計算機アーキテクチャ 134－23
（1999. 8. 3）

Router Design and Kernel System of a Compact Parallel Graphics Accelerator

Tran Cong So and Katsuhiro Yamazaki
Department of Computer Science
Ritsumeikan University

Abstract: The Volume Visualization Accelerator (VoViAc) system is a compact parallel system with a 3D torus interconnection network designed for volume graphics visualization. With the development process, we have designed and implemented a Router and a parallel Kernel system for the VoViAc. This paper describes the Router and Kernel system in the aspects of designing and implementing them for a very compact parallel system. Primary results of the Router and Kernel system will be presented.

1 Introduction

Recently, performance of PCs and workstations is much more powered, but for volume graphics visualizations, especially in interactive real-time application, these powers are still not sufficient. Research on volume graphics acceleration still attracts many researchers to invest their work to find out other better ways for volume visualization, both in speed and cost-performance factor.

A number of previous special purpose hardware approaches can be found in the literature such as the CUBE-4 [1], VIRIM[2], and the ReVoler/C40 [3]. The key issue of CUBE-4 is the skewed memory organization for beam parallelism. The hardware of VIRIM consists of several modules, each composed of a geometric unit for volume rotation, resampling, and gradient computation and a ray-casting unit for the final image generation. The Revolver/C40 uses pipelining of stages in volume rendering. In each stage, an array of processors processes data in parallel.

The Volume Visualization Accelerator (VoViAc) system has been developing since 1995 for volume graphics speed-up purpose [4]. The VoViAc architecture bases on the 3D-torus topology that is scalable. The main feature of the VoViAc is to process light ray in pipelining fashion. Figure 1 shows a prototype of the VoViAc with 8 processing elements connecting with a Host computer and a 2D-memory array. The VoViAc receives volume data set from the Host; processes it with ray casting or 3DDDA-algorithm and outputs image to display's memory or sends it back to the Host for other uses.

Figure 2 illustrates a processing element (PE), which consists of a processor, local memory and a Router. PEs use DSP (digital signal processor) for high speed in mathematical (image) processing. PEs communicate with each other by the Router over an interconnection network. The Router is designed to

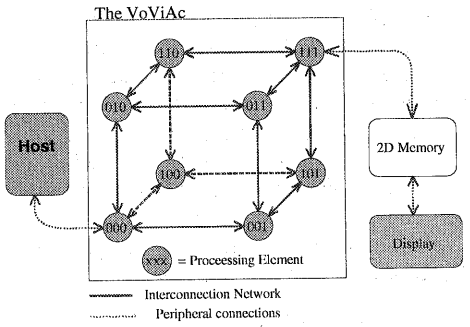

Figure 1 System architecture

implement on FPGA (Field Programmable Gate Array) to reduce the development time. The local memory has 512 Kbytes of SRAM. PE's boards have additional circuits supporting for FPGA programming, clocking, etc.

The development process of the VoViAc is divided into many parts [5]. At first, we designed the system architecture, built printed circuit boards and assembled components together. Then, we constructed the Router design, implemented Kernel system and support software. In the constrains of implementing a compact parallel system, the Router

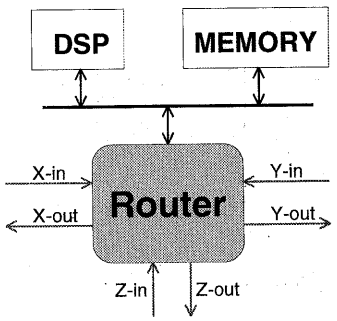

Figure 2 Processing element structure

and the Kernel system need to be compact to be fit into the system. So that, the Router and Kernel system must be implemented as small and as fast as possible and guarantee for reliable operations. Therefore, the Router was designed in traditional way with a schematic editor (Mentor Graphics EDA tools) to achieve hand-tuning optimized circuit. The Router's layout in FPGA was full floor-planned to carry out the best timing solution. The Kernel system was coded in an assembly language, it could be fitted in a small portion of system memory and acquired the fastest speed. In the following sections, we would like to describe the implementations and performance evaluations of the Router and the Kernel system.

2 The Router

The VoViAc is a distributed memory parallel system, hence, PEs use messages to communicate with the others in the system. To route messages over the interconnection network, we implement a device called Router in each PE. When a thread (process) sends a message, the Kernel packetizes it into packet(s) and then sends packet(s) to the Router. The Router then automatically routes packets to the destination, where received packets are reassembled back into original message.

2.1 Network Topology

The VoViAc's intercommunication network (IN) is a 3D torus as shown in Figure 3. PEs are connected to six neighbors, called north, south, east, west, up, and down. PEs on the edge of the mesh are connected to PEs on the opposite edge, creating the torus structure. Routers show up as valves at the junctions of connections to control the flow of packets in the interconnection network. In each dimension, the Router has an input port and an output port connecting to the previous and next nodes. A pair of injection and extraction channel conjoins the Router to local DSP.

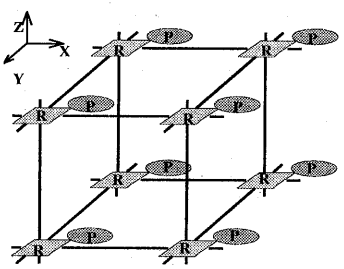

Figure 3 Network topology

2.2 Wormhole Routing

There are various types of switching technique used in interconnection networks such as circuit switching, packet switching, virtual Cut-through, wormhole, mad post-man switching [7]. We have chosen the wormhole routing to be implemented in the VoViAc, due to the wormhole technique does not require buffering complete packets in intermediate node(s), so that, allows us to construct smaller and faster Router. In wormhole routing, message packets are broken up into flits (a byte in the VoViAc) to pipeline throughout the network. Several first flits contain routing information and form packet header; the others contain packet data. When a packet reaches a Router, its header is examined and then the Router forwards the header to required output port as soon as the routing decision is made without waiting for the whole packet to be received into the Router. As a result, succeeding flits of the packet just simply follow preceding flits along the same channel. At the same time, a channel from the input port to the output port in the Router is occupied until the whole packet has passed over the Router.

2.3 Flow Control and Routing Algorithm

Flow control policy is how to control network traffic flow without causing congestion or deadlock. The wormhole routing uses blocking policy incases of packet collision. When the first packet occupied the requisite output channel, the second packet is being blocked from advancing. When the output channel is free from the first packet, the second packet is then locating the channel and advancing. The blocking flow control is easy to be implemented but this may cause deadlock if we do not have a suitable routing algorithm.

In parallel system, the message priority is important. When two packets require the same channel at the same time, a packet with higher priority will allocate to the channel. Thus, a message with higher priority will be routed faster incase of high traffic appears in the IN. This supports for multi-level priority task. For instance, kernel-to-kernel communication should have higher priority than application-to-application communication.

Routing algorithm deterministically or adaptively computes routing path to guide a packet to reach destination. In the VoViAc's Router, a deterministic modified E-Cube routing algorithm, which based on dimension-ordering routing, is used to archive the compactness and fastness of the Router. The modified E-Cube routing is applied for 3D-mesh, 3D-torus topologies. This algorithm is described as follows:

The next node address $a\{x_a, y_a, z_a\}$ of a 3D-torus is calculated from the present node address $p\{x_p, y_p, z_p\}$ and the destination node address $d\{x_d, y_d, z_d\}$ as follows:

$$a\{x_a, y_a, z_a\} = p\{x_p+i, y_p+j, z_p+k\}$$

With

i= 1 if $(x_p \oplus x_d \neq 0)$

j=1 if $(y_p \oplus y_d \neq 0$ and $j = 0)$

k= 1 if $(z_p \oplus z_d \neq 0$ and $j = 0$ and $k = 0)$

i, j, k = (0,1) and a, p, d are binary coded addresses

In this algorithm, a packet always routes in X dimension first, then Y dimension, and lastly Z dimension. When the packet reaches the same level of destination address in a dimension, it detours to the next dimension until it reaches the destination. This algorithm eliminates deadlock or live-lock situation.

However, this dimension ordering routing do not offer minimal routes for the torus, but somehow, this deadlock-free routing, which lets packets traverse through longer path, sometimes can reduce network traffic for other reasons.

2.4 Broadcasting and Multicasting

A broadcast pattern is the case of one-to-all communication while the multicast pattern corresponds to one-to-many communication. So, in some respect we can consider broadcast as a special case of unicast communication. The multiple destinations are encoded and appended to the packet header. Such multicast packet is called multi-destination packet. To perform better traffic and distance-cost of multicast, the tree-base multicast algorithm [7] is used. Packet routing still follows the routing algorithm described in the above section.

2.5 Message and Packet Format

Messages carry information between PEs. For processing and routing purpose, a message is created with the following structure (Figure 4):

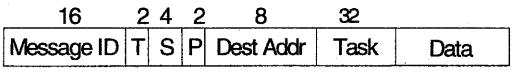

Figure 4 Message format

Message ID is used to identify the message. T field is the routing Type of the massage. S field is the binary address of the sender node. Destination address field contains 8 bits corresponding to the addresses of destination node(s). P field is the Priority of the message. Task field contains the task that destination node(s) must process. Data field is the data for the task. The message has minimum length at one packet.

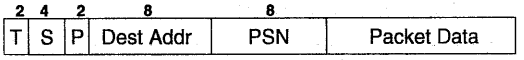

Figure 5 Packet format

Messages are packaged into packet(s) to transmit over the interconnection network (IN) through Routers. Messages are later reassembled from packets at the destination node. Packet format is shown in Figure 5. Packets have fixed length at 31 bytes. The 1st and 2nd bytes are header of a packet that contains T, S, P and destination address parameters. The header guides the packet to reach the destination. The 3rd byte is always Packet Serial Number (PSN). PSN is used to reassemble packets to form original messages in case the messages required more than one packet to form. The bytes from 4th to 31st are data of the packet.

2.6 Router Structure

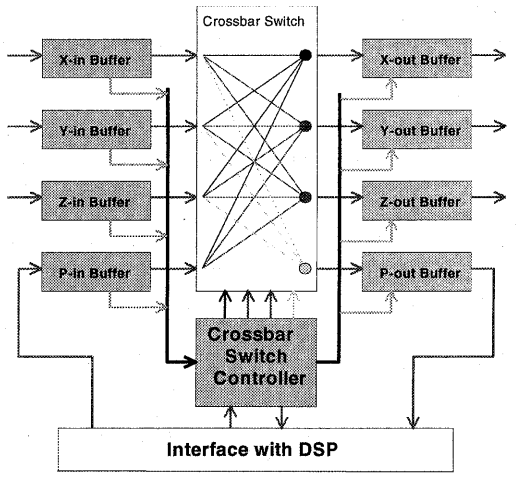

Figure 6 Router structure

Figure 6 illustrates the Router structure. The main part of this Router is a Crossbar switch, which connects inputs to outputs to perform packet routing. The Crossbar switch can tie up an input to one or many outputs at a time. The one-to-one connecting performs unicast communication while the one-to-many connecting does multicast communication. The Input buffer controls the link with previous Router and does the destination address decoding. The Crossbar switch controller governs crossbar switch states, does scheduling for channel request; and manipulates the output buffer to change packet's header incase of multicast and broadcast. The P-input and P-output buffers are the corre-

sponding injection and extraction channels for the local DSP. The Interface with DSP circuitry provides DSP control over the Router.

When a packet reaches an input buffer, its header is firstly latched. Routing information, such as destination, priority and routing type are then decoded to carry out a routing decision. After that, a crossbar channel request asks the Crossbar controller to provide a routing channel. The Crossbar controller then queues up the request in a FIFO buffer. When the request is in its order to be solved, Crossbar switch controller manipulates the Crossbar to connect the input to the corresponding output and establishes the requested channel. If the packet is a multi-destination packet, the header of the output packet will be changed appropriately with output dimension. When the whole packet has passed over, the input buffer de-asserts its channel request; therefore, the Crossbar switch controller orders the crossbar switch to disconnect the channel. The Router finishes a routing cycle.

The Router can do many routing decisions concurrently. For example, maximal four unicast communications (X -> Y, Z -> X, Y-> P, P -> Z) can occur at the same time.

2.7 Router Implementation

The Router was designed with Mentor Graphics tools. The design was based on schematic (Idea Station) to allow hand-tuning optimization of performance and size. After simulated in functionality wiht QuickSim, the design was re-targeted into Xilinx netlist and then done place and routing by Xilinx tools (Xmake). With the back-annotation delay information, we did timing simulation before debug in the real FPGA chip. The design has been entirely floor-planned to be able to fit into FPGA and also optimize timing specifications.

Table 1 Estimate of device utilization (4010DPQ160)

Function	Quantity	Utilization (%)
I/O pins	91	71
CLB FG function generators	555	69
CLB H function generators	93	23
CLB flip-flops	340	43
Bus resources	47	59

3 Kernel system

3.1 Kernel Structure

The VoViAc's Kernel is like a pseudo operating system for each node. The Kernel structure is shown in Figure 7. The Kernel must be enough functionality to allow the VoViAc to communicate with the host computer but be small enough to fit into tiny EEPROM. Also the Kernel has to supply most basic device drivers and interrupts for applications running on the Kernel. At start up, the Kernel is the software, which the VoViAc run to check itself and to communicate with the host computer. Therefore, a part of the Kernel must be contained in EEPROM. The other part of the Kernel will load from the host computer.

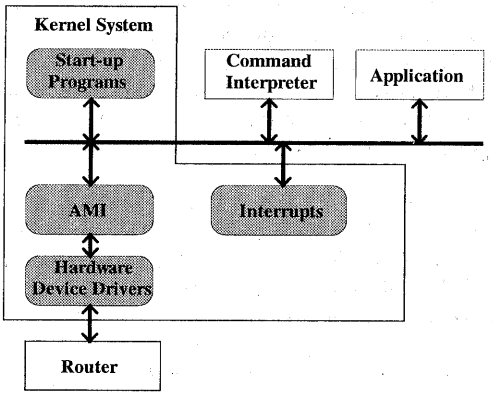

Figure 7 Kernel structure

The Kernel is system software, which supports for the following responsibilities:

- Setup the system hardware at power start-up.
- Communication program with the Host computer to load the other part of the Kernel.
- Communication program with other nodes to check the overall system or to complete the Kernel.
- Supply the most basic functions, interrupts and hardware device drivers of the system for application that isolate the applications from system hardware and from system software.

3.2 Setup Procedures

At power-on, the start-up procedures are executed to setup the VoViAc. The start-up process includes several specific procedures:

- Node setup procedure configures the hardware of each node, checks hardware functionality.
- System setup procedure checks the IN statuses in all nodes to create overall status variables.
- Kernel loading procedure loads the other part of Kernel from the Host. It also completes environment variables such as interrupt table and software environment statuses. At last, this procedure loads the command interpreter to the VoViAc.

3.3 Hardware device drivers

Hardware device drivers are used to control the hardware. The following device drivers are provided:

- ***Router device driver (RDD)*:** This driver exists in all nodes and supports for interfacing with the Router. An application interacts with Router device driver though the AMI to send and receive messages over the IN and moreover do synchronization between processes.

In sending, an application sends a message request to the RDD. The RDD creates the message structure and sends the message to the bottom of outgoing queue. Meanwhile, the RDD packages the top message into packet(s) and orderly sends packets to the Router. The Router then sends packet(s) over the IN to the destination. In receiving, the Router receives packets and sends them to the RDD. The RDD reassembles the original message from packets with information in the header and the PSN and then puts it to the bottom of incoming queue. Meanwhile, the RDD processes the top-level message and creates an event to the application.

The RDD supports for blocking (synchronous) and non-blocking (asynchronous) message send and receive. In blocking mode, the RDD waits for the completion of sending or receiving then returns control to the calling process. On the contrary, in non-blocking mode, the RDD returns control to the calling process before the actual sending or/and receiving has been completed.

The RDD supports for point-to-point and collective communication between nodes. Point-to-point communication does communicate between a pair of nodes while collective communication takes place among all nodes. Also, the RDD has ability to do barrier synchronization among processes.

- ***Host communication device driver*:** Only the root node has to use this driver to communicate with the host computer. This driver supports the command interpreter program in VoViAc to communicate with the Monitor program in the Host to provide application loading, debugging, testing and benchmarking. Also, this driver supplies a simple file system for applications while co-operate with the Monitor.

3.4 Application Message Interface

The principal role of the Kernel is to provide message-passing interface for applications that run on the VoViAc. Application Message Interface (AMI) supplies function calls for application to send, receive, inquire messages, and to do barrier synchronization among nodes. At this time, we have implemented some of the most basic functions for the AMI. The functions are called through a software interrupt. The following is the brief description of the completed functions.

Blocking Send. A blocking send requires the Kernel to complete the transfer before returning to application. Input parameters include address of data location to be sent, length of data, destination address, ID of message to be sent and routing type.

Unblocking Send. An unblocking send does not require the Kernel to confirm that the message has been sent. So that, after the function is called, we can not assure that the data that was sent is available to be changed or not. To confirm that, we uses the un-blocking send test function.

Blocking receive. This function scans the input message buffer repeatedly to find out a required message. Inputs are physical address of the receive buffer, data length, ID of the message to be received and source node address of the message.

Unblocking receive. The function scans input message buffer for only once. It returns the data to the application if message is found.

Barrier synchronization. This function provides synchronization for processes of application that are running on PEs. When called, this function sends a special message to all nodes and then waits for reply. Once the replies from all nodes have been received, the function returns to the application.

Unblocking send test. After sending a message by an un-blocking send function, we do not know whether the data was already sent or not. Hence, this function provides a method to check it out. The function checks the message ID in the output message buffer. If the message has not been sent, then the message ID should appear in the buffer. The function returns code to indicate the status of the sending.

In the AMI, user decides whether a unicast or a multicast is done when programming application by putting the destinations address in destination field before calling send function.

3.5 The Monitor and Command Interpreter

The Monitor's programs running on the Host computer provide manipulation of VoViAc's operations. At startup, Monitor's loading program loads Command Interpreter to the VoViAc. And while co-operating with the Command Interpreter, the Monitor loads, executes and debugs parallel programs on the VoViAc. Thus, the Monitor and the Command Interpreter must be coupled tightly in operations.

The Monitor supports mainly for debugging application. The Monitor allows users to view and change contents of registers, memory locations in

all PEs. Users also can run the application in step-by-step or with breakpoint fashion.

4 Results

The VoViAc system is now executable. At the very first step, a very simple program that calculates the summation of a data array in parallel running on 8 nodes has returned a correct result. In this sample program, the Host sends the same program to all nodes and each node calculates its own summation. Then, the node 0 (root node) collects temporary data from other nodes; computes the final result and finally send it to the Host.

For the Router and Kernel system evaluation, we have measured traverse time of message passing in the VoViAc. In the first experiment, we measure the ping response time (send a request message and wait for a response message) from node 0 to other nodes. The result shows that the ping response times almost do not change with network distances (from 1 to 3) between node 0 and other nodes with range of traverse times varies from 9765 to 9788 clock cycles. So, we can ensure that the overhead cost is mostly caused by message processing software (Kernel) for various data copying, packetizing, packet reassembling, buffers management, etc. This affirms that the latency of wormhole routing is not much changed by the network distance.

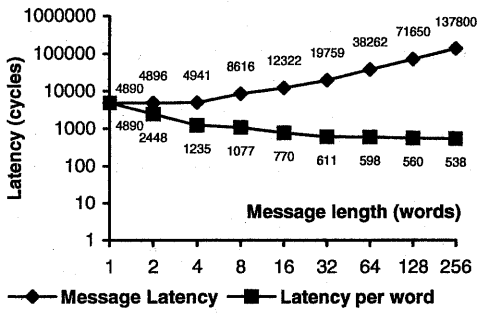

Figure 8 Message length and average latency

In Figure 8, the relation between message length and average latency of unicasting is shown. The message latency is increased approximately with message's length, while latency cost per word is reduced. This can be explained by the cost of message creation and packetization. With large message, the latency cost is still high with the cost of data transfer, buffers management, and packetization and wait-states in the interaction between DSP and the Router.

5 Conclusions

This paper has described the implementation and evaluation of the Router and Kernel system for the VoViAc mainly in message passing mechanism, the Router design, and Kernel software supporting for message handle and system start-up. Some simple test procedures and measures have been done to show that the VoViAc system is now executable and ready for further development.

The VoViAc' Router and Kernel needs to be improved with more functions to be completed. In near future, multi-cast and broadcast routing will be examined and Kernel will be optimized to limit the high-cost of message routing. The Monitor's debug functions need to be developed as well. Some other test programs should be implemented on the VoViAc to evaluate the VoViAc in more details such as sorting, encoding, mandelbrot, etc., before heading forward for volume graphics visualization.

Acknowledgement

We would like to thank Mentor Graphics Japan Co., Ltd. for providing us CAD tools in the higher education program.

References:

[1] A.Kaufman and R.Bakalash, "Memory and Processing Architecture for 3D Voxel-base Imagery," IEEE Computer Graphics and Applications, Vol.8, No.6, pp.10-23, 1988.

[2] T.Gunther, at al, "VIRIM: A Massive Parallel Processor for Realtime Volume Visualization in Medicine," Proceeding 9th EuroGraphics Workshop on Graphics Hardware, Volume EG94HW, EuroGraphics Technical Report Series, pp. 103-108, 1994.

[3] 對馬, 他, "ボリュームレンダリング専用並列計算機－ReVolver/C40, " 並列処理シンポジウムJSPP'95論文集, pp.11 - 18, 1995.

[4] 山崎, 山崎, "ボリューム可視化用アクセラレータの検討, " 情報処理学会第51回全大, 4S-11, Vol.2, pp. 337-338, 1995.

[5] T.C.So, K.Nakajima, and K.Yamazaki, "The message routing mechanism and kernel system of a parallel graphics accelerator," Proc. 56th IPSJ'98 Record, 1N-05, pp. 105-106, 1998.

[6] Kai Hwang, "Advanced Computer Architecture: Parallelism, Scalability, Programmability," McGraw-Hill, 1993.

[7] J.Duato, S.Yalamanchili, and L.Ni, "Interconnection networks: An engineering approach," IEEE Computer Society Press, 1997.