ICOT Technical Memorandum: TM-0825

TM-0825

線形不等式を解く制約ソルバーの並列計算について

川岸太郎, 坂根清和,
生駒憲治

November, 1989

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191 ~5
Telex ICOT J32964

Institute for New Generation Computer Technology

線形不等式を解く制約ソルバーの並列計算について

Parallel Computation of A Constraint Solver for Linear Inequalities

川岸太郎 Taro Kawagishi
坂根清和 Kiyokazu Sakane
生駒憲治 Kenji Ikoma

新世代コンピュータ技術開発機構
Institute for New Generation Computer Technology

等式あるいは不等式によって制約条件が与えられた場合、この制約を満たす実数変数の具体的な値を見つける必要のあることが度々ある。ここではこれらの制約のうち線形なものを扱って制約中に現われる実数変数それぞれの取り得る上限値・下限値を導き出すSUP-INFアルゴリズムを取り上げ、その並列実行による高速化について述べる。

The constraints given by equalities and inequalities often appear in actual problems. We take up SUP-INF method which finds the upper bound values and lower bound values of variables for a given set of linear inequalities. To make a parallel implementation of SUP-INF method, which uses recursive symbolic computation, we consider the way tasks should be spread among multi-processors.

1．はじめに

機械設計におけるパラメトリック設計問題や空間配置問題など様々な問題において等式、不等式によって制約条件が与えられた場合、それに現われるいくつかの実数変数の値を求める必要がある。変数に対する制約条件を扱うには受動的な方法と能動的な方法とが考えられる。受動的な方法とは一定のジェネレータで解の候補値を逐次生成しこれが与えられた制約条件を満たすか否かをテストしていくものである（generate andtest）。これに対して能動的な方法は与えられた制約条件から変数の取り得る上限値と下限値を導いて、変数の解の範囲を限定する方法である。能動的な方法により候補値の範囲を限定してからgenerate and testを行えば、探索の範囲が狭められるので能率的である。この意味で等式・不等式を満たす変数の範囲を限定する制約ソルバーは解の探索に有効である。

ここでは線形不等式で表される制約を扱い制約中のそれぞれの変数が取得る区間を導き出すSUP-INFアルゴリズムを取り上げて、これを並列実行によって高速化することについて述べる。ここで並列計算とはメッセージ交換型のマルティプルプロセッサーによる並列処理を意味する（[Chikayama]）。

このSUP-INFアルゴリズムは不等式に現れる一次式を変形しながら再帰的に計算を行うものであるが、このアルゴリズムのなかで並列にプロセスを割り当てられる部分はどこであり、またメッセージとしてプロセッサー間で通信し合うのが適当であるのはどの部分であるかを調べる。

2．SUP-INF法について

(a) SUP-INF法の扱う問題

線形制約条件

$$\begin{matrix} a11 \cdot X1 + \cdots + a1n \cdot Xn \leq b1 \\ (CT) \qquad\qquad \sim \\ am1 \cdot X1 + \cdots + amn \cdot Xn \leq bm \end{matrix}$$

が与えられたとする。ここでa11,‥,amnは実数値、X1,‥,Xnは実数変数とする。

SUP-INF法は上の線形制約条件の下で各変数の取りうる上限値と下限値を求める方法である。つまり線形不等式による制約が定める凸多面体に対して、それを入れる最小の区間を求めているわけである。ここで等式は不等号の向きを逆にした二つの不等式で書き表せるから、制約条件は全て不等式で表されるものとする。

(b) 不等式制約の可解性の判定

制約条件(CT)に現われる各変数Xについて上限値と下限値をそれぞれsup(X;CT), inf(X;CT) と表すことにする。これをSUP-INF法のアルゴリズムに従って計算してみていづれかの変数について

$$inf(X;CT) > sup(X;CT)$$

となれば制約(CT)は解を持たないことが分かる。

また全ての変数Xについて

$$inf(X;CT) \leq sup(X;CT)$$

であることが分かれば制約(CT)は矛盾しないことが判り各変数の取り得る範囲が得られる。

(c) SUP-INF法のアルゴリズムについて

SUP-INF法は、各変数に関する条件付き上限式、下限式を再帰的に計算して、その変数の取りうる上限値と下限値を求める。以下に示すこのアルゴリズムは[Shostak]によるものである。

［定義2.1］

F1,F2,…,Fm が線形形式である時に、

最小値関数 min(F1,F2,…,Fm) を mini-linear form といい、

最大値関数 max(F1,F2,…,Fm) を maxi-linear form という。

［定義2.2］

制約CTに現われる変数についての上限式、下限式を次のように定める。制約CTを変数Xについて整理して、

CT :: { X≦U1, X≦U2, ..., X≦Um,
X≧L1, X≧L2, ..., X≧Ln }, (Ui, Li は線形形式) とする。

この時,

upper(X;CT) = min(U1,U2,....,Um),		m>1 のとき (mini-linear form)
= U1,		m =1 のとき
= infinite,		上限式がないとき
lower(X;CT) = max(L1,L2,....,Ln),		n>1 のとき (maxi-linear form)
= L1,		n =1 のとき
= minus-infinite,		下限式がないとき

とする。

［定義2.3］

次の補助関数を定義する。

(1) 変数Xとmini-linear form F = min(F1,…,Fm)に対して、

supp(X,F) = min(supp(X,F1),…,supp(X,Fm))

supp(X,Fj) = supremum{X: X ≦ Fj, X以外の変数は定数とする}

(2) 変数Xとmaxi-linear form F = max(F1,…,Fn)に対して、

inff(X,F) = max(inff(X,F1),…,inff(X,Fn))

inff(X,Fj) = infimum{X: X ≧ Fj, X以外の変数は定数とする}

と定義する。
suppの計算式を示しておく：

1. supp(X,F) = F　　Fが数値のとき
2. supp(X,F) = infinite　F=X のとき
3. F = a*X+G と書けるとき
   - 3.1 a>1 のとき　supp(X,F) = infinite
   - 3.2 a<1 のとき　supp(X,F) = G/(1-a)
   - 3.3 a=1 のとき
     - 3.3.1 Gが数値でないとき　supp(X,F) = infinite
     - 3.3.2 G≧0 のとき　supp(X,F) = infinite
     - 3.3.1 G<0 のとき　supp(X,F) = minus_infinite
4. F = min(F1,…,F2) と書けるとき
   supp(X,F) = min(supp(X,F1),…,supp(X,F2))

inff についても同様である。

[定理2.1] ([Shostak],[Bundy])

CT　：与えられた線形不等式制約
F　：Mini-linear form あるいは Maxi-linear form（リスト形式で示す）
Vs　：変数の集合（リスト形式で示す）

に対して、sup(F,Vs;CT), inf(F,Vs;CT)なる手続きを以下のアルゴリズムに従って再帰的に定義すると、sup([X],[ ];CT)によって変数Xの上限値が、inf([X],[ ];CT)によって変数Xの下限値が求められる。

さらに詳しくいうとsup(X,Vs;CT)は制約条件CTの下でVsに現われる変数を定数として固定した時の変数Xの上限値を与え、inf(X,Vs;CT)は制約条件CTの下でVsに現われる変数を定数として固定した時の変数Xの下限値を与える。

アルゴリズム：

s1. sup([c],Vs;CT) = c　　c が定数の時
s2. sup([X],Vs;CT) = X　　変数XがVsに属している時
s3. sup([X],Vs;CT) = supp(X,Z)　　変数XがVsに属していない時
　ここで Z = sup(upper(X;CT),Vs+{X};CT).
s4. sup(a*F,Vs;CT) = a*sup(F,Vs;CT)　a > 0 のとき
　= a*inf(F,Vs;CT)　a < 0 のとき
s5. sup(a*X+F,Vs;CT) = sup(a*X+sup(F,Vs+{X};CT),Vs;CT)
　ここでXがsup(F,Vs+{X};CT)に現われないときさらに
　sup(a*X,Vs;CT) + sup(F,Vs+{X};CT)
　と書くことができる
s6. sup(min(F1,…,Fm),Vs;CT) = min(sup(F1,Vs;CT),…,sup(Fm,Vs;CT))

またinf(F,Vs;CT)についても同様である。

i1. inf([c],Vs;CT) = c　　c が定数の時
i2. inf([X],Vs;CT) = X　　変数XがVsに属している時
i3. inf([X],Vs;CT) = inff(X,Z)　　変数XがVsに属していない時
　ここで Z = inf(lower(X;CT),Vs+{X};CT).
i4. inf(a*F,Vs;CT) = a*inf(F,Vs;CT)　a > 0 のとき
　= a*sup(F,Vs;CT)　a < 0 のとき
i5. inf(a*X+F,Vs;CT) = inf(a*X+inf(F,Vs+{X};CT),Vs;CT)
　ここでXがinf(F,Vs+{X};CT)に現われないときさらに
　inf(a*X,Vs;CT) + inf(F,Vs+{X};CT)

と書くことができる

i6. inf(max(F1,…,Fm),Vs;CT) = max(inf(F1,Vs;CT),…,inf(Fm,Vs;CT))

(d) アルゴリズムの計算の進行

不等式制約(CT)が与えられたときSUP-INF法はsup(X;CT)とinf(X;CT)を計算するために上の手続きsupとinfを互いに再帰的に呼んでいく。手続きsupを計算するのに必要なのは各変数の上限を表現するmini-linear form（各変数について制約(CT)を整理したときのそれ以外の変数で書かれた上限式の集合）であり、手続きinfの場合は各変数の下限を表すmaxi-linear formである。

各変数の上限値・下限値を求める上のアルゴリズムは大まかには次のように行われる：

1. 各変数の上限式集合を求めこれをFとしてsup（または下限式集合のinf）を呼ぶ（ステップs3,i3）。
2. Fが一次式の集合のときにはこれを一つ一つの一次式に分けてそれぞれを独立に計算する（ステップs6,i6）。
3. 一次式一つからなるFについてはこのなかの単項を一つづつ取り出して変数集合Vsのなかに入れ、Fから変数一つ分少なくなった一次式に対する結果を求めそれを再びその単項とマージする（ステップs5,i5）。
4. そしてFが係数1の単項式になるまで手続きを再帰的に呼び、この後ステップs2,s3,i2,i3を行う。

SUP-INFアルゴリズムではこの様にして sup([Form],[U,V,W,‥];CT)（Formはminあるいはmaxによる一次式の結合とする）の計算は一つの変数Xに対するsup([X],[U,V,W,‥];CT) の形式のものに還元される。

このsup([X],[U,V,W,‥];CT)の形式のサブゴールはsup(U;CT)あるいはinf(U;CT)を計算する途中で現れることもあるし、またsup(V;CT)あるいはinf(V;CT)を計算する途中でも現れ得る。つまり異なる変数に対する計算について同じサブゴールsup([X],[U,V,W,‥];CT)が複数回呼ばれるので、そのまま計算するならば何度も重複して計算しなければならない。

(e) SUP-INFアルゴリズムの特徴

SUP-INFアルゴリズムの特徴を次のようになる。

(e-1) 手続きsup,infは再帰的に自分と他とを呼んで上限値、下限値の計算をする。

(e-2) 手続きsup,infが持っているべきデータは各変数の上限式と下限式である。

(e-3) 次の様に各変数ごとに、またsupとinfごとに独立して計算できる。

sup(X1;CT), inf(X1;CT)
sup(X2;CT), inf(X2;CT)
～
sup(Xn;CT), inf(Xn;CT)

(e-4) 上の手続きsupとinfの計算の途中で異なるサブゴールに対する中間結果であって共有されるものが多くある。

この(e-4)について次に説明をする。

2.(d)で述べたように手続きsupとinfともに再帰呼び出しのあるレベルでは同一のサブゴールを持つことがある。従って逐次計算の場合を考えるならば、サブゴールsup([X],[U,V,W,‥];CT), inf([X],[U,V,W,‥];CT) の結果が新しく得られる度にここに現われる([X],[U,V,W,‥])をノードのキーとしてそれぞれをワーキングメモリーに格納しておけば次に同じものが呼ばれたときにこれらを再利用して、効率的に計算を進められることが判る。SUP-INF法の場合このノードの呼出し関係は一次式中の変数の並びかた、アルゴリズム内での式の展開の仕方に依存するため計算の進行に伴いダイナミックに決まっていく。

この様にサブゴールの中間結果を再利用することはこの問題に限らずFibonacci数列などの再帰計算式についても同様に適用できる。Fibonacci数列の様に再帰計算のサブゴールの呼び出し関係のグラフが一意的に決まる場合については、特定の初期値について数値アルゴリズムのサブゴールのグラフを作って効率の良い計算をすることが行われている（[Clocksin]）。しかし一般の問題について初期値を特定しない場合のサブゴールグラフをコンパイルすることはできていない。

またSUP-INF法のようにサブゴールのグラフがデータの表現方法に依存していて一意に決まらない場合はダイナミックに中間結果を格納していくしかないといえる。

3．SUP-INF法の並列計算

次にメッセージ交換型マルティプルプロセッサー上でSUP-INF法をどのように並列計算してこの実行を高速化することする事ができるかを考える。並列環境はMulti-PSI上の並列OSであるPIMOSを考える（[Chikayama]）。Multi-PSIは16台または64台のプロセッサが2次元メッシュ状に接続されていて、フロントエンドプロセッサからプログラムを起動する構造になっている疎結合マシンである。個々のプロセッサ間のデータのやり取りにはパケット通信を使う。

メッセージ交換型の並列計算機で問題になるのはプロセッサからプロセッサへデータを転送するのには大きな時間がかかることである。現在のMultiPSIでは他のプロセッサのメモリにアクセスするには同じプロセッサ内のメモリにアクセスするのに比べて2桁大きい時間がかかる。従ってデータの転送は頻繁には行なわず転送するデータも小さくする必要がある。

SUP-INF法のアルゴリズムの特徴(e-1)～(e-4)を見てみるとアルゴリズムのなかでプロセスを並列化できる可能性のあるのは次のような部分である。

(1) 不等式に現れる変数ごとにプロセスを振り分ける。
(2) 各変数の上限値、下限値の計算それぞれにプロセスを振り分ける。
(3) 各サブゴールに現れるmini-linear formあるいはmaxi-linear formの独立な一次式ごとにプロセスを振り分ける。

(1)については変数ごとにプロセスは独立でありタスクの大きさも同程度であると考えられるので異なるプロセッサーに並列に振り分けられる。(2)についても全く同じである。

(3)についてはサブゴールのタスクの大きさが問題になってくる。つまり再帰呼び出しの深さが増すとアルゴリズム上は並列なサブゴールでもタスクの大きさが個々に違ってくる。このため複数のサブプロセスの結果をマージする時点で全てのサブプロセスが終了するまで先にサブプロセスを終了したプロセッサーが待っていなければならないということが起きる。またサブゴールのタスクの大きさが予めは分からないので例え並列に割り振ったとしても実際に起動されるサブプロセスが小さすぎて、一つのプロセッサーに割り当てても割り当てと結果を親プロセスに渡すことにそれ以上の時間がかかってしまうことがある。

従って(3)のサブゴールを並列プロセスに分けるのには、タスクの大きさを計る何らかの方法を持っていてタスクの大きさが或る程度以上ものを均質に分散させる必要がある。

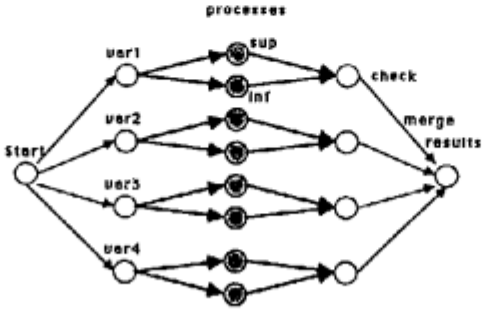
また2.(e)の部分結果の再利用を並列計算でも行なうには次のようにする：

(1) 各プロセッサー内でワーキングメモリを持ち部分結果を格納する。
(2) 新たに計算された部分結果はその計算コストを計っておいて、このコストが異なるプロセッサーにこの部分結果を通信するコストより十分大きいと考えられる場合にのみメッセージで転送する。

つまり特定のノードの中間結果が必要になるごとにワーキングメモリへ参照しに行き、既にあるものなら利用する。またメモリに登録されていないものは他のプロセッサから通信を受けていないかどうかを確認する。その結果まだ通信を受けていないことがわかった場合は新たに計算してメモリに格納する。そして計算量が一定以上のときにのみこれを他のプロセッサーに転送する。ここで計算量が通信コスト以上のものに限って結果を転送するのは、小さな計算をメモリに登録したり見に行って通

信するよりは同じサブゴールの計算であっても再度計算した方が時間がかからないからである。

以上のことからSUP-INFアルゴリズムは次のよう並列計算するのが適当であることが判る：
(1) 不等式に現れる変数の上限値・下限値の計算を変数それぞれに対しプロセスを振り分けてプロセッサを割り当ててる。
(2) 上の２．で述べたノードの部分結果を上のプロセスが持っているワーキングメモリに格納し、他のプロセスにメッセージ通信をして転送する。



こうするとsup, infとも変数それぞれに対して独立に計算が行えるので2×N個のプロセスができ（Nは変数の数）、この部分が並列に実行される。また部分計算結果をメッセージとしてプロセッサー間で通信し合う事により重複した計算を除くことができ、これによる速度の向上も見込まれる。
以下に並列言語KL1（[瀬]）のプログラム風に書いた処理を示す：

```
sup_inf([Var|Vars],CT,Result,Stream) :- true |
        sup_inf1(Var,CT,Result1,Stream)@processor(PE),
        sup_inf(Vars,CT,Result2,Stream).

sup_inf1(Var,CT,Result,Stream) :- true |
        sup(Var,CT,Sup,Stream),
        inf(Var,CT,Inf,Stream)@processor(PE1).

sup(Var,Sup,CT,Stream) :- true |
        comm_manager(Stream,Channel)@priority(4000),
        sup1([Var],[ ],Sup,CT,WkMemory,Channel)@priority(3000).
```

実質的な計算を行なっているのはsup1であり、このなかで或るノードの値が必要になったときローカルメモリを調べてもし見つからなかったならChannelを通じてget要求を出す。新しい部分結果が計算されたときはChannelを通じてsend要求を送る。またcomm_managerはChannelを通じて要求があったときのみStreamを通じて他のプロセッサからの転送データを受けたり送ったりする。

4．まとめ

上で示した並列計算の方式では2×N個の並列なプロセスは一番上位で分岐しているプロセスであるのでプロセッサへの分散によるオーバーヘッドは一番少なくて、Nが大きいときにはその数だけの速度向上が得られると考えられる。

不等式に現れる変数の数をNとしアルゴリズムのサブゴールの深さdまで並列なサブプロセスを割り

振るとするとN^d個のサブプロセスが作られるので、プロセッサの数に制限がなく通信にかかる時間が無視できると仮定すると、速度向上はN^dのオーダーとなる。

どの深さのサブゴールまで独立なプロセッサに割り当てて意味のあるプロセスとなるかは不等式の大きさ（アプリケーションのタスクの大きさ）と並列計算機の通信時間のオーバーヘッドに依存する問題であり実際にプログラムを実行してみる必要がある。この点について実験的に評価してみることが今後の課題である。

【参考文献】

[Bledsoe] W.W Bledsoe, A new method for proving certain Presburger formulas.4th IJCAI, 1977.

[Bundy] A. Bundy, The Computer Modelling of Mathematical Reasoning, Chapter8. Academic Press,1983.

[Butler,Lusk,McCune,Overbeek] R.Butler, E.Lusk, W.McCune, R.Overbeek, Parallel Logic Programming for Numeric Applications, Third International Conference on Logic Programming, Lecture Notes in Comp.Sci.225,Springer,1986,pp375-388.

[Chikayama] T.Chikayama,H.Sato,T,Miyazaki, Overview of the Parallel Inference Machine Operating System(PIMOS), Proceedings of FGCS'88,Vol1,pp230-251,1988

[Clocksin] W.A.Clocksin, A Technique for Translating Clausal Specifications of Numerical Methods into Efficient Program, J.Logic Programming 1988:5,231-242.

[淵] 淵一博監修, 並列論理型言語GHCとその応用 共立出版

[大木 他] 大木,澤本,坂根,藤井 Sup-Inf法に基づいた制約論理型プログラミング言語 日本ソフトウェア 科学会第５会大会,1988

[川村 他] 川村、大和田、溝口 CS-Prolog:拡張単一化に基づくConstraint Solver Proc. of LPC'87,1987, p21.

[Sacks] Elisha Sacks, Hierarchical Reasoning about Inequalities. Proceedings of AAAI-87,649-654.

[Shostak] R.E.Shostak, On the SUP-INF Method for Proving Presburger Formulas, Journal of the Association for Computing Machinery,Vol24,1977,pp529-543.