卒業論文

並列処理学習支援のための
Javaによる可視化[1]

氏名：浅野 剛史
学籍番号：2210950003-1
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。これから並列処理を学ぶ人を対象に初心者でも並列処理をイメージとしてつかみやすいことを目的とする。手法としてJavaを用いてアプレットとして作成したのでホームページ上にも公開できるよう配慮した。

本論文では、まず、並列処理について述べた後、Javaの基本構造とJavaアプレットによる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、並列処理学習支援の実現のため、「並列処理を学習しようとする初心者の理解を助ける」、という観点で研究を行っている。

目 次

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と思われる。このような改善はいずれ信号伝播速度の限界である光の速さという壁を超えることは出来ないでしょう。そこで私は、並列処理による実行性能の向上に興味を持ちました。しかし、並列処理を学習するときになんらかの本を１冊読めば並列処理が逐次処理より計算速度が速く、計算時間が少なくて済むから時間の節約ができて、より多くの計算ができてとても便利なことはわかります。しかしそれでは、なんとか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得られるのか、そうではありません。並列化にもその性能向上を妨げるいくつかの点があります。それを踏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とわかります。　そこで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並列処理を学習する多くの方にわかりやすいようなものをと思い、アニメーションを作成した。手法としてはJavaを用いてアプレットとして作成した。

Javaの言語仕様はC++の様なオブジェクト指向言語であり、最大の特徴はJavaがプラットフォーム独立ということです。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す。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す。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ります。

また、アプレットは、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を言います。

さらには、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HTMLの中から呼び出すことができ、HTMLにはできない処理たとえばブラウザにページを表示したあとでその表示位置を移したり、データの入力やイベントにより処理の流れを変えるといったアクティブな動作ができます。Javaの機能として、主に、１．グラフィックスの描画、２．マウス・イベント、キー・イベントなどのイベント処理、３.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、４．マルチスレッドによるイメージのスクロールやアニメーション、５．時間機能の利用、６．メディアトラッカー、サウンドなどのマルチメディア、７新しいフレーム、ダイアログボックスの作成などがあげられます。 JavaはSun Microsystems社が1995年に発表した言語です。

実現方法としては、通信のオーバーヘッドによる問題、逐次処理部分のオーバーヘッドによる問題さらに負荷分散のばらつきによる問題について、四角を異なる色で塗り別ける

ことを処理に見立てたアプレット及び、加算処理とその加算回数を議論のベースとし、それぞれの問題点のイメージをつかみやすいようにしたアプレットによって表現し、並列処理を学習するにあたり理解をよりスムーズにできるようなものを、と心掛け作成した。

本論文では、2章で並列処理について、3章で並列処理イメージの可視化、4章で可視化の実現とその評価を行い、さらに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　並列処理の目的と形態

2.1.1　目的

並列プログラムを書く目的は主に次の 3 点が挙げられる。

- 高性能追求
- プログラミングが楽になるため
- 本質的に並列なものを扱うため

最初の理由は、「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。第 2 の理由は、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。

2.1.2　SIMD と MIMD

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、本論文では現在主流である MIMD 方式に着目していく。

2.1.3　スレッドとマルチスレッド

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。

図１ :マルチスレッド

また、本論文で述べる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。

2.2　MIMD方式における並列化

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。

（a）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。

（b）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。

(c) 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、異なるものをNUMAと呼ぶ。

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[3]。

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当然ながら、逐次プログラムにはない他のプロセッサとデータを通信することが必要である。このための時間は無論逐次プログラムにはないオーバーヘッドとなる。プロセッサ間の通信速度はマイクロプロセッサの性能向上に追い付いていないためこの問題は深刻である。

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。

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、実際には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。

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

1. Javaとは

 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。Javaの概念図を図２に示す。

 最近では、WebといえばJava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）などの基盤として脚光を浴びている。

図２ :Javaの概念

2. アプレットの概念

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

3.2 Javaの基本構造

1. JavaAPI

   Javaでプログラムを作るには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(Application Programming Interface)と呼ぶ。APIは表1に示すように8種類のパッケージに分けて管理されている。

2. APIクラスの階層

   APIクラスは、java.lang.Objectクラスをベースにして階層構造(継承を重ねたサブクラスほど下位の階層になる)になっています。下にAPIのクラス階層の一部を図4に示す。

表１　:JavaAPIのパッケージ
(APIを利用するには、利用したいクラスをimport文でロードする。)

図４　:APIのクラス階層

3. ユーザークラスの定義

ユーザークラスの定義は `class` 文を用いて次のように行う。

```
[ 修飾子 ]  class  ユーザクラス名  [ extends  スーパークラス名 ]
        [ implements  インターフェース名 ] {
    変更するメソッドの定義 {
        .
        .
    }
}
```

図５ :クラスの継承と実装

3-a 修飾子

通常 pblic を指定します。pblic を指定したクラスは、そのクラスのパッケージ外からもそのクラスを参照できます。pblic が指定されていないと、そのクラスはクラスのパッケージ内からしか参照できません。

3-b 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びます。ユーザークラスはスーパークラスの機能を継承します（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します。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びます。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します。

3-c インターフェース

C＋＋などの一般的なオブジェクト指向言語では、複数のクラスを多重継承できるが、プログラム構造が複雑になるため、Javaでは1つのクラスの継承しか認めないようにしてあります（単一継承）。この代用としてインターフェースが用意されています。インターフェースは単一継承するクラスに付加的なメソッドを実装（implements）し（図5参照）、機能拡張をはかるものです。インターフェースは、クラスのデータ部を持たないメソッドだけを定義した一種のクラスと考えることができます。

4. クラス、オブジェクト、メソッド

importでロードしたクラスはユーザークラスのスーパークラスとして使用したり、次のようにオブジェクトを生成するのに使います。イメージを図6に示す。

```
import java.lang.*;

String msg1,msg2;
```

オブジェクトを操作するためメソッドを使います。使用できるメソッドはオブジェクトのクラスごとに決められています。たとえばStringクラスのsubstringメソッドには次のように使います。

```
msg1=”Hello”;

msg2=msg1.substring(0,3);
```

図6 :オブジェトの生成

5. 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7に示す順序でメソッドが呼び出されます。ユーザは必要に応じて各メソッドの定義内容を変更します。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ます。

図7 :アプレットの動作順序

3.3 色塗りによるイメージの可視化

2.3章の 通信のオーバーヘッドによる問題について、正方形を4つに区切ったそれぞれの領域を互いに異なる色で塗り分けるという作業を処理にみたて、1台と4台で処理した時を例にアプレットを作成した。

逐次処理では他のプロセッサとデータを通信する必要がないため、正方形を1色目から順に4色目まで塗り分けて処理を終了します。一方、並列処理ではまずそれぞれのプロセッサにデータを受け渡すところから開始します。正方形を4等分しそれぞれに送ります。

また、通信時間を3段階 (SLOW、MIDDLE、FAST) に設定しそれぞれにおける並列効果を見ることができます。

実現方法を以下に示す。

（1）部品機能

Button クラス（java.awt.Button）で start , stop , REstart ボタンを作成。ボタンのクリックで action イベントが発生し、それを Boolean クラスで判定し、スレッドが停止/再開するようにした。また、Checkbox クラスで、slow,middle,fast 3つのチェックボックスを作成し、通信時間を3段階に設定できるようにした。なお、チェックボックスは、CheckboxGroup によってグループ化し、グループ内のチェックボックス1つだけが選択されるようにした。

（2）表示方法

Graphics クラス（java.awt.Graphics）において、drawRect メソッドで色塗りの対象となる四角を描画。色が塗られていく様子は、drawLine メソッドで描画した直線の始点座標を run でパラメーターとして増加させ、移動さしている。同様にデータの受け渡し（通信）の様子も run メソッドで始点の座標を動かすことで描画した。

3.4 加算処理によるイメージの可視化

2.4 章の逐次処理のオーバーヘッドと、2.5 章の負荷分散のばらつきによる問題について、A~J の１０個の数値の加算を１台と３台のプロセッサで行った場合を例に比較しています。逐次処理では A~J の１０個の数値を順に加算します。並列処理ではまず、１０個の数値をそれぞれ、３個、３個、４個にして３台のプロセッサに振り分け、部分和を求めます。次に、部分和の合計を求め、１０個の数値の合計を求めます。なお、議論を簡単にするために演算時間としては加算の回数をベースとします。また、アプレット内でマウスをクリックすることにより、動作の停止/再開が可能となっています。

実現方法を以下に示す。

（1）表示方法

Graphics クラス（java.awt.Graphics）で setFont を用い、フォントを設定。drawString メソッドで、A~J の文字列を描画している。また、drawRect、fillRect メソッドで処理の時間を示すメーターを作成した。

（2）加算処理の様子の描画

まず、length メソッドで文字列の長さを取得する。A + B + C +…、の文字列の場合、偶数番目の文字が「+」にあたる。このことを利用して、run メソッドで偶数番目の文字を取得し、paint メソッドで文字を赤く表示（setColor）し、加算が行われていることを示した。同時に、１つ前の「+」の文字を取得し、paint メソッドで赤から黒に戻している。

（3）マウスイベント

mouseDown メソッドを用いて、マウスによるイベント処理を行っている。クリックのたびに true/false を反転させ、スレッドの停止（th.suspend）、再開（th.resume）を可能にした。

4　実現と評価

4.1　結果

１．色塗りのアプレット実行画面を以下の図８～１０に示す。アプレット上部のボタンはstartをクリックするとアプレットが開始し、stopを押せば一時停止、REstartで動作が再開する。また、チェックボックスで通信の時間を３段階に設定できます。なお図８～１０は通信時間MIDDLEの場合です。

(1) 処理開始。

- 並列処理では、まず１／４に区切った１片をそれぞれのプロセッサに渡します。
- 逐次処理の方は１つ目の四角を塗りはじめています。

図８　:アプレット１

(2) それぞれで処理中...

- 並列処理は４つともすでに塗り終りかけています。
- 逐次処理はまだ２枚目を塗っている途中です。

図９ :アプレット１

(3) 処理がおわりもとに返します。

- 並列処理はもうすぐ元に戻って終りです。
- 逐次処理はまだ３枚目を塗っている途中です。

図１０ :アプレット１
(2.3 章の通信のオーバーヘッドによる問題について)

速度向上について

a）通信時間 SLOW のとき速度向上０．８倍となり、並列効果が得られない場合もあることを示した。

b）通信時間 MIDDLE のとき速度向上１．３３倍となり、並列効果が少し得られた場合を示した。

c）通信時間 FAST のとき速度向上２．７倍となり、理想の４倍とは差があるがそこそこの並列効果が得られた場合を示した。

２．加算処理のアプレット実行画面を図１１に示す。アプレット上部が逐次処理、下部が並列処理を表す。今現在、加算処理が行なわれている部分にあたる「+」の文字をを順に赤色に変えていく。X、Y、Z はそれぞれ A + B + C、D + E + F、G + H + I + J の部分和を表し、さらに最後に１箇所で合計し１０個の和 SUM を求める。なお、アプレット右部のメーターは処理にかかった時間を示すように作成されている。また、アプレット内でマウスボタンをクリックする度にアプレットが一時停止、再開をくり返す。

1 2

3 4

5 6

…9

図１１ :アプレット２
(2.4 と 2.5 章の逐次処理部分のオーバーヘッドと負荷分散のばらつきについて)

速度向上について[2] （アプレット２）

１０個の数値を加算するのだから加算回数を数えると
逐次処理の場合は、１台で９回加算すればよい。

３台での並列処理では、

a）それぞれ、担当の範囲について部分加算を行う。
b）部分和を１台に送り同期をとる。
c）部分和の加算を行う。

ここで、計算機性能を

・1加算にかかる時間を1ユニットとする。
・転送のオーバーヘッドを0ユニットと仮定する。
・同期のオーバーヘッドを0と仮定する。

で比較する。

逐次処理にかかる時間は9ユニットである。

並列処理にかかる時間は

a）部分加算に要する時間は、それぞれ2ユニット・2ユニット・3ユニット
b）部分和の転送にかかる時間は、今回は0ユニット

a)+b）ここで同期をとるため、ここまでにかかる時間を集計すると最も時間がかかるものに引っ張られるので3ユニットがここまでの時間となる。

c）部分和の加算にかかる時間は2ユニット
合計5ユニットかかることになる。

したがって、３台で行った時の速度向上は

```
  １台で実行した時間      9
--------------------------------- = １．８
  ３台で実行した時間      5
```

となる。（理想的には3倍になってほしい。）

ここで、a)+b）で一番遅い時間まで引っ張られたのが 2の負荷分散のばらつき。
c）で1台だけが実行する時間が 1の逐次処理のオーバーヘッドである。

4.2 評価

本研究の目的として、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」についてその理解の手助けを目指したことについては、初心者でも理解を助けるものになったと思われる。

しかし、アプレット2に関して、数値として代わりにアルファベットを使用したが加算処理を例にしてるため、実際の数値を使用した方がわかりやすいと言う意見もいただいた。この点については、アプレットのプログラム上「１０」が1と0になってしまいうまく動作しなくなるという問題があった。しかし数値を使った方がわかりやすいと思うので、改良の必要があると思われる。

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ということで同じ研究室の石丸 智和氏と協力し、Java を用いてアニメーションを製作した。Java を学習するとともに、並列処理を初めて学ぶ人への導入を目的としたため、ユーザーインターフェースを重点に考え研究してきた。

今後の課題として作成したアプレットの問題点を以下に挙げる。

- 並列処理は処理問題の規模が大きいほど、その効果を発揮するが、アニメーションとして処理問題の規模が大きい場合を表現できなかった。
- プロセッサの処理能力の違いや、通信時間などの細かい設定などが不十分である。

上記の点をふまえた上での今後の課題としては、処理する問題の大きさを変えたり、各プロセッサの処理速度の違いなども設定できるようにするなど改良の余地がある。また、並列効果を理想的に得られた状況も理解できると、より深い理解が得られると思われる。

謝辞

本研究の機会を与えて下さり、ご指導をいただきました山崎勝弘教授に深く感謝いたします。 また、本研究にあたり、貴重な御意見や、アドバイスを頂きました本研究室の米田健治氏ならびに大学院生の方々に深く感謝いたします。

さらに、共同研究者の石丸 智和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] 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

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

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

[5] 柴山 潔 : コンピュータアーキテクチャ , オーム社 , 1997

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

[7] 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

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

[9] 石丸 智和 : 並列処理学習支援のための Java による可視化 [2], 立命館大学理工学部情報学科卒業論文,1999

付録

本研究で作成した 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

Java コンパイラを使ってソースファイルをコンパイルすると、バイトコードと呼ばれるアプレットを実行するのに必要な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、ブラウザに読み込まれるファイルは、ソースファイルでなくこのバイトコードのファイルです。バイトコードのファイル名は、拡張子.class を付けたクラスファイルです。アプレットを組み込みには、HTML ファイルにアプレットタグ< APPLET >< / APPLET >を用いてアプレット情報を記述します。

１．色塗りのアプレット

アプレットを操作するには、まずアプレット上部のチェックボックスで、通信の速度を選択します。次に「START」ボタンをクリックすると、アニメーションが開始します。アプレット動作中に「STOP」ボタンをクリックするとアニメーションが停止し、「RESTART」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。

２．加算処理のアプレット

アプレットのフィールド内で、マウスをクリックするとアニメーションが開始します。アニメーション動作中にマウスをクリックすると動作が停止し、途中の様子が観察できます。再び、マウスをクリックすると動作が再開します。

内部仕様

変数

１．色塗りのアプレット

int 型の変数 a,b,c,d を定義

`a,c` ： 領域の移動（通信）
`a,b,c,d` ： 逐次処理の様子
`b` ： 並列処理の様子

２．加算処理のアプレット

int 型の変数 noA,widthA,widthB,widthC,widthD,i,j,k,l を定義
String 型の変数 msgA,msgB,msgC,msgD,cenA,cenB,cenC,cenD,
befA,befB,befC,befD を定義

`msg＊` ： 処理する文字列を取得
`no＊` ： 文字列の長さを取得
`cen＊` ： 赤く表示する文字を取得
`i,j,k,l` ： アプレットの動作制御

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．色塗りのアプレット

（１）部品機能

Button クラス （java.awt.Button）で start , stop , REstart ボタンを作成。ボタンのクリックで action イベントが発生し、それを Boolean クラスで判定し、スレッドが停止／再開するようにした。また、Checkbox クラスで、slow,middle,fast ３つのチェックボックスを作成し、通信時間を３段階に設定できるようにした。なお、チェックボックスは、CheckboxGroup によってグループ化し、グループ内のチェックボックス１つだけが選択されるようにした。

(2) 表示方法

Graphics クラス(java.awt.Graphics)において、drawRect メソッドで色塗りの対象となる四角を描画。色が塗られていく様子は、drawLine メソッドで描画した直線の始点座標を run でパラメーターとして増加させ、移動さしている。同様にデータの受け渡し(通信)の様子も run メソッドで始点の座標を動かすことで描画した。

２．加算処理のアプレット

(1) 表示方法

Graphics クラス(java.awt.Graphics)で setFont を用い、フォントを設定。drawString メソッドで、A~J の文字列を描画している。また、drawRect、fillRect メソッドで処理の時間を示すメーターを作成した。

(2) 加算処理の様子の描画

まず、length メソッドで文字列の長さを取得する。A + B + C +…、の文字列の場合、偶数番目の文字が「+」にあたる。このことを利用して、run メソッドで偶数番目の文字を取得し、paint メソッドで文字を赤く表示(setColor)し、加算が行われていることを示した。同時に、１つ前の「+」の文字を取得し、paint メソッドで赤から黒に戻している。

(3) マウスイベント

mouseDown メソッドを用いて、マウスによるイベント処理を行っている。クリックのたびに true/false を反転させ、スレッドの停止(th.suspend)、再開(th.resume)を可能にした。