Visual ℃: プログラム負荷に対する内省を促す可視化手法

山下　洋毅[†]　　宮下　芳明[†]

Visual ℃: A Visualization Method to Generate Self-Reflection for Program Load

HIROKI YAMASHITA[†]　　HOMEI MIYASHITA[†]

1. はじめに

コンピュータの性能が日進月歩で成長する今日でも，なるべく負荷をかけず軽快に動作するプログラムが求められている．しかし，プログラミングの中級者ですらコンピュータにどれだけの負荷を与えているのかを考えず，とりあえず動作するソースコードを記述してしまうケースは多い．これは，軽いプログラムでも重いプログラムでも，見た目は同じようにコンピュータに実行させることができてしまうためである．

そこで本稿では，記述したプログラムの実行量を可視化し，プログラムがコンピュータにどれだけ負荷を与えているのかをプログラマに提示して内省を促すことを目的とする．プログラム実行量の可視化手法として，サーモグラフィのように色相を変化させる着色をソースコードに行うことで表現する．

人間の身体や機械，そしてもちろんコンピュータも，無理をするとその箇所が発熱する．こうした熱を可視化する方法の 1 つにサーモグラフィがある．サーモグラフィは，物体から放射される赤外線を色相の分布図として表示することで，どこが熱くどこが熱くないのかが一目瞭然に見ることができる．サーモグラフィは身体の異常診断や機械の安全検査など多くの局面で活用され，またその映像はテレビ等のメディアで目にする機会も多く，一般の人々にとってなじみが深い．

本稿では，サーモグラフィ表示を用いることで，プログラムがコンピュータにどれだけ無理をさせているか（すなわちどれだけ負荷を与えているか），またどこが負荷を与えているかを直感的に理解させるシステム，Visual ℃を提案する．

本稿の第二筆者らは，視覚デザインツールにおいて，編集中の画像における美観要因（輝度分散－サイズ分散）をサーモグラフィ表示して支援を行う研究 [1] や，音楽の即興演奏において不協和度に応じた温度提示を行うインタフェースを用いた支援 [2] を行っている．これらのシステムでは両方とも，美観を損ねる，あるいは不協和な響きを得る，といったネガティブな要因が大きくなるほど「熱い（赤い色相）」ものとして提示している．本稿でも，コンピュータに対する負荷が高いほど温度が高くなるメタファを用いたシステムデザインを採用している．

2. 関連研究

プログラミング支援についての研究は数多く行われており，ソースコードの可視化はその学習や理解に有効であることが知られている．嶋田らは，C 言語を対象にフローチャートと関数関連図で表現するシステムを開発した [3]．武田らは，アルゴリズムの可視化を行うために，構造のフローチャートによる自動表示と各変数値の変化の様子を可視化する機能を基本として，変数の動的な様子をグラフ表示する機能，関数の呼び出し関係の木構造表示，フローチャートの簡易表示などの拡張機能を開発した [4]．鈴木らは，平面上に構造情報を可視化し，その平面の奥行き方向に履歴を表現することで 3 次元的に関数実行履歴を可視化するツール SEIV を開発した [5]．他にも，設計段階で用いられる抽象度を持った図形を用いて可視化するもの [6]，変数の基本的処理のみを可視化するもの [7]，ソーティングと木構造に関するアルゴリズムのみを可視化するもの [8]，並列言語 linda のプログラムの実行状態を 3 次元可視化するシステム VisuaLinda[9] などがある．

これらの研究は，どれもプログラム構造を直感的に理解できるようにするための可視化と支援である．別のアプローチとして，奥田らは，プログラムの並列化支援可視化システム NaraView のデータ依存ビューを用いて，配列参照情報を可視化し，並列化・最適化を

† 明治大学理工学部情報科学科
Department of Computer Science, Meiji University

プログラマに促す研究を行っている [10]．三吉らは，逐次型プログラムの並列化の際に必要となる，ユーザによるデータ分割戦略の決定を支援するシステムを開発した [11]．松崎らは，ユビキタス環境におけるプロセス配備の戦略を扱うフレームワークを提案し，プロセスの実行効率化により増加する負担の軽減を図っている [12]．林らは，ソフトウェアメトリクスの値の変化をコードエディタ上に可視化することにより，開発者のプログラム変更を支援する手法を提案している [13]．

プログラムの実行速度を速くする手段としてプロファイルがある．これは，プログラムの実行回数やCPU 時間の情報を解析することを指す．実行速度を考慮してソフトウェアを開発する場合は，このプロファイルを用いることで，プログラムのどの部分に時間がかかっているかを解析し，コンピュータになるべく負荷を与えないようなプログラムを記述するようにする．窪田らは，実行時に得られるプロファイル情報などを用いて，実行時にプログラムを書き換える性能最適化手法を提案している [14]．神尾らは，オブジェクト指向プログラムの実行時に，各メソッドが同じ値の引数で何回呼び出されたかを調べるプロファイラを作成した [15]．筆者らが提案する Visual ℃も一種のプロファイルであるが，より直感的な可視化手法を目指している．

3. システム概要

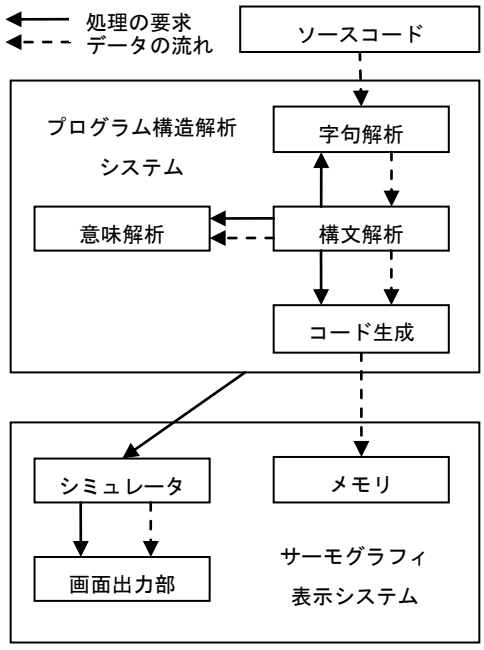

図1 システム概要図

本稿で提案する Visual ℃のシステムは， C 言語のサブセット言語を対象とし，その実行量をサーモグラフィ表示して可視化するシステムである．図 1 のように，プログラム構造解析システムとサーモグラフィ表示システムの 2 つのシステムで構成されている．ソースコードのサーモグラフィ表示は以下のような手順で行われる．

① C 言語プログラムのソースファイルを読み込む.
② そのソースファイルに記述されたプログラムの構造を解析し，目的コードに変換する.
③ 目的コードを実行して，その目的コードにあたるソースコードの行の色をその行が熱くなったように色相変化させる.

プログラム構造解析システムは①および②を行い，サーモグラフィ表示システムが③を行う.

3.1 プログラム構造解析システム

プログラム構造解析システムは，与えられたソースコードの構造を解析するシステムであり，字句解析，構文解析，意味解析，コード生成の 4 つで構成されている．プログラム構造解析システムの流れは，以下のようになっている.

① C 言語プログラムのソースファイルを読み込む.
② 字句解析が構文解析から処理の要求を受け，字句解析を行い，トークンを構文解析に返す.
③ そのトークンを用いて構文解析を行い，木構造をした構文木を生成する．またこのとき，意味解析によってソースコード内の意味的な誤りがないかどうかの検査も行う.
④ 構文木から目的コードを生成する.

3.1.1 字句解析

字句解析は，ソースコードをトークンの並びにまとめることである．まずソースコードを 1 文字単位で読み込み，字句ごとにまとめる．次にその字句が何を意味しているのかを解析し，解析結果をトークンとして構文解析に渡す.

3.1.2 構文解析

構文解析は，字句解析によって得られたトークンからプログラムにおける文法上の構造，すなわち構文を明らかにすることである．具体的には，字句解析からトークンを得て，プログラム内に文法的な誤りがないかどうかを検査し，木構造をした構文木を生成する．またプログラム内で宣言または定義された識別子は，識別子の名前とそれに付随する情報を記号表と呼ぶ表に登録して管理する.

3.1.3 意味解析

意味解析は，プログラム内に意味的な誤りがないか

どうかを解析することである．これは主に型の使用に誤りがないかどうかの検査，すなわち型検査であり，構文木の式や関数呼出しについて，記号表を参照しながら行う．他には，未定義の関数や変数の検査，識別子や分岐文の不正な使用の検査なども行う．

3.1.4 コード生成

コード生成は，構文木から目的コード，つまり目的プログラムを生成することである．このプログラム構造解析システムは，式や文を単位として意味解析が終了したらすぐに目的コードを生成する．そして，生成された目的コードと，該当するソースコードの行番号，その行の実行量（このときは 0）をサーモグラフィ表示システムのメモリに格納する．したがって，ソースコードの読み込みが終了するとともに，そのソースコードに対する目的プログラムが完成する．

3.2 サーモグラフィ表示システム

サーモグラフィ表示システムは，目的コードを実行して，その目的コードにあたるソースコードの行を着色するシステムであり，メモリ，シミュレータ，画面出力部の 3 つで構成されている．サーモグラフィ表示システムの流れは，以下のようになっている．

① プログラム構造解析システムが終了すると同時に，メモリにシミュレータが実行する目的コードとその目的コードにあたるソースコードの行番号が全て格納される．

② シミュレータが目的コードを実行すると同時に，その目的コードにあたるソースコードの行の実行量（初期値 0）を増やし，その行の行番号と実行量を画面出力部に渡す．

③ 画面出力部が，受け取った行番号と行の実行量を用いて，該当行を着色する．

3.2.1 メモリ

メモリは，プログラム構造解析システムのコード生成のときに用いられ，シミュレータが実行する目的コードとその目的コードにあたるソースコードの行番号が格納される．

3.2.2 シミュレータ

シミュレータは，プログラム構造解析システムが終了次第実行され，メモリに格納された目的コードを実行する．シミュレータが目的コードを実行する際に，その目的コードにあたるソースコードの行に対して実行量を増やし，行番号と実行量を画面出力部に渡す．

3.2.3 画面出力部

画面出力部は，受け取った行番号と実行量を用いてソースコードの行を着色する．受け取った実行量が多ければ多いほど，色相を赤方に偏移させる．したがって，ソースコード 1 行あたりの目的コードが多ければ多いほど，その行がより赤くなっていく．画面出力部によって表示される結果を図 2 に示す．

4. 適用例

```
int fibo(int n)                  R:   0
{                                R:   6
    int i, f1, f2, f0;           R:   0
                                 R:   0
    f1 = 1;                      R:   2
    f2 = 1;                      R:   2
    f0 = 1;                      R:   2
    for(i = 3; i <= n; i++)      R:  38
    {                            R:   0
        f0 = f1 + f2;            R:  32
        f1 = f2;                 R:  16
        f2 = f0;                 R:  16
    }                            R:  48
    return f0;                   R:   2
}                                R:   0
                                 R:   0
void main()                      R:   0
{                                R:   5
    int i;                       R:   0
                                 R:   0
    i = fibo(10);                R:   3
    printf("%d", i);             R:   3
}                                R:   4
                                 total run:  179
```

(a) 通常版

```
int fibo(int n)                  R:   0
{                                R: 654
    if(n <= 2)                   R: 327
    {                            R:   0
        return 1;                R: 110
    }                            R:   0
    else                         R:   0
    {                            R:   0
        return fibo(n-1)+fibo(n-2); R: 432
    }                            R:   0
}                                R:   0
                                 R:   0
void main()                      R:   0
{                                R:   5
    int i;                       R:   0
                                 R:   0
    i = fibo(10);                R:   3
    printf("%d", i);             R:   3
}                                R:   4
                                 total run: 1538
```

(b) 再帰版

図2 フィボナッチ数列のソースコード

開発したシステムを，フィボナッチ数列を求めるプログラムに対して適用した例を以下に示す．

フィボナッチ数列とは，以下のように定義する数列 f_n のことである．このとき，n は 1 以上とする．

$$f_n = \begin{cases} 1 & (n \le 2) \\ f_{n-1} + f_{n-2} & (n \ge 3) \end{cases}$$

今回，このフィボナッチ数列 f_n を求めるプログラムとして，for 構文による繰り返しを用いた通常版と再帰を用いた再帰版の 2 種類を用意した．図 2 に，フィボナッチ数列 f_{10} の値を求めその値を出力する通常版と再帰版のソースコードと，それを読み込んだ時のシステムの実行結果を示す．各行の横には，その行の実行量 R を記した．またソースコードの最下部にはプログラムの総実行量 total run を示した．

この２つのプログラムは，一見すると考え方が違うだけのバリエーションであって，むしろ再帰版のほうがスマートなソースコードにすら感じられる．実際，インデックスが小さいうちはプログラムを実行しても両方とも短時間で結果を返す．しかしインデックスが大きくなってくると，再帰版の方が通常版より遥かに時間がかかってゆく．10 項目を計算させている図 3 の（a）と（b）の total run を比較すると再帰版の実行量が通常版の 10 倍近くになっており，この差はさらに開いていく．これはもちろん再帰版の計算に大きな無駄が含まれているからであり，提案システムによる可視化を行えば，その実行量の差を明示できるとともに原因の究明にも役立つといえる．

5. おわりに

本稿では，プログラムがコンピュータに与える負荷に対して内省を促すために，プログラムの実行量をサーモグラフィ表示するシステムを提案した．これを用いることで，記述したプログラム内の無駄に動作している場所を直感的に理解することができた．

このシステムの有効性を評価するために，明治大学理工学部情報科学科 1 年生を対象としたプログラミングの実習授業で使用する予定である．これに合わせ，プログラムの実行量が多かった場所を直した後に，プログラムが改善されたことを提示する機能，及びなぜ実行量が多いのかをプログラミング初心者でもわかるように提示する機能の実装を行っている．

参 考 文 献

1) 宮下芳明，西本一志，宮田一乘: 画像構成要素の輝度-サイズ分散を用いた視覚デザイン支援手段の検討，インタラクション 2006 論文集，pp. 117-124 (2006).
2) 宮下芳明，西本一志: 温度で制約を緩やかに提示するシステム Thermoscore を用いた即興演奏支援，情報処理学会研究報告 ヒューマンインタフェース 2004-HI-110，Vol. 2004，No. 90，pp. 13-18 (2004).
3) 嶋田一行，葛崎偉: フローチャートと関数関連図によるプログラムの視覚化，電子情報通信学会総合大会講演論文集，pp. 444-445 (2000).
4) 武田寛昭，山下英生: C 言語プログラムに対するアルゴリズム可視化システム，広島工業大学紀要．研究編，Vol. 42，pp. 247-253 (2008). http://harp.lib.hiroshima-u.ac.jp/handle/harp/860
5) 鈴木宏紀，山本晋一郎，阿草清滋: プログラム実行情報の視覚化による理解支援ツール，情報処理学会研究報告 ソフトウェア工学 120-12，Vol. 1998，No. 64，pp. 77-84 (1998).
6) 市川至，小野越夫，毛利友治: プログラム可視化システム，情報処理学会論文誌，Vol. 31，No. 12，pp. 1801-1811 (1990).
7) 大森康正，上野晴樹: アルゴリズムアニメーションを用いたプログラミング教育システム，情報処理学会第 53 回全国大会講演論文集，pp. 4-287-288 (1996).
8) 黄寧，丸山桃代，宮寺康造，横山節雄: プログラム可視化によるプログラミング教育支援，電子情報通信学会技術研究報告 ET，Vol. 99，No. 31，pp. 1-6 (1999).
9) 高田哲司，小池英樹: VisuaLinda: 並列言語 linda のプログラムの実行状態の 3 次元可視化，竹内彰一編，インタラクティブシステムとソフトウェア II: 日本ソフトウェア科学会 WISS’94，pp. 215-223，近代科学社 (1994).
10) 奥田宗継，中西恒夫，笹倉万里子，城和貴，福田晃: 配列参照パターンによるプログラム並列化・最適化支援，情報処理学会研究報告 計算機アーキテクチャ 134-25，Vol. 1999，No. 67，pp. 145-150 (1999).
11) 三吉郁夫，森眞一郎，中島浩，富田眞治: プログラム並列化におけるデータ分割支援システム，情報処理学会第 47 回全国大会講演論文集，pp. 5-105-106 (1993).
12) 松崎和賢，本位田真一: ユビキタス環境における動的なプロセス配備のためのプログラミング支援フレームワーク，情報処理学会論文誌，Vol. 47，No. 12，pp. 3188-3202 (2006).
13) 林晋平，佐伯元司: メトリクス値の変化の可視化によるプログラム変更の支援，情報処理学会研究報告 ソフトウェア工学 2008-SE-159，Vol. 2008，No. 29，pp. 115-122 (2008).
14) 窪田昌史，阪口陽祐，津田孝夫: 実行時情報を用いた性能最適化手法，情報処理学会研究報告 ハイパフォーマンスコンピューティング 77-5，Vol. 1999，No. 66，pp. 23-28 (1999).
15) 神尾貴博，増原英彦: オブジェクト指向プログラムの高速化を支援するプロファイラ，情報処理学会論文誌：プログラミング，Vol. 46，No. SIG 1(PRO 24)，pp. 1-9 (2005).