法政大学学術機関リポジトリ
HOSEI UNIVERSITY REPOSITORY

PDF issue: 2025-07-06

バッファを考慮した並列機械モデルにおける衝突確率の算法に関する研究

Guo, Longjiang / 郭, 龍江

(出版者 / Publisher)
法政大学大学院理工学研究科
(雑誌名 / Journal or Publication Title)
法政大学大学院紀要. 理工学研究科編
(巻 / Volume)
64
(開始ページ / Start Page)
1
(終了ページ / End Page)
4
(発行年 / Year)
2023-03-24
(URL)
https://doi.org/10.15002/00026440

バッファを考慮した並列機械モデルにおける衝突確率の算法に関する研究

A Study on Computation of Collision Probability in a Parallel Machines Model with Buffers

郭　龍江
Guo Longjiang

指導教員 千葉英史

法政大学大学院理工学研究科システム理工学専攻修士課程

Line production system is used for continuous production in manufacturing industry in order to improve production efficiency. The probability of occurring a collision is one of the most important factors in relation to the productivity of the line production system. In this paper, we present a simulation algorithm to compute collision probability in a parallel machines model with buffers. This simulation algorithm iteratively uses a method of deciding whether a collision occurs or not. This method makes use of two binary heaps to efficiently decide whether a collision occurs. Throughout simulations, we investigate the relationships between the number of machines, the number of buffers, tact time, and collision probability. Collision probability decreases as tact time increases, and there is a minimum tact time at which collision probability is about zero. The minimum tact time dramatically decreases as the number of machines increases. The minimum tact time decreases as the number of buffers increases. We confirm that the CPU time is independent of the number of buffers.

Key Words *: collision probability, computation, parallel machines model, buffer*

1. はじめに

製造業では，連続生産を行う場合，効率的な生産のためにライン生産方式を採用する．材料の投入間隔が決まっているライン生産方式では，一定時間内により多くの製品を製造できるように投入間隔を短くすることが求められる．しかし，投入間隔を短くすると，以前に投入されたジョブの処理が完了する前に，新たなジョブが機械に入ることがあり，材料間の衝突が起こる可能性が高くなる．衝突の発生は，生産ロスをもたらす．投入間隔が等しい場合，各機械の前にバッファを設置して材料を一時的に保管することで衝突の確率を下げることができるが，バッファを多く設置するとコストアップになるだけでなく，生産性も低下する．従って，投入間隔，バッファ数，衝突確率の関係を究明することは，非常に重大な意義がある．文献 [2] では，並列機械モデル上で衝突確率を計算する高速なアルゴリズムが提案された．文献 [3] では，直並列機械モデル上でも衝突確率を計算する高速なアルゴリズムが提案された．文献 [1, 2, 3] では，機械がジョブを加工処理するのにかかる時間を Erlang 分布で表している．

本研究では，バッファを考慮した並列機械モデルにおけるジョブの衝突判定アルゴリズムを提案する．提案アルゴリズムを PC に実装し，投入間隔，バッファ数，衝突確率の関係をシミュレーションにより検討する．本研究によれば，衝突確率を低く抑えつつ，できるだけ短い投入間隔を設定することができる．対象とする並列モデルは以下の３点を特徴とするモデルである: (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．

2. バファ付き並列機械モデル

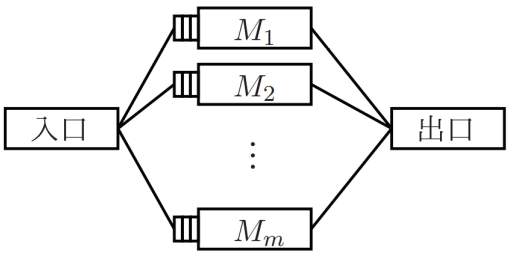

図 1 $b = 3$ のときの並列機械モデルの例

以下の表記を用いる．

- M_1，M_2，...，M_m: m 台の並列機械.
- J_1，J_2，...，J_n: n 個のジョブ.
- T_i: 各ジョブ J_i の処理時間（確率変数）.
- t_{tact}: タクトタイム．すなわち，ジョブの投入間隔.
- b: 各機械のバッファ数.

バッファ付きの並列機械モデルの例を図 1 に示す．このモデルの動作は，ジョブが $J_1, J_2, \ldots, J_n$ の順番で一定の時間

間隔 t_{tact} で空いている機械に投入され，その機械で処理された後，出口に運ばれるというものである．空いている機械がなければ，最も待機ジョブ数が少ない機械にあるバッファに入って待ち状態になる．ここでの待ちのルールは「先入れ先出し」という FIFO とする．ジョブに関する搬送時間はゼロとする．機械での処理内容は同じであるが，処理時間について多少の偏りが生じるため，処理時間は確率分布変数として，さらに互いに独立であると仮定する．各ジョブを投入するとき，すべての機械のバッファに空きがなければ，衝突が起こる．衝突確率は少なくとも 1 回の衝突が起こる確率とする．

3. 衝突確率の算法

衝突確率を計算するために，衝突の有無を判定する必要がある．そこで，衝突の判定と衝突確率の計算の 2 つの部分に分けて説明する．

(1) 衝突の判定

衝突判定の方法は，以下の操作 1，2 を各ジョブ J_i の投入時に対して繰り返す．その中，**部分ジョブ和**とは，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である．

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 M とする．その最小の部分ジョブ和の値によって，M 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 2 分 Heap データ構造を 2 つ利用する．1 つ目の Heap 構造は，処理中の機械を管理し，キーは完了時刻で定め，根に最小の完了時刻を持つ機械番号が対応する．2 つ目の Heap 構造は，機械番号を管理し，キー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．FIFO のルールでバッファにある待ち状態のジョブを管理するため，データ構造キューを利用しする．

具体的には，操作 1 は以下のようになる．

Step 1 ジョブを処理している機械の中で，最小の完了時刻を持つ機械を探す．

Step 2 Step 1 で探索した機械の処理完了時刻とジョブ J_i の投入時刻を比較する．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 で探索した機械の部分ジョブ和を 1 つ減らす．この機械のバッファに待ちジョブが存在すれば，Step 4 へ．存在しなければ，Step 1 に戻る．

Step 4 Step 1 で探索した機械のバッファにあるジョブの中で，先頭にあるジョブの処理を開始させ，Step 1 に戻る．

操作 2 は以下のようになる．

Step 1 すべての並列機械の中で，最小の部分ジョブ和を持つ機械を探す．

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 以下ならジョブをバッファにいれて待ち状態にする．部分ジョブ和が $b+1$ なら，衝突と判定する．

以上から，衝突判定のアルゴリズムは Algorithm 1 のようになる．

Algorithm 1 COLLISION-DECISION

入力: ジョブ数 n，機械数 m，バッファ数 b，タクトタイム t_{tact}，処理時間 $t_1, t_2, \ldots, t_n$.

出力: 1(衝突が生じるとき)，0(衝突が起こらないとき).

```
1: S_1 ← ∅
2: for j ← 1 to m do
3:    M_j.waitingjobs ← 0
4:    S_2[j] ← j
5:    InitQueue(Q_j)
6: end for
7: for i ← 1 to n do
8:    J_i.arrivaltime ← (i − 1) · t_tact
9:    while S_1 ≠ ∅ and S_1[1].completiontime ≤
      J_i.arrivaltime do
10:      M_j ← EXTRACTMIN(S_1)
11:      M_j.waitingjobs ← M_j.waitingjobs − 1
12:      HEAPIFYUP(M_j)
13:      if Q_j ≠ ∅ then
14:         J_k ← DEQUEUE(Q_j)
15:         J_k.processingtime ← t_k
16:         M_j.completiontime ← M_j.completiontime +
            J_k.processingtime
17:         INSERT(S_1, M_j)
18:      end if
19:   end while
20:   M_j ← S_2[1]
21:   if M_j.waitingjobs = 0 then
22:      J_i.processingtime ← t_i
23:      M_j.completiontime ← J_i.arrivaltime +
         J_i.processingtime
24:      INSERT(S_1, M_j)
25:      M_j.waitingjobs ← M_j.waitingjobs + 1
26:      HEAPIFYDOWN(M_j)
27:   else if M_j.waitingjobs ≤ b then
28:      ENQUEUE(Q_j, J_i)
29:      M_j.waitingjobs ← M_j.waitingjobs + 1
30:      HEAPIFYDOWN(M_j)
31:   else
32:      return 1   { 衝突が生じる }
33:   end if
34: end for
35: return 0   { 衝突が生じない }
```

Algorithm 1 では，ヒープに関する以下の基本的な関数を使う．

- InitQueue(Q): キューの Q を初期化して空にする．

- ENQUEUE(Q, J_i): ジョブ J_i をキュー Q に追加する．時間計算量は $O(1)$.
- DEQUEUE(Q): キュー Q の先頭にあるジョブを取り出す．時間計算量は $O(1)$.
- INSERT(S_1, M_j): 処理中の機械の集合 S_1 に機械 M_j を追加し，処理完了時間の大小関係によって最小ヒープを構成する．時間計算量は $O(\log m)$.
- HEAPIFYUP(M_j): 機械 M_j に対応するノードから上に向かってヒープ化する．時間計算量は $O(\log m)$.
- HEAPIFYDOWN(M_j): 機械 j に対応するノードから下に向かってヒープ化する．時間計算量は $O(\log m)$.
- EXTRACTMIN(S_1): 処理中の機械の集合 S_1 の根にある機械を取り出した後，処理完了時間の大小関係によって最小ヒープを構成する．時間計算量は $O(\log m)$.

Algorithm 1 の時間計算量は以下のように見積もることができる．1 つ目の for ループ (2 行目から 6 行目) では，m 台の機械が持っている部分ジョブ和をそれぞれ 0 にし，サイズ m の Heap 構造 S_2 に 1 から m までの機械番号を追加し，m 個のキュー Q をそれぞれ空にして初期化する．その計算量は $O(m)$ である．while ループ (9 行目から 19 行目) において，ジョブの完了処理の総回数が高々n 回あることと，ヒープに関する手続きを考慮すると，計算量が $O(n \log m)$ となる．よって，Algorithm 1 の時間計算量は $O(m + n \log m)$ となる．

(2) 衝突確率の計算

一定の時間間隔 t_{tact} で n 個のジョブを m 台のバッファ付きの並列機械に投入し，衝突の有無を判定する．この操作を N 回繰り返し，衝突ありの回数 n_c を数える．この場合，衝突確率は n_c/N として求めることにする．反復回数 N が多いほど，精度の高い確率になると考えられる．衝突確率の算法を Algorithm 2 に示す．

Algorithm 2 COLLISION-PROBABILITY

入力: ジョブ数 n，機械数 m，バッファ数 b，タクトタイム t_{tact}，正整数 N，確率分布のパラメータ.

出力: 衝突確率.

```
1: n_c ← 0
2: for loop ← 1 to N do
3:     処理時間 t_1, t_2, ..., t_n を生成する.
4:     if COLLISION-DECISION = 1 then
5:         n_c ← n_c + 1   { 衝突が生じる }
6:     end if
7: end for
8: return n_c/N
```

Algorithm 2 の時間計算量は以下のように見積もることができる．for ループにおいて N 回反復し，毎回衝突判定する．よって，Algorithm 2 の時間計算量は $O(mN + nN\log m)$ となる．

4. 数値実験

第 3 章で説明した手法を PC（CPU: 11th Gen Intel Core i3-1115G4 3.00 GHz，RAM: 8 GB，OS: Windows 10 Education）上で実装した．Erlang 分布が実際の処理時間を表現するのに十分な柔軟性を持っていることはよく知られている．Erlang 分布の確率密度関数は次のように定義される．

$$f(x; k, \lambda) = \frac{\lambda^k x^{k-1} e^{-\lambda x}}{(k-1)!},\ x \geq 0.$$

ここで、2 つのパラメータ λ と k は，それぞれ正の実数と正の整数である．その期待値と分散はそれぞれ k/λ と k/λ^2 である．このように，Erlang 分布のもとでは，パラメータ λ と k を適切に設定することで，期待値と分散を互いに独立にすることができる．Erlang 分布の確率密度関数はパラメータの設定次第でベル型になり，x が 0 より小さいなら関数値は 0 となる．したがって，Erlang 分布は実際の処理時間を表現するのに十分な柔軟性を持っている．

計算実験 1 では，各ジョブの処理時間は期待値が 1，分散がそれぞれ 1，0.1，0.01，0.001，0.0001，0.00001，0.000001 の Erlang 分布に従うとして，機械数 $m = 50$，バッファ数 b=0，ジョブ数 $n = 10000$，反復回数 $N = 10000$ に設定した．実験 1 の結果を図 2 に示す．タクトタイムの増加とともに衝突確率の変化は 3 段階の傾向からなると見てとれる．第 1 段階では，タクトタイムの増加に伴って，確率が 1 のままで変化することはない．第 2 段階では，小さな区間で，衝突確率が 1 から 0 になるまで急激に減少する．処理時間が従う分布の分散が大きいほど，この段階の曲線は緩やかになる．第 3 段階では，0 のままである．ほぼゼロになるタクトタイムの最小値を**最小タクトタイム**と呼ぶ．この値は，二分探索法で求めることができる．

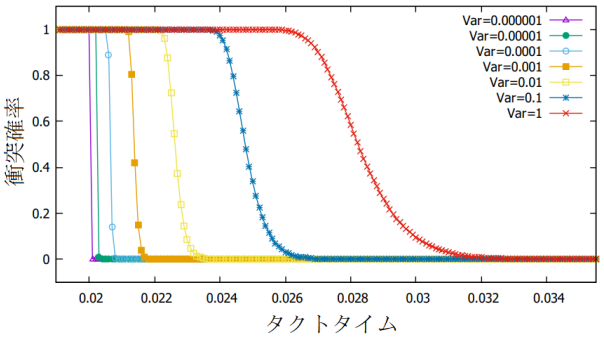

図 2 タクトタイムと衝突確率の関係

計算実験 2 では，各ジョブの処理時間は期待値が 1，分散がそれぞれ 0.1，0.01，0.001 の Erlang 分布に従うとして，バッファ数 $b = 0$，ジョブ数 $n = 10000$，反復回数 $N = 10000$ に設定した．実験 2 の結果を図 3 に示す．図 3 から，最小タクトタイムは機械数 m の増加とともに減少する．m が小さいとき（大体 10 以下），その減少率は大きく，m が大体 10 を超えると，ゆるやかになるのが分かる．

計算実験 3 では，各ジョブの処理時間は期待値が 1，分散がそれぞれ 0.1，0.01，0.001 の Erlang 分布に従うとして，機械数 $m = 10$，ジョブ数 $n = 10000$，反復回数 $N = 10000$ に設定した．実験 3 の結果を図 4 に示す．図 4 から，バッファ数 b が 3 以下のとき，最小タクトタイムは b の増加とともに減少する．バッファ数 b が 3 以上になると，最小タクトタイムは b の増加とともに不変となるのが分かる．

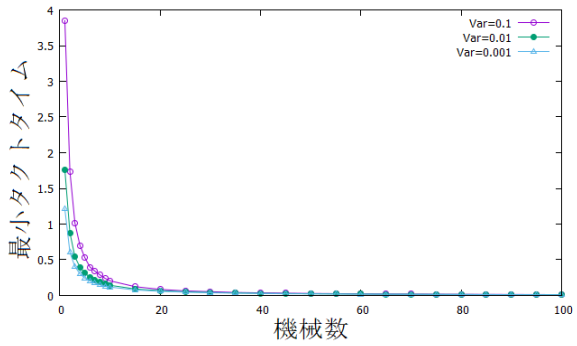

図 3 機械数と最小タクトタイムの関係

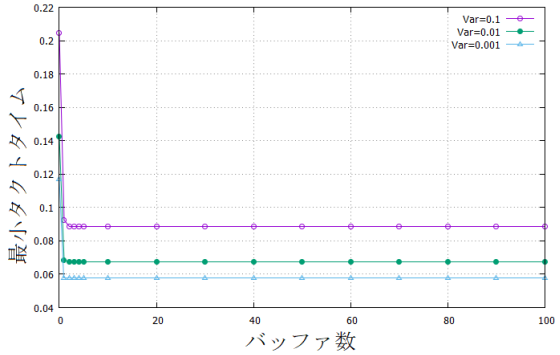

図 4 バッファ数と最小タクトタイムの関係

タクトタイムを最小タクトタイムとするとき，衝突確率を求めるのにかかる CPU 時間を調べた．その結果を図 5 に示す．図 5 から，実装したアルゴリズムの CPU 時間が分散が違っても変わらないこと，$b = 0$ のときを除いて CPU 時間がバッファ数に依存しないことを確認できる．

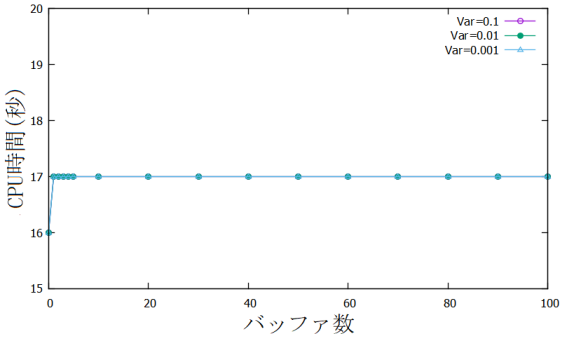

図 5 衝突確率を求めるのに要した CPU 時間（秒）

5. 考察

最小タクトタイムが分かれば，衝突確率がほぼゼロであることが保証されるまま最小の投入間隔を実現し，生産ラインの効率向上に貢献することができると考えられる．最小タクトタイムを短縮できれば，同じ時間でより多くの製品を生産できるようになる．高品質な機械を用いて生産すると，処理時間が従う Erlang 分布の期待値が同じでも，分散が小さいなら，最小タクトタイムを短縮できる．機械とバッファをより多く増加して最小タクトタイムを短縮することができるが，コストの面でも考えると，機械とバッファは多ければ多いほどよいというものではない．本研究により，最も適切な機械数とバッファ数を選ぶことができると考えられる．また，Erlang 分布の期待値と分散はパラメータで独立に設定できるが，k が正整数のため，期待値が 1 のとき，分散の最大値は 1 である．

6. 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，最小タクトタイムや CPU 時間の観点から考察を行った．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] E. Chiba, H. Fujiwara, Y. Sekiguchi, and T. Ibaraki, “Collision Probability in an In-Line Equipment Model under Erlang Distribution”, IEICE Transactions on Information and Systems, vol.E96-D, no.3, pp.400–407, 2013.

[2] 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.

[3] T. Otsuka and E. Chiba, “An efficient method to compute collision probability in a series-parallel machines model”, International Journal of Japan Association for Management Systems, vol.12, no.1, pp.51–58, 2020.