Co-processor System Design for Fine-Grain Message Handling in KUMP/D

Hiroshi Tomiyasu[1], Shigeru Kusakabe[1],
Tetsuo Kawano[1]*, and Makoto Amamiya[1]

Department of Intelligent Systems Graduate School of Information Science and Electrical Engineering Kyushu University Kasuga, Fukuoka 816 Japan

Abstract. In parallel processing, fine-grain parallel processing is quite effective solution for latency problem caused by remote memory accesses and remote procedure calls. We have proposed a processor architecture, called Datarol-II, that promotes efficient fine-grain multi-thread execution by performing fast context switching among fine-grain concurrent processes. We are now building a prototype multi-media machine KUMP/D (Kyushu University Multi-media Processor on Datarol-II) on the basis of the fine-grain multi-threading architecture. In the design of the KUMP/D, we used the commercial microprocessor for its processing element, and designed a co-processor, called FMP(Fine-grain Message Processor), for fine-grain message handling and communication control. In this paper, we show the KUMP/D processor design and its performance evaluation.

1 Introduction

In parallel processing, one of the most critical issues is the latency problem caused by remote memory accesses and remote procedure calls. To solve this problem, fine-grain parallel processing is quite effective. We have proposed a processor architecture, called Datarol-II[7][2]. The Datarol-II architecture is an evolution of the Datarol architecture[1] which surmounts drawbacks of dataflow computation scheme, such as unnecessary data copy caused by-value data access mechanism, increasing memory access by operand matching. The main idea of the Datarol architecture is to extract a multi-thread control program, called Datarol, from a dataflow graph, to eliminate redundant flow controls in the graph by introducing a by-reference data access concept.

The Datarol-II processor architecture is an optimized version of original Datarol architecture. The Datarol-II executes fine-grain threads by means of a program-counter-based execution pipeline with high-speed registers.

We are now building a prototype of multi-thread machine KUMP/D (Kyushu University Multi-media Processor on Datarol-II)[2]. In the design of the KUMP/D, we used a commercial high speed microprocessor Pentium rather than a completely new designed processor for its processing element.

By using a commercial high-end microprocessor, we can build a new machine in a short design cycle that follows the mainstream of processor design technology. Current commercially available microprocessors are designed as a single processor or, at most, low-scale parallel processor, and they are tuned to offer very

* Currently, he is at NTT Software Laboratories.

high performance in internal sequential execution. However, their functions lack in the high speed communication for supporting the higher-scale multi-thread parallel processing. The key issue in designing a cost effective multi-threading parallel machine is to develop a very simple high speed message handling and communication control hardware device, or co-processor, which complements the high speed internal execution by commercial microprocessor.

We design a new simple mechanism, FMD (Fine-grain Message Driven mechanism). A co-processor, FMP (Fine-grain Message Processor), is an external hardware to realize efficient fine-grain message handling based on the FMD. The FMP incorporates fine-grain data and synchronization messages into the main computation minimally interrupting the core processor.

In the paper, we introduce the KUMP/D machine and its processing element construction with Pentium and FMP, and discuss the feasibility of the design from the viewpoint of cost/performance.

2 Overview of the KUMP/D

Fig. 1 shows an overview of the KUMP/D. Each processor element(PE) is based on the Datarol-II architecture [7], which is especially designed for fine-grain parallel processing. The PE construction is described in subsequent sections.

For the scalability, as the inter-PE network of the KUMP/D, we have chosen a 2-D torus network[3], which is also suitable for mapping images on PEs. This interconnection network is specialized for fine-grain message communication, which includes parameter passing and result value returning. This network has a deadlock-free structure based on its hierarchical packet buffers[4][12].

Since the interconnection network is not suitable for long I/O packet transmissions, KUMP/D has a specialized I/O network, which consists of serial links connecting the PEs in a ring structure, for communication of long packets such as video I/O packets and disk I/O packets[8]. One important feature of the I/O network is that it supports a mechanism of the synchronization of each video frame and process execution[12].

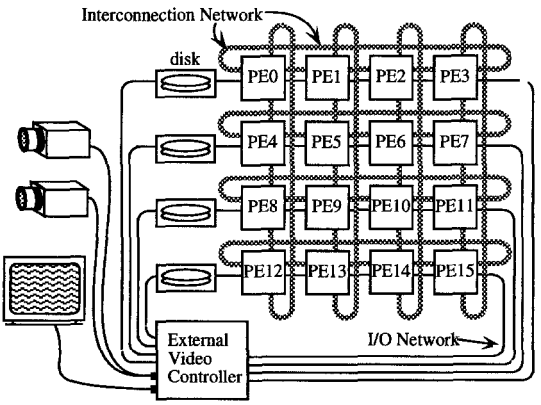

Fig. 1. Overview of the KUMP/D

3 FMD(Fine-grain Message Driven mechanism)

The FMD (Fine-grain Message Driven mechanism) is a revised version of the Datarol-II execution mechanism. The FMD is a message driven execution mechanism, and an FMD message is simpler and more general than that of the Datarol-II. Basic FMD message is an explicitly addressed remote memory write message which also contains a continuation thread after its write operation. The FMD uses a remote memory write message and a remote thread activation message.

Basic runtime model of the FMD is similar to that of the Datarol. A number of function instances are created during program executions. A function instance has a shared program code and private execution environment, which is called an instance frame. A program code is split into threads. A thread is a code block that is executed without any interrupts until its termination point. A context switch may occur at the end of a thread. A context is realized as an instance frame on memory.

By assuming the order of message sent to the same destination instance from an instance is preserved, the FMD simplifies link-receive parameter passing of the Datarol-II mechanism.

In the FMD, a caller instance sends messages which write argument data into the callee instance frame, and the caller also sends a message which activates the destination thread[2] as shown in Fig. 2.

In a function application, at first, a caller instance issues a message to get a new instance frame. Second, the caller sends a message to the callee that has newly obtained instance to activate initial thread. Then, the caller sends a set of parameter data by link message to the callee. After the caller sends all the data needed to an entry thread, the caller activate the entry thread by start instruction.

Synchronizations of threads in an instance are realized by means of messages. A thread that needs multiple inputs is activated after all the data arrived. In the FMD, threads are repeatedly activated by messages along with the dependency, and PE repeatedly executes activated thread during a program execution.

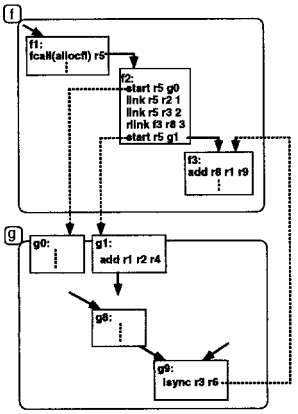

Fig. 2. Function invocation in FMD

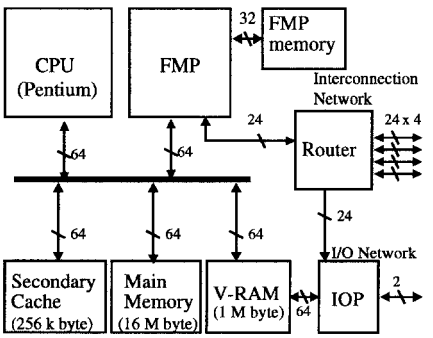

Fig. 3. Processor Element(PE)

[2] We call such threads that are formally activated by receive messages in Datarol-II "entry threads" in the FMD

4 FMP(Fine-grain Message Processor)

The FMP is an implementation of the FMD mechanism, which assists fine-grain message passing, thread synchronization, remote memory access, and instance frame management.

The PE construction is shown in Fig. 3. When issuing a message, the CPU writes a set of data to the FMP. Messages from the FMP to the CPU is written to secondary cache, and the FMP starts a thread execution. In this thread, the CPU gets these messages by explicitly inserted load instructions. To reduce bass traffic, FMP has its own high speed memory called "FMP memory." In this section, we describe main functions of the FMP.

4.1 Thread Synchronization Mechanism

In KUMP/D, we use a structure data called "SyncCell" for thread synchronization. Each SyncCell has an instance frame address (frame), an instruction pointer to starting thread address (IP), and a synchronization counter (count). In function application, an initial thread gets SyncCells and initializes these values.

Message passing among threads and thread synchronizations are operated following way:

- A CPU sends arguments for the destination thread, and issues a SYNC FMP instruction. The SYNC FMP instruction specifies the SyncCell number and gives the FMP a synchronization point of a thread.
- When an FMP receives the SYNC message, the FMP decreases and checks synchronization counter. If a synchronization has proved successful, then the thread is activated.
- The FMP releases the SyncCell, and puts the instance frame address and the starting thread address of active thread into the queue called "Thread Queue."
- At the end point of each thread, a CPU gets a new active thread from Thread Queue, and executes the thread.

4.2 Instance Frame Management

To reduce overhead for memory management, the KUMP/D uses fixed size instance frames for small instances. The FMP has several frame stacks[3] These frame stacks consist of pointers which point to the start addresses of free instance frames. Instance frame allocation and release are implemented by these stack operation.

When an application program requires a larger size instance frame, an interruption is occurred, and a memory management routine supplies a required frame.

[3] Currently, the prototype of the KUMP/D supports four frame stacks.

5 FMP instructions

FMP instructions to issue a message are implemented as several kinds of memory access instructions in CPU. In this section, we explain the FMP instructions and FMP registers accessible from user programs, where FMP registers are internal registers in FMP and are provided as an interface to CPU. (For more details, see [2])

Table 1 shows a list of FMP instructions. In this table, Rs indicates a register name of the CPU, FRa and FRs are register names of the FMP. When a CPU issues an FMP instruction, the CPU sets a CPU register "Rs" and FMP registers beforehand. Registers in square brackets mean value of the register.

Instruction	Operand	Function and Message Format <OP,ADDR,DATA>	Notes
		FMP Register Operation	
SETFRA	*Rs*	Set an FRa register	
READFRA	*Rs*	Read an FRa register	
SETFRS	*Rs*	Set an FRs register	
READFRS	*Rs*	Read an Frs register	
		SyncCell Operation	
ALLOCSCELL	*Rs, count*	Get a SyncCell	[*Rs*]=Frame
SETSCELLIP	*Rs*	Set an Instruction Pointer(IP) into the SyncCell	[*Rs*]=IP
		Message Issue	
LINK	*Rs*, $offset$	<SET,[FRa]+$offset$,[*Rs*]>	[*Rs*]=data [FRa]=Instance Frame
RLINK	*Rs*, $offset$	<SET,[FRa]+$offset$,([FRs],[*Rs*])>	[*Rs*]=$offset$ [FRa]=Instance Frame [FRs]=SyncCell
START	*Rs*	<START,[FRa],[*Rs*]>	[*Rs*]=IP [FRa]=Instance Frame
SYNC	*Rs*	<SYNC,[*Rs*],->	[*Rs*]=SyncCell
LSYNC	*Rs*	<LSYNC,[FRa],[*Rs*]>	[*Rs*]=data [FRa]=SyncCell
		Othres	
ALLOCFL	*Rs*	Get an Instance Frame	[*Rs*]=size
FREEFL	*Rs*	Release an Instance Frame	[*Rs*]=size

Table 1. FMP instructions (for user program)

6 Performance Analysis

In this section, we evaluate the performance of the KUMP/D. To put it concretely, we estimate the performance in sending/receiving messages for both the KUMP/D and the Datarol-II. Because the message passing is the operations very freaquently executed in fine-grain multi processing, evaluation of message handling performance is very important. The Datarol-II processor is suitable for performance comparison, since it has high speed hardware mechanism for message handling[7].

6.1 Preparatory Analysis

In a PE of the KUMP/D, since the CPU bus (memory bus) is used by FMP instructions to issue a message as well as usual memory accesses, the memory bus will easily be a bottleneck. In a PE of the Datarol-II, the potential bottleneck is also access to Register Buffer(RB), which is used as cache memory in Datarol-II[7]. Accordingly, we examine the capacity in handling messages (packets) for three typical operations by focusing on the frequency of memory bus occupations.

(1) **Function invocation (frame allocation and activation of initial thread):** The call instruction of the Datarol-II PE corresponds to the following five KUMP/D instructions which perform a frame allocation and an activation of initial thread.

- allocscell, setscellip } set SyncCell
- allocfl } frame allocation
- setfra, start } initial thread activation

(2) **Parameter passing:** In KUMP/D, a parameter passing uses following three instructions, while two instructions (link and receive) are used in Datarol-II.

- setfra, link } message for argument data
- start } entry thread activation

(3) **Receiving result value:** In KUMP/D, receiving a return value uses six instructions, while 3 instructions (rlink, return and receive) are used in Datarol-II.

- allocscell, setscellip } set SyncCell
- setfra, rlink } rlink message
- setfra, lsync } return message

Table 2 shows the number of memory bus cycles in the Datarol-II PE. In the Datarol-II PE, a memory write (W_D) occurs in FU to set continuation data. When receiving a packet in CU, a memory read (R_D) for continuation data and a write for packet data (W_D) occur. Table 3 shows the number of memory bus cycles in the KUMP/D PE. In KUMP/D PE, CPU uses memory bus to issue an FMP instruction (I_K). FMP writes the contents of message to the memory when received a message(W_K).

	FU	CU	total
(1)	$1W_D$	$1R_D$+$1W_D$	$1R_D$+$2W_D$
(2)	$1W_D$	$1R_D$+$1W_D$	$1R_D$+$2W_D$
(3)	$2W_D$	$2R_D$+$2W_D$	$2R_D$+$4W_D$

Table 2. number of bus cycles (Datarol-II)

	FU	CU	total
(1)	$5I_K$	$1W_K$	$5I_K$+$1W_K$
(2)	$3I_K$	$1W_K$	$3I_K$+$1W_K$
(3)	$6I_K$	$2W_K$	$6I_K$+$2W_K$

Table 3. number of bus cycles (KUMP/D)

Where, R_D is the number of cycles used in read access from RB, and W_D in write access into RB. I_K is the number of memory bus cycles used when CPU issues an FMP instruction, and W_K in write access into memory.

To compare KUMP/D with Datarol-II, we estimate the memory bus occupation time by using a clock rate of a KUMP/D PE (66 M Hz)[4] as a standard. A PE of the KUMP/D executes with this shortest clock cycle, since FMP interface to receive instructions from CPU can be built with simple logic.

We assume the external cache is an S-RAM with $15ns$ access time, and memory access time of the KUMP/D (R_K and W_K) is 2-clock. On the other hand, we assume the clock speed of the Datarol-II will be an half of Pentium, since Datarol-II PE needs custom design. In Datarol-II PE, since one RB access will be finished within a basic cycle, we assume $R_D = W_D = 2$ clocks.

Table 4 lists the number of clocks for three operations discussed above by using this assumption. From this table, we can calculate that Datarol-II needs 32 clocks and the KUMP/D 46 clocks for a 2-arity function call with one result. The KUMP/D PE uses the memory bus about 1.4 times more than Datarol-II PE. If the access time of the memory bus dominates the execution time in PE, the capacity for message handling of the KUMP/D is about 70% compared with Datarol-II. Moreover, the number of instructions in KUMP/D is 17, while 8 in Datarol-II.

	Datarol-II	KUMP/D
(1)	6	12
(2)	6	8
(3)	12	16

Table 4. Total memory bus cycles. (All accesses are assumed as two clocks.)

As discussed above, regarding message handling, the performance of the KUMP/D PE is assumed to be about 70% compared to Datarol-II processor which runs at about half clock rate of the KUMP/D.

Since the increase of the clock speed-up of recent microprocessors pis so rapid, the gap of clock rate between such microprocessors and custom-processors will extend [5]. Judging from this expectation, we can conclude that message handling performance of the KUMP/D is almost the same as Datarol-II from a practical point of view. In addition to that, KUMP/D PE has much higher peak performance with fast clock rate and superscaler facility than that of the Datarol-II.

6.2 Performance of Message Handling

We use the behavior level simulations of the Datarol-II and KUMP/D for some sample programs (see Fig. 4 and Fig. 5) to estimate the performance of PE. In

[4] Current high-end microprocessors work at higher clock rate than 200 M Hz. However, these processors do not achieve drastic speed up at bus cycles, therefore we can apply these analysis of bus cycles for current microprocessors. If a new circuit technology produces higher bus cycle, we will be able to achieve better performance for message handling by designing the FMP for higher clock rate.

[5] Even though we estimate that Datarol-II PE will work at half of the KUMP/D PE (33MHz), it is hard to accomplish such a clock rate with custom design.

this simulation, we use hand compiled code. For **Fibonacci(n)** and **Queen(n)**, which are very fine-grain programs, elapsed time in Datarol-II is dominated by the frequency of memory bus usage. When executing such programs on KUMP/D, the memory bus will also be a bottleneck, since the memory bus usage in KUMP/D is 1.4 times more than Datarol-II. These results agree well with above estimates.

In larger problem size, i.e. n, on **Fibonacci(n)**, the reverse of performance is caused by the difference of cache line filling mechanism. In order to reduce the latency by cache miss hit, the Datarol-II processor has implicit loading mechanism[7], which predicts next instance frame from the Thread Queue and loads all cells of the frame. However, since the size of variables in **Fibonacci(n)** is too small, this mechanism has overhead.

In **Queen(n)**, which has a little longer threads and more variables, the difference of performances is smaller, there is no reverse of performance.

Since the frequency of message issues is smaller in practically used applications than these programs, the difference of performances will be smaller. Especially, in compute intensive applications, this approach of the KUMP/D will be a reasonable solution.

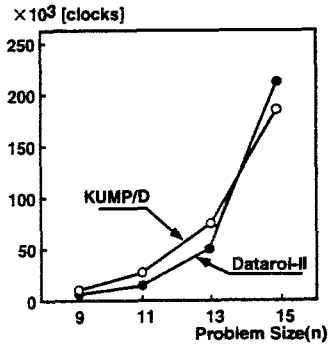

Fig. 4. Execution Time of Fibonacci(n)

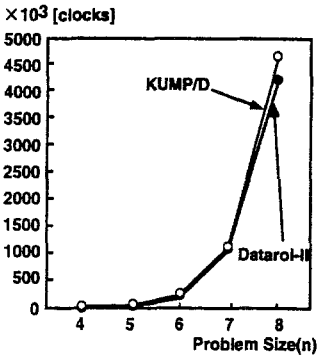

Fig. 5. Execution Time of Queen(n)

6.3 Performance of the Prototype

In the prototype of the KUMP/D, we use a Pentium(66 M Hz) processor, and the number of processor is 16.

Since the speed of an I/O serial link is about 33M bytes/sec, and there are four serial lines, occupation rate by video data will be around 0.41. p

In this condition, Table 5 shows our performance predictions of the KUMP/D for image processing by using hand compiled codes.

A PE of the KUMP/D will have enough performance for real time image processing, except for extremely heavy application such as Ray-tracing.

In addition, to evaluate the balance of processor performance and inter connection network, we predict a FFT program on the 16 processor system. Since image mapping to each PE is completely free by using External Video Controller, we can perform FFT by nearest neighbor communications. In this configuration, the prototype of the KUMP/D will perform 512 × 512 point 2D FFT in 5 video frames(NTSC).

Even in the prototype of the KUMP/D, this system will have enough performance to perform real time image processing in NTSC video image. We will achieve higher performance easily by increasing the number of PEs and performance improvement of core processor.

Problem	Performance
Ray-tracing	1 M [polygon/sec] 10 to 20 [objects/frame]
Polygon rendering	60 to 80 k [polygon/sec]
Canny filter	3.7 k [pixel/frame]
Region Segmentation	21.6 k [pixel/frame]

Table 5. Performance of PE

7 Related Work

In the design of an architecture, cost and performance trade-off should always be taken into account considering the current and future trends of the VLSI and microprocessor technology. One direction of multi-threading architecture research is to find a way to implement a cost effective machine still preserving the dataflow concept. Our research is on this direction, and several machine architectures were designed[7][2].

However, these architectures required special hardware designs, and could not get the benefit of the commercially available microprocessor technology.

Other projects like at MIT[10], ETL[11], and others[5] took the similar approach, and they shared the same problem. Our FMD approach is an attempt to use a off-the-shelf microprocessor for developing the fine-grain multi-threading architecture. In addition to using the off-the-shelf microprocessor, we considered two important points for the design of a massively parallel machine. One point is that the architecture should be tuned to efficient data/message communication, since communication is dominant in massively parallel computations. Another point is that the dataflow concept, whether it is in fine level or in coarse level, is quite natural as a parallel processing model.

Current commercially available microprocessors are designed as a single processor or, at most, low-scale parallel processor, and they are tuned to offer very high performance in internal sequential execution. However, their functions lack in the high speed communication for supporting the higher-scale multi-thread parallel processing. The key issue in desgining a cost effective multi-threading parallel machine is to develop a very simple high speed message handling and communication control hardware device, or co-processor, which complements the high speed internal execution done by commercial microprocessor.

*T[9] at MIT and EARTH[6] is on the similar approach. Different point of *T and EARTH from our approach is that these architectures use commercial microprocessors for both the internal execution and the communication and synchronization control. Since these architectures use the general purpose microprocessor for communication and synchronization control, message handling and synchronization functions have to be implemented in software. This causes an imbalance between the internal execution and the external communication control, and declines the performance in fine-grain multi-threading.

8 Conclusion

In this paper, we gave a co-processor design of the KUMP/D , and showed performance evaluation of the KUMP/D PE. We designed this architecture using a commercial high-end microprocessor and a external co-processor FMP which is a very simple high speed message handling and communication control hardware device. The KUMP/D PE has the same performance of fine-grain parallel processing as a custom designed Datarol-II PE, which works at half the rate of the KUMP/D clock speed. Since KUMP/D CPU itself has quadruple performance compared with Datarol-II PE, KUMP/D PE will achieve better performance than Datarol-II PE in practically used applications.

We can build a high-performance fine-grain multi-thread machine in a short term that follows the mainstream of processor design technology, because this approach solves a problem in special hardware design such as expensive development cost.

The KUMP/D efficiently handles asynchronous fine-grain parallel processes and has high throughput and high flexibility necessary for the next generation multi-media applications, which require complex computer vision and computer graphics algorithms.

References

1. M. Amamiya, and R. Taniguchi, "Datarol: A Massively Parallel Architecture for Functional Language," Proc. SPDP, pp. 726-735 (1990).
2. M. Amamiya, T. Kawano, H. Tomiyasu and S. Kusakabe,"A Practical Processor Design For Multithreading," Proc. of the Sixth Symposium on Frontiers of Massively Parallel Computing, pp.23-32, 1996.
3. W. J. Dally "Performance Analysis of k-ary n-cube Interconnection Networks," IEEE Transactions on Computer, Vol. 39, No. 6, pp. 775-785 (1990).
4. I. S. Gopal, "Prevention of Store-and-Forward Deadlock in Computer Networks, "IEEE Transactions on Communications, Vol. COM-33, No. 12, pp. 1258-1264 (1985).
5. V. G. Grafe and J. E. Hoch, "The Epsilon-2 Multiprocessor System," *Journal of Parallel and Distributed Computing*, Vol.10, No.4, pp.309-318, 1990.
6. H. H. J. Hum, G. R. Gao, et.al., "A Design Study of EARTH Multiprocessors," *Proc. 8th IEEE International Conference on Parallel Architecture and Compilation Techniques (PACT'95)*, pp.59-68, 1995.
7. T. Kawano, S. Kusakabe, R. Taniguchi, and M. Amamiya, "Fine-grain Multithread Processor Architecture for Massively Parallel Processin," Proc. of HPCA'95(First IEEE Symposium on High-Performance Computer Architecture), pp.308-317, 1995.
8. A. L. Narasimha, Reddy and James C. Wyllie, "I/O Issues in a Multimedia System, " IEEE COMPUTER, pp. 69-74 (1994).
9. R. S. Nikhil, G. M. Papadopoulos and Arvind, "*T: A Multithread Massively Parallel Architecture," Proc. 19th ISCA, pp.156-167, 1992.
10. G. M. Papadopoulos and D. E. Culler, "Monsoon: an Explicit Token-Store Architecture," Proc. 17th , pp.82-91, 1990.
11. S. Sakai, Y. Yamaguchi, K. Hiraki, and T.Yuba, "An Architecture of a Dataflow Single Chip Processor," Proc. 16th ISCA, pp.46-53, 1989.
12. H. Tomiyasu, T. Kawano, R. Taniguchi and M. Amamiya, "KUMP/D: the Kyushu University Multi-medea Processor," Proc. Computer Architectures for Machine Perception '95, pp.367-374, (1995).