Datarol-II: A fine-grain massively parallel architecture

Kawano, Tetsuo
Department of Intelligent Systems, Kyushu University

Kusakabe, Shigeru
Department of Intelligent Systems, Kyushu University

Taniguchi, Rin-ichiro
Department of Intelligent Systems, Kyushu University

Amamiya, Makoto
Department of Intelligent Systems, Kyushu University

出版情報：Lecture Notes in Computer Science. 817, pp.781-784, 1994-07. Springer
バージョン：

Datarol-II : A Fine-Grain Massively Parallel Architecture

Tetsuo KAWANO, Shigeru KUSAKABE,
Rin-ichiro TANIGUCHI and Makoto AMAMIYA

Department of Information Systems, Graduate School of Engineering Sciences,
Kyushu University

Abstract. In this paper, we introduce the Datarol-II processor, that can efficiently execute a fine-grain multi-thread program, called Datarol. In order to achieve the efficient multi-thread execution by reducing context switching overhead, we introduce an implicit register load/store mechanism in the execution pipeline. A two-level hierarchical memory system is also introduced in order to reduce memory access latency. The simulation results show that the Datarol-II processor can tolerate remote memory access latencies and execute a fine-grain multi-thread program efficiently.

1 Introduction

In a massively parallel computer, one of the most important problems is latencies caused by remote memory accesses and remote procedure calls. Therefore, to eliminate the PEs' idling time caused by these latencies, a new architecture, that realizes more efficient context switching among fine-grain concurrent processes, should be developed.

We propose the Datarol-II processor architecture, that can perform fast context switching so that the latencies can become tolerable.

2 Architecture of Datarol-II Processor

Fig.1 shows the structure of the Datarol-II processor element (PE).

FU (Function Unit) executes the Datarol-II assembly instructions.
MU (Memory Unit) stores operand data for each instance[1].
AC (Activation Controller) controls the activation of threads.
CU (Communication Unit) handles input packets from other PEs.
RQ (Ready Queue) stores waiting threads.
SMU (Structure Memory Unit) offers I-structure memory access.

In Datarol architecture, each instance has its own logical registers. MU holds these register values for each instance in Operand Memory (OM). FU has several

[1] An instance in Datarol corresponds to a process

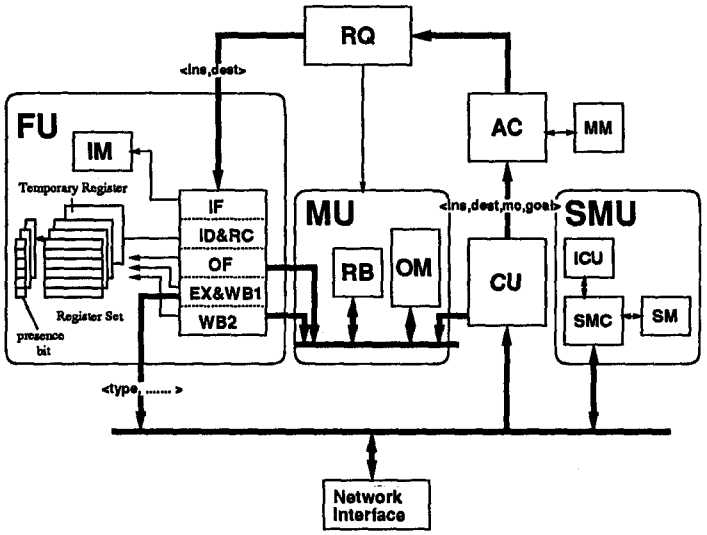

Fig. 1. The Datarol-II Processor Element

physical register sets. Before each thread execution, FU has to read logical register values from MU and store them into one of the register sets. In each physical register, there is a presence bit that is checked every time FU reads the register value. If the presence bit is off, the register value is not read from the physical register, but from MU. Since this register loading mechanism is embedded in the FU pipeline, FU does not have to issue *load* instructions to load logical register values into physical registers. Moreover, when the physical register values are overwritten, these changes are automatically reflected to MU, therefore there is no need for explicit *store* instructions. Since this implicit load/store mechanism considerably reduces context switching overheads, the Datarol-II processor can perform efficient context switching among fine-grain threads.

In addition, a high speed memory, called Register Buffer (RB), is introduced in MU to reduce memory access time. The register values of several instances are loaded into RB by RQ request before being accessed from FU. By means of this mechanism, when FU starts a new thread, it can read the register values very quickly from RB.

3 Evaluation

To evaluate the Datarol-II processor, we developed a software simulator. In this section, we show the simulation results of a Datarol-II machine consisting of Datarol-II PEs connected to a 2-D torus network. We used the **Matrix** benchmark program : a 64 × 64 matrix multiplication program executed by 16 × 16 PEs. The innermost loop corresponds to each instance, so there are 64 × 64 concurrently executed instances (4 × 4 instances for each PE).

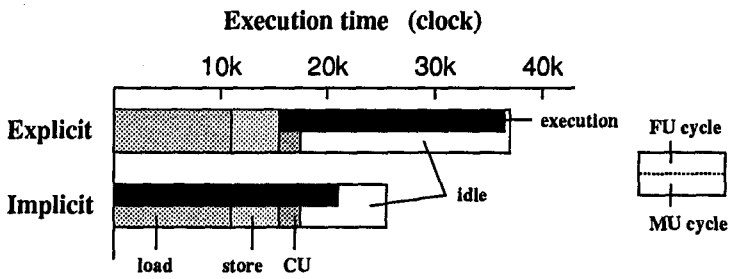

Fig. 2. Effect of implicit register load/store mechanism

Effect of Implicit Register load/store : In a Datarol-II processor, register values are loaded from MU and also automatically stored into MU at execution time. Now, we show the effect of this implicit register load/store mechanism by comparing the results with those of explicit *load/store* instructions(See **Fig.2**). In the explicit case, *load/store* instructions are inserted in the program, then every thread loads the register values from MU by means of *load* instructions at the head of the thread, and stores the required register values into MU at the end of the thread. On the other hand, in the implicit case, register values are loaded and stored automatically by a hardware mechanism without load/store instructions. In **Fig.2**, we can see that in the explicit case the load/store instructions account for a considerable increase in execution time. On the other hand, in the implicit case, MU accesses are executed in the FU pipeline, overlapping with the thread execution. Therefore, the context switching overhead caused by *load/store* instructions, is eliminated, and the total execution time is significantly shorter.

Latency Tolerance : To confirm the latency tolerance of Datarol-II, we vary both the throughput of the PE-PE connection links and the minimum latency needed to sent a packet to an adjacent PE. **Fig.3** shows the utilization ratio of the execution units with varying network parameters. To estimate the execution of the **Matrix** program, a throughput of 0.4 packets per clock [2] is needed for each PE-PE connection, and the maximum tolerance of remote memory access latency is also estimated to be 300 clocks [3] to keep the execution units utilization ratio high. From the left chart, it can be seen that the utilization ratio of the execution units is high, when the throughput is sufficient. In the right chart, we significantly increase the latencies and observe the utilization ratio of the execution units. When the average remote memory access latency is less than

[2] Each iteration has 20 instructions including 2 remote memory accesses. The average remote memory access distance is 8 hops, and there are 2 links per PE. Then, a throughput of $(2 \times 8/2)/20$ packets per clock is needed.

[3] Each PE executes 16 instances concurrently. Then the maximum tolerable latency is estimated to be $20 \times (16 - 1)$ clocks.

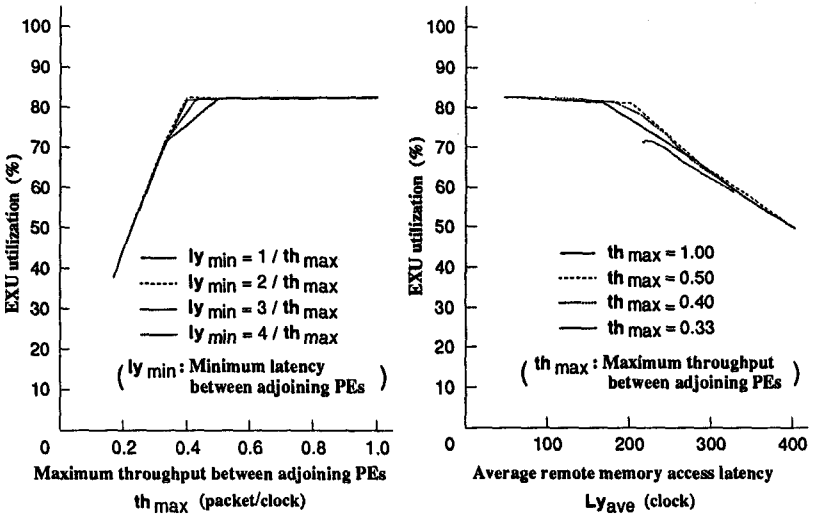

Fig. 3. Datarol-II performance with varying network parameters

200 clocks [4] and the throughput is sufficient, the utilization of execution units is high. According to these results, it is clear that Datarol-II has good latency tolerance.

4 Conclusions

In this paper, we proposed the Datarol-II processor architecture, which can eliminate the processors' idling time caused by remote memory access latency by performing more efficient context switching. We have evaluated the Datarol-II processor architecture with a software simulator. The simulation results showed that (1) the implicit register load/store mechanism eliminates the overhead of context switching, (2) when the network throughput is sufficient for the target program, the Datarol-II processor tolerates latencies, and its execution is highly efficient.

References

1. M.Amamiya and R.Taniguchi, "Datarol: A Massively Parallel Architecture for Functional Language", Proc. SPDP, pp.726-735, (1990)
2. S.Kusakabe, T.Hoshide, R.Taniguchi and M.Amamiya, "Parallelism Control and Storage Management in Datarol PE", Proc. IFIP World Congress, Vol.1, pp.535-541, (1992)

[4] Almost all the latencies are less than 300 clocks