リアルタイムビデオオーバーレイを用いた並列クイックソートの視覚化

Visualizing Parallelized Quick Sort with Realtime Video Overlay

川崎 高志 (36000714), 大向 一輝 (36000726)
Takashi KAWASAKI(36000714), Ikki OHMUKAI(36000726)

Abstract: Visualization is a great method to illustrate new paradigm, such as parallel processing. We made a visual system which illustrates parallelized quick sort using realtime graphical pazzle. In this paper, we explain this system in detail.

1　はじめに

本発表では, 先日行われたオープンキャンパスにおいて展示, 説明した「リアルタイムビデオオーバーレイを用いた並列クイックソートの視覚化」のシステムの概要について説明する. このシステムは, その最大の目標が, 楽しさであり, オープンキャンパス当日には, そのシステムの複雑さよりもむしろ, 単純に動画パズルが並ぶといった表面的なことに説明の主眼がおかれており, 並列のシステムであるということはそれが紹介されるにとどまっていた. この論文では, その詳細な説明を行うとともに, 今後の課題などについても考察する.

2　用いられたシステム

今回, 用いたシステムは, Pentium III(500MHz) x 2, メモリ 128MB のマシン 8 台で, OS として, Debian/GNU Linux (カーネル 2.2.15) がインストールされている. また, ビデオ入力のためのボードとして, サーバとなるマシンに Bt878 が搭載されたキャプチャカードを 1 枚搭載している.

3　Linux でのビデオ入力, 画像処理

Linux でのビデオ入力に関しては, カーネル 2.2.X では, Video for Linux(V4L)[1]が標準で提供されており, キャラクタデバイスとしてカーネルに組み込み, デバイスファイル (`/dev/video`) を作成するだけでほとんどの準備が完了する. また, その利用方法も非常に簡単であり, ビデオデバイスを開き, 画像サイズ, 受信形式 (NTSC,SECAM,PAL), チャンネルを指定すれば BGR 並び順の画像データが得られる. GUI フレームワークとしては, GTK1.2 を利用し, ビデオ入力と GTK の橋渡しには, Imlib を利用した.

4　マシン間通信

マシン間通信には, TCP/IP を用いた. これは, 通信ライブラリの MPI や PVI に比べて設定が容易なことが一つの理由である. しかし一方で, それぞれのノードでのプログラムの起動を個別に行う必要があるなどの問題もある. 起動に関しての問題は, rsh による起動を行うシェルスクリプトを記述することによって解決した (Fig. 1, Fig. 2). また, その際に X Window System の権限の問題が発生したが, これは`xauth` で適切に設定することによって解決した.

```
#!/bin/sh
export DISPLAY=localhost:0.0
/home/kawasaki/Video/ssort -S192.168.14.70 &
```

Fig. 1　bootclient スクリプト

```
#!/bin/sh
/home/kawasaki/Video/ssort
    → $1 $2 $3 $4 $5 $6 $7 $8 $9 -p4 -W2 &
sleep 2
rsh freund01 /home/i2k/bootclient &
sleep 2
rsh freund02 /home/i2k/bootclient &
sleep 2
rsh freund03 /home/i2k/bootclient &
```

Fig. 2　boot4 スクリプト

5　並列クイックソート

並列処理に適したソート手法としては, バイトニックソートが有名であるが, 今回のシステムでは, クイックソートを採用した. その理由としては, 仕組みが簡単であるということや, ソートの初期の段階を除けば, 並列処理しやすいということがある. また, GUI の処理にはリアルタイム性が要求されるため, 今回用いたクイックソートは, 通常見ることができるような再帰構造を取ることはできなかった. GUI ベースの処理では, 通常, イベントドリブンという手法が用いられるため, 特定の処

[1] http://roadrunner.swansea.uk.linux.org/v4l.shtml

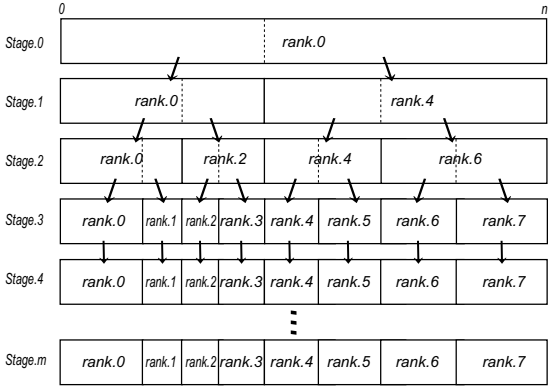

Fig. 3　並列クイックソート

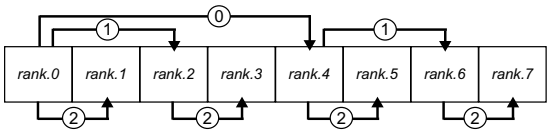

Fig. 4　通信負荷の軽減

理がプロセッサを占有することはできない. そして, 並列化の方法としては, 8 台の場合, 最初の 1 ステップは, 1 台, 2 ステップ目は, 2 台, 3 ステップ目は, 4 台, 4 ステップ以降は 8 台という処理分散の方法を採った. この場合, 最初のステップでは遊ぶプロセッサも出るが, ほとんど問題にならないレベルであると思われる.

6　画像の配信

今回のシステムでは, 画像を並べ替えて元の画像に戻すために, 画像をキャプチャできる 1 台以外のマシンでもキャプチャした画像を取得する必要がある. そして, 本来なら, それぞれのマシンが必要としている画像 (パズルのピースと考えるとわかりやすい) はその画像の一部分であるのだが, その判定処理のためには, ソート中のデータとの問い合わせの必要があるだけでなく, 処理としてかなりの負荷があると思われた. そこで, 今回のシステムでは, 特別な処理を行わずに, 同じ画像をすべてのマシンに送信した. また, キャプチャを行っているマシンが画像を配信せざるを得ないのだが, この負荷を少しでも軽減するよう, Fig. 4 に示すような通信の方法を採った. この図での◯で囲まれた数字はその通信が行われるべきフェーズを示している.

7　表示に関する問題

キャプチャされたデータは, rank.0 のマシンで BGR → RGB の変換が施された後, 画像の配信で説明された方法ですべてのマシンに配信される. その後, 各々のマシンは, ウィンドウに対しての描画を行うが, この時

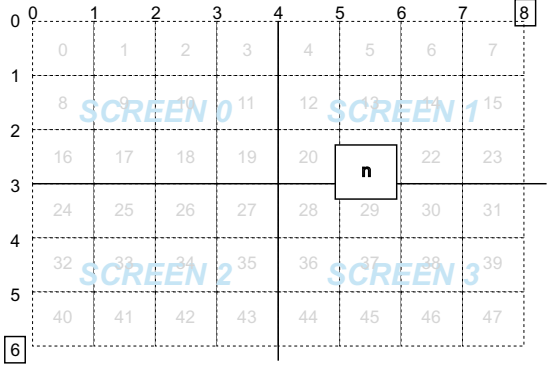

Fig. 5　仮想スクリーン

に`gdk_imlib_render()` という API を利用する. この際, 非常に莫大な時間がかかるため, 最終的なプログラム全体のリアルタイム性に影響を及ぼしている. また, ソートの処理は前に述べたとおりだが, あるマシンが表示すべきデータはそのマシンがソートを行っているデータとは無関係である. あるマシンが表示すべきデータはそのマシンの表示すべきエリアに存在するすべてのセルに対して行わなければならない. 数値 n を保持しているセルが表示すべきエリアは, Fig. 5 に示すような大きなスクリーンがあるとすれば, $X = nMOD8, Y = n/8$ の示すセルの画像になる. 実際には, 640x480 の画像を扱っているので, 1 セルは, 80x80 であり, (80X, 80Y)-(80X+79, 80Y+79) のエリアを表示する.

8　おわりに

今回のシステムはほぼ当初の目的を達成できたと考えている. Linux でもここまでのシステムを構築できると言うことを示したという意味もあると思われる. しかし, 細かな部分で言えば, もっと負荷分散を徹底できたのではないかということや, あるいはソートのやり方をもっと工夫できたのではないかなど, 問題とすべき点は多い. これらに関してはこれからの課題としていきたい.

参考文献

1) 奥村晴彦『C 言語による最新アルゴリズム事典』(技術評論社, 1991)

2) Neil Matthew, Richard Stones 葛西重夫 訳『Linux プログラミング』(ソフトバンク, 1999)

3) 飯尾淳『Linux による画像処理プログラミング』(オーム社, 2000)

4) 竹田英二『GTK+ではじめる X プログラミング』(技術評論社, 1999)

5) 田中ひろゆき『GTK+入門』(ソフトバンク, 1999)