A low power embedded dataflow coprocessor

Yijun Liu, Steve Furber
APT group, School of Computer Science
University of Manchester, Oxford Road, Manchester, UK, M13 9PL
yijun.liu@cs.manchester.ac.uk
steve.furber@manchester.ac.uk

Abstract

Power consumption has become one of the most important concerns in microprocessor design. However, the potential for further power-saving in microprocessors with a conventional architecture is limited because of their unified architectures and mature low-power techniques. An alternative way is proposed in this paper to save power — embedding a dataflow coprocessor in a conventional RISC processor. The dataflow coprocessor is designed to execute short code segments very efficiently. The primary experimental results show that the dataflow coprocessor can increase the power efficiency of a RISC processor by an order of magnitude.

1 Introduction

Because of its high flexibility, von-Neumann architecture is most commonly used. In a von-Neumann architecture, every single operation of data processing is accompanied by at least one instruction fetching and decoding. So the bandwidth of instructions is much bigger than that of data, which is referred to as von-Neumann bottleneck. Fixed-length instruction formats and a load-store architecture used by RISC processors result in an even lower code density than that of CISC processors. The big instruction bandwidth means that von-Neumann processors spend much more energy in instruction operations such as fetching and decoding than data processing. The sequential execution also makes von-Neumann machines 'remember' only one instruction at one time which means, after the execution of one instruction, a processor will automatically fetch another instruction and forget the instruction it just executed. This phenomenon makes conventional microprocessor very inefficient to execute iterative program segments with few instructions. Since small loops dominate the actual program traces of processors [1], the power dissipated by rearranging a program counter, refetching and decoding instructions is significantly wasted. As a result, the conventional von-Neumann microprocessors using a von-Neumann architecture may not be very efficient in terms of power consumption because of their unique architecture.

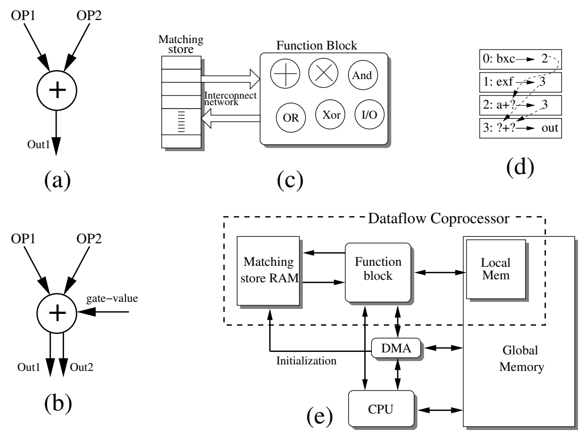
To improve microprocessors' power-efficiency, people either use hardware additions, such as a loop cache, or use new architectures, like reconfigurable processors. As an alternative, we propose another architecture — embedding a dataflow coprocessor in a RISC processor.

2 The proposed dataflow architecture

Normally, dataflow machines [2] are built to achieve high performance by optimizing their hardware for fine-grained parallel computation. However, in this paper, a dataflow engine is embedded in a conventional microprocessor (main processor) to specifically improve its power efficiency. The dataflow engine has no ability to fetch instructions and needs the CPU to initialize it. However, after initialization, the dataflow engine can finish the calculation independently and send the result(s) back to the CPU. If the CPU wants to execute the program again, it does not need to initialize the dataflow coprocessor again. The only thing it need to do is sending inputs and waiting for outputs, so the energy used for refetching instructions is greatly reduced.

In a dataflow machine, operations are executed in an order determined by the availability of input data and spaces to put results. A dataflow operation should satisfy two requirements: **(1)** Each input port of a function unit should have a valid data token; **(2)** Each output port (destination) of a function unit should be empty, which means that new results must not overwrite the old ones it generates before if its consumers are not fast enough.

Figure(a) illustrates a basic dataflow element, which has two inputs and one output. Once a dataflow element satisfies the two requirements mentioned above, the two input operands will be sent to a function unit. The function unit will finish the calculation and send the result to input(s) of other dataflow elements. In real programs, conditional



instructions and branches (which duplicate output results) are very common. We propose a new basic dataflow element as shown in Figure(b). The new dataflow element has three inputs — two operands and one conditional Boolean value (gate-value) — and two output branches. Each output branch has a Boolean value. The result is only sent to the branch(es) having the same Boolean value as gate-value. The new dataflow element can easily support conditional instructions and branches.

Figure(c) shows the primary architecture of the dataflow engine. It includes two major parts — a matching store RAM and a function block. The function block contains several function units, such as adders and multipliers, and an I/O module through which the dataflow engine communicates with the outside. The RAM acts as a big switch, which sends operands to the corresponding function units and relays results back to the destination addresses inside the RAM. Each row of the RAM can contain an instruction. Using this architecture, function units can be used by any instructions inside the matching store. Figure(d) gives an example of $x = a + (b \times c) + (d \times e)$ to show how this architecture works. If all input operands of an instruction in the matching store are valid and its destinations are free, the instruction is 'fired'. The input operands are sent to the corresponding function unit. After the execution of the function unit, the result will be sent to the destination addresses to 'fire' other instructions.

To support data-stream computations, a block of local memory is included in the dataflow architecture. Apparently, it is very inefficient if every single data item moving between the local memory and the global memory goes through the function block and the main CPU. The coprocessor controls a DMA to exchange a big block of data between the local memory and the global memory when necessary. For the CPU, the local memory inside the dataflow engine is just a part of the global memory and it shares the address space with other parts of the global memory. The CPU can access any addresses of the local memory. Figure(e) illustrates an architecture that implements an efficient communication between the dataflow coprocessor and a conventional RISC processor. By the new architecture, the dataflow coprocessor can cope with a big data stream without the main processor's assistance.

3 Experimental result

One prototype of 16-bit dataflow engine containing a matching store of 32 instructions and 8Kb internal data memory was implemented. Five programs were used as benchmarks to evaluate the dataflow coprocessor as follows: $SUM = \sum_{i=1}^{100} i$; a FIR algorithm; an IIR algorithm; a FFT algorithm; and an IDEA encryption algorithm. The prototype is simulated using a $0.18\mu m$ technology and a 1.8 volt supply voltage. These benchmarks are also simulated in an ARM processor — AT91R40008. The comparison is shown in Table1. The simulation of the dataflow engine is based on schematic avoiding wire capacitances, we estimate power results based on layout will increase by a factor of two. Even through, the dataflow engine can improve the power efficiency of AT91R40008 by an order of magnitude.

Table 1. Power comparison

Benchmark	SUM	FIR	IIR	FFT	IDEA
ARM (nJ)	292.0	14.6	15.3	21.9	175.9
Coprocessor(nJ)	12.7	0.61	0.67	0.86	8.05
Ratio	23.0	24.1	22.9	25.5	21.9

4 Conclusions

A low power prototype architecture is proposed in this paper — a dataflow coprocessor embedded in a RISC processor as a 'computation engine' to improve power-efficiency. The experiments show that the dataflow coprocessor may increase the power efficiency of a RISC processor by an order of magnitude. The reason for the high power-efficiency of the dataflow coprocessor is due to its small size and ability to memorize the most commonly used programs.

References

[1] L. H. Lee, et al. "Instruction fetch energy reduction using loop caches for embedded applications with small tight loops", *ISLPED'99*.

[2] A. H. Veen. Dataflow machine architecture. *ACM Computing Surveys (CSUR)*, Dec. 1986.