数値制約ソルバーのスケーラブルな並列化

石井 大輔　美添 一樹　鈴村 豊太郎

本研究では，数値制約充足問題の区間計算に基づくソルバーをスケーラブルに並列化するための手法を提案する．提案する並列化手法は各 CPU コアにワーカーを配置し，大域的負荷分散スキーム (Global Load Balancing, GLB) に基づきワーカ同士が協調しながら並列に探索処理を行う．GLB 法では，ランダムなワーカ選択と，ライフライン (超立方格子状のワーカ間ネットワーク) に基づいた 2 段階のワークスティーリングを実施し，branch and prune アルゴリズムが扱う探索空間を分散する．提案手法を並列プログラミング言語 X10 を用いて実装した．TSUBAME2.5 の 900 コア上で，ベンチマーク問題を用いた実験結果では，最大 751 倍の速度向上を達成した．

1　はじめに

数値制約充足問題 (NCSP, 3 節) と NCSP の自動求解ツールであるソルバーは，実数領域の諸々の問題のモデリングと高信頼な解析のための有用な技術である [20] [6] [4]．NCSP では，算術的な制約と，その充足解を探索すべき領域を表す矩形 (区間ベクトル) が与えられ，branch and prune アルゴリズムが，矩形を区分けしたり，矛盾する箇所を削減したりしながら，解集合を指定精度で包む矩形集合を計算する．しかし，NCSP の求解処理は計算量が大きく，既存の逐次ソルバーでは扱えるインスタンスの数が限られていた．そこで，多数の CPU コア上でスケールするように NCSP ソルバーを並列化することができれば有用である [7]．

探索空間を区分け・分散する手法は，一般に CSP ソルバーを並列化する際の単純かつ効率的なアプローチだと考えられている．しかし最新の (実数ドメイン以外の) 並列 CSP ソルバーでは数 100 コア程度での並列化にとどまっている (2 節)．最近，Saraswat ら [18] が大域的負荷分散 (GLB) の枠組みを提案した．GLB は，不均一な並列タスクのための大域的なワークロードの分散機能とタスクの終了判定機能を提供するものである (4. 1 節)．

本研究では，GLB を用いた並列 NCSP ソルバーを提案する (4. 2 節)．ソルバーは並列プログラミング言語 X10 とその標準ライブラリが含む GLB ライブラリにより簡潔に実装され，GLB の単位タスクとして IBEX ライブラリが提供する数値制約伝播器の逐次実装を利用している．5 節では，提案する並列ソルバーを TSUBAME2.5 の 900 コア上で実行した実験結果を報告する．

2　関連研究

Gent らによるサーベイ [8] では，並列 CSP ソルバーを，(1) 探索空間の区分け・分散によるもの，(2) 異質なワーカ群の協調によるもの (cf. ポートフォリオ)，(3) 制約伝播処理の並列化によるもの，の 3 つに分類している．本研究は (1) のアプローチをとる．

探索空間の分散において難しいのは，部分木を均一になるように分散し，各ワーカをアクティブに保つことである．探索木が不均衡になるとき，探索木の適切

Scalable Parallelization of a Numerical Constraint Solver.

Daisuke Ishii, 東京工業大学, Tokyo Institute of Technology.

Kazuki Yoshizoe, 東京大学, University of Tokyo.

Toyotaro Suzumura, IBM ワトソン研究所, IBM T.J. Watson Research Center.

な区分け方法を予測するのは難しく，求解中に動的に負荷分散することが必要となる．そのために，ワークスティーリング手法を用い，あるワーカのワークロードが枯渇しているとき，当該ワーカが他ワーカへリクエストを送信し，その対応としてワークロードを得ることができる．多くの既存並列ソルバーでは，多少なりとも中央集権的な構成をとっており，スケール度合いが限定されていた (例: 40 コア [19], 64 コア [13] [3], 256 コア [2])．

著者らは，GLB を用いて同一の処理を行うワーカを協調させた並列 NCSP ソルバーを開発してきた [7] [12]．本研究では，NCSP 実装を Realpaver から IBEX に変更するとともに (3.1 節参照)，よりスケーラブルな並列化を目指す．

最適化問題のための branch and bound アルゴリズムについて，ワークスティーリングに基づき探索木を分散させ，並列化を行う手法が多数提案されている [9] [14] [16]．アルゴリズムが NCSP の求解処理と似ているものの，既存の研究は離散ドメインを対象としていた．本研究では，実数ドメインを区間近似する求解処理の並列化に取り組む．

3 数値制約プログラミング

数値制約プログラミングは離散ドメインの制約プログラミング [17] の拡張として提案され，区間解析 [15] の技法を援用しながら研究が行われてきた．数値制約は実数 $\mathbb{R}$ の部分集合を変数ドメインとする．とくに，**区間** $[a,b] := \{r \in \mathbb{R} \mid a \leq r \leq b\}$ (a と b は浮動小数点数) と**矩形** (box, 区間ベクトル) $([a_1,b_1],\ldots,[a_n,b_n])$ を扱う．本稿ではボールド体 (例: $\boldsymbol{v}$) で区間と矩形を表す．$\mathbb{I}$ で区間の集合を表し，$\mathbb{I}^n$ で n 次元の矩形の集合を表す．数値制約の求解は精度保証付き数値計算と branch and prune アルゴリズムに基づき行われる．数値制約ソルバーは，制約を記述する実数関数の**区間拡張**を計算し，高信頼に制約を評価する．例として，加算の区間拡張を $[a_1,b_1]+[a_2,b_2] := [\lfloor a_1+a_2 \rfloor, \lceil b_1+b_2 \rceil]$ と与えることができる ($\lfloor\cdot\rfloor$ と $\lceil\cdot\rceil$ は下向き・上向き丸めを表す)．Branch and prune アルゴリズムは，制約を (区間評価により) 充足する付値を探索する分割統治法である．

数値制約充足問題 (numerical constraint satisfaction problem, NCSP) を 3 つ組として定義する．各要素は，変数 $v = (v_1,\ldots,v_n)$，矩形として与えられる初期ドメイン $\boldsymbol{v}_0 \in \mathbb{I}^n$，連立式の形をした制約 $c(v) := f(v) = 0 \wedge g(v) < 0$ ($f : \mathbb{R}^n \to \mathbb{R}^{\#f}$, $g : \mathbb{R}^n \to \mathbb{R}^{\#g}, \#f, \#g \in \mathbb{N}$) となっている．NCSP の**解**は制約 $c(\tilde{v})$ を充足する変数への付値 $\tilde{v} \in \boldsymbol{v}_0$ である．集合 $\{\tilde{v} \in \boldsymbol{v}_0 \mid c(\tilde{v})\}$ を**解集合**という．

3.1 Branch and prune アルゴリズム

Branch and prune アルゴリズム [20] は NCSP を求解するための標準的な方法である．アルゴリズムは，NCSP と精度 $\epsilon \in \mathbb{R}_{>0}$ を入力とし，解集合を精度 ϵ で近似する矩形集合 $S \in \mathbb{I}^n$ を出力する．

アルゴリズムを図 1 に示す．矩形を格納するキュー L を用意し，2–11 行目のループにおいてキューが空になるまで以下の計算を行う．3 行目では，キューから矩形を取り出し，Prune 手続きを適用して矩形の端領域の削減を行う．本研究では Prune の実装として，HC4Revise [1] 手続きや区間ニュートン法に基づく基本的な実装を用いた．矩形に Prune を適用すると，矩形は (i) 解を含まない，(ii) 精度が良い (1 辺の最大幅が ϵ 以下)，(iii) 解集合の内部，(iv) その他，のいずれかに評価される．(i) の場合，矩形は捨てられ (4 行目)，(ii) と (iii) の場合，矩形は S に追加され (6 行目)，(iv) の場合，Branch に渡される (8 行目)．Branch 手続きは矩形をある辺の中点で分割し，その結果をキュー L に追加する (8 行目)．本研究では，辺の選択にラウンドロビン法を用いた．

NCSP ソルバーの (逐次) 実装として，IBEX[†1] や Realpaver [10] が開発されている．本研究では，Prune の実装として IBEX の一部を用いた．

数値ドメインの CSP に特徴的な，並列求解に関する性質をいくつか挙げることができる．まず，Prune の計算量が大きいため (本研究の実験では平均で約 1 ミリ秒かかる)，求解プロセスのボトルネックとなってしまう点が挙げられる．つぎに，Prune の適用により，探索木が不均衡となる点が挙げられる．Prune は

†1 http://www.ibex-lib.org/

Input: NCSP $(v, \boldsymbol{v}_0, c)$, precision ϵ
Output: set of boxes S

$L := \{\boldsymbol{v}_0\}; S := \emptyset$
while $L \neq \emptyset$ **do**
  $\boldsymbol{v} := \mathsf{Prune}_c(\mathsf{PollFirst}(L))$
  if $\boldsymbol{v} \neq \emptyset$ **then**
    if $\mathrm{wid}\,\boldsymbol{v} \leq \epsilon \vee \boldsymbol{v}$ is an inner box **then**
      $S := S \cup \{\boldsymbol{v}\}$
    else
      $L := \mathsf{AddLast}(L, \mathsf{Branch}(\boldsymbol{v}))$
    end if
  end if
end while
return S

図 1　Branch and prune アルゴリズム

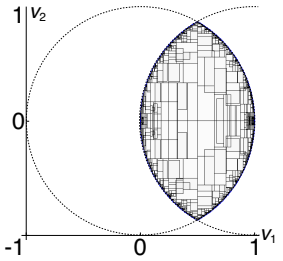

図 2　解集合を近似する矩形集合

探索の対象である矩形を大きく削減したり，さらには矩形全体を除去することもある．そのため，探索木の各節点において Prune を適用して探索空間を削減するのが望ましいが，他方，均等な分散が難しくなってしまう．3 つ目に，NCSP の制約の数が変数の数よりも少なく，かつ小さい ϵ が与えられると，解の矩形の数が大きくなり，探索木が末広がりの形となる点が挙げられる．そのため，各枝を並列求解することの効果が期待できる．最後に，探索木の異なる枝の求解処理間では通信が不要である点が挙げられる．そのため，各枝の処理には既存の逐次実装をそのまま利用することができる．

3.2　例

(v_1, v_2) 平面の 2 つの円盤の共通部分を NCSP $(v, \boldsymbol{v}_0, c)$ として表すことができる．ただし，$v := (v_1, \ldots, v_4)$，$\boldsymbol{v}_0 := ([-1,1],[-1,1],[0,1],[0,1])$，$c := (v_1^2 + v_2^2 - v_3,\ (v_1-1)^2 + v_2^2 - v_4) = 0$ とする．$\epsilon = 0.01$ として branch and prune アルゴリズムで求解した結果を図 2 に示す．

4　並列化手法

本研究では，探索木を分割し，並列動作するワーカ群へと分散することにより，branch and prune アルゴリズムを並列化する手法を提案する．しかし，効率的な並列化のために，探索木を均等に分散する方法は自明でない．本研究では，並列プログラミング言語 X10 [5] とその大域的負荷分散ライブラリを用いた手法を提案する．

4.1　X10 GLB ライブラリ

GLB は X10 の標準ライブラリに含まれる**大域的負荷分散** (global load balancing) 機能の実装である．ライフラライングラフに基づくワークスティーリングアルゴリズム [18] を実装している．GLB は，AI 分野における探索のような，部分タスクの負荷の予測が難しいタスクの並列化に適している．

GLB の計算は協調的な**ワーカ**群によってなされる．各 X10 プレースにワーカが 1 つ配置される．各ワーカはアクティブかアイドルの 2 状態をとりつつ，同一の手続きにより保持するワークロードを処理する．

アクティブ状態: ワークロードを保持しているワーカは，タスクを一定量終えるごとに，他ワーカからのリクエストを受け取るキューを確認し，保持するワークロードの一部をリクエスト元のワーカへ送信する．

アイドル状態: ワークロードが枯渇したワーカは，2 フェーズからなるワークスティーリング処理を行う．まず**第 1 フェーズ**として，当該ワーカは，ランダムに他ワーカを選び，リクエストを送信し，リクエスト先からワークロードが分け与えられることを期待する．リクエスト先からワークロードが得られなかった場合は，**第 2 フェーズ**として，**ライフライン** (lifeline) と呼ばれる超立方体状のワーカ間ネットワーク上で自

分と隣接するワーカに対してリクエストを送信する．第 2 フェーズのライフライン越しの通信は全体のタスク終了処理も兼ねている．

GLB は X10 で実装されており，複数ベンチマークタスクについて 16,000 プレース程度までスケールすることが報告されている [21]．

GLB にはおもに 4 つのパラメタがある．$n \in \mathbb{N}$ は，主タスクの個数を表し，ワーカがワークロードの分散処理を行う間隔を指定する．$w \in \mathbb{N}$ はワークスティーリングの第 1 フェーズで送信するランダムリクエストの個数を指定する．$l \in \mathbb{N}$ と $z \in \mathbb{N}$ は，ライフライングラフの 1 辺のノード数と，1 ノードでの分岐数を指定する．$w := 0$ とすると，第 1 フェーズが無効化される．l と z については，まず l を指定し，z を l^z がワーカ数以上の最小値となるようにすればよい．l を最小値 2 にすると直径が最小の密なライフライングラフとなる．

4.2 GLB を用いた NCSP ソルバーの実装

GLB の TaskQueue インターフェースを NCSP に適合させたクラス QueueImpl の実装を行った．QueueImpl は探索すべき矩形を格納したキュー L と解の矩形集合 S を保持し，branch and prune アルゴリズムをカプセル化する．TaskQueue の各メソッドを以下のように実装した．

- process(n) は，branch and prune アルゴリズムのメインループの繰り返し n 回分，求解処理を行う．
- split() は L を均等に分割し，片方を L に保持し，もう片方を戻り値とし，GLB ライブラリに分散させる．
- merge($\tilde{L}$) は矩形の集合 $\tilde{L}$ を (リクエスト先から) 受け取り，L に格納する．

実装は，X10 部分が約 1,000 行，C++部分が約 2,400 行となっており，`https://github.com/dsksh/icpx10` にて入手可能である．

5 評価実験

文献由来の問題を用い，TSUBAME2.5 上で並列 NCSP ソルバーの評価を行った．

5.1 TSUBAME2.5 スーパーコンピュータ

TSUBAME2.5 は東京工業大学のスーパーコンピュータである．[†2] TSUBAME2.5 の各ノードは 2 つの Intel Westmere EP 2.93GHz プロセッサ (計 12 コア) と 54GB のローカルメモリを備える．実験は，75 ノード (900 コア) 上で，高々900 X10 プレースを用いて行った．X10 は，バージョン 2.4.3.2 のネイティブコンパイラと MPI バックエンド (Open MPI 1.6.5) を使用した．

5.2 実験結果

経済学モデルに関する NCSP [11] の 2 インスタンスと，球と平面の交差領域を表した NCSP [6] の 2 インスタンスを求解した．GLB のパラメタは，$n := 1$，$w :\in \{0, 1\}$，$l := 2$，$z := \lceil \log_l P \rceil$ と設定した．各インスタンスの仕様と実験結果を表 1 に示す．列 “problem”，“size”，“ϵ”，“# sol”，“# br”，“w” は，問題名，変数の数，精度，解の矩形の数，Prune 適用の数，パラメタ w の値を示している．以降の列は実験結果を示している．t_j は j プレースを用いた実行時間 (最短値に下線)，ar_{900} は，各ワーカが branch and prune 処理をしている時間の求解時間全体に占める割合の平均値 (900 プレース使用時)，$\#\ \mathrm{sb}_{900}$ は負荷分散のために 900 プレース使用時にプレース間で送信された矩形の総数を表している．並列求解処理の速度向上を図 3 に示す．

5.3 考察

実験において，並列ソルバーは 900 プレース・コアまでスケールし，751 倍の速度向上を達成した (並列化効率 0.83)．

サイズが小さい 2 インスタンス (*eco8* と *sp2-2*) において，最高の速度向上は設定 $w = 0$ により得られた．これらの求解処理では，$w = 0$ のときワーカのアクティブ率 (表 1 の ar_{900} を参照) が高くなった．負荷分散の第 1 フェーズを無効化し，最初からライフラインを用いた通信オーバーヘッドの大きい負荷分散を実施している ($\#\ \mathrm{sb}_{900}$ を参照)．小さいインスタ

†2 `http://tsubame.gsic.titech.ac.jp/en`

表 1　実験結果

problem	size	ϵ	# sol	# br	w	t_1	t_{600}	t_{900}	ar_{900}	# sb_{900}
Economics	8	10^{-8}	8	42 279	0	37s	0.36s	0.31s	40%	36 K
(*eco8*)					1		0.68s	0.57s	17%	21 K
Economics	10	10^{-8}	16	1 967 440	0	3 090s	6.9s	5.1s	75%	1 440 K
(*eco10*)					1		7.1s	4.8s	84%	601 K
2D sphere and	2+2	0.001	3 042 524	3 312 827	0	610s	1.4s	1.2s	84%	3 370 K
plane (*sp2-2*)					1		1.7s	1.5s	66%	1 080 K
4D sphere and	2+4	0.001	19 269 626	26 859 765	0	8 550s	16.7s	11.7s	91%	20 300 K
plane (*sp2-4*)					1		17.0s	11.4s	94%	6 290 K

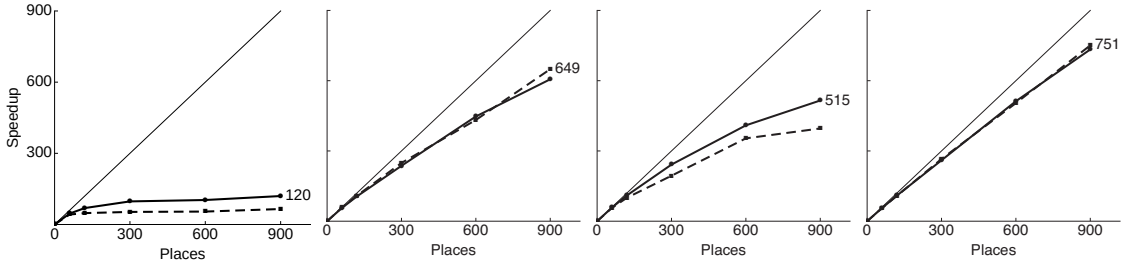

図 3　求解処理の速度向上

(左から順に *eco8*, *eco10*, *sp2-2*, *sp2-4* の結果を示す．実線は $w = 0$，点線は $w = 1$ の結果に対応する．)

ンスでは求解時間が限られるため，高速な負荷分散を行い，すぐに終了処理へ移ることが有利になっていると考えられる．

サイズが大きい 2 インスタンス (*eco10* と *sp2-4*) については，ランダムなワークスティーリングを有効にした設定 ($w = 1$) が無効にした設定よりも効率が良かった．$w = 1$ の場合，負荷分散のための通信回数を節約できるメリットがあり，これが大きいインスタンスでは有効に働いていると考えられる．

ワークロード分散処理の間隔を指定する n に関しては，最も頻繁な設定 $n = 1$ とした．branch and prune アルゴリズムの各繰り返し毎に分散処理が行われることになるが，Prune 1 回の処理は 1 ミリ秒前後かかるため，十分なワークロード分散のためにこの頻度が必要となる．

他に本実験の特徴として，求解にかかる実行時間が短くてもある程度の速度向上が得られた点が挙げられる．たとえば，*eco8* と *sp2-2* は逐次処理に 37 秒および 610 秒かかるが，900 プレース使用時に 120 倍および 515 倍の速度向上が得られている．

6　まとめと今後の課題

本論文では，branch and prune アルゴリズムによる NCSP 求解を GLB を用いて並列化する手法について述べた．実験では 900 プレース・コアまで線形に近い速度向上が得られることを確認した．

今後の課題として，数値最適化問題の求解手続きを並列化したり，より大規模な並列化に取り組むことが挙げられる．

謝辞　本研究は科研費 25880008, 15K15968, 25700038, 26280024, 23240005 と，JST ERATO の補助を得て行った．

参考文献

[1] Benhamou, F., Granvilliers, L., Goualard, F., and Puget, J.-F.: Revising Hull and Box Consistency, *ICLP*, 1999, pp. 230–244.

[2] Bergman, D., Cire, A. A., Sabharwal, A., Samulowitz, H., Saraswat, V., and Hoeve, W.-J. V.: Parallel Combinatorial Optimization with Decision Diagrams, *CPAIOR*, *LNCS* 8451, 2014, pp. 351–367.

[3] Bordeaux, L., Hamadi, Y., and Samulowitz, H.: Experiments with Massively Parallel Constraint Solving, *IJCAI*, 2006, pp. 443–448.

[4] Caro, S., Chablat, D., Goldsztejn, A., Ishii, D.,

and Jermann, J.: A branch and prune algorithm for the computation of generalized aspects of parallel robots, *Artificial Intelligence*, Vol. 211(2014), pp. 34–50.

[5] Charles, P., Grothoff, C., and Saraswat, V.: X10: an object-oriented approach to non-uniform cluster computing, *Proc. of OOPSLA*, 2005, pp. 519–538.

[6] Ishii, D., Goldsztejn, A., and Jermann, J.: Interval-based projection method for under-constrained numerical systems, *Constraints Journal*, Vol. 17, No. 4(2012), pp. 432–460.

[7] Ishii, D., Yoshizoe, K., and Suzumura, T.: Scalable Parallel Numerical CSP Solver, *CP*, *LNCS* 8656, 2014, pp. 398–406.

[8] Gent, I. P., Jefferson, C., Miguel, I., Moore, N. C. A., Nightingale, P., Prosser, P., and Unsworth, C.: A Preliminary Review of Literature on Parallel Constraint Solving, *Workshop on Parallel Methods for Constraint Solving*, 2011.

[9] Grama, A., Gupta, A., Karypis, G., and Kumar, V.: *Introduction to Parallel Computing*, Addison Wesley, 2003.

[10] Granvilliers, L. and Benhamou, F.: Algorithm 852: RealPaver: An Interval Solver using Constraint Satisfaction Techniques, *ACM Transactions on Mathematical Software*, Vol. 32, No. 1(2006), pp. 138–156.

[11] Hentenryck, P. V., Michel, L., and Benhamou, F.: Newton: Constraint Programming over Nonlinear Constraints, *Science of Computer Programming*, Vol. 30, No. 1-2(1998), pp. 83–118.

[12] Ishii, D., Yoshizoe, K., and Suzumura, T.: Scalable Parallel Numerical Constraint Solver Using Global Load Balancing, *ACM SIGPLAN Workshop on X10*, 2015, pp. 33–38.

[13] Jaffar, J., Santosa, A., Yap, R., and Zhu, K.: Scalable distributed depth-first search with greedy work stealing, *ICTAI*, IEEE, 2004, pp. 98–103.

[14] Lüling, R., Monien, B., Reinefeld, A., and Tschöke, S.: Mapping Tree-Structured Combinatorial Optimization Problems onto Parallel Computers, *Solving Combinatorial Optimization Problems in Parallel*, Vol. 7141, No. 7141, 1996, pp. 115–144.

[15] Moore, R. E.: *Interval Analysis*, Prentice-Hall, 1966.

[16] Otten, L. and Dechter, R.: Towards Parallel Search for Optimization in Graphical Models, *ISAIM*, 2010.

[17] Rossi, F., Beek, P. V., and Walsh, T.: *Handbook of Constraint Programming*, Foundations of Artificial Intelligence, Vol. 2, Elsevier, 2006.

[18] Saraswat, V., Kambadur, P., Kodali, S., Grove, D., and Krishnamoorthy, S.: Lifeline-based global load balancing, *PPoPP*, 2011, pp. 201–212.

[19] Schulte, C.: Parallel search made simple, *TRICS (Techniques foR Implementing Constraint programming Systems)*, 2000, pp. 41–57.

[20] Van Hentenryck, P., McAllester, D., and Kapur, D.: Solving Polynomial Systems Using a Branch and Prune Approach, *SIAM Journal on Numerical Analysis*, Vol. 34, No. 2(1997), pp. 797–827.

[21] Zhang, W., Tardieu, O., Grove, D., Herta, B., Kamada, T., and Saraswat, V.: GLB : Lifeline-based Global Load Balancing Library in X10, *PPAA*, 2014, pp. 31–40.