2003年日本オペレーションズ・リサーチ学会
秋季研究発表会

CPLEX MIP Optimizer のPUBB2フレームワークによる並列化について

01507034 神戸商科大学 *藤江 哲也 FUJIE Tetsuya
01207140 東京農工大学 品野 勇治 SHINANO Yuji
東京農工大学 鴻池 祐輔 KOUNOIKE Yuusuke

1 はじめに

混合整数計画問題(Mixed Integer Programming problem, MIP problem)は，線形計画問題の一部またはすべての変数に整数制約が付加された問題であり，多くの応用例が知られている．それと同時に，最適解を求めることが非常に難しい問題であることも知られている．しかし，近年の分枝カット法 (branch-and-cut algorithm) およびそれを取り巻く環境の進歩によって，大規模な問題も実際に解けるようになっている．分枝カット法は，分枝限定法における子問題の評価に切除平面法を適用した解法であり，分枝限定法と同様に分枝カット法に対する汎用並列フレームワークが複数の研究者によって開発されている．

本稿では，混合整数計画ソルバーの並列化を試みる．これは，分枝カット法を(線形計画ソルバーを用いることを例外として)一から構築する標準的な方法とは異なり，混合整数計画ソルバーそのものの並列化である．本研究で使用する混合整数計画ソルバーはILOG社 CPLEX MIP Optimizer Version 8.0[1] であり，その並列化に汎用並列分枝限定法ツールPUBB2を用いた．本研究を行う背景は次の通りである．

(i) PUBB では，問題特有の実装部分と分枝限定法に共通する部分を明確に分離している．PUBB2では，さらに動作環境の分離を果たしている．本研究は，PUBB2プロジェクトの一環であり，混合整数計画ソルバーを問題特有の実装部分ととらえている．

(ii) それ以上に重要なこととして，近年，混合整数計画ソルバーが急速に進歩していることが挙げられる．本研究で使用するCPLEX MIP Optimizerは，現在利用可能な最高水準のソルバーの一つであり，PUBB2のような汎用並列フレームワークによる高速化をを評価することは重要なことと考えられる．

2 PUBB2による並列化

次の形の混合整数計画問題 (MIP) を考える：

$$\begin{aligned} &\text{最小化} \quad c^{\top}x \\ &\text{条　件} \quad Ax \leq b,\ x_j : \text{整数}\ (j = 1, \ldots, p). \end{aligned}$$

ただし，$c \in R^n$, $A \in R^{m \times n}$, $b \in R^m$, $p \leq n$ である．分枝操作によって，子問題 (MIP_k) は

$$\begin{aligned} &\text{最小化} \quad c^{\top}x \\ &\text{条　件} \quad Ax \leq b,\ x_j : \text{整数}\ (j = 1, \ldots, p), \\ &\qquad \ell_j \leq x_j \leq u_j\ (j = 1, \ldots, p) \end{aligned}$$

の形をした混合整数計画問題となる．本研究で提案する方法は，基本的に複数のCPLEX MIP Optimizerが相異なる子問題，すなわち混合整数計画問題 (MIP_k) に対する分枝カット法を実行する．紙面の都合上，分枝カット法に関する記述は省略する．混合整数計画問題に対する分枝カット法は[2](およびその中の参考文献)等に詳しい．

PUBB2[3] は，PUBB(Parallelization Utility for Branch-and-Bound algorithms) を再構成したものである．PUBB2は，ユーザ層，PUBB2コア層およびプロバイダ層の3層から成る．PUBB2はC++で実装されており，PUBB2コア層には分枝限定法を実行するために必要な抽象クラス群が用意されている．ユーザ層は問題固有の実装，プロバイダ層は動作環境の実装に関するものであり，それぞれPUBB2コア層に用意されている抽象クラスの派生クラスを実装する．

本研究では，CPLEX MIP Optimizerに基づいたユーザ層の実装を行う．具体的には，CPLEX MIP Optimizerに用意されているコールバック関数/クラスを利用する．

プロバイダ層には，逐次処理のための実装やGRID環境で実行するための実装等が含まれる．本研究では，PCクラスタ環境でMaster-Slave型の実装を行った．実行の際には複数のタスクが起動

する．各タスクは未処理の子問題を保持する子問題プール(ローカルプール)を有しており，ローカルプールの情報を基にタスク間で子問題の送受信が行われる．各タスクはCPLEX MIP Optimizerを実行し，与えられた子問題に対する分枝カット法を実行する．そして，タスク間でやりとりされる子問題の送受信のため，ローカルプールとCPLEX MIP Optimizer独自のプール(内部プール)との間で子問題の送受信が行われる．このような実装に加え，(1) ローカルプールと内部プールの通信を頻繁に行うのは望ましくない．実際，あるタスクでローカルプールが空となった場合にのみ通信を行えば十分である．よって，通常プロバイダ層での実装となる，負荷分散に関する関数をユーザ層に用意した．(2) 通常のMaster-Slave型の実装とは異なり，MasterもMIP Optimizerを実行する．よって，Masterのローカルプールが空の場合にのみ，分枝操作により得られる2つの子問題のうちの一つをローカルプールに出力する，などさまざまな工夫を施した．

3 数値実験

16台のPCから成るPCクラスタを用いた．各PCはPentium III 1GHzを2つ搭載，メモリ1GHzであり，Intel Pro/1000 MT LAN desktop Adapterを使用した．16台のPCはNETGEAR GS 516T Gigabit Switchにより繋がれている．また，プロバイダ層の実装にPVM3.4.3を用いた．

問題集は，H. Mittelmannのホームページ[1]を参考にした．まず，CPLEX MIP Optimizer単体ですべての問題を実行し，適度に難しい問題を並列環境で実行した．表1は，逐次処理での計算時間および加速率(逐次計算時間との比)の結果の一部である．この表では，問題名に続いて問題の規模を(制約数，変数の数，0-1整数変数の数，一般整数変数の数)として表示している．また，分枝カット法を実行する前に行われる前処理を適用する場合としない場合について実験を行った．なお，この表の数字は5回の試行による平均値である．

この表から，まず前処理が非常に有効であることがわかる．また，加速率は個々のインスタンスに強く依存することがわかる．表には示していないが，暫定解が最適解に等しくなる時間が加速率に影響を与えており，これが理由のひとつとして考えられる．また，他の理由を探るため，計算時間の関数として上界値および下界値をプロットした．その結果，十分な加速率が得られないケースでは，ほとんど分枝なしで解き終わってしまうような子問題の送受信が頻繁に行われていることがわかった．よって，このような無駄な送受信を回避するための負荷分散を提案することが今後の課題の一つといえる．

表 1: 逐次処理での計算時間と加速率

前処理	逐次計算時間(秒)	タスク数 2	4	8	16	24
問題名 qiu (1192, 840, 48, 0)						
有	1412.74	0.93	1.54	2.23	2.77	1.18
無	1211.84	1.18	1.16	1.47	21.01	38.68
問題名 swath2 (884, 6805, 2406, 0)						
有	5092.63	1.75	2.86	3.53	3.74	3.64
無	2795.99	0.58	1.07	3.97	3.83	3.48
問題名 seymour1 (4944, 1372, 451, 0)						
有	8605.53	1.13	1.80	1.56	1.72	2.16
無	8501.69	1.37	2.03	2.32	2.65	2.79
問題名 markshare2_1 (7, 74, 54, 0)						
有	1982.47	0.91	3.13	8.62	22.85	11.83
無	7954.91	7.21	20.57	18.73	17.69	55.76
問題名 mod011 (4480, 10958, 96, 0)						
有	4886.01	1.47	2.28	3.24	3.82	2.58
無	17314.76	1.59	2.14	3.03	3.11	3.28
問題名 fast0507 (507, 63009, 63009, 0)						
有	23108.70	0.94	2.86	3.07	4.71	4.53
無	28592.92	1.66	2.32	2.43	4.63	5.50

最後に，CPLEX MIP Optimizerで実行される分枝カット法の情報がすべて得られるわけではなかったことも報告する．例えば，分枝カット法が保持するグローバルな情報(グローバルカットや疑似コスト等)を利用することができなかったが，これらは効率的な並列化に必要とされる．したがって，このような，高速化のために必要とされる付加的な実装も今後の課題と言うことができるだろう．

[1] ftp://plato.asu.edu/pub/milpc.txt

参考文献

[1] ILOG CPLEX 8.0 Reference Manual (2002).

[2] A, Martin, "General Mixed Integer Programming: Computational Issues for Branch-and-Cut Algorithms," Lecture Notes in Computer Science, 2241 (2001) 1-25.

[3] Y. Shinano, Y. Kounoike and T. Fujie, "PUBB2: A Redesigned Object-Oriented Software Tool for Implementing Parallel and Distributed Branch-and-Bound Algorithms," submitted.