日本オペレーションズ・リサーチ学会
2021年 秋季研究発表会

バッファを考慮した並列機械モデル上での衝突確率の算法

05000853　筑波学院大学　*大塚帯紀　OTSUKA Taiki
01309090　法政大学　千葉英史　CHIBA Eishi

1. はじめに

本研究では，製造システムへの応用を念頭において，特に次の 3 点に注目した製造モデルを扱う: (1) 同一製品を大量生産すること，(2) タクトタイム方式から単位時間当たりの生産数を見積もること，(3) できるだけ原材料間の衝突を避けること．文献 [1] で，このような製造モデルとして**並列機械モデル**が提案され，このモデル上での衝突確率の効率的な算法も与えられた．

本研究では，バッファを考慮した並列機械モデルに注目する．ここで，バッファは衝突を回避するための原材料の待機場所のことであり，製造システムにおいてしばしば重要な役割を担う．バッファを考慮した並列機械モデル上での衝突確率の効率的な算法を提案する．また，提案した算法が実際に高速に動作することを示すために，PC 上に実装して，計算機実験を通してその性能を示す．

2. バッファを考慮した並列機械モデル

以下の表記を用いる．

- $M_1, M_2, \ldots, M_m$: m 台の並列機械.
- $J_1, J_2, \ldots, J_n$: n 個のジョブ.
- T_i (> 0): ジョブ J_i の処理時間.
- t_{tact} (> 0): タクトタイム．すなわち，ジョブの機械への投入間隔.
- b $(\in \mathbb{Z}_+)$: 各機械のバッファ数.

本研究における並列機械モデルの例を図 1 に示す．ジョブについて，$J_1, J_2, \ldots, J_n$ の順番に，一定の t_{tact} 間隔で入口から投入される．処理時間 T_i は確率変数であり，互いに独立である．各ジョブの投入時，アイドル機械があれば，その機械で処理されて，出口へと運ばれる．各ジョブの投入時，アイドル機械がなければ，最も空いている機械のバッファで一時的に待ち状態になる．ここでの待ちのルールは FIFO とする．各ジョブの投入時，各機械のバッファがすべて使用されているならば，**衝突**が生じる．処理するジョブが与えられるとき，衝突が生じる確率を**衝突確率**と呼ぶ．

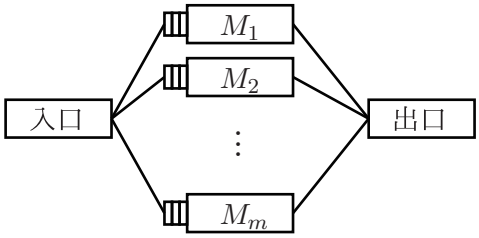

図 1: $b = 3$ のときの並列機械モデル.

3. 衝突確率の算法

シミュレーションから衝突確率を求めるために，衝突判定法を提案する．機械が処理しているジョブの個数とバッファに格納しているジョブの個数の和を，**部分ジョブ和**と呼ぶ．

衝突判定法は，以下の操作 1，2 を各ジョブ J_i に対して繰り返す．

操作 1 ジョブ J_i を投入する時，ジョブ J_{i-1} の投入時以降に完了するジョブに対して完了処理を行う．

操作 2 ジョブ J_i を投入する時，最小の部分ジョブ和を持つ機械を探索する．その最小の部分ジョブ和の値によって，探索した機械が，アイドル機械であるか，バッファに余裕があるか，全てのバッファを使用しているかを調べて，それぞれの状態に応じてジョブ J_i の操作を行う．

操作 1，2 は，サイズ m の Heap 構造を利用している．操作 1 の Heap 構造は，ジョブ番号を管理し，順序関係はジョブの処理完了時刻で定め，根に最小の処理完了時刻を持つジョブ番号が対応されるようにする．操作 2 の Heap 構造では，機械番号を管理し，順序関係は部分ジョブ和で定め，根に最小の部分ジョブ和をもつ機械番号が対応するようにする．

操作 1 は以下のように書ける．

Step 1 ジョブを処理している機械の中で，最初に処理を完了する機械を探す.

Step 2 Step 1 で探索した機械の処理完了時刻とジョブ J_i の投入時刻を比較．投入時刻の方が遅ければ，Step 3 へ，それ以外なら操作 1 は終了．

Step 3 Step 1 で探索した機械のバッファに待ちジョブが存在すれば，そのジョブの中で，先頭にあるジョブの処理を開始させ，Step4 へ．

Step 4 Step 1 で探索した機械の部分ジョブ和を 1 つ減らし，Step 1 に戻る．

操作 2 は以下のように書ける．

Step 1 並列の機械の中で，最も小さい部分ジョブ和をもつ機械を探す．

Step 2 Step 1 で探索した機械の部分ジョブ和が 0 ならジョブの処理を開始し，部分ジョブ和が b 以下ならジョブを待ち状態にする．部分ジョブ和が $b+1$ なら，衝突と判定する．

操作 1 は，機械の探索で Heap 構造を利用し，各ジョブに対し 1 回の完了処理を実行することから，全体の計算量は $O(n \log m)$ である．操作 2 についても同様に考えて，$O(n \log m)$ である．よって，衝突判定法の計算量は $O(n \log m)$ となる．

衝突確率は，衝突判定法を繰り返し実行することで得られる．繰り返し回数を c とすれば，衝突確率を算出するための計算量は $O(cn \log m)$ となる．

4. 計算機実験

提案手法を PC（CPU: Intel Core i5 2.30 GHz, RAM: 32 GB, OS: Windows 10 Professional）上で実装した．処理時間は期待値 100，分散 $10,000$ の指数分布に従うとして，$m = 10$，$n = 10,000$，$c = 50,000$ に設定した．

一般に，衝突確率は t_{tact} の増加とともに減少する特徴がある．そして，タクトタイムがある値を超えると，その後のタクトタイムに対する衝突確率は，ほぼゼロになる．そのようなタクトタイムの閾値（i.e. **最小タクトタイム**）は，衝突確率がほぼゼロとなるタクトタイムの最小値のことであり，t^*_{tact} と表記する．t^*_{tact} は，2 分探索法を使って効率的に求めることができる．様々なバッファ数 b に対する t^*_{tact} の計算結果を図 2 に示す．図 2 から，t^*_{tact} は b の増加とともに減少する; b が小さいとき（大体 10 以下），その減少率は大きく; b が大体 10 を超えると，ゆるやかになるのが分かる．

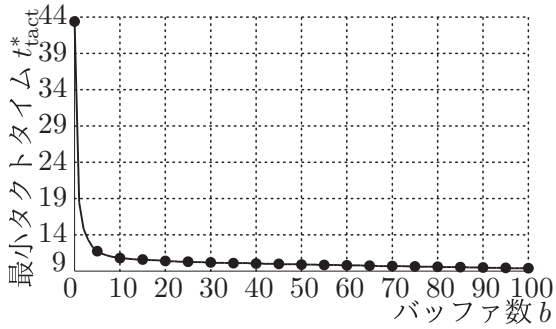

図 2: 最小タクトタイム t^*_{tact} の計算結果．

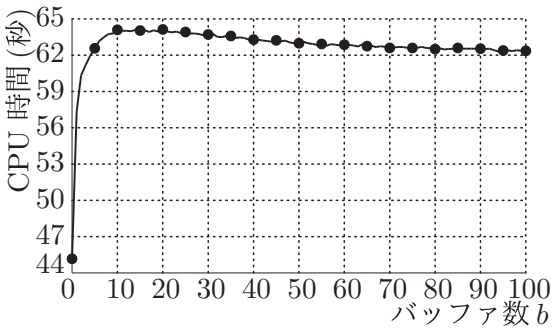

図 3: 衝突確率を求めるのに要した CPU 時間（秒）．

次に，様々なバッファ数 b に対して，t_{tact} を図 2 での t^*_{tact} に設定して，衝突確率を求めるのに要する CPU 時間を調べた．その結果を図 3 に示す．図 3 から，実用的な時間で衝突確率を計算することができ，$b = 0$ のときを除いて，CPU 時間がバッファ数にほぼ依存しないことも確認できる．

5. おわりに

本研究では，バッファを考慮した並列機械モデル上での衝突確率の効率的な算法を提案した．また，提案手法を実装して，最小タクトタイムと計算時間の観点からそれぞれ考察を行った．今後の課題として，直並列型モデルへの拡張，搬送時間の考慮など，より一般的なモデル上での衝突確率の算法に関する研究が挙げられる．

参考文献

[1] T. Otsuka and E. Chiba, “A framework for computing collision probability in a parallel machines model,” International Journal of Japan Association for Management Systems, vol.10, no.1, pp.117–124, 2018.