OS/omicronにおける並列処理用C言語プリプロセッサ「ECPP」の設計

4P-4

小松徹、並木美太郎、高橋延匡
(東京農工大学)

1．はじめに

当研究室では、スーパーパーソナルコンピュータ用のオペレーティングシステムとしてＯＳ／omicron を開発している。

ＯＳ／omicron はマルチタスクＯＳで、タスクの生成や消滅、ＰＶセマフォによる同期基本命令、メッセージ通信などのＳＶＣを用意している。しかし、実用規模のアプリケーションの作成や並列アルゴリズムの研究に、ＳＶＣ（Super Visor Call）による記述が適切かどうか疑問である。ＳＶＣはＯＳが用意する低レベルの命令であり、記述対象とのギャップが大きくなる可能性がある。

ＯＳ／omicron をマルチプロセッサで動かす計画もあり、今後、並列処理の種々の実験が行われることを考えると、ＳＶＣに変わる言語が必要になることは明白である。

以上の理由から筆者は、種々の並列処理の実験、および並列アルゴリズムを記述すためる言語の必要性を感じ、ＥＣＰＰ（Extended C for Parallel Programming）を設計した。

2．ＥＣＰＰの設計および実現方針

2．1　設計方針

（1）アプリケーション指向

研究室内部にはまだ並列処理記述言語がない。したがって、ＥＣＰＰはＯＳ／omicron における並列処理の実験、および並列アルゴリズムの記述を第１目標として設計した。

（2）同期命令を扱いやすくする

並列処理の最も扱いの難しいものの一つに同期命令がある。一般に低レベルな同期命令は強力ではあるが、扱いが難しい。

ＯＳ／omicron では同期基本命令として、セマフォを採用している。セマフォは単純ではあるが非常に強力な同期基本命令である。しかし、セマフォによる同期は、一度でも操作を抜かしたり、間違えたりすると期待していた結果を得ることができない。同期が何重にもなる場合には、セマフォによる同期は非常に複雑になる。アプリケーションを手軽に記述するためには、もっと高度な同期命令を導入する必要がある。

同期の問題を解決するために、ＥＣＰＰでは、ＣＳＰに代表されるメッセージ指向モデルを導入する。このモデルの特徴は同期がメッセージの送受信によって行われる点にある。また、ハードウェアに依存する部分が少ないのも特徴である。

同期を入出力命令に集中することにより、プログラミングが大幅に楽になる。また、データの流れを明確に把握しやすくなる。プログラマは解こうと思う問題をいくつかの独立したプロセスに分解し、あとはプロセス間のデータの受渡しを考えるだけでよい。

（3）プロセス単位の並列処理

並列処理はさまざまなタイプがあるが、ＥＣＰＰでは逐次プロセス単位での並列処理を目指す。プロセスは通信による疎な結合で結ばれる。プロセスは独立であり他のプロセスと共有するデータは存在しない。また親子関係も存在しない。同期を入出力に集中させ、プロセスを独立させることによって、部分的変更が全体に及ぼす影響を最小限に食い止める。デバッグや、仕様変更などが簡単に行えるようする。

プロセスにプライオリティはない。すべては平等であり、実行順序は入出力命令以外では要求されない。プロセスがどのように実行されるかはすべてスケジューラの責任であり、ＥＣＰＰでは関与、指定できない。

（4）構造化

複雑なアプリケーションでも、論理的な部分でまとめて扱えるように、プロセスを構造化できるように設計する。

以上４点を中心としてＥＣＰＰを設計した。

2．2　実現方針

現在、当研究室ではＯＳ／omicron のマルチプロセッサ化が進められており、新しい並列処理言語が要求されている。当研究室では多方面から並列処理の研究を進める計画であり、以下のような要求がある。

(1) 覚えやすく
(2) 理解しやすく
(3) 書きやすい

以上３つの条件を満たした言語を早急に作らねばならない。しかし、全く新しい言語を一から開発するのは非常に時間のかかることである。また、研究室内部でも、新しい言語の教育には時間がかかる。当研究室では独自に開発したＣコンパイラ“ＣＡＴ”でシステムを開発しているので、Ｃ言語に対する知識は皆持っている。研究室内部の者にとっては、構文などもＣ言語に近い方が違和感が少ないと考えられる。

以上の理由から、ＥＣＰＰはＣＡＴのプリプロセッサとして設計した。

3．ＥＣＰＰの特徴と機能

（1）Ｃ言語に近い構文

ＥＣＰＰでは、プロセス内のシーケンシャルな部分で、ポインタを含めたＣ言語で使用できるすべてのテクニックが使用できる。

ただし、プログラムは通信によるプロセスの疎な結合からなるので、プロセス以外の外部の変数に対するアクセスはで

A Design of the Concurrent Programming Language ECPP on OS/omicron
Tohru KOMATSU,　Mitarou NAMIKI,　Nobumasa TAKAHASHI
Tokyo University of Agriculture and Technology

きない。また、プロセス内部の変数が外からアクセスされることもない。

（２）構造ファイル

ＥＣＰＰでは構造の定義をプログラムの定義と分離する。構造を定義するために“構造ファイル”を導入する。プロセスの構造はすべて構造ファイルの中で定義される。

```
process main {
        process yuko {
        }
        process aiko {
                process keiko {
                }
                process kazumi {
                }
        }
        process hiroko {
                process mayumi {
                }
                process ryouko {
                }
        }
}
```

図１　構造宣言の例

ＥＣＰＰによる構造化は、代表者を決め、外からはその代表者にしかアクセスできないようにするものである。例えば以下の図２に示すように

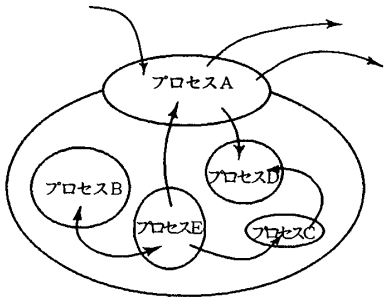

図２　構造化されたプロセスの例

プロセスＡは外から参照されるが、プロセスＢからＥは外からは参照されることはない。したがって、他の構造化されたプロセス内に同じ名前のプロセスが存在しても問題はない。

プロセスＡからＥまで相互に自由に通信可能である。ただし、プロセスＢからＥは直接外と通信できない。外のプロセスと交信できるのはＡだけである。

プロセスは“自分と同じレベル”にあるプロセスと通信できる。その構造を決定するのが構造ファイルである。

ＥＣＰＰにおける構造化されたプロセス間通信の例を示す。

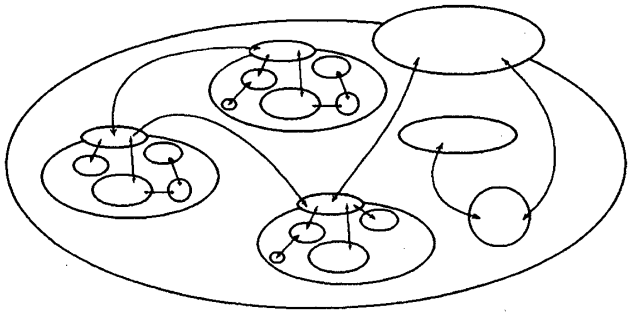

図３　ECPPの通信の例

構造ファイルは、プロセスの構造を記述するためのものである。通信構造体の型宣言などは行わない。

（３）同期および入出力命令

プロセス間の同期は通信に集中させる。ユーザはＳＶＣでセマフォを用いた煩わしい制御をする必要がない。入出力命令は先に準備が整った方が相手を待ち、お互いに通信が可能になったとき初めて同時に実行される。通信は同期命令なのでバッファリングはない。入出力命令はプロセスの名前で管理され、通信するときには相手を必ず指定しなければならない。

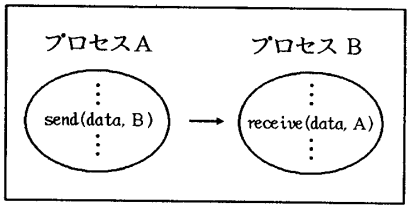

図４　プロセス間通信の例

プロセスＡのsend命令が先に呼ばれた場合、Ａは表を調べることによってプロセスＢがまだrecieve 状態にないことを知る。Ａは自ら表に自分が出力可能状態にあることを記録して、スケジューリングの対象からはずれる。後からＢがreceive 命令に到達したとき、Ａが起動されて通信が同時に実行される。

（４）プロセスの起動、終了

ＯＳ／omicron のＳＶＣでは、プロセスの生成と実行、停止と削除は別に扱っている。しかし、ＥＣＰＰはプロセスを生成と同時に起動し、プロセスが終了したとき削除を行なう。また、プロセスは入出力待ち以外では停止しない。

プロセスは動的に生成、消滅する。起動するときは、あるモジュールに対して名前を与えて起動することができる。プロセスは名前によって管理されるから、同じ構造を持った名前の異なる複数のプロセスを同時に実行させることができる。

（５）資源の排他制御

資源に対する排他制御は際どい領域を用いる。プリンタなどの装置やファイルシステムはすべて際どい領域で管理される。

４．おわりに

今回はＥＣＰＰの機能設計について説明した。今後これらの設計をもとにＥＣＰＰを作成し、ＯＳ／omicron における並列処理の実験を行う予定である。

５．参考文献

(1) C.A.R. Hoare
"Communicating Sequential Processes"
Communications of the ACM
Volume 21 Number 8 p666-p677, August 1978
(2) 武山潤一郎：タスクの制御方式に関する研究、
修士論文、東京農工大学、（１９８５．１）