The Journal Of Supercomputing manuscript No.
(will be inserted by the editor)

Efficiency Analysis of Modern Vector Architectures

Vector ALU Sizes, Core Counts and Clock Frequencies

Adrian Barredo · Juan M. Cebrian
Mateo Valero
Marc Casas · Miquel Moreto

Received: date / Accepted: date

Abstract Moore's Law predicted that the number of transistors on a chip would double approximately every two years. However, this trend is arriving at an impasse. Optimizing the usage of the available transistors within the thermal dissipation capabilities of the packaging is a pending topic.

Multi-core processors exploit coarse-grain parallelism to improve energy efficiency. Vectorization allows developers to exploit data-level parallelism, operating on several elements per instruction and thus, reducing the pressure to the fetch and decode pipeline stages.

In this paper we perform an analysis of different resource optimization strategies for vector architectures. In particular we expose the need to break down voltage and frequency domains for LLC, ALUs and vector ALUs if we aim to optimize the energy efficiency and performance of our system. We also show the need for a dynamic reconfiguration strategy that adapts vector register length at runtime.

Keywords Vector · Efficiency · DVFS · Power Wall

Adrian Barredo
E-mail: adrian.barredo@bsc.es

Juan M. Cebrian
E-mail: juan.cebrian@bsc.es

Mateo Valero
E-mail: mateo.valero@bsc.es

Marc Casas
E-mail: marc.casas@bsc.es

Miquel Moreto
E-mail: miquel.moreto@bsc.es

1 Introduction

The ever-increasing computational requirements of applications has always been a challenge for hardware and software designers. The end of Dennard scaling translated into a stagnation of CPU clock frequency. After 2010 the frequency increase would be limited to about 5% on a per year basis [13], limiting the main source of performance gains. Therefore, computer architects and software developers focus nowadays on parallelism to improve performance. Parallelism is one of the most important keystones in high performance computing (HPC). While instruction-level (ILP) and thread-level parallelism (TLP) have been extensively studied, data-level parallelism (DLP) is usually underutilized in CPUs, despite its huge potential.

ILP can be understood as the freedom to choose the most convenient instruction interleaving, while maintaining program order. Superscalar (out-of-order) processors exploit ILP to execute independent instructions in parallel. One key problem of such designs is the increasing speed gap between the memory subsystem and cores. This made the memory access latency difficult to hide (*Memory Wall* [47]). In addition, the continuous ILP enhancements exponentially increased the complexity per unit area, leading to a heat density that can not be easily handled. This problem is known as the *Power Wall* [34].

Multi-cores can further increase performance by exploiting TLP. In fact, processor vendors focus nowadays on multi-core designs, even in the mobile market. These designs can alleviate the Power and the Memory Wall problems, but not without limitations. Multi-cores in conjunction with low voltage/frequency can improve the performance per watt, but also significantly increase the circuit sensitivity to electrical noise. Therefore, future processors will have greater tendencies towards hardware faults [42]. Second, heterogeneity of the hardware builds up a *Programmability Wall* [12]. When the programmer is faced with this kind of complex architectures, optimizing for the hardware becomes almost impossible [9].

Finally, DLP can be exposed to the hardware by means of vector computations, where a single instruction operates over several input elements [5,17]. Vector supercomputers appeared in the 1970s [38,14,17,6,45]. These designs exploited DLP with long vectors of thousands of bits. Nowadays, hardware manufacturers focus on short-vector designs (up to 2048-bits), although there are some exceptions (NEC's SX-Aurora features 16,384-bit vectors [35]).

The Single-Instruction Multiple-Data (SIMD) execution model is a far more common vector implementation. The usage of SIMD Instruction Set Architecture (ISA) extensions is ubiquitous in microprocessors from all market segments [23,24,4,41,18,3]. DLP exploitation is not limited to CPUs, Graphic Processing Units (GPUs) are alternative architecture implementations that benefit greatly from DLP. GPUs use a massive multi-core architecture able to execute sets of threads in a lock-step model.

The current trend shows that the vector register size has doubled every four years [20]. 512-bit SIMD implementations from Intel [40,24] and Fujitsu [49] are recent commercial examples. However, the generation of efficient vector

code with increasing register size has several obstacles to overcome. Key performance limiting factors include: a) at a design level, horizontal operations, data structure conversion and divergence control, and b) at a hardware level, register bank size, cache and memory bandwidth and resource underutilization. Ultimately, the effectiveness of a vector architecture depends on its ability to vectorize large quantities of code [39].

In the next chapters, we analyze different resource optimization strategies for vector architectures. In particular, this paper makes the following contributions over the state-of-the-art:

- Create a realistic implementation of a Generic SIMD (G-SIMD) architecture with variable register size based on Intel's SIMD extensions.
- Design a set of vectorized benchmarks representative of different application patterns that are compatible with the G-SIMD ISA.
- Analyze the impact of the vector register size in power and memory.
- Study the influence of DVFS in G-SIMD architectures.
- Analyze the effects of different clock domains on G-SIMD.
- Propose micro-architectural changes to deal with the resource requirements of future vector applications.

The paper is organized as follows. Section 2 discusses the related work and the multi-dimensional optimization problem we are facing. Next, Section 3 presents the simulation environment and the benchmarks used in the evaluation. Section 4 shows the performance and energy evaluation of our design space exploration. Finally, Section 5 summarizes the main conclusions and future work.

2 Motivation and Related Work

In the 1980s and early 1990s, quantitative performance evaluations became the predominant driver to determine how to build effective, cutting-edge microprocessors and computer systems. In contrast, other metrics like cost and area were not specially restrictive. In the mid- to late- 1990s, power per unit area became a concern for architects [27]. While Moore's Law scaling succeeded in reducing the feature sizes of semiconductor devices, their power density and high processor clock rates resulted in microprocessor designs difficult or impossible to cool down.

2.1 The Power Problem

Power and cooling concerns are a twenty-first century issue for computing but they have existed for a long time. For example, the ENIAC machine built in 1947 dissipated 174 kW.

CMOS power consumption can be divided into several categories: dynamic power, leakage power, glitching power, and others. The first two factors are explained below:

- Dynamic Power. It has been the dominant power component for many years, and is given by the equation $P = CV^2Af$, where:
  - *Capacitance (C)*: Total capacitance of the circuit components largely depends on the wire lengths of on-chip structures. Circuit designers are the ones that can influence this metric.
  - *Supply voltage (V)*: Supply voltage (V or Vdd) has dropped steadily with each technology generation. Because of its quadratic influence on dynamic power, this has a large impact on power-aware design.
  - *Activity factor (A)*: It is a fraction between 0 and 1 that refers to how often wires actually transition from 0 to 1 or 1 to 0. Clock signal typically has the highest switching activity, most other wires in the design have activity factors below 1.
  - *Clock frequency (f)*: The clock frequency has a huge impact on power dissipation. Not only does clock frequency directly influence power dissipation, but it also indirectly shapes power by its effect on supply voltage. Typically, maintaining higher clock frequencies may require maintaining a higher supply voltage. This way, the combined V^2f portion of the dynamic power equation has a cubic impact on power dissipation.
- Leakage Power. While dynamic power dissipation represented the predominant factor in CMOS power consumption for many years, leakage energy has been increasingly prominent in recent technologies. It represents 20% [27] or more of power dissipation in current designs and its proportion is expected to increase in the future. Leakage energy can come from several sources, including gate leakage and sub-threshold leakage.

2.1.1 DVFS

Most power control mechanisms focus on dynamic adjustments to supply voltage, clock frequency, or both, and they go under the broad title of *Dynamic Voltage and Frequency Scaling* (DVFS). In the dynamic power equation, a reduction on voltage translates into quadratic power reduction. Reducing supply voltage, however, might possibly reduce the performance of systems as well. In particular, reducing supply voltage often slows transistor switching speed, reducing their maximum working clock frequency.

Unfortunately, the useful voltage range of transistors is rapidly shrinking, to the point of having a negligible effect in the power equation [15]. In practice, we can no longer expect quadratic reductions in power as a trade-off for linear reductions in frequency. Fortunately, it is also well known that performance is not always proportional to frequency [28]. For example, the performance of memory-bound programs is largely unaffected by frequency scaling. Memory-bound code is characterized by a high miss ratio in the last level cache (LLC), which creates long stalls in the processor pipeline. In this case, scaling down frequency causes computation to overlap with memory access without harming the total execution time (as long as we can keep similar memory level parallelism, or MLP) [36, 26].

2.2 Vector Processors

Vector processors are known to be very energy efficient and yield high performance whenever there is enough *Data Level Parallelism* (DLP) [29]. DLP is extracted by applying the same operation on more than one data element simultaneously. The vector paradigm goes back as early as the 1940s [46,19, 25]. Seymour Cray developed a high-performance vector machine called the Cray-1 [37], which used a register-to-register design and included fast non-vector functional units. This set the stage for an era between the mid-1970s until the early 1990s where vector architectures were the design choice for high-performance machines [17].

The ideas found in vector architectures matured and evolved over time. Espasa [16] showed that vector processors can improve their performance and hide latency by applying techniques such as decoupling, out-of-order execution and multithreading. Vector architectures behave well when it comes to applications with high DLP, outperforming scalar designs while saving energy in the process. However, large registers have several drawbacks. For example, if an application can not make full use of every register, then a hardware resource is being wasted. For this reason, fabricating chips with long vector registers for commodity processors has been considered senseless for a long time. Moreover, as the authors in [44] explained, many applications have small data sets or iterate over an iteration space which is smaller than the vector register length.

Nowadays, multimedia applications (rich in DLP) have seen an increasing usage. This trend is expected to continue in the next years. Multimedia extensions or Single-Instruction Multiple-Data (SIMD) extensions have been introduced to deal with the requirements of such applications. SIMD extensions are ubiquitous in all market segments [23,24,4,41,18,3].

2.2.1 Power in Vector Processors

Lemuet at al. [30] discussed the potential of energy efficiency of vector processors as accelerators for high performance computing systems. Lee at al. [29] confirmed that vector-based micro-architectures are more area and energy efficient than scalar-based micro-architectures, even for fairly irregular data-level parallelism. They also explored a series of micro-architectural optimizations to improve performance, area, and energy efficiency of baseline vector cores.

Low-power techniques such as clock gating [31], power gating [43] [21] or DVFS can be combined with vector micro-architectures to further reduce power consumption and increase energy efficiency. H. Inoue presented in [22] a paper studying the impact of SIMD processors in power and energy efficiency in sorting algorithms. They tried different configurations of SIMD width and memory bandwidth and explained the need of balancing memory bandwidth, SIMD width and the number of threads to minimize energy consumption. In [11], the authors perform a similar study for multiple benchmarks from different fields of application. S. Majzoub performed a study of SIMD power

Table 1 Configuration of the gem5 simulations (similar to Intel Icelake).

OoO Core details	
Fetch, decode, rename bandwidth	4 insts/cycle
Dispatch, issue, commit bandwidth	4 insts/cycle
Branch Target Buffer	1 way, 2048 entries
Branch predictor	Bimode, 8K+8K entries
Fetch Queue	32 entries
Fetch Buffer	16B
Decode Buffer	56-μops
Load Queue	72 entries
Store Queue	56 entries
Physical Registers	168 integer + 168 floating point
Issue Queue	97 entries
Re-order Buffer	224 entries
Functional Units	1 Int ALU + 3 Int/FP/SIMD ALU
L1 instruction cache	32KB, 8-way, 1 cycle access latency
L1 data cache	32KB, 8-way, 4 cycle access latency
L2 unified cache	4MB, 16-way, 12 cycle access latency

consumption in multi-core processors [33]. This work explores the technique of voltage islands, where a finite number of supply voltages can be applied to different blocks of a design, depending on the power requirements of the system. In particular, power supplies are associated to cores depending on the nature of the instructions being executed.

Albright et al studied the effects of DVFS in Embedded SIMD Multiprocessors [2]. For the experiments, they chose an example electroencephalography (EEG) application. The application is divided into blocks (according to the application shape), whose granularity was different for every experiment, and searched for their optimal DVFS settings. Their results outperformed traditional DVFS methods and demonstrated a minimum of 50% improvement in performance per watt at 1% performance reduction, whereas a typical DVFS scheme achieved a maximum of 32% improvement at the same point.

3 Experimental Methodology

3.1 Full-System Simulation Infrastructure

We employ *gem5* [7] to simulate an x86 full-system environment that models the operating system and the architecture in detail. The simulated system runs 16.04 Ubuntu with a 4.9.4 Linux kernel. gem5 is extended to support Intel's SSE, AVX-2 and AVX-512 instructions. These extensions have been developed to simulate a processor similar to an x86 Icelake processor. ALU latencies and pipeline stages have been selected to emulate this processor. Table 1 summarizes the main simulated core parameters. We simulate one, two and four-core processors using the detailed memory models of gem5 (with a two-level MESI protocol).

3.1.1 Variable Vector-length ISA (G-SIMD)

We have extended the simulated processor with a variable vector-length ISA (G-SIMD). G-SIMD will help us to perform an extensive evaluation based on vector register size. In this evaluation the studied register sizes range from 128 to 2048 bits. In a fully configurable hardware, vector registers would have, in our case, a maximum size of 2048 bits. The most optimal one would be chosen depending on the application properties. If the selected size was less than 2048, the upper bits in the vector register would be clock-gated, to avoid an unnecessary energy consumption.

When it comes to vector units, the CPUs contain 512-bit vector ALUs (VPUs), that is, VPUs can operate 512-bits of data simultaneously in a SIMD manner. Whenever G-SIMD operates on registers wider than 512-bits, the operation is pipelined (as in traditional vector processors). Extra latencies due to pipelining are considered in the results. This decision was taken after an initial design space exploration that showed unreasonably high energy consumption for 1024 and 2048-bit VPUs (a similar pipeline design is used by NEC in [35]).

3.1.2 Study of a Potential Dynamic Reconfiguration Mechanism

This paper focuses on a static evaluation that serves as a design space exploration to assess the potential benefits of a dynamic reconfiguration mechanism. In this case, scenarios are configured at the beginning of the simulations. The evaluation focuses on three different factors (excluding the register size), that alter the performance and energy of the applications. These factors are a) clock frequency for ALUs, b) clock frequency for VPUs and c) last-level cache frequency. Their consideration is based on the fact that vector architectures have better tolerance for high memory latency with vector length. There are two reasons for increased latency: a) if bandwidth does not scale with register size, we will require several memory requests; and b) if we cannot fit all the data into L1 cache the overall latency to access the data will increase (since we have to reach other levels in the memory hierarchy). Our hypothesis is that reducing VPUs frequency independently of the memory subsystem can partially hide memory latency, while using less power for computation in memory-bound codes. For CPU-bound codes, we do the opposite, reducing the operating frequency of the LLC, since they offer better tolerance for slow latency. Furthermore, scaling vector register size usually translates into an increment in the ratio of time that the processor spends running scalar code (Amdahl's law). Our second hypothesis is that separating VPUs into a new clock domain, that operates at a lower frequency than the scalar units, will reduce power without a significant performance penalty. When it comes to arithmetic operations, long vectors will not require to operate at a high frequency, but scalar units will, since they become a bottleneck.

The simulations consider three possible scenarios taking into account the previous factors. When it comes to the last-level cache, a faster one has a lower response latency to memory requests.

```
1  int max_v_length = valib_get_max_vector_length();
2  valib_set_vector_length(max_v_length);
3  vector_function_call(); // vector−length agnostic function
```

Fig. 1 Vector length setter/getter

- ALU @2GHz, VPU @1GHz and fast last-level cache.
- ALU @2GHz, VPU @1GHz and slow last-level cache.
- ALU @4GHz, VPU @1GHz and slow last-level cache.

Power consumption is evaluated with McPAT [32] using a process technology of 22nm (currently the lowest possible), a voltage of 0.6V and the default clock gating scheme. We incorporate the changes suggested by Xi et al. [48] to improve the accuracy of the models. The energy consumption has been dimensioned for the different sizes of the vector register file as well as for the 512-bit VPUs. The operating frequency of the described scenarios is also considered in the power model.

3.2 Benchmarks

Our application suite is based on eight real applications. We employ a hash function (Hash), a quadratic equation benchmark (Quadratic), a matrix-block multiply (MatMul) and a vector reduction (Vreduction) kernel and the swaptions (Swaptions), blackscholes (Blackscholes) and streamcluster (Streamcluster) applications from the ParVec [10] benchmark suite. The former three benchmarks are single-threaded. The latter ones were programmed using the pthreads [8] library to simulate their performance in multi-core scenarios.

We have created our own ISA to be capable of simulating several vector lengths with the same binaries. This new ISA imitates AVX-2 and AVX-512 encoding to provide the same functionalities (e.g. memory addressing, predication, gather/scatter). This ISA was created using macro-instructions, so no optimizations are done by the compiler. Whenever an instruction from this ISA is encountered, our simulator executes it based on the vector length specified by the programmer. Another option is to allow the application to run with the maximum hardware register size, but as we will see later, sometimes it is not the best alternative. The code to perform this process is shown in Figure 1. When it comes to memory instructions, several memory requests are generated when the vector length is wider than the blocksize.

The *vector_function_call* function contains code, such as loops, which will adapt themselves to the specified vector register size. Figure 2 depicts the SIMD function from Streamcluster. Every *valib* function represents a macro-instruction, which is converted into a vector-length agnostic ISA instruction depending on the sources and destination indices specified in the function call. The instruction opcode is implicit in the function name. For example, line 7 represents a vector substraction for float-type elements, where source registers are 1 and 2 and destination register is 3. In this case, the number of iterations

```
1   float volatile zero = 0.0f, result = 0.0f;
2   valib_set_fl(VR0, &zero);
3
4   for (int i = 0; i <= dim − factor; i = factor + i) {
5     valib_ld_fl(VR1, &p1.coord[i]);
6     valib_ld_fl(VR2, &p2.coord[i]);
7     valib_sub_fl_fl_fl(3, 1, 2);
8     valib_mul_fl_fl_fl(1, 3, 3);
9     valib_add_fl_fl_fl(0, 0, 1);
10  }
11
12  valib_red_add_fl(VR0, &result);
13  return result;
```

Fig. 2 Streamcluster vector-length agnostic version

depends on the *factor* variable, which is obtained at runtime by dividing the vector register size by the data type.

4 Evaluation

In this section, we show the simulation results for the applications described in Section 3.2. For every benchmark, we describe the impact of varying the number of threads, vector register size, VPU frequency and last-level cache response time on performance, EDP^2 (as an energy metric that emphasizes performance) and MFlops per watt (as an energy metric that emphasizes low power, and used in the Green500 classification [1]).

Blackscholes. This benchmark shows a linear benefit in performance as register size increases (Figure 3-left), until 512 bits, when the vector scaling becomes sublinear. However, if we increase the frequency of scalar ALUs, the overall performance scaling of the vectorized code improves. This can be clearly seen for 4 cores and 2048-bit registers, where the speedup jumps from 34x to almost 49x. This is due to the fact that Blackscholes is a CPU-bound application. Having vector and scalar units operating at the same frequency makes the latter become a bottleneck. A faster last-level cache does not benefit performance in CPU-bound applications, so we can reduce LLC frequency to save energy. Figure 3-right shows the EDP^2 metric for Blackscholes. Here we can see that for register sizes under 512 bits it is not beneficial to run on different ALU and VPU frequencies, regardless of the number of cores. However, it is critical for 2048-bit registers, where we can see 2x improvements in EDP^2. In addition, we can see a clear advantage of running with a slow LLC for all register sizes and core counts.

For Blackscholes, the best performance outcome is obtained when choosing the configuration: 2048-bit vector registers, fast scalar units and four threads. However, if we want to maximize the performance per watt, the best results are obtained running 1024-bit registers with slow LLC (Figure 4).

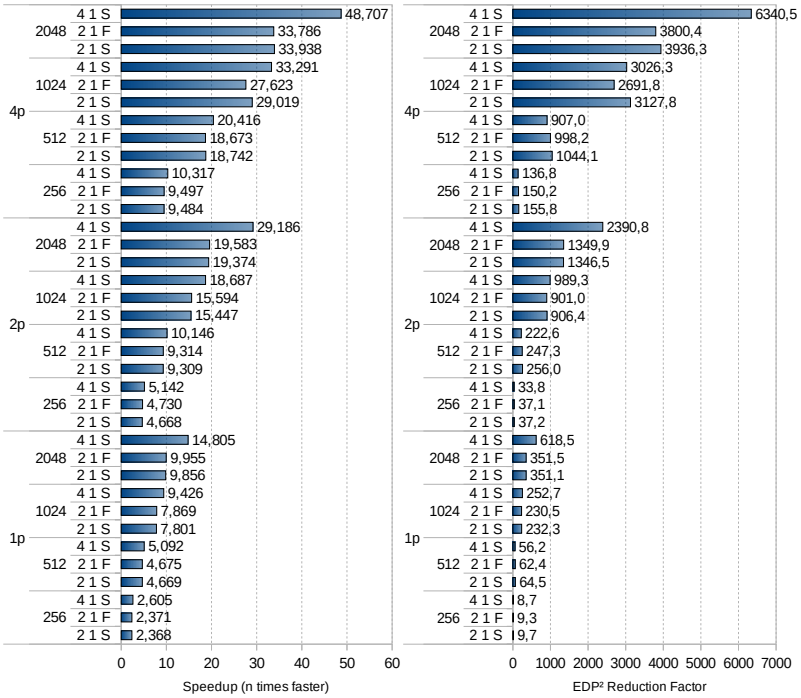

Fig. 3 Speedup (left) and EDP^2 improvement (right) for Blackscholes (more is better) normalized to 1 core with 128-bit registers running at 1Ghz with slow LLC. Y axis shows different combinations of core count, reg. size, and frequencies (scalar, vector and LLC).

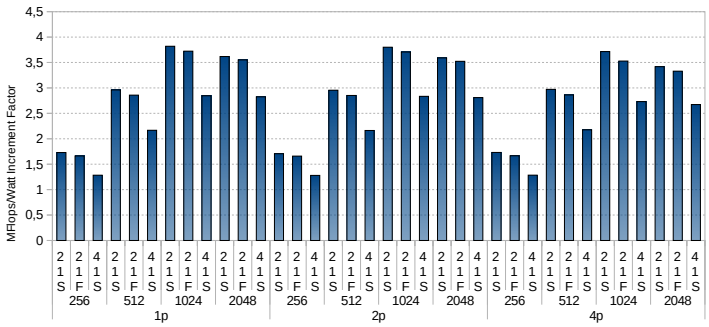

Fig. 4 MFlops per Watt improvement (more is better) for Blackscholes with different combinations of core count, reg. size, and frequencies (scalar, vector and LLC).

Hash. For Hash, the results when increasing the register size are far from the theoretical linear scaling values (Figure 5-left). Hash scales up to 512-bit registers, and becomes sublinear after that. Faster last-level caches provide a slight improvement in performance, but overall Hash is CPU-bound. Hash benefits from faster ALUs, since scalar operations become eventually the main bottleneck. For 2048-bit registers, going from 2Ghz to 4Ghz improves performance by an additional 2.7x. EDP^2 numbers show a similar behavior (Figure

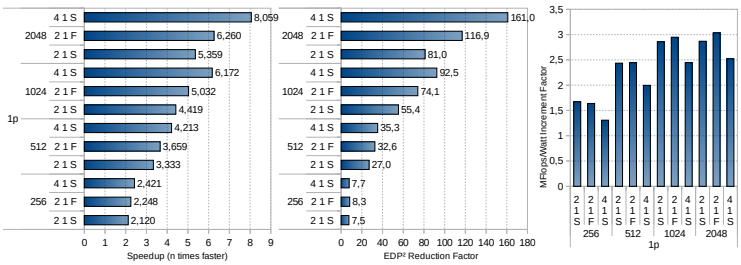

Fig. 5 Speedup (left) EDP^2 improvement (center) and MFlops per Watt improvement (right) for Hash (more is better) normalized to 1 core with 128-bit registers running at 1Ghz with slow LLC. Y axis shows different combinations of reg. size, and frequencies (scalar, vector and LLC).

5-center), showing significant benefits from both fast LLC and faster ALUs as we increase the register size.

From a performance perspective, the best configuration for Hash would be using fast ALUs and fast LLC, however, from a pure energy efficiency perspective, a 2048-bit register size with fast LLC and running ALUs at 2Ghz is preferred (Figure 5-right).

MatMul. Scalability with vector register size for MatMul is sublinear (Figure 6-left). Moreover, it stops scaling for a vector register size of 1024 bits, regardless of the number of cores. MatMul is a memory-bound application, where LLC latency is critical to performance. Faster ALUs show an improvement to execution, meaning that despite of being memory bound, the scalar code is a bottleneck to vector scalability. When it comes to power consumption, the cost of faster last-level caches is negligible compared to the baseline.

The best EDP^2 is achieved running a fast LLC and fast ALUs, but with a register size of 1024 bits. However, from a power perspective, the best efficiency is achieved limiting the register size to 512 bits at 2Ghz with a fast LLC.

Quadratic. Figure 8-left depicts performance improvements for Quadratic. This application shows very limited scalability with vector register size. When running with 512-bit registers, a 2.1x is achieved from the theoretical 4x. Wider vectors generate the same results. A fast last-level cache does not provide any performance benefit, but a faster ALU does. This means that scalar operations are slowing down the application. But a fast ALU is really costly, doubling the energy requirements of the baseline.

In Quadratic, the best results with regards to EDP^2 are achieved using 512/1024-bit vector registers, slow last-level cache and fast ALUs. However, from a power perspective, the optimal MFlops per watt are obtained running 512-bit registers with slow LLC and ALUs running at 2Ghz.

Streamcluster. Figure 9-left shows the performance scalability of Streamcluster. Streamcluster is a slightly memory-bound application, which is reflected in the benefits when working with a faster last-level cache. It is also important

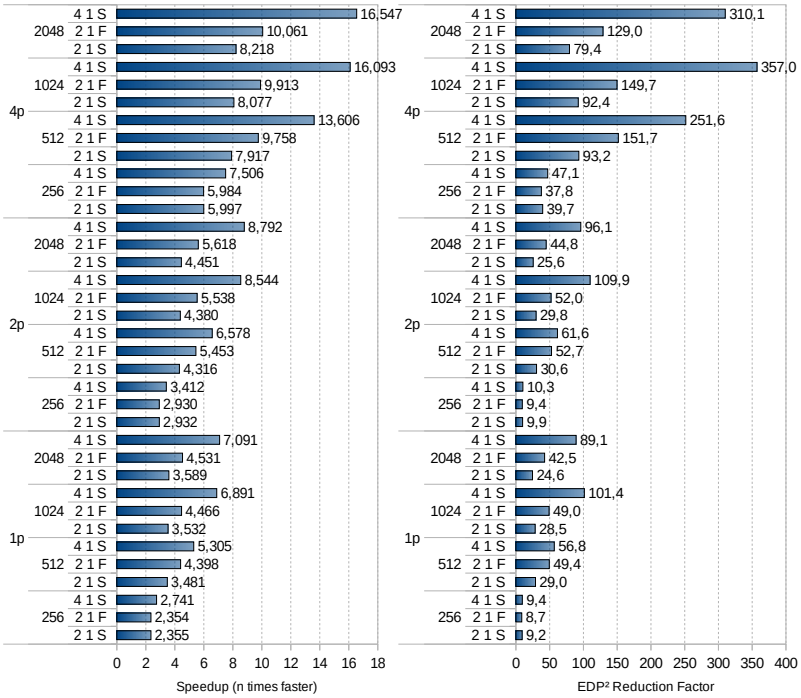

Fig. 6 Speedup (left) and EDP^2 improvement (right) for Matmul (more is better) normalized to 1 core with 128-bit registers running at 1Ghz with slow LLC. Y axis shows different combinations of core count, reg. size, and frequencies (scalar, vector and LLC).

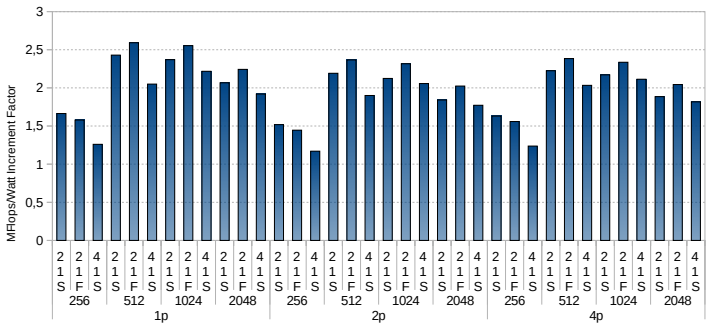

Fig. 7 MFlops per Watt improvement (more is better) for Matmul with different combinations of core count, reg. size, and frequencies (scalar, vector and LLC).

to note that having more than one core improves scalability with vector register size. This happens because of the additional cache capacity as we increase the number of cores. Having ALUs running at faster frequencies prevents the scalar computations from becoming the bottleneck, as the latency of vector memory requests can be hidden better. Figures 9-right and 10 show the energy scalability of this application. Faster ALUs are really costly, whereas a

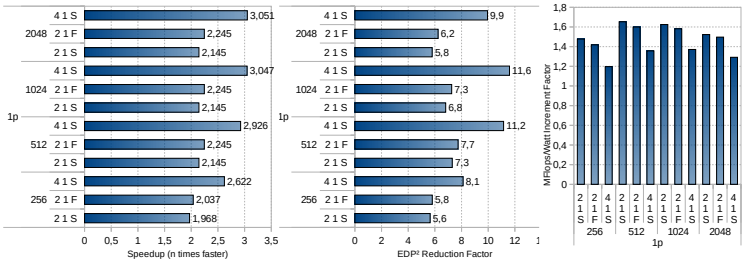

Fig. 8 Speedup (left) EDP^2 improvement (center) and MFlops per Watt improvement (right) for Quadratic (more is better) normalized to 1 core with 128-bit registers running at 1Ghz with slow LLC. Y axis shows different combinations of reg. size, and frequencies (scalar, vector and LLC).

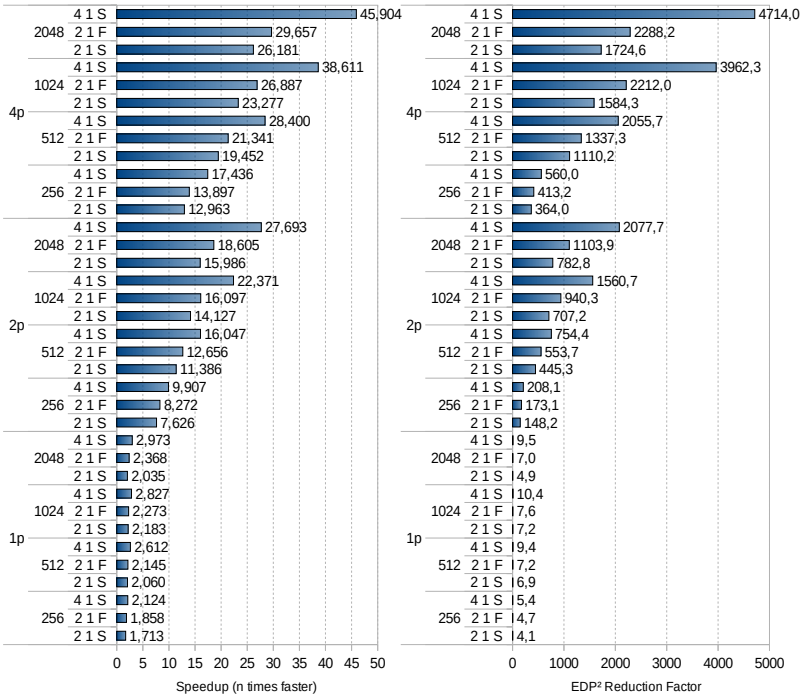

Fig. 9 Speedup (left) and EDP^2 improvement (right) for Streamcluster (more is better) normalized to 1 core with 128-bit registers running at 1Ghz with slow LLC. Y axis shows different combinations of core count, reg. size, and frequencies (scalar, vector and LLC).

faster last-level cache is not that much. However, the power consumption on multicore scenarios remains stable along the different configurations.

In Streamcluster, the best EDP^2 is obtained when using four threads and 2048-bit vector registers with fast ALUs. However, when focusing on pure energy efficiency, the best Mflops per watt are obtained running on two cores with 2Ghz ALUs and a fast LLC.

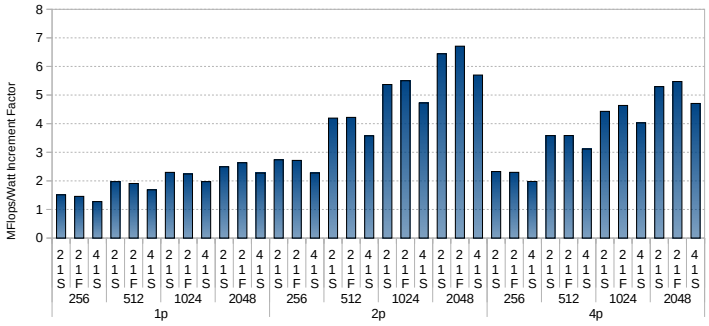

Fig. 10 MFlops per Watt improvement (more is better) for Streamcluster with different combinations of core count, reg. size, and frequencies (scalar, vector and LLC).

Swaptions. When analyzing the performance scalability of Swaptions (Figure 11-left) we see a sublinear benefit from register size. In fact, vector length stops being important for performance at 512 bits. Swaptions is a CPU-bound application, as it does not benefit from a faster last-level cache. Faster ALUs accelerate the scalar execution, which was slowing down the application execution time. Running on several cores provides really good outcomes. However, a four-threaded scenario does not get much benefit from longer vectors (maybe input related). Figures 11-right and 12 show the energy scalability of this application. Power per core on multicore scenarios remains stable along the different configurations.

The best EDP^2 for this application is obtained when running: four threads, 1024-bit vector registers, slow last-level cache and fast scalar units. However, fast scalar units should only be considered if there are no power constraints, as they have a huge effect on MFlops per watt for all core configurations.

Vreduction. Figure 13-left shows very limited scalability with vector register size, that disappears after 512-bit registers. This is probably because our memory subsystem is limited to 512-bit memory movements, doing several memory requests when dealing with wider registers. Faster last-level caches do not provide much benefit, supporting the assumption that we are dealing with a bandwidth issue. Faster ALUs are of help in this benchmark, but is not as critical as for other applications.

Figure 13-right shows that the best EDP^2 is achieved running four cores, 512-bit registers and fast ALUs. MFlops per watt vary little with core count, and peak at 512-bit registers with slow LLC and ALUs running at 2Ghz (Figure 14).

Evaluation Summary Previous results demonstrate the need for a runtime reconfiguration strategy that adapts vector register length at runtime. There can be two options depending on whether the objective is to obtain best performance or a trade-off between performance and energy. In the case of compute-bound applications, wider vector registers provide the best performance but it

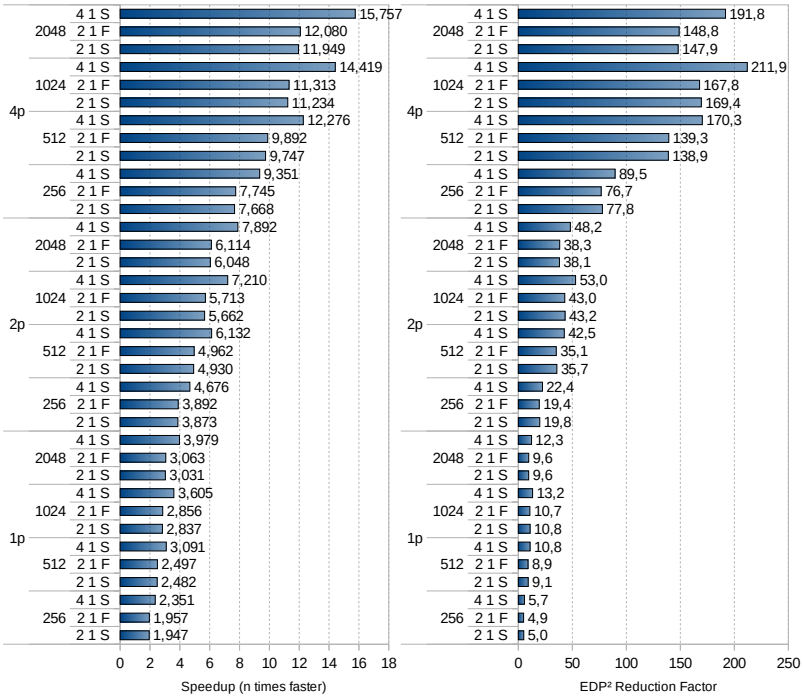

Fig. 11 Speedup (left) and EDP^2 improvement (right) for Swaptions (more is better) normalized to 1 core with 128-bit registers running at 1Ghz with slow LLC. Y axis shows different combinations of core count, reg. size, and frequencies (scalar, vector and LLC).

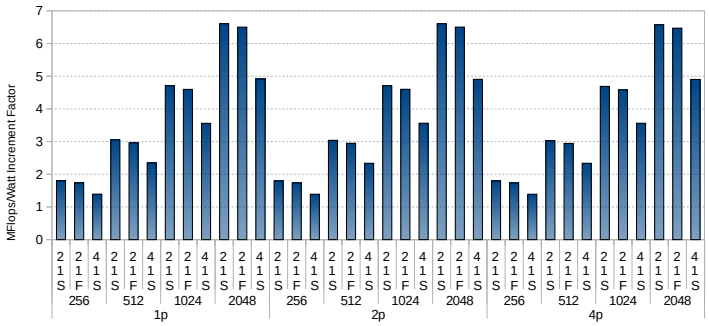

Fig. 12 MFlops per Watt improvement (more is better) for Swaptions with different combinations of core count, reg. size, and frequencies (scalar, vector and LLC).

is also important to make scalar units operate at higher frequency since they become the bottleneck. Faster last-level caches do not improve performance since the memory is not a limitation in this type of applications, so a slower one leads to a reduction in the energy consumption. Vector register sizes should only be reduced if the energy needs to be decreased but performance will be lineary affected. In the case of memory-bound applications, vector registers beyond 512 bits do not translate into significant performance benefits. This is

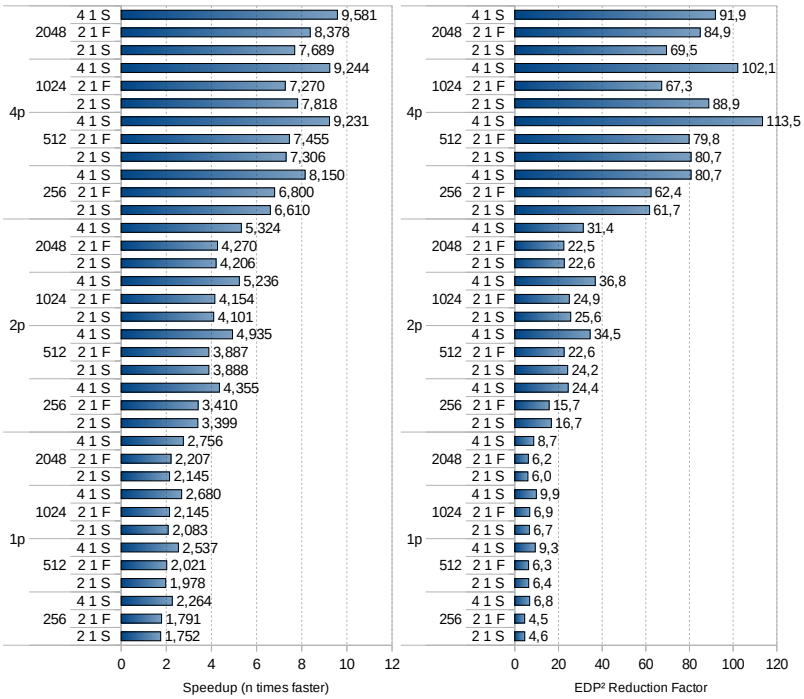

Fig. 13 Speedup (left) and EDP^2 improvement (right) for Vreduction (more is better) normalized to 1 core with 128-bit registers running at 1Ghz with slow LLC. Y axis shows different combinations of core count, reg. size, and frequencies (scalar, vector and LLC).

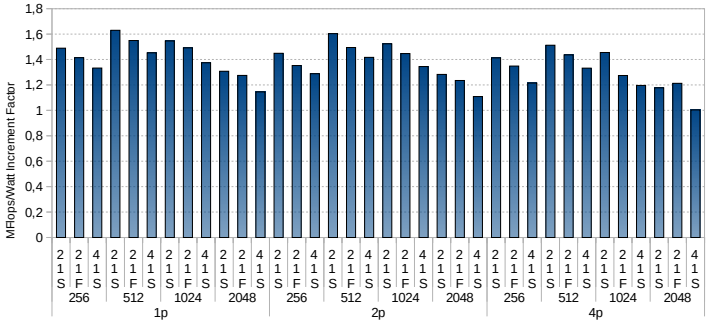

Fig. 14 MFlops per Watt improvement (more is better) for Vreduction with different combinations of core count, reg. size, and frequencies (scalar, vector and LLC).

a factor which depends on the number of compute instructions per memory ones. Faster last-level caches should be considered to obtain better speedups because memory is a limiting element. Scalar units also become the bottleneck the wider the vector registers, so their frequency should be scaled accordingly.

5 Conclusions

This paper shows a detailed analysis of different resource optimization strategies for vector architectures. In particular it assumes different voltage and frequency islands for the last level cache (LLC), the scalar arithmetic units (ALUs) and the vector processing units (VPUs).

Scalability with core count and vector register size is shown for performance, EDP^2 (as an energy metric that emphasizes in performance) and MFlops per watt (as an energy metric that emphasizes in low power, and used in the Green500 classification).

Results expose the need to break down voltage and frequency domains for LLC. Almost all benchmarks show significant benefits in performance and EDP^2 when using fast ALUs, especially for wide vector registers, since the scalar code becomes critical for performance (Amdahl's law). Energy efficiency can be further improved on CPU-bound benchmarks by reducing the operating frequency of the LLC, since latency is not critical, but bandwidth and memory-level parallelism are.

Results also show the need for a runtime reconfiguration strategy that adapts vector register length at runtime. This is feasible in architectures such as ARM's SVE. Not all benchmarks show perfect scalability with register size, and we also see a relationship between core count and vector scalability. We aim at developing a dynamic reconfiguration mechanism that adapts both frequency and vector register length based on performance counter information as future work.

References

1. The Green 500 (2018). URL https://www.top500.org/green500/
2. Albright, R.K.: Optimizing Performance/watt of Embedded SIMD Multiprocessors Through a Priori Application Guided Power Scheduling. Oregon State University (2012)
3. AMD: 3DNow! Technology Manual. Motorola (2000)
4. ARM NEON Technology
5. Asanoviċ, K.: Vector Microprocessors. Ph.D. thesis (1998)
6. Barnes, G.H., Brown, R.M., Kato, M., Kuck, D.J., Slotnick, D.L., Stokes, R.A.: The ILLIAC IV Computer. IEEE Transactions on Computers **C-17**(8), 746–757 (1968)
7. Binkert, N., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M.D., Wood, D.A., Beckmann, B., Black, G., Reinhardt, S.K., Saidi, A., Basu, A., Hestness, J., Hower, D.R., Krishna, T.: The gem5 simulator. ACM SIGARCH Computer Architecture News **39**(2), 1 (2011)
8. Butenhof, D.R.: Programming with POSIX Threads. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1997)
9. Casas, M., Moreto, M., Alvarez, L., Castillo, E., Chasapis, D., Hayes, T., Jaulmes, L., Palomar, O., Unsal, O., Cristal, A., Ayguade, E., Labarta, J., Valero, M.: Runtime-Aware Architectures. pp. 16–27 (2015)
10. Cebrian, J.M., Jahre, M., Natvig, L.: ParVec: Vectorizing the PARSEC Benchmark Suite. Computing pp. 1077–1100 (2015)
11. Cebrián, J.M., Natvig, L., Meyer, J.C.: Performance and energy impact of parallelization and vectorization techniques in modern microprocessors. Computing **96**(12), 1179–1193 (2014)
12. Chapman, B.: The multicore programming challenge. Advanced Parallel Processing Technologies p. 4847 (2007)

13. Cogez, P., Graef, M., Huizing, B.: ITRS Executive Summary 2011. International technology roadmap for semiconductors (2011)
14. Cray Research, I.: Cray X-MP Series Model 48 Mainframe Reference Manual (1984)
15. Dennard, R., Gaensslen, F., Rideout, V., Bassous, E., LeBlanc, A.: Design of ion-implanted MOSFET's with very small physical dimensions. IEEE Journal of Solid-State Circuits **9**(5), 256–268 (1974)
16. Espasa, R.: Advanced vector architectures. Ph.D. thesis, Universitat Politècnica de Catalunya (1997)
17. Espasa, R., Valero, M., Smith, J.E.: Vector Architectures : Past , Present and Future. Proceeding ICS '98 Proceedings of the 12th international conference on Supercomputing pp. 425–432 (1998)
18. Fuller, S.: Motorola AltiVec Technology. Motorola (1998)
19. Haley, A.: Deuce: a high-speed general-purpose computer. Proceedings of the IEEE - Part B: Radio and Electronic Engineering **103**(2S), 165–173 (1956)
20. Hennessy, J.L., Patterson, D.A.: Computer Architecture, Sixth Edition: A Quantitative Approach, 6th edn. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2017)
21. Hu, Z., Buyuktosunoglu, A., Srinivasan, V., Zyuban, V., Jacobson, H., Bose, P.: Microarchitectural techniques for power gating of execution units. In: Proceedings of the 2004 international symposium on Low power electronics and design - ISLPED '04, p. 32. ACM Press, New York, New York, USA (2004)
22. Inoue, H.: How SIMD width affects energy efficiency: A case study on sorting. In: 2016 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS XIX), pp. 1–3. IEEE (2016)
23. Intel Corporation: Intel 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture. (2012)
24. Intel Corporation: Intel 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference. (2015)
25. Jesshope, R.W.H., R., C.: Parallel Computers Two: Architecture, Programming and Algorithms. IOP Publishing Ltd., Bristol, UK, 2nd edition (1988)
26. Jimborean, A., Koukos, K., Spiliopoulos, V., Black-Schaffer, D., Kaxiras, S.: Fix the code. Don't tweak the hardware: A new compiler approach to Voltage-Frequency scaling
27. Kaxiras, S., Martonosi, M.: Computer Architecture Techniques for Power-Efficiency. Synthesis Lectures on Computer Architecture **3**(1), 1–207 (2008)
28. Koukos, K., Black-Schaffer, D., Spiliopoulos, V., Kaxiras, S.: Towards More Efficient Execution: A Decoupled Access-Execute Approach
29. Lee, Y., Avizienis, R., Bishara, A., Xia, R., Lockhart, D., Batten, C.: Exploring the Tradeoffs between Programmability and Efficiency in Data-Parallel Accelerators pp. 129–140 (2011)
30. Lemuet, C., Sampson, J., Francois, J., Jouppi, N.: The Potential Energy Efficiency of Vector Acceleration. In: ACM/IEEE SC 2006 Conference (SC'06), pp. 1–1. IEEE (2006)
31. Li, H., Bhunia, S., Chen, Y., Vijaykumar, T.N., Roy, K.: Deterministic Clock Gating for Microprocessor Power Reduction
32. Li, S., Ahn, J.H., Strong, R.D., Brockman, J.B., Tullsen, D.M., Jouppi, N.P.: McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In: Proceedings of the 42nd Annual ACM/IEEE International Symposium on Microarchitecture (MICRO), pp. 469–480 (2009)
33. Majzoub, S.: Voltage island design in multi-core SIMD processors. In: 2010 5th International Design and Test Workshop, pp. 18–23. IEEE (2010)
34. Mudge, T.: Power: A first-class architectural design constraint. Computer **34**(4), 52–58 (2001)
35. NEC: Vector Supercomputer SX Series: SX-Aurora TSUBASA (2017)
36. Qiang Wu, Martonosi, M., Clark, D., Reddi, V., Connors, D., Youfeng Wu, Jin Lee, Brooks, D.: A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance. In: 38th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'05), pp. 271–282. IEEE
37. Russell, R.M.: The CRAY-1 Computer System. Proc. Comm. ACM Computer Proc. WJCC Comm. ACM McCarthy, J. Time Sharing Computer Systems Pt. I, AFIPS Press N.J. N.J. N.J **36**(12), 657–675 (1971)

38. Russell, R.M.: The CRAY-1 computer system. Commun. ACM **21**(1), 63–72 (1978)
39. Satish, N., Kim, C., Chhugani, J., Saito, H., Krishnaiyer, R., Smelyanskiy, M., Girkar, M., Dubey, P.: Can Traditional Programming Bridge the Ninja Performance Gap for Parallel Computing Applications? In: Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA), pp. 440–451 (2012)
40. Sodani, A.: Knights landing (KNL): 2nd Generation Intel Xeon Phi processor. In: Hot Chips (2015)
41. Stephens, N., Biles, S., Boettcher, M., Eapen, J., Eyole, M., Gabrielli, G., Horsnell, M., Magklis, G., Martinez, A., Premillieu, N., Reid, A., Rico, A., Walker, P.: The ARM Scalable Vector Extension. IEEE Micro **37**(2), 26–39 (2017)
42. Transactions, I., Technology, D., Technology, D., Techniques, J.C.m.P.: The Reliability Wall for Exascale Supercomputing (August 2015) (2012)
43. Usami, K., Goto, Y., Matsunaga, K., Koyama, S., Ikebuchi, D., Amano, H., Nakamura, H.: On-chip detection methodology for break-even time of power gated function units. In: IEEE/ACM International Symposium on Low Power Electronics and Design, pp. 241–246. IEEE (2011)
44. Villa, L., Espasa, R., Valero, M.: Effective usage of vector registers in advanced vector architectures. In: Proceedings 1997 International Conference on Parallel Architectures and Compilation Techniques, pp. 250–260. IEEE Comput. Soc
45. Watson, W.J.: The TI ASC: A Highly Modular and Flexible Super Computer Architecture. In: Proceedings of the December 5-7, 1972, Fall Joint Computer Conference, Part I (AFIPS), pp. 221–228 (1972)
46. Wilkinson, J.H.: The {Pilot ACE}. In: Automatic Digital Computation, pp. 5–14 (1954)
47. Wulf, W.A., McKee, S.A.: Hitting the memory wall. ACM SIGARCH Computer Architecture News **23**(1), 20–24 (1995)
48. Xi, S., Jacobson, H., Bose, P., Wei, G.Y., Brooks, D.: Quantifying sources of error in McPAT and potential impacts on architectural studies. In: International Symposium on High Performance Computer Architecture (HPCA), pp. 577–589 (2015)
49. Yoshida, T.: Introduction of Fujitsu's HPC processor for the Post-K computer. In: Hot Chips (2016)