卒 業 論 文

並列処理学習支援のための Javaによる可視化(2)

氏　　名 ： 石丸 智和

学 籍 番 号 ： 2210950016-3

指 導 教 員 ： 山崎 勝弘 教授

提　出　日 ： 1999年2月17日

立命館大学 理工学部 情報学科

内容梗概

本論文では、並列処理学習支援のための Java による可視化について述べる。並列処理のアキレス健ともいわれる並列処理による速度向上を妨げる要因である 「通信オーバーヘッド」、「逐次処理部分のオーバーヘッド」、「負荷のばらつき」について、その理解を深めるためにイメージを可視化することによって初心者でも並列処理の理解を深めることを目的とする。手法としては Java を用いてアプレットとして作成した。

Java とは１９９５年に Sun Microsystems が発表した言語で、プラットフォーム独立であり、WWW 経由で実行可能であるアプレットを作成することができるオブジェクト指向言語である。また、Java の持つクラスライブラリを使うことにより、簡単に様々な機能を実現できる。

本論文では、まず、並列処理と Java の基本構造について述べた後、具体的な可視化の方法について述べ、作成したアプレットの実行結果とその評価について述べている。本研究において、教育支援の実現のため、「いかにユーザの理解を助けるか」、という視点で研究を行ってきている。

目 次

1 はじめに **1**

2 並列処理 **3**

2.1 並列処理の目的 3

2.2 MIMD 方式における並列化 4

2.3 通信オーバーヘッドによる問題 5

2.4 逐次処理部分のオーバーヘッドによる問題 6

2.5 負荷分散のばらつきによる問題 6

3 Java による並列処理イメージの可視化 **7**

3.1 Java の概要と基本構造 7

3.2 可視化の方法 11

3.3 実行画面 12

4 吸引袋を用いた並列処理の可視化と評価 **13**

4.1 実行の流れ 13

4.1.1 理想的な場合 13

4.1.2 負荷にばらつきがある場合 17

4.1.3 負荷にばらつきがない場合 18

4.1.4 通信時間の違う場合 20

4.2 評価 21

5 おわりに **22**

謝辞 **23**

参考文献 **24**

付録 **25**

1　はじめに

コンピュータの歴史において、その実行性能の向上は、素子技術や実装技術の発展に頼って来た面が大きかった。これは、いずれ信号伝播速度の限界である光の速さという壁にぶち当たるのは目に見えている。そこで、並列処理による実行性能の向上に目を向けてみた。並列処理というものを学習すると、並列処理が逐次処理より処理速度が速く、時間の節約ができるので、とても便利だということがわかる。しかし、誰にでも簡単に利用できるとは言えず、一部の専門家だけしか実現できないのが現状であり、初心者が気軽に取り掛かることが困難である。そのことをふまえて、並列プログラムを書けるように学習すれば並列処理のありがたみを知ることができるのか ?そうではない。並列化にはその性能向上を妨げるいくつかの問題がある。それをふまえた上で並列処理を学べば、並列処理はいろいろと工夫することによって速度が上がり、努力しがいのあるものだということがわかるであろう。

本研究では、並列処理のアキレス健ともいわれる並列処理による速度向上を妨げる要因である「通信オーバーヘッド」、「逐次処理部分のオーバーヘッド」、「負荷のばらつき」について、その理解を深めるためにイメージを可視化することによって初心者でも並列処理の理解を深めることを目的とする。手法としては Java を用いてアプレットとして作成した。

Java は、オブジェクト指向言語で、言語仕様は C++と非常によく似ている。Java が他のプログラミング言語と大きく異なる点は、Java がプラットフォーム独立ということである。プラットフォーム独立とは、プラットフォーム (PC 上の Windows95 や WindowsNT、Macintosh、UNIX) に関係なく、アプリケーションが動作できるということである。Java では、ソースレベルに止まらず、バイナリレベルにおいてもプラットフォーム独立である。[4] この事によって、多くの異なる機種で動かす事が可能となる。Java は Sun Microsystems 社が 1995 年に発表した言語である。また Java はアプリケーションとアプレットに分かれている。アプレットとは、WWW 経由で Web ブラウザ上にダウンロードされて、実行されるプログラムの事である。

また、Java ではネットワーク、インターネット関係に対応するオブジェクト ・クラスが用意されていて、これを使用することにより、インターネットを介するだけで簡単にアプレットを呼び出し、実行することができる。Java の機能として、主に、 1 . グラフィックスの描画、 2 . ボタン、テキスト ・フィールド、チェックボックス、などの GUI 部品の利用、 3 . 時間機能の利用などが挙げられる。[1]

さらに、実現方法としては、上記で述べたような Java の機能を有効に利用して青い丸 1 個を 1 つの処理対象、袋 1 つを 1 台のプロセッサと見立てる。基本的にはこの処理対象とプロセッサでアニメーションを構成している。このアニメーションは、並列処理のプロセッサや処理対象の数、処理速度、通信時間を Java の機能のボタン、テキストフィールドなど

の GUI 部品を用いて設定できるようにしている。プロセッサ (袋) はそれぞれの性能の違い (処理速度) を表すために、袋のスピードの違いでプロセッサの性能の良し悪しを表現し、これを用いることによ って、 「負荷のばらつき」を示した。通信時間は、 5 段階で設定でき、これにより通信時間の早いとき遅いときと比較できる。従 って、目で見ることにより、並列処理にとって通信時間は大きなウエイトを占めているというのが分かる。本研究では、このように様々な場合を可視化する事によ って、並列処理学習を支援する。

本論文では、 2 章で並列処理による効果を妨げる要因について、 3 章で Java の概要と基本構造と並列処理イメージの可視化の方法と機能、 4 章で可視化の実行と評価で行い、 5 章で本論文の結語と今後の課題を示す。

2　並列処理

2.1　並列処理の目的

一般に、正しく作動する逐次プログラムを作るのは大変であるのだが、ましてや正しく動作する並列プログラム記述するのは遥かに大変である。[7] では、なぜあえて、並列プログラムを書く必要があるのか、私は、並列処理を人間の仕事に例えて考えてみた。1 人だと 1 日かかる仕事があるとする。この仕事を 2 人で協力すれば、約半日で終らせることができるかもしれない。 このように、計算機でも同じようなことが考えられるはずである。とても複雑な計算をするプログラムを作ったとき、ワークステーションで普通に実行させると、計算が終了するまで 1 日かかってしまう場合があるはずである。このプログラムをなんとか分割して 2 台のワークステーションを使って実行すると，約半日で計算が終了するかもしれない。 [8] このように，複数台の計算機や，CPU を同じ目的のために同時に用いて計算することを並列処理と呼び、単に複数のワークステーションを同時に動かしても並列処理とは呼ばない。同じ目的のために動かして，初めて並列処理と呼ぶのである。

次は、並列処理の目的について述べたいと思う。目的は、主に次の 3 点が挙げられる。

1. 高速性能追求

   これは、 仕事を分担して早く終わらす」ということを主眼としたもので、並列プログラミングの主な観点である。これは、経過時間 (elapsed time, wallclock time) を短くするということで、ＣＰＵ時間を短くするという事ではない。つまりは処理時間を短縮することである。一般に複数のプロセッサで並列処理を行うとＣＰＵ時間は増加する。[8] それ以外では、信頼性の向上がある。これは、複数の計算機で全く同じプログラムを実行させておくものである。こうしておけば、ある計算機が故障しても、他の計算機が故障した計算機の処理を代行することができるはずである。処理効率は落ちるだろうが、システム全体のダウンは免れるだろう。

2. プログラミングが楽になるため

   これは、マルチスレッドプログラミングはこの観点から行うことも多い。スレッドとはプログラムの実行の道筋のことであり、マルチスレッドとは 1 つのプロセス中に複数のスレッドが動作することである (図 1 参照)。

   マルチスレッドを用いると、同一プロセス中の複数のプログラムコードを同時に実行できる。マルチスレッドは 1 つのプロセス内で動作するためスレッド間でアドレス空間を共有する。各スレッドはプログラムカウンタ、スタック、スタックポインタを個別に持つが、プログラム、変数領域を共有して処理を行う。[9]

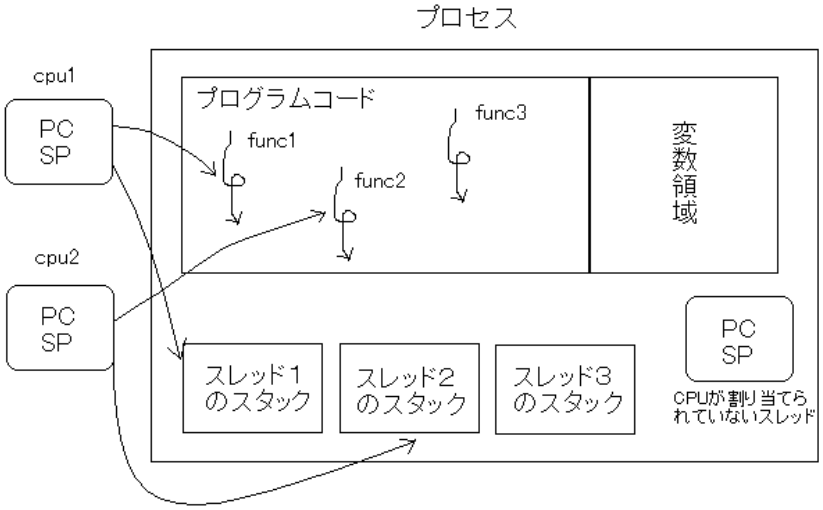

図 1: マルチスレッド

また、本論文で後に述べる Java についてもマルチスレッドを記述できるように対応している。

3. 本質的に並列なものを扱うため

以上が、基本的な並列処理の目的である。

2.2 MIMD 方式における並列化

一言で並列処理といってもいろいろな形態がある。フリン (M．J．Flynn) は、命令とデータの流れの並列性に着目してコンピュータを次のように分類した。

- SISD (single instruction stream , single data stream)
  単一命令流単一データ流。逐次型コンピュータを指す。
- SIMD (single instruction stream , multiple data stream)
  単一命令流多重データ流。大量データに対する同一演算、行列計算、画像処理などの大規模数値計算。並列化の効果大。あるタイミングにおける命令はやっていることは同じである。
- MIMD (multiple instruction stream , multiple data stream)
  多重命令多重データ流。複数の命令が複数のデータに対するそれぞれ異なる処理を制御する。並列化の可能性は大きいが、適用できる問題が限定される。並列化の効果はあまり大きくない。あるタイミングにおける処理がさまざまで柔軟性は高い。

以後、現在主流である MIMD 方式に着目していきたいと思う。

MIMD 方式における並列化の方法としては、まず機能的分割、データ分割、両者の混合と必要に応じて仕事を与えていく枝刈り法があげられる。実際にはプロセッサの台数を増加させる場合、純粋に機能的な分割だけを用いることはほとんどなく (並列性をそこまで出すのは難しいため)、何らかの方法でデータ分割を扱う必要がある。

MIMD マルチプロセッサは、マルチプロセッサ上で走るプロセスが互いに共有メモリを介して通信を行う。共有メモリの構成方法は以下のようなものが挙げられる。

- 集中共有メモリ

  プロセッサ n 台と m バンクのメモリが相互結合網を介して結合される。m 個のバンクを同時にアクセスできる。

- 分散共有メモリ

  各プロセッサにローカルメモリを分散配置したもの。

  全体としては 1 つのアドレス空間にみえる。各バンクにアドレスを割り振ることによって、物理的には各プロセッサに分散配置されたメモリを、論理的には共有メモリとしてアクセスできる。このため仮想共有メモリとも呼ばれる。

- 部分共有メモリ

  プロセッサ毎の局所メモリと、プロセッサ全体の共有メモリがある。

ここで各プロセッサから共有メモリへのアクセス時間がアドレスによって同じものを UMA(uniform memory architecture)、異なるものを NUMA(non-uniform memory architecture) と呼ぶ。[2]

また、MIMD マルチプロセッサではキャッシュメモリの使用が一般的である。頻繁に使用するプログラムやデータがキャッシュに存在することになるため、プロセッサ間の共有資源である相互結合網や共有メモリでの競合を抑えることができる。

2.3 通信オーバーヘッドによる問題

並列プログラムでは、他のプロセッサとデータを通信すること/あるいは全プロセッサで同期をとることが必要になる。このための時間は当然のことながら逐次プログラムにはないオーバーヘッドとなる。[7] ここでは、特に、通信のオーバーヘッドについて述べたいと思う。通信は並列プロセスが相互に影響を与えながら計算を進めるためには必要なものであるが、この通信は、単にデータの移動であるので、純粋なオーバーヘッドとしてあらわ

れる。これを減らすためには、アルゴリズムを修正、改良し通信の削減を行うのが一番効果的である。

この問題は唯一ハードウェア / アーキテクチャで対応できる部分である。

2.4 逐次処理部分のオーバーヘッドによる問題

アムダールの法則より、逐次処理部分の割合が全体の $s(0 <= s <= 1)$ 存在すれば n 台のプロセッサを用いて得られる速度向上は、

$$\frac{1}{s+(1-s)/n}$$

になる。例えば 1 台実行時に逐次部分の割合が 5 %あれば、たとえ１０００台のプロセッサを用いても、得られる速度向上は高々２０倍である。[7]

この問題に対する対策としては、基本的にはアルゴリズムやライブラリレベルで、並列部分の割合が大きくなるようにアルゴリズムを工夫する。

2.5 負荷分散のばらつきによる問題

並列システムでは n 台のプロセッサを用いた場合、すべて並列処理可能であれば理想的には $1/n$ の時間に並列実行部分を短縮できるはずである。しかし、各プロセッサで負荷のばらつきがある場合、最も遅いプロセッサに引っ張られることになる。[7]

この問題に対する対策は、主にアルゴリズム / ライブラリ / 並列化コンパイラで対応すべきである。

3　Java による並列処理イメージの可視化

3.1　Java の概要と基本構造

まず、Java について述べる前に Web ブラウザというものを忘れてはならないと思う。これは、インターネットで注目されており、ハイパーテキスト形式でネットワーク上に分散されたデータをリンクするシステムである。Web ブラウザのおかげで、インターネットユーザは、文字だけではなく絵を見たり音声を聞きながら、世界中の情報を手軽に検索することができるようになった。これにより、インターネットは、一部の専門家だけのものではなくなった。つまり、多くの一般の人たちに注目されるようになったのである。そして、1995 年、またまたインターネットに新たな旋風がまきおこった 。

それは Java の登場である。Java は、１９９５年５月に米国サンフランシスコで開催された SunWorld ' 95 で、サンマイクロシステムズによって発表されたオブジェクト指向プログラミング言語である。この言語は、HotJava というブラウザで、アニメーション、グラフィックス、音声などを実行 ･表示するためのものとして知れわたった。そして、ネットスケープ ･コミュニケーションズの製品 (Netscape Navigator) に Java の対応が決定され、ブラウザ用言語としての地位はゆるぎないものとなった 。　最近では、Web といえば必ずといってよいほど Java が取りあげられているが、産業界などでも Java は、イントラネット (社内サーバーとインターネットを組み合わせ外部との業務効率化する) や EC (エレクトロニックコマース :電子商取引) などの基盤として異常なほど脚光を浴びている。[8]

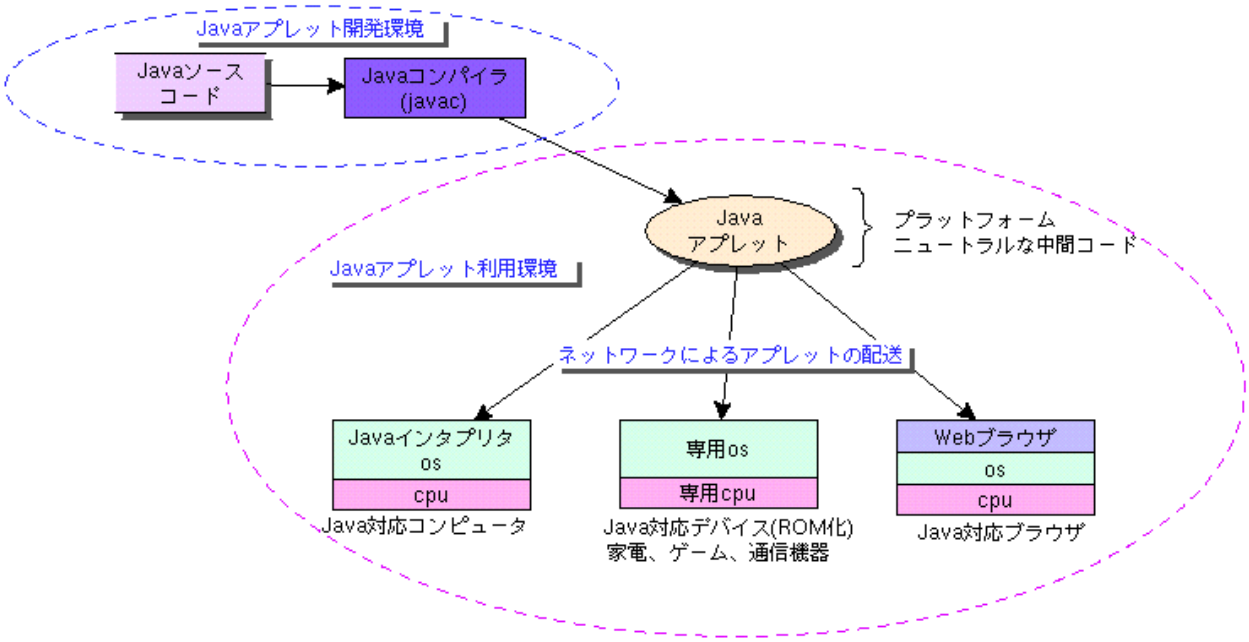

図 2: Java の概要

アプレットの概念を図 3 に示す。Java ソースファイル (Applet クラス) をコンパイルして作成したアプレットは、すべてのマシンで使用できるように中間コードと呼ばれるコードで出力される。 HTML のタグがアプレットの中間コードを解釈しながら実行して行く。つまり、Java アプレットを使用できるのは Java インタープリタを内蔵したブラウザ (HotJava,Netscape など) である。

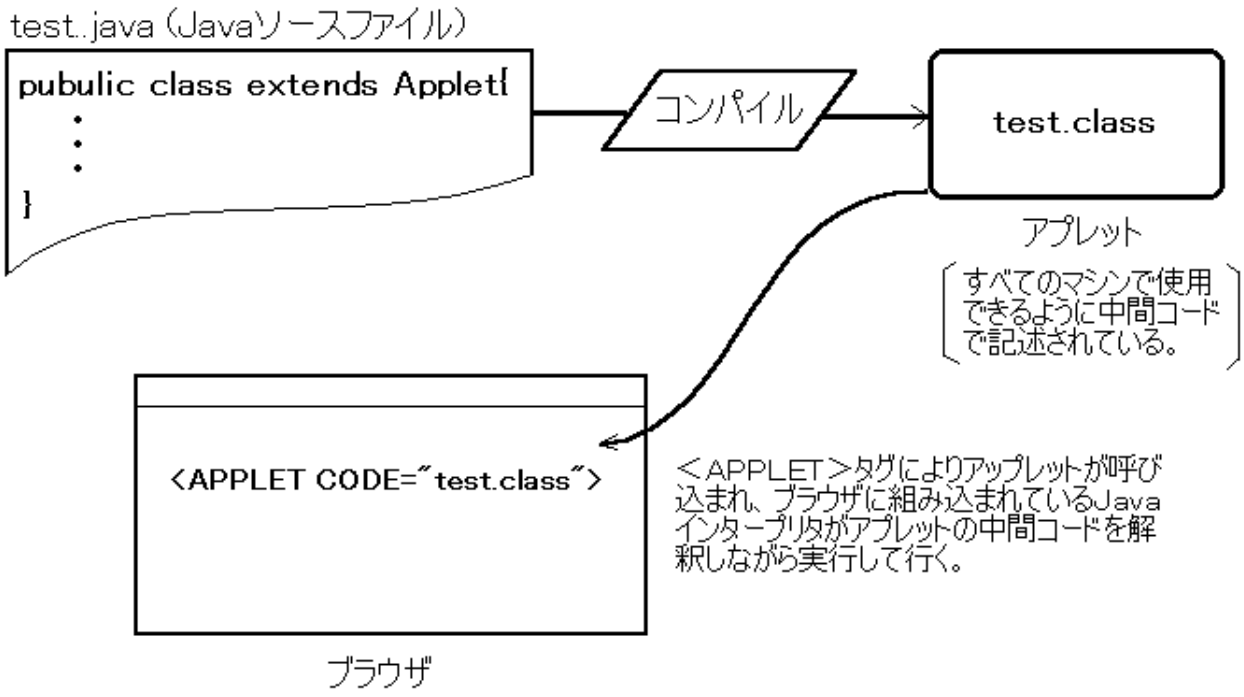

図 3: アプレットの概念

次に、Java の基本構造について述べる。Java でプログラムを作るのに既存のクラスとインターフェースを利用し、ユーザ独自のクラスを作成する。この既存のクラスとインターフェースを API と呼ぶ。さらに、API パッケージの中で定義されている色々なパッケージを使用して、グラフィックスの描画、ボタン、テキスト ・フィールド、チェックボックス、などの GUI 部品の利用などが実現できるようになったり、他にも様々な機能を実現できるようになる。API パッケージは 7 種類に分類でき、さらにそれらの中に複数のクラスが定義されている。ここでは、その中から本研で使用したクラスとオブジェクトいくつか紹介したいと思う。

- Applet クラス

  これは Panel から派生されたクラスである。本研究では、メソッドのほとんどは、Applet の上位クラスで実装されているものであったので、Applet クラスを利用する際にはそれを意識する必要はなかった。

- Thread クラス

  これは、１つのプログラムの中で並列に動作する実行単位のことである。Thread クラスを使えば、複数 Thread を簡単に実現できる。アプレットでは、これを利用してアニメーションを実現した。また、大量のリソースを使う場合などにも Thread を使って、読み込みと実行を並列に行うことも可能であるが、本研ではそのことについては触れていない。

- オブジェクト Graphics

  これは、絵や文字などをウィンドウに表示するためのものである。このオブジェクトは、Component の派生クラスである Applet クラスメソッド paint() に、パラメータとして引き渡されるものです。paint() メソッドに引き渡された Graphics 描画メソッドを使用すると、直接表示域に描画されるものである。

- Runnable

  これは、Thread クラスインスタンス作成に target を渡すものである。このコンストラクタは、本書のアニメーションアプレットのすべてに使用しているものである。ちなみに、Runnable というのは、Java 言語の 『インターフェース』という機能である。このインターフェースを実装するクラスは、以下のように implements を使う。

  ```
  public class test extends Applet implements Runnable {
  ```

  上記の test クラスは、スーパークラス (* 後述) に Applet、インターフェースに Runnable を持つものであり、インターフェースとは 『約束事を定義したもの』と考えるとよいであろう。具体的に言うと、Runnable インターフェースの約束事は、run() メソッドを実装することである。

- マルチスレッド機能の実現方法

  1. アプレットクラスを定義する際に Runnable インターフェースを実装する。
  2. メソッド start() を、スレッド生成し、実行を開始をする。
  3. メソッド run() を生成して、その中でアプレットの動作を記述する。
  4. スレッドを保持するためのインスタンス変数を定義する。

- アプレットの動作順序

  アプレットがブラウザから呼ばれ実行されるときは、図 4 に示す順序でメソッドが呼び出される。ユーザは必要に応じて各メソッドの定義内容を変更する。変更しなければ、Applet クラスで定義されている基本メソッドが実行される。

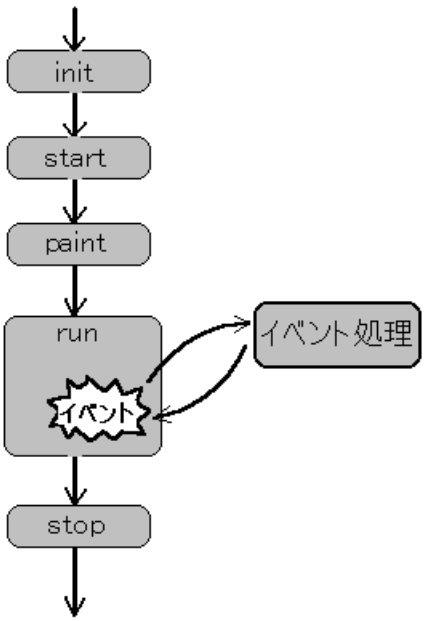

図 4: アプレットの動作順序

- スーパークラス
  作成するユーザークラスの元になるクラスをスーパークラスと呼ぶ。ユーザークラスはスーパークラスの機能を継承する。(図 5 参照) 変更また拡張する機能 (メソッド) だけをユーザークラスの定義部で記述する。スーパークラスを継承および拡張 (extends) して作ったクラスをサブクラスと呼ぶ。アプレットを作るには、Applet クラスをスーパークラスにしてユーザークラスを作成する。[9]

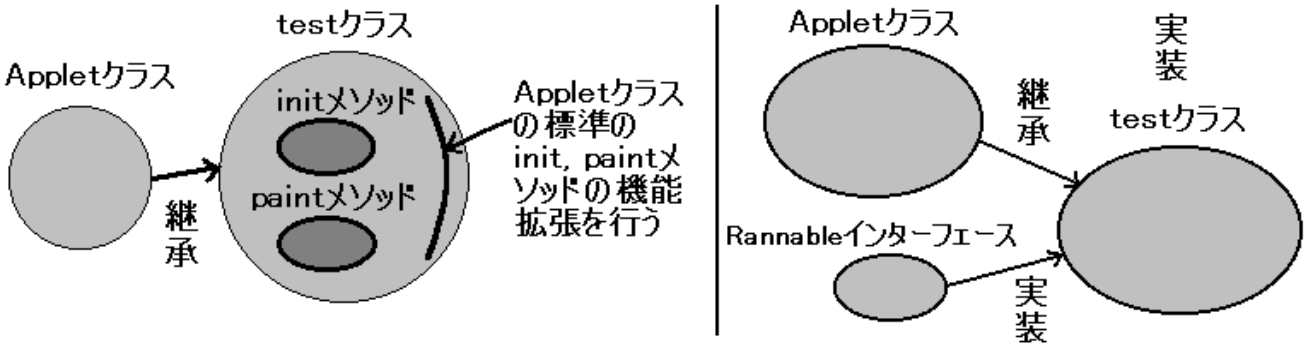

図 5: クラスの継承と実装

3.2 可視化の方法

Java のスレッド機能を用いてアニメーションを表示する。

1. プロセッサ (袋) と処理対象 (青い玉)

   袋は、ペイントツールを用いて描画し、青い玉は、fillOval メソッドを用いて描画した。

2. Color クラス

   グラフィックスを描画するときの色を指定するには、Color クラスを利用する。色情報は、Color クラスのオブジェクトに記憶されるようになっており、描画の指定をする時には、まず Color クラスのオブジェクトを生成する。さらに、よく利用する色は、あらかじめ Color クラスの変数として用意されている。これは、「static 変数」という特別な変数である。

3. 背景色

   getBackground() メソッドを用いて背景色を指定した色にする。背景色と描画色を似た色にすると、分かりにくくなるからである。本研究では、白色を背景色にしている。

4. paint() メソッドと repaint() メソッド

   repaint() は、ウインドウシステムにスクリーンを update する必要があることを知らせるために用いる。このメソッドが呼ばれると、ウインドウシステムが paint() ルーチンを呼び出して、新しく画面に表示する。repaint() は、単に、スクリーンを更新するリクエストを与えるだけで即座に戻ってくる。その効果は、非同期であるので、複数の paint リクエストがあった場合には、最後の一つだけが実行される場合もある。[1]

さらに補足すると、アプレットに 1 種類の部品だけであれば、判別する必要はないが、このインターフェース複数の種類の部品が存在する場合、instanceof 演算子を用いる必要がある。また、instanceof 演算子はキャストが成功しているかをチェックする [1]。

3.3 実行画面

ここでは、図 6 の場合を例に説明する。

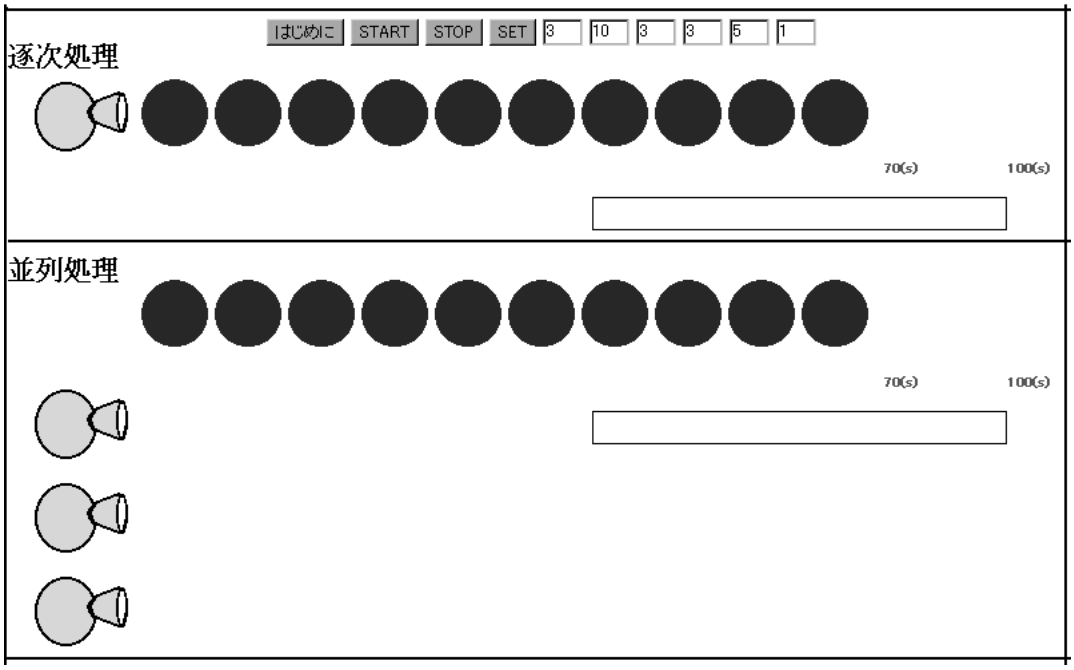

図 6: インターフェース部

- 青い玉 1 個を 1 つの処理対象、袋 1 つを 1 台のプロセッサと見立てる。
  以下、並列処理の袋を上から A、B、C と呼ぶ。
- １０個の青い玉を用いて、袋が 1 つの場合と 3 つの場合とで比較した。
- 右の棒グラフは、それぞれの処理時間を表した。
- テキストフィールドの左から、並列処理の袋の数、青い玉の数、A、B、C の処理速度、通信時間が設定できるようになっている。(ただし、処理速度、通信時間は 5 段階表示である)
- すべての数値を代入した後で、「SET」ボタンを押す。
- 「START」ボタンを押すと、アプレットが動き出す。
- 「STOP」ボタンを押すと、アプレットは停止する。
- 「はじめに」のボタンを押すと、このような説明が出てくる。

4　吸引袋を用いた並列処理の可視化と評価

4.1　実行の流れ

4.1.1 では、理想的な場合、つまりすべてのプロセッサの処理速度が同じで、負荷均衡もとれている場合のアプレットについて説明する。そうすることにより、並列処理の利点がより明確なものになると考える。さらに、4.1.2、4.1.3、4.1.4 では、並列処理による速度向上を妨げる要因を理解するために、様々な場合で比較、検討している。

4.1.1　理想的な場合

図 7 から図 13 までのアプレットから並列処理の理解を深める。

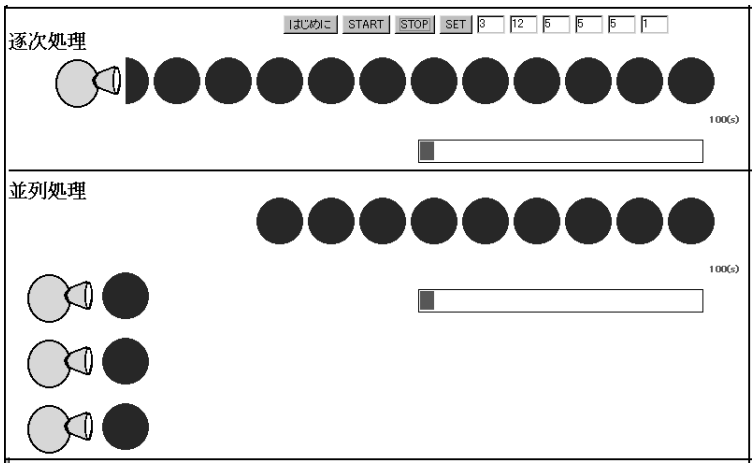

図 7: 玉の振り分け

- 逐次処理ー 1 個目の玉を吸い始める。
- 並列処理ー 1 2個の玉を 3 つの袋 (プロセッサ) に振り分け始める。
  (通信のオーバーヘッド)

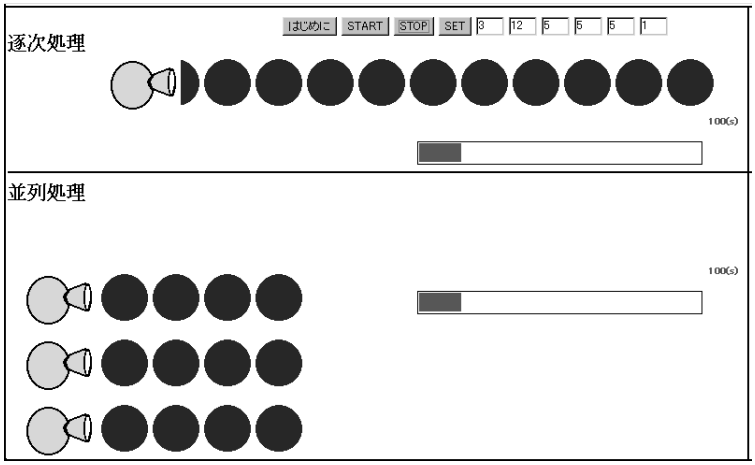

図 8: 振り分け終了

- 逐次処理ー 2 個目を吸取 っている。
- 並列処理ーすべての玉の振り分けを終える。

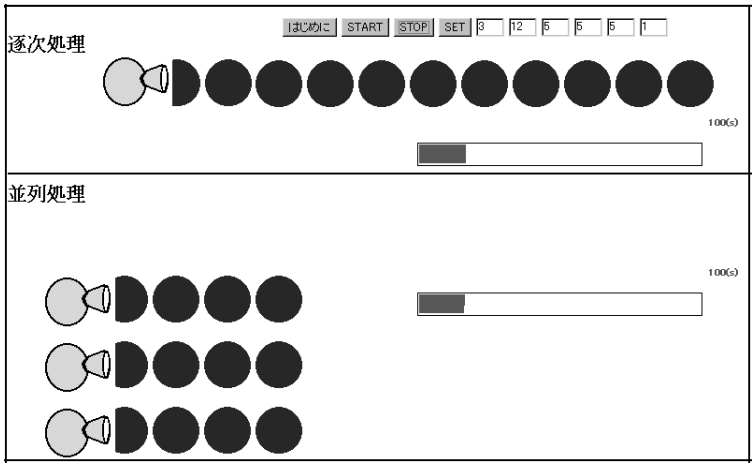

図 9: 処理の途中

- 逐次処理ーまだ、2 個目を吸い取 っている。
- 並列処理ー 3 つの袋がそれぞれ吸い始める。

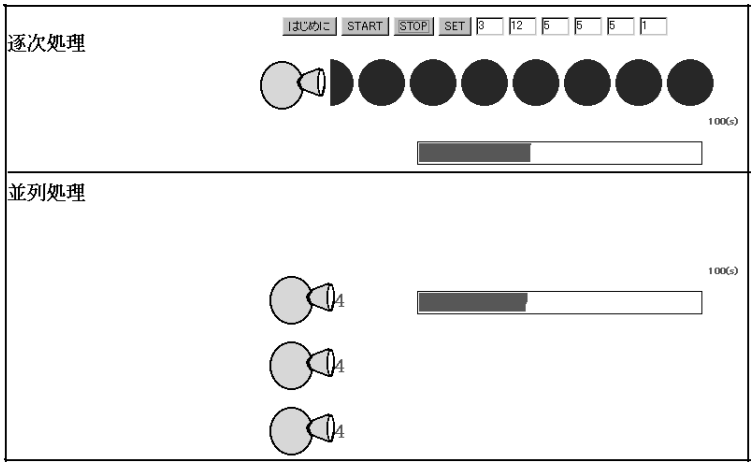

図 10: 同時に吸い終える

- 逐次処理ー 5 個目を吸い取っている。
- 並列処理ーすべての玉を吸い終えて、それぞれ４、４、４と表示する。

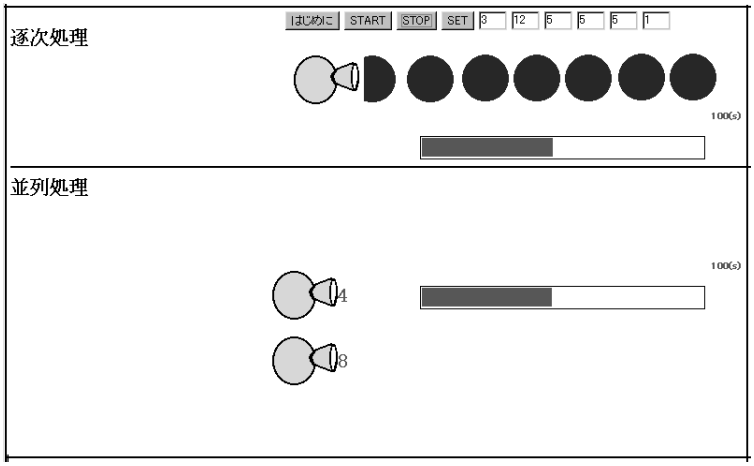

図 11: 逐次処理部分のオーバーヘッド

- 逐次処理ー 6 個目を吸い取っている。
- 並列処理ー B が C を吸いとる。そして、 8 と表示する。ここで、B が C を吸取ったのは逐次処理部分のオーバーヘッドを表現するためである。

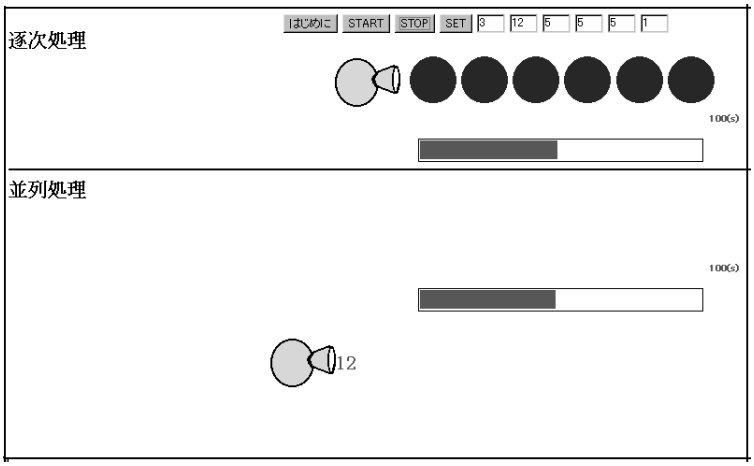

図 12: 並列処理終了

- 逐次処理の方はまだ処理を終えていない。
- 並列処理は図 11 と同様にして、１２と表示する。

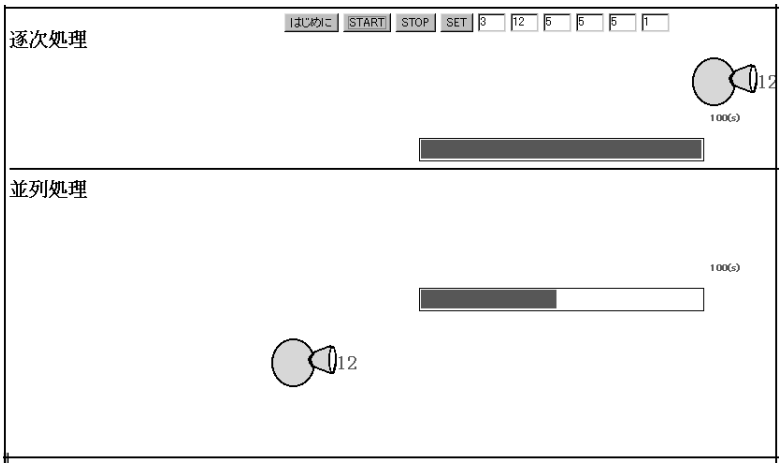

図 13: 逐次処理終了

- 逐次処理も処理を終え、１２と表示する。

4.1.2　負荷にばらつきがある場合

ここでは、C の処理速度が最も早く、振り分ける玉も上から 4、 3、 3 と設定してある場合について説明する。最初に玉を振り分ける時や処理の途中等は、4.1.1 と同じなので、ここでは C が 3 個の玉を吸い終え、負荷のばらつきが出たところまでのアプレットから載せたいと思う。

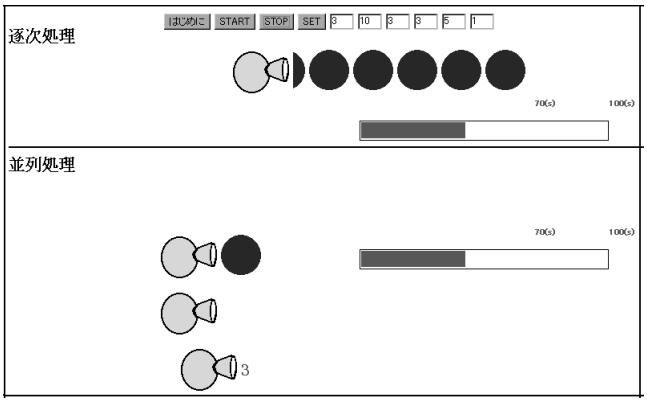

図 14: C の処理終了

- 逐次処理ー 5 個目を吸取 っている。
- 並列処理ー C が 3 個吸い終り、 3 と表示する。

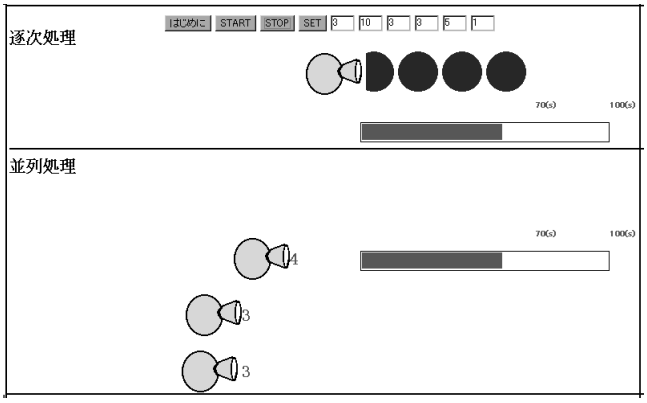

図 15: 同期をとる

- 逐次処理ー 6 個目を吸い終える。
- 並列処理ーすべての玉を吸い終えて、それぞれ 4、 3、 3 と表示する。A がすべて吸い終えるまで B と C は待機する。 (負荷のばらつき)

4.1.3　負荷にばらつきがない場合

ここで、4.1.2 の場合と違うのは、A の処理速度が一番早いと言うことである。4.1.2 の場合と同様に処理の途中からのアプレットを表示する。

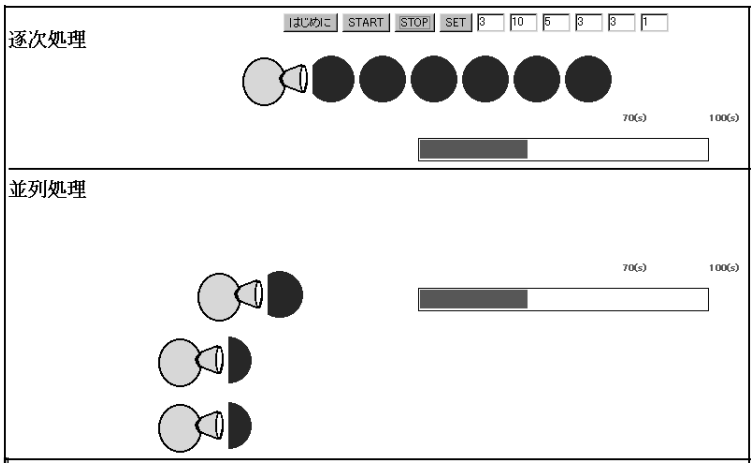

図 16: 処理の途中

- 逐次処理ー 5 個目を吸いはじめている。
- 並列処理ーまもなく、全てのプロセッサが玉を吸い終える。

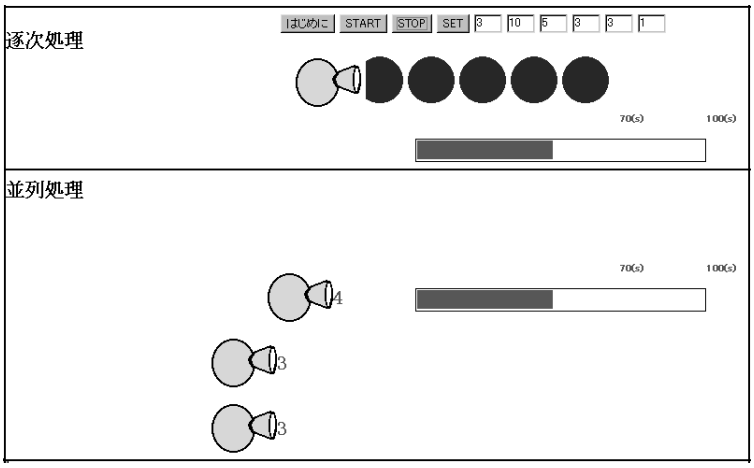

図 17: 同期をとる

- 逐次処理ー 6 個目を吸い取っている。
- 並列処理ー全ての玉を吸い終える。 (負荷のばらつきは見られない)

- 4.1.2 と 4.1.3 のアプレットの比較

4.1.2 と 4.1.3 の違いは今まで述べてきた通り、どのプロセッサの処理速度を最も早くしているかと言うことである。図 18 と図 19 はそれぞれの結果を示したものである。

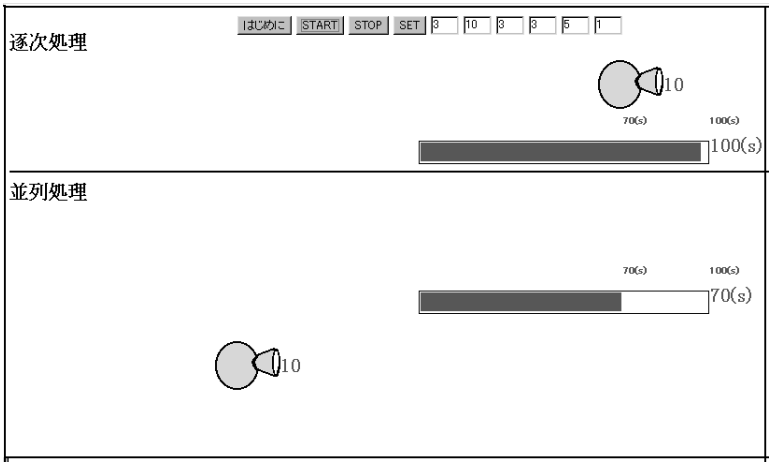

図 18: 負荷のばらつきがある場合

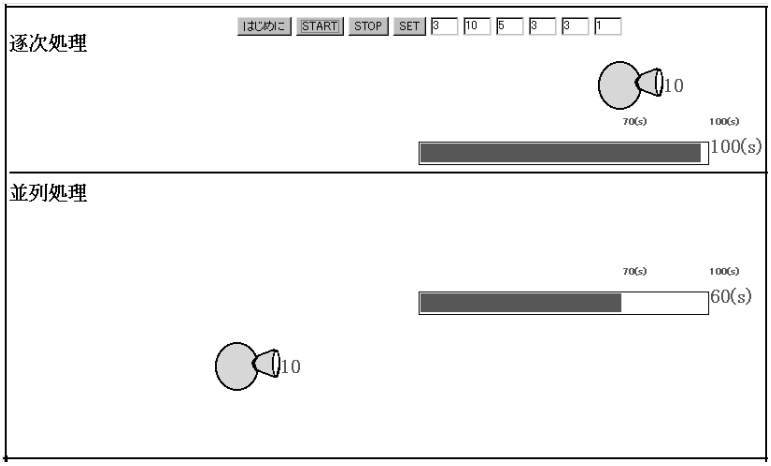

図 19: 負荷のばらつきがない場合

図 18 と図 19 を比較すると分かるように、負荷均衡がとれてる場合とそうでない場合とでは、全体の処理時間が変わってくる。今回は、プロセッサが 3 台、処理対象が 10 個で設定してあるので、それぞれが処理する数も４、３、３と異なってくる。2 つの図の違いは、4.1.2 では C の処理速度を速く、4.1.3 では A の処理速度を速くしているところだが、そこ部分だけを変えるだけで負荷のばらつきを押さえることができるということが分かる。また、実際に図 18 と図 19 で比較してみると、10 秒の差が出ている。

4.1.4　通信時間の違う場合

ここでは、通信時間だけを変えたアプレットを比較することにより、その効果を検討する。

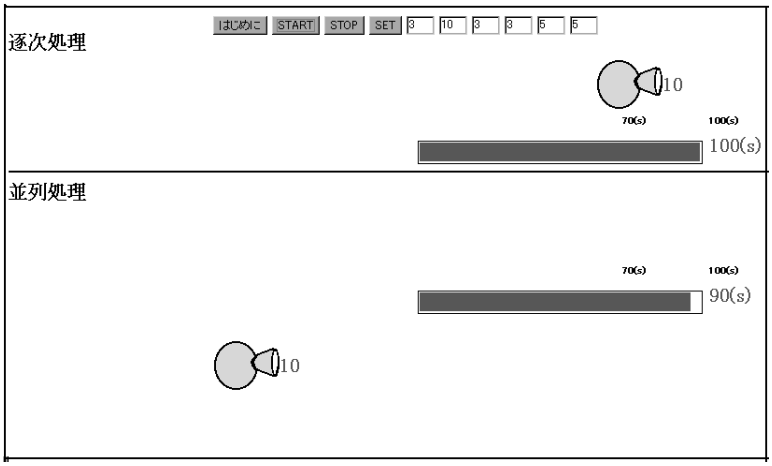

図 20: 通信時間の長いとき

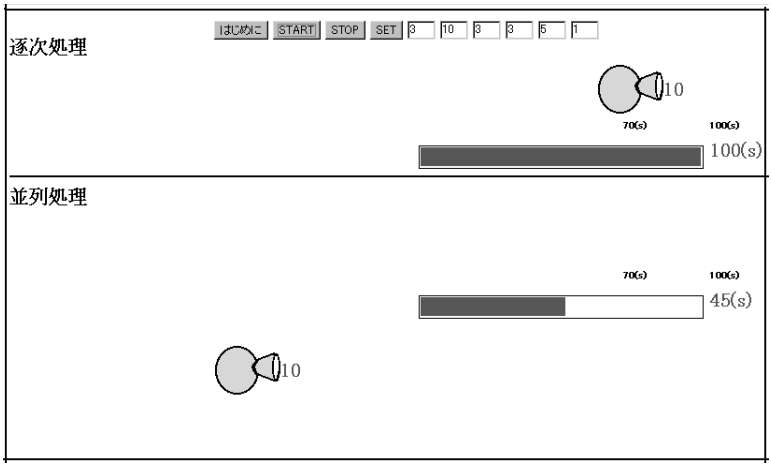

図 21: 通信時間の短いとき

図 20 と図 21 とを比較すると、通信時間の短い方の速度向上が大きいのは明らかである。しかし、さらに具体的な数値を出すことによって理解を深めようと思う。

- 通信時間の長い場合

  1 台の実行時間... 100 秒
  ———————————————— =1.1 倍である。
  3 台の実行時間... 90 秒

- 通信時間の短い場合

  1 台の実行時間... 100 秒
  ———————————————— =2.2 倍である。
  3 台の実行時間... 45 秒

理想としては、3 倍の速度向上を得たいのだが、実際にはどちらもそのようにはならないのは、今まで述べてきた 「通信時間のオーバーヘッド」、 「負荷のばらつき」、 「逐次処理部分のオーバーヘッド」によるものである。ここで注目したいのは、 2 つの図は 、通信時間だけしか設定を変えていないのに速度向上が、倍ほど違うということである。このことからも、通信時間の短縮は大きなウエイトを占めているのが分かる。

4.2 評価

本研究の全体の評価として、並列処理のアキレス健ともいわれる並列処理による速度向上を妨げる要因である 「通信オーバーヘッド」、 「逐次処理部分のオーバーヘッド」、 「負荷のばらつき」については、初心者の理解を助けるものになったと考える。また、これで並列処理のすべてを理解するというのは困難と思うが、並列処理学習支援の導入としては、おもしろいものに仕上がったと考える。また、図 11 でも示している逐次処理部分のオーバーヘッドのイメージは理解しずらいとの意見も頂いたので、改良の余地も残されているように思える。

5　おわりに

本研究では、並列処理学習支援のための可視化 (アニメーション) を、浅野 剛史と協力し、Java を用いて実現した。Java を学習するとともに、どのように可視化をすれば、並列処理を初めて学ぶ者の導入をスムーズに行えるかを研究してきた。また、作成したアプレットの問題点として、以下のものが挙げられる。

- 処理対象を円 (玉) として表現していたためデータの個数が制限されるので、規模の大きな並列処理の可視化が不可能
- 使用するマシン、ブラウザソフトにより、動作時間に大きな違いがあるため、場合によってはアプレットが起動しない。

以上のような問題をふまえた上での今後の課題としては、処理対象の玉の数を 100 個、1000 個と増やした上で、本論文にもあったような通信時間の違いによる比較を行うことである。これにより、並列処理の速度向上も理想に近くなると考える。また、この機能があれば、初心者のさらなる理解も期待できるであろう。

謝辞

本研究の機会を与えて下さり、数々の助言を頂きました山崎勝弘教授に心より感謝致します。 また、本研究にあたり、いろいろな面で貴重な御意見や、励ましを頂きました本研究室の皆様にも心より感謝します。

さらに、お忙しい中数々の質問に答えていただいた、米田 健治氏、上平 祥嗣氏に深く感謝します。そして、共同研究者の浅野 剛史氏にこころから感謝します。

参考文献

[1] 河西 朝雄 ：Internet Language3 Java 入門 , 技術評論社 (1997)

[2] 馬場 敬信 ：コンピュータアーキテクチャ , オーム社 (1994)

[3] 植田 龍男 ：Java プログラミング実践講座 , アスキー出版局 (1997)

[4] PETER van der LINDEN 著 ; 中田 秀基 訳 ：just JAVA , アスキー出版局 (1997)

[5] 有賀 妙子; 竹岡 尚三 ：Java プログラムデザイン, ソフトバンク (1998)

[6] 戸松 豊和 ：JAVA プログラムデザイン , ソフトバンク (1997)

[7] 湯浅 太一; 安村 通晃; 中田 登志之 : はじめての並列プログラミング (bit 別冊) , 共立出版 (1998)

[8] 並列処理入門 ：http://www.tuat.ac.jp/ kiwasawa/para/Parallel.html

[9] 浅野 剛史 ：並列処理学習支援のための Java による可視化 (1) , 立命館大学理工学部情報学科卒業論文 (1999)

付録

ここでは、本システムで使用するために作成したJavaのプログラムについて、外部仕様と内部仕様の説明をする。

外部仕様

ソースプログラムは–.javaという拡張子が付けられ、ソースをコンパイルすると拡張子–.classがついたクラスファイルが作成される。Javaアプレットはブラウザで実行するため、クラスファイルをHTMLファイルに組み込む。組み込みには直接HTMLファイルに、クラスファイル名にアプレットタグをつけて記述する。アプレットを起動するにはブラウザでHTMLファイルを開けばよい。

アプレットを操作するには、最初にテキストフィールドにデータを入力し、『SET』ボタンをクリックしてキャストを行う。次に『START』ボタンをクリックし、アニメーションが動き出す。なお、『STOP』ボタンをクリックするとアニメーションが停止し、もう一度『START』ボタンをクリックすれば、停止したところからアニメーションが再開する。『はじめに』ボタンをクリックすれば、このアニメーションの説明が新しいウインドウ上に表示される。

内部仕様

変数

はじめに以下の変数と配列を定義する。

```
int x,y1,y2:袋の動き
int x1,x2:青い玉の数
Button B1,B2,B3,B4:ボタンを作る

Image imgs1[]:イメージを格納する
TextField MyF[]:テキストフィールドを作る
String args[]:テキストフィールドのデータを格納する
```

ソースプログラム

ソースプログラムの大まかな枠組みは以下の様になる。

```
import java.awt.*;
import java.applet.*;
import mywindow;
public class ファイル名 extends Applet implements Runnable{

変数の定義などをする部分　(この部分は以下で詳しく説明する )

public void init(){}
public void paint(){}
public void run(){}
public boolean action(){}
}
```

1. クラス定義

   import 文により API パッケージを利用可能にする。ただし、mywindow はフレームクラスを継承させたユーザクラスである。本アプレットでは, Runnable インタフェースを実装しており、クラス定義部分で宣言している。変数の定義部分では、変数、インスタンスなどを定義している。

2. init() メソッド

   アプレットを起動すると最初に実行されるメソッドである。また、このメソッドは自動的に実行されるものである。さらに、GUI 部品　(ボタン、テキストフィールド) の生成およびレイアウト、フォントの設定、背景色の設定などを行っている。

3. paint() メソッド

   実際に描画を行うメソッドである。このメソッド内には、drawImage() メソッドを用いて袋の絵の取り込み、fillRect() メソッドを用いて円の描画、drawString() メソッドを用いた文字の描画を記述してある。、drawImage() メソッド、fillRect() メソッドの引数や条件文には、run() メソッドで使う変数や配列を設定している。よって、run() メソッドでのループの度に paint メソッドは呼び出されるので、これによりアニメーションを実現できる。

4. run() メソッド

   袋の制御を行うものである。public boolean action() メソッド内に記述された start() メソッドによって、run() メソッドが呼び出され処理が移る。run() メソッドでは袋の

動きを制御している。また、Thread.sleep() メソッドにより paint() メソッドを呼び出すタイミングを調整している。

5. action() メソッド

この中ではボタンをクリックしたときに起こる動作を定義している。『SET」ボタンをクリックするとテキストフィールドから渡された文字列をキャストし、キャストされた文字列型の配列を paint() メソッドに渡し、描画する。『START」ボタンをクリックするとスレッドが生成され、start() メソッドにより処理が run() メソッドに移る。『STOP」をクリックすると suspend() メソッドによりスレッドが停止し、もう一度 『START」ボタンをクリックすると、スレッドが一時停止している所から再度スレッドが走り出し、run() メソッドで処理が再開される。。『はじめに」ボタンを押すと、mywindow クラスを呼び出し、アニメーションの説明を載せたウインドウを表示する。