並列処理プログラミング実験

5S－5

豊福 一正　阿刀田 央一　冨澤 眞樹　田村 仁
東京農工大学大学院工学研究科

1.はじめに

近年、並列計算機や並列処理用言語の発展が著しい。研究開発のレベルにとどまらず、パーソナルユースのレベルにも浸透しつつある。これによって今後、並列処理プログラム記述の機会の増加が予想される。並列処理プログラムを書くとき、プログラマはどのような発想をし、それをどのようにプログラムに反映させ、完成させるのであろうか。そこで、これまで逐次処理プログラムを書いた経験のあるプログラマに並列処理用言語を教え、実際に並列処理プログラムを記述させる実験を行った。

この実験の結果をもとにプログラマのプログラム完成までの思考についての考察を行った。

2.実験方法

実験は、ワークステーション上の並列計算機シミュレータと、並列処理用言語として、参考文献[1]で提唱された、C//を用いて行った。C//とはC言語に並列処理を行うための並列手続呼出と排他制御を付加した言語で（表１）、C言語の知識を持っていれば、付加された部分を覚えるだけで容易に並列処理プログラムを書くことができる言語である。被験者は、C言語を用いたプログラミング経験があり並列処理の経験はない、学部の２年生、３年生、４年生、合計９人であった。C言語の経験のあるものとしたのは、C言語の知識不足がC//の理解を妨げないためである。

表１ C//に付加された並列プリミティブ

内容	対応する文、演算子、変数
並列呼出し、受け取り	トランスファ変数、//=演算子、//演算子
呼び出した処理の中止	中止演算子
共有変数の取り扱い	ロック文、キーホール変数
選択的な結果の受け取り	among文

実験に先立って被験者に、事前に製作したC//に関する資料を配布した。資料には、C//の文法についてC言語から付加された部分を中心に記述し、並列計算機ハードウェアについては一切記述していない。C//は、ハードウェアに依存せずに並列処理を記述できる。ハードウェアについての情報を与えず、言語の情報だけを与えられてどのような発想をするかを観測するため、このようにした。被験者に与えた知識はこの資料だけで、講義等は一切行なわなかった。個々の質問は受け解答するが、被験者は基本的にマニュアルだけをもとにプログラムを作成した。

実験は、被験者に一日一題のプログラミング課題を与え、プログラムを記述させることで進行した。与えた課題は、数のサーチ、数のソート、数値計算である。被験者が課題のプログラミングの際、実際に書いたプログラムをコンパイル毎に、UNIXのRCSを用いて自動的に記録し、行動や質問とそれに対する応答などをメモを取ることで記録した。

実験に際して、被験者に２つのルールを課した。

Investigation of Parallel Programming
Kazumasa TOYOFUKU, Oichi ATODA,
Masaki TOMISAWA, Hitoshi TAMURA
Tokyo University of Agriculture and Technology
2-24-16 Nakamati, Koganei, Tokyo 184, Japan

一つはプログラムを書く前に必ず仕様書を書き、仕様の変更時にはそれを申告することである。これは、被験者が課題を渡された時点で、どの様な発想でプログラミングに取りかかり、プログラミングの過程でそれがどの様に変わっていくかの資料とするためである。そしてもう一つは、会話の禁止である。プログラミング中に被験者が不明に思った点については、被験者と実験者との筆談によって対応し、実験が終わるまでは実験に関する一切の被験者間の会話も禁止とした。これは、会話による質問の応答が、一人の被験者が持った疑問の解答を他の被験者にも与えてしまうといった、不必要な情報の流出を防ぐためである。

一度の実験で２～３題の課題を行い、課題毎にレポートを与えた。それとは別に全体の感想などを書くレポートも与え、全ての課題とレポートを含めて一度の実験時間は２０時間程度であった。

3. 実験結果

被験者の記述したプログラムには、次のような並列化の方法があった。

- 人海戦術的な関数呼出し（同一関数の複数呼出し）。
- 並列呼出し関数から、さらに他の関数の並列呼出し。
- 再帰拡散による並列化。

その方法を作り出す経過として、次のようなものがあった。

- 逐次でプログラムを作って、それを並列処理に直す。
- 始めから並列処理プログラムにする。
- 並列処理を一時逐次処理に直し、再並列化。

また、次のような例も見られた。

- 並列呼出しをしてはいるが、論理的に並列処理になっていない。
- 変数が共有変数になっているのに気が付かない。
- 並列処理を断念して逐次処理化。
- メモリがあふれるほどの大量の並列呼出し。
- 並列化によるオーバーヘッドの方が大きくなるような処理の並列化。

これらの結果から、簡単な並列処理プログラミングは特に何の障害もなく可能であることが分かった。しかし、逐次処理では見られない排他制御のような処理は敬遠しがちであるし、並列呼出しのオーバーヘッドなどはその情報を与えられない限り気付かない。つまり、少なくとも本実験の被験者の場合、並列プログラミングのベースとして逐次処理プログラミングがあり、逐次処理で考えてみて並列化可能にみえる部分を並列処理に置きかえているようである。

4. おわりに

本実験では、逐次処理プログラミングの経験があり、並列処理プログラミングの経験はない者を対象に、並列処理プログラミング実験を行なった。

この結果、並列処理プログラミングの初心者は逐次処理をベースに並列処理プログラミングを発想し、記述するという結果を得た。

今後は、逐次処理プログラミングの経験がない者に並列処理プログラミングを行なわせたり、ハードウェアの知識も与えてプログラミング実験を行なってみたりする必要があるだろう。

[参考文献]

[1]冨澤ほか：発注受領型分散制御機構を持つ密結合並列計算機用言語 C//の設計と実現，電子情報通信学会論文誌 D-I Vol. J75-D-I No. 11 pp. 1025-1036(1922)