RICA: Reduced Interprocessor–Communication Architecture
– Concept and Mechanisms –

Shuichi SAKAI† Hiroshi MATSUOKA† Yuetsu KODAMA† Mitsuhisa SATO†
Andrew SHAW* Hideo HIRONO† Kazuaki OKAMOTO† Takashi YOKOTA†

† Real World Computing Partnership
Tsukuba, Ibaraki 305, JAPAN
Email: {sakai, matsuoka, hirono, okamoto, yokota}@trc.rwcp.or.jp
† Electrotechnical Laboratory
Tsukuba, Ibaraki 305, JAPAN
Email: {kodama, msato}@etl.go.jp
* Massachusetts Institute of Technology
Cambridge, MA 0239, USA
Email: shaw@lcs.mit.edu

Abstract

One of the most significant issues in building general purpose massively parallel computers is the integration of computation and communication in an efficient and cost-effective manner.

This paper presents a way of integrating computation and communication from the viewpoint of the processor architecture. It firstly states the concept of RICA, Reduced Interprocessor–Communication Architecture, which means the simplified and fused structure of communication and computation. Hardwired simple direct invocation of threads, and fusion of execution pipelines and message handling pipelines are the mechanisms for RICA.

1 Introduction

Building a general purpose massively parallel computer is generally regarded as the most challenging theme in computer architecture. How to realize interprocessor communication is one of the central issues in such a system.

This paper presents a way of the integrating computation and communication in each processing element (PE). It firstly proposes the concept of RICA, Reduced Interprocessor–Communication Architecture, and then it focuses on the mechanisms of RICA.

Based on RICA, the authors have been developing the EM-series parallel computers[2][6][14]. Among them, EM-4[2][4] has been fully operational since April 1990.

2 RICA: Reduced Interprocessor–Communication Architecture

There are two issues in realizing a fast communication structure in the PE architecture. One is providing a fast message handling (or remote data handling) mechanism for receiving and sending data. The other is overlapping communication with computation.

2.1 Fast and Simple Message Handling

For fast message handling, it should not require more than a few clocks to execute the first instruction after message input. The first instruction here means the one that is purely an essential part of the new thread invoked by the message, i.e. it is not the message-handling instruction nor is it data store instruction for saving the environment of the previous thread. Ideally, message handling and context switching should not require executing processor instructions, but be directly supported in hardware. Software overhead should be eliminated for communication and context switching.

The hardware message handling and thread invocation mechanisms should straightforward pass instruction address and data to instruction execution hardware. Active message [9] is the idea to support this by software, but we insist that hardware support and reorganization of PE clearly produce much better performance.
2.2 Fusing Communication and Computation

For overlapping communication with computation, we may use an attached commercial processor to handle the network interface. Commercial machines such as Paragon [7] adopt this scheme. Advantages of this approach are: (1) it is not necessary to develop new hardware for communication, and (2) software development is easy since there is a cross compiler for such a processor. Disadvantages are: (1) hardware amount increases since the attached processor is usually over-designed for communication, and (2) time for communication gets longer, since there is software overhead for handling messages.

Dataflow architectures [1], in contrast, provide an excellent network interface. Typically, dataflow architectures have minimal overhead for handling messages since the interface is integrated into the main processor pipeline. In addition, a multiple-context pipeline has no context-switching overhead as far as performing packet executions. One of the drawbacks of dataflow computers is that they cannot provide local execution mechanisms with side-effects. Current multi-threaded architectures, which are the successors of dataflow computers [2] [5] [10], solve this problem by combining register architecture with message handling architecture. What this paper presents can be regarded as an advanced concept and mechanisms based on the multi-threaded architectures.

It is cost-effective to make a pipeline containing message handling stages and instruction execution stages which can be overlapped. For message handling, we can adopt pipelines independent of execution pipelines. This can be regarded as an extension of a cyclic pipeline seen in dataflow architectures. We can also regard it as an extension of a superscalar toward computation-communication integration.

The instructions are continuously executed in a sequential thread for exploiting locality, as normal RISC pipelines are.

Note that connection between the message handling pipelines and the instruction execution pipeline should be as smooth as possible. Otherwise, this interface will be the bottleneck.

The organization of pipelines are called is called pipeline fusion [6].

2.3 Abstract Architecture

The previous subsections lead us to the fused architecture where message reception is performed in a simplified hardware mechanism and a set of asynchronous pipelines perform message handling and instruction execution.

Figure 1 illustrates an abstract PE architecture for massively parallel computation which has the features described above. We call this architecture RICA, Reduced Interprocessor-Communication Architecture.

![RICA Architecture](image)

Figure 1: Reduced Interprocessor-Communication Architecture.

3 Mechanisms in RICA

Figure 2 shows how RICA works.

In the interconnection network, each message should be transferred in a pipelined way, typically in way of virtual cut through, for keeping high the data supply rate to PE. It is buffered in a hard-wired queue, if necessary\(^1\). Message queueing should be overlapped with instruction execution.

In RICA, a message header has two addresses: an instruction address and a data address. The instruction address indicates the first instruction in the thread which is invoked by the message. The data address is the base address of the working area of the thread. It takes a single clock cycle to pass these two addresses to an instruction address register (IAR) and a data address register (DAR), respectively. In this way, invocation of a new thread without overhead is realized in RICA.

Before passing the address, if necessary, a micro-synchronization, is performed.

Instructions are executed in a RISC architecture. For fast local execution, the architecture might use caches and register files. As soon as the first data arrives, the first instruction is invoked without waiting for the arrivals of the rest of the message data. This is achieved by extending the scoreboard mechanism of super-scalar processor.

\(^1\)We do not assume FIFO message ordering.
Overlapped with execution stages, there is another pipeline dedicated to message generation and message output. Output messages will be buffered in an output queue.

The whole architecture illustrated in Figure 2 provides fast message handling and also provides a powerful communication–computation overlappings. However, from the viewpoint of cost–effectiveness and feasibility, it is not sufficient that the thread invocation and the fused pipeline are powerful. The structure should be as simple and efficient in a quantitative sense. To achieve this, we propose the following limitations:

1. Although this architecture has a multi-threaded pipeline, it is not necessary to overlap many threads, since it includes a fairly fast single-threaded pipeline within the multi-threaded pipeline. Overlapping two threads will be sufficient.

2. Special hardware for micro-synchronizations should not be complicated. Dyadic matching without associative memory accesses [2] [3] may be implemented directly in hardware, but complex micro-synchronizations should not.

3. It is not easy to add special data paths between synchronization logic and the storage (or a cache). Dyadic matching usually requires a read-modify-write which accesses the data storage twice. Instruction execution also accesses the data storage while read and write instructions are executed. We have to make a simple and fast storage interface (or cache interface) for micro-synchronizations. Multi-ported caches will be powerful in such a condition. However, it is not cost-effective to have a port for each type of
memory access.

RICA is a natural extension from RISC in the sense that:

1. it includes RISC architecture for executing sequential codes; and

2. the concept of RISC, simplification for efficiency, is adopted in parallel processing architecture in RICA.

4 Conclusion

This paper shows the concept of Reduced Interprocessor-Communication Architecture, RICA and the mechanisms for realizing it.


Near future plans are as follows. The authors are now building a next generation computer EM-X, an improved machine [14]. Many fundamental mechanisms such as resource management and I-structure handling are improved in EM-X. An EM-X prototype with hundreds of PEs will be operational in 1993.

Far future plans. The authors are developing a new massively parallel computer RWC-1. The details of RWC-1 will be reported in another paper.

Acknowledgements

The authors would like to thank Dr. Shimada, director of RWCP, Dr. Ohta, director of Computer Science Division in ETL, Dr. Furuya, manager of Massively Parallel and Neural Systems Division of RWCP, Dr. Yamaguchi, chief of Computer Architecture Section in ETL, for their heartfelt advice. They also would like to express thanks to Prof. Yonezawa and his group in the University of Tokyo and Prof. Arvind in MIT for their research contributions on realizing the programming model on RICA. They also would like to acknowledge the members in the RWC Tsukuba Research Center for the fruitful discussions.

References


