IEEE Transactions on Computers, Vol. 64, No. 3, March 2015, pp. 805-817

Performance-Energy Optimizations for Shared Vector Accelerators in Multicores

Spiridon F. Beldianu and Sotirios G. Ziavras
Department of Electrical and Computer Engineering
New Jersey Institute of Technology
Newark, NJ 07102, USA

Abstract

For multicore processors with a private vector coprocessor (VP) per core, VP resources may not be highly utilized due to limited data-level parallelism (DLP) in applications. Also, under low VP utilization static power dominates the total energy consumption. We enhance here our previously proposed VP sharing framework for multicores in order to increase VP utilization while reducing the static energy. We describe two power-gating (PG) techniques to dynamically control the VP's width based on utilization figures. Floating-point results on an FPGA prototype show that the PG techniques reduce the energy needs by 30-35% with negligible performance reduction as compared to a multicore with the same amount of hardware resources where, however, each core is attached to a private VP.

Index Terms: Accelerator, Vector Processor, Multicore Processing, Performance, Energy.

I. Introduction

Hardware accelerators achieve high performance with low energy cost due to customization. VPs, SIMD (Single-Instruction, Multiple-Data) units and GPUs (Graphics Processing Units) are in the forefront due to the pervasiveness of DLP-dominant applications [3, 8-10, 12, 14, 15, 22, 30]. However, VPs are not always efficient for various reasons. (i) The effective DLP of programs is often reduced due to intermittent control. (ii) The vector length may vary across applications or within an application [2]. (iii) Applications may have unknown data dependencies within instruction sequences. This is serious for dynamic environments with limited capabilities and strict performance and/or energy requirements. (iv) As the ratio of

arithmetic operations to memory references decreases, ALU utilization is reduced. (v) Finally, parallelism efficiency may decrease with increased resource counts [16].

Although GPUs originally contained dedicated graphics pipelines, modern general-purpose GPUs [16] have replaced them with numerous execution units and rely on optimized routines for high performance. VPs and GPUs differ in many aspects, such as: (i) GPUs have many more parallel execution units than VPs (thousands compared to dozens), therefore GPUs are less area and power efficient with limited DLP. (ii) GPU's hierarchical structure is more complex. (iii) VPs hide memory latencies using deeply pipelined block transfers whereas GPUs rely on complex multithreading. VPs also may optionally use multithreading. (iv) Since GPUs lack a control processor, hardware is needed to identify address proximity in addresses. (v) GPUs employ branch synchronization markers and stacks to handle masks whereas VPs rely on the compiler. (vi) Contrary to VPs that run all scalar code on the host, each SIMD unit in a GPU supports scalar operations since the host and the GPU use different address spaces and data transfers take thousands of clock cycles. (vii) Finally, GPU programming requires specialized languages such as CUDA or OpenCL [16]; in contrast, VP programming involves the offloading of rather simple macros or directives. These differences increase the area overheads and the power consumption of GPUs. Also, the computer architecture field relies on heterogeneity (without any single solution being the panacea), so it applies to GPUs as well for DLP. Our work has value for systems implemented on FPGAs or ASICs. FPGAs often achieve better performance and lower energy consumption than GPUs. For sliding-window applications involving vector operations, FPGAs achieve 11x and 57x speedups compared to GPUs and multicores, respectively, while consuming much less energy" [29].

VP architectures for multicores must adhere to three design requirements: (i) *High resource utilization of vector pipelines* achieved via software-based DLP and hardware-based simultaneous VP sharing by aggregating many vector instruction streams. We assume VPs composed of vector lanes [7]. A lane contains a subset of the entire VP's vector register file, an ALU and a memory load/store unit. (ii) *Reduced impact of static power* on the energy budget via VP sharing. Static power increasingly becomes critical due to reduced feature sizes and increased transistor counts [5]. Performance does not often follow an upward trend with finer resources, primarily because of decreases in average transistor utilizations Further transistor scaling may create a utility economics wall that will force chip areas to be frequently powered down [6]. Finally, (iii) VP sharing that can facilitate efficient *runtime resource and power management*. In contrast, a per-core VP leaves much less room for runtime management. To satisfy the three design requirements while releasing resources for other use, we investigate two VP architectural contexts [10, 14]. We propose here an energy estimation model, two PG techniques and energy management frameworks for adjusting the number of active lanes based on the utilization.

II. Related Work

VIRAM emphasized vector lanes [7]. SODA [8] and AnySP [9] contain SIMD integer pipelines for software defined radio. Soft VPs for FPGAs appeared in [12, 15]. All these VPs are designed to interface a single scalar core and use fixed resources. Their rigidity becomes a weakness in dynamic environments with desired energy/performance levels. Lanes may be time-shared by short-vector or scalar threads [19]. In contrast, we target at simultaneous multithreading for maximizing the utilization of lane units.

Starting at 32nm, subthreshold leakage is high even for high-k metal gate technology; for the 28 nm TSMC High Performance (HP) process, suitable for GPUs and CPUs, leakage reaches up to 40% of the dissipated power [27-28]. To reduce static power, modules may be turned off by gating the ground or the supply voltage. Dynamic Voltage and Frequency Scaling (DVFS) reduces the clock frequency and/or the voltage but is less beneficial for leakage dominant components (e.g., SRAMs or register files) [23]. Multi-threshold CMOS circuits reduce static power in low voltage and power, and high-performance applications [1, 11]. DVFS or multi-thresholding can be complementary to our work. We do not intend to combine existing power reduction techniques; such an approach can be time consuming and will deviate from the main objective of showing that VP sharing can be enhanced via intelligent power management.

A sleep instruction used with PG can turn off a unit [24]. PG overheads increase for finer granularity. PG techniques turn on/off chosen cores as the utilization varies in datacenters [21]. The core population that minimizes the energy is derived theoretically in [13, 17]. Instruction-level energy estimation finds statically the number of active streaming multiprocessors (SMs) that minimize the dynamic energy for CUDA workloads on NVIDIA GPUs [22]. This approach is not quite effective as static power is ignored and the optimization is coarse since it relies on active SMs (instead of active CUDA cores in an SM). Using static-time application profiling, GPU leakage power is reduced by shutting down the shader, geometry or other executions units [30]. L1 and L2 cache leakage is reduced by applying PG to SRAMs [31]. Our PG-driven framework could help us develop techniques for minimizing GPU energy at fine levels. Memory-bound applications will benefit mostly due to underutilized cores in SMs. Intel Advanced Vector eXtensions (AVX) were introduced in 2011 for Sandy Bridge processors. The

AltiVec SIMD instruction set is found in PowerPCs. However, vector instruction extensions generally consume large core area while showing low resource utilization on the average.

A. Intel Phi Coprocessor and NVIDIA Kepler GPU

In contrast to our proposed coprocessor design approach, the systems discussed in this section are not efficient for applications with varying DLP. They target exclusively at applications with sustained high DLP. However, they are two of the most advanced affordable commercial platforms for data-intensive computing. Similar to our motivation for this work, the designers of the Intel Xeon Phi coprocessor recognized recently the importance of resource centralization for vector processing in high-performance computing [3]. Phi is implemented on a large 4.6inx5.9in printed circuit board, is IP-network addressable and contains 61 CPU cores in a ring. Phi differs substantially from our approach since each Phi core is attached to a private vector processing unit (VPU). A core runs in-order up to four threads; a thread uses up to 32 registers in the 512-bit wide VPU. Phi is easier to program than GPUs by adding simple directives to compiled code. Its efficiency is prohibitively low with reduced DLP, so Intel recommends about 240 threads running in parallel. Phi supports PG for cores and entire VPUs. Not only is ours very different on-chip vector architecture due to vector-lane sharing across cores, but we also propose in this paper lane-based dynamic power management within vector units.

NVIDIA's most advanced Kepler chip has enormous complexity and 7.1 billion transistors [26]. It contains up to 15 streaming multiprocessors (SMXs) and has been manufactured with TSMC's 28nm process. An SMX contains 192 single-precision CUDA cores, 64 double-precision units, 32 special-function units, and 32 load/store units. CPU cores can launch simultaneously work for a single GPU and SMXs may create work dynamically. Thus, a subset

of SMXs, or units in them, may be sometimes idle due to work imbalance. However, Kepler was optimized based on performance per watt when all units are active [26], therefore it does not support runtime power management despite the large number of resources in SMXs.

Our work differs from these platforms as follows: (a) It facilitates on-chip vector-unit sharing among cores instead of assigning exclusive accelerators to cores. (b) Our shared accelerator deals gracefully with one or many simultaneously running applications that display varying DLP. (c) And, our accelerator implements fine-grain, low-overhead energy management using information extracted statically and dynamically. Our work here can be adapted to develop a PG runtime framework for Kepler- and Phi-like systems. For Phi, the VPU design should be lane-based to support the PG of individual lanes. Also, a mechanism must be devised for individual VPU lanes to be shared across cores. Each SMX in Kepler schedules threads in groups of 32, which are called warps. With up to 64 warps per SMX, 2048 threads may be present in an SMX. The objective will become to minimize the energy consumption of individual SMXs by adjusting dynamically their number of active CUDA cores based on the number of threads/warps.

III. Architectures for VP Sharing in Multicores

Intelligent VP sharing on multicores increases VP efficiency and application throughput compared to multicores with per-core private VPs [14]. Let us summarize two of our VP sharing contexts [10, 14]. *Coarse-grain Temporal Sharing* (CTS) multiplexes temporally sequences of vector instructions arriving from the cores. A core gets exclusive VP control and releases it by executing lock and unlock instructions. The VP runs a vector thread to completion, or until it stalls, before switching to another thread. CTS increases lane utilization primarily for interleaved long sequences of scalar and vector code. The utilization of VP lanes increases but not

necessarily of their functional units since a single thread may not fully utilize lane resources. *Fine-grain Temporal Sharing* (FTS) multiplexes spatially in each lane instructions from different vector threads in order to increase the utilization of all vector units. It resembles simultaneous multithreading for scalar codes. FTS achieves better performance and energy savings [10, 14].

Fig. 1 shows a representative dual-core for VP sharing. Each lane contains a subset of elements from the vector register file (VRF), an FPU and a memory load/store (LDST) unit. A vector controller (VC) receives instructions of two types from its attached core: instructions to move and process vector data (forwarded to lanes) and control instructions (forwarded to the Scheduler). Control instructions facilitate communications between cores and the Scheduler for acquiring VP resources, getting the VP status and changing the vector length. The VC has a two-stage pipeline for instruction decoding, and hazard detection and vector register renaming, respectively. The VC broadcasts a vector instruction to lanes by pushing it along with vector element ranges into small instruction FIFOs.

Each lane in our proof-of-concept implementation has a vector flag register file (VFRF), and separate ALU and LDST instruction FIFOs for each VC. Logic in ALU and LDST arbitrates instruction execution. LDST interfaces a memory crossbar (MC). LDST instructions use non-unit or indexed vector stride. Shuffle instructions interchange elements between lanes using patterns stored in vector registers. LDST instructions are executed and committed in order. ALU instructions may commit out of order. The ALU contains a write back (WB) arbiter to resolve simultaneous stores. Vector register elements are distributed across lanes using low-order interleaving; the number of elements in a lane is configurable. Scratchpad memory is chosen to drastically reduce the energy and area requirements. Scratchpad memory can yield, with code optimization, much better performance than cache for regular memory accesses [25]. Also,

cacheless vector processors often outperform cache-based systems [20]. Cores run (un)lock routines for exclusive DMA access. For a VP with M lanes, K vector elements per lane and a *vector length VL* for an application, up to $K*M/VL$ vector registers are available. The Scheduler makes lane assignment and configuration decisions.

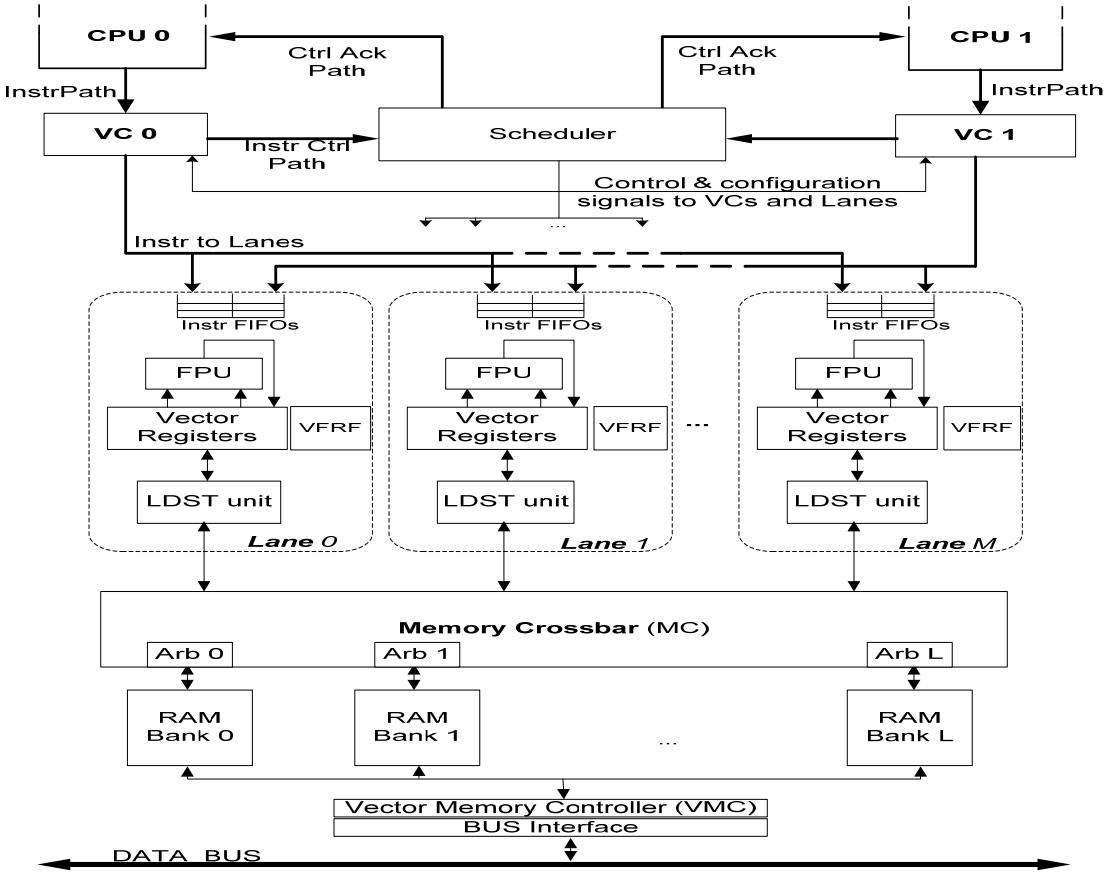


Fig. 1. A representative architecture containing two scalar cores and M vector lanes in a shared VP.

We prototyped in VHDL a dual-core on a Xilinx Virtex-6 XC6VLX130T FPGA. The VP has M= 2, 4, 8, 16 or 32 lanes and an M-bank memory. The soft core is MicroBlaze, a 32-bit RISC by Xilinx that employs the Harvard architecture and uses the 32-bit Fast Simplex Link (FSL) bus for coprocessors. The VP is scalable with the lane population, except for the MUXF8 FPGA primitive that relates to the crossbar. Without loss of generality, the crossbar could be replaced with an alternative interconnect for larger multicores. The LUT counts for a VP lane and the core

are 1100 and 750, respectively. The flip-flop counts are 3642 and 1050, respectively. For TSMC 40nm HP ASIC implementation, the lane and core gate counts are 453,000 and 212,000, respectively. These resource numbers boost further our claim that coprocessor sharing is critical.

IV. Benchmarking

Benchmarking involves handwritten assembly-language code for the VP in the form of macros embedded in C code compiled with Xilinx MicroBlaze gcc. Five applications were developed: 32-tap finite impulse response filter (**FIR32)**, 32-point decimation-in-time radix-2 butterfly FFT (**FFT**), 1024x1024 dense matrix multiplication (**MM**), **LU** decomposition, and Sparse Matrix Vector Multiplication (**SpMVM**). They are summarized in Table 1. They have similarities with GPU-oriented benchmarks [16] (e.g., matrix-vector products and LU Laplace solver). The cores in our evaluation run independent threads. Several scenarios were created for each benchmark involving loop unrolling, various *VL*s and instruction rearrangement optimizations.

The FIR32 combinations are: (i) CTS or FTS; (ii) *VL*= 32, 64, 128 or 256; (iii) no loop unrolling, or unrolling once or three times; and (iv) instruction rearrangement. FFT assumes a five-stage butterfly for complex multiply, add and shuffle operations. Eight scenarios involve: (i) CTS or FTS; (ii) *VL*= 32 or 64; (iii) no loop unrolling or unrolling once; and (iv) instruction rearrangement. MM uses SAXPY for 14 scenarios of: (i) CTS or FTS; (ii) *VL*= 32, 64, 128 or 256; (iii) no loop unrolling or unrolling once; and (iv) instruction rearrangement.

LU generates the L(ower) and U(pper) triangular matrices for a dense 128×128 matrix using the Doolittle algorithm for three scenarios. *VL* is decreased successively in Gaussian elimination. SpMVM uses a matrix in compressed row storage format and has two stages. In stage *SpMVM-k1*, non-zeros of a matrix row are multiplied with respective vector elements. In stage *SpMVM-*

k2, products are added along each matrix row; to balance the workload across lanes, matrix rows are ordered according to their number of non-zeros.

Table 1. Benchmarks.

Benchmark	Implementation	Vector Lengths	Loop Unrolled
FIR	Outer product	32, 64, 128, 256	0, 1, 3 times
FFT	Radix-2 butterfly	32, 64	0, 1 time
MMU	Scalar-vector product	32, 64, 128, 256	0, 1 time
LU	Doolittle algorithm	<128	0, 1 time
SpMVM	Nonzero multiply-k1 + addition-k2	32, 64	0, 1 time

V. Performance and Power Models

A. Performance Model

The average ALU utilization of a lane is defined as the average number of results produced in 100 clock cycles. Similarly, the average LDST utilization is the average number of 32-bit words sent and received via the crossbar in 100 clock cycles. The ALU or LDST utilization $U_{ALU/LDST}$ is the product of the average instruction throughput $IT_{ALU/LDST}$ (i.e., instructions issued in 100 clock cycles) and the number of elements from a vector register located in this lane (i.e., *VL*/*M*):

$$U_{ALU/LDST} = IT_{ALU/LDST} * VL / M \qquad (1)$$

The execution time (clock cycles) of a given kernel is inversely proportional to the product of the ALU utilization per lane and the number of active lanes; K_{kernel}, determined by experimentation, is a constant dependent on the kernel workload (i.e., number of FPU operations) divided by 100.

$$t_{exec} = \frac{K_{kernel}}{M * U_{ALU}} \qquad (2)$$

B. Dynamic and Static Power Estimation

Static information is extracted from the application using profilers embedded in software development environments for the cores (e.g., GNU gprof, Intel VTune Amplifier XE). Special

runtime registers monitor instruction path utilizations inside lanes. Dynamic power estimation relies on lane unit activity rates. Like Power Factor Approximation [18], it focuses on a unit's input statistics. Input activity rates are deduced via timing simulations. Our model uses fixed combinations of process corner and values for voltage, frequency and temperature; it is simple to extend for other values since only constants change as discussed below. Execution times and utilization figures are obtained with ModelSim simulations using the RTL system model. The Xilinx XPower tool provides the dynamic power dissipation based on data stored in simulation record files (these .vcd files record the switching activities of all logic and wires, which are generated by ModelSim during the timing simulations with the place-and-route netlist). Timing simulations employ real floating-point data. For high accuracy, power measurements are taken during time windows where executing kernels do not stall due to DMA transfers.

Results show a linear dependence between an ALU's dynamic power P_{ALU} and utilization (or activity rate) that can be approximated with $P_{ALU} = \left(\sum_i K_{exe(i)} w_i\right) U_{ALU}$, where w_i is the fraction of utilization that targets execution unit i; $K_{exe(i)}$ is a constant coefficient measured in mW per percent of utilization (mW/%) and determined by experimentation. VRF utilization is assumed proportional to $2U_{ALU} + U_{LDST}$ since a LDST instruction has either a read or a write, whereas an ALU instruction has one or two reads and one write. VRF and LDST dynamic power depend linearly on VRF and LDST utilization, respectively. Moreover, the crossbar and memory dynamic power depend almost linearly on LDST utilization. Small errors are due to fine-grain effects (e.g., access patterns and toggling rates in netlist signals due to data randomness).

Table 2 summarizes the power model equations for VP components. All *K*s are constant coefficients measured in mW per percent of utilization (mW/%), and are found by experimentation followed by linear approximation. They are shown in Table 3 along with their

standard deviation and the mean absolute dynamic power estimation error of VP components. The estimation of dynamic power using unit utilizations results in a 13% confidence interval.

The total dynamic power dissipation for M lanes and L memory banks is:

$$P_{TOTAL}^{D} = 2P_{VC} + M * P_{LANE} + L * P_{MEM\_BANKs} = 2P_{VC} + M * \left(P_{ALU\_CTRL} + P_{ALU\_EXE} + P_{LDST} + P_{VRF}\right) + L * P_{MEM\_BANKs} \quad (3)$$

For a vector kernel, the ratio $U_{LDST} / U_{ALU} = \alpha$ is assumed constant (it is known in advance or the number of memory accesses depends on some computed values). The total dynamic power is:

$$P_{TOTAL}^{D} \approx M * U_{ALU} * \left[2K_{VC} * \frac{1}{VL} * (1+\alpha) + K_{ALU\_CTRL}^{DATA} + \sum_{i} K_{exe(i)} w_i + \right. \\ \left. + \alpha * K_{LDST}^{DATA} + K_{VRF} * (1+2\alpha) + \alpha * K_{MEM\_BANKs} \right] + M * U_{ALU} * \frac{M}{VL} * \left(K_{ALU\_CTRL}^{INSTR} + \alpha * K_{LDST}^{INSTR}\right) \quad (4)$$

For a kernel with fixed VL: (i) The first part in the equation is constant. (ii) The second part increases linearly with M. M has small impact on dynamic energy since $K_{kernel} * \left(K_{ALU\_CTRL}^{INSTR} + \alpha * K_{LDST}^{INSTR}\right) / VL$ is small, especially for large VL; e.g., FIR32, VL=64, loop unrolled three times: $E^{D} \approx K_{kernel} * (324.5 + M * 5.625)$. The needed lane controllers increases with M. (iii) For identical VP sharing and kernel, dynamic energy should stay almost unchanged independent of the number of lanes since it basically depends on utilizations.

The total dynamic energy is obtained from Equations 2 and 4:

$$E^{D} = P_{TOTAL}^{D} * t_{exec} \approx K_{kernel} * \left[2K_{VC} * \frac{1}{VL} * (1+\alpha) + K_{ALU\_CTRL}^{DATA} + \sum_{i} K_{exe(i)} w_i + \right. \\ \left. + \alpha * K_{LDST}^{DATA} + K_{VRF} * (1+2\alpha) + \alpha * K_{MEM\_BANKs} \right] + K_{kernel} * \frac{M}{VL} * \left(K_{ALU\_CTRL}^{INSTR} + \alpha * K_{LDST}^{INSTR}\right) \quad (5)$$

As transistors shrink, the delay/power gap between FPGA and ASIC designs narrows. Power measurements on our prototype are adjusted since the VP does not utilize all FPGA resources;

XPower results are scaled based on the fraction of used FPGA primitives. Table 4 shows the static power dissipation breakdown for 16x16-VP (16 lanes and 16 memory banks).

Table 2. Dynamic power model equations.

Component	Model	Details
Instruction queues and ALU controller	$P_{ALU\_CTRL} \approx K^{INTSR}_{ALU\_CTRL}\frac{M}{VL}U_{ALU} + K^{DATA}_{ALU\_CTRL}U_{ALU}$	Depends on the instruction and data throughputs
ALU execution units: Add/Subtract; Multiply (MUL); MISC: NEG.ABS,MOV	$P_{ALU\_EXE} \approx \sum_i P_{exec(i)} = \sum_i K_{exec(i)}U_{exec(i)} = U_{ALU}\sum_i K_{exe(i)}w_i$	$\sum_i w_i = 1$ for all *i*s w_i is the fraction of ALU utilization that targets the unit *i*
Instruction queues and LDST controller	$P_{LDST} \approx \left(K^{INTSR}_{LDST}\frac{M}{VL} + K^{DATA}_{LDST}\right)U_{LDST}$	Depends on the instruction and data throughputs
Vector register file (VRF)	$P_{VRF} \approx K_{VRF}\left(2U_{ALU} + U_{LDST}\right)$	Linear function of the vector instruction throughputs
Vector controller (VC)	$P_{VC} \approx K_{VC}I_{TH} = K_{VC}\frac{M}{VL}\left(U_{ALU} + U_{LDST}\right)$	Linear dependence on vector instruction throughput I_{TH}
Memory banks and crossbar	$P_{MEM\_BANKs} \approx K_{MEM\_BANK}\frac{M}{L}U_{LDST}$	***MxL*-VP**: VP with M lanes and L memory banks

Table 3. Mean absolute error for dynamic power estimation. *K* coefficients with their standard deviation.

	w_{ADD_SUB} / w_{MUL} / w_{MISC}	Mean Absolute Error (%)	
		VP	VP, MC and VM
FIR	0.48/0.48/0.04	6.83	7.89
FFT	0.36/0.36/0.28	8.98	10.43
MM	0.5/0.5/0	6.29	7.74
LU	0.5/0.5/0	8.72	9.76
SpMVM-k1	0/0.99/0.01	7.98	10.11
SpMVM-k2	0.96/0/0.4	10.20	13.72
OVERALL		8.16	9.95
By linear approximation (μW / %) $K^{INTSR}_{ALU\_CTRL} \approx 28$ - standard deviation (std.) = 3.8 $K^{DATA}_{ALU\_CTRL} \approx 18$ - std.=1.9 $K_{ADD\_SUB} \approx 215$ - std.=14 $K_{MUL} \approx 71$ (uses DSP48E1 FPGA module) - std.=3.9 $K_{MISC} \approx 18$ - std.= 1.1		$K^{INTSR}_{LDST} \approx 34$ - std. = 4.76 $K^{DATA}_{LDST} \approx 55$ - std.=7.1 $K_{VRF} \approx 34$ - std.=2.8 $K_{VC} \approx 240$ - std. 28 $K_{MEM\_BANK} \approx 147$ - std.= 15	

VI. Energy Minimization

We prototyped various $M\times L$-VPs with M={2, 4, 8, 16} lanes and L=16 memory banks. PG is implemented with sleep transistors, isolation cells and circuits to control power signals; the static

power P_{OFF}^{LANE} of a lane in the OFF state is not zero. Commercial FPGAs lack PG and standby power states [4]. We assume that the static power of PGed lanes can be removed except for ~15% [24]. The vector controllers, crossbar and memory banks are always active. The static power P_{ST}^{M} of M active lanes is given by Equation 6, where P_{ST}^{VP} and P_{ST}^{LANE} are shown in Table 4.

$$P_{ST}^{M} = P_{ST}^{VP} - (L - M) * \left(P_{ST}^{LANE} - P_{OFF}^{LANE} \right) \quad (6)$$

Table 4. Static power consumption breakdown for 16x16-VP (16 lanes and 16 memory banks) on the XC6VLX130t FPGA (the internal supply voltage relative to GND is 1V; the junction temperature is 85^0 C).

Component	**Static Power (mW)**
Total XC6VLX130t	1544
Entire VP, VM, MC and VC $\left(P_{ST}^{VP}\right)$	470 (i.e., 16×25+70)
VP Lane $\left(P_{ST}^{LANE}\right)$	25
VM, MC and VC	70

`

A. Total Energy Minimization

Fig. 2 shows the normalized energy consumption for various execution scenarios, each involving 10,000 FPU operations. Normalization is in reference to 2x16-VP that yields minimum LU energy in Fig. 2.b; 16×16-VP consumes 2.4 times more energy. LU's performance does not change noticeably starting with 4×16-VP, regardless of the population of active lanes, due to stalls caused by scalar divisions on the core and CPU memory accesses. More static power is consumed by extra lanes without any substantial improvement. Static approaches dynamic energy on a 45nm FPGA for medium to low activity. An ASIC simulation with Synopsys tools for a 40nm TSMC HP process shows that the static power of our shared VP is about 33% of the total power dissipation. Also, the static power increases exponentially with the temperature.

The optimal lane population that minimizes the energy is small for low-scalability kernels. For a scalable kernel (e.g., FFT with *VL*=64), the energy drops with additional lanes. It is imperative

to develop a methodology for activating and deactivating vector lanes under workload changes in order to optimize the energy and performance.

Using Equations 2, 5 and 6, the total energy consumption is:

$$E_{TOTAL} = t_{exec} * \left[P_D + P_{ST}^{L} - (L - M) * \left(P_{ST}^{LANE} - P_{OFF}^{LANE} \right) \right] = E^{D} + K_{\ker nel} * \frac{P_{ST}^{L} - (L - M) * \left(P_{ST}^{LANE} - P_{OFF}^{LANE} \right)}{M * U_{ALU}^{M}} \quad (7)$$

where U_{ALU}^{M} is the ALU utilization of M×L-VP. From Equation 5, the dynamic energy is almost independent of M, as expected for a good VP; it really depends on the amount and type of work. Thus, our objective becomes to find the optimal value of M that minimizes the static energy:

$$M_{\min} = \arg\min_{M \in \Phi} \left\{ \frac{P_{ST}^{L} - (L - M) * \left(P_{ST}^{LANE} - P_{OFF}^{LANE} \right)}{M * U_{ALU}^{M}} \right\} \quad (8)$$

where Φ is the set of permissible values for M.

B. Dynamic PG with Static information (DPGS)

When a VP request/release event occurs under DPGS, apriori kernel information is used to compute the optimal lane population that minimizes the energy. Since the static power variables in Equation 8 are fixed for a given VP, the only information needed is U_{ALU}^{M} for all permissible values of M. A simple way to obtain U_{ALU}^{M} is to employ offline simulations of single kernel executions and combinations involving kernel pairs (since vector threads from both cores may run concurrently). A look-up table can contain the optimum value of M for kernel pairs (θ_i, θ_j), where $i, j \in \Theta$ and Θ is the set of all known kernels, including the idle one. Fig. 3.a presents hardware extensions to our VP architecture for supporting software-controlled DPGS. They include a PG sequencer (configured by software) and other elements (sleep transistors and isolation cells). Operating system interrupt routines for power management run on a core or a

power control unit (e.g., similar the Intel7 Nehalem). However, obtaining offline the combined utilization of VP units for all pairs of vector kernels is impractical since kernels may start executing randomly. Also, some kernels may not be known apriori.

C. Adaptive PG with Profiled information (APGP)

For effective PG-driven energy minimization, we use embedded hardware profilers to dynamically monitor lane unit utilizations. The decision to adjust the population of active lanes based on a core's request/release involves specialized hardware. To find the optimal number of lanes that minimizes the energy consumption at runtime, we use the next theorem.

Theorem 1. If the total energy consumption of a kernel for the *M*-lane VP configuration is smaller than the total energy consumption for the *N*-lane configuration, then:

$$\frac{U_{ALU}^{M}}{U_{ALU}^{N}} > RTh_{M/N} \qquad (9)$$

where $RTh_{M/N}$ is a constant depending on *M* and *N*. Additionally, $RTh_{M/N} \leq 1$ for *M*>*N*.

Proof: From $E_{TOTAL}^{M} < E_{TOTAL}^{N}$, a conclusion in Section V.B ($E_{D}^{M} \cong E_{D}^{N}$) and Equation 7:

$$\frac{U_{ALU}^{M}}{U_{ALU}^{N}} > \frac{N}{M} * \frac{P_{ST}^{L} - (L-M) * \left(P_{ST}^{LANE} - P_{OFF}^{LANE}\right)}{P_{ST}^{L} - (L-N) * \left(P_{ST}^{LANE} - P_{OFF}^{LANE}\right)} \qquad (10)$$

where the right-hand term is $RTh_{M/N}$. For *M*>*N*, $U_{ALU}^{M} \leq U_{ALU}^{N}$ since a lane's ALU utilization decreases, or stays constant, when the number of lanes increases. Thus, $RTh_{M/N} \leq 1$.■

Ideal scalability implies $U_{ALU}^{M} = U_{ALU}^{N}$. To evaluate Equation 9 after a VP event, we profile unit utilizations for various VP configurations. We propose a dynamic process where the VP state is changed successively in the right direction (i.e., increasing or decreasing the number of active lanes) until optimality is reached. Since for most of our scenarios minimum energy is reached for

$M \in \{4,8,16\}$, our runtime framework assumes four possible VP states with 0, 4, 8 and 16 active lanes. Fig. 3.b shows hardware extensions for APGP. Each profiler, attached to a VC, monitors ALU and LDST utilizations for the kernel. It captures the average ALU utilization based on the instruction stream flowing through the VC during a chosen time window, as per Equation 1. Simulations show that a window of 1024 clock cycles gives highly accurate results.

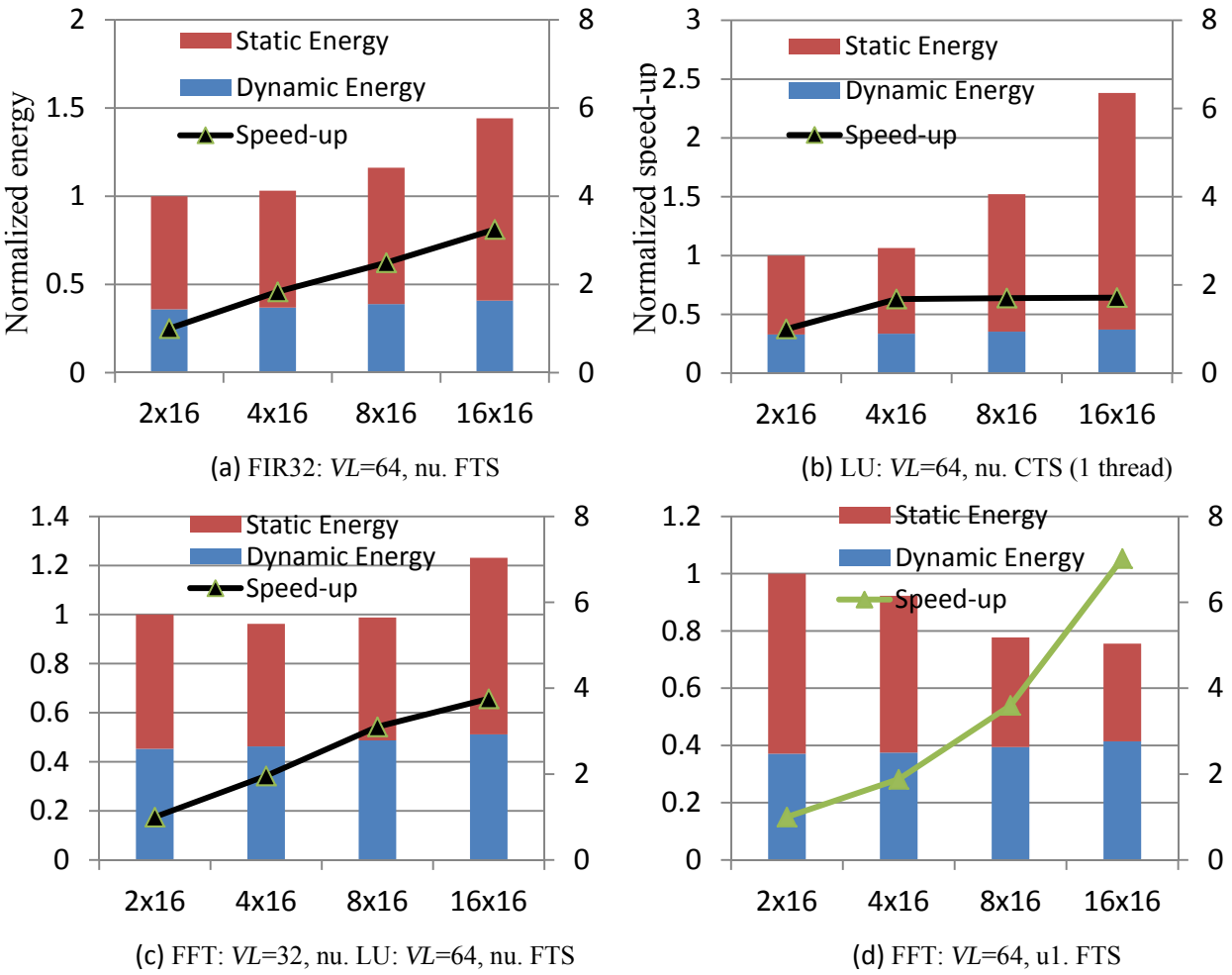
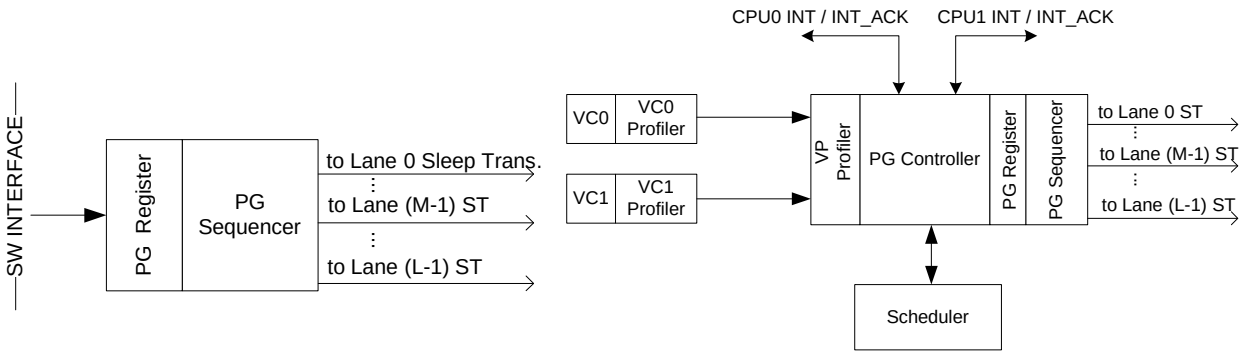


Fig. 2. Normalized energy consumption (bars, values on left axis) for a workload with 10K floating-point operations and various kernels. Normalization is in reference to 2x16-VP; nu – no: loop unrolling; u1- loop unrolled once. FTS is applied to two threads. The normalized speed-up (values on right axis) is shown as a line.

The PG Controller (PGC) aggregates utilizations from both threads (using the profilers) and implements the PGC state machine of Fig. 4. We use two types of thresholds: (i) the absolute

threshold $ATh_{M->N}$ which is used if the ratio U^M_{ALU} / U^N_{ALU} is not available for the current kernel combination and $M \rightarrow N$ represents a transition from the M- to the N-lane VP, and (ii) the relative threshold $RTh_{M/N}$ of Equation 9. $RTh_{M/N}$ is used to compare utilizations with profiled M and N lanes. Absolute thresholds are chosen empirically such that, for a given ALU utilization the probability that the current configuration will be kept is minimum if a configuration of lower energy consumption exists. Absolute thresholds enable PGC to initiate a state transition if there is a probability that the current state is not optimal. For example, $ATh_{8->16}$ is such that the probability $P(U^8_{ALU} < ATh_{8->16} | 16L \min energy) \cong 0$. $RTh_{M/N}<1$ for $M>N$, as per the theorem. Besides these thresholds, the PG Controller contains the profiled utilization registers $U^M_{ALU}, M \in \{4,8,16\}$ (one for each VP configuration).



(a) (b)

Fig. 3. Hardware for (a) DPGS and (b) APGP. In DPGS or APGP, software or the PG Controller, respectively, configures the PG Register. VP Profiler: aggregates utilizations from VCs. ST: Sleep Transistor (Header or Footer).

As per APGP, after a VP request/release event that may change the utilization figures, and thus the optimal configuration, the utilization registers are reinitialized. Bit *Vld* in Fig. 4 shows if the ALU utilization register U^M for the M-lane VP contains an updated value (the ALU subscript in the utilization variable is dropped to simplify the discussion). If the VP is initially idle (0L),

PGC will power up eight lanes to enter the 8L state. We bypass the 4L configuration since 8L has the highest probability to be the optimal energy state for our scenarios. VP uses data from at least one profile window in order to update utilization figures. If one of the inequalities based on the absolute threshold is met, PGC initiates a transition to another state. After each profile window, the utilization register for the current state is updated. A transition between two stable VP operating states involves the following steps and three transitional VP non-operating states:

1. *INT state:* Stop the Scheduler to acknowledge new VP acquire requests and send a hardware interrupt to core(s) that have acquired VP resources.
2. *PW state:* After ACKs from all cores, reconfigure the PG Sequencer for a new VP state.
3. *CFG state:* Reconfigure the Scheduler with the chosen number of active lanes and enable it to acknowledge new VP acquire requests.

In a new state, the utilization register is updated after a full profile window. If one of the inequalities is met, a transition occurs. Up to three transitions are allowed after a VP event in order to avoid repetitive transitions that increase time/energy overheads. The resources consumed by the profilers and the PGC are less than 1% of the VP's resources. As PGC events are rare, the PGC's dynamic power consumption is insignificant compared to that of the VP.

VII. Simulation Model and Experimental Setup

Our simulator models vector-thread executions for various VP configurations. The model is based on performance and power figures gathered from RTL and netlist simulations, as described in Section V. It contains information necessary to compute the execution time and energy consumption for any combination of kernels (θ_i, θ_j) running in any VP state. Each combination

of kernels (θ_i, θ_j) is represented by the utilization(s) $U^M(\theta_i)$ and $U^M(\theta_j)$ and the total power $P^M(\theta_i, \theta_j)$ for kernels running on the M-lane VP, for all values of $M \in \{4,8,16\}$.

The model accounts for all time and energy overheads due to state transitions, as shown in Table 5. Since our lane implementation is almost eight times larger in area than a floating-point multiply unit in [24], which is PGed in one clock cycle, we assume that a VP lane wakes up in eight clock cycles. One lane is powered up/down at a time by the PG Sequencer to avoid excessive currents in the power net. VP components that are not woken up or gated during state transitions consume static energy as usual.

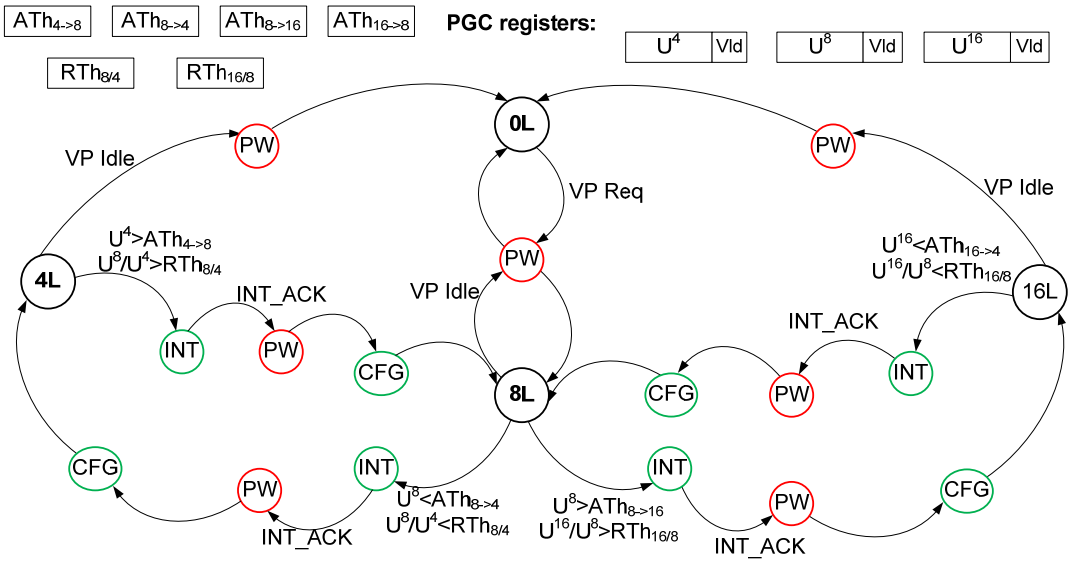


Fig. 4. PG Controller (PGC) state machine and PGC registers for state transitions under APGP. INT, PW and CFG are transitional VP (i.e., non-operating) states. 4L, 8L and 16L are stable VP operating states that represent the 4-, 8- and 16-lane VP configurations. *M*L is a PGC state with *M* active lanes, $M \in \{0,4,8,16\}$; INT is a PGC state where the PGC asserts an interrupt and waits for an Interrupt Acknowledge (INT_ACK); PW is a PGC state where some of the VP lanes are powered-up/down; CFG is a PGC state where the Scheduler is reconfigured to a new VP state. Threshold registers are fixed during runs and utilization registers are updated for every profile window. The registers store 8-bit integers. The Vld bit is used to show that the ALU utilization register U^M, with M= 4, 8 or 16, for the *M*-lane VP configuration does not contain an updated value.

For diverse workloads, we created benchmarks composed of random threads running on cores. Each thread has VP busy and idle periods. During idle periods the core is often busy either with memory transfers or scalar code. A thread busy period is denoted by a vector kernel θ_i and a

workload expressed in a random number of floating-point operations; a thread idle period is denoted by a random number of VP clock cycles. Ten fundamental vector kernels were used to create execution scenarios. Two versions of each kernel in Section IV were first produced with low and high ALU utilization, respectively. The kernel workload is uniformly distributed so that enough data exists in the vector memory for processing without the need for DMA. By adding an idle kernel, 55 unique kernel pairs were produced plus 10 scenarios with a single active kernel on a core. Based on Section VI.C, we get the values: $ATh_{4->8} = 50\%$, $ATh_{8->16} = 60\%$, $ATh_{8->4} = 50\%$, $ATh_{16->8} = 72\%$, $RTh_{8/4} = 0.6739$, and $RTh_{16/8} = 0.7581$.

Table 5. Time and energy overheads for PGC state transition.

	Time overhead (cycles)	**Energy overhead**
Call interrupt routine	20 (for MicroBlaze)	Based on actual runs
Save vector registers (M-lane configuration)	$No\_dirty\_vregs^{1} * \frac{VL}{M}$ [1]*# of* vector registers to be saved/restored	Time overhead × [(Dynamic power to store the vector registers) + (Static power)]
Power up (one lane at a time)	8×(No of lanes to be powered up) [24]	20×(Time overhead)×(Static power when the lane is ON) [24]
Power down (one lane at a time)	0	(8 cycles)× (Static power when the lane is ON) [24]
Acquire VP and restore the vector registers (N-lane configuration)	$10 + No\_dirty\_vregs * \frac{VL}{N}$ 10 cycles to acquire VP and restore reg.	Time overhead × [(Dynamic power to load the vector registers) + (Static power)]

VIII. Experimental Results

Fig. 5 shows the breakdown of VP normalized execution time and energy consumption when the majority of kernels have low ALU utilization. The ratio of low to high utilization kernels in a thread is 4:1. The idle periods between consecutive kernels in a thread are uniformly distributed in the ranges: [1000, 4000], [5000, 10000] and [10000, 30000] clock cycles.

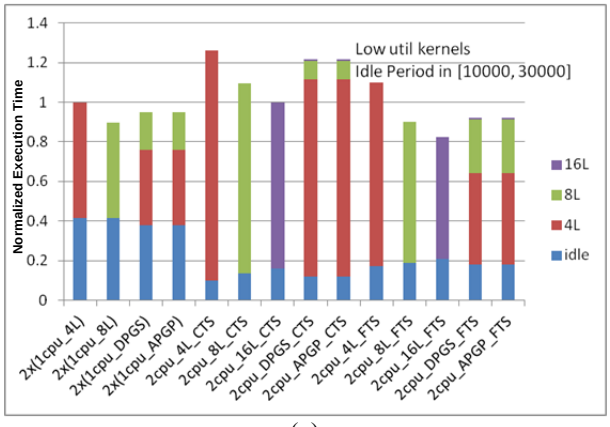
Our conclusions are: (i) FTS generally produces the lowest energy consumption. For either CTS or FTS, DPGS or APGP minimizes the energy consumption compared to scenarios without PG. (ii) Except for two scenarios, 2x(1cpu_8L) and 2cpu_16L_FTS, FTS with DPGS or APGP

also minimizes the execution time. These PG schemes also yield 30-35% and 18-25% less energy as compared to 2x(1cpu_8L) and 2cpu_16L_FTS, respectively. (iii) Scenarios with two cores and a private per-core VP yield lower execution time than CTS because CTS does not sustain high utilization across all lane units. (iv) DPGS or APGP applied to CTS reduces the energy compared to 2x scenarios. As idle periods decrease, CTS becomes less effective; e.g., a 5% gain in consumption for DPGS-driven CTS with a slowdown of 70% compared to 2x(1cpu_4L). Finally, (v) time-energy overheads due to state transitions are negligible; they are not shown in Fig. 5-7. The total time overhead is upper-bounded by 0.3% and 0.7% of the total execution time for DPGS and APGP, respectively. The total energy overhead is upper-bounded by 0.23% and 0.57% of the total energy consumption for DPGS and APGP, respectively.

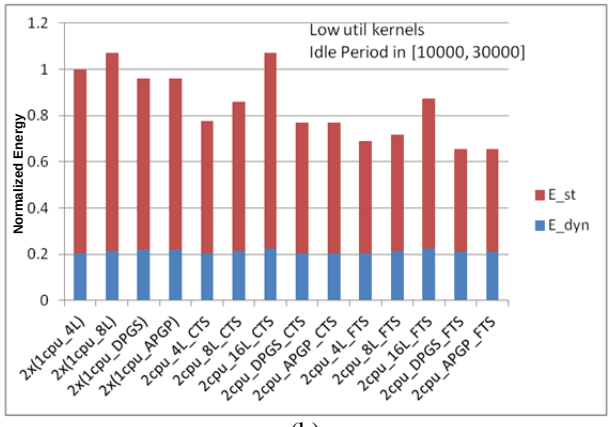
Fig. 6 shows the normalized execution time and energy consumption for threads containing kernels with balanced ALU utilization figures; the ratio between low and high utilization kernels in a thread is 1:1. FTS under DPGS or APGP yields the minimum energy while the performance is better than FTS with eight fixed lanes. Fig. 7 shows the normalized execution time and energy consumption for threads dominated by high ALU utilization kernels; the ratio between low and high utilization kernels is 1:4. As the number of thread kernels with high ALU utilization increases, the portion of time spent in the 16L state increases for FTS under DPGS or APGP. The performance of the PG schemes is better than that of a fixed VP with eight lanes, and approaches the performance of the 16L FTS-driven configuration. As expected, the energy is reduced drastically with FTS and DPGS or APGP compared to all other scenarios.

IX. Conclusions

We proposed two energy reduction techniques to dynamically control the width of shared VPs in multicores. We first introduced an energy estimation model based on theory and observations deduced from experimental results. Although we presented detailed results for an FPGA prototype, ASIC simulations show that this model is also valid for ASICs; only the values of some model coefficients, that depend on the chosen hardware platform anyway, must change. For given vector kernels, VP's dynamic energy does not vary substantially with the number of vector lanes. Consequently, we proposed two PG techniques to dynamically control the number of lanes in order to minimize the VP's static energy. *DPGS* uses apriori information of lane utilizations to choose the optimal number of lanes. *APGP* uses embedded hardware utilization profilers for runtime decisions. To find each time the optimal number of lanes that minimize the static energy, the VP state is changed to reach optimality for the given workload. Benchmarking shows that PG reduces the total energy by 30-35% while maintaining performance comparable to a multicore with the same amount of VP resources and per-core VPs.



(a)



(b)

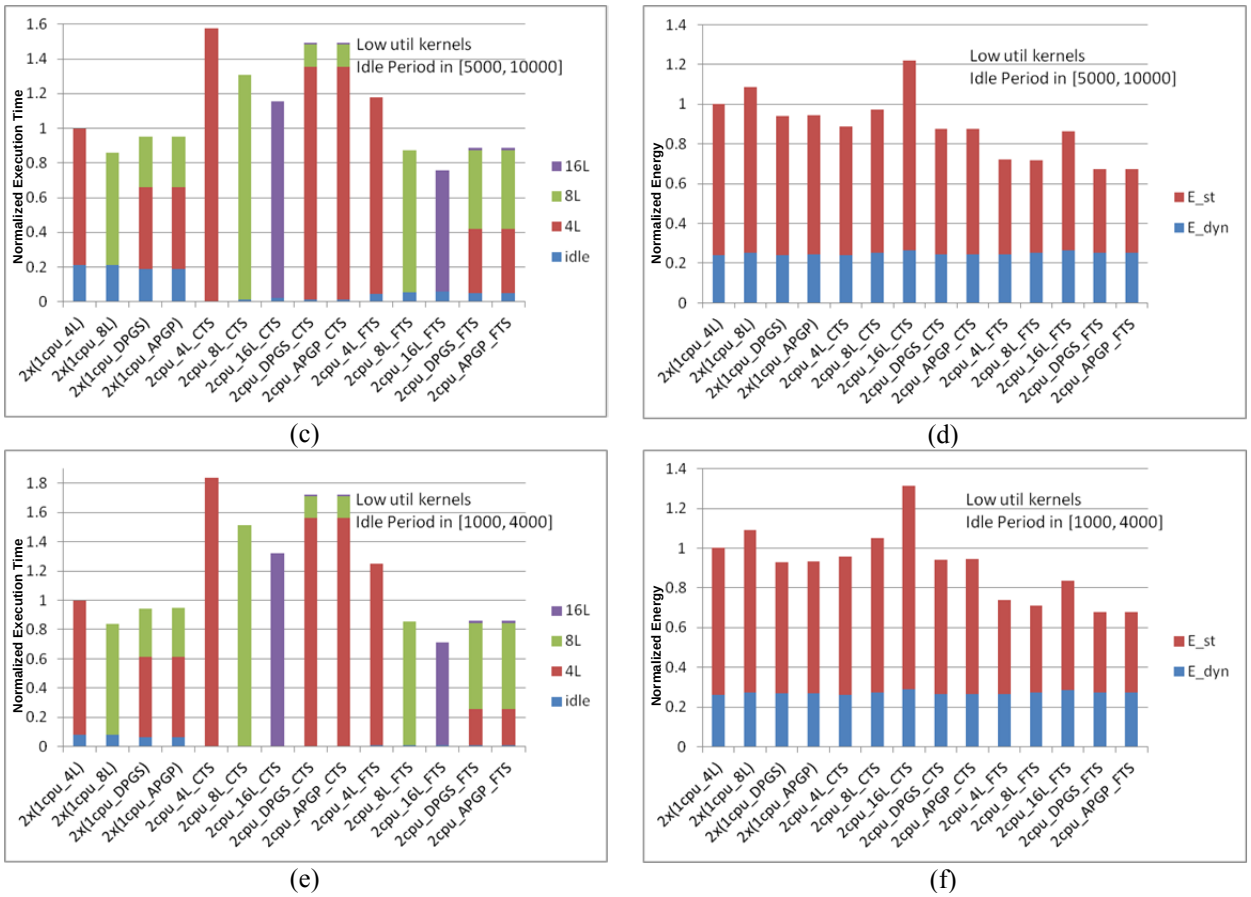
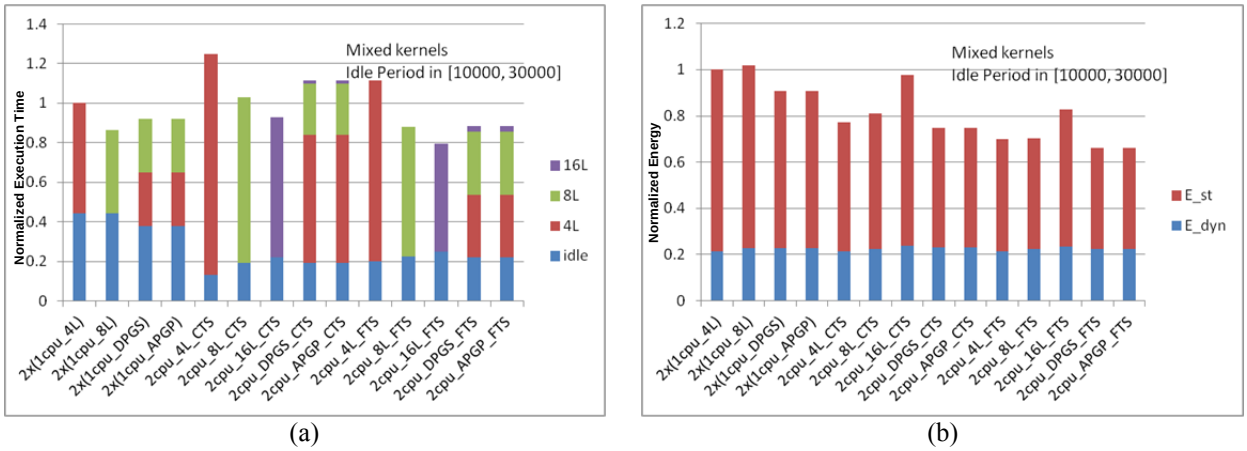
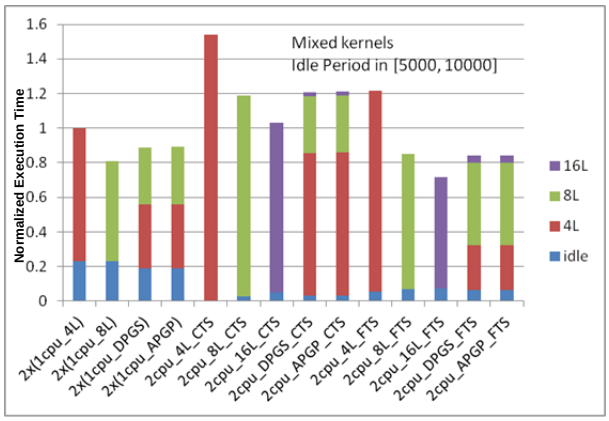
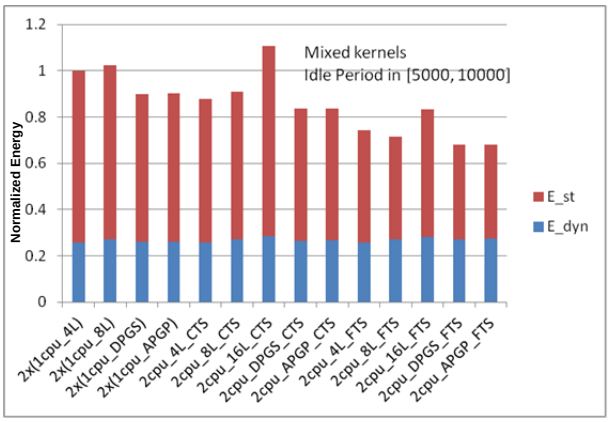


Fig. 5. Normalized execution time (a, c, e) and normalized energy consumption (b, d, f) where the majority of kernels in a thread have low ALU utilization, for various idle periods. The ratio of low to high utilization kernels in a thread is 4:1. E_st and E_dyn are the energy consumptions due to static and dynamic activities, respectively. "2x" means 2 cores/CPUs of the type that follows in parentheses, such as "(1cpu_4L)" which means 1 core having a private VP with 4 lanes. Whenever CTS or FTS shows, it implies 2 cores with VP sharing.

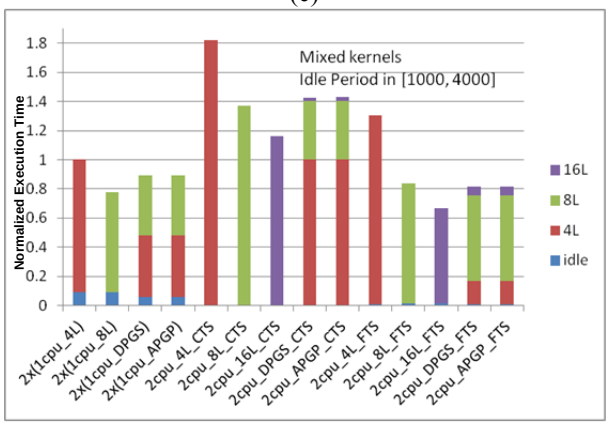




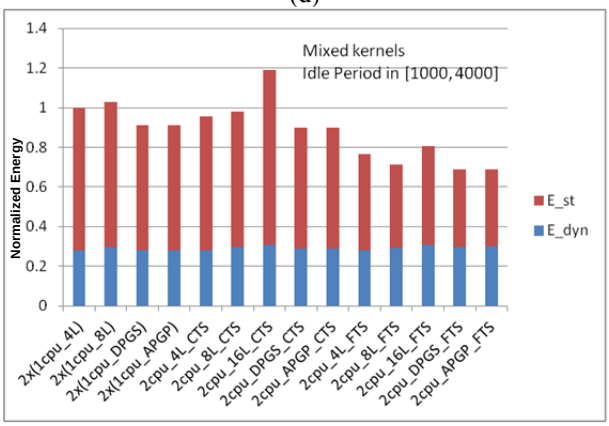
(c)



(d)

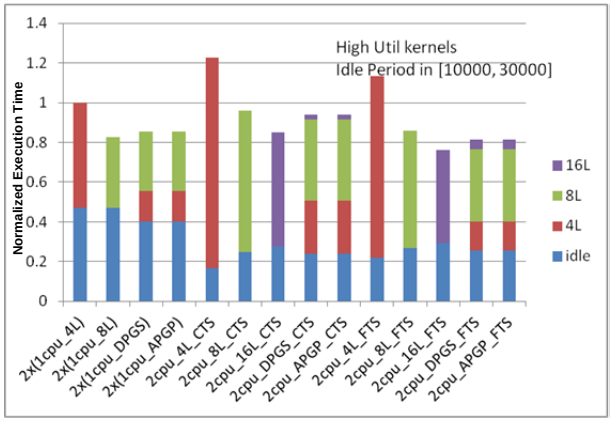


(e)

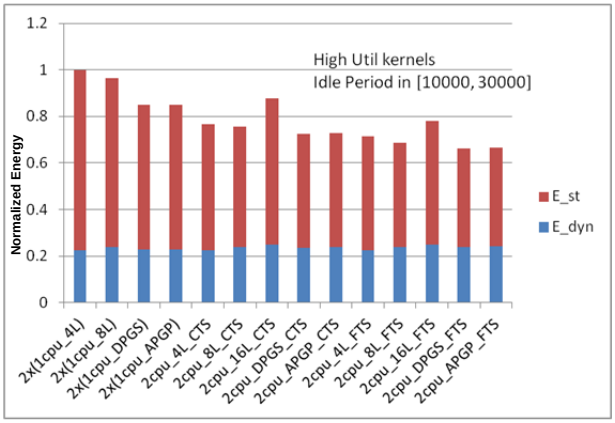


(f)

Fig. 6. Normalized execution time (a, c, e) and normalized energy consumption (b, d, f) for threads with balanced utilization kernels, for various idle periods. The ratio of low to high utilization kernels in a thread is 1:1.



(a)



(b)

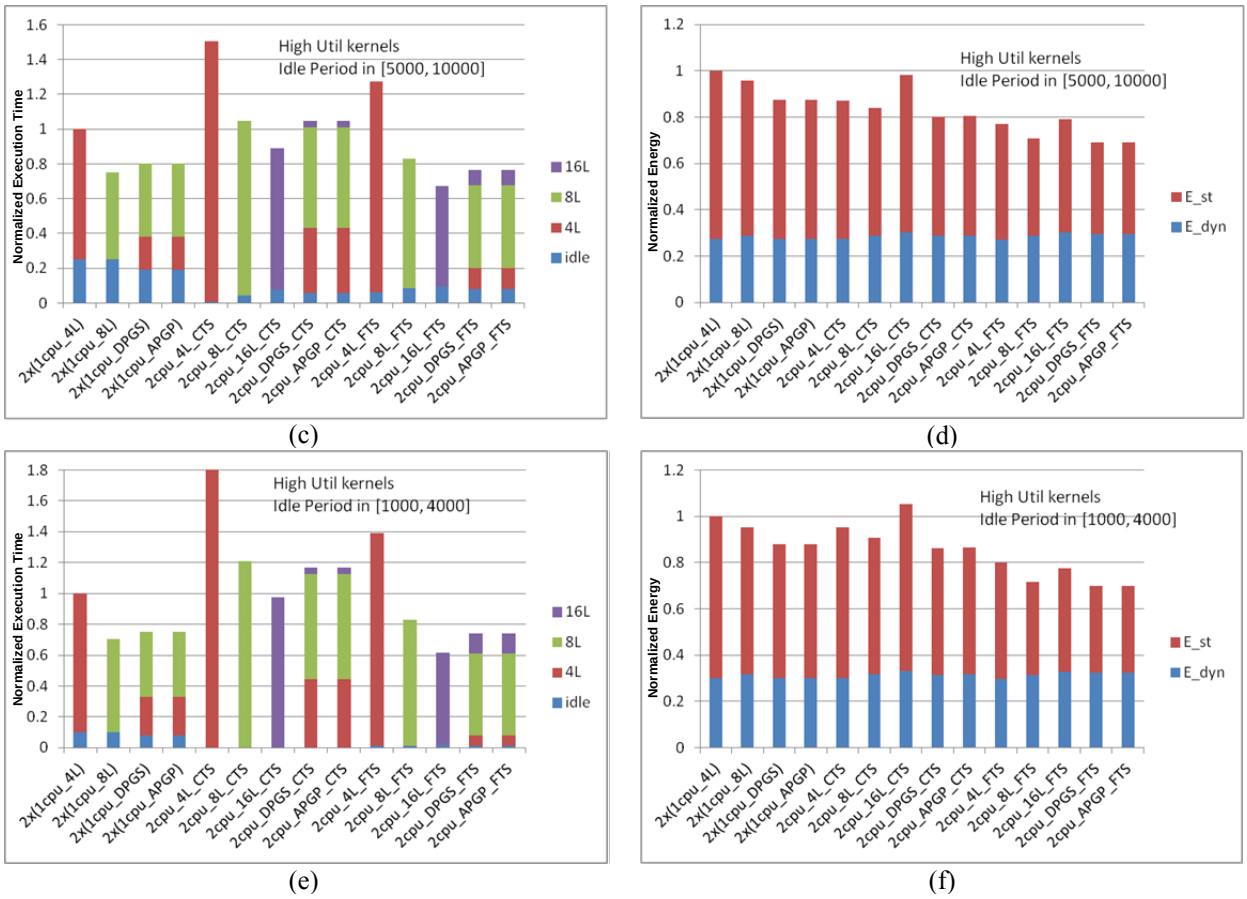


Fig. 7. Normalized execution time (a, c, e) and normalized energy consumption (b, d, f) for threads dominated by high utilization kernels, for various idle periods. The ratio of low to high utilization kernels in a thread is 1:4.

References

[1] T. Hiramoto and M. Takamiya, "Low power and low voltage MOSFETs with variable threshold voltage controlled by back-bias", *IEICE Trans. Elec.*, vol.E83, no.2, 2000, pp.161-169.

[2] M. Woh, et al., "Analyzing the scalability of SIMD for the next generation software defined radio," *IEEE Int. Conf. Acous., Speech Sign. Proc.*, March-April 2008, pp. 5388-5391.

[3] G. Chrysos, "Intel Xeon Phi coprocessor," *Hot Chips Symp.*, Cupertino, CA, Aug. 2012.

[4] S. Ishihara, M. Hariyama, and M. Kameyama, "A low-power FPGA based on autonomous fine-grain power gating," *IEEE Trans. VLSI*, vol.19, no.8, pp.1394-1406.

[5] M. Keating, D. Flynn, R. Aitken, A. Gibsons, and K. Shi, *Low power methodology manual for system on chip design*, NewYork:NY, Springer, 2008.

[6] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," *38th Ann. Int. Symp. Comp. Arch.*, 2011, pp. 365-376.

[7] C. Kozyrakis and D. Patterson, "Scalable, vector processors for embedded systems," *IEEE Micro*, vol. 23, no. 6, Nov.-Dec. 2003, pp. 36- 45.

[8] Y. Lin, et al., "SODA: A low-power architecture for software radio," *33rd IEEE Ann. Int. Symp. Comp. Arch.*, Boston, 2006, pp. 89-101.

[9] M. Woh, et al., "AnySP: anytime anywhere anyway signal processing," *IEEE Micro,* vol.30, no.1, 2010, pp.81-91.

[10] S. F. Beldianu and S. G. Ziavras, "On-chip vector coprocessor sharing for multicores," *19th Euromicro Int. Conf. Paral. Distr. Network-Based Sys.*, Feb. 2011, pp.431-438.

[11] M. Anis, S. Areibi, and M. Elmasry, "Design and optimization of multithreshold CMOS (MTCMOS) circuits," *IEEE Trans. CAD*, vol. 22, no.10, Oct. 2003, pp. 1324- 1342.

[12] H. Yang, and S. G. Ziavras, "FPGA-based vector processor for algebraic equation solvers," *IEEE Intern. Systems-On-Chip Conf.*, 2005, pp. 115-116.

[13] J. Li and J. F. Martinez, ''Power-performance considerations of parallel computing on chip multiprocessors,'' *ACM Trans. Arch. Code Optim.*, Dec. 2005, pp. 397-422.

[14] S. F. Beldianu and S. G. Ziavras "Multicore-based vector coprocessor sharing for performance and energy gains," *ACM Trans. Embedded Comp. Syst.*, vol. 13, no. 2, Sept. 2013.

[15] J. Yu, C. Eagleston, C. H.-Y. Chou, M. Perreault, and G. Lemieux, "Vector processing as a soft processor accelerator," *ACM Trans. Reconf. Tech. Sys.*, vol. 2, no. 2, June 2009, pp. 1-34.

[16] A. Bakhoda, et al., “Analyzing CUDA workloads using a detailed GPU simulator,” *IEEE Int. Symp. Perf. An. Sys. Softw.,* April 2009, pp.163-174.

[17] V. A. Korthikanti and G. Agha, “Towards optimizing energy costs of algorithms for shared memory architectures,” *22nd ACM Symp. Paral. Alg. Arch.*, NY, 2010, pp. 157-165.

[18] S. Powell and P. Chau, "Estimating power dissipation of VLSI signal processing chips: the PFA techniques," *IEEE Workshop VLSI Sign. Proc.*, 1990, pp. 250-259.

[19] S. Rivoire, R. Schultz, T. Okuda, and C. Kozyrakis, "Vector lane threading," *Int. Conf. Paral. Proc.*, Aug. 2006, pp.55-64.

[20] L. Oliker, et al., “Evaluation of cache-based superscalar and cacheless vector architectures for scientific computations,” 18th Ann. Intern. Conf. Superc., Nov. 2003.

[21] J. Leverich, et al., “Power management of datacenter workloads using per-core power gating,” *IEEE Comput. Archit. Lett.*, vol. 8, no. 2, July 2009, pp. 48-51.

[22] Y. Wang and N. Ranganathan, "An instruction-level energy estimation and optimization methodology for GPU," *IEEE 11th Int. Conf. Comp. Inf. Tech.*, Aug.-Sept. 2011, pp. 621-628.

[23] W. Wang and P. Mishra, "System-wide leakage-aware energy minimization using dynamic voltage scaling and cache reconfiguration in multitasking systems," *IEEE Trans. VLSI. Syst.,* March 2011, pp. 1-9.

[24] S. Roy, N. Ranganathan, and S. Katkoori, "A framework for power-gating functional units in embedded microprocessors," *IEEE Trans. VLSI Syst.*, vol. 17, no. 11, Nov. 2009, pp.1640-1649.

[25] O. Avissar, R. Barua and D. Stewart, “An optimal memory allocation scheme for scratch-pad-based embedded systems,” *ACM Trans. Emb. Comp. Sys.*, vol. 1, no. 1, 2002, pp. 6–26

[26] *NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110: The Fastest, Most Efficient HPC Architecture Ever Built*," White Paper, NVIDIA Corp., 2012.

[27] *Reducing System Power and Cost with Artix-7 FPGAs*, Xilinx 2013, http://www.xilinx.com/support/documentation/white_papers/wp423-Reducing-Sys-Power-Cost-28nm.pdf.

[28] *28 Nanometer Process Technology*, 2012 http://www.nu-vista.com:8080/download/brochures/2011_28 Nanometer Process Technology.pdf.

[29] J. Fowers, G. Brown, P. Cooke, and G. Stitt, "A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications," *20th ACM/SIGDA Int. Symp. FPGAs*, Monterey, CA, Feb. 22–24, 2012, pp. 47-56.

[30] P.-H. Wang, C.-L. Yang, Y.-M. Chen, and Y.-J. Cheng, "Power gating strategies on GPUs," *ACM Trans. Archit. Code Optim.,* vol. 8, no. 3, Oct. 2011.

[31] Y. Wang, S. Roy, and N. Ranganathan, "Run-time power-gating in caches of GPUs for leakage energy savings," *Des. Autom. Test Europe,* March 2012, pp.300-303.