

HIGH-PERFORMANCE AND ENERGY-EFFICIENT HETEROGENEOUS SUBWORD PARALLEL INSTRUCTIONS

Jongmyon Kim and D. Scott Wills

Microelectronics Research Center
Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, Georgia 30332-0250
Tel: +1-404-894-6879. Fax: +1-404-894-9959
{jmkim and scott.wills}@ece.gatech.edu

ABSTRACT

High instruction throughput and energy efficiency are becoming increasingly important design requirements for embedded and mobile computing systems. This paper presents the Quantized Color Pack eXtension (QCPX) ISA to improve execution performance of multimedia processing applications on programmable superscalar processors while reducing the energy consumption for these applications. QCPX exploits parallelism within the color space representation (YCbCr: luminance-chrominance) in addition to generic subword parallelism exploited by existing multimedia instruction set extensions (e.g., MMX, SSE, MDMX). We evaluate the performance (execution time in cycles) and energy consumption using QCPX on a media benchmark suite that includes vector median filter, scalar median filter, edge detection, and vector quantization. Our experiment results indicate that a 32-bit QCPX version achieves speedups ranging from 205% to 562% compared with that of a 32-bit baseline RISC version and 90% to 100% over the 32-bit MDMX-like version on identically configured, dynamically scheduled ILP superscalar processors. In addition, the QCPX version reduces the energy consumption from 69% to 83% over the baseline version and 47% to 50% over the MDMX-like version due to the significant reduction of executed instructions and cache accesses.

1. INTRODUCTION

Since mobile color image and video products operate using the limited energy available in a battery [1], the energy-efficiency of high performance embedded processors is a critical design consideration. However, these multimedia applications have large input data sets that require many memory accesses resulting in higher energy consumption [2]. This demand in data parallelism-

rich media applications has led to the development of multimedia extension instruction sets (or subword-parallel instructions) for general-purpose superscalar processors. Examples are Intel's MMXTM [3], SSETM and SSE-2 [4], Hewlett Packard's MAX2 for the PA-RISC architecture [5], Sun Microsystems' VIS for SPARC [6], MIPS's MDMX [7], Alpha's MVI [8], and Motorola's ALTIVEC for PowerPCTM architecture [9]. The main idea of these extensions is to combine data parallel execution and low-precision data formats in the exploitation of homogeneous (i.e., uniform) subword parallelism within the context of a dynamically-scheduled superscalar instruction level parallel (ILP) machine. In subword parallelism [10], the processor's datapath is partitioned into multiple low-precision segments (e.g., 8-bit pixels) while operating in parallel on these subwords. These extended instructions are specialized and are generally used only for multimedia applications.

In this paper, we present a novel multimedia extension for ILP processors, Quantized Color Pack eXtension (QCPX) that achieves homogeneous subword data parallelism and heterogeneous (non-uniform) within the color-space representation (e.g., YCbCr) while also reducing the energy consumption for multimedia processing applications. Since a luminance-chrominance space [11] allows coding schemes to exploit the properties of human vision by allocating significantly less bits to the high-frequency chrominance components that are perceptually less significant, the chrominance space can be implemented using shorter fields (e.g., Cb and Cr fields are 4 bits rather than 8) while providing satisfactory image quality. Therefore, greater subword parallelism is achieved by packing more of these smaller pixel values into a word, allowing more pixel data to be processed in parallel. By including a specific color data representation supported in hardware, the performance can be improved to support color operations.

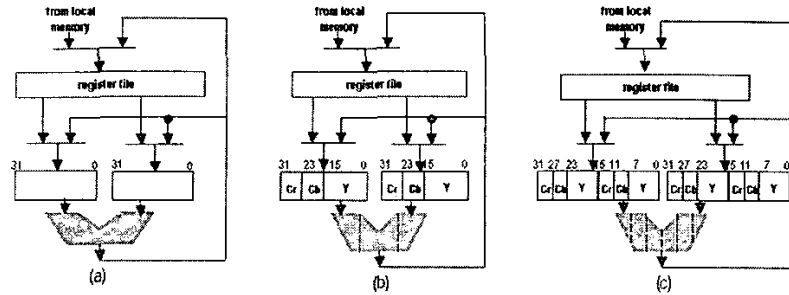


Figure 1. Type of operations. (a) Baseline 32-bit operation. (b) The 8-8-16-bit QCPX operation. (c) Two sets of 4-4-8-bit QCPX operation.

To evaluate the impact of QCPX, we coded and profiled four color image processing applications using SimpleScalar tools [12] extended with the proposed new QCPX instructions and the power estimator Wattch [13], analyzing execution performance, cache behaviors, workload characterizations, and its corresponding energy consumptions. Experiment results on these media applications show that QCPX employing six-way subword parallelism (six pixels in a 32-bit width) achieves speedups ranging from 205% to 562% and reduces energy consumption ranging from 69% to 83% compared with that of the baseline ISA due to the significant reduction of branch frequencies, cache accesses, and total number of executed instructions. In addition, QCPX achieves an overall average speedup about 97% and reduces overall average energy consumption of 47% over the MDMX-like multimedia extension by processing six rather than three pixels in a 32-bit word.

This paper is organized as follows. Section 2 describes the QCPX instruction set along with pictorial examples. Section 3 describes the benchmark applications and experimental methodology for this study. Section 4 evaluates the performance and energy consumption of the QCPX version for the benchmark applications and analyzes the results. Finally, conclusions are offered in section 5.

2. QCPX INSTRUCTIONS

The Quantized Color Pack eXtension (QCPX) instruction set is targeted to accelerate color image and video dominated processing. QCPX instructions operate on two types of the data format such as a packed 16-8-8-bit for high-precision YCbCr data results and two sets of the 8-4-4-bit YCbCr data in a 32-bit word width. Figure 1 shows a baseline 32-bit operation and two types of the QCPX operations. Typical multimedia instruction set extensions such as MMX and MDMX store true color pixels as a packed 32-bit word containing an 8-bit red, blue, green, and alpha channel, and exploit the subword parallelism on the RGB components while wasting operations on the

alpha values [14]. QCPX eliminates RGB limitations storing packed 16-bit YCbCr value in a 16- or 32-bit word width. It improves performance through the use of a specific color data representation supported in hardware. A 32-bit QCPX integer function unit includes two 16-bit color values (8-bits for Y and 4-bits each for Cr and Cb) to be performed in a single cycle with a small increase in functional unit complexity. QCPX instructions are classified in four groups: (1) parallel ALU (Arithmetic Logical Unit) instructions, (2) expand and pack instructions, (3) compare instructions, and (4) special-purpose instructions. A summary and examples of QCPX are presented below.

2.1. Parallel Color-word Arithmetic and Logical Instructions

Parallel color-word arithmetic and logical instructions include `ADD_CRCBY` (signed, unsigned saturation, and modulo), `SUBTRACT_CRCBY` (signed, unsigned saturation, and modulo), `MULTIPLY_CRCBY`, `BCAST_CRCBY` (broadcast), `AVERAGE_CRCBY`, `DIVIDE_CRCBY`, and `SHIFT_CRCBY` [`left|right`] operations, which are the most frequent operations in many color image and video computations. For example, the `AVERAGE_CRCBY` instruction adds two source operands included YCbCr value then performs a divide by two. Figure 2-(a) and (b) illustrates the color-parallel average and broadcast instructions, respectively.

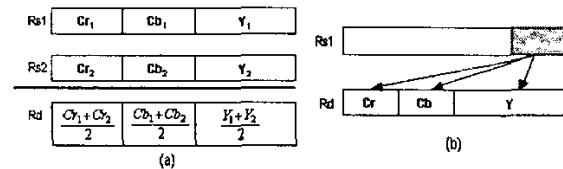


Figure 2. (a) Parallel average instruction. (b) Broadcast instruction.

2.2. Expend and Pack Instructions

The EXPEND_CRCBY instruction expands a packed 16-bit YCbCr data in a 32-bit register (2x 8-bit for Cb and Cr and a 16-bit for Y) for parallel add and subtract instructions needed more high precision results. Similarly, the PACK_CRCBY instruction converts from the larger fixed data back to pixel data, clipping and truncating the 32-bit YCbCr value to a partitioned 16-bit register for YCbCr output. Figure 3 shows the EXPEND_CRCBY and PACK_CRCBY instructions.

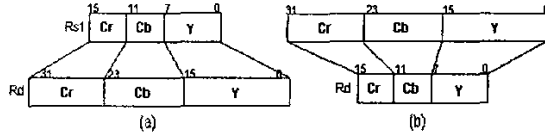


Figure 3. (a) Expand instruction. (b) Pack instruction.

2.3. Parallel Color-word Compare Instructions

Parallel color-word compare instructions allow a 32-bit superscalar processor to compare two sets of the 8-4-4-bit YCbCr values. This is an important operation for the image enhancement (e.g., scalar median filter), edge detection, and 3-D rendering to determine which objects are in front of others. The compare instructions include CMPEQZ_CRCBY, CMPGTZ_CRCBY, CMPLTZ_CRCBY, MIN_CRCBY, and MAX_CRCBY. The first three instructions perform on the packed YCbCr data in parallel in a single instruction to produce predicate masks that can be used to reduce branches. In some cases, there is a need to select different data based on a condition query performed on the incoming data [3]. The MIN_CRCBY instruction returns the smaller of their two operands included the packed YCbCr data, while the MAX_CRCBY instruction returns larger. Figure 4 shows the MIN_CRCBY and MAX_CRCBY instruction.

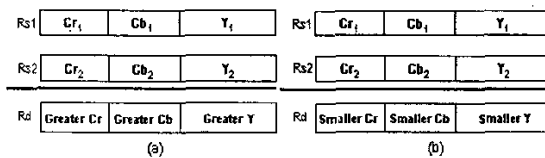


Figure 4. (a) Parallel minimum instruction. (b) Parallel maximum instruction.

2.4. Special-purpose Instructions

Special-purpose QCPX instructions include MACC_CRCBY (multiply-accumulate), ADACC_CRCBY (absolute-distance-accumulate), REDUC_CRCBY (reduction), and ZACC (zero-accumulate) instructions

that provide the most computational benefit of all QCPX instructions. Figure 5 shows an absolute-distance-accumulate instruction. Each of the ADACC_CRCBY instruction calls saves its sum in a partitioned 64-bit special-purpose register until the ZACC instruction is called. A 64-bit register (HI and LO register in the SimpleScalar processor) is used to avoid the result of width problems to act as the destination register. This mechanism supports the extra space required for the result to be stored in full precision. These special-purpose instructions are commonly used in color image compression and motion estimation algorithms such as vector quantization, JPEG, H.261, and MPEG-1 (or -2).

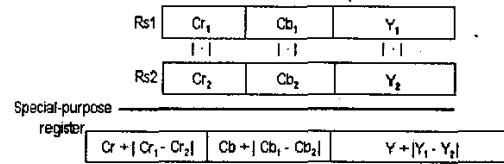


Figure 5. Absolute-distance-accumulate instruction.

3. EXPERIMENTAL SETUP

3.1. Benchmarks

To cover a range of color image and video processing workloads, a media benchmark suit was selected. It includes four applications (vector median filter: *vmf*, scalar median filter: *smf*, edge detection: *edge*, and vector quantization: *VQ*). The applications are briefly described in Table 1. These kernels are representative of computation intensive cores application benchmark suits such as MediaBench [15]. All the color image benchmarks were run with Quad-CIF resolution (176x144) pixel 3-band (i.e., channel) input images.

Table 1. Descriptions of the benchmarks used in this study.

Benchmarks	Description
<i>smf</i>	Scalar median filter which eliminates impulse noise spikes from an image by replacing 3-band channels (e.g., Y, Cb, and Cr) with the median channels of a small neighborhood surrounding the pixels
<i>vmf</i>	Vector median filter to remove impulse noise of 176 x 144 color image by ordering vectors according to their relative magnitude differences in a 3x3 window
<i>edge</i>	Edge detection to create a series of 2-D spatial gradient magnitudes of 176 x 144 color image
VQ	Vector quantization to compress data of 176 x 144 resolution color image

3.2. Processor Configuration and Tools

A modified version of the SimpleScalar simulator [12] was used to profile instruction execution for the QCPX ISA, baseline ISA, and MDMX-like ISA. Wattch [13], an architectural-level simulator, was used to estimate the energy consumption in each case. Figure 6 shows a methodology framework used in this study to evaluate the performance and energy consumption using QCPX on a dynamically-scheduled superscalar ILP processor architecture.

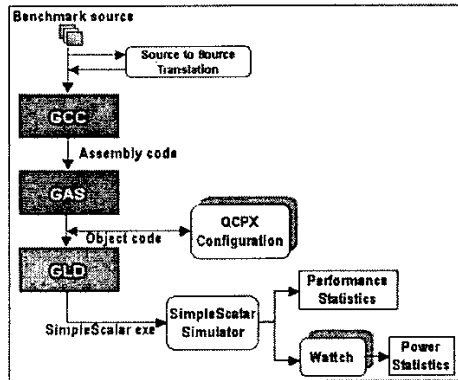


Figure 6. Methodology framework.

The QCPX and MDMX-like version of the benchmark applications were created by replacing fragments of the baseline assembly language with ones containing QCPX or MDMX-like instructions. The QCPX, baseline, and MDMX-like versions of the applications have the same parameters, data set, and calling sequence. Since the target platform is an embedded system, operating system interface code (e.g., file system access) is not included in this study. Additional optimizations typical of embedded compilers were applied to baseline version before the QCPX and MDMX-like versions were created. All benchmark applications are executed on the same technology and processor configuration, which is summarized in Table 2.

Table 2. Processor parameters and memory configuration.

Parameter	Value
Technology	0.35 μ m
MHz	200
Fetch width	4 instructions/cycle
Decode width	4 instructions/cycle
Issue width	4 instructions/cycle
Commit width	4 instructions/cycle
RUU (window) size	16 instructions
LSQ (Load Store Queue)	8 instructions
Fus	ALU:4, MULT:1, FP-ALU:4, FP-MULT:1

L1 D-cache	128-set, 4-way, 32-byte line, LRU, 1-cycle hit, total of 16KB
L1 I-cache	512-set, direct-mapped 32-byte line, LRU, 1-cycle hit, total of 16KB
L2 unified cache	1024-set, 4-way, 64-byte line, LRU, 6-cycle hit, total of 256KB
Memory latency (memory width)	18 cycles for first chunk, 2 thereafter (64 bits)
Instruction TLB	16-way, 4096 byte page, 4-way, LRU, 30 cycle miss penalty
Data TLB	32-way, 4096 byte page, 4-way, LRU, 30 cycle miss penalty
QCPX FUs	ALU: 4, MULT: 1

4. ANALYSIS OF RESULTS

In the experiment, the three different versions of the benchmark programs, (1) the baseline 32-bit RISC ISA without subword parallelism, (2) the baseline plus the 32-bit MDMX-like ISA [7], and (3) the baseline plus the 32-bit QCPX ISA were executed on the modified SimpleScalar simulator. Their profile statistics are then analyzed using the Wattch power simulator to determine each benchmark's energy consumption for an equivalent technology and processor configuration. The execution cycle count and corresponding energy consumption of application version forms the basis of study comparison. Table 3 summarizes each application version's performance and energy consumption.

4.1. Performance of the Applications

Figure 7 illustrates the execution performance (speedup) of QCPX plus baseline ISA over the baseline version and MDPX-like plus baseline version. Significant speedups using QCPX are achieved over the baseline version, ranging from 205% for scalar median filter (*smf*) to 562% for edge detection (*edge*).

4.1.1. Edge detection

The *edge* application shows the most execution improvement due to the presence of the pipelined multiply-accumulate instruction in QCPX of computing sum of multiply between Y, Cb, and Cr data and coefficient values in parallel. Using the `BCAST_CRCBY` instruction, each of coefficient values saved in memory is distributed in Cr, Cb, and Y positions, and then the `MACC_CRCBY` instruction accumulates its sum in a partitioned 64-bit register (32-bit HI and LO in SimpleScalar) rather than saving to memory, reducing the large number of executed instructions as well as cache accesses. The *edge* application using QCPX is 562% and 90% faster than the baseline version and MDMX-like version, respectively.

4.1.2. Scalar median filter

For the *smf* application, using the MAX_CRCBY and MIN_CRCBY instruction, nine registers containing two sets of packed YCbCr data are compared to sort YCbCr data in parallel, reducing the large number of condition instructions. Therefore, a bubble-sort algorithm is highly accelerated. The QCPX version exploits an overall speedup of about 205% over the baseline version and 98% over the MDMX version.

4.1.3. Vector median filter

For the *vmf* application [16], the most time critical operation is an absolute-distance accumulation that computes sum of absolute magnitude differences for each Y, Cb, and Cr values in parallel. Using the ADACC_CRCBY instruction, nine pairs including two sets of YCbCr data are compared to one another via the sum of absolute magnitude differences in parallel, reducing a large amount of logic overhead instructions. The QCPX version achieves an overall speedup of 402% as compared with that of the baseline version and 99% over the MDMX-like version.

4.1.4. Vector quantization

The *VQ* application has become a popular image and video compression technique [17]. Compression is achieved by subdividing the image into small blocks (e.g., 4x4 pixels), then finding the best match for each block among the available codewords. The *VQ* application using QCPX shows an overall speedup of 503% over the baseline version and 100% over the MDMX-like version due to the significant reduction of ALU and branch instructions by using the ADACC_CRCBY instruction computing the absolute sum of differences between two 4x4 blocks between the YCbCr data and codewords in parallel.

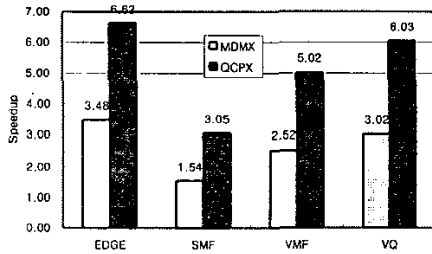


Figure 7. Speedups over the baseline ISA.

4.2. Effect of QCPX on Energy Consumption

Shorter execution times result in lower energy consumption when execution platforms employ identical clock rates, implementation technology, and processor parameters [18]. QCPX significantly reduces energy

consumption on color image processing applications. Experimental results on the four applications used in this study show that the 32-bit QCPX version reduces energy consumption from 69% (*smf*) to 83% (*edge*) over the 32-bit baseline ISA version and 47% to 50% over the 32-bit MDMX-like version due to the significant reduction of executed instructions and cache accesses. Figure 8 illustrates the relative total energy consumptions of the QCPX and MDMX-like version over the baseline version. As expected, the *edge* application using QCPX shows the highest reduction rate of the energy consumption over the baseline version due to the largest reduction rate of total executed instructions and cache accesses, while the *smf* application using QCPX achieves the least reduction rate of the energy consumption. Since *VQ* has the longest execution cycle count, it consumes the greatest energy of the applications.

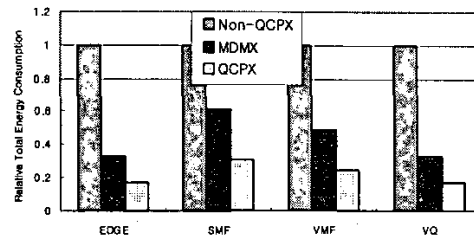


Figure 8. Relative total energy consumptions over the baseline version.

Since branches and caches accesses are reduced using QCPX through the processing of YCbCr values in parallel, less energy is spent on speculative execution and caches accesses. Figure 9 illustrates the relative energy consumptions (branches and caches) of the QCPX and MDMX-like version over the baseline version.

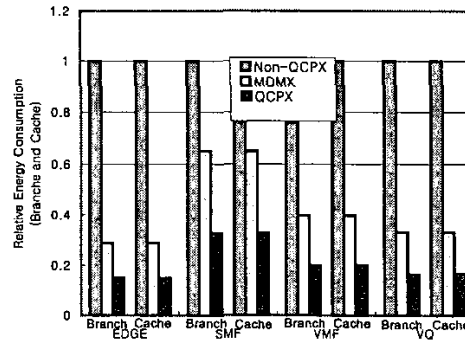


Figure 9. Relative energy consumptions for branch prediction and cache accesses over the baseline version.

Table 3. Applications performance and energy consumption with and without QCPX and MDMX-like ISA.

Applications	ISA Type	Energy (nJ)	Cycles	# of executed instructions
edge	Baseline	2,363,881,535	26,792,919	49,610,086
	MDMX-like	769,115,583	7,694,965	18,454,284
	QCPX	403,962,418	4,048,031	9,410,699
smf	Baseline	3,402,506,799	33,135,080	74,510,711
	MDMX-like	2,100,066,242	21,530,736	47,950,961
	QCPX	1,050,184,585	10,872,693	23,978,441
vmf	Baseline	10,251,416,928	116,221,809	215,405,293
	MDMX-like	4,963,526,013	46,186,466	105,317,925
	QCPX	2,493,677,436	23,153,363	53,011,538
VQ	Baseline	49,691,891,408	534,835,781	1,046,836,441
	MDMX-like	16,457,019,225	177,191,374	378,512,060
	QCPX	8,227,934,578	88,624,598	189,268,792

5. CONCLUSIONS

In this paper, we have examined the impact of the QCPX instruction set on execution performance and energy consumptions for media processing applications with the programmable superscalar processor simulator. Simulation results indicate that the QCPX version achieves higher code density and faster execution cycle time compared with that of the 32-bit baseline version and the 32-bit MDMX-like version due to processing of two packed YCbCr data in parallel. In addition, QCPX improves energy consumptions significantly over the baseline version and MDMX-like version in each benchmark application due to the significant reduction of executed instructions and cache accesses. Ongoing research will address the QCPX effectiveness on other color image and video applications with multi-processor architectures, meeting the requirements of demanding multimedia applications by providing concurrent data-stream processing.

REFERENCES

- [1] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: A first step towards software power minimization," *IEEE Trans. VLSI Systems*, vol. 2, no. 4, pp. 437-445, 1994.
- [2] K. Diefendorff and R. Dubey, "How multimedia workloads will change processor design," *IEEE Computer*, vol. 30, no. 9, pp. 43-45, 1997.
- [3] A. Peleg and U. Weiser, "MMX technology extension to the Intel architecture," *IEEE Micro*, vol. 16, no. 4, pp. 42-50, 1996.
- [4] S. K. Raman, V. Pentkovski, and J. Keshava, "Implementing streaming SIMD extensions on the Pentium III processor," *IEEE Micro*, vol. 20, no. 4, pp. 28-39, 2000.
- [5] R. B. Lee, "Subword parallelism with MAX-2," *IEEE Micro*, vol. 16, no. 4, pp. 51-59, 1996.
- [6] M. Tremly, J. M. O'Connor, V. Narayanan, and L. He, "VIS speeds new media processing," *IEEE Micro*, vol. 16, no. 4, pp. 10-20, 1996.
- [7] MIPS extension for digital media with 3D. Technical Report <http://www.mips.com>, MIPS technologies, Inc., 1997.
- [8] R. Sites, Ed. Alpha Reference Manual, Burlington, MA: Digital, 1992.
- [9] H. Nguyen and L. John, "Exploiting SIMD parallelism in DSP and multimedia algorithms using the AltiVec technology," *In ICS'99*, pp. 11-20, 1999.
- [10] R. Lee, "Accelerating multimedia with enhanced microprocessors," *IEEE Micro*, vol. 15, no. 2, pp. 22-32, 1995.
- [11] G. Sharma, M. J. Vrhel, and H. J. Trussell, "Color imaging for multimedia," *Proc. of the IEEE*, vol. 86, no. 6, pp. 1088-1108, 1998.
- [12] T. Austin and D. Burger, "The SimpleScalar Tool Set, Version 2.0," TR-1342, Computer Sciences department, University of Wisconsin, Madison.
- [13] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," *Intl. Symposium on Computer Architecture*, pp. 83-94, 2000.
- [14] A. Peleg, S. Wilkie, and U. Weiser, "Intel MMX for multimedia PCs," *Communications of the ACM*, vol. 40, no. 1, pp. 25-38, 2000.
- [15] C. Lee, M. Potkonjak, and W. H. Smith, "MediaBench: A tool for evaluating and synthesizing multimedia and communications systems," *Proc. IEEE/ACM Intl. Sym. on Microarchitecture*, pp. 330-335, 1997.
- [16] J. M. Kim, et al, "Impulse noise removal on an embedded, low memory SIMD processor," *Proc. of the 14th IEEE Intl. Conf. on DSP*, pp. 1257-1260, 2002.
- [17] K. S. Wu and J. C. Lin, "Fast VQ encoding by an efficient kick-out condition," *IEEE Trans. Circuit and Systems for Video Technology*, vol. 10, no. 1, pp. 59-62, 2000.
- [18] V. Tiwari, S. Malik, and A. Wolfe, "Compilation techniques for low energy: An overview," *Proc. Symp. Low Power Electron.*, pp. 38-39, 1994.