリーは無効化されている．MPU ではベンチマークプログラムが実行される．DMA コントローラーでは一定の周期でバースト転送が行われ，その帯域利用率の要求量は $U_{\mathrm{DMA}} = 0.180, 0.359, 0.539$ と変化させた．

表 1 System constants

MPU burst length (32bits word)	8
MPU read throughput (M bursts/s)	19.68
MPU write throughput (M bursts/s)	13.33
DMA burst length (32bits word)	16
DMA read throughput (M bursts/s)	16.67
DMA write throughput (M bursts/s)	11.77

モデルに用いる性能パラメーター T および U は表 2 のとおりである．時間についてはベンチマークを呼び出しているプログラムに C 言語の `gettimeofday()` 関数を埋め込み，呼び出しオーバーヘッド分 $t = 0.161$(s) を引いたものを 10 回測定し，平均化している．メモリーアクセス回数 N は，FPGA 評価ボードの L2 キャッシュ部分に埋め込んだキャッシュエミュレーター中にある，L1 キャッシュミスカウンターの値をプログラム開始，終了時に読み取ることで計測した．なお，キャッシュヒットは共有メモリーアクセスの競合には寄与しないため，表の数字には含まれていない．

表 2 負荷パラメーター

pg	T (s)	$N_{\mathrm{R}}(\times 10^6)$	$N_{\mathrm{W}}(\times 10^6)$	U
jp	0.412	0.8226	0.3258	0.153
la	62.337	16.2544	5.0265	0.0181
ma	1.117	1.7621	0.9480	0.137
tb	1.921	5.5888	3.1883	0.260
tr	4.061	12.9010	8.5785	0.305
td	3.061	2.9707	2.1318	0.0967
tm	5.204	12.8343	6.5853	0.210
ty	4.169	12.0594	7.3973	0.267

測定結果を本手法による見積もりの値と比較したものを表 3 に示す．

本論文で説明してきた，ランダムで一様なアクセスで近似した手法 (表 3 sim.(R)) の場合，誤差は最大で 12.6%程度ある．しかもいずれも誤差はマイナスであり，本手法による見積もりが競合による影響を過小評価していることがわかる．

一方 5 節で後述するバースト的なアクセスを仮定した手法 (表 3 sim.(B)) と比較した場合，全体的に誤差は最大で 4.2%にとどまっており，より見積もりの精度が向上している．

表 3 ベンチマーク実験と本手法の見積もりとの間の実行時間の比較．$U_{\mathrm{DMA}} = 0.539$.

pg	T'/T exp.	T'/T sim.(R)	err (%)	T'/T sim.(B)	err (%)
jp	1.117	1.037	-7.2	1.087	-2.7
la	1.012	1.005	-0.7	1.010	-0.2
ma	1.076	1.033	-4.0	1.078	0.2
tb	1.189	1.064	-10.5	1.152	-3.1
tr	1.232	1.077	-12.6	1.181	-4.2
td	1.050	1.024	-2.5	1.054	-0.4
tm	1.168	1.050	-10.0	1.122	-4.0
ty	1.202	1.066	-11.3	1.157	-3.8

図 6 は `typeset` について，横軸に DMA の帯域利用率の要求量をとって本手法の見積もり値 (ランダム一様，およびバースト的) と比較したものである．いずれの場合も，どの領域でもバースト的アクセスを仮定した見積もりのほうが誤差が少ないことがわかる．

5. 議　　論

本手法では，一つのステップ内のメモリーアクセスはランダムかつ一様に起きているという近似のもとで計算を行っているが，先に述べたようにアプリケーションによってはバースト的なメモリーアクセスを行うことが知られている．

そこで今回の実験で MPU 側でバースト的アクセ

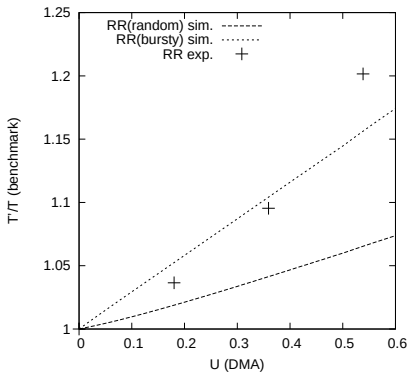

図 6 `typeset` ベンチマーク (RR exp.) の実行時間とシミュレーションによる見積もりの比較．RR sim(random) はランダムで一様な近似，RR sim(bursty) はバースト的アクセスの近似の場合．

スが起きていると仮定して見積もりの補正を試みる．そこで以下のような極端なケースを考える: バースト領域 ($U \sim 1$) と非バースト領域 ($U \sim 0$) が一つのステップ内に混在し，この領域の分かれ方はランダムである．この場合，衝突が起きる確率自体はこれまで用いてきた枠組みと変化しない．しかし，衝突が起きた場合，対象としているプロセッサー (i) からのアクセスはほとんどがバースト領域に所属するため，まるで $U_i = 1$ であるプロセッサーからのアクセスのように感じるであろう．そこで，式 (19) および (22) においてのみ，$U_i \to 1$ とし，その他の変数についてはすべてランダムかつ一様なアクセスのケースと同じままとする．

この結果を実験と比較したのが表 3 および図 6 に示されたバースト的アクセスを仮定した場合の見積もり値である．バースト的アクセスは平均的アクセスより競合がひどくなることは知られており，この見積もりでもそれが再現できていて，実験との一致も改善されている．今後は，使用したベンチマークプログラムを詳細に分析し，バースト性の有無を確認する必要がある．

6. ま と め

本研究ではシステムアーキテクチャー設計段階の粗い粒度の UML モデルに適した，共有メモリへのアクセス競合を考慮した性能評価のためのシミュレーション手法を提案した．

その中で，平均値分析 (MVA) と同様の伸び率 A の平均値での近似を利用して，計算量を削減したメモリーアクセス競合の計算方法を，ラウンドロビン調停など他の調停方法にも拡張可能な形で定式化した．また具体的にラウンドロビン調停において比較的高い帯域利用率でも適用できる式を具体的に示した．提案手法はサイクル精度シミュレーション，および FPGA 評価ボードと MiBench ベンチマークを用いた実験と比較した．誤差は最大 12.6%であったが，これはメモリーアクセスのバースト性を仮定した補正を行うことにより最大で 4.2%まで改善した．

今後の課題としては優先度つきの調停方法への拡張および，より現実的なシステムへの適用がある．

謝　　辞

実験および議論で多くのご協力をいただいた中村健太氏に感謝する．

参 考 文 献

1) Hennessy, J. L. and Patterson, D. A.: *Computer Architecture, Fourth Edition, A Quantitative Approach*, Elsevier, Morgan Kaufmann Publishers, chapter 4, pp. 196–284 (2007).
2) Fortier, P. J. and Michel, H. E.: *Computer Systems Performance Evaluation and Prediction*, Digital Press, chapter 7, pp. 201–249 (2003).
3) Open SystemC Initiative (OSCI): *SystemC specification* (2007).
4) Austin, T., Larson, E. and Ernst, D.: SimpleScalar: An Infrastructure for Computer System Modeling, *IEEE Computer*, Vol. 35, No. 2, pp. 59–67 (2002).
5) Object Management Group: *Systems Modeling Language (SysML) Version 1.0 Specification* (2007).
6) Object Management Group: *MARTE specification* (2008).
7) Balsamo, S., Marco, A. D., Inverardi, P. and Simeoni, M.: Model-based performance prediction in software development: a survey, *IEEE Transactions on Software Engineering*, Vol. 30, No. 5, pp. 295–310 (2004).
8) Cortellessa, V., Pierini, P. and Rossi, D.: Integrating Software Models and Platform Models for Performance Analysis, *IEEE Transactions on Software Engineering*, Vol. 33, No. 6, pp. 385–401 (2007).
9) Ono, K., Toyota, M., Kawahara, R., Sakamoto, Y., Nakada, T. and Fukuoka, N.: A Model-

based Method for Evaluating Embedded System Performance by Abstraction of Execution Traces, *Proceedings of 6th European Conference on Modelling Foundations and Applications (ECMFA 2010)* (Kühne, T., Selic, B., Gervais, M.-P. and Terrier, F.(eds.)), Lecture Notes in Computer Science, Vol. 6138, Paris, France, Springer, pp. 233–244 (2010).

10) 河原亮, 小野康一, 中田武男, 豊田学, 坂本佳史, 福岡直明: MARTE プロファイルを適用した UML モデルによる組込みシステムの性能評価シミュレーション, 第 9 回情報科学技術フォーラム (FIT 2010) 論文集 第 1 分冊, 電子情報通信学会, pp. 51–58 (2010).

11) Uhlig, R. A. and Mudge, T. N.: Trace-Driven Memory Simulation: A Survey, *ACM Computing Surveys*, Vol. 29, pp. 128–170 (1997).

12) Hennessy, J. L. and Patterson, D. A.: *Computer Architecture, Fourth Edition, A Quantitative Approach*, Elsevier, Morgan Kaufmann Publishers, chapter 1, pp. 1–62 (2007).

13) Hoogendoorn, C. H.: A General Model for Memory Interference in Multiprocessors, *IEEE Transactions on Computers*, Vol. C-26, No. 10, pp. 998–1005 (1977).

14) Mudge, T. N., Hayes, J. P., Buzzard, G. D. and Winsor, D. C.: Analysis of Multiple-Bus Interconnection Networks, *Journal of Parallel and Distributed Computing*, Vol. 3, pp. 328–343 (1986).

15) Smilauer, B.: General Model for Memory Interference in Multiprocessors and Mean Value Analysis, *IEEE Transactions on Computers*, Vol. C-34, pp. 744–751 (1985).

16) 河原亮, 中村健太, 小野康一, 中田武男, 坂本佳史: 共有メモリーシステムの性能評価のための粗粒度のシミュレーション手法, 組込みシステムシンポジウム 2010 (ESS2010) 講演集, pp. 23–32 (2010).

17) Kawahara, R., Nakamura, K., Ono, K., Nakada, T. and Sakamoto, Y.: Coarse-grained simulation method for performance evaluation a of shared memory system, *Proceedings of the 16th Asia and South Pacific Design Automation Conference (ASP-DAC 2011)*, pp. 413–418 (2011).

18) 柴田誠也, 石川拓也, 本田晋也, 冨山宏之, 高田広章: バス調停の遅延時間見積もりのための確率的数学モデル, 組込み技術とネットワークに関するワークショップ (ETNET2011) 論文集 (2011).

19) Bobrek, A., Paul, J. M. and Thomas, D. E.: Stochastic Contention Level Simulation for Single-Chip Heterogeneous Multiprocessors, *IEEE Transactions on Computers*, Vol. 59, pp. 1402–1418 (2010).

20) Darema-Rogers, F., Pfister, G. F. and So, K.: Memory access patterns of parallel scientific programs, *Proceedings of the 1987 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, SIGMETRICS '87, New York, NY, USA, ACM, pp. 46–58 (1987).

21) Guthaus, M. R., Ringenberg, J. S., Ernst, D., Austin, T. M., Mudge, T. and Brown, R. B.: MiBench: A free, commercially representative embedded benchmark suite, *Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop*, Washington, DC, USA, IEEE Computer Society, pp. 3–14 (2001).

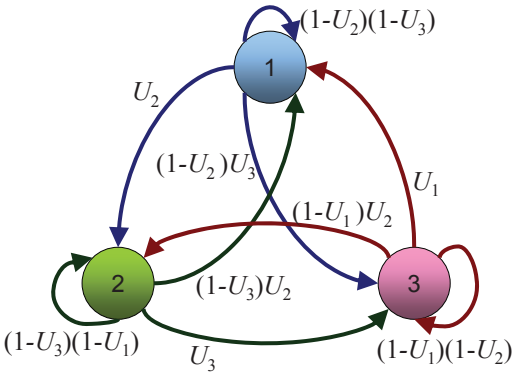

図 7 3 プロセッサーでのラウンドロビン調停の Markov モデルの状態遷移図の例．丸中の数字はプロセッサー番号を，状態遷移の矢印につけられたラベルは遷移確率をあらわす．

付　　録

A.1 ラウンドロビン調停の Markov モデル

ここでは 3.3 節で，帯域利用率が小さいとき用いられたラウンドロビン調停の Markov モデルとその定常確率分布 $P_{\mathrm{RR}}^{\mathrm{st}}$ を求める．調停器はプロセッサーの選択に対応した N の状態を持ち，$i \in \{0, 1, 2, \cdots, N-1\}$ と整数で順番付けされている．時間はサイクルによって離散化されるとし，1 アクセスあたり 1 サイクルかかるものとする．調停器は各サイクルでアクセスパターンにしたがって次の状態へと遷移する．ここで，各プロセッサーからのアクセスは前のステップや状態とは独立であるという仮定をおく．このプロセスは Markov モデルで記述することができる (実際には調停待ちプロセッサーによるアクセスの再送があるので独立ではない．これは帯域利用率が高いときに顕著である．)．

この条件の下で，$p^{(n)}(i)$ をあるサイクル n において調停器が状態 i にいる確率，$R_{i,j}$ を調停器が状態 i から状態 j に遷移する確率とする．Markov モデルの時間発展は以下のようになる．

$$p^{(n+1)}(j) = \sum_i R_{i,j} p^{(n)}(i). \tag{26}$$

したがって，定常状態においてはサイクル n と $n+1$ において p が等しいことから以下の固有方程式を解くことによって定常確率分布 $p(j) = P^{\mathrm{st}}_{\mathrm{RR}}(j)$ が得られる．

$$\sum_i R_{i,j} p(i) = p(j) \tag{27}$$

具体的に遷移確率 R を求める．例を図 7 に示す．非対角項 (ほかのプロセッサー選択状態への遷移の場合) は，現在選択の状態から順序的に最も近く，かつアクセスがあったプロセッサーが選択されるから，

$$R_{i,j} = \prod_{k=1}^{a(i,j)-1} n(r(i+k))b(j), \quad (j \neq i), \tag{28}$$

対角項 (現在のプロセッサー選択状態から移動しない場合) は，ほかのプロセッサーからのアクセスがない場合にのみ発生するので

$$R_{j,j} = \prod_{k=1}^{N-1} n(r(j+k)), \tag{29}$$

となる．ここで各記号は以下のとおりである．

$$r(k) = k \bmod N \tag{30}$$

$$a(i,j) = a, \text{ such that } r(i+a) = j, \tag{31}$$

$$n(i) = (1 - U_i), \quad b(i) = U_i \tag{32}$$

$$(n(i) + b(i) = 1) \tag{33}$$

この時，次の形の式を式 (27) に代入すると満たすことから，これが定常確率分布であることがわかる．

$$p(i) = \frac{U_i}{C}. \tag{34}$$

ここで，C は規格化定数である．

$$C = \sum_k U_k. \tag{35}$$