An Event-Driven Massively Parallel Fine-Grained Processor Array

Declan Walsh and Piotr Dudek
School of Electrical and Electronic Engineering
The University of Manchester, UK
declan.walsh@postgrad.manchester.ac.uk; p.dudek@manchester.ac.uk

***Abstract*—A multi-core event-driven parallel processor array design is presented. Using relatively simple 8-bit processing cores and a 2D mesh network topology, the architecture focuses on reducing the area occupation of a single processor core. A large number of these processor cores can be implemented on a single integrated chip to create a MIMD architecture capable of providing a powerful processing performance. Each processor core is an event-driven processor which can enter an idle mode when no data is changing locally. An 8 × 8 prototype processor array is implemented in a 65 nm CMOS process in 1,875 µm × 1,875 µm. This processor array is capable of performing 5.12 GOPS operating at 80 MHz with an average power consumption of 75.4 mW.**

Keywords—MIMD; many-core; event-driven; fine-grained; processor array; parallel processing; scalable

I. INTRODUCTION

In recent years the semiconductor industry has shifted towards an increased number of processor cores on an integrated chip operating in parallel to achieve more energy efficient processing rather than increasing clock frequencies of processors. These multi-core architectures can have a small number of cores on a chip such as conventional dual-core, quad-core, and 8-core processors etc. [1, 2]. Further to these, there can be many-core processors with a large number of processor cores integrated on a single chip [3-5]. Other forms of multi-processors include Graphics Processing Units (GPUs), and fine-grained parallel processor arrays or cellular processor arrays operating using the SIMD (Single Instruction Multiple Data) processing paradigm. This includes FPGA implementations of processor arrays [6-8], and vision-chips [9-11] which integrate a very large number of processing elements onto a chip dedicated to perform image processing tasks. By simplifying individual processors to provide less complex operations, and to consume less area on a chip, a large number of these processing cores can be integrated onto a chip. These architectures may have less powerful individual processing cores with a limited functionality, however, with a large number of them operating in parallel, often reaching tens of thousands of processors, a powerful processing performance can be achieved.

The network topology of these processors [6-11] is commonly a two-dimensional mesh network where each processor is connected to its four nearest direct neighbours and only local communications are performed. This network topology reduces the long wire interconnects which also reduces power and enables operations at higher frequencies.

The SIMD implementations are useful for performing tasks which require data-level parallelism. However, for tasks which require more complex processing than simple pixel-parallel operators, e.g., event-driven networks [12], message-passing inference networks [13], or even some low-level image processing operations [14, 15], SIMD processing may not be optimal and a more flexible MIMD (Multiple Instruction Multiple Data) solution is more suitable.

This paper presents an event-driven fine-grained MIMD processor array. A single core is designed to be simple and compact with its own instruction memory. A large number of processor cores can be arranged in a two-dimensional mesh to form a processor array. The processors can operate in a low-power idle mode with a near instantaneous wake-up. Each processor can operate on its own local data and execute its own instruction and the processors are intended to closely cooperate to solve a single problem.

II. MULTI-CORE ARCHITECTURE

The multi-core processor array architecture is shown in Fig. 1 in which each processor core communicates locally with its four nearest direct neighbours. With a modular architecture such as this, rows or columns of cores can easily be added to the design making it extremely scalable.

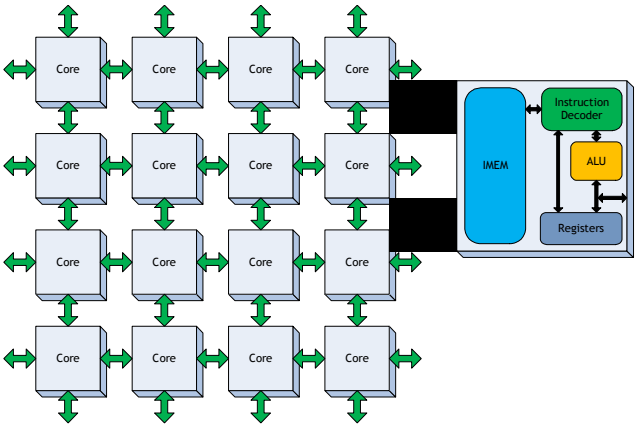

Fig. 1: Multi-core architecture in two-dimensional topology

This work has been supported by EPSRC; Grant No.: EP/H023623/1.

A. Processor Core Architecture

The individual processor core is shown in Fig. 2 with control signals omitted for purposes of clarity. Each processor core contains an instruction memory for storing the program to be executed up to a maximum of 256 instruction words, an instruction decoder, an 8-bit ALU, a program counter, a register bank, and an event-driven mechanism enabling the processor core to detect particular events that occur locally to the processor. The processor core is controlled by a 20-bit instruction word where each instruction can perform an ALU operation, or control the program flow using conditional or unconditional jump instructions.

The instruction memory is a synchronous read single-port (SP) RAM. An instruction word is fetched in one clock cycle, then decoded and executed in the second clock cycle with the processor capable of executing one instruction per clock cycle. These instructions are decoded by the instruction decoder which issues the signals for performing the required operations.

The register bank is comprised of sixteen 8-bit registers; nine of these are general-purpose registers, four are used for neighbourhood communication, one is used as a bit-mask for particular ALU operations, one input register, and one output register. Each of the four 'NEWS' registers used for neighbourhood communication (NEWS0, NEWS1, NEWS2, NEWS3) is connected to all four neighbouring processor cores to the north, south, east, and west directions, and can be read by those cores. Therefore each processor core can read a total of sixteen NEWS registers from its four neighbours, four from each neighbour. The input register (SENSOR) is read-only and is written to externally and the output register (ACTUATOR) is for external output.

Operations are performed by the ALU. One of the data inputs can come from one of the 16 local registers, one of the 16 neighbouring NEWS communication registers, a direct 8-bit value encoded as part of the instruction word, or an 8-bit value from the event vector. The second input must come from one of the 16 local registers. The ALU can perform one of eight operations per clock; add (with carry), shift left, shift right, compare, AND, OR, XOR and NOT. The output of the ALU is connected to the inputs of the local registers. There are also two 1-bit outputs which connect to two flag registers, the Carry flag (C) and the Zero flag (Z) respectively. These flags can be used for conditional branch operations.

B. Event-Driven Component

In signal processor arrays, when an algorithm is executing in a loop, the same instructions are executing iteratively. If input data is not changing and computations are being performed on the same data over and over, then this processing will always produce the same result, effectively making the processing redundant.

To reduce this processing, and also to reduce power consumption, an event-driven mechanism is implemented in the architecture. After a sequence of instructions has been executed, and no local or input data has changed, and no neighbouring NEWS communication register data has changed, i.e., no further processing is required, the processor core can enter an idle state where no instructions are executed. This can be done using a 'halt' instruction which disables the program counter if no events need attending. The processor suspends operation entering a low-power idle mode. The hardware Event Vector then indicates when an event (data being written to particular registers) has occurred with each bit in the vector corresponding to an event on one register. There are six event bits (with the two additional bits tied to logic '0'); input register (SENSOR), four neighbouring NEWS registers, and the output register (ACTUATOR).

The operation of the event-driven mechanism is shown in Fig. 3. This example begins with the processor executing instructions. When a halt instruction is executed, the program counter stops incrementing and enters the idle mode. When an event (SENSOR event) is detected by the Event Vector, the program counter begins counting again resulting in execution of the following instructions, beginning with clearing the event signal, similar to an interrupt service routine.

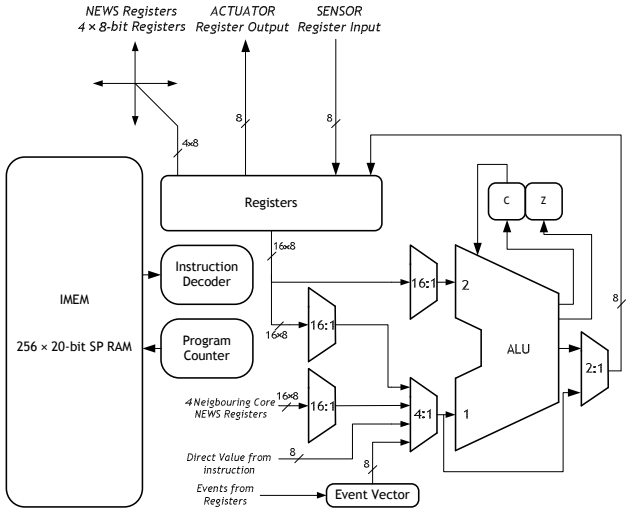

Fig. 2: Processor core architecture

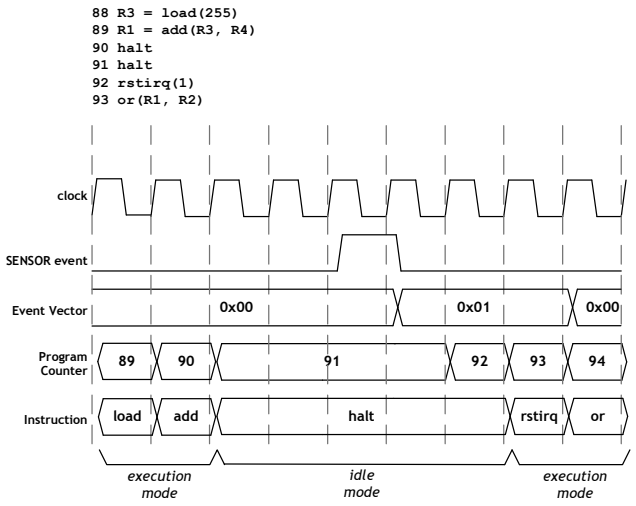

Fig. 3: Timing diagram of event-driven mechanism

C. Chip-To-Chip Communication

The design is scalable and array size on chip can be simply increased by tiling additional processor cores. It is also possible to tile multiple chips to increase the size of an array in a system. For example, if a chip implements an 8 × 8 array of processors, then an 8 × 8 array of chips can be used to implement a 64 × 64 array of processors. To facilitate this, a chip-to-chip communication block is implemented. Due to I/O pad constraints, a chip-to-chip communication block serializes all the data from the NEWS registers to be transmitted across the chip boundary. It is then deserialized on the other side of the boundary and synchronized to the new chip clock. This is done using a total of four wires for transmission in one direction as shown in Fig. 4. Data is transmitted across the boundary synchronous to the clock from the transmitting side of the boundary and then synchronized with the clock on the receiving side of the boundary. These blocks form an identical copy of the NEWS communication registers on the receiving side as on the transmitting side. Event signals for the NEWS communication registers are also created on the receiving side. The same blocks are required for receiving data from a neighbouring chip. These blocks are repeated for all four chip boundaries.

D. I/O Interface

To facilitate the input and output of data to and from the processor array, an I/O interface is implemented. This is used for writing instructions to the instruction memories, writing data to the processor cores, and reading data from the processor cores. This is shown in Fig. 5 for eight processors per row where data is written to processor input registers and read from processor output registers using shift registers. In Fig. 5(a), processor data is shifted in eight bits at a time until one row of data can be shifted down to top row of the processor array and is repeated until all rows have been written to. This is controlled by a finite state machine (FSM). In Fig. 5(b), the reverse process is applied for reading data from the array using eight bits, i.e., data is shifted up one row, and is then shifted out. This is done until all data is shifted out. In addition, the instruction data is shifted into a 20-bit shift register serially through a 1-bit interface and then written to the instruction memories each time a 20-bit instruction word is shifted in. The instructions can be written to all cores in parallel, rows or columns of cores, or individual cores.

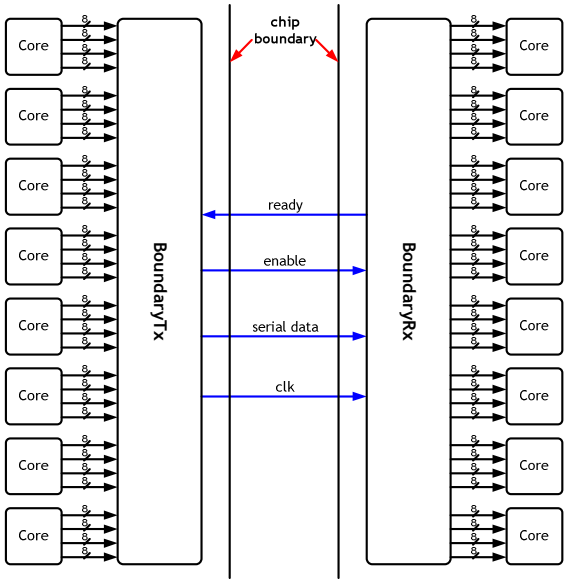

Fig. 4: Chip-to-chip communication block

III. Implementation

An 8 × 8 processor array including communication blocks and I/O interface, has been designed and tested using RTL code synthesized on an FPGA which occupies 23,264 LUTs and 17,406 flip-flops on a Xilinx Spartan-6 XC6SLX150 FPGA with an operating frequency of 42 MHz. This RTL code has then been synthesized, and placed-and-routed to form an ASIC implementation of the processor array.

The design is implemented using a CMOS 65 nm process with 1 polysilicon layer and 8 metal layers. A low leakage, regular voltage library of standard cells is used which have a nominal operating voltage of 1.2 V while the I/O cells have an operating voltage of 2.5 V. The design is implemented on a single ASIC block of 1,875 µm × 1,875 µm.

The RTL code is synthesized, including clock gating, to standard cells, and a place-and-route process is performed to create a layout of the architecture. The overall design is separated into smaller blocks with each block implemented individually, i.e., processor core, communication block, I/O interface. Therefore, one processor core is implemented using RTL to GDSII flow and by connecting – or tiling – multiple cores together, a large array is created. The same process is followed for all blocks in the design. The area occupation of each block is shown in Table 1. An 8 × 8 processor array is implemented in 1,248 µm × 1,232 µm. These blocks are all

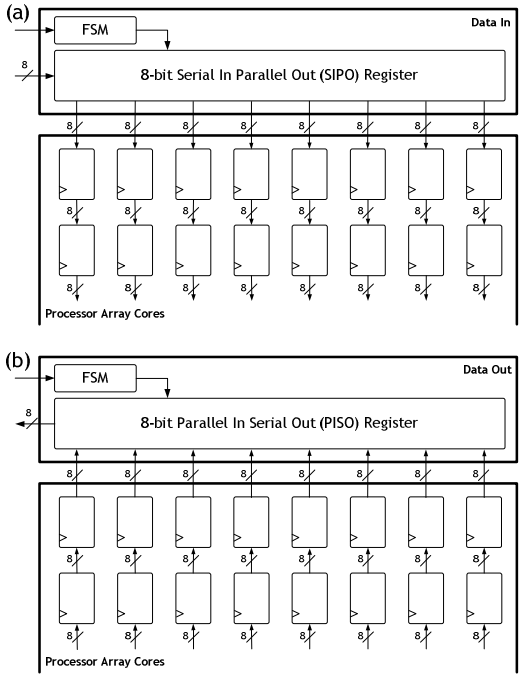

Fig. 5: (a) Data input; and (b) data output for processor array

connected together using a layout editor and the full design verified using LVS verification to compare the layout to a Verilog netlist. The full layout of the 8 × 8 tiled processor array and the additional components interconnected, with a chip micrograph inset is shown in Fig. 6.

Timing analysis results show that each block in the design can operate at a clock frequency of 80 MHz. Power analysis shows the average power consumption of each processor core to be 1.1 mW with each communication block consuming approximately 1.2 mW and the I/O interface consuming an average of 0.2 mW.

The implemented 8 × 8 processor array is designed to offer a peak performance of 5.12 GOPS (giga operations per second) operating at 80 MHz with an average power consumption of 70.4 mW for the processor array, or 72 GOPS/W. Including the communication blocks and I/O interface, the power consumption of the implemented design is 75.4 mW.

IV. CONCLUSION

An event-driven fine-grained parallel processor array is presented which is implemented in a 65 nm CMOS process. Each processor core occupies 156 µm × 154 µm and includes its own instruction memory. The complete 8 × 8 processor array with peripheral communication components is implemented on a 1,875 µm × 1,875 µm ASIC block and can offer a peak performance of 5.12 GOPS.

Due to the scalability of the architecture, it is expected that arrays of much larger resolution could be implemented over a larger area. Using the same 65 nm CMOS process, an array of 64 × 64 processors could be implemented in less than 10 mm × 10 mm which, operating at 80 MHz would offer a peak performance of 327.68 GOPS and consume 4.5 W of power. Furthermore, with technology scaling, it can be expected that even larger arrays could be implemented. In a 28 nm CMOS process it can be extrapolated that a 128 × 128 array can be implemented in less than 10 mm × 10 mm offering a performance of 4.9 TOPS if the clock speed is increased to 300 MHz.

Table 1: Area occupation of each block

	Horizontal Width (µm)	Vertical Height (µm)	Total Area (µm²)
Processor Core (×64)	156.4	154.2	24,117
North/South Communication	1,243.3	62.4	77,580
East/West Communication	61.6	1,196.4	73,662
I/O Interface	426.9	51.6	22,029

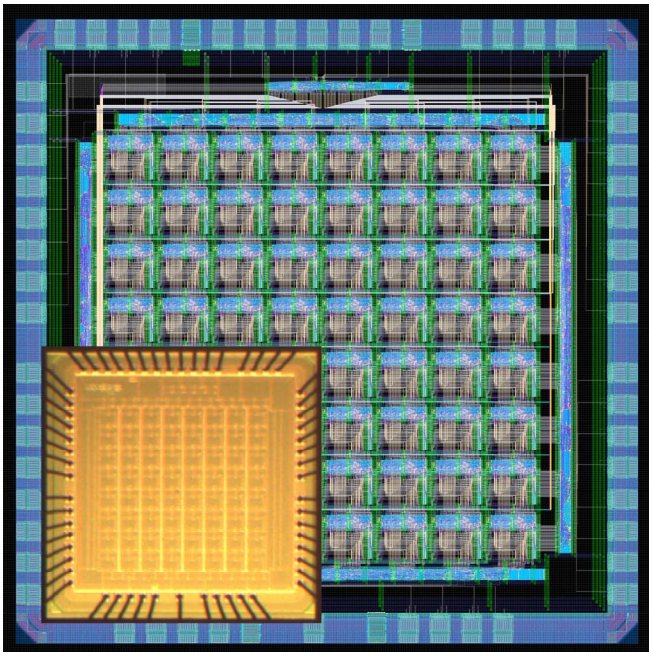

Fig. 6: Layout of 8 × 8 processor array with chip communication blocks on each boundary and I/O interface above; and inset, a chip micrograph.

REFERENCES

[1] S. Rusu et al., "A 65-nm dual-core multithreaded Xeon processor with 16-MB L3 cache," IEEE Journal of Solid-State Circuits, vol. 42 (3), 2007, pp. 17-25.

[2] S. Rusu et al., "A 45 nm 8-core enterprise Xeon processor," IEEE Journal of Solid-State Circuits, vol. 45 (1), 2010, pp. 7-14.

[3] A.M. Jones and M. Butts, "Teraops hardware: a new massively parallel MIMD computing fabric IC," Proceedings of IEEE Hot Chips Symposium, Stanford, CA, 2006.

[4] A. Olofsson, "A closer look at the Epiphany-IV 28nm 64-core coprocessor," 8th International Conference on High Performance and Embedded Architectures and Computers, Berlin, Germany, 2013.

[5] S. Bell et al., "Tile64 processor: a 64-core SoC with mesh interconnect," IEEE International Solid-State Circuits Conference (ISSCC 2008), San Francisco, CA, 2008, pp. 88-89.

[6] A. Nieto, V.M. Brea and D.L. Villarino, "SIMD array on FPGA for B/W image processing," International Workshop on Cellular Neural Networks and Their Application (CNNA 2008), Santiago de Compostela, Spain, 2008, pp. 202-207.

[7] Y.J. Li, W.C. Zhang and N.J. Wu, "A novel architecture of vision chip for fast traffic lane detection and FPGA implementation," IEEE Intl Conf on ASIC (ASICON 2009), Changsha, China, 2009, pp. 917-920.

[8] D. Walsh and P.Dudek, "A compact FPGA implementation of a bit-serial SIMD cellular processor array," IEEE Workshop on Cellular Nanoscale Networks and Applications (CNNA 2012), Turin, Italy, 2012.

[9] S. Carey et al., "A 100,000 fps vision sensor with embedded 535GOPS/W 256×256 SIMD processor array," Symposium on VLSI Circuits (VLSIC 2013), Kyoto, Japan, 2013, pp. C182-C183.

[10] A. Lopich and P. Dudek, "A general purpose vision processor with 160×80 pixel-parallel SIMD processor array," IEEE Custom Integrated Circuits Conference (CICC 2013), San Jose, CA, 2013.

[11] Á. Rodríguez-Vázquez et al., "The Eye-RIS CMOS vision system," In: H. Casier, M. Steyaert, and A.H.M. Van Roermund, eds. Analog Circuit Design. Netherlands: Springer, 2008, pp. 15-32.

[12] Y. Liu, G. Xie, P. Chen, J. Chen and Z. Li, "Designing an asynchronous FPGA processor for low-power sensor networks," International Symposium on Signals, Circuits and Systems (ISSCS 2009), Iasi, Romania, 2009, pp. 261-266.

[13] M. Cook, L. Gugelmann, F. Jug, C. Krautz, and A. Steger, "Interacting maps for fast visual interpretation," in Proceedings of the International Joint Conference on Neural Networks (IJCNN 2011), San Jose, CA, 2011, pp. 770–776.

[14] A. Lopich and P. Dudek, "Global operations on SIMD cellular processor arrays: towards functional asynchronism," International Workshop on Computer Architectures for Machine Perception and Sensing (CAMPS 2006), Montreal, Canada, 2006, pp.18-23.

[15] P. Mroszczyk and P. Dudek, "Trigger-Wave Asynchronous Cellular Logic Array for Fast Binary Image Processing," IEEE Transactions on Circuits and Systems - I: Regular Papers (in print).