ICOT Technical Memorandum: TM-0882

TM-0882

並列SUP-INF法による線形不等式制約ソルバー・ユーザーズマニュアル

川岸　太郎

May, 1990

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

並列 SUP-INF 法による線形不等式制約ソルバー・ユーザーズマニュアル

本 Technical Memo は並列言語 KL1 で書かれた SUP-INF アルゴリズムによる線形不等式制約ソルバーのプログラムマニュアルである。このプログラムを開発した研究の概要、プログラムの仕様、そしてアルゴリズムの詳細について以下に述べる。

(1) 本研究の概要について

本研究では、線形の等式・不等式で表現される制約式を解析する制約ソルバー SUP-INF 法を並列環境 Multi-PSI で高速に実行させることを目的として、並列処理言語 KL1 上にインプリメントした。

SUP-INF アルゴリズムは、線形な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にこれらの制約中に現われる実数変数それぞれの取り得る上限値・下限値を計算するものである。ここでは並列実行して高速化することを目的としてこれを並列言語 KL1 上に実現した ([川岸])。

一般に線形等式・不等式のサイズを N としたとき SUP-INF 法の計算複雑度は 2^N である。SUP-INF 法を逐次実行するならば、不等式のサイズが大きくなるに従い問題の大きさが増し計算量が増大する。今回の並列 SUP-INF 法では、独立な $2N$ 個のプロセスを生成するのでプロセッサの数がこれ以上有るとしたとき少なくとも $2N$ 倍の速度向上が見込まれ、不等式のサイズが大きくなる程並列実行による効果は大きくなる。また異なるプロセス間で重複している中間結果をメッセージ通信により共有するのでこの分だけさらに高速化される。

(2) "並列 SUP-INF 法による線形不等式制約ソルバー"の仕様について

プログラム SUP-INF は次の 6 つのモジュールからなっている。これらの内容の概略について次にしめす:

ファイル	モジュール	実行内容
supinf.kl1	supinf	主モジュール、SUP-INF アルゴリズムの実行部分
put_in_dic.kl1	put_in_dic	与えられた線形不等式を内部表現に変換する述語、 この内部表現を内部の辞書に格納する述語
assign_value.kl1	assign_value	計算結果の変数の数値を辞書に格納する述語
formula_proc.kl1	formula_proc	内部表現による一次式について演算を行う様々な述語
s_util.kl1	s_util	リスト処理などの雑用述語
supgo.kl1	supgo	SUP-INF アルゴリズムを呼び出す述語

これらのなかでモジュール supgo のなかの述語 supgo がトップレベルである。述語 supgo の引数の意味を以下に示す。

```
supgo(ExampleNum,PEs,FileN,Sw)
```

ExampleNum	(in) :	例の番号
PEs	(in) :	使用するプロセッサの数
FileN	(in) :	結果を書き出すファイル名
Sw	(in) :	中間結果をメモリに格納して計算するかどうかのスウィッチ ('off' または 'on')

実際に supgo を呼ぶには次のように述語 example に不等式を与える必要がある。線形の不等式の与え方はこの例で分かるように、変数をアトムで書き、係数は実数で、単項式は + または - でつなぐ。不等号は <,>, =<,>= が使える。

```
supgo(ExampleNum,PEs,FileN,Sw) :- true |
        example(ExampleNum,Exps,Vars),
        supinf:supinf(Exps,Vars,0,PEs,Sw,Res,Dic),
```

```
        fel:open_file(Stream,FileN,w,normal),
        Stream=[putb(string#"Result: "),puttq(Res),nl,
                putb(string#"Dic: "),puttq(Dic),nl,nl].

example(21,Exps,Vars) :- true |
        Exps=[y=<x+5.0, x+y=<5.0, y+2.0*x=<10.0, y>=2.0*x-14.0, x+y+5.0>=0.0 ],
        Vars=[x,y,z].
```

supgo はそのなかで述語 supinf を呼ぶ。supinf は結果を引き数 Dic に返すが、Dic には既に格納されている上限式と下限式に加えて、各変数の上限値と下限値が返される。出力される Dic のフォーマットについては下の supinf の説明を参照のこと。上の例では supgo(21,8,result_file,off) を呼べば、変数 x と y の上限値と下限値は result_file に書かれる。

ただしここではファイルへの出力のため KL1 プログラムライブラリ FLIB (ICOT Technical Report 529: FLIB User Manual を参照のこと) の述語 fel:open_file を使っているが必ずしも必要とは限らない。

次に述語 supinf について説明する。これの呼びだしは

```
supinf(Exps,Vars,Start_PE,PEs,RegSw,Dic1,^Result,^Dic2)
```

であって各引き数の意味は次のようである。

Exps	(in) :	一次式のリスト (変数はアトムで書かれている)
Vars	(in) :	SUP と INF を求めるべき変数を表すアトムのリスト
Start_PE	(in) :	ジョブを開始するプロセッサ番号
PEs	(in) :	使用するプロセッサの数
RegSw	(in) :	中間結果をメモリに格納して計算するかどうかの スウィッチ ('off' または 'on')
Dic1	(in) :	変数アトムの名前に従って tree 構造に並べた上限式と 下限式のリスト Format:: t(Left,Node,Right) Node = [VarX,ValueX,UpperX,LowerX, UpperLinearFormsX,LowerLinearFormsX] VarX は変数を表すアトム ValueX は変数 VarX の値 (確定した場合) UpperX は変数 VarX の上限値 LowerX は変数 VarX の下限値 UpperLinearFormsX は変数 VarX の上限を表す一次式のリスト LowerLinearFormsX は変数 VarX の下限を表す一次式のリスト 最初は nil/0 として呼び出すこと
Result	(out) :	'success' または 'fail'
Dic2	(out) :	Dic1 に新しい上限値と下限値を加えた辞書

述語 supinf は次のように先ず最初にモジュール put_in_dic の normalize で入力の一次式を内部標準形に変換して、put_ineq_in_dic で辞書に格納する。並列なプロセッサへの割り振りは各変数の SUP と INF の計算に対して行っている。

```
supinf(Exps,Vars,Start_PE,PEs,RegSw,Result,Dic) :-
    wait(Exps), wait(Vars) |
        supinf(Exps,Vars,Start_PE,PEs,RegSw,nil/0,Result,Dic).

supinf(Exps,Vars,Start_PE,PEs,RegSw,Dic1,Result,Dic2) :- true |
        put_in_dic:normalize(Exps,NormExps,Dic1,Dic11),
```

```
        supinf1(NormExps,Vars,Start_PE,PEs,RegSw,Dic11,Result,Dic2).

supinf1(NormExps,Vars,Start_PE,PEs,RegSw,Dic11,Result,Dic2) :-
    wait(Dic11) |
        put_in_dic:put_ineq_in_dic(NormExps,Dic11,Dic12),
        sup_inf(Dic12,Vars,Start_PE,PEs,RegSw,Result,Dic2).

sup_inf(Dic,Vars,Start_PE,PEs,RegSw,Result,DicNew) :-
    wait(Dic) |
            %%% get a stream for DataStream
        s_util:sync_copy(Vars,VarsC,VarsCC),
        sup_inf2(Dic,VarsC,Start_PE,PEs,RegSw,SupInfList,DataStream,Sync),
        ( wait(Sync) ->
          revise_dic(Dic,VarsCC,SupInfList,Result,DicNew) ).

sup_inf2(Dic,[],_PEbase,_PEs,_RegSw,SupInfList,DataStream,Sync) :-
    true |
        SupInfList=[], Sync=1.
sup_inf2(Dic,[Var|Vars],PEbase,PEs,RegSw,SupInfList,DataStream,Sync) :-
    atom(Var),integer(PEbase),integer(PEs),atom(RegSw) |
        next_PE(PEbase,PEs,1,PE1),
        next_PE(PEbase,PEs,2,PEbase2),
        s_util:sync_copy(Dic,DicC,DicCC),
        sup_inf3(DicC,Var,PE1,PEs,RegSw,Sup,Inf,Res_c,
                 DataStream)@processor(PE1),
        sup_inf2(DicCC,Vars,PEbase2,PEs,RegSw,SupInfList1,
                 DataStream,Sync1),
        SupInfList=[(Sup,Inf,Res_c)|SupInfList1],
        ( wait(Sync1),wait(Res_c) -> Sync=1 ).

sup_inf3(Dic,Var,PE,PEs,RegSw,Sup,Inf,Res_c,DataStream) :-
    wait(Dic),atom(Var),atom(RegSw) |
        s_util:sync_copy(Dic,DicC,DicCC),
        sup(Var,SupL,DicC,RegSw,DataStream),
        next_PE(PE,PEs,1,NextPE),
        inf(Var,InfL,DicCC,RegSw,DataStream)@processor(NextPE),
        check_sup_inf(SupL,InfL,Sup,Inf,Res_c).
```

述語 sup と inf が実際の計算を行う部分であるが、これについてはプログラムは省略してアルゴリズムの詳細を次の (3) で述べる。

(3) SUP-INF 法による線形不等式制約ソルバーの並列化 (詳細)

(3-1) 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 and test) 。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[Bledsoe]と[Shostak]によって与えられたアルゴリズムであり、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([川村 他],[大木 他])。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。

(3-2) SUP-INF法について

(a) SUP-INF法の扱う問題

線形不等式による制約条件(CT):

$$\begin{aligned} a_{11} * X_1 + \cdots + a_{1n} * X_n &\leq b_1 \\ a_{21} * X_1 + \cdots + a_{2n} * X_n &\leq b_2 \\ \vdots \qquad\qquad\quad &\vdots \ \vdots \\ a_{m1} * X_1 + \cdots + a_{mn} * X_n &\leq b_m \end{aligned}$$

が与えられたとする。ここで$a_{11}, \cdots, a_{mn}$は実数値、$X1, \cdots, Xn$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X_jについて

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

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_iについて

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

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$[inf(X_i;CT), sup(X_i;CT)]$として与えられる。

(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 1 $F_1, F_2, \cdots, F_m$が線形形式である時に、
最小値関数 $min(F_1, F_2, \cdots, F_m)$ を *mini-linear form* といい、
最大値関数 $max(F_1, F_2, \cdots, F_m)$ を *maxi-linear form* という。

定義 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、

$$CT: \quad \{X \leq U_1, X \leq U_2, \cdots, X \leq U_m, \\ X \geq L_1, X \geq L_2, \cdots, X \geq L_n\}, \quad (U_i, L_i \text{ は線形形式とする。})$$

この時、

$upper(X;CT)$	$=$	$min(U_1, U_2, \cdots, U_m)$	$m > 1$ のとき (*mini-linear form*)
	$=$	U_1	$m = 1$ のとき
	$=$	*infinite*	上限式がないとき
$lower(X;CT)$	$=$	$max(L_1, L_2, \cdots, L_n)$	$n > 1$ のとき (*maxi-linear form*)
	$=$	L_1	$n = 1$ のとき
	$=$	*minus-infinite*	下限式がないとき

とする。

定義 3 次の補助関数を定義する。

(i) 変数 X と *mini-linear form* $F = min(F_1, \cdots, F_m)$ に対して、

$$supp(X, F) = min(supp(X, F_1), \cdots, supp(X, F_m))$$
$$supp(X, F_j) = supremum\{X; X \leq F_j, X\text{以外の変数は定数とする}\}$$

(ii) 変数 X と *maxi-linear form* $F = max(F_1, \cdots, F_n)$ に対して、

$$inff(X, F) = max(inff(X, F_1), \cdots, inff(X, F_n))$$
$$inff(X, F_j) = infimum\{X; X \geq F_j, X\text{以外の変数は定数とする}\}$$

と定義する。*supp* の計算式を示しておく：

1. F が数値のとき	$supp(X, F) = F$
2. $F = X$ のとき	$supp(X, F) =$ *infinite*
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 $G \geq 0$ のとき	$supp(X, F) =$ *infinite*
3.3.1 $G < 0$ のとき	$supp(X, F) =$ *minus-infinite*
4. $F = min(F_1, \cdots, F_n)$ と書けるとき	$supp(X, F) = min(supp(X, F_1), \cdots, supp(X, F_n))$

inff についても同様である。

定理 1 ([Shostak],[Bundy]) 次のように条件を与えるとする

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)

$sup(F, Vs; CT)$, $inf(F, Vs; CT)$ 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	X が Vs に含まれる時
s3	$sup([X], Vs; CT)$	=	$supp(X, Z)$ ここで Z = sup(upper(X;CT),Vs+X;CT)	X が Vs に含まれない時
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	$sup(min(F1, \cdots, Fm), Vs; CT)$	=	$min(sup(F1, Vs; CT), \cdots, sup(Fm, Vs; CT))$	

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	X が Vs に含まれる時
i3	$inf([X], Vs; CT)$	=	$infp(X, Z)$ ここで Z = inf(lower(X;CT),Vs+X;CT)	X が Vs に含まれない時
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	$inf(max(F1, \cdots, Fm), Vs; CT)$	=	$max(inf(F1, Vs; CT), \cdots, inf(Fm, Vs; CT))$	

(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：

(i) 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）。

(ii) 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）。

(iii) 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）。

(iv) 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 $sup([Form], [U, V, W, \cdots]; CT)$（$Form$ 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 $sup([X], [U, V, W, \cdots]; CT)$ の形式のものに還元される。

この $sup([X], [U, V, W, \cdots]; CT)$ 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 $sup([X],[U,V,W,\cdots];CT)$ 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(図 1)。

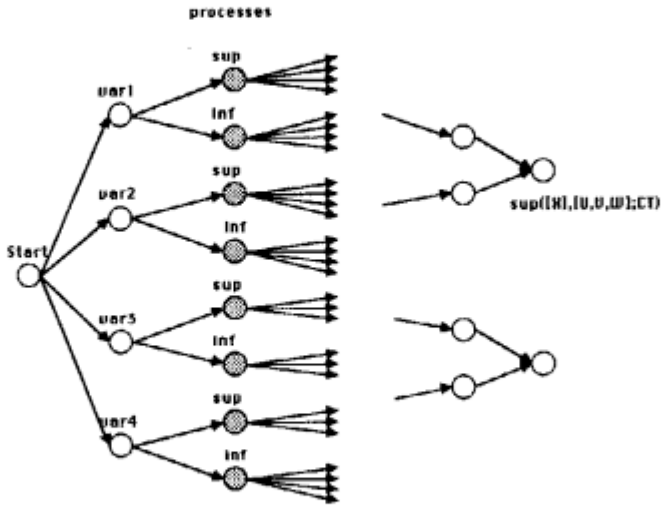

図 1: サブゴールの呼び出し

(e) SUP-INF アルゴリズムの特徴
SUP-INF アルゴリズムの特徴は次のようになる。

(i) 手続き sup と 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。

(ii) 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。

(iii) 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。

$$\begin{array}{lcl} sup(X1;CT) & , & inf(X1;CT) \\ sup(X2;CT) & , & inf(X2;CT) \\ & \cdots & \\ sup(Xn;CT) & , & inf(Xn;CT) \end{array}$$

(iv) 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)-(iv) について次に説明をする。

(3-2)(d) で述べたように手続き sup と inf ともに再帰呼び出しの中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 $sup([X],[U,V,W,\cdots];$ $CT),inf([X],[U,V,W,\cdots];CT)$ の結果が新しく得られる度にここに現われる $([X],[U,V,W,\cdots])$ 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。

(3-3) 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32台、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)-(i)～(e)-(iv)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。

(i) 不等式に現れる変数ごとにプロセスを振り分ける。

(ii) 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。

(iii) 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。

(i)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。(ii)についても全く同じである。

(iii)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(iii)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(3-2)(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：

(i) 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。

(ii) 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(図2)：

(i) 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る。

(ii) 上の (3-2) 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。

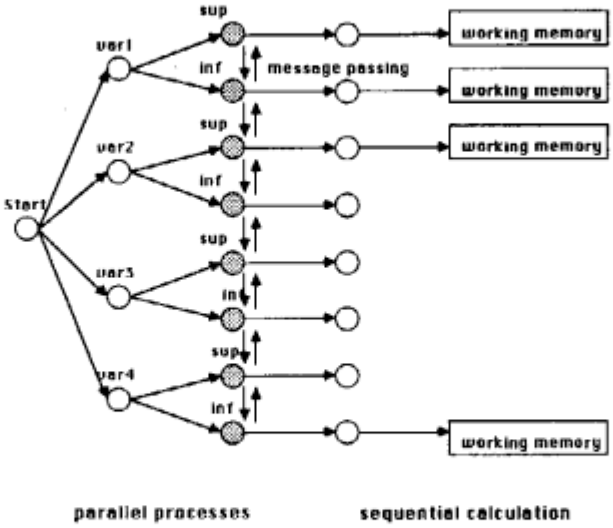

図 2: 並列プロセスの振り分け

こうすると sup も inf とも変数それぞれに対して独立に計算が行えるので $2 \times$ 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（[淵]）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。

(3-4) まとめ

上で示した並列計算の方式では $2 \times$ 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。

また非線形制約についてもSUP-INF法と同じ様に変数を上限式、下限式で繰り返し置き換えていく方法が考えられている([Sacks])が、非線形のときには一つの変数の上限式、下限式を他の変数で表すことができないことが殆どなので、この方法で解ける問題には限界がある。この方法について検討してみることも今後の課題となろう。

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.

[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.

[川岸] 線形不等式を解く制約ソルバーの並列計算について 情報処理学会ソフトウェア基礎論研究会, 1989年12月.

[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.