卒業研究報告書

題 目

PRAM シミュレータの作成

指導教員

石水隆助手

報告者

01-1-26-067

北浦 邦浩

近畿大学理工学部電気工学科

平成 17 年 2 月 19 日提出

第1章　序論[1)]

1.1.　並列コンピュータの概要

今日存在する大多数のコンピュータは、概念的には非常によく似ており、それらのアーキテクチャと演算子は、概ね J. von Neumann が考案し、1940 年代に定式化された設計原理に従っている。J. von Neumann 型コンピュータの基本的な流れは、次の通りである。まず制御装置（Control　Unit）は命令（Instruction）とオペランド（Operand）を記憶装置（Memory Unit）から取ってきて、それらを処理装置（Processing Unit）に送る。次にその命令は処理装置で実行され、その結果がメモリに戻される。この動作系列が各命令ごとに繰り返される。J. von Neumann 型コンピュータの処理装置、制御装置、記憶装置がそれぞれただ１つあり、一度にただ一つの命令を実行する。

一方、複数の処理装置またはプロセッサからなる並列コンピュータでは、ある問題を解く際に、その問題の異なる部分問題を同時に解き、全体の結果を導くのに協力し合い並列処理を行う。並列コンピュータを用いて並列処理することによって、単一プロセッサのコンピュータを用いて問題を解くのにかかる時間よりも非常に少ない時間で解くことができる。

1.2.　並列コンピュータの必要性

並列コンピュータは様々な分野で求められている。例えば、大気圏外の一組の人工衛星は１秒間に、10^{10} ビットの割合でデータを収集している。そのデータは地球の気象、汚染、農業、天然資源に関する情報を含んでいる。この情報が適切に使われるためには、少なくとも１秒間に 10^{13} 回以上の演算速度で処理される必要がある。

また、海洋学者は、海洋の大域的な循環の数値シミュレーションを行っている。彼らの目標の一つは、南半球の海洋の熱がどのように南極に伝わるかを知ることであり、これは大域的な温暖化問題を理解する重要なステップである。正確な結果を得るために、科学者は海洋を東西に 4096 の領域、南北に 1024 の領域、垂直方向に 12 層に分割することを計画している。このモデルは、 $4096\times1024\times12=48\times2^{20}\fallingdotseq50\times106$ の 3 次元セルを持つことになる。このモデルを 1 回実行すると、10 分間の海洋の循環をシミュレートすることができ、そのために約 30×10^{9} 回の不動小数点が必要とされる。

医療分野では、外科チームは、手術の準備のために、患者の体の３次元画像を再構成して特殊ディスプレイで見たいと思っている。そこでは、患者に触れることなく、自由に画像を回転し、器官の断面図を得て、生体の詳細を観察し、その効果を見ながら手術をシミュレートする必要がある。少なくても 1 秒間に 10^{15} 回以上の操作の処理速度が必

要である。このように並列コンピュータは高速な処理を求められる様々な分野で必要とされている。

1.3. 実際的な並列処理の出現と課題

ここ二十年ほどの間に、並列処理は高速計算に対する魅力的で実行可能なアプローチとなってきた。コンピュータハードウェアの値下がりによって、数万プロセッサを用いて並列マシンを組み立てることが可能になった。計算機科学者は、理論と実際の双方から並列コンピュータを研究し始めた。自作のプロトタイプによる実験結果は、大多数の理論的な研究結果が正しいことを示している。

並列コンピュータが研究室から市場に移るまでには 20 年以上かかり、1980 年代の中半になって初めて、マルチプロセッサで組み立てられた商業並列コンピュータが出現してきた。各種クラスのコンピュータを基礎にした並列処理が最終的に実際的なのかを示している。ミニコンピュータ、メインフレーム、伝統的なスーパーコンピュータの性能の増加率は年 20%以下であった。これに対してマイクロプロセッサの性能の増加率は年 35%であった。

年々のマイクロプロセッサと伝統的スーパーコンピュータの性能の相対的な接近によって、数十台、数百台、または数千台のマイクロプロセッサよりなる並列コンピュータが、商業的に存立可能であるようになった。ピーク効率において、Intel の Paragon XP/S, MasPar の MP-2, Thinking Machines の CM-5 などのマルチプロセッサをベースにした並列コンピュータは、Cray Y/MP や NEC SX-3 などの単一プロセッサからなる伝統的なスーパーコンピュータの速度を上回っている.

並列的ハードウェアが得られるに従って、効率的で実際的、経済的に上手くいく方法で問題を解くために、並列コンピュータのプログラムをどのように作成したら良いかということが並列計算の課題になってきた。逐次的な世界と同様に、並列ハードウェア上で実際に計算を実行するためには、効率の良い並列アルゴリズム、プログラミング言語、コンパイラ、オペレーティングシステム等が必要である。

1.4. 並列計算モデル

並列アルゴリズムの設計、開発は実際の並列計算機を抽象化した並列計算モデルで行われる。代表的な並列計算機モデルとして PRAM(Parallel Random Access Machine)がある。PRAM は共有メモリ型並列計算機モデルであり、データの局所性及び通信にかかるコスト等は考えなくても良い単純なモデルである。

1.5. 本研究の目的

本研究では、PRAM 上のアルゴリズムを実行させることができるシミュレータを作成することにより、PRAM アルゴリズムの正当性と計算量の評価を実験的に行うことができるようにする。

第2章　原理

2.1.　並列計算モデル [2)]

並列計算モデルは、大きく２つに分類することができる。

・ 並列システム(Parallel System)

・ 分散システム(Distributed System)

並列システムとはプロセッサが協調しながら動くものである。全てのプロセッサが１つの目的のために協調して動く。すなわち、プロセッサ同士が共同して働き、非常に強く結合されている。高速でデータ交換できることを仮定している。強結合であるために、大量データをプロセッサ間で転送できる。この場合全てのプロセッサがほぼ同一となり、均一システムと呼ばれている。

それに対して分散システムは、必ずしも同一ではないプロセッサが資源を共有しながら、弱く結合された(データ転送速度が低速である)状態で作業を行うというものである。これはプロセッサ自体が離れた場所にあってもよいことを表している。トランザクション(１つのまとまった処理)の実行に際して、大量のデータをプロセッサ間で共有することなく、少量のデータのみ交換する。

並列システムと分散システムの両方に共有するのは、非常に多くの独立したプロセッサからなっているということである。並列システムと分散システムの最も大きな違いは協調するかしないかということと、結合度の強さである。逐次計算の場合には計算のみを考えれば十分であったが、並列計算では通信も考える必要がある。並列計算ではプロセッサ間のデータ転送の方法も様々であり、これが並列計算の最も難しい側面である。

並列システムにおいてはCPUとメモリの結合の仕方に多くの方式が考えられる。この結合がどのようになっているか、また、その結合のもとでどのように通信が行われるか、ということを定義しなければならない。したがって、モデルが一つしかないということはない。もっとも、ある種のモデルより使われるであろうということは考える。

本研究においては、並列システムを対象とする。

2.1.1.　並列計算モデル概要 [3)]

計算解析の初期の時代から逐次計算が使われていた。逐次計算の基本的かつ重要なモデルであるフォンノイマン(Von Noeumann Machine)は、非常に単純な構成で、データを処理するCPU(中央処理ユニット、(Central Processing Unit)とデータを蓄える為のメモリ(Memory)からなるものである。この２つの要素の間が双方向のチャネルで結ばれており、CPUからメモリに特定のデータへ要求ができて、反対方向にその答え(データ)がおくられる。 このような計算モデルをランダムアクセスマシン RAM(Random Access Machine)と呼ぶ。基本的なコストとして、チャネルの数に対する制限や、全ての命令の実行時間の和などを考えることができる。

メモリへのアクセス時間についてはメモリがLSIで表現されているとすると、非常にサイズが小さい為に、アクセス時間も短くなると考えることができる。実際にはメモリ量が大きくなるとメモリをアクセスする時間が遅くなる。しかし、メモリをアクセスする時間が変わるとしても、一番長くかかるものを持ってきて、それを基準としてモデル化することができる。従って、RAMではメモリ内のどの位置にあるセルであってもそのセル内のデータを取ってくるには、同じ時間だけかかるものとし、データをメモリ上のどこに割り当てても計算時間は影響しないと仮定する。逐次計算においては、RAMが標準的なモデルとして使用される。

一方、並列計算では合意できるようなモデルを１つ決めることが困難である。
理由として、計算以外に複数のプロセッサ間の通信を定義しなければならず、この通信構造について合意を得ることが困難であるためである。このため、通信構造を単純化した共有メモリ(Shared Memory Model)が一般に使われている。

2.1.2. PRAM[4)]

図１のように、複数のプロセッサがメモリを共有したコンピュータを共有メモリコンピュータ(Shared Memory Computer)呼ぶ。共有メモリコンピュータによる並列計算機モデルは、並列ランダムアクセスマシン PRAM(Parallel Random Access Machine)とも呼ばれる。

RAMと同様にPRAMでは共有メモリの読み書きも含めて全ての命令は１単位時間で行われると仮定されている。また、１命令ごとに全てのプロセッサで同期が取られる。共有メモリであるためPRAMアルゴリズムではデータの局所性を考える必要がない。またプロセッサ間の通信にかかるコストも無視されている。このためPRAM上では並列アルゴリズムの設計開発を簡易に行うことができ、並列アルゴリズムの解析も行い易い。加えて、PRAM 上で設計したアルゴリズムは、多くの場合他の並列設計モデル上でも効率良く動かせる。このため、並列アルゴリズムの設計は多くの場合PRAM上で行われる。

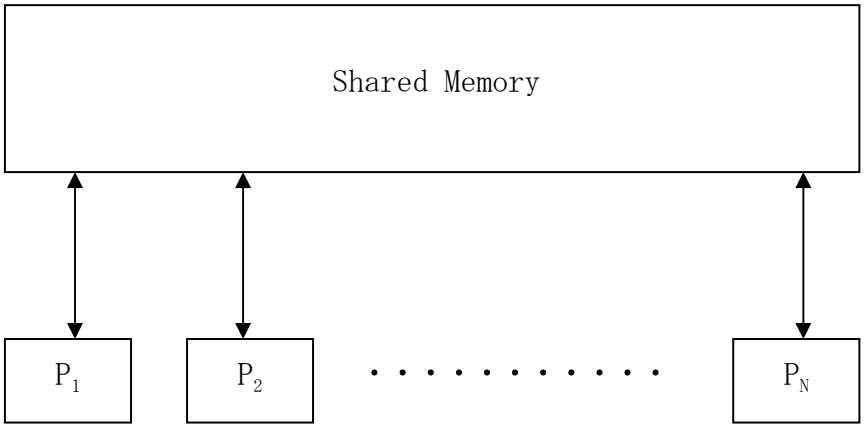

図１　共有メモリコンピュータ
Fig.1.Shared memory computer

共有メモリコンピュータの二つのプロセッサが共有メモリ(SM,Shared Memory)を通して通信し合う。これは、あるグループに属する人たちが掲示板を使うのと似ている。プロセッサ i がプロセッサ j にある数値を送りたい場合、これは２ステップで実行される。まず、プロセッサ i が、プロセッサ j の知っている所定の位置の共有メモリに、その数値を書き込む。その後、プロセッサ j がその位置から先ほどの数値を読み出せばよい。

並列アルゴリズムの実行中、N個のプロセッサは、入力データを読んだり、中間結果を読んだり書いたり、最終結果を書いたりするために、共有メモリにアクセスする。PRAMではプロセッサが読み出したり書き込んだりするメモリ位置が異なっているならば、すべてのプロセッサが共有メモリに同時にアクセスできる。一方、二つ以上のプロセッサが同じメモリ位置に同時にアクセスする場合は、それができるかどうかに従って、PRAMは、次のような４種類に分類される。

(1)EREW(Exclusive-Read Exclusive-Write)PRAM　どの二つのプロセッサも同じメモリ位置から同時に読み出したり書き込んだりできない。
(2)CREW(Concurrent-Read Exclusive-Write)PRAM　複数のプロセッサが同じメモリ位置から読み出すことができるが、どの二つのプロセッサも同じメモリ位置に同時に書き込むことはできない。
(3)ERCW(Exclusive-Read Concurrent-Write)PRAM　複数のプロセッサが同じメモリ位置に書き込むことができるが、どの二つのプロセッサも同じメモリ位置に同時に読み込むことはできない。
(4)CRCW(Concurrent-Read Concurrent-Write)PRAM　複数のプロセッサが同じメモリ位置に読み出すことも書き込むこともできる。

メモリ内の同じアドレスに多重の読み出しの権利を認めることは、原理的に何の問題も引き起こさない。概念的には、その位置から読み出すいくつかのプロセッサは、そのメモリの内容のコピーを作り、各プロセッサの局所メモリに格納する。一方アドレスに対する書き込みは複数のプロセッサが異なる値を書き込もうとした時に、どのプロセッサを優先するかという問題が発生する。

この問題への対処法により、CWCR PRAMは以下のような３種に細分化される。

(1)共有型(Common)CRCW PRAM　書き込みを行おうとした全てのプロセッサが同じ値を書き込もうとした時のみ値が成立する。
(2)任意型(Arbitrary)CRCW PRAM　書き込みを行おうとしたプロセッサのどれか一つが書き込みに成功する。
(3)優先型(Priority)CRCW PRAM　書き込みを行おうとしたプロセッサのうち、最も優先順位が高いものの書き込みが成功する。

2.2. 並列アルゴリズムの解析 [5)]

2.2.1. 実行時間

並列コンピュータを製作する背景にある主要な理由は、計算の高速化であるので、並列アルゴリズムを評価する最も重要な測度は実行時間である。並列コンピュータ上である問題を解くとき、実行時間はそのアルゴリズムを実行し始めてから終了するまでにかかる時間として定義される。もしすべてのプロセッサが必ずしも同時に起動したり終了したりしないときには、実行時間は最初のプロセッサが計算し始めてから最後のプロセッサが計算し終わるまでにかかる時間であると定義する。

2.2.2. O記法

計算時間などの計算量の評価は問題のサイズ(入力データの長さ、あるいは個数)が大きくなるにつれて、どのような変化するかで表される。その表し方がO記法である。

正の数 n の関数 $f(n)$ と $g(n)$が、すべての $n \geq n_0$ に対して、$0 \leq g(n) \leq c\, f(n)$ であるような正の定数 c, n_0が存在するならば、関数 $g(n)$は高々オーダ$(f(n))$であるといい、記号$O(f(n))$で表す。

$$g(n) = O(f(n)) \tag{1}$$

このとき、関数 $g(n)$の上界(Upper Bound)は$O(f(n))$であるという。この表記法では、上界の表現の最も高次の項のみに注目すればよく、また最高次の項にかかる定数係数を省略して表現する。このためこの表記法では、対数の底を定数と仮定すれば、定数の底の変化は対数を定数倍変化させるだけであるので底を明示する必要がない。

2.2.3. 速度向上比

サイズNの問題に対して、基地の最速逐次アルゴリズムの実行時間を T_1、 P台のプロセッサを用いた並列アルゴリズムの実行時間を Tとすると、この問題を並列化して解くことによる速度向上比 S(Speedup Ratio)は次式で表される。

$$S = \frac{T_1}{T} \tag{2}$$

複数のプロセッサを用いるのは、1 台のプロセッサより速く解けることを期待するためであるので、$S > 1$と考えてよい。明らかにSが大きければ大きいほど、その並列アルゴリズムは高速なアルゴリズムである。

2.2.4. コスト

並列アルゴリズムのコスト(Cost)を、使用プロセッサ数と並列実行時間の積で定義する。サイズ n の問題に対して、並列アルゴリズムのコストを n の関数として $C(n)$ と表すと、使用プロセッサ数 $P(n)$ と並列実行時間 $T(n)$ より、

$$C(n) = P(n) \times T(n) \tag{3}$$

ある問題を解く逐次的アルゴリズムの実行時間の下界がすでに知られていると仮定する。その問題に対する並列アルゴリズムのコストが、一定の乗数因子の範囲内でこの下界に一致するならば、その並列アルゴリズムはコスト最適(Cost Optimal)であるという。

2.3. PRAM シミュレータ

並列アルゴリズムの正方性の解析、およびその時間計算量の評価は各アルゴリズムごとに行わなければならない。そこで本研究では並列アルゴリズムの評価を実験的に行うため PRAM シミュレーションを作成した。

第 2 図に本研究で作成した PRAM シミュレータを用いた C 風並列言語プログラムの実行方法を示す。本研究で作成した PRAM シミュレータは C 風並列言語プログラムを C 言語プログラムに変換する。先ず input.txt というファイルに C 風並列言語プログラムを記述し、PRAM シミュレータと同じディレクトリに置く。その後 PRAM シミュレータを実行させると、output.c というファイルに C 言語プログラムが出力される。

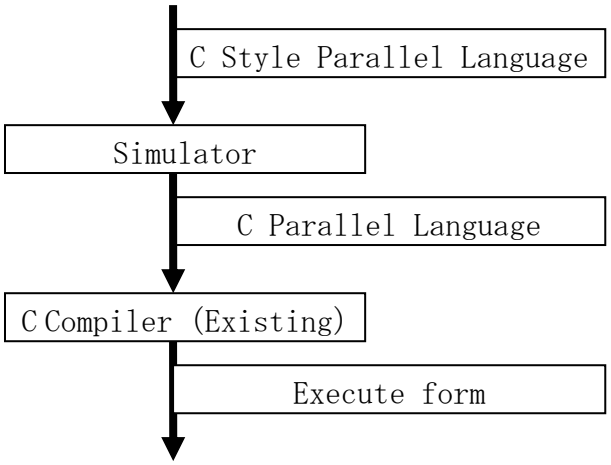

図２　PRAM シミュレータの実行の流れ
Fig.2　Date Flow By PRAM Simulator

第3章　方法

3.1.　並列言語の設計

本研究では、PRAM アルゴリズムを実行できるための並列言語を作成する。作成する言語はC言語をもとに並列計算ができるようにParallelという命令を加えた言語である。

3.1.1.　並列言語の必要性

PRAM アルゴリズムを実際の計算機あるいはシミュレータで実行させるためには、何らかの言語でそのアルゴリズムを記述しなければならない。そこで本研究ではPRAMアルゴリズムを記述できる並列言語を作成する。

3.1.2.　並列言語の仕様

本研究で作成したC風並列言語では、命令Parallelにより並列処理を記述することができる。Parallelの書式は以下の通りである。

Parallel(式1,式2)文

Parallel文と書いた場合、プロセッサ番号式1からプロセッサ番号式2までのプロセッサが後ろに続く文を並列に実行される。またこの時 int 型変数_p にはその文を実行中のプロセッサ番号が格納される。

本研究で作成したC風並列言語の文法を以下に示す。

<NAME>::=英字{英字|数字|‘_’}|“_p”
INT::= 数字 { 数字 }
STRING::= ダブルクォート { 任意の文字 } ダブルクォート
CHAR::=シングルクォート 任意の文字 シングルクォート

<Program>::=<Main_function>

<Main_function>::=“main” “(” “)” <Block>

<Block>::= “{“ { <Var_decl> } { <Statement> } “}”

<Ver_decl>::= “int” NAME[“[“ INT “]”]{ “,” NAME [“[“ INT “]”]} “;”

<Statement>::= <If_statement> | <While_statement>|<For_statement> | <Assignment>
|<Writeint_statement>|<Writechar_statement>
|<Parallel_statement>| “{ “ { <Statement> } “}” | “;”

<If_statement>::= “if” “(“<Expression> “)”
<Statement>{ “else” <Statement>}

<While_statement>::= “While” “(“<Expression> “)” <Statement>

<For_statement>::= “for” “(“ (<Forassignment>) “;” (<Expression>) “;”
“(“(<Forassign>) “)” <Statement>

<Writeint_statement>::= “writeint” “(“ <Expression> “)” “;”

<Writechar_statement>::= “writehcar” “(“<Expression> “)” “;”

<Printf_statement>::= “Printf” “(“ 文字列 { “,” <Expression> } “)” “;”

<Parallel_statement>::= “parallel” “(“ < Expression > “,” < Expression > “)” <Statement>

<Assignment>::= 变数 (“= “ <Expression> | “++” | “-- “) “;”

<Forassignment>::= 变数 (“=“ <Expression> | “++” | “--")

<Lefthand_side>::= NAME | NAME “[“ <Expression> “]”

<Expression>::= <Logical_term> { “||” <Logical_term> }

<Logical_term>::= <Logical_factor> { “&&” <Logical_factor> }

<Logical_factor>::= <Arithmtic_expression>
| <Arithmtic_expression> “== “ <Arithmtic_expression>
| <Arithmtic_expression> “!= “ <Arithmtic_expression>
| <Arithmtic_expression> “< “ <Arithmtic_expression>
| <Arithmtic_expression> “<= “ <Arithmtic_expression>

```
                | <Arithmtic_expression> “>“ <Arithmtic_expression>
                | <Arithmtic_expression> “>=“ <Arithmtic_expression>

<Arithmtic_expression>::= <Arithmtic_term> { ( “+” | “-” ) <Arithmtic_term> }

<Arithmtic_term>::=  <Arithmtic_factor>  {  ( “*”  |  “/” |  “%” )
                <Arithmtic_factor>

<Arithmtic_factor>::= <Unsigned_factor> | “!” <Arithmtic_factor>
                | “-” <Arithmtic_factor>

<Unsigned_factor>::= NAME | NAME “[“ <Expression> “]” | INT | CHAR
                | “(“ <Expression> “)” | “Readint” | “Readchar”
```

この言語では parallel の中に更に parallel を用いることができない。
また、else 節は一番近くにある、他の else 節と対応していない if 文と対応する。C 言語と異なり代入は文であり式ではない。したがって式の途中で代入文は扱えない。また++,--も同様に独立した文として扱わなければならない。

この言語では、C 言語の関数には対応していない。また変数名の先頭に_は使えない。ただし_p はプロセッサ番号を表す変数として使える。

3.2. PRAM シミュレータの設計

この章では、前章で説明した文法に従って作った C 風並列言語プログラムを C 言語プログラムに変換する PRAM シミュレータについて説明をする。

3.2.1. PRAM シミュレータの必要性

作成した C 風並列言語プログラムは、それが正しい解を出力するかどうか、またその実行時間が早いかどうかを確認する必要がある。しかし、実際に PRAM を実現できるような並列計算機を作ることは困難である。そこで PRAM の動作をシミュレートできる PRAM シミュレータが必要である。

3.2.2. PRAM シミュレータのプログラム

本研究では優先型 CRCW PRAM の動作をシミュレートできる PRAM シミュレータを作成した。

作成した PRAM シミュレータプログラムを付録に示す。この PRAM シミュレータは C 風並列言語プログラムを C 言語プログラムに変換する。また変換した際に C 風並列言語

プログラムを PRAM 上で実行させたとき、その実行時間がいくらになるかを表示する機能を付加する。

3.3. C 風並列言語の実行時間

C 風並列言語プログラムの実行時間は命令の種類に関わり無く 1 つの命令につき 1 単位時間かかると仮定している。parallel 文については parallel 文内の並列処理する文を各プロセッサが実行した実行時間のうち最も長いものを parallel 文の実行時間とする。

3.4. 実験方法

シミュレーションしたい C 風並列言語プログラムを作りそのファイル名を input.txt とし、PRAM シミュレータと同じディレクトリに置く。次に PRAM シミュレータを用いて C 風並列言語プログラムを C 言語に変換し既存の C コンパイラを用いてコンパイルし、実行する。この時実行時に出力される解が正しい事を確認する。また正しい場合はその実行時間を測定し、実行時間が理論値と一致しているかを確認する。

第4章　結果

4.1.　プログラムの実行結果

いくつかの C 風並列言語プログラムが C 言語に正しく変換され、PRAM への実行時間を計測できる事を確認した。正しい解が出力されかつその値は理論値とほぼ等しくなった。

付録　に例として使用した C 風並列言語プログラムを示す。また図３にその実行時間を示す。プログラム１「sum」は N 台のプロセッサを用い、N 個の要素の足し算を実行するプログラムであり、プログラム 2「pointer jump」は N 台のプロセッサを用い、N 頂点に対する pointer jump を計算するプログラムである。

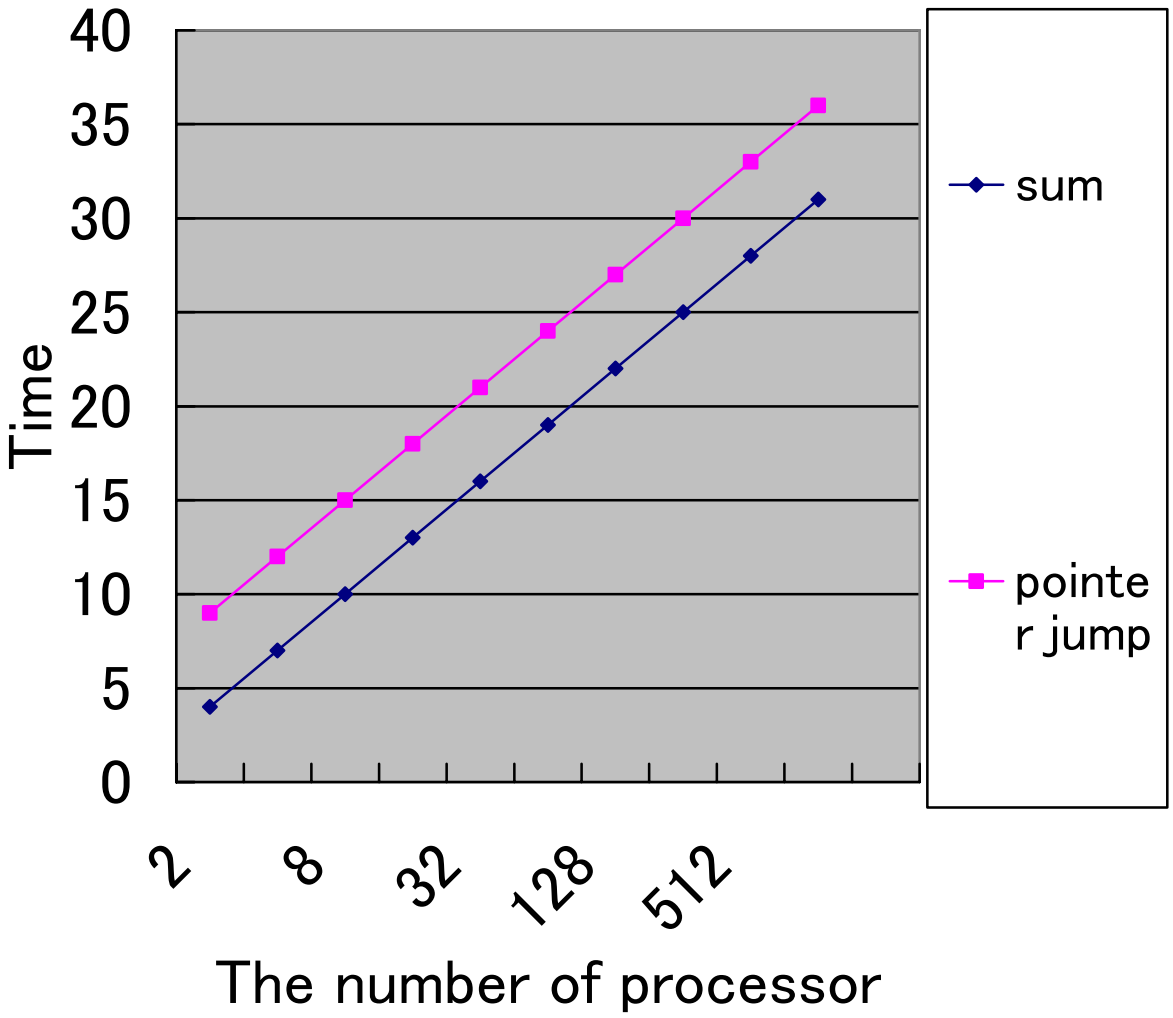

図３　PRAM シミュレータによる sum および pointer jump の実行時間

Fig.3 Running time of sum and pointer jump on PRAM simulator

第5章　考察

5.1.　実行時間の解析

データ数ｎの時、足し算、ポインタジャンプともに理論上の計算量は logｎに比例する。図３より、付録に示すＣ風並列言語プログラムは足し算、ポインタジャンプともに実行時間はデータ数の対称に比例していることが判る。

5.2.　シミュレータの操作性への考察

ファイル名は現在入力ファイル input.txt、出力ファイル output.c と決まっている。操作性の向上のために今後の課題として入出力ファイル名をオプションで自由に変えられるように改良するということが考えられる。

5.3.　並列言語の妥当性

今回作成したシミュレータには対応していない C 言語の命令や関数がある。また parallel 文中で parallel 文を使用できない。よってこれらに対応させることが今後の課題である。

第6章　結論

6.1.　結論

本研究では、PRAM アルゴリズムを記述できる C 風並列言語を作成した。また C 風並列言語を C 言語に変換する PRAM シミュレータを作成した。この PRAM シミュレータは C 風並列言語プログラムを PRAM 上で実行させたときの動作をシミュレートし、またその時の実行時間を測定できる。

また今後の課題としては、操作性を向上させる事および、対応させる C 風並列言語をより強力なものにする事等があげられる。

謝辞

研究をするにあたり、アルゴリズムの基礎から並列処理についてまで、数え切れないほどの、御指導や御助言など大変尽力していただき、石水隆助手には、誠に感謝申し上げます。

参考文献

1）渋沢　進：並列分散処理入門、培風館、東京、1～6（1998）
2）石畑　清：アルゴリズムとデータ構造、岩波書店、東京、224～225（1989）
3）石畑　清：アルゴリズムとデータ構造、岩波書店、東京、229～232（1989）
4）石畑　清：アルゴリズムとデータ構造、岩波書店、東京、244～247（1989）
5）渋沢　進：並列分散処理入門、培風館、東京、39～42,46,48（1998）

付録

```
/*=================================================================*/
/*                          PRAMシミュレータ                       */
/*                                  最終更新日：    2005/02/15  */
/*=================================================================*/
/*=================================*/
/*            INCULUD              */
/*=================================*/

#include<stdio.h>
#include<string.h>
#include<ctype.h>
#include <stdlib.h>      //exitを使用する為
#include" sotuken.h"       //token_typeを使用する為

/*=================================*/
/*             DEFINE              */
/*=================================*/

#define ID_MAX_SIZE           16
#define STR_MAX_SIZE         255
#define ID_TABLE_SIZE        255

/*=================================*/
/*       グローバル変数宣言        */
/*=================================*/

FILE *file_input, *file_output;

char currentc, nextc, str[STR_MAX_SIZE], id[ID_MAX_SIZE],
idtable[ID_TABLE_SIZE][ID_MAX_SIZE];
int line = 0, column = 0, numvar = 1, value, inParallel = 0,typetable[ID_TABLE_SIZE],
sizetable[ID_TABLE_SIZE], inassign = 0;
//str[STR_MAX_SIZE]   ストリング（文字列読んだ場合は文字列そのもの）
//id[ID_MAX_SIZE]ID   （変数名）
//idtable[ID_TABLE_SIZE][ID_MAX_SIZE]  変数名の表
```

```
//line 列、column 行 numvar 変数の個数
//value 整数であった時：その値、inParallel  並列か否か
//typetable 変数が配列か否か、sizetable変数が配列であった場合その大きさ
//inassign 代入文の右辺の中にいるかどうかを表す
//グローバル変数は、プログラム全体で共有する特別なデータだけに使います。
token_type nexttoken;
token_type token;

/*================================*/
/*       プロトタイプ関数宣言       */
/*================================*/
int main( void );
LOCAL int getToken();
LOCAL int cget( void );

LOCAL void syntaxError();

LOCAL token_type parseProgram();
LOCAL token_type parseMain();
LOCAL token_type parseBlock();
LOCAL token_type parseStatement();
LOCAL token_type parseVarDecl();
LOCAL token_type parseIf();
LOCAL token_type parseWhile();
LOCAL token_type parseFor();
LOCAL token_type parseAssignment();
LOCAL token_type parseAssignmentInFor();
LOCAL token_type parseWriteint();
LOCAL token_type parseWritechar();
LOCAL token_type parseReadint();
LOCAL token_type parseReadchar();
LOCAL token_type parsePrintf();
LOCAL token_type parseParallel();
LOCAL token_type parseExpression();
LOCAL token_type parseLogicalTerm();
LOCAL token_type parseLogicalFactor();
LOCAL token_type pareseArithmeticExpression();
```

```
LOCAL token_type pareseArithmeticTerm();
LOCAL token_type pareseArithmeticFactor();
LOCAL token_type unsignedFactor();

LOCAL void file_closed( void );

/*************************************************************************
  @関数名 : main
  @引数   : 無し
  @戻り値 : int            - OSに制御を返します
  @解説   : メイン関数
*************************************************************************/
int main( void )
{

    line=0; column=0;

    file_input = fopen( “input.txt”,  “r” );    //入力ファイル.テキストモードでオープンして1文字づつ読み込む
    file_output = fopen( “output.c” ,” w” );     //出力ファイル.テキストモードでオープンして書き込む
    nextc = ’ ’;
       parseProgram();
    file_closed();          //解放し忘れがあるかもしれないので保険
    return(0);              //OSに制御を返します(終了)
}
/*************************************************************************
  @関数名 : getToken
  @引数   : 無し
  @戻り値 : int
  @解説   : 次のTOKENを入れるという関数
*************************************************************************/
LOCAL int getToken( void )  /* 次のTOKENを入れるという関数 */
{

    int count;                      /* l は何文字の～ (ex変数) か？ */
    token = nexttoken;
```

```
    do
    {
          cget();
    }
    while(currentc == ' '||currentc == '¥t'||currentc == '¥n');
if ( currentc == EOF )
{
     nexttoken = TOKEN_EOF;       /* EOFを呼んだ場合TOKEN_EOFを返す */
}else if ( currentc >= 'a' && currentc <= 'z'
  || currentc >= 'A' && currentc <= 'Z'
  || currentc == '_')                          //_を認める
    {
     count = 0;
     id[0] = currentc;

     while ( nextc>='a' && nextc<='z'
          || nextc>='A' && nextc<='Z'
          || nextc>='0' && nextc>='0' && nextc<'9'
          || nextc =='_' )
     {
         currentc=cget();

         if (count<ID_MAX_SIZE)
         {
             count++; id[count]=currentc;
         }
     }
     id[count+1]='¥0';           /*文字列の終わりは必ず '¥0' */

          /*文字列が一致するか判定strcmp(id=“main”) */
     if (strcmp(id,” main”)==0)
     {
         nexttoken = TOKEN_MAIN;
           }else if (strcmp(id,” file”)==0){
         nexttoken = TOKEN_FILE;
     }else if (strcmp(id,” if”)==0){
         nexttoken = TOKEN_IF;
     }else if (strcmp(id,” else”)==0){
```

```
            nexttoken = TOKEN_ELSE;
        }else if (strcmp(id," int" )==0){
            nexttoken = TOKEN_INT;
              }else if (strcmp(id," char" )==0){
            nexttoken = TOKEN_CHAR;
        }else if (strcmp(id," while" )==0){
            nexttoken = TOKEN_WHILE;
              }else if (strcmp(id," for" )==0){
            nexttoken = TOKEN_FOR;
        }else if (strcmp(id," parallel" )==0){
            nexttoken = TOKEN_PARALLEL;
        }else if (strcmp(id," readint" )==0){
            nexttoken = TOKEN_READINT;
        }else if (strcmp(id," readchar" )==0){
            nexttoken = TOKEN_READCHAR;
        }else if (strcmp(id," writeint" )==0){
            nexttoken = TOKEN_WRITEINT;
        }else if (strcmp(id," writechar" )==0){
            nexttoken = TOKEN_WRITECHAR;
              }else if (strcmp(id," printf" )==0){
                    nexttoken = TOKEN_PRINTF;
        }else{
            nexttoken = TOKEN_NAME;
        }
    }else if(  currentc >= '0' && currentc <= '9' ){     /* 文字を数字に変換
currentc-'0' */
        value = currentc - '0';
        while ( nextc>='0' && nextc<='9' )
        {
            currentc=cget();
            value = value*10+currentc-'0';
        }
        nexttoken = TOKEN_INTEGER;
    }else if(  currentc == '=' ){
        if (nextc == '=')
        {
          currentc=cget();
          nexttoken=TOKEN_EQUAL;
```

```
    }else{
      nexttoken = TOKEN_ASSIGN;
    }
}else if(  currentc == '!' ){
    if (nextc == '=')
    {
      currentc=cget();
      nexttoken=TOKEN_NOTEQ;
    }else{
      nexttoken=TOKEN_NOT;
    }
}else if(  currentc == '<' ){
    if (nextc == '=')
    {
        currentc=cget();
        nexttoken=TOKEN_LESSEQ;
    }else{
        nexttoken=TOKEN_LESS;
    }
}else if(  currentc == '>' ){
    if (nextc == '=')
    {
      currentc = cget();
      nexttoken = TOKEN_GREATEQ;
    }else{
      nexttoken = TOKEN_GREAT;
    }
}else if(  currentc == '&' ){
    if (nextc == '&')
    {
        currentc = cget();
        nexttoken = TOKEN_AND;
    }else{
        syntaxError() ;
    }
}else if(  currentc == '|' ){
    if (nextc == '|')
    {
```

```
        currentc = cget();
        nexttoken = TOKEN_OR;
          }
   }else if(currentc == 0x22 ){
          count = 0;
    while (nextc != 0x22 && nextc != EOF)
          {
                currentc = cget();
                if (count<STR_MAX_SIZE)                                    {
                      str[count] = currentc;
                      count++;
                }
          }
          str[count]='¥0';
          currentc = cget();
          if(  currentc == EOF ) syntaxError();
          printf( "%s" , str);
          nexttoken = TOKEN_STRING;
}else if( currentc == '+' ){

          if (nextc == '+')
    {
                currentc=cget();
                nexttoken=TOKEN_INC;
    }else{
                nexttoken = TOKEN_ADD;
          }
}else if(  currentc == '-' ){
          if (nextc == '-')
    {
                currentc=cget();
                nexttoken=TOKEN_DEC;
    }else{
                nexttoken = TOKEN_SUB;
          }
}else if(  currentc == '*' ){
    nexttoken = TOKEN_MUL;
}else if(  currentc == '/' ){
```

```
            if (nextc == '*'){
                    currentc=cget();
                    currentc=cget();
                    while ((currentc != '*' || nextc != '/') && currentc !=
EOF)currentc=cget();
                    if(currentc == EOF)syntaxError( "coment" );
                    currentc=cget();
                    nexttoken = getToken();
            }else{
                    nexttoken = TOKEN_DIV;
            }
    }else if(  currentc == '%' ){
        nexttoken = TOKEN_MOD;
    }else if(  currentc == ';' ){
        nexttoken = TOKEN_SEMICOLON;
    }else if(  currentc == '(' ){
        nexttoken = TOKEN_LPAREN;
    }else if(  currentc == ')' ){
        nexttoken = TOKEN_RPAREN;
    }else if(  currentc == '{' ){
        nexttoken=TOKEN_LBRACE;
    }else if(  currentc == '}' ){
        nexttoken = TOKEN_RBRACE;
    }else if(  currentc == '[' ){
        nexttoken = TOKEN_LBLACKET;
    }else if(  currentc == ']' ){
        nexttoken=TOKEN_RBLACKET;
    }else if(  currentc == ',' ){
        nexttoken=TOKEN_COMMA;
    }else if(  currentc == '¥'' ){
        if (nextc != '¥'')
        {
            currentc=cget();
            value=(int)currentc;
            if (nextc == '¥'')
            {
                currentc = cget();
                nexttoken=TOKEN_CHARACTER;
```

```
            }else{
                syntaxError();
            }
        }else{
            syntaxError();
        }
    }else{
        syntaxError() ;
    }
       token = nexttoken;
       printf( “%d  “,token);
    return(nexttoken);
}

/*************************************************************************
  @関数名 : cget
  @引数   : 無し
  @戻り値 : int            - OSに制御を返します
  @解説   : 次の文字を1文字読み出してついでに１文字先読みしておく。
                    行、文字判別。
*************************************************************************/
LOCAL int cget( void )
{
    currentc = nextc;
    nextc = getc( file_input );           //  cgetによって1文字先読み

    if (currentc == ’¥n’)                 //  何行、何文字目かを判別
    {
        line++;
        column = 0;
    }
    else
    {
        column++;
    }
printf( “%c” ,currentc);
    return currentc;
}
```

```
/*************************************************************************
  @関数名 : syntaxError
  @引数   : s
  @戻り値 : 無し              - OSに制御を返します
  @解説   : 文法エラー
*************************************************************************/
LOCAL void syntaxError(s)char *s;
{
        printf(“%s”,s);
    printf(“%d行目、%d文字目%cでエラーが出ました。¥n”, line, column, currentc);
    exit(0);
}

/*★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★*/
/*★                        構文解析                                    ★*/
/*★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★*/

/*************************************************************************
  @関数名 : parseProgram
  @引数   : 無し
  @戻り値 : token_type            -
  @解説   : プログラム解析
*************************************************************************/
LOCAL token_type parseProgram( void )
{
        fprintf(file_output,” #include<stdio.h>¥n” );
        fprintf(file_output,” #include<string.h>¥n” );
        fprintf(file_output,” #include<ctype.h>¥n” );
        fprintf(file_output,” #include<stdlib.h>¥n” );

        fprintf(file_output,” #define _P 1024¥n” );

        fprintf(file_output,” int _i, _t=0, _p=0, _u[_P], _p1, _p2, _max(),
        _readint();¥n” );

        getToken();
        if (token != TOKEN_MAIN)syntaxError( “parseProgram” );
```

```
        parseMain();

        fprintf(file_output,” int _max(_p1, _p2) int _p1,  _p2;{¥n” );
        fprintf(file_output,” int _i;¥n” );
        fprintf(file_output,” int _max = _u[_p1];¥n” );
        fprintf(file_output,” for(_i = _p1+1;_i <=_p2;_i++)¥n” );
        fprintf(file_output,” if(_max <= _u[_i])¥n” );
        fprintf(file_output,” _max = _u[_i];¥n” );
        fprintf(file_output,” return(_max);¥n” );
        fprintf(file_output,” }¥n” );

        fprintf(file_output,” int _readint() {¥n” );
        fprintf(file_output,” int _i;¥n” );
        fprintf(file_output,” scanf(%c%cd%c,%c_i);¥n” ,0x22,0x25,0x22,0x26);
        fprintf(file_output,” return _i;¥n” );
        fprintf(file_output,” }¥n” );

        if(token != TOKEN_EOF)   syntaxError(“parseProgram”);
        printf(“コンパイル終了¥n”);
        return(token);
}

/*************************************************************************
  @関数名 : parseMain
  @引数   : 無し
  @戻り値 : token_type            - OSに制御を返します
  @解説   : main関数の解析
*************************************************************************/
LOCAL token_type parseMain( void )
{


  if ( token != TOKEN_MAIN ) syntaxError(“parseMain”);
  fprintf(file_output,” main”);
  getToken();
  if ( token != TOKEN_LPAREN )  syntaxError(“parseMain”);
  fprintf(file_output,” ( “);
```

```
  getToken();
  if ( token != TOKEN_RPAREN )  syntaxError( “parseMain” );
  fprintf(file_output,” )” );
  getToken();
  if ( token != TOKEN_LBRACE )  syntaxError( “parseMain” );
  fprintf(file_output,” {¥n” );
  getToken();

  parseBlock();
  return( token );
}


/*************************************************************************
  @関数名 : token_type parseBlock(
  @引数   : 無し
  @戻り値 : token_type
  @解説   : ブロック解析
*************************************************************************/
LOCAL token_type parseBlock( void )
{
       strcpy(idtable[0],” _p” );  //_pは何も宣言しなくても変数として使える。
       typetable[0] = 0;           //_pは宣言しなくてもint型である。

  if( token != TOKEN_RBRACE )
  {
      while( token == TOKEN_INT ||token == TOKEN_CHAR )
      {
                if (token == TOKEN_INT)
                     fprintf(file_output,” int  “);
                else fprintf(file_output,” char  “);
                     getToken();
                     parseVarDecl();
      }
      while( token != TOKEN_RBRACE )
      {
          parseStatement();
      }
```

```
  }
  fprintf(file_output,” printf(¥”実行時間は¥%cdです。¥”,_t);”,0x25);
  fprintf(file_output,” }¥n” );
  getToken();
  return( token );
}

/*************************************************************************
  @関数名 : parseVarDecl
  @引数   : 無し
  @戻り値 : token_type
  @解説   : バーデクル{バー(変数)デクル(宣言)}の解析
  typetable[ID_TABLE_SIZE], sizetable[ID_TABLESIZE];
*************************************************************************/
LOCAL token_type parseVarDecl( void )
{
       int i;

  if( token != TOKEN_NAME )syntaxError( “parseVarDecl” );
       /*この次に変数の重複してないかチェックが必要*/
  for(i = 0; i < numvar; i++)if (strcmp(idtable[i],id) == 0)break;    //初期
値設定
  if (i != numvar)syntaxError( “parseVarDecl” );
  strcpy(idtable[numvar],id);                                          //変数
がidtableに入っていく。そのたびnumvarが増える
  fprintf(file_output,” %s”, id);               //ここでidを出す。
  getToken();
  if( token == TOKEN_LBLACKET )
  {
       fprintf(file_output,” [ “);
       getToken();
    if( token != TOKEN_INTEGER ) syntaxError( “parseVarDecl” );
       fprintf(file_output,” %d”,value);
       getToken();
    if( token != TOKEN_RBLACKET )  syntaxError( “parseVarDecl” );
       fprintf(file_output,” ]” );
       fprintf(file_output,” ,__%s[%d]”,id, value);
       getToken();
```

```
        typetable[numvar] = 1;                          //配列であれば１
        sizetable[numvar] = value;
}
else
                {
                fprintf(file_output,” =0,__%s” ,id);
                typetable[numvar] = 0;
                }
        numvar++;
while( token == TOKEN_COMMA)
{
          fprintf(file_output,” ,” );
          getToken();
  if( token != TOKEN_NAME )  syntaxError( “parseVarDecl” );
     for(i = 0; i < numvar; i++)if (strcmp(idtable[i],id) == 0)break;
     if (i != numvar)syntaxError( “parseVarDecl” );
     strcpy(idtable[numvar],id);
     fprintf(file_output,” %s” , id);
     getToken();   /*この次に変数の重複してないかチェックが必要*/
  if( token == TOKEN_LBLACKET )
  {
                fprintf(file_output,” [ “);
                getToken();
      if( token != TOKEN_INTEGER )  syntaxError( “parseVarDecl” );
                fprintf(file_output,” %d” ,value);
                getToken();
      if( token != TOKEN_RBLACKET )  syntaxError( “parseVarDecl” );
                fprintf(file_output,” ]” );
                fprintf(file_output,” ,__%s[%d]” ,id, value);
     //変数コピー用（コンピュータ数台使うので）
                getToken();
                typetable[numvar] = 1;                  //配列であれば１
                sizetable[numvar] = value;
  }
     else
     {
                fprintf(file_output,” =0,__%s” ,id);
                typetable[numvar] = 0;
```

```
        }
numvar++;
  }
  if(token != TOKEN_SEMICOLON)syntaxError( “parseVarDecl” );
  fprintf(file_output,” ;¥n” );
  getToken();
  return( token );
}

/*************************************************************************
  @関数名 : parseStatement
  @引数   : 無し
  @戻り値 : token_type
  @解説   : ステイトメント（文）の解析
*************************************************************************/
LOCAL token_type parseStatement( void )
{

  switch ( token )
  {
      case TOKEN_IF:
          parseIf();
          break;
      case TOKEN_WHILE:
          parseWhile();
          break;
      case TOKEN_FOR:
          parseFor();
          break;
      case TOKEN_NAME:
          parseAssignment();
          break;
      case TOKEN_WRITEINT:
          parseWriteint();
          break;
      case TOKEN_WRITECHAR:
          parseWritechar();
          break;
```

```
        case TOKEN_READINT:
         parseReadint();
         break;
        case TOKEN_READCHAR:
         parseReadchar();
         break;
        case TOKEN_PRINTF:
         parsePrintf();
         break;
      case TOKEN_PARALLEL:
         parseParallel();
         break;
        case TOKEN_LBRACE:
               fprintf(file_output,"{¥n");
               getToken();
               while(  token != TOKEN_RBRACE )
               {
                        parseStatement();
               }
               fprintf(file_output,"}¥n");
               getToken();
               break;
        case TOKEN_SEMICOLON:
               fprintf(file_output,";¥n");
               getToken();
         break;
        default:
               syntaxError("parseStatement");
               break;
  }
  return(token);
}

/*************************************************************************
  @関数名 : parseIf
  @引数   : 無し
  @戻り値 : token_type
  @解説   : if文の解析
```

```
**********************************************************************/

LOCAL token_type parseIf(void)
{

        if(token != TOKEN_IF) syntaxError( “parseIf” );
        fprintf(file_output,” {¥n if” );
        getToken();
        if(token != TOKEN_LPAREN) syntaxError( “parseIf” );
        fprintf(file_output,” ( “);
        getToken();
        parseExpression();
    if(token != TOKEN_RPAREN) syntaxError( “parseIf” );
        fprintf(file_output,” )” );
        getToken();
        parseStatement();
        if(token == TOKEN_ELSE)
        {
                fprintf(file_output,” else” );
                getToken();
                parseStatement();
        }
        if(inParallel == 1)fprintf(file_output,” _u[_p]++;}¥n” );
        else fprintf(file_output,” _t++;¥n }¥n” );
        return(token);
}

/**********************************************************************
  @関数名 : parseWhile
  @引数   : 無し
  @戻り値 : token_type
  @解説   : while文の解析
**********************************************************************/

LOCAL token_type parseWhile(void)
{

        if(token != TOKEN_WHILE) syntaxError( “parseWhile” );
```

```
        fprintf(file_output,” while”);
        getToken();
        if(token != TOKEN_LPAREN) syntaxError(“parseWhile”);
        fprintf(file_output,” (“);
        getToken();
        parseExpression();
    if(token != TOKEN_RPAREN) syntaxError(“parseWhile”);
        fprintf(file_output,”){¥n”);
        getToken();
        if(inParallel == 1)fprintf(file_output,” ;¥n_u[_p]++;¥n”);
        else fprintf(file_output,” _t++;¥n”);
        parseStatement();
        fprintf(file_output,”}¥n”);
        return(token);
}

/*************************************************************************
  @関数名 : parseFor
  @引数   : 無し
  @戻り値 : token_type
  @解説   : for文の解析
*************************************************************************/

LOCAL token_type parseFor(void)
{

        if(token != TOKEN_FOR)syntaxError(“parseFor”);
        fprintf(file_output,” for”);
        getToken();
        if(token != TOKEN_LPAREN)syntaxError(“parseFor”);
        fprintf(file_output,” (“);
        getToken();
        if(token == TOKEN_NAME)parseAssignmentInFor();
        if(token != TOKEN_SEMICOLON)syntaxError(“parseFor”);
        fprintf(file_output,” ;”);
        getToken();
        if(token != TOKEN_SEMICOLON)parseExpression();
        if(token != TOKEN_SEMICOLON)syntaxError(“parseFor”);
```

```
        fprintf(file_output,” ;” );
        getToken();
        if(token == TOKEN_NAME)parseAssignmentInFor();
    if(token != TOKEN_RPAREN)syntaxError( “parseFor” );
        fprintf(file_output,” ){¥n” );
        getToken();
        if(inParallel == 1)fprintf(file_output,” _u[_p]++;¥n” );
        else fprintf(file_output,” _t=_t+2;¥n” );
        parseStatement();
        fprintf(file_output,” }¥n” );
        return(token);
}

/*************************************************************************
  @関数名 : parseAssignment
  @引数   : 無し
  @戻り値 : token_type
  @解説   : 代入文の解析                    配列を使えるようにする。
*************************************************************************/

LOCAL token_type parseAssignment(void)

{
        int i;

        if(token != TOKEN_NAME)syntaxError( “parseAssignment” );
        for(i = 0; i < numvar; i++)if (strcmp(idtable[i],id) == 0)break;
        if (i == numvar)syntaxError( “parseAssignment” );
        fprintf(file_output,” %s” , id);
        getToken();
        if(token == TOKEN_LBLACKET)

        {

                fprintf(file_output,” [ “);
                getToken();
                parseExpression();
                if(token != TOKEN_RBLACKET) syntaxError( “parseAssignment” );
```

```
            fprintf(file_output,” ]” );
            getToken();
        }
        if(token == TOKEN_ASSIGN){
        fprintf(file_output,” = “);
        getToken();
        inassign = 1;
        parseExpression();                          //右辺
        inassign = 0;
        }
        else if(token == TOKEN_INC){
            fprintf(file_output,” ++” );
            getToken();
        }
        else if(token == TOKEN_DEC){
            fprintf(file_output,” --” );
            getToken();
        }
        else syntaxError( “parseAssignment” );
        if(token != TOKEN_SEMICOLON) syntaxError( “parseAssignment” );
        if(inParallel == 1)fprintf(file_output,” ;¥n_u[_p]++;¥n” );
        else fprintf(file_output,” ;¥n_t++;¥n” );
        getToken();
        return(token);
}


/*************************************************************************
  @関数名 : parseAssignmentInFor
  @引数   : 無し
  @戻り値 : token_type
  @解説   : 代入文の解析              for文専用
*************************************************************************/

LOCAL token_type parseAssignmentInFor(void)

{
        int i;
```

```
        if(token != TOKEN_NAME) syntaxError();
        for(i = 0; i < numvar; i++)if (strcmp(idtable[i],id) == 0)break;
        if (i == numvar)syntaxError( “parseAssignmentInFor” );
        fprintf(file_output,” %s” , id);
        getToken();
        if(token == TOKEN_LBLACKET)

        {

                fprintf(file_output,” [ “);
                getToken();
                parseExpression();
                if(token != TOKEN_RBLACKET) syntaxError( “parseAssignmentFor” );
                fprintf(file_output,” ]” );
                getToken();
        }
        if(token == TOKEN_ASSIGN){
        fprintf(file_output,” = “);
        getToken();
        parseExpression();
        }
        else if(token == TOKEN_INC){
                fprintf(file_output,” ++” );
                getToken();
        }
        else if(token == TOKEN_DEC){
                fprintf(file_output,” --” );
                getToken();
        }
        else syntaxError( “parseAssignment” );
        return(token);
}
/*************************************************************************
  @関数名 : parseWriteint
  @引数   : 無し
  @戻り値 : token_type
  @解説   : writeintの解析
```

```
*********************************************************************/

LOCAL token_type parseWriteint(void)
{

        if(token != TOKEN_WRITEINT) syntaxError( “parseWriteint” );
        fprintf(file_output,” printf” );
        getToken();
        if(token != TOKEN_LPAREN) syntaxError( “parseWriteint” );
        fprintf(file_output,” (¥” %cd¥” ,” , 0x25);
        getToken();
        parseExpression();
    if(token != TOKEN_RPAREN) syntaxError( “parseWriteint” );
        fprintf(file_output,” )” );
        getToken();
        if(token != TOKEN_SEMICOLON) syntaxError( “parseWriteint” );
        if(inParallel == 1)fprintf(file_output,” ;¥n_u[_p]++;¥n” );
        else fprintf(file_output,” ;¥n_t++;¥n” );
        getToken();
        return(token);
}

/*********************************************************************
  @関数名 : parseWritechar
  @引数   : 無し
  @戻り値 : token_type
  @解説   : Writecharの解析
*********************************************************************/

LOCAL token_type parseWritechar(void)
{

        if(token != TOKEN_WRITECHAR) syntaxError( “parseWritechar” );
        fprintf(file_output,” printf” );
        getToken();
        if(token != TOKEN_LPAREN) syntaxError( “parseWritechar” );
        fprintf(file_output,” (¥” %cc¥” ,” , 0x25);
        getToken();
```

```
        parseExpression();
    if(token != TOKEN_RPAREN) syntaxError(“parseWritechar”);
        fprintf(file_output,” )” );
        getToken();
        if(token != TOKEN_SEMICOLON) syntaxError(“parseWritechar”);
        if(inParallel == 1)fprintf(file_output,” ;¥n_u[_p]++;¥n” );
        else fprintf(file_output,” ;¥n_t++;¥n” );
        getToken();
        return(token);
}




/*************************************************************************
  @関数名 : parseReadint
  @引数   : 無し
  @戻り値 : token_type
  @解説   : Readintの解析 独立した文です（この後に式はだめ）
*************************************************************************/

LOCAL token_type parseReadint(void)
{
        int i;

        if(token != TOKEN_READINT) syntaxError(“parseReadint”);
        fprintf(file_output,” scanf”);
        getToken();
        if(token != TOKEN_LPAREN) syntaxError(“parseReadint”);
        fprintf(file_output,” (¥” %cd¥” ,” , 0x25);

        getToken();

        if(token != TOKEN_NAME) syntaxError(“parseReadint”);
        /*parseLefthandsideの代わり*/
        for(i = 0; i < numvar; i++)if (strcmp(idtable[i],id) == 0)break;
        if (i == numvar)syntaxError(“parseReadint”);
        fprintf(file_output,” &%s” , id);
```

```
        getToken();
        if(token == TOKEN_LBLACKET)

        {

                fprintf(file_output," [ ");
                getToken();
                parseExpression();
                if(token != TOKEN_RBLACKET) syntaxError( "parseReadint" );
                fprintf(file_output," ]" );
                getToken();
        }

    if(token != TOKEN_RPAREN) syntaxError( "parseReadint" );
        fprintf(file_output," )" );
        getToken();
        if(token != TOKEN_SEMICOLON) syntaxError( "parseReadint" );
        if(inParallel == 1)fprintf(file_output," ;¥n_u[_p]++;¥n" );
        else fprintf(file_output," ;¥n_t++;¥n" );
        getToken();
        return(token);
}

/*************************************************************************
  @関数名 : parseReadchar
  @引数   : 無し
  @戻り値 : token_type
  @解説   : Readcharの解析
*************************************************************************/

LOCAL token_type parseReadchar(void)
{
        int i;

        if(token != TOKEN_READCHAR) syntaxError( "parseReadchar" );
        fprintf(file_output," scanf" );
        getToken();
        if(token != TOKEN_LPAREN) syntaxError( "parseReadchar" );
```

```
        fprintf(file_output,” (¥” %cc¥” ,” , 0x25);
        getToken();

        if(token != TOKEN_NAME) syntaxError( “parseReadchar” );
        /*parseLefthandsideの代わり*/
        for(i = 0; i < numvar; i++)if (strcmp(idtable[i],id) == 0)break;
        if (i == numvar)syntaxError( “parseReadchar” );
        fprintf(file_output,” &%s” , id);
        getToken();
        if(token == TOKEN_LBLACKET)

        {

                fprintf(file_output,” [ “);
                getToken();
                parseExpression();
                if(token != TOKEN_RBLACKET) syntaxError( “parseReadchar” );
                fprintf(file_output,” ]” );
                getToken();
        }

    if(token != TOKEN_RPAREN) syntaxError( “parseReadchar” );
        fprintf(file_output,” )” );
        getToken();
        if(token != TOKEN_SEMICOLON) syntaxError( “parseReadchar” );
        if(inParallel == 1)fprintf(file_output,” ;¥n_u[_p]++;¥n” );
        else fprintf(file_output,” ;¥n_t++;¥n” );
        getToken();
        return(token);
}
/*************************************************************************
  @関数名 : parsePrintf
  @引数   : 無し
  @戻り値 : token_type
  @解説   : Printfの解析
*************************************************************************/

token_type parsePrintf(void)
```

```
{

        if(token != TOKEN_PRINTF) syntaxError( “parsePrintf” );
        fprintf(file_output,” printf” );
        getToken();
        if(token != TOKEN_LPAREN) syntaxError( “parsePrintf” );
        fprintf(file_output,” ( “);
        getToken();
        if(token != TOKEN_STRING) syntaxError( “parsePrintf” );

        fprintf(file_output,” ¥” %s¥” “,str);//ストリング

        getToken();
        while(token == TOKEN_COMMA)
        {
                fprintf(file_output,” ,” );
                getToken();
                parseExpression();
        }
    if(token != TOKEN_RPAREN) syntaxError( “parsePrintf” );
        fprintf(file_output,” )” );
        getToken();
        if(token != TOKEN_SEMICOLON) syntaxError( “parsePrintf” );
        if(inParallel == 1)fprintf(file_output,” ;¥n_u[_p]++;¥n” );
        else fprintf(file_output,” ;¥n_t++;¥n” );
        getToken();
        return(token);
}

/*************************************************************************
  @関数名 : parseParallel
  @引数   : 無し
  @戻り値 : token_type
  @解説   : パラレル関数の解析
              parallel(式１、式２)
              仕様書にプロセッサ番号（変数）。
              プロセッサ番号式１からプロセッサ番号式２までを実行しなさい。
```

```
                ・パラレルが始まった時点で全ての変数をコピーする。
                ・パラレルの中での代入文での右辺（？）の変数は、全てコピーされた
                変数にする。
**********************************************************************/

token_type parseParallel(void)
{
        int i;

        if(token != TOKEN_PARALLEL) syntaxError( “parseParallel” );

        for(i = 1; i < numvar ; i++)        //copyvar     i=1というのは_pはコピ
ーしないから
        if(typetable[i])                    //typetableが０の場合
        {
                fprintf(file_output,” for(_i = 0; _i < %d ;
_i++)¥n” ,sizetable[i]);    //配列の場合
                fprintf(file_output,” __%s[_i] = %s[_i];¥n” , idtable[i],
idtable[i]);
        }
        else fprintf(file_output,” __%s = %s;¥n” , idtable[i], idtable[i]);
        //これは配列ではない場合

        fprintf(file_output,” for” );
        if(inParallel == 1) syntaxError( “parseParallel” );
        inParallel = 1;
        getToken();
        if(token != TOKEN_LPAREN) syntaxError( “parseParallel” );
        fprintf(file_output,” (_p = (_p1 = “);
        getToken();
        parseExpression();                                  /*変数*/
        if(token != TOKEN_COMMA) syntaxError( “parseParallel” );
        fprintf(file_output,” );_p <= (_p2 = “);
        getToken();
        parseExpression();
        fprintf(file_output,” );_p ++” );
    if(token != TOKEN_RPAREN) syntaxError( “parseParallel” );
        fprintf(file_output,” ){¥n” );
```

```
        getToken();
        fprintf(file_output,” _u[_p]=0;¥n” );
        parseStatement();                          //変数を書き換える
        fprintf(file_output,” }¥n” );
        fprintf(file_output,” _t=_t+_max(_p1, _p2);¥n” );

        inParallel = 0;
        return(token);
}


/*************************************************************************
  @関数名 : parseExpression
  @引数   : 無し
  @戻り値 : token_type
  @解説   : ＯＲの解析
*************************************************************************/

LOCAL token_type parseExpression(void)
{
        parseLogicalTerm();
        while(token==TOKEN_OR)
        {
                fprintf(file_output,” |” );
                getToken();
                parseLogicalTerm();
        }
        return(token);
}

/*************************************************************************
  @関数名 : parseLogicalTerm
  @引数   : 無し
  @戻り値 : token_type
  @解説   : ＡＮＤの解析
*************************************************************************/

LOCAL token_type parseLogicalTerm(void)
```

```
{
        parseLogicalFactor();
        while(token==TOKEN_AND)
        {
                fprintf(file_output,” &” );
                getToken();
                parseLogicalFactor();
        }
        return(token);
}

/*************************************************************************
  @関数名 : pareseLogicalFactor
  @引数   : 無し
  @戻り値 : token_type
  @解説   : ＝、＞、＞＝、＜、＜＝、の解析
*************************************************************************/

LOCAL token_type parseLogicalFactor(void)
{
        pareseArithmeticExpression();
        if (token==TOKEN_EQUAL||token==TOKEN_NOTEQ||
                token==TOKEN_LESS||token==TOKEN_GREAT||
                token==TOKEN_LESSEQ||token==TOKEN_GREATEQ)
        {
                switch( token )
                {
                case TOKEN_EQUAL:
                        fprintf(file_output,” == “);
                        break;
                case TOKEN_NOTEQ:
                        fprintf(file_output,” != “);
                        break;
                case TOKEN_LESS:
                        fprintf(file_output,” < “);
                        break;
                case TOKEN_GREAT:
                        fprintf(file_output,” > “);
```

```
                break;
        case TOKEN_LESSEQ:
                fprintf(file_output,” <= “);
                break;
        case TOKEN_GREATEQ:
                fprintf(file_output,” >= “);
                break;
        }
        getToken();
        pareseArithmeticExpression();
    }
    return(token);
}

/*************************************************************************
  @関数名 : pareseArithmeticExpression
  @引数   : 無し
  @戻り値 : token_type
  @解説   : ＋、－の解析
*************************************************************************/

LOCAL token_type pareseArithmeticExpression(void)
{
    pareseArithmeticTerm();
    while(token==TOKEN_ADD||token==TOKEN_SUB)
    {
        if(token==TOKEN_ADD)fprintf(file_output,” +” );
        else fprintf(file_output,” -” );
        getToken();
        pareseArithmeticTerm();
    }
    return(token);
}

/*************************************************************************
  @関数名 : pareseArithmeticTerm
  @引数   : 無し
  @戻り値 : token_type
```

```
  @解説   : ＊、/の解析
**************************************************************************/

LOCAL token_type pareseArithmeticTerm(void)
{
        pareseArithmeticFactor();
        while(token==TOKEN_MUL||token==TOKEN_DIV)
        {
                if(token==TOKEN_MUL)fprintf(file_output," *" );
                else fprintf(file_output," /" );
                getToken();
                pareseArithmeticFactor();
        }
        return(token);
}

/**************************************************************************
  @関数名 : pareseArithmeticFactor
  @引数   : 無し
  @戻り値 : token_type
  @解説   : ！、-  の解析
**************************************************************************/

LOCAL token_type pareseArithmeticFactor(void)
{
        while (token==TOKEN_NOT||token==TOKEN_SUB)
        {
                if(token==TOKEN_NOT)fprintf(file_output," !" );
                else fprintf(file_output," -" );
                getToken();
        }
        unsignedFactor();
        return(token);
}

/**************************************************************************
  @関数名 : unsignedFactor
  @引数   : 無し
```

```
  @戻り値 : token_type
  @解説   : 変数、整数、文字列、READINT、READCHAR、} その他の解析
*****************************************************************/

LOCAL token_type unsignedFactor(void)
{
       int i;

       switch ( token )
       {
      case TOKEN_NAME:
             for(i = 0; i < numvar; i++)if (strcmp(idtable[i],id) == 0)break;
             if (i == numvar)syntaxError( "unsignedFactor" );
             if(inassign && inParallel && (strcmp( "_p" ,id) !=
0))fprintf(file_output," __%s" , id);     //inassignかつinpalallelであれば、_s
のかわりに__sをつかう
             else fprintf(file_output," %s" , id);
             getToken();
             if(token == TOKEN_LBLACKET)
             {
                    fprintf(file_output," [ ");
                    getToken();
                    parseExpression();
                    if(token != TOKEN_RBLACKET)
syntaxError( "unsignedFactor" );
                    fprintf(file_output," ]" );
                    getToken();
             }
      break;
      case TOKEN_INTEGER:
               fprintf(file_output," %d" ,value );
          getToken();
          break;
      case TOKEN_CHARACTER:
               fprintf(file_output," ¥'%c¥' ",value );
          getToken();
          break;
        case TOKEN_LPAREN:
```

```
                fprintf(file_output,” ( “);
                getToken();
                parseExpression();
                if(token!=TOKEN_RPAREN)syntaxError();
                fprintf(file_output,” )” );
                getToken();
            break;

          case TOKEN_READINT:
                fprintf(file_output,” _readint();” );
                 getToken();
                 break;
          case TOKEN_READCHAR:
                fprintf(file_output,” getchar();” );
                 getToken();
                 break;
          default:syntaxError( “unsignedFactor” );
       break;
  }
       return(token);
}

/*************************************************************************
  @関数名 : file_closed
  @引数   : 無し
  @戻り値 : 無し
  @解説   : オープンしたファイルを閉じます.
*************************************************************************/


LOCAL void file_closed( void )
{
    if(file_input)
    {
        fclose(file_input);
    }

    if(file_output)
```

```
    {
        fclose(file_output);
    }
}
```

```
/*==========================================================================*/
/*                          PRAMシミュレータのヘッダー                      */
/*                                                    2005/02/15            */
/*--------------------------------------------------------------------------*/
/*          更新履歴:                                                       */
/*                                2004/11/24    define追加                  */
/*                                2004/12/03  define追加                    */
/*==========================================================================*/
/*==================================*/
/*             INCULUD              */
/*==================================*/
/*==================================*/
/*             DEFINE               */
/*==================================*/
#define LOCAL           static   //staticこのファイル以外で使用しない
/*==================================*/
/*             型宣言               */
/*==================================*/
typedef enum {
    /* 追加はこれより↓ */
    TOKEN_NULL,             //  0
    TOKEN_MAIN,             //     1
    TOKEN_FILE,             //     2
    TOKEN_IF,               //     3
    TOKEN_ELSE,             //     4
    TOKEN_INT,              //  5
    TOKEN_WHILE,            //     6
    TOKEN_FOR,              //     7
    TOKEN_NAME,             //     8
    TOKEN_READINT,          // 9   a=readintと書くとキーボードから整数 を一個
```

```
                                取ってaに入れます
TOKEN_READCHAR,         // 10   a=readcharと書くとキーボードから文 字を一
                                文字呼んでaに入れます
TOKEN_WRITEINT,         // 11   writeint(a)と書くと画面に整       数を１つ
                                出力します
TOKEN_WRITECHAR,        // 12   writechar(a)と書くと画面に文字を１ つ出力
                                します（0~255まで）
TOKEN_PRINTF,           // 13
TOKEN_STRING,           // 14
TOKEN_EQUAL,            // 15   ==
TOKEN_NOTEQ,            // 16   !=
TOKEN_LESS,             // 17   <
TOKEN_GREAT,            // 18   >
TOKEN_LESSEQ,           // 19   <=
TOKEN_GREATEQ,          // 20   >=
TOKEN_AND,              // 21
TOKEN_OR,               // 22
TOKEN_NOT,              // 23   !
TOKEN_ADD,              // 24   +
TOKEN_SUB,              // 25   -
TOKEN_MUL,              // 26   *
TOKEN_DIV,              // 27   /
TOKEN_MOD,              // 28   %
TOKEN_ASSIGN,           // 29   ＝（代入）
TOKEN_SEMICOLON,        // 30
TOKEN_LPAREN,           // 31   (
TOKEN_RPAREN,           // 32   )
TOKEN_LBRACE,           // 33   {
TOKEN_RBRACE,           // 34   }
TOKEN_LBLACKET,         // 35   [
TOKEN_RBLACKET,         // 36   ]
TOKEN_COMMA,            // 37
TOKEN_INTEGER,          // 38   整数
TOKEN_CHARACTER,        // 39   文字（’１文字’）
TOKEN_PARALLEL,         // 40   パラレルという命令です（並列）
TOKEN_CHAR,             // 41
TOKEN_INC,              // 42   ++
TOKEN_DEC,              // 43   --
```

```
    TOKEN_EOF,                // 44

    /* ここから下には追加しない！！ */
    TOKEN_MAX,
} token_type;



/*=========================================*/
/*          付録1-1 sum プログラム          */
/*          C風言語（C言語変換前）          */
/*=========================================*/

main()
{
int a[1024], i, n;

n=readint;
for(i=0; i<n; i++)a[i]=1;
for(i=1; i<n; i=i*2)parallel(0,n/(2*i)){
a[_p]=a[2*_p]+a[2*_p+1];
}
writeint(a[0]);
}
/*=========================================*/
/*          付録1-2 sum プログラム          */
/*              C言語（変換後）             */
/*=========================================*/

#include<stdio.h>
#include<string.h>
#include<ctype.h>
#include<stdlib.h>
#define _P 1024
int _i, _t=0, _p=0, _u[_P], _p1, _p2, _max(), _readint();
main(){
int a[1024],__a[1024],i=0,__i,n=0,__n;
n=_readint();;
```

```
_t++;
for(i=0;i<n;i++){
_t=_t+2;
a[i]=1;
_t++;
}
for(i=1;i<n;i=i*2){
_t=_t+2;
for(_i = 0; _i < 1024 ; _i++)
__a[_i] = a[_i];
__i = i;
__n = n;
for(_p = (_p1 =0);_p <= (_p2 =n/(2*i));_p ++){
_u[_p]=0;
{
a[_p]=__a[2*_p]+__a[2*_p+1];
_u[_p]++;
}
}
_t=_t+_max(_p1, _p2);
}
printf( “%d” ,a[0]);
_t++;
printf( “実行時間は%dです。” ,_t);}
int _max(_p1, _p2) int _p1,  _p2;{
int _i;
int _max = _u[_p1];
for(_i = _p1+1;_i <=_p2;_i++)
if(_max <= _u[_i])
_max = _u[_i];
return _max;
}
int _readint() {
int _i;
scanf( “%d” ,&_i);
return _i;
}
```

```
/*=========================================*/
/*       付録2-1  pointer jump プログラム    */
/*           C風言語 (C言語変換前)           */
/*=========================================*/

main()
{
int p[1024], i, n;
n=readint;

p[0]=0;

parallel(1,n-1)p[_p]=_p-1;

for(i=1; i<=n; i=i*2)parallel(0,n-1)p[_p]=p[p[_p]];

parallel(0,n-1)writeint(p[_p]);
}}

/*=========================================*/
/*       付録2-2  pointer jump プログラム    */
/*               C言語 (変換後)              */
/*=========================================*/

#include<stdio.h>
#include<string.h>
#include<ctype.h>
#include<stdlib.h>
#define _P 1024
int _i, _t=0, _p=0, _u[_P], _p1, _p2, _max(), _readint();
main(){
int p[1024],__p[1024],i=0,__i,n=0,__n;
n=_readint();;
_t++;
p[0]=0;
_t++;
for(_i = 0; _i < 1024 ; _i++)
__p[_i] = p[_i];
```

```
__i = i;
__n = n;
for(_p = (_p1 =1);_p <= (_p2 =n-1);_p ++){
_u[_p]=0;
p[_p]=_p-1;
_u[_p]++;
}
_t=_t+_max(_p1, _p2);
for(i=1;i<=n;i=i*2){
_t=_t+2;
for(_i = 0; _i < 1024 ; _i++)
__p[_i] = p[_i];
__i = i;
__n = n;
for(_p = (_p1 =0);_p <= (_p2 =n-1);_p ++){
_u[_p]=0;
p[_p]=__p[__p[_p]];
_u[_p]++;
}
_t=_t+_max(_p1, _p2);
}
for(_i = 0; _i < 1024 ; _i++)
__p[_i] = p[_i];
__i = i;
__n = n;
for(_p = (_p1 =0);_p <= (_p2 =n-1);_p ++){
_u[_p]=0;
printf(“%d”,p[_p]);
_u[_p]++;
}
_t=_t+_max(_p1, _p2);
printf(“実行時間は%dです。”,_t);}
int _max(_p1, _p2) int _p1,  _p2;{
int _i;
int _max = _u[_p1];
for(_i = _p1+1;_i <=_p2;_i++)
if(_max <= _u[_i])
_max = _u[_i];
```

```
return _max;
}
int _readint() {
int _i;
scanf( “%d” ,&_i);
return _i;
}
```