Vol.2013-HPC-141 No.10
2013/10/1

PGAS言語X10による 数値制約充足問題ソルバーRealpaverの並列化

石井 大輔[1,a)]　鈴村 豊太郎[1,2,3,b)]

概要：数値制約充足問題 (NCSP) は，制御工学やロボティクス等の問題を実数変数を持つ等式・不等式制約を用いて記述し，充足解を求める問題である．実数変数のドメインを区間ベクトルとして与え，制約伝播と探索を駆使し，指定精度で解を包む区間ベクトル集合を効率よく計算する手法が発展している．有限・離散ドメインの制約充足問題については並列化手法が研究されているものの，実数ドメインにおける多くの提案手法は逐次処理を前提としており，その並列化手法は明らかでなかった．とくに under-constrained NCSP では，領域状の解集合を多数の区間ベクトルで包む計算を行うが，このような求解プロセスには並列性が見込まれる．本研究では，PGAS 言語 X10 を用いて既存の NCSP ソルバー Realpaver を拡張し，計算機クラスタ上でスケール可能な並列ソルバーを構築した．提案手法では，探索木を複数プレースに分散し，各プレースで自律的な求解タスクを走らせることで並列化を行う．また，均等に探索木を分散するための前処理を提案する．東京工業大学 TSUBAME2.0 上で 16 プレースを用いた実験結果では，well-constrained NCSP に対して最大 7.9 倍，under-constrained NCSP に対して最大 9.7 倍の速度向上を得た．

1. はじめに

数値制約充足問題 (NCSP, 3.2 節) は，実数変数を持つ等式・不等式制約の充足解を求める問題である．制御工学やロボティクス等の問題を NCSP として直截的に記述し，ソルバーを用いて高信頼な求解を行うことができる (例: [1])．Branch-and-prune と呼ばれるアルゴリズムに基づき，実数変数のドメインを box (区間ベクトル) として与え，区間計算 (3.1 節)，局所無矛盾性に基づく制約伝播，探索を駆使し，指定精度で解を包む box 集合を効率よく計算する手法が発展している (4 章)．有限ドメインの制約充足問題については並列化手法が研究されているものの [2][3][4]，実数ドメインにおける多くの提案手法は逐次処理を前提としており，その並列化方法は明らかでなかった．

近年，under-constrained NCSP あるいは限量化 NCSP (制約中で限量子とパラメタ変数を用いた NCSP) の求解技術が発展している [5][1]．Under-constrained NCSP は領域状の解集合を持ち，制約求解系は解集合を多数の box で包む計算を行う (例: 図 5)．点状の解を探索する well-constrained NCSP の求解プロセスに較べ，under-constrained NCSP の求解プロセスには並列性が見込まれる．

本研究では，PGAS 言語 X10 (3.3 節) を用いて既存の NCSP ソルバー Realpaver[6] を拡張し，計算機クラスタ上でスケール可能な並列ソルバーを構築する．提案手法では branch-and-bound アルゴリズムの並列化手法を参考に，複数計算ノードの並列タスクが自律的に並列処理を行う構成とする (5 章)．求解中に計算ノード間で動的に探索空間を送受信し，負荷分散を行う．また，under-constrained NCSP の求解を効率よく並列化するために，上記並列処理の前処理として幅優先探索を行い，より均等な探索空間の分割を行う手法を提案する (5.3 節)．本研究では，Realpaver の逐次実装をそのまま利用し，提案手法の「軽量な」実装を行った (6 章)．大規模クラスタ (東京工業大学 TSUBAME2.0) 上で実装を用いてベンチマーク問題を求解する評価実験を実施した (7 章)．

2. 関連研究

有限ドメイン・連続ドメインの制約充足問題の並列求解法が研究されている．サーベイ [7] では既存研究を，(a) 制約伝播手続きの並列化手法，(b) 異質な求解エージェント群を協調させる手法 (cf. ポートフォリオ手法)，(c) 探索空間を分割し，複数ワーカで探索する手法 [2][3][4]，に分類している．本研究は (c) に属する．NCSP 求解について，(a) や (b) のアプローチでは研究されているものの [8][9][10]，

1 東京工業大学 - Tokyo Institute of Technology
2 IBM 東京基礎研究所 - IBM Research Tokyo
3 JST CREST
a) dsksh@acm.org
b) suzumura@cs.titech.ac.jp

(c) のアプローチによる手法は著者らの知る限り提案されていない．

Branch-and-bound アルゴリズムの探索木を複数プロセッサ上のワーカに分散し，探索を行う手法が多く研究されている [11][12][13]．また制約充足問題ソルバー [2][3] や SAT ソルバー [14][4] についても同様の研究がなされている．本研究では [11] を参考に並列探索処理を設計した．

[11][13] では，カットオフ深さを設定し，探索木の分散を先延ばしすることで大きな分割木を生成する手法を提案している．本研究でも同様のアイディアにより，under-constrained NCSP の探索木を均等分割するために，分割前に幅優先探索する手法を提案する．

3. 基本技術

3.1 区間計算

区間計算 [15] は浮動小数点数の計算を区間の計算に拡張したものである．区間計算は実数計算の保守的近似であり，結果として得られる区間は実数計算がとりうるすべての結果を計算誤差とともに包む．区間 $\boldsymbol{x} = [\underline{x}, \overline{x}]$ により集合 $\{r \in \mathbb{R} \mid \underline{x} \leq r \leq \overline{x}\}$ を表す．$\overline{x}, \underline{x} \in \mathbb{R}$ で区間 $\boldsymbol{x}$ の上限と下限を表す．本稿ではスカラーと同様に x と $\boldsymbol{x}$ で実ベクトルと区間ベクトルを表す．d 次元 **box** (区間ベクトル) $\boldsymbol{x}$ は d 区間の組 $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d)$ である．

3.2 数値制約プログラミング

数値制約充足問題 (numerical constraint satisfaction problem, NCSP) [16] を下記の要素からなる 3 つ組 $(v, \boldsymbol{v}_0, c)$ で表す．

- d **変数**の組 $v = (v_1, \ldots, v_d)$.
- d 次元 box として与えられる**初期ドメイン** $\boldsymbol{v}_0$.
- **制約** $c \equiv f(v) = 0 \wedge g(v) \geq 0$ (ただし $f : \mathbb{R}^d \to \mathbb{R}^p$, $g : \mathbb{R}^d \to \mathbb{R}^q$).

本稿では d, p, q で問題中の変数，等式，不等式の数を表す．制約を満たす変数 v への値 $\tilde{v} \in \boldsymbol{v}_0$ の付値を NCSP の**解**という．また集合 $\Sigma := \{\tilde{v} \in \boldsymbol{v}_0 \mid c(\tilde{v})\}$ を**解集合**という．NCSP は，$d < p$ がなりたつとき **over-constrained** (制約過多)，$d = p$ がなりたつとき **well-constrained** (制約過不足がない)，$d > p$ がなりたつとき **under-constrained** (制約不足) であるという．一般に，well-constrained NCSP は離散的な解集合を持ち，under-constrained NCSP は連続的な解集合を持つ．

NCSP の標準的な求解手法として，4 章で説明する **branch-and-prune** アルゴリズム [17] がある．branch-and-prune アルゴリズムは，**局所無矛盾性** (local consistency) に基づき近似的に制約充足性を判定することにより，NCSP を NP 完全な探索問題として扱う．代表的な局所無矛盾性である Box 無矛盾性に基づく Prune 実装の計算量は $O(ewd^2)$ である (e はユーザ定義制約中の演算子・初等関数の数，w は初期ドメイン一辺の最大サイズ，d は変数の数を表す)[18].

Require: NCSP $(v, \boldsymbol{v}_0, c)$
Ensure: box 集合の組 S
1: $L := \{\boldsymbol{v}_0\}$; $S := \emptyset$
2: **while** $L \neq \emptyset$ **do**
3: $\quad \boldsymbol{v} := \mathsf{Extract}(L)$
4: $\quad \boldsymbol{v} := \mathsf{Prune}(\boldsymbol{v})$
5: $\quad$ **if** $\boldsymbol{v} \neq \emptyset$ **then**
6: $\quad\quad \tilde{v} := \mathsf{Select}(v)$
7: $\quad\quad$ **if** $\tilde{v} \in v$ **then**
8: $\quad\quad\quad L := L \cup \mathsf{Branch}(\tilde{v}, \boldsymbol{v})$
9: $\quad\quad$ **else**
10: $\quad\quad\quad S := S \cup \{\boldsymbol{v}\}$
11: $\quad\quad$ **end if**
12: $\quad$ **end if**
13: **end while**
14: **return** S

図 1 Branch-and-prune アルゴリズム.

3.3 PGAS 言語 X10

X10[19] は，並列分散環境における高性能・高生産プログラミングの実現を目標に，IBM Research により開発されているプログラミング言語である．X10 は計算環境モデルとして **PGAS** (partitioned global address space) を採用しており，複数のプレースにまたがったデータ構造と，軽量な**並列タスク**による並列分散処理とを，簡潔に記述するための構文を備えている．構文の例として，プレース毎の分散処理のための `at`，並列タスクを生成するための `async`，同期のための `finished`，`when`，`atomic` 等がある．

X10 プログラムを Java または C++のプログラムに変換し，実行可能ファイルを生成するコンパイラが開発されている．生成プログラムの実行環境として，ソケット通信バックエンドや MPI バックエンド等があり，用途に応じてコンパイル時に切り替えることができる．また Java と C++との外部インターフェース機構を備え，本研究ではこの機能を使って X10 と C++間の接続を行った．

4. Branch-and-prune アルゴリズム

本章では NCSP 求解アルゴリズム branch-and-prune について説明する．アルゴリズムを図 1 に示す．アルゴリズムは，NCSP を入力として受け取ると，近似的な全解探索を行い，初期ドメイン中の解集合を包む box の集合 S を出力する．branch-and-prune は 4 つのサブルーチンを持ち，問題や求解アプローチに応じて実装を用意する．

- Extract は各繰り返しにおいて，解の探索空間である L から box $\boldsymbol{v}$ を選択する．この Extract の実装によって，深さ優先探索や幅優先探索等の探索法を設定する

ことができる．

- Prune は相補的な 2 つの処理を行う．1 つ目に探索空間 $\boldsymbol{v}$ から制約と矛盾する付値を除去する．NCSP の求解では局所無矛盾性に基づき，ある区間の端部分を取り除く．本研究では数値制約のための局所無矛盾性である Hull 無矛盾性，Box 無矛盾性，区間ニュートン法を複合的に適用する手法 [20] を用いる．2 つ目に，区間ニュートン法等を用い，$\boldsymbol{v}$ 内の解の存在保証を行う [5]．保証に成功した場合は S に追加する．問題が under-constrainded で領域状の解集合をもつ場合には，$\boldsymbol{v}$ が解集合の内部 **box** であることを保証できる．その場合，探索を打ち切り，効率化を図ることができる．
- Select は変数集合 $v = \{v_1, \ldots, v_d, \bullet\}$ から 1 つの要素を選択する．本研究ではラウンドロビン方式で順に変数を選ぶように実装した．また Select は探索打ち切りの判定も行い，要素 $\bullet$ が探索打ち切りを表す．本研究では探索打ち切り条件として，box $\boldsymbol{v}$ の最大幅が規定幅 (求解精度) より小さいかどうか，あるいは内部 box であるかどうかを用いる．
- Branch は box $\boldsymbol{v}$ を 2 分割する．分割された box は L に戻され，以降各 box について探索を行う．分割の際，box のどの辺を選ぶか (変数選択) が探索の効率化のために重要となる．本研究では box 毎にラウンドロビン方式で変数選択を行う．

Branch-and-prune による探索において，条件がよければ Prune 1 回の適用で探索空間を大幅に削減できる (例: Prune として区間ニュートン法を用いた場合，2 次収束)．一方，探索空間が解を含まなければ，多くの場合，Prune 適用の結果として空集合が得られる．このように Prune が探索木を枝刈りし，不平衡化を招くため，一般に branch-and-prune の探索空間サイズを事前に予測するのは難しい．

5. Branch-and-prune の並列化

まず，branch-and-prune アルゴリズムの `while` ループの中身 (3–12 行目) を 1 つの `async` 文とし，並列タスクにすることで，プレース内での探索処理を粗粒度で並列化できる．ただし本研究では，6 章でも述べるように，計算量が大きい Prune 手続きを並列化せず `atomic` 処理とするため，プレース内での並列化による大きな効果は得られないと考えられる．

本研究では，速度向上を得るために，branch-and-prune の探索処理を複数プレースに分散することで並列化を行う．そのために，[11] 11 章で述べられている深さ優先 branch-and-bound アルゴリズムの並列化手法と同様に実装を行う．実装は各プレースにおいて図 2 に示すような同一処理を行う分散的な構成とし，複数プレース間で動的な負荷分散を実施する．各プレースは探索空間の 1 部分を担当して

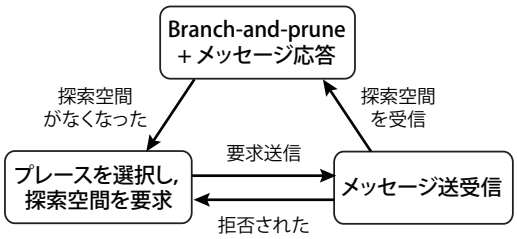


図 2　各プレースにおける処理.

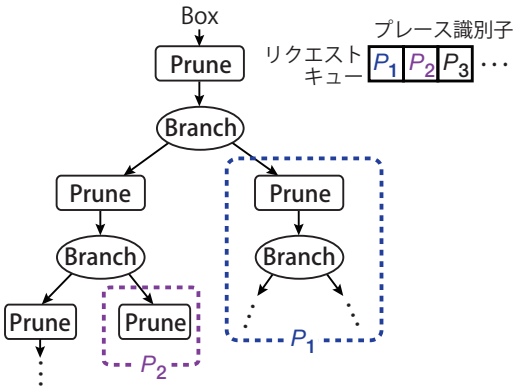


図 3　探索木の分散処理.

探索を行い，プレース間で動的に探索空間 (box) の受け渡しを行う．プレースにおいて手持ちの探索空間がなくなった場合，他のプレースを 1 つ選び，要求を送り，当該プレースが要求に応えて探索空間を送り返すのを待つ．要求先でも探索空間が枯渇していた場合は拒否メッセージを受けるので，再度プレースを選んで要求を送りなおす．計算の終了判定には Dijkstra の方法を用い，全プレースで探索が終了したことを確認する ([11] 11.4.4 節を参照)．

5.1 探索空間の分散手法

各プレースは探索空間要求を受け取ると，探索の最中に探索空間の一部を送信する．プレース横断的な branch-and-prune による探索処理を図 3 に示す．各プレースに探索空間要求を格納するためのキューを設け，要求を受け取ると，要求元のプレース識別子を保持する．図中の探索木の各節点では，入力された探索空間 (box) に Prune を適用した後，Branch を適用して 2 分割している．その際，キューに要求があれば，分割した box の片方をキュー内の識別子 (P_1 や P_2) のプレースに送信する．もう片方については同プレースで探索を進める．Box に Prune を適用した結果が空集合になった場合は探索を終了し，他プレースに探索空間を要求する．図において，P_2 は受け取った box に Prune を 1 回適用したのみで探索を終了している．

5.2 初期ドメインの分散

求解処理の初期においては，探索開始前にあらかじめ各プレースに探索空間の要求を登録しておき，図 4 のようなツリー状に NCSP の初期ドメインがプレース P_0 から全プ

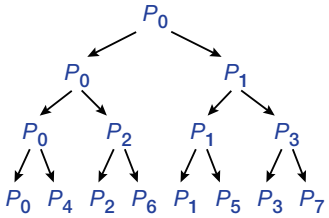


図 4　初期ドメインの 8 プレースへの分散経路.

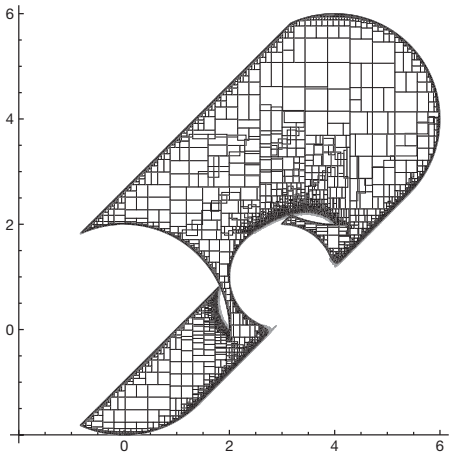


図 5　Under-constrained NCSP 求解結果の例 (*serial*).

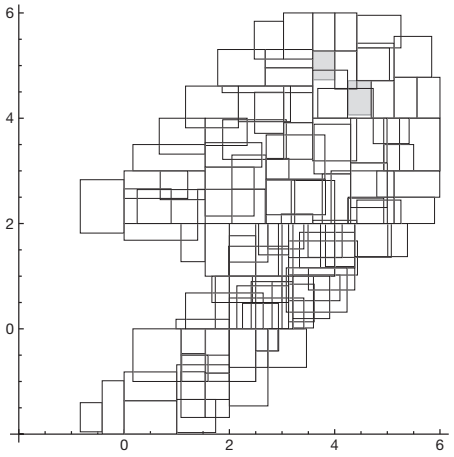


図 6　前処理の例.

レースに分散されるようにする．具体的には，プレース P_i からプレース P_j，ただし $j := i \bmod 2^{\lfloor \log_2 i \rfloor}$，への要求を設定する．探索木が高さ n の完全二分木であれば 2^n プレースに探索空間が行き渡る．

5.3　領域状の解集合計算のための前処理

Under-constrained NCSP は領域状の解集合を持ち，ソルバーは図 5 のような解集合を包む box 集合を計算する．事前に解集合の領域の位置・形状がわかれば，均等に求解処理を分割することが可能になる．そこで提案手法では，事前に指定した深さまで幅優先探索を行い，図 6 のように解集合を低精度で包む box 集合を求め，box を各プレースに分散してから，図 2 の求解処理を始めるようにした．

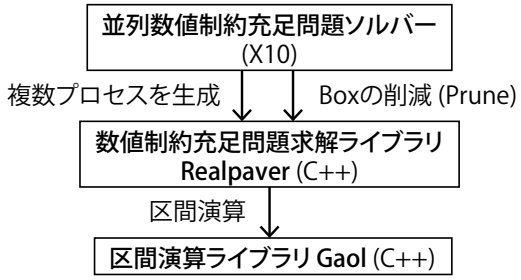


図 7　実装の構成.

前処理では，まずプレース P_0 にて下記の処理を行う．

(1) 幅優先 Branch-and-prune を実行し，初期ドメインから規定サイズ以下の box 集合 B を求める．アルゴリズム中の box 集合 L 内の全 box が規定サイズ以下になるまで実行し，$B := L$ とすればよい．また各繰り返しにおいて B を box のサイズでソートする．

(2) B を 2 分割し，片方をプレース P_1 に送る．

次に，前節の経路図 4 に従い，各プレースで下記の処理を行い，box 集合を分散する．

(1) Box 集合 B を受け取ったら，branch-and-prune の L に設定し，L の要素数が B の 2 倍になるまで幅優先 branch-and-prune を実行する．

(2) B を 2 分割し，片方を他プレースに送る．

上記手続きにより，分散経路の木が 1 段高くなるに従い，2 倍の個数の box が各プレースに行き渡るようにしている．図 6 は規定 box 幅を 2.5 とし，8 プレースで上記手続きを行った結果を示している．

6.　実装

提案手法の実装を X10 (バージョン 2.3.1) と C++ (gcc バージョン 4.3.4) で行った．実装の構成を図 7 に示す．実装では C++で実装された既存の区間演算ライブラリ Gaol (バージョン 4.0.1)[*1] と NCSP 求解ライブラリ Realpaver[6] (バージョン 1.1 を拡張したもの [5])[*2] を利用し，図 2 の処理と branch-and-prune アルゴリズムを X10 で実装した．X10 実装では各プレースに図 2 の処理を走らせる．Branch-and-prune アルゴリズム 4 行目の Prune で C++の Realpaver の API を呼び出すようにしている．Realpaver は逐次処理を前提にしているため，X10 側での上記 API 呼び出しは `atomic` 文にする必要がある．Prune の処理は branch-and-prune アルゴリズムのボトルネックであるため (逐次実行時に実行時間の 95% 以上を占める)，本実装ではプレース内部での速度向上がほとんど期待できないと考えられる．

[*1] http://sourceforge.net/projects/gaol/
[*2] http://pagesperso.lina.univ-nantes.fr/~granvilliers-l/realpaver/

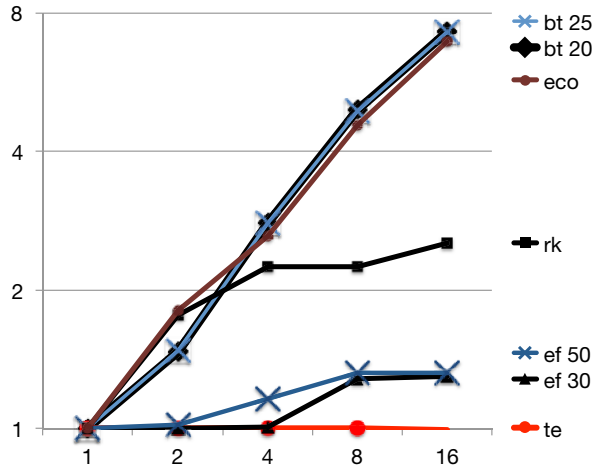


図 8　Well-constrained NCSP 求解の速度向上 (横軸: プレース数, 縦軸: 速度向上).

7. 評価実験

提案手法による NCSP 求解の性能向上を評価するために実験を行った．実験には東京工業大学のスーパーコンピュータ TSUBAME2.0 の 4 ノードを用いた．各ノードは，CPU: Intel Xeon 2.93GHz (6 コア) × 2，RAM: 54GB，内部結合: QDR InfiniBand × 2 (80Gbps) というスペックになっている．X10 は MPI バックエンド (OpenMPI バージョン 1.6.3) を用い，オプション `X10_NTHREADS` を 12 に，`GC_NPROCS` を 12 に設定した．各 NCSP インスタンスについて 1, 2, 4, 8, 16 プレースを用いて求解を行った．各プレース数による求解時の，各ノードにおけるユーザレベル/カーネルレベルの CPU 使用率はそれぞれ，4.5%/0.2%, 4.5%/0.2%, 4.5%/0.2%, 8.5%/0.1%, 17%/0.1%程度であった．おおまかにプレース数分の CPU コアに対応する値となっている．

7.1 Well-constrained NCSP の求解

INRIA COPLIN プロジェクトのベンチマーク集[*3] の 5 問題 7 インスタンスの求解を行った．求解精度 (求解処理を打ち切る最大 box 幅) は 10^{-8} とした．各インスタンスの性質と実験結果を表 1 に示す．各列は，問題名，サイズ (変数・制約の数)，解の個数，1 プレースで求解したときの実行時間 t_1，そのときの探索における分岐数 br_1，4 プレースで求解したときの実行時間 t_4，そのときの各プレースでの分岐数の標準偏差 σ_4，16 プレースで求解したときの実行時間 t_{16}，そのときの各プレースでの分岐数の標準偏差 σ_{16}，を表す．ただし標準偏差は比較のため，総分岐数 br_1 で割っている．求解の速度向上を図 8 に示す．

実験結果では，*bt* 2 インスタンスと *eco* について 16 プレースを用いて 7.0–7.9 倍の速度向上を得た一方，*ef* 2 インスタンスについては速度向上は 1.3 倍にとどまり，*te* では速度向上を得ることができなかった．速度向上の有無は探索木の平衡度と分割の可否による．*ef* と *te* については，表 1 の分岐数の標準偏差 σ_4/br_1 および σ_{16}/br_1 が同列の他インスタンスに較べ大きく，各プレースが探索した探索木のサイズがばらついていることがわかる．実際，両問題の探索木は平衡しておらず，*ef* では木の片側の枝がつねに高々高さ 2 となっており，*te* ではつねに高さ 1 となっている．そのため提案手法では，最低でも木全体の高さ (*te* では br_1 と等しい) 分の Prune および Branch の計算を並列化できない．その他に，インスタンスサイズによらず速度向上が得られることがわかった．

[*3] http://www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/benches.html

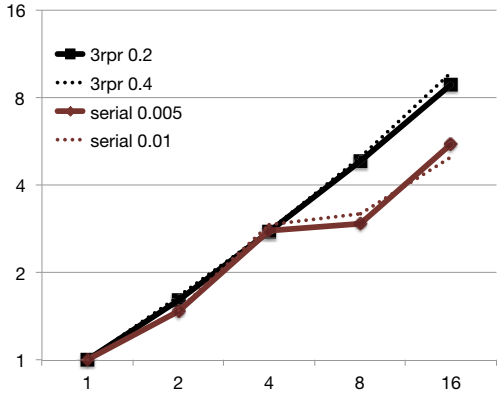


図 10　速度向上の求解精度による差異.

7.2 Under-constrained NCSP の求解

文献 [5][1] で述べられている NCSP 4 問題の求解を行った．求解は 5.3 節の前処理を適用しない場合とする場合の 2 通りで行った．また問題 *serial* と *3rpr* については 2 通りの精度で求解を行った．各問題の性質および実験結果を表 2 に示す．2 列目のサイズを射影変数の数+パラメタ変数の数，3 列目を求解精度とした以外は表 1 と同様の情報を示している．求解の速度向上を図 9，10 に示す．問題 *serial* を精度 0.01 で求解した結果を図 5 に示す．

serial 以外の問題で 16 プレース使用時に 8.9–9.7 倍の速度向上を得た．*serial* では 5.0 倍の速度向上を得た．8 プレース以上を用いたすべての求解処理において，前処理を用いたほうが速度向上が大きくなった．

sp の解集合は初期ドメインへの Branch 適用 2 回の分割面について対象的な形をしており，2 あるいは 4 プレースを使用時はほぼ等しく探索空間が分散される．しかし 3 回目の Branch 適用以降は非対称に探索空間が分割されていくため，前記以外のプレース数では各プレースの探索木の大きさにばらつきが出てしまう．8 あるいは 16 プレース使用時，2 プレースにおいて全探索空間の約 1/4 の求解を行う結果となり，実行時間がほぼ変わらなかった．前処理を用いると，2 あるいは 4 プレース使用時では用いない場合

表 1　Well-constrained NCSP 求解の実験結果.

問題	サイズ	解の個数	t_1	t_4	t_{16}	br_1	σ_4/br_1	σ_{16}/br_1
Broyden tridiagonal (*bt*)	20	2	11.5	4.12	1.58	23345	0.10	0.04
	25	2	349	131	44.1	561559	0.11	0.04
Economics (*eco*)	8	8	49.3	18.8	7.1	54766	0.13	0.04
Extended Frudenstein (*ef*)	30	1	4.5	4.47	3.46	90	0.28	0.16
	50	1	33.4	28.8	25.2	150	0.28	0.16
Robot kinematics (*rk*)	9	16	27.4	12.2	10.8	28669	0.18	0.12
Trigexp 1 (*te*)	100	1	32.5	32.5	32.7	98	0.43	0.24

表 2　Under-constrained NCSP 求解の実験結果.

問題	サイズ	精度	解の個数	前処理	t_1	t_4	t_{16}	br_1	σ_4/br_1	σ_{16}/br_1
Sphere and plane (*sp*)	2+2	0.01	52217	off	21.7	5.54	5.45	62589	0.00	0.08
				on		6.08	2.23		0.03	0.02
Sailboat (*sail*)	2+2	0.1	27473	off	29.6	17.5	8.09	36361	0.22	0.08
				on		8.12	3.33		0.02	0.02
Serial robot (*serial*)	2+2	0.01	71716	off	33.9	16.6	11.5	93465	0.19	0.10
				on		11.53	6.77		0.12	0.07
		0.005	186632	on	88.7	31.7	16.1	245465	0.13	0.07
3-RPR robot (*3rpr*)	3+3	0.4	258259	off	135	41.1	17.4	258811	0.04	0.03
				on		48.5	13.9		0.80	0.02
		0.2	814139	on	381	138	42.9	816313	0.09	0.02

よりも若干並列化の効率が悪くなっているものの，それ以外のプレース数でも相応の速度向上を得ることができた．

sail と *serial* は図 5 にも見られるように解集合が複雑な形状をしており，また位置によっては内部でも細かい精度の box 群を計算する必要がある．そのため，前処理を用いない求解処理において，各プレースの探索木の大きさにばらつきが出たものと考えられる．*sail* については前処理を適用することで速度向上を大きく改善することができた．*serial* については前処理を適用し速度向上を改善できたものの，16 プレース使用時で 5.0 倍にとどまった．

求解精度を 1/2 倍した場合の速度向上を比較すると (図 10)，どちらの設定でも同様の結果となった．

8. まとめと今後の課題

本研究では，well-constrained および under-constrained NCSP を求解するための branch-and-prune アルゴリズムの並列化手法を提案し，既存実装 Realpaver を PGAS 言語 X10 により拡張することで提案手法を実装した．実験結果では，well-constrained NCSP について 16 プレース使用時に最大 7.9 倍の速度向上 (並列化効率 50% 程度)，under-constrained NCSP について 16 プレース使用時に最大 9.7 倍の速度向上 (並列化効率 60% 程度) を得た．

提案手法は探索木を分散することで並列化を図ったものであり，計算量の大きい Prune は並列化していない．今後の課題として，Prune も並列化し，複数プレース間・単一プレース内部の双方で速度向上を得ることが挙げられる．また，本研究では branch-and-prune の求解パラメタを固定して開発を行ったが，異なる求解パラメタを持つ並列タスク群による求解も効果的だと考えている．文献 [1] では NCSP に基づき並列ロボットの解析を行っているが，提案手法を用いることで未解決問題を扱うことが可能になると見込まれる．

参考文献

[1] Caro, S., Chablat, D., Goldsztejn, A., Ishii, D. and Jermann, C.: A branch and prune algorithm for the computation of generalized aspects of parallel robots, *Proc. of International Conference on Principles and Practice of Constraint Programming (CP)*, *LNCS*7514, pp. 867–882 (2012).

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

[3] Chu, G., Schulte, C. and Stuckey, P.: Confidence-based work stealing in parallel constraint programming, *Proc. of CP*, *LNCS*5732, pp. 226–241 (2009).

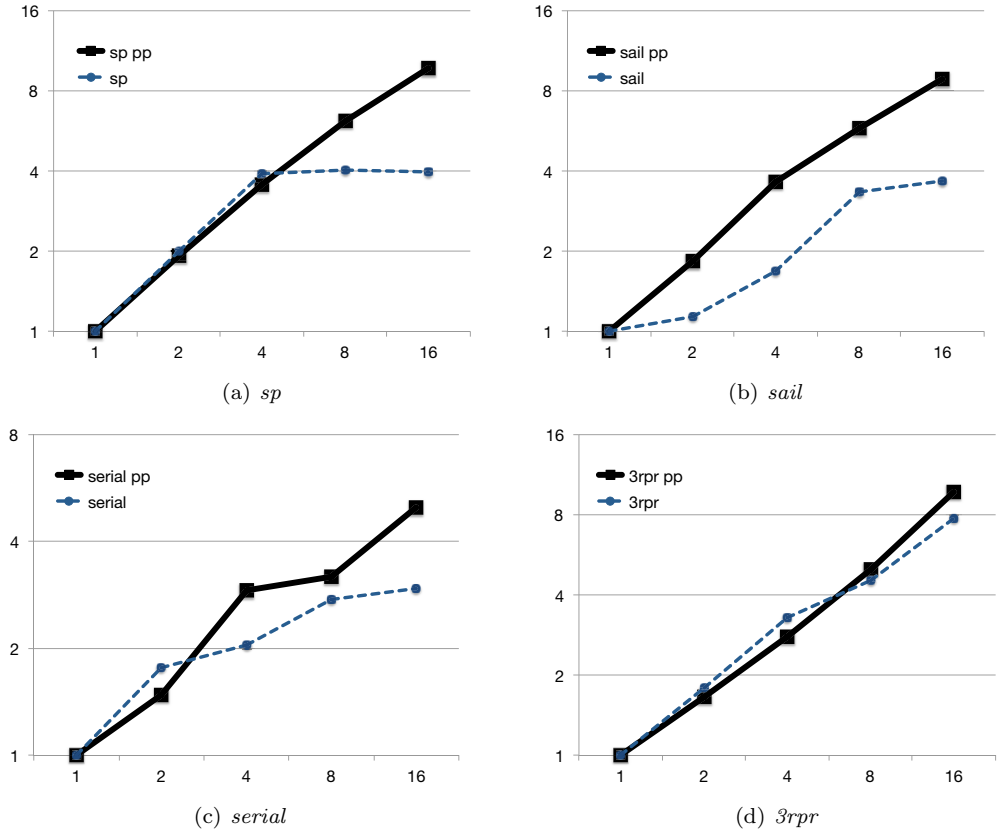
[4] Schubert, T., Lewis, M. and Becker, B.: PaMiraXT: Parallel SAT Solving with Threads and Message Passing., *JSAT*, Vol. 6, pp. 203–222 (2009).

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

[6] Granvilliers, L. and Benhamou, F.: Algorithm 852: RealPaver: an interval solver using constraint satisfaction techniques, *ACM TOM*, Vol. 32, No. 1, pp. 138–156 (2006).

[7] 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, *Proc. of Workshop on Parallel Methods for Constraint Solving*, 13 pages (2011).

[8] Granvilliers, L. and Hains, G.: A conservative scheme



(a) *sp* (b) *sail*

(c) *serial* (d) *3rpr*

図 9 Under-constrained NCSP 求解の速度向上 (横軸: プレース数，縦軸: 速度向上).

for parallel interval narrowing, *Information Processing Letters*, Vol. 74, No. 3-4, pp. 141–146 (2000).

[9] Goualard, F. and Goldsztejn, A.: A Data-Parallel Algorithm to Reliably Solve Systems of Nonlinear Equations, *Proc. of International Conference on Parallel and Distributed Computing, Applications and Technologies*, pp. 39–46 (2008).

[10] Goldsztejn, A. and Goualard, F.: Box consistency through adaptive shaving, *Proc. of SAC*, pp. 2049–2054 (2010).

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

[12] Gendron, B. and Crainic, T.: Parallel branch-and-bound algorithms: Survey and synthesis, *Operations Research*, Vol. 42, No. 6, pp. 1042–1066 (1994).

[13] Otten, L. and Dechter, R.: Towards Parallel Search for Optimization in Graphical Models, *Proc. of ISAIM* (2010).

[14] Jurkowiak, B., Li, C. M. and Utard, G.: A Parallelization Scheme Based on Work Stealing for a Class of SAT Solvers, *Journal of Automated Reasoning*, Vol. 34, No. 1, pp. 73–101 (2005).

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

[16] Rossi, F., van Beek, P., and Walsh, T.: Handbook of Constraint Programming (Foundations of Artificial Intelligence), Elsevier (2006)

[17] 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, pp. 797–827 (1997).

[18] Bordeaux, L.: The complexity of filtering algorithms for numerical constraints, Technical report (1999).

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

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