A Distributed Image Processing Environment VIOS II

Hiroshi Matsuo and Akira Iwata
Dept. of Electrical and Computer Eng., Nagoya Institute of Technology
Gokiso, Showa, Nagoya, JAPAN 466

Abstract

We proposed a network computing software environment for parallel image processing, **VIOS**. It is composed of three kind of processes; Visual Programming Editor(**VPE**), Object Manager(**OM**), and Image Processing Unit(**IPU**). In this paper, the second version, **VIOS II** is proposed. In **VIOS II**, a new parallel processing language for **VPE** and a virtual processor for **IPU** have been developed. Utilizing these functions, fine grain level parallel programming has been realized. The performance of **VIOS II** are investigated by several image processing algorithms.

I. Introduction

There are a lot of high-end computers available today. However, even using those computes, image processing and image recognition algorithms still demand enormous computing time.

AVS, which supports remote execution module[1], allows large scale tasks to be performed by the large-sized computer or super super-computer which are connected by local area network. Then trivial tasks and display tasks can be done by a local workstation.

Recently, it is easy to get many workstations in one research group. By such a reason, a network computing software environment **VIOS** for parallel image processing has been proposed. This software environment uses many workstations which are connected by local area network at the same time. It is composed of three kind of processes; Visual Programming Editor(**VPE**), Object Manager(**OM**) and Image Processing Unit(**IPU**).

However **VIOS** could not give enough performance, because **VIOS** can not describe parallel section within a module. Moreover it required a relatively long time for generating a new **IPU** process and transferring image between two **IPU**s running on different workstations.

In this paper, **VIOS II**, the second version of **VIOS**, is proposed. In **VIOS II**, a new parallel processing language for **VPE** and a virtual processor for **IPU** have been introduced. Utilizing these functions, fine grain level parallel programming has been realized. The performance of **VIOS II** are investigated with several image processing algorithms.

II. The Architecture of VIOS II

The **VIOS** system is composed of three kind of processes named **IPU**, **OM** and **VPE** as shown in Fig.1. These processes run on different workstations which are connected by local area Network.

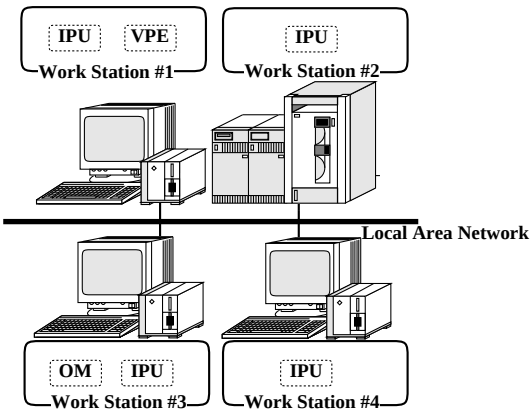

Fig.1: The structure of **VIOS**

VIOS II is an improved version of the **VIOS**. The following features has been introduced in **VIOS II**.

- New parallel language for image processing.
- Description of finer grain level parallel programming.
- **IPU** and **OM** which are implemented using "thread" (SUNOS calls it light weight process)

In **VIOS II**, a new concept for describing finer grain level parallel programming is proposed. In previous **VIOS** system, only modules can be executed in parallel. Module is a basic processing unit in **VIOS**. Input and output data of the module must be image objects. In **VIOS II**, an image object is divided into sub image objects, hence the same module can be executed at the same time by different workstations(**IPU**). For a program to process sub image object in parallel, it must be divide into many pieces, each of which should be worked on at the same time. But it is difficult to detect which part can be executed in parallel. Therefore in **VIOS II**, the programmer must specify whether the data is private or shared, and which part of the module program is parallelized.

III. Image partition for parallel processing

When the program in the module can be executed parallely, the image object is divided for distributed processing in **VIOS Ⅱ**. The type of image division can be classified as follows considering the required input data for processing in the module and independency for generating output data.(Fig.2)

pixel parallel each output pixel value can be calculated independently from other output pixel value. (ex. image subtraction, Laplacian filter etc.)

row parallel each output row or column value can be calculated independently from other output row or column value. (ex. matrix multiply, one dimensional Fourier transform etc.)

block parallel each output block value can be calculated independently from other output block value. (ex. image compression etc.)

non parallel parallel execution is impossible.

The neighboring pixel width needed for executing module is called "surrounding pixel width". For example, 3×3 stencil calculation like Laplacian operator needs 1 surrounding pixel width.

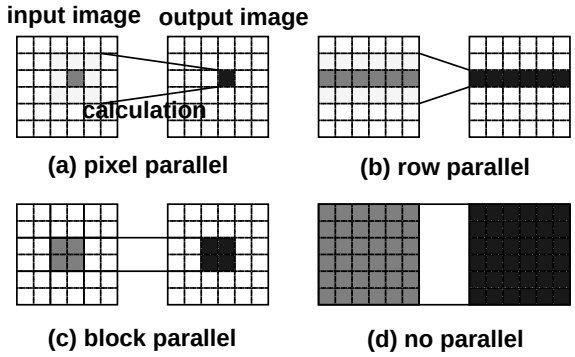

Fig.2: The classification of an image division of **VIOS**

IV. The FUNCTION OF THE THREE PROCESSES

A. Visual Programming Editor

VPE illustrated in Fig.3 is a user interface process for **VIOS Ⅱ**. A user can make a program by connecting each icons which implies module. When an image object is pointed, the image object is displayed in the window. Moreover, **VPE** has a programming language for image processing named VPE-P. Using this language, new modules can be added to the **VPE**.

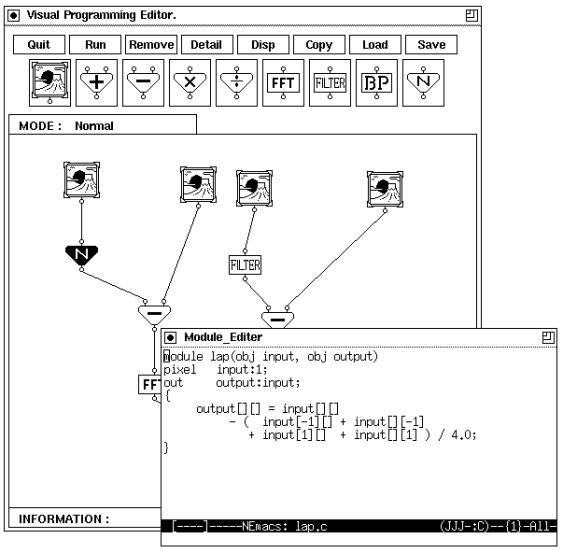

Fig.3: Programming using **VPE**

B. Image processing language VPE-P

VPE-P has C like syntax. A description of a new module of programming for image processing is possible using VPE-P. The data types which are supported by VPE-P are integer, float, complex and image object. As mentioned in the previous section, it is difficult to detect which part can be executed in parallel. Therefore in **VIOS Ⅱ**, the programmer must specify whether the data is private or shared, and where the codes are parallelized. Fig.4 shows a Laplacian filter program written in VPE-p.

```
module Lap(obj input, obj output)
pixel   input:1;
out     output:input;
{
      output[][] = input[][]
       - (  input[-1][] + input[][-1]
          + input[1][]  + input[][1]  ) / 4.0;
}
```

Fig.4: VPE-P program(Laplacian filter)

Module name is defined first. Second and third lines define division rule, pixel type of input and output image objects and surrounding pixel width. The remaining is the program part for this module.

C. Image Processing Unit

IPU is a virtual processing machine which execute instructions from **OM**. Many **IPU** can be located on a local area network. **IPU** consists of two kind of virtual processors named MP (management processor) and IP (image processor). The aim of MP is to define a new module and to create IP. IP which is created by MP executes the image processing algorithm that it receives from **OM**. Hence one MP and more than one IP are executed on virtual machine **IPU**.

Table.1: Operators of IPU-p

variable definition	def
stack operation	pop , exch , dup , roll
arithmetic operation	add , sub , mul , div , neg , mod sin , cos , tan , asin , acos , atan sqrt , exp , log , pow
logic operation	eq , ne , gt , ge , le , lt and , or , not , xor , true , false
flow control	if , ifelse , for , repeat
image attribution	width , height , waku , hotx , hoty type , itype , createobj , obj get , aget , put , aput
module definition	fdef
image operation	addi, subi, fft, filt, load, save, disp, join, split

The instruction of **IPU** is a stack based language like FORTH and Postscript which needs a small cost to interpret. IPU-p have two kind of facilities, one is module definition and the other is program execution. Table.1 shows operators of IPU-p. In addition to the module which is described by VPE-p, it is possible to translate it to C++, which is description language of **IPU** and adding it to the new "image operation".

```
/ADD 2 1 0
{   /output exch def
    /input2 exch def /input1 exch def
    output input1 waku input1 type
        input1 input2 2 itype createobj
    0 1 output height { /y exch def
      0 1 output width { /x exch def
        output x y input1 x y get
                  input2 x y get add put
      } for } for
} fdef
image1 :uranus:image2
(kekka) obj  ADD
```

Fig.5: Example program of IPU-p

Figure.5 shows a sample IPU-P program (image addition).

D. Object Manager

OM is a process which manages database of image objects and also manages **IPU**s.

Each facilities are described as follows.

- DATA BASE
  OM have the following information in order to schedule tasks. The following information are sent from each **IPU** and **VPE** .
  - **IPU** information
    The name of the workstation which execute **IPU**. Performance of each workstation. CPU load of each workstation.
  - **Image object information**
    Attribute of image object, location of image object, information of image division.
- COMPILER
  OM compiles VPE-P to IPU-P and send IPU-P to all **IPU**s. When **IPU**s receive the module program described by IPU-p, they store this program in their own process.
- SCHEDULER
  OM finds the best **IPU** according to the algorithm which we proposed[2], and distributes the image processing module.

V. Implementation using thread

IPU consists of some processes in **VIOS**. Creating new process using fork system call needs a lot of time. Furthermore, it needs complex procedure to communicate between inter processes(IPC)[3]. Performance of a control process (**OM**) is a key to the system performance improvement.

By such reasons, **OM** and **IPU** are implemented using thread. Creating thread and communication between thread need only small cost. Moreover some workstation with multi CPU system have a capabilities to run threads parallelly[4].

A. Implementation of **OM** *using thread*

Functions of **OM** are divided into communication, scheduling and compiling of which each function is implemented using thread. The database on **OM** is defined as a global variable. These threads can get information from the database easily. Fig.6 shows a relationship between each function implemented by thread.

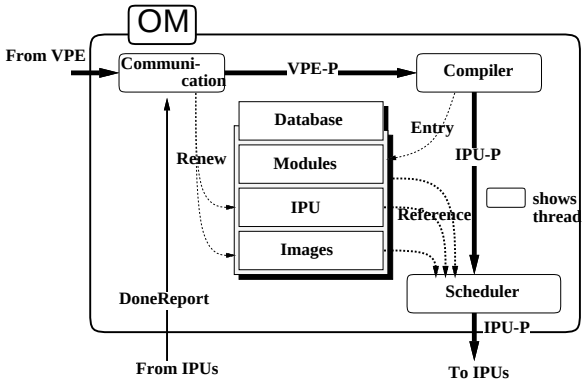

Fig.6: Relation between each function on **OM**

B. Implementation of **IPU** *using thread*

IPU consists of two kind of virtual processors named MP and IP. These two virtual processors are implemented by thread. A scheduler which controls the priority between each IP, communication thread and

transfer thread for transferring image object is also implemented by thread. Fig.7 shows a relationship between each function implemented by thread.

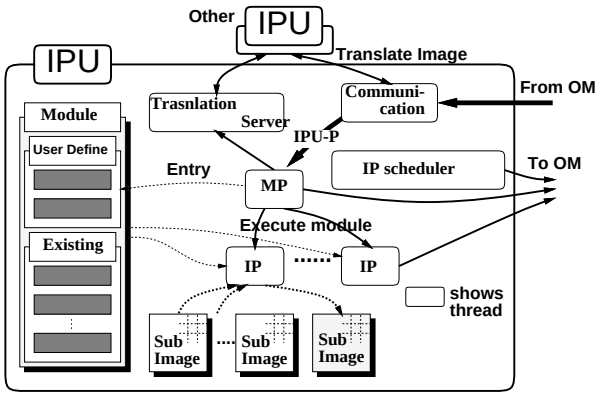

Fig.7: Relationship between each function on **IPU**

VI. Performance evaluation of **VIOS Ⅱ**

We implemented **VIOS Ⅱ** on SUN workstation. The evaluation of the **VIOS Ⅱ** system has been performed using this system. First, median filter operation and Laplacian filter operation (program 1) were programmed for evaluating the performance. This operation has less data transfer between each **IPU**s. The performance is measured by increasing the number of workstations which execute **IPU**. Median filter is a filter which calculate a median value in 7x7 pixel region, and Laplacian filter is calculated in 3x3 pixel region. These filter can be executed in parallel for each pixel. Image object is composed of 512 x 512 pixel matrix and the pixel type is float.

The second experiment program is two dimensional FFT program. Image object is also composed of 512 x 512 pixel matrix and pixel type is complex. This program is practically composed of 4 steps. First, image object is distributed to each **IPU**s on row unit. Second, one dimensional FFT from row direction is calculated with several **IPU**s. Third,each calculated result is collected and re-distributed to each **IPU**s on column unit. Fourth, one dimensional FFT from column direction is calculated with several **IPU**s. In first and third step, approximately 9MB of data are being transferred between **IPU**s.

Performance progression is shown in Fig.8. A solid line indicates a performance of program 1, dotted line indicates a performance of program 2. In program 1, as the number of **IPU** increases, performance also increases. Execution time is 3 times faster when running 5 **IPU**s on different workstation compared with one **IPU**. 5 times faster when running 10 **IPU**s. However by increasing the number of **IPU**, the communication overhead between each **IPU** becomes a large burden. Therefore, the performance would not be improved much when the number of **IPU** is increased more than 8 or 10 processors.

In program 2, as the number of **IPU** increases, performance does not increase much. This is because transfer time is not negligible compared to calculation time for FFT. As a result, not so high performance can be expected in a program which needs a lot of data transfer. But recently, a local area network system which has fast data transfer capability has been developed. Now FDDI is easy to get. If near feature ATM network can be use easily, then this problem can be solved.

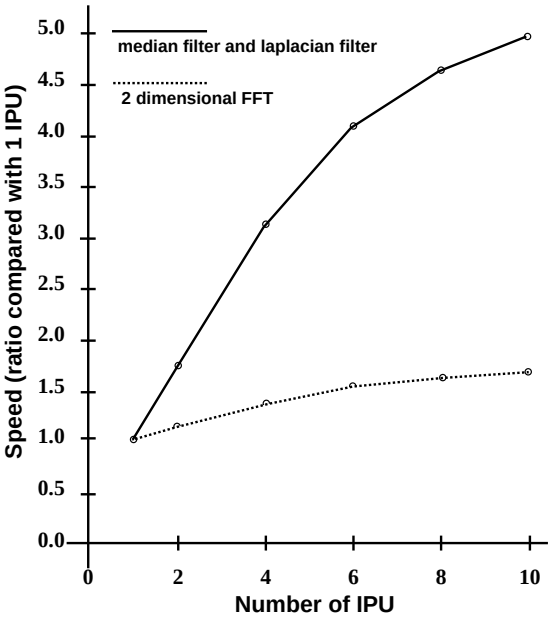

Fig.8: Processing speed versus the number of **IPU**

VII. Conclusion

Distributed image processing environment **VIOS Ⅱ** which uses many workstation connected by local area network is proposed. This system is composed of three kind of process **VPE**, **OM**, and **IPU**. This system has no special hardware, only uses workstations connected by network. It then becomes possible to use many workstation efficiently. From now on we have a plan to develop an efficient module distribution algorithm using this system.

References

[1] Proceedings of First International AVS User Group Conference (1992)

[2] Hiroshi Matsuo, Kinichi Wada, Akira Iwata, Nobuo Suzumura: " A Distributed Image Processing System VIOS",Trans., IEICE, vol. J75-D-II,8, pp.1328-1337(1992)

[3] W.Richard Stevens:"UNIX NETWORK PROGRAMING",PRENTICE HALL(1992)

[4] Sun Microsystems Inc.:"Programing Utilities & Libraries",Sun Microsystems Inc.(1990)