マルチスレッド機構を用いたLOTOS仕様実行方式とその評価

2E−6

安本慶一[1]　東野輝夫[2]　谷口健一[2]　松浦敏雄[3]

[1] 滋賀大学経済学部情報管理学科　[2] 大阪大学基礎工学部情報工学科　[3] 大阪市立大学生活科学部

1　はじめに

形式記述言語LOTOS [3] によるシステムの仕様を効率良く実行するためには，マルチランデブと呼ばれる複数並列プロセス間でのイベントの同期実行を高速に実現する必要がある．本稿では，マルチスレッド機構 [1] を用いた効率の良いLOTOS仕様の実行方式を提案する．

2　LOTOS仕様の実装法

LOTOSではイベントの時間的実行順序を指定することにより，システムの仕様を記述する．実行順序の指定にはアクションプレフィクス (a;β)，選択 ($\alpha[]\beta$)，並列 ($\alpha|||\beta$)，同期 ($\alpha|[G]|\beta$)，逐次 ($\alpha >> \beta$)，割込 ($\alpha[> \beta$) が使用できる [3]．

2.1　LOTOS仕様の実装方針

マルチスレッド機構を用いてLOTOS仕様を効率良く実装するには，仕様に指定されている動作を，並列処理可能な単位動作 (ランタイムプロセスと呼ぶ) に分解し，それぞれをスレッドとして処理するのが望ましい．本稿では，ランタイムプロセス間に指定されているオペレータの階層関係を表す構文木を制御領域として生成し，各ランタイムプロセスが制御領域内の適当なノードに参照・書込を行うことで，これらの間の実行依存関係 (選択，割込，同期) が実現されるような仕組みを考える．

2.1.1　動作式の基本動作式群への分解

動作式Bが$B_1\ op\ B_2$で表され，かつopが$|||, |[G]|, [>$のいずれかである場合には，Bを部分動作式B_1, B_2に分割する．分割された動作式B_1, B_2についても同様に分割を行い，分割が不可能になるまで繰り返す．一方，選択実行 ($B_1[]B_2[]...[]B_k$) においては，全ての$B_i(1 \leq i \leq k)$が$a;B$の形式である場合，その全体を1つのスレッド内で実現する．従って，Bが選択実行 ($B_1[]B_2[]...[]B_n$，以後$\sum_{i=1}^{n} B_i$と記述) の時は，逐次系列の選択実行$\sum_{i=1}^{k} B_i(B_i = a_i; B_i')$の部分と逐次系列でない動作式の選択実行$\sum_{i=k+1}^{n} B_i(B_i \neq a; B_i')$の部分に分割する．以上のようにして分解された各部分動作式を**基本動作式**と呼び，Bに含まれる基本動作式の集合を$F(B)$で表す．

2.1.2　目的コードの内容

基本動作式は (1) イベント系列 ($a; B$)，(2) イベント系列の選択 ($\sum_{i=1}^{k} a_i; B_i$)，(3) 逐次実行 ($B_1 >> B_2$)，(4) プロセス呼出し ($P[g_1, \cdots, g_n](E_1, \cdots, E_m)$) のいずれかとなる．(1)，(2) の形式の動作式をランタイムプロセスと呼ぶ．

各基本動作式RはC言語の関数$C(R)$として実現される．$C(R)$の内容は，(1)，(2) では，「各イベントの実行可能性を制御領域を参照して判定し，実行可能なら，$C(R)$が実行を行うことを制御領域に書込み，イベントを実行，不可能ならイベントを実行せず$C(R)$を終了」である．

(3) の逐次実行 ($B_1 >> B_2$) では，「$F(B_1)$に含まれる各基本動作式に対応するCの関数をスレッドとして実行し，それらが全て終了するのを待った後，$F(B_2)$に対応するCの関数群をスレッドとして実行」のように実現する．(4) の場合は，プロセスPの動作式B_Pに対して「$F(B_P)$に対応するCの関数群をスレッドとして実行」となる．$F(B_1), F(B_2), F(B_P)$に (3)，(4) の形式の基本動作式が含まれる場合は再帰的に上記の処理を行う．また，基本動作式の部分動作式でランタイムプロセスに分解可能なものについては，分解後のランタイムプロセスを実現するCの関数を生成する．

```
process P[a,b,c,d]:noexit :=
    ((a;b;exit >> c;d;exit) []1 c;a;exit [] d;b;exit)
>> (a;c;exit |[a]| d;a;b;exit)
endproc
```

上記仕様から次のような動作を行う目的コードを生成する．以下では，オペレータ[]，$|[a]|$を実現するための制御領域，ランタイムプロセス群 ($a; b; exit(\equiv R_1)$, $c; d; exit(\equiv R_2)$, $c; a; exit[]d; b; exit(\equiv R_3)$, $a; c; exit(\equiv R_4)$, $d; a; b; exit(\equiv R_5)$)，基本動作式$R_1 >> R_2$を実現するCの関数が既に生成されていると仮定している．

まず，プロセスPの動作式B_Pが$B_1 >> B_2$の形の基本動作式で，$F(B_1) = \{R_1 >> R_2, R_3\}$なので，$C(R_1 >> R_2)$, $C(R_3)$が制御領域のノード$[]^1$を参照点として並列に実行される．$C(R_1 >> R_2)$では前述のようにまず$C(R_1)$を実行し，$C(R_1)$の終了後，$C(R_2)$を実行する．$C(R_1)$, $C(R_3)$が参照している制御領域のノードはともに$[]^1$なので，これらのうち最初にイベントを実行可能になったランタイムプロセスが実行権を獲得する ($C(R_1)$の場合はイベントa, $C(R_3)$の場合は，イベントc, dのどちらかが実行可能になれば，実行権の獲得作業に移る)．実行権を獲得したランタイムプロセスは，ノード$[]^1$に$C(R_1)$(もしくは$C(R_3)$) が実行を行う事を書込み，イベントを実行する．後で実行可能になったランタイムプロセスはノード$[]^1$に書込まれた内容を見て他のランタイムプロセスが既に実行されていることを検出しイベントの実行をやめプロセスを終了する．$C(R_1), C(R_3)$が終了したら，$F(B_2) = \{R_4, R_5\}$から，$C(R_4), C(R_5)$が制御領域のノード$|[a]|$を参照点として並列に実行される．R_4はイベントaについて同期が必

An Implementation and Evaluation of LOTOS Specifications using Multi-thread Mechanism

Keiichi YASUMOTO, Teruo HIGASHINO, Kenichi TANIGUCHI and Toshio MATSUURA

Department of Information Processing and Management, Faculty of Economics, Shiga University, Hikone 522, Japan

表 1: マルチランデブの生起条件

P_i	P_j	同期条件	作用
$a!E_i$	$a!E_j$	$val(E_i) = val(E_j)$	値の照合
$a!E_i$	$a?x:t$	$val(E_i) \in domain(t)$	値の代入
$a?x:t$	$a?y:u$	$t = u$	値の生成

同期の効果として, 代入 $x \leftarrow val(E_i)$, 値の生成 $x, y \leftarrow V, V \in domain(t)$ が行われる ($val(E)$ は式 Eの正規形, $domain(t)$ はソート t の値域).

要であることを知り, ノード $||[a]||$ に $C(R_4)$ が同期待ちであることを登録する. 一方, $C(R_5)$ は同期の必要がないイベント d の実行後, 次のイベント a の実行時にノード $||[a]||$ を見て同期相手を探す. この場合同期相手 $C(R_4)$ があるので, ノード $||[a]||$ を通じて同期可能であることを相手に通知し, a を実行する. 以上のように動作が進行して行く.

3 同期の実現

同期実行 $(B_1||[G]||B_2)$ の場合, $R \in F(B_1) \cup F(B_2)$ の各 $R(\equiv \sum_{i=1}^{k}(a_i; B_i))$ は, 以下のことを行う必要がある. ここで, Rが最初に実行可能なイベント群 $\{a_1, ..., a_k\}$ を Eとする. Eのうち G に指定されているゲートを持つイベントの集合を E_sとする. 各 $a_k \in E_s$について,

(1) 同期条件 (表 1) を満たす a_kの同期相手が '$||[G]||$' に登録されているか調べる.

(2) 登録されていなければ, a_kのゲート名と入出力値を '$||[G]||$' に登録し, 同期する相手を待つ. 後に同期相手が見つかれば, a_kを同時に実行する.

(3) 登録されていれば, '$||[G]||$' を通じて相手に同期相手が見つかったことを知らせ a_kを同時に実行する.

E_sのうち同期可能になったイベントの集合を E'_sとする. 同期オペレータが階層的に指定されている場合は, E'_s について, 上部構造内のオペレータ ($||[G']||$) に対して再帰的に上記のことを行う必要がある (この際, E'_s の各イベントの入出力値は「値の代入」により 2 つのプロセス間で一致させた値を用いる). 最上位のオペレータ ($||[G^*]||$) について同期相手が見つかった時, 同期は成功する. ただし, ランタイムプロセス間に選択, 割込が指定されている場合, 複数同期グループ間の排他制御が必要になる (実現法は文献 [5] 参照).

4 本実行方式の評価

目的コードの実行効率を他のコンパイラ COLOS [2] および TOPO [4] と比較した. COLOS は目的コードの実行に SUN のライトウェイトプロセスを用いている.

並列処理の数が増えた時のオーバーヘッドの増加度合の調査, およびコンパイラ間の比較のため, 数百のランタイムプロセスを並列実行した時, 単位時間あたりに実行されるイベントの数を測定した. Sun SparcStation IPX (24MB RAM) で測定した結果を表 2に示す. 並列実行においては, 本コンパイラの生成する目的コードが COLOS, TOPO に比べて, それぞれ 2.5〜3 倍, 30〜60 倍程度効率が良い. また, 1 つのスレッドに 8KB のスタック領域を割り当てた場合, 我々のコンパイラでは, 数千のランタイムプロセス群を含む LOTOS 仕様のコンパイル・実行が可能であった.

同期実行の性能を調べるため, 同期するプロセス (制約) の数を増やした時の単位時間あたりに実行されるイベント数の変化を調べた. 100 個のランタイムプロセスの並列実行からなる仕様 Bに対して, $B||B$, $B||B||B$, $B||B||B||B$, $B||B||B||B||B$を実験用の仕様として用いた. 結果を表 3に示す.

表 2: 並列処理における単位時間あたりの実行イベント数

ランタイムプロセスの数	Ours	COLOS	TOPO
100	1724/s	700/s	57/s
200	1214/s	421/s	26/s
300	928/s	276/s	17/s
400	712/s	227/s	12/s
500	565/s	183/s	9/s

表 3: 同期処理における単位時間あたりの実行イベント数

動作式	Ours	COLOS	TOPO
B	1724/s	700/s	57/s
$B\|\|B$	244/s	509/s	12/s
$B\|\|B\|\|B$	195/s	365/s	0.9/s
$B\|\|B\|\|B\|\|B$	175/s	273/s	—
$B\|\|B\|\|B\|\|B\|\|B$	155/s	224/s	—

表 3によれば, 同期実行が多く含まれる場合には, COLOS の方が早い. しかし, COLOS では, 扱える LOTOS 仕様のクラスを制限している [2]. そのような制限された記述に対しては我々の実行方式をもっと精密化し, さらに効率的にすることも可能である. 我々のコンパイラは, 動作式の記述に LOTOS の全てのオペレータを使用でき, 動作式の形に制約を課していないなど, より広いクラスの LOTOS 仕様を対象にしている.

5 おわりに

本実行方式では LOTOS の選択, 並列, 同期, 逐次, 割込などの基本オペレータ, プロセス呼出しにおけるゲートのリラベリングなどを扱うことができ, 抽象データ型についても, 我々が設計・開発している関数型言語のコンパイラを用いて自動実装を行っている (ただしクラスは関数型に限る) [5]. また, 生成される目的コードは我々が開発した移植性に優れたマルチスレッド機構 [1] を用いて動作するため, 多くのアーキテクチャ上で動作する. 以上より, 本コンパイラは実際のプロトコルの開発などに広く適用可能であると思われる.

参考文献

[1] 安倍広多, 松浦敏雄, 谷口健一: BSD UNIX 上での移植性に優れた軽量プロセス機構の実現, 情報処理, Vol. 36, No. 2, pp. 296-303 (1995).

[2] Dubuis, E.: An Algorithm for Translating LOTOS Behavior Expressions into Automata and Ports, *Proc. of the 2nd Formal Description Techniques (FORTE'89)*, pp.163-177(1990).

[3] ISO : LOTOS: A Formal Description Technique based on the Temporal Ordering of Observational Behaviour, *ISO 8807*(1989).

[4] Manas, J. A., Salvachia, J.: Aβ: a Virtual LOTOS Machine, *Proc. of the 4th Formal Description Techniques(FORTE'91)*, pp.445-460(1991).

[5] Yasumoto, K., Higashino, T., Abe, K., Matsuura, T. and Taniguchi, K.: A LOTOS Compiler Generating Multi-threaded Object Codes, *Proc. of the 9th Formal Description Techniques (FORTE'95)*(to appear in Oct. 1995).