B-011

スレッドを隠蔽する並列プログラミングモデルの検討・実装
General Model for Parallel Programming without Handling Threads

佐々木 徹* 金丸 将平* ジョ ヨンジュン†
Tohru Sasaki Shohei Kanamaru YongJoon Joe

1 はじめに

現在，並列プログラミングには複数の手法 [1][2][3] が存在する．その手法の一つとしてスレッドを用いた並列化手法がある．本論文ではスレッドによる並列化を抽象化する並列モデルを提案する．また，それらのフレームワークとしての実装及びその効果について考察する．

2 並列プログラミングのモデル化

2.1 決定性有限オートマトンの拡張

本論文では並列プログラミングの抽象モデル (以下，並列プログラミングモデルと呼ぶ) を提案する．並列プログラミングモデルでは，決定性有限オートマトン (Deterministic Finite Automaton) の状態遷移図を拡張した無閉路有向グラフを使用する．この無閉路有効グラフを並列プログラミングモデルでは実行関連グラフと呼ぶ．この実行関連グラフの例を図 1 に示す．

並列プログラミングモデルはこの実行関連グラフを基に構成される．実行関連グラフはプログラム全体の実行フローを表現している．グラフの各ノードにはそれぞれ小規模なプログラム (以下，並列ユニットと呼ぶ) が対応付けされる．この各プログラムは基本的にそれぞれ独立に作成・実行され，お互いに影響することはない．図 1 では 6 個の小規模なプログラムが存在することになる．

2.2 実行関連グラフにおけるプログラムの動作順序

実行関連グラフの有向辺は実行順序を表している．辺の始点になっているプログラムが終了した際に終点になっているプログラムが実行されることを意味している．図 1 では，0 番のノードに対応付けられたプログラムの動作が終了すると，ノード 1, 2, 3 のプログラムが実行される．

実行関連グラフの有向辺には “ AND 辺 ” と “ OR 辺 ” の 2 種類が存在する．これらはそれぞれ終点側のノードに対応付けられたプログラム (以下，単にノードといった場合そのノードに対応付けられているプログラムを指す) の実行のタイミングが異なる．AND 辺はあるノードに向かっている有向辺の始点側のプログラムの全てが実行終了した際に実行が開始される．一方，OR 辺はあるノードに向かっている有向辺の始点側のプログラムのうちいずれか一つが実行終了した際に実行が開始される．図 1 では，AND の有向辺を点線の矢印で，OR の有向辺を実線の矢印で表現している．このとき，AND 辺の終点であるノード 6 のプログラムはノード 4, 5 の両方が終了した際に実行が開始される．一方，OR 辺の終点であるノード 5 のプログラムは，ノード 1, 2 のうちどちらか一方が終了した時点で実行を開始することになる．

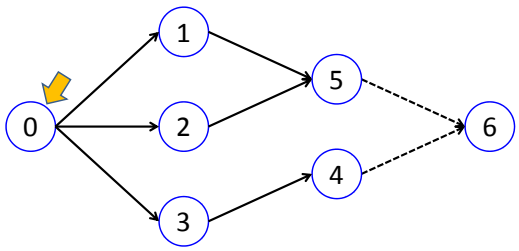

図 1 並列プログラミングモデルの実行関連グラフ

2.3 並列プログラミングモデルの持つ並列性と同期性

2.2 で示した実行関連グラフでは，並列化における様々な状況を表現できる．図 1 のノード 0 からノード 1, 2, 3 への有向辺はそこから処理が並列で行われることを意味する．ノード 1, 2, 3 はそれぞれノード 0 の実行が終了した際に実行される．したがって，ノード 1, 2, 3 は同時に処理されることとなり並列処理を実行関連グラフにて表現していることに他ならない．

また，AND 辺は処理の同期機構となっている．図 1 では，ノード 6 がノード 4, 5 の両方の終了時に実行される．これは，ノード 4, 5 の終了においてバリア同期を設け，ノード 6 の実行を待機したことを意味する．

3 フレームワークの実装

並列プログラミングモデルに基づくプログラミングフレームワークを実装し，その効果について検証する．フ

* 九州大学大学院システム情報科学府情報知能工学専攻
† 九州大学大学院システム情報科学府情報学専攻

表 1　実験を行ったコンピュータの諸元表

OS	Windows 7 Home Premium 64bit
CPU	Intel(R) Core i7 M640@2.80GHz
コア数	4
スレッド数	4
メモリ	8.0GByte

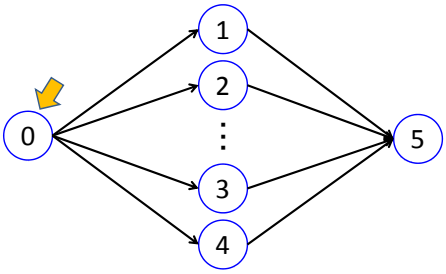

図 2　検証のモンテカルロ法における実行関連グラフ

レームワークは実行関連グラフの作成と各ノードに対応づけるプログラムの作成・指定のみで使用できる．並列プログラミングモデルに従い，スレッド等の並列化機構は隠蔽した．

3.1　実装手法

本論文で実装したフレームワーク (以下，並列フレームワークと呼ぶ) は C++ で実装し，並列ライブラリとして Boost[*1]の Thread を利用した．また，並列ユニット間でのデータ交換を目的とし，全ての並列ユニットからアクセスできる共有メモリを用意した．さらに，Semaphore，Mutex などの排他機構に関してもフレームワークに用意し，並列ユニット内で使用可能とした．

3.2　検証

並列フレームワークの検証実験として，モンテカルロ法による円周率の導出を各手法で行い，その実行時間を比較した．実験は，一般的なシングルスレッドプログラム，Boost の Thread を用いた並列プログラム (以下，マルチスレッドプログラムと記載)，並列フレームワークを用いた並列プログラム (以下，フレームワークプログラムと記載) の 3 つで行った．実験を行ったコンピュータの諸元表は表 1 に示す．

実験の結果，マルチスレッドプログラム，及びフレームワークプログラムのいずれもシングルスレッドプログラムと比べより短時間で実行された．実験結果を図 3 に示す．図 3 より，すべての場合においてマルチスレッドプログラムの方がフレームワークプログラムの実行時間を上回っている．特に，スレッド数が 100 以上ではその傾向は顕著である．これは直接スレッドを生成するマルチスレッドに比べ，フレームワークプログラムの実行関連グラフに基づく実行プログラムの決定によるオーバーヘッドがあるためと考えられる．しかし，スレッド数 10 前後では差は僅かであり，利便性を考慮した場合，問題のない程度であると考えられる．

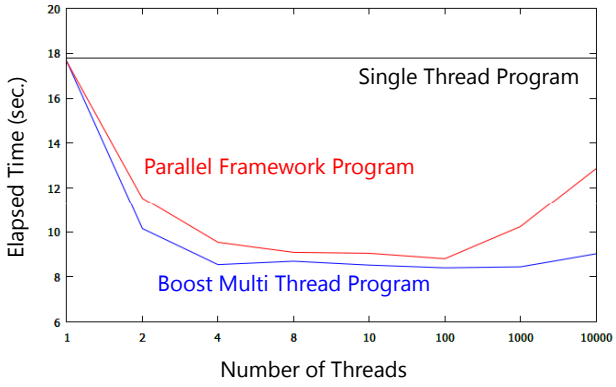

図 3　各プログラムでのプログラム実行時間

4　まとめと今後の展望

決定性有限オートマトンの状態遷移図を拡張した並列プログラミングモデルを用いた，スレッド機構を隠蔽した並列化を提案した．この手法を実装したフレームワークでは，スレッドの隠蔽により並列記述の簡素化を実現した．今後の展望としては並列モデルに分岐条件を盛り込み，さらに自由度の高い並列化の実現が挙げられる．

謝辞

論文作成にあたり，多大なるご支援を頂いた九州大学大学院システム情報科学研究院 内田誠一教授にお礼申し上げる．

参考文献

[1] 石川 裕，小中 裕喜，前田 宗則，友清 孝志，堀 敦史, “超並列プログラミング言語 MPC++ の概要, ” 情報処理学会研究報告. [プログラミング-言語基礎実践-] 93(73), 81-88, 1993.

[2] 和泉 秀幸, 中島 毅, “実行モデルに基づく並列プログラミング支援環境の構築, ” 情報処理学会研究報告. ソフトウェア工学研究会報告 98(64), 85-92, 1998.

[3] 若谷　彰良, “マルチコア時代の並列プログラミング,” 甲南大学紀要. 知能情報学編 1(2), 223-247, 2008.

[*1] http://www.boost.org/