ISSN: 2321-2152

IJMECE

International Journal of modern electronics and communication engineering

E-Mail
editor.ijmece@gmail.com
editor@ijmece.com

www.ijmece.com

ISSN23212152www.ijmece.com

Vol 8, Issue 4 Oct 2020

Accelerating Biomedical Image Processing Applications using BioThreads, a Novel VLIW-Based Chip Multiprocessor

ARAVA SUMAN KUMAR REDDY [1], S. AMALA[2]

Abstract

Here, we talk about how the BioThreads system-on-chip multiprocessor may be used to speed up imaging pho to plethysmography (IPPG) and other biomedical signal processing applications. Bio Threads, a multiprocessor based on the open-source VLIW hardware LE1, effectively manages parallelism at the instruction, data, and thread levels. It also provides a new technique for the dynamic generation and allocation of software threads to uncommitted processor cores by implementing important POSIX Threads primitives directly in hardware, as custom instructions. An FPGA board and a host system are utilized with a high-speed image-acquisition system to speed up the computation of an oxygen saturation map of live tissue in this work. The results show that as the number of hardware threads increases, the core kernels of the blood perfusion assessment run almost linearly faster. Both standard-cell and FPGA technologies were used to create the Bio Threads processor, with full real-time performance being attained with 4 cores on the former and 10 dual-issue cores on a mid-range Xilinx Virtex6 device. Scalability of the suggested method to a state-of-the-art FPGA vendor supplied soft CPU core was shown by an 8-core LE1 VLIW FPGA prototype of the system achieving 240 times quicker ex elution time than the scalar Micro blaze processor. Biomedical image processing, FPGAs, IPPG, microprocessors, and multicore processing are some of the terms that may be used as index terms.

BIOMEDICAL INTRODUCTION AND INSPIRATIONAL FACTORS

The ability to make important decisions and carry out medical interventions in real time based on hard facts, derived in real time from physiological data, is essential for in-vitro and in-vivo assessment [1, 2]. This is because real-time execution of signal processing codes is the key to enabling safe, accurate, and timely decision-making. Several imaging techniques, such as laser Doppler [3, 4], optical coherence tomography [5, 6], and imaging photoplethysmography [5, 6], have been presented in recent years for use in biomedical image processing. However, a real-time biomedical image processing system based on very large-scale integration (VLSI) systems technology is necessary for any of these methods to reach their full potential. For instance, the frame size and frame rate employed by an IPPG system are strongly connected to the quality and availability of physiological information. The degree to which such a system can function in real time is crucial to its usefulness from the standpoint of its end users, and practical implementations of the system aspire to be independent and portable to realize its full potential. Here, cutting-edge computer architecture ideas used in high-performance consumer and telecoms systems-on-a-chip (SoC) [7] could provide the necessary data streaming and execution bandwidth for the real-time execution of algorithms that would otherwise be executed offline (in batch mode) using more conventional techniques and platforms (such as sequential execution on a PC host). The expected performance advantages are shown by the fact that our research platform's single-core design is six times faster than the performance of a scalar embedded processor, as shown in a quantitative comparison presented in this paper (results and discussion). Standard-cell (ASIC) [8] and field-programmable gate array (FPGA) (9) based embedded systems often make use of scalar embedded processor cores with a defined instruction-set-architecture (ISA). These processors are an excellent middle ground for doing low-complexity signal processing jobs, user interface processing, low-level/bandwidth protocol processing, and embedded operating system (eOS) functions. Unfortunately, most signal processing application backbone techniques demand high-throughput execution and high-bandwidth data transport, two areas where they fall significantly short. In [10], the capabilities of three such scalar engines aimed towards field-programmable gate arrays (FPGAs) are compared in an innovative way.

Assistant professor1,2
DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
P.B.R.VISVODAYA INSTITUTE OF TECHNOLOGY & SCIENCE
S.P.S.R NELLORE DIST, A.P , INDIA , KAVALI-524201

ISSN23212152www.ijmece.com

Vol 8, Issue 4 Oct 2020

THE POWER OF BIOTHREADS

The LE1 open-source processor serves as the foundation for the Bio Threads CMP, which adds execution primitives to enable fast image processing and dynamic thread allocation and mapping to uncommitted CPU cores. The Bio Threads design outlines a multiprocessor with shared and distributed memory. The Bio Threads multiprocessor architecture is a hybrid between explicit and implicit threading models because it requires the user to name software threads in the code but also provides hardware support for creating, managing, synchronizing, and terminating threads. Key Thread primitives like pthread_create/join/exit and pthread_mutex_init/lock/try lock/unlock/destroy are fully supported in hardware by the LE1's thread management. The thread control unit (TCU) is a hardware block that initiates and terminates the execution of multiple LE1 cores and is responsible for servicing specialized hardware requests. Multiple contexts (cores) compete for access to the TCU, a point of explicit serialization where Thread's command requests are serialized internally and served in turn. Because the TCU handles low-level Thread services, the LE1 doesn't need an OS; a typical pthread_create instruction takes fewer than 20 clocks to execute. This sets the LE1 VLIW CMP apart from all other VLIW multicore processors and is its defining characteristic. A high-level diagram of the Bio Threads engine is shown in Fig. 1. The scalar platform is comprised of the service processor (the Xilinx Micro blaze, 5-stage pipeline 32-bit CPU), its subsystem based on the Core Connect [38] bus architecture, and the LE1 chip multiprocessor (CMP) which runs the signal processing kernels. The internal structure of a single LE1 context is shown in Fig. 2. Instruction Fetch Engine, Execution Core, Pipeline Controller, and Load/Store Unit Make Up the Central Processing Unit. Both an instruction cache and a tightly connected instruction RAM (IRAM) are viable options for the IFE's instruction storage and execution mechanism. These are accessed at the beginning of each cycle, and they give back a long instruction word (LIW) that has many Rescopes in it.

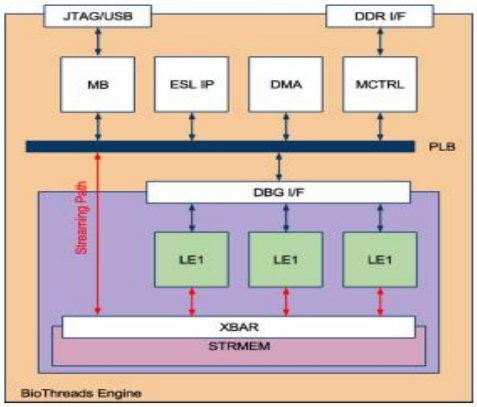

Fig. 1. Bio Threads Engine showing LE1 cores, memory subsystem and overall architecture.

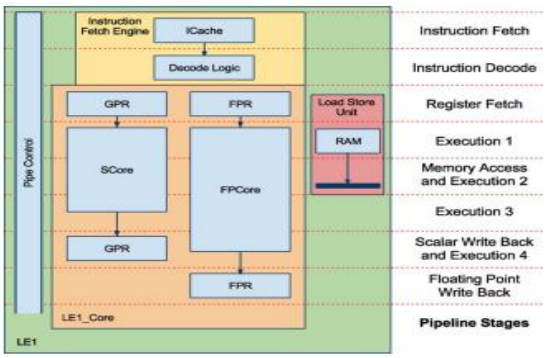

Fig. 2. Open-source LE1 Core pipeline organization.

Dispatch

The IFE controller handles debugging into the I Cache/IRAM and interfaces with external memory for reloading the ICache. An optional branch predictor unit is available for the IFE, with implementations in both set-associative and fully-associative (CAM-based) architectures based on the 2-bit saturating counter technique (Smith predictor).The LE1_CORE section contains the primary data routes used by the CPU during execution. The number of clusters is adjustable, and each cluster has its own set of registers. There are three types of processing units (cores) in a cluster: integer (SCORE), custom instruction (CCORE), and floating point (FPCORE). However, a consistent exception resolution point is shared by the integer and floating-point data paths to enable a precise exception programming paradigm. PIPE_CTRL is the main control mechanism. It is a chain of pipelined state machines that coordinates the execution data paths and keeps tabs on the general instruction flow. During debug activities, PIPE_CTRL handshaking the host and maintaining CPU control registers and decoding logic. The LE1_CORE primarily accesses system memory through the LSU. It can access the shared data memory (STRMEM) and perform memory operations at a rate of up to ISSUE_WIDTH (VLIW architectural width) per cycle. For up to 8 clients and 8 banks (8 8), the latter is a 2- or 3-stage pipelined cross-bar design that grows relatively well (in terms of speed and area), as illustrated in Tables I and II. Microarchitecture improvements are possible since the number of LSU customers (LSU_CHANNELS) does not have to match the number of such banks. Fig. 3 depicts this STRMEM block configuration. Finally, as seen in Fig. 3, numerous processing cores may be created in a CMP configuration to enable the use of shared-memory TLP. A dual-LE1, single-cluster Bio Threads system talking to a shared RAM for streaming data is shown in the diagram.

Thread's control unit

The TCU handles hardware context management and dynamic allocation. The allocation of software threads to execution resources (HC, hyper contexts) is managed by this series of hierarchical state machines. All running hyper contexts and the host may make Threads queries to it. It is a synchronization point for all running hyper contexts and keeps track of a set of hardware (state) tables. The TCU lives in the DEBUG_IF (Fig. 1), where it takes advantage of the pre-existing hardware infrastructure to halt, restart, read, write, and interact with the host in order to di erectly manage the operation mode of every hyper context (HC). The Context TCU, which controls the local (per-context, in the PIPE_CTRL block) delivery of Threads instructions to the centralized TCU, is a crucial building component in thread management. One of the active HCs in a given context decides each clock which commands may be executed on the context's TCU; once a command is approved, control is transferred to the TCU in the DBG_IF. When the Threads command is complete, the Context TCU sends the return values back to the requesting HC. Thread control structure for a single shared-memory system is shown in Fig. 5. The diagram shows a system with a range of contexts (0–); Because of this simplification, each context has access to the global STRMEM for host-initiated DMA transfers and/or retrieving the argument for void pthread_exit(void *valuator). Table III details the commands that may be used.

TABLE I BIOTHREADS REAL-TIME PERFORMANCE (DUAL-ISSUE LE1 CORES, FPGA AND ASIC)

ISSN23212152www.ijmece.com

Vol 8, Issue 4 Oct 2020

2-Issue LE1			FPGA Results (100 MHz)						ASIC Results (300 MHz)		
			No custom ISA			Profile-driven compilation /unroll/inline/custom ISA			Profile-driven compilation /unroll/inline/custom ISA		
Config	LE1 Cores	Data Memory Banks	Cycles ($x10^7$)	Real Time (Sec)	Speed Up	Cycles ($x10^7$)	Real Time (Sec)	Speed Up	Cycles ($x10^7$)	Real Time (Sec)	Speed Up
1X1	1	1	901.73	90.17	1.00	155.47	15.55	5.80	155.47	5.18	17.40
2X1	2	1	555.36	55.54	1.62	95.75	9.58	9.42	95.75	3.19	28.25
2X2		2	493.87	49.39	1.83	82.73	8.27	10.90	82.73	2.76	32.70
4X1	4	1	506.56	55.66	1.78	87.34	8.73	10.32	87.34	2.91	30.97
4X2		2	325.02	32.50	2.77	54.44	5.44	16.56	54.44	1.81	49.69
4X4		4	266.42	26.64	3.38	43.68	4.37	20.65	43.68	1.46	61.94
8X1	8	1	662.58	66.26	1.36	114.24	11.42	7.89	114.24	3.81	23.68
8X2		2	285.90	28.59	3.15	47.89	4.79	18.83	47.89	1.60	56.49
8X4		4	174.05	17.41	5.18	28.53	2.85	31.60	28.53	0.95	94.49
8X8		8	134.34	13.43	6.71	21.60	2.16	41.75	21.60	0.72	125.25

TABLE II BIOTHREADS REAL-TIME PERFORMANCE (QUAD-ISSUE LE1 CORES, FPGA AND ASIC

4-Issue LE1			FPGA Results (100 MHz)						ASIC Results (300 MHz)		
			No custom ISA			Profile-driven compilation /unroll/inline/custom ISA			Profile-driven compilation /unroll/inline/custom ISA		
Config	LE1 Cores	Data Memory Banks	Cycles ($x10^7$)	Real Time (Sec)	Speed Up	Cycles ($x10^7$)	Real Time (Sec)	Speed Up	Cycles ($x10^7$)	Real Time (Sec)	Speed Up
1X1	1	1	757.79	75.78	1.19	130.65	13.07	6.90	130.65	4.36	20.71
2X1	2	1	421.34	42.13	2.14	72.64	7.26	12.41	72.64	2.42	37.24
2X2		2	397.50	39.75	2.27	66.58	6.66	13.54	66.58	2.22	40.63
4X1	4	1	281.89	28.19	3.20	46.21	4.62	19.51	46.21	1.54	58.54
4X2		2	219.13	21.91	4.12	35.23	3.52	25.60	35.23	1.17	76.79
4X4		4	203.14	20.31	4.44	35.02	3.50	25.75	35.02	1.17	77.24
8X1	8	1	264.87	26.49	3.40	44.37	4.44	20.32	44.37	1.48	60.97
8X2		2	149.98	15.00	6.01	24.59	2.46	36.68	24.59	0.82	110.03
8X4		4	113.52	11.35	7.94	18.25	1.83	49.41	18.25	0.61	148.22
8X8		8	103.14	10.31	8.74	17.78	1.78	50.71	17.78	0.59	152.12

RESULTS AND DISCUSSION

Several experiments calculating blood volume changes in real time using the Bio Threads platform are shown here. There are two main categories for these findings: A) Performance (real-time) results, which are concerned with the actual time it takes to compute the blood perfusion map, and B) SoC platform outcomes. These specifications are for a 0.13, 1-poly, 8-metal (1P8M) standard-cell process with a Xilinx Virtex6 LX240T FG1156 [41] FPGA. It's worth noting that although the standard-cell library we have access to in the lab is very dated, the FPGA device is built on a nearly cutting-edge silicon node (40 nm, TSMC). It is clear that the performance gap between the standard-cell (300 MHz) and the FPGA objective (100 MHz) in Tables I and II is not indicative of what may be anticipated when aiming for a standard-cell process at an advanced silicon node (40 nm and below). A. Outcomes of Performance Processors on the target platform have begun running the IPPG algorithms once the 60 frames were transmitted there. When processing was complete, the calculated frame was sent back to the host for presentation. Table I displays the actual execution time for the FPGA platform's 2-wide LE1 system (a VLIW CMP with dual-static-issue cores); ASIC results were simulated. The tabular data is organized into columns with the following labels:

• Setup:

The big picture of LE1_CORES's Bio Threads implementation

EMORY_BANKS

"LE1 Cores" Finds out how many processing cores the LE1 has.

The maximum number of concurrent load/store operations permitted by the streaming memory system is denoted by the number of memory banks, as seen in Fig. 3. As will be seen shortly, this has a considerable impact on the efficiency of the system as a whole.

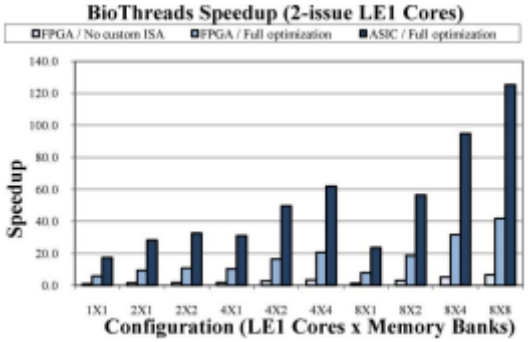

Fig. 3. Speedup of Bio Threads performance for 2-wide LE1 subsystem (FPGA and ASIC).

The other three characteristics are either determined through RTL simulations (ASIC implementation) or tested on an FPGA platform (100 MHz LE1 subsystem and service processor, as illustrated in Fig. 1). The FFT findings were achieved both with and without any special instructions to speed things up.

Temporal Patterns:

The time in milliseconds that the fundamental signal-processing algorithm takes to run.

Execution time (in seconds):

The amount of time it really took to run the algorithm (as measured by the service processor for FPGA targets or computed through RTL simulation for ASIC targets).

Accelerate:

How much faster Bio Threads configurations are than the worst-case scenario of a single-core, single-bank FPGA system without the FFT special instructions. User-directed function in-lining, compiler-driven loop unrolling, and custom instructions were found to provide the highest performance in a prior research of signal processing kernel acceleration (FFT) on the LE1 processor [42]. The foregoing solutions resulted in an 87% decrease in cycles, allowing for the real-time execution of the IPPG algorithm stages.

CONCLUSIONS

Methodology for employing a unique, adjustable VLIW CMP to speed up biomedical signal processing codes was addressed, and its viability was assessed. When it comes to designing, benchmarking, and optimizing silicon platforms for use in consumer electronics and telecommunications, the expertise of one team is not enough to meet the needs of experts in the biomedical signal processing domain. We made a conscious decision to stick with tools already in the hands of biomedical signal processing practitioners, such as MATLAB and LABVIEW, and our Bio Threads tools architecture was built to make C-level programming easier in this area. Using algorithms developed in the Me bedded MATLAB subset, we showed how the configurable, extensible Bio Threads engine can be used to compute in real-time or near-real-time the blood perfusion of living tissue. The autogenerated C code was then passed on to the toolchain, which compiled it into an application binary and performed coarse architecture space evaluation to identify the best Bio Threads configurations that achieve the required level of performance. After the FPGA-based CMP had been programmed, data sets were fed from the host system (through the LABVIEW front-end) and accelerated calculations were performed.

REFERENCES

[1] K. Rajang and L. M. Patnaik, "CBP and ART image reconstruction algorithms on media and DSP processors," Microprocess. Microsyst., vol. 25, pp. 233–238, 2001.

[2] O. Dandekar and R. Shekhar, "FPGA-Accelerated deformable image registration for improved target-delineation during CT-guided interventions," IEEE Trans. Biomed. Circuits Syst., vol. 1, no. 2, pp. 116–127, 2007.

[3] K. Wardell and G. E. Nilsson, "Duplex laser Doppler perfusion imaging," Microvasc. Res., vol. 52, pp. 171–182, 1996.

[4] S. Srinivasan, B. W. Pogue, S. D. Jiang, H. Dehghani, C. Kogel, S. Soho, J. J. Gibson, T. D. Tosteson, S. P. Poplack, and K. D. Paulsen, "Interpreting haemoglobin and water concentration, oxygen saturation, and scattering measured in vivo by near infrared breast tomography," Proc. Natl. Academy Sciences USA, vol. 100, no. 21, pp. 12349–12354, 2003.

[5] S. Hu, J. Zheng, V. A. Chouliaras, and R. Summers, "Feasibility of imaging photoplethysmography," in Proc. Conf. BioMedical Engineering and Informatics, Sanya,, China, 2008, pp. 72–75.

[6] P. Shi, V. Azorin Peris, A. Echiadis, J. Zheng, Y. Zhu, P. Y. S. Cheang, and S. Hu, "Non-contact reflection photoplethysmography towards effective human physiological monitoring," J Med. Biol. Eng., vol. 30, no. 30, pp. 161–167, 2010.

[7] V. A. Chouliaras, J. L. Nunez, D. J. Mulvaney, F. Rovati, and D. Alfonso, "A multi-standard video accelerator based on a vector architecture," IEEE Trans. Consum. Electron., vol. 51, no. 1, pp. 160–167, 2005.

[8] ARM Cortex M3 Processor Specification, Sep. 2010 [Online]. Available: http://www.arm.com/products/processors/cortex-m/cortexm3.php [9] Microblaze Processor Reference Guide, Doc. UG081 (v10.3), Oct. 2010 [Online]. Available: http://www.xilinx.com

[10] D. Mattson and M. Christensson, "Evaluation of Synthesizable CPU Cores," Master's thesis, Dept. Computer Engineering, Chalmers Univ. Technology, Goteborg, Sweden, 2004.

[11] V. A. Chouliaras and J. L. Nunez, "Scalar coprocessors for accelerating the G723.1 and G729A speech coders," IEEE Trans. Consum. Electron., vol. 49, no. 3, pp. 703–710, 2003.

[12] N. Vassiliadis, G. Theodoridis, and S. Nikolaidis, "The ARISE reconfigurable instruction set extensions framework," in Proc. Intl Conf. Emb. Computer Systems: Architectures, Modelling and Simulation, Jul. 16–19, 2007, pp. 153–160.

[13] Xilinx XPS Mailbox" V1.0a Data Sheet, Oct. 2010 [Online]. Available: http://www.xilinx.com

[14] M. F. Dossis, T. Themelis, and L. Markopoulos, "A web service to generate program coprocessors," in Proc. 4th IEEE Int. Workshop Semantic Media Adaptation and Personalization, Dec. 2009, pp. 121–128.

[15] B. Gorjiara, M. Reshadi, and D. Gajski, "Designing a custom architecture for DCT using NISC technology," in Proc. Design Automation, Asia and South Pacific Conf., 2006, pp. 24–27.

[16] V. Kathail, S. Aditya, R. Schreiber, B. Ramakrishna Rau, D. Cronquist, and M. Sivaraman, "PICO: Automatically designing custom computers," IEEE Comput., vol. 35, pp. 39–47, 2002.

[17] R. Thomson, S. Moyers, D. Mulvaney, and V. A. Chouliaras, "The UML-based design of a hardware H.264/MPEG 4 AVC video decompression core," in Proc. 5th Int. UML-SoC Workshop (in Conjunction with 45th DAC), Anaheim, CA, Jun. 2008, pp. 1–6.

[18] The AutoESL AutoPilot High-Level Synthesis Tool, May 2010 [Online]. Available: http://www.autoesl.com/docs/bdti_autopilot_final.pdf

[19] Y. Guo and J. R. Cavallaro, "A low complexity and low power SoC design architecture for adaptive MAI suppression in CDMA systems," J. VLSI Signal Process. Syst. Signal Image Video Technol., vol. 44, pp. 195–217, 2006.

[20] S. Leibson and J. Kim, "Configurable processors: A new era in chip design," IEEE Comput., vol. 38, no. 7, pp. 51–59, 2005.

[21] N. T. Clark, H. Zhong, and S. A. Mahlke, "Automated custom instruction generation for domain-specific processor acceleration," IEEE Trans. Comput., vol. 54, no. 10, pp. 1258–1270, 2005.