Optimized Hardware for Suboptimal Software: The Case for SIMD-aware Benchmarks

Juan M. Cebrián, Magnus Jahre and Lasse Natvig
Dept. of Computer and Information Science (IDI)
NTNU Trondheim, NO-7491, Norway.
E-mail: {juanmc,jahre,lasse}@idi.ntnu.no

***Abstract*—Evaluation of new architectural proposals against real applications is a necessary step in academic research. However, providing benchmarks that keep up with new architectural changes has become a real challenge. If benchmarks don't cover the most common architectural features, architects may end up under/over estimating the impact of their contributions.**

In this work, we extend the PARSEC benchmark suite with SIMD capabilities to provide an enhanced evaluation framework for new academic/industry proposals. We then perform a detailed energy and performance evaluation of this commonly used application set on different platforms (Intel® and ARM® processors). Our results show how SIMD code alters scalability, energy efficiency and hardware requirements. Performance and energy efficiency improvements depend greatly on the fraction of code that we can actually vectorize (up to 50x). Our enhancements are based in a custom built wrapper library compatible with SSE, AVX and NEON to facilitate general vectorization. We aim to distribute the source code to reinforce the evaluation process of new proposals for computing systems.

***Keywords*-Performance Analysis, Benchmarking, Energy Efficiency, MSRs, Vectorization, SIMD.**

I. Introduction

Historically, the computer architecture community has been remarkably successful at translating the increasing transistor supply into a performance improvement for the end user. This is in part due to the quantitative evaluation methodologies employed to evaluate novel techniques [1]. In addition, the end of Dennard scaling [2] is pushing energy efficiency to replace performance as the main design goal for microprocessors across all market segments. Due to the overwhelming complexity of modern processors, simulation and evaluation on real hardware are the most popular quantitative research methods [3]. Benchmarks are the cornerstone of these methodologies. The key idea is to use a selection of representative applications to analyze the design characteristics of new hardware or software proposals [4]. It is usually very challenging to select representative applications that will cover all design aspects of the processor since their requirements tend to change rapidly.

In this green computing era, both current and future processor designs embrace parallelism and heterogeneity to meet the computation requirements of an application without wasting resources [5]. The use of specialized cores to run frequently used algorithms (e.g., media decoders, hashing,

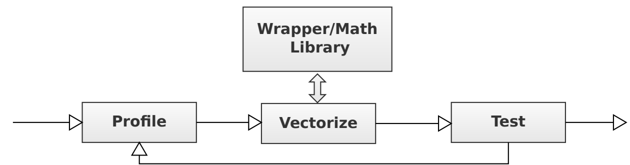


Figure 1: Per-benchmark vectorization process workflow.

encryption) has become common in modern processors, especially in the embedded domain. Last, but not least, vectorization or single instruction multiple data (SIMD) capabilities are a key component to maximize performance per watt in highly computational intensive applications. Vectorization capabilities are available in processors designed for different market segments, from embedded devices (ARM NEON), desktop and servers (SSE and AVX) to GPUs and accelerators (e.g., Xeon Phi™ or NVIDIA® K20).

There are two key factors that force the constant update of benchmarking environments: a) socioeconomic demands that force the appearance of new markets, applications, algorithms and methods and b) the feedback loop that comes from the architectural changes requires benchmarks to be updated to catch up with new features. This second factor is the main motivation for this paper. Our evaluation shows that SIMD-aware applications exhibit different behavior regarding key architectural aspects, like cache and renaming logic usage or memory bandwidth requirements. Unfortunately, it is hard for the computer architecture community to propose novel techniques that leverage these insights since SIMD-aware benchmarks are not in widespread use (e.g., PARSEC, SPLASH-2, SPEC are not fully SIMD-enabled). To make things worse, SIMD issue width has increased at the same rate as cores have been added to the chip and this trend is expected to continue [1]. Without SIMD-aware benchmarks, architects are practically blind to the effects of this trend.

To alleviate this problem, we have vectorized 8 of 12 of the benchmarks of the popular PARSEC benchmark suite. Support includes SIMD-extensions for both x86 and ARM architectures (SSE, AVX and NEON). To facilitate rapid vectorization we developed a wrapper library that maps functions to the aforementioned SIMD technologies, but can

be easily extended to support others. Figure 1 shows the procedure we used to vectorize the benchmarks. First, we profile each benchmark to detect regions of the code that are suitable for vectorization. Second, we vectorize these regions using our wrapper and math libraries extending them if necessary. In this step, the benchmarks are re-written using intrinsic-like[1] macros that will be translated to real intrinsics if supported by the architecture or to *inline* functions that implement that specific functionality. Finally, we verified correctness and gathered energy and performance results. This is a textbook optimization sequence [6]. Automatic vectorization techniques were tested on unmodified code using both GCC and ICC. Automatic vectorization resulted in marginal performance variations (± 2%). The reason is that PARSEC's code is currently not vectorization friendly. We chose an intrinsic-based manual vectorization rather than novel auto-vectorization techniques (e.g. [7] [8]) to provide users with better control over the generated code.

Armed with the vectorized benchmark suite, we perform a thorough performance and energy evaluation on Intel Sandy and Ivy Bridge and ARM Cortex-A15 architectures. We observe that the vectorized benchmarks fall into three categories: scalable, resource-limited and code/input limited. For the *scalable benchmarks*, performance increases proportionally to the SIMD-width. Hardware mechanisms that modify resource-sharing and operation latency may have a negative effect on these benchmarks due to the increased resource utilization on the SIMD implementation. In addition, SIMD-scalable applications may affect the optimal number of active cores per task. *Resource-limited* benchmarks exhibit a speedup for certain SIMD-widths but then either stop scaling (input size related) or hit an architectural bottleneck. This group of applications can lead the research community to propose architectural modifications to solve the aforementioned bottlenecks. Finally, *code/input limited* benchmarks either have a limited supply of vectorizable work (nature/organization of the input, not size related) or the algorithms/implementation are vectorization unfriendly. The result is that the SIMD-aware implementation performs similarly to the non-vectorized version. Some applications are simply not suited for vectorization, as are the sequential phases for parallelization.

II. Background

Vector supercomputers appeared in the early 1970s and remained unmatched in terms of performance up until late 2004 (Earth Simulator). However, vector processors were specially designed for supercomputers resulting in high prices when compared with regular commercial processors. The first widely-deployed desktop SIMD system dates from 1996, with Intel's Pentium™ processors with MMX™ extensions using 64 bit registers. SSE (Streaming SIMD Extensions) extended MMX to work with 128 bit registers while Advanced Vector Extensions (AVX) extends SSE to 256 bit registers. Both SSE and AVX can be programmed using intrinsics, assembly, or automatic compiler vectorization. Intel claims that use of AVX improves energy efficiency [9], and we see it clearly quantified in our results. Other technologies, such as AMD®'s 3DNow!™ or ARM's NEON, follow a similar evolution. Hennessy *et al.* [1] argue that, under space constraints, the performance and energy improvements of SIMD architectures is greater than adding more cores.

SIMD instructions not only reduce cache and pipeline pressure (less instructions need to be fetched, decoded and issued), but also have a potential speedup based on the width of the SIMD registers. In addition, SIMD instructions have a huge impact on energy efficiency on Intel processors. Cebrian *et al.* [10] showed how applications running on Intel processors dissipate roughly the same average power independently of how many bits were used from the SIMD registers. This translates into potential energy savings for vectorization-friendly applications.

There are, however, important disadvantages when using SIMD functionality. SIMD architectures increase cache bandwidth requirements, since the core needs SIMD width times more data to be moved to the registers. SIMD relies on large register files that increase power dissipation and chip area and usually requires low level programming (intrinsics). Finally we should keep in mind that not all applications are suitable for vectorization.

III. Vectorizing PARSEC

Starting from the official PARSEC 3.0b benchmark suite, we profile each application to disclose per-function runtime. This step will guide us in the manual vectorization process using well known techniques [6], focusing our efforts on those functions that consume most of the processor resources. Due to space limitations we have removed specific implementation details and per-benchmark profiling information, but it can be provided on demand. For further details on PARSEC benchmark specifics please refer to [11].

Table I provides a summary of the key characteristics of the vectorized PARSEC benchmarks including:

- Support for SSE, AVX and NEON instructions.
- Application hotspots where to focus vectorization.
- Data structure organization: AoS (Array of Structures) or SoA (Structure of Arrays).
- Divergent branches in the code that limit vectorization.
- Only works with inputs divisible by SIMD width.
- Direct translation to SIMD (DT) or requires major code/algorithm changes (MCC).
- Group: scalable (S), resource-limited (RL) and code/input (CI) limited.

[1]Functions available for use in a given programming language whose implementation is handled specially by the compiler.

Program	Domain	Vect.			Hotspots	Data Struc.	Div. Branch.	SIMD Width	Changes	Group
		SSE	AVX	NEON						
blackscholes	Financial	x	x	x	yes	SoA	yes	yes	DT	S
canneal	Engineering	x	x	x	yes	AoS	no	yes	MCC	RL
fluidanimate	Animation	x	x	x	yes	AoS	yes	no	MCC	CI
raytrace	Rendering	x	-	-	yes	AoS	-	-	-	-
streamcluster	Data Mining	x	x	x	yes	SoC	no	yes	DT	RL
swaptions	Financial	x	x	x	no	SoC	yes	yes	MCC	CI
vips	Media Proc.	x	x	x	no	both	no	filter dep.	MCC	RL/CI
x264	Media Proc.	x	x	x	no	both	-	-	-	RL/CI

Table I: Evaluated PARSEC benchmark characteristics.

A. SIMD Wrapper and Math Libraries

We have developed a SIMD wrapper library to help in the vectorization process. The wrapper library is basically a set of macros that translate to either real intrinsics or *inline* functions that implement a specific functionality. This is only necessary for those instructions that don't have a hardware implementation on a specific architecture. For example, ARM NEON does not support floating point division, thus the division intrinsic is implemented as a reciprocal multiply plus some Newton-Raphson steps to refine the estimate.

```
1 #define _MM_ALIGNMENT 16
2 #define SIMD_WIDTH 4
3 #define _MM_ABS _mm_abs_ps
4 #define _MM_CMPLT _mm_cmplt_ps
5 #define _MM_TYPE __m128
6 ...
7 __attribute__((aligned (16))) static const int absmask[] = {0x7fffffff, 0
      x7fffffff, 0x7fffffff, 0x7fffffff};
8 #define _mm_abs_ps(x) _mm_and_ps((x), *(const __m128*)absmask)
```

Code 1. SIMD wrapper library example (SSE-float).

Code 1 shows a small example of the structure of this wrapper library for SSE-floats. If a benchmark requires to calculate the absolute value of a variable using SIMD it will call the macro "_MM_ABS((_MM_TYPE)variable)". If we have specified that the code should be compiled for SSE optimizations using single precision floating point, macros will be translated to intrinsics based on the wrappers shown in Code 1 as "_mm_abs_ps((__m128)variable)". Since Intel processors do not have hardware support for absolute value, "_mm_abs_ps" is implemented manually by means of an AND function and a mask (line 8). If we select a different target for this example, such as ARM NEON, the wrapper library would translate the code as "vabsq_f32((float32x4_t)variable)" (supported in hardware).

Common math functions have also been implemented using different SIMD instruction sets (SSE, AVX and NEON) into a small math library. The math library is based on [12] extended in [13] to include AVX functionality and then extended by ourselves to include support for double precision floating point operations and NEON.

B. Vectorization Process

The blackscholes and streamcluster benchmarks are quite straightforward to vectorize, basically replacing the code by the equivalent operations based on our wrapper and math libraries. Swaptions required deep code modifications, achieving similar speedup results as commercial libraries [14]. However, the vectorized code is limited by the data dependencies in the algorithm itself. Canneal can be directly translated into SIMD code, however the speedup achieved is marginal due to high cache miss-rate and low amount of vector instructions. After checking the code we noticed that the current implementation does not use any kind of cache blocking mechanism. An easy solution to both problems is to modify the code to make use of this well known optimization technique. By using blocking the scalar code now runs twice as fast as the original version obtaining a similar (better) quality solution. In addition, the benchmark can be easily extended to support SIMD optimizations using cache blocks as indexes. The x264 benchmark is still under development, so in order to get proper SSE, NEON and AVX code we only upgraded to the latest version (0.135.2345).

Vips "benchmarking" mode only uses a subset of filters from the whole Vips application. We vectorized the most time-consuming of these filters (convolution, linear transformation and interpolation) by using explicit SSE/NEON/AVX implementation. Our wrapper library only supports one translation mode, that is, all code will either be translated to SSE, AVX or NEON, but not to a combination of them. We cannot rely on our wrapper library for this benchmark since not all filters can make use of eight-float registers, so we have to combine SSE and AVX code. Fluidanimate has all the ingredients to be vectorization unfriendly including divergent branches, small number of arithmetic computations for each loaded data, variable number of loop indexes (usually non-divisible by SIMD width), high output variance due to floating point rounding errors and small input size. Despite all that we have managed to extract some performance from vectorization.

The raytrace benchmark is already implemented using SSE instructions. PARSEC's source of Intel's raytrace is configured to "emulate" SSE instructions. Emulation is achieved using a wrapper library that translates SSE intrinsics into a set of non-SIMD instructions after solving some memory issues[2]. The raytrace benchmark cannot be placed anywhere in our classification since the computational requirements of the scalar version are those of a

[2]http://software.intel.com/en-us/forums/topic/278198

Cache	Size	Sharing	Ways of associativity	Line size	Latency (cycles)
Level 1 Instruction	32KB	Private	8	64B	4
Level 1 Data	32KB	Private	8	64B	4
Level 2	256KB	Private	8	64B	12
Level 3	8MB	Shared	16	64B	24
Main Memory	16GB	Shared	-	-	DDR3@1333 (0.8ns)

Table II: Memory hierarchy information for IB and SB.

Cores	Processor type	L1 Cache	L2 Cache	Memory
2	Cortex-A15@1.7Ghz	32KB+32KB	1MB Shared	2GB LP-DDR3@800Mhz

Table III: Specifications for the Arndale development board.

SSE emulator, completely different from those of the real SSE version. Finally, the benchmarks "bodytrack", "dedup", "ferret" and "freqmine" are still work in progress. Neither of them presents clear hotspots on which we can focus our vectorization efforts. This will most likely place them in the code/input limited group, making their presence non-critical to the conclusions of this work.

IV. Methodology

This section describes details of our evaluation infrastructure. We have analyzed the vectorized benchmarks on an Intel Core™ i7-3770K Ivy Bridge (IB), an i7-2600K Sandy Bridge (SB) and an Exynos™ 5 Dual (Samsung®) processor. Ivy and Sandy processors have four physical cores with eight threads and their memory hierarchy is described in Table II. Their main difference is that IB is built on a 22-nm process and uses tri-gate transistors (reduced leakage power). Both offer support for SSE4.2 and AVX instructions. Exynos 5 Dual has two Cortex-A15 processors with NEON units (128 bit registers, single precision FP) described in III.

The analyzed applications are based on PARSEC 3.0b and compiled using GCC 4.7 with -O2 flag on a Ubuntu 12.04.3 distribution with Kernel 3.6.9 for the Intel systems, and Linaro 13.02 distribution with Kernel 3.8.0 for the ARM board. Automatic vectorization was tested using both GCC and Intel ICC compilers with performance variations of $\pm$ 2%. We disabled automatic vectorization whenever using manual vectorization. The evaluated ARM processor is pre-assembled on Insignal® Arndale development board. Executions were performed sequentially 10 times, using the native input sizes of the PARSEC benchmark suite. The results are reproducible and stable, with a relative standard deviation less than 1%. We discovered a 3 to 10% variation in the energy measurements between a "cold" (30°C) processor and a "warm" ($50 - 65$°C) processor. Leakage power depends exponentially on temperature, reducing the energy efficiency of the processor as temperature increases. Cold runs are thus discarded to minimize variability in the energy measurements. Systems run with minimal console configurations, to reduce OS influence on the results.

A. Measuring Power

Until recently, empirical power estimates were based on performance counters provided by the processor, but the second generation of Core i5 and i7 extends the MSRs[3] with energy information, enabling applications to examine their energy efficiency without using external devices. This instrumentation introduces an overhead proportional to the sampling rate. Since we sample every minute to prevent counter overflow the overhead is minimal.

We use the RAPL[4] MSR interface to retrieve per-package (socket) energy information. The latest versions of PAPI libraries (5.0 or higher) [15] include support for reading Intel's energy registers using the RAPL component. In our setup we are interested in three registers, PKG, PP0 and PP1. PKG stores the whole package energy information, while PP0 (power plane 0) stores core energy information and PP1 stores GPU energy information. Additional energy components will be added as other manufacturers incorporate such energy information. There is presently no equivalent to Intel's energy MSRs on our ARM platforms, so our evaluation for these systems is currently limited to full system power ("at-the-wall" power). We use a Yokogawa's WT210 device to measure it.

Comparing systems based on energy consumption alone can motivate the use of simple cores with low frequency. *Energy-Delay Product* (EDP) metric puts greater emphasis on performance. However, as Sazeides *et al.* discuss in [16], EDP can be misleading when comparing platforms designed for different market segments. Therefore, we will show performance and energy metrics for comparisons between platforms. Other units, such as EDP or ED^2P can be calculated from our measurements.

V. Experimental Results

This section presents our energy and performance evaluation of the selected SIMD-enabled PARSEC benchmarks. We will show energy information for all benchmarks in all evaluated platforms. Ivy Bridge and Sandy Bridge processors share the same architecture but are built with different technologies, rendering the runtime results very similar, and thus we will only show one common figure for them. We cannot provide detailed performance counter information for all benchmarks due to space limitations. We present results according to the classification: scalable (S), resource-limited (RL) and code/input limited (CI), and show detailed information for one benchmark from each group. Scalable benchmarks scale linearly with SIMD width. Resource limited scale reasonably well for SSE-NEON (2-3x per thread) but not for AVX. Code/Input limited do not get much performance improvement from the SIMD (less than 2x performance improvement), due to divergent branches,

[3]Model Specific Registers
[4]Running Average Power Limit

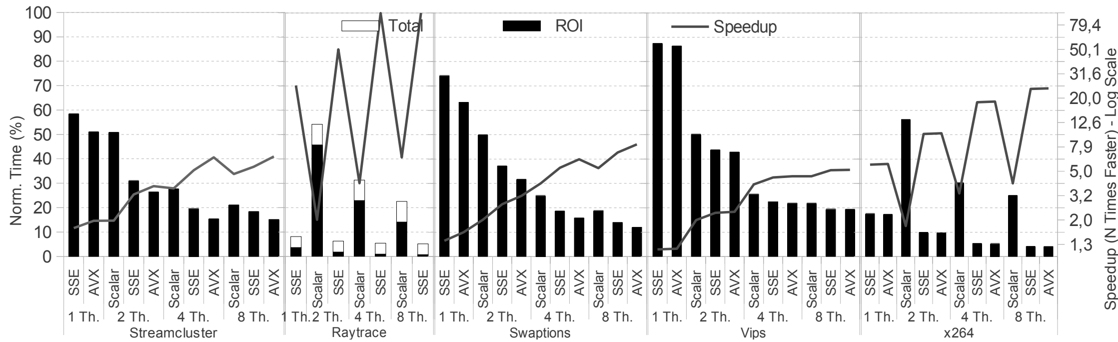


Figure 2: Normalized runtime (to 1 thread-scalar) and speedup (secondary axis) for IB.

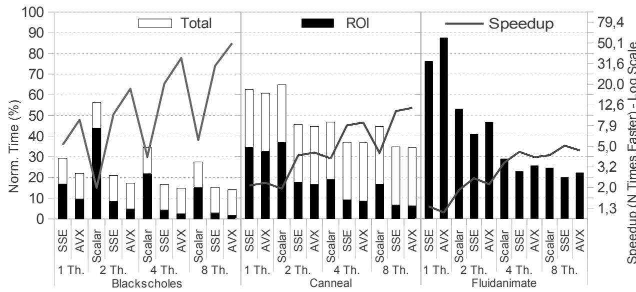


Figure 3: Normalized runtime (to 1 thread-scalar) and speedup (secondary axis) for IB.

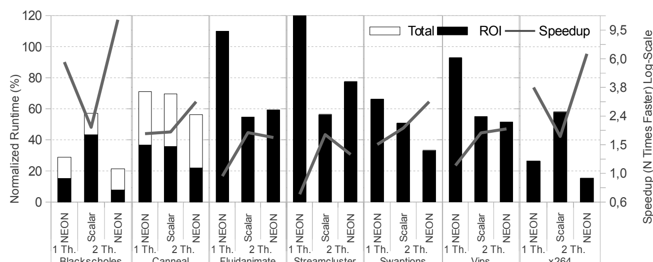


Figure 4: Normalized runtime (to 1 thread-scalar) and speedup (secondary axis) for Arndale.

type conversions, AoS to SoA conversions and other SIMD requirements not satisfied by the application.

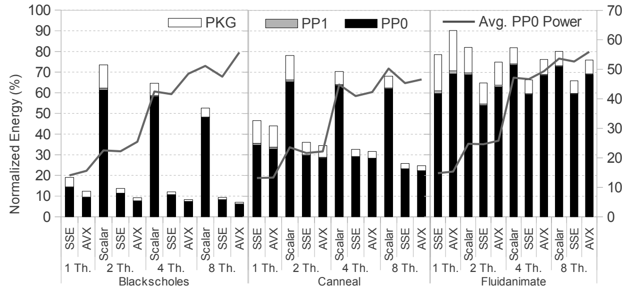
A. Performance Results

Figures 2 and 3 show runtime distribution (Total and ROI[5]) for the analyzed Ivy and Sandy Bridge platforms. Blackscholes and x264 using SSE/AVX extensions behave even better than their "equivalent" implementation running 4/8 threads, reaching 50x and 24x speedups respectively when running on both AVX and 8 threads. Speedup achieved by SIMD is consistent across thread counts. Raytrace SSE emulation slows the application by almost 25 times per thread, giving a complete different behavior. For resource-limited applications speedups of SSE and AVX are around 2-3x per thread, while for code-limited applications the small amount if SIMD instructions we can generate barely improves overall performance by 10%. Results for the Cortex-A15 processor (Figure 4) show similar trends. However, the ARM processor provides less resources than the analyzed Intel processors, as seen in the results. Streamcluster is heavily constrained by the memory subsystem of the Cortex-A15/Arndale board. Fluidanimate no longer experiences any speedup from SIMD instructions, but a slowdown. The limited SIMD instruction set and resources stall the critical path of the application. For some applications, the IO phase covers a significant portion of the total runtime. An interesting alternative could be to trade cores for SIMD implementations and run alternative workloads on idle cores to achieve better throughput.

B. Energy Results

Figures 5 to 8 provide energy information for all analyzed platforms. However, the analyzed ARM processor shows system energy and average system power rather than processor energy and power, since it does not implement any energy performance counters. All groups of applications benefit from substantial energy savings when using the SIMD implementation. Even though performance achieved using AVX and eight threads is equivalent for scalable benchmarks, the energy saving from vectorization is four times better. Blackscholes benchmark can be run using roughly 6% of the original energy when running on eight threads using AVX technology. The same can be applied to resource-limited benchmarks, where the AVX implementation outperforms both four and eight thread implementations. For code/input limited applications energy improvements are not that significant, but they still benefit from SIMD features. If we compare Ivy and Sandy bridge architectures we can see very similar energy trends. However, when looking at absolute numbers there is a big reduction in the power dissipated by both architectures. In swaptions, the average core power (PP0) goes down from 60W for SB to 35W for IB, while package power (PKG), goes down from 64W to 38W. There is also a slight difference in the package to

[5]Region of Interest, usually parallel phase of the application.



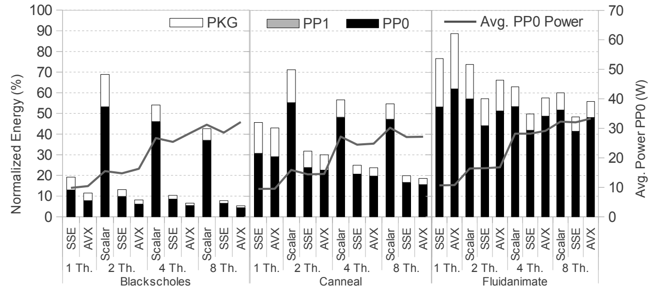


Figure 5: Normalized energy (to 1 thread-scalar) and average PP0 power (secondary axis) for SB (left) and IB (right).

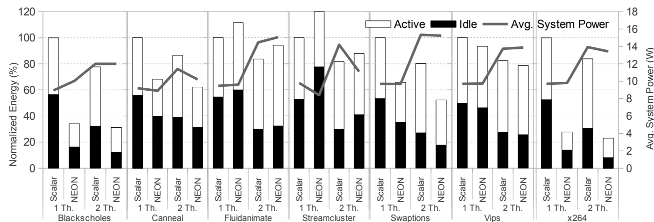


Figure 6: Normalized energy (to 1 thread-scalar) and avg. system power (secondary axis) for Arndale.

core power distribution for these architectures. In our setup, the IB machine is water-cooled, reducing benchmark peak temperature by 20C (vips on 8 threads reached 69C for SB and 49C for IB). We could expect 5% energy increase every 6-8C for IB [17] if it was using air cooling. This temperature related reduction seems to affect more to the PP0 than to the rest of the package, that is why the percentage values change between platforms.

Energy trends for the Cortex-A15 are very similar to those of the Intel processors, except for those applications that take longer to execute, where energy efficiency is reduced. There are two key differences between these platforms. First, resource-limited benchmarks experience a reduction on the average power since when the processor stalls to wait for the memory subsystem it can go into a low power mode and thus save some power. Second, ARM platforms show a slight average power increase when running in SIMD mode. Cortex A15 is not designed for HPC, showing lower power but similar energy. However, the lack of energy MSRs makes it hard to compare with Intel platforms.

While a reduction in execution time leads to a reduction on leakage power, higher resource utilization increases dynamic power (assuming the same core voltage/frequency), usually translating into limited total energy savings. Nevertheless, at least on this generation of Intel processors, regular arithmetic instructions and SIMD instructions use approximately the same average power as regular instructions. The speedup of the application "for free" in terms of power is one of the main benefits of vectorization over threading. Tests on Cortex-A15 processor (using NEON) behave slightly differently. This processor has specific SIMD units, so NEON instructions burned more power than regular instructions, giving reduced energy savings, but still being more efficient than threading.

C. Scalable Benchmark Case Study: Blackscholes

Now we will proceed with a detailed analysis of one application for each group (scalable, resource-limited and code/input limited). We will show the instruction count reduction ratio and the cache miss rate for different number of threads in all the evaluated platforms. In addition, for Intel platforms, we provide details about pipeline/front-end stalls (this information is not available on Cortex-A15).

Figures 9 and 10 show detailed performance counter information for the Blackscholes benchmark. Despite the limitations due to branch divergence, the blackscholes application shows good speedup and energy improvements. While the last level cache miss rate remains mostly unchanged for the IB machine, it has a considerable reduction on the SB. The total number of LLC accesses for SIMD implementations on the SB almost doubles that of the IB. However, we could not determine if this improvement is due to a different replacement policy or just background noise from the OS, since both architectures are supposed to be very similar. The miss rate of the data L1 cache follows the same trend for all architectures, increasing linearly with SIMD width. Figures also show how stalls due to L1 data misses for the SB processor increase slightly, but remain unchanged for the IB architecture. This again suggests that SB and IB have slightly different architectures even though IB is a "tick" step (die shrink). However, dominant stall factor remains resource related (RS[6]). The need for more reservation stations increases with SIMD width for this application. The use of Hyperthreading technology translates into a reduction on the total stalled cycles ("other" includes non-stalled cycles).

In terms of total instruction count, the SSE version is able to reduce them by a factor of 7.6x, while AVX increases that factor to 15x. This means that the new SIMD code covers most of the application instructions. For Cortex-A15 this

[6]Reservation Stations.

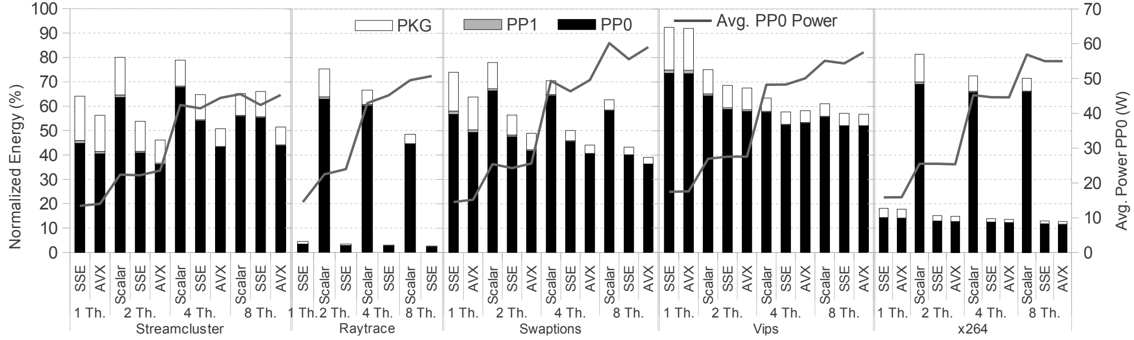


Figure 7: Normalized energy (to 1 thread-scalar) and average PP0 power (secondary axis) for SB.

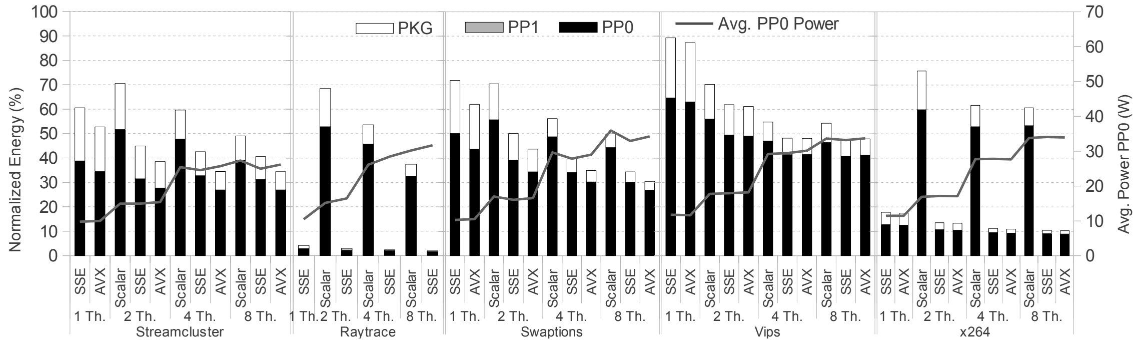


Figure 8: Normalized energy (to 1 thread-scalar) and average PP0 power (secondary axis) for IB.

reduction is around 4.5x for NEON. Blackscholes relies on math libraries that are replaced by our own SIMD-enabled math library when running in SIMD mode. Implementing this library with the reduced instruction set of ARMv7 increases the total instruction count for NEON, but still behaves similarly to SSE.

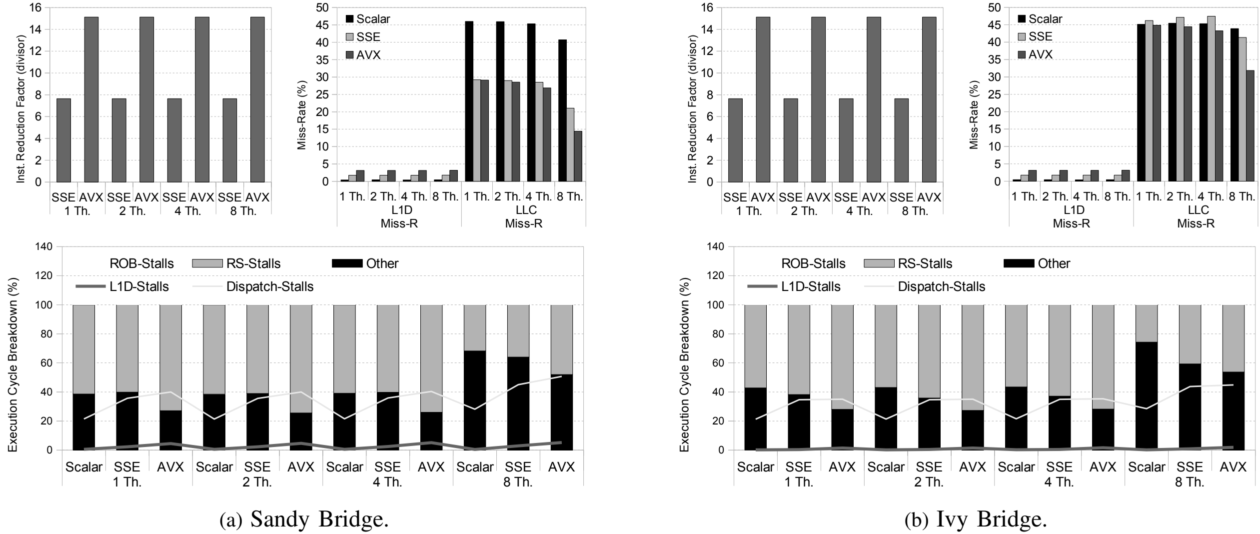
D. Resource Limited Benchmark Case Study: Canneal

Figures 11 and 12 show detailed performance counter information for the canneal benchmark. Note that we are using a cache-blocked version of the canneal benchmark. Using clusters of 64 elements rather than individual random values allows to reduce runtime of the ROI by half as compared to the non cache-blocked original version. This is mainly because of the huge reduction on the L1 data cache miss rate (reduced from 15% to 2.3% for Intel platforms). Our cache-blocked version, in essence, increases computational requirements by clustering nodes. This makes the application not only faster and more energy efficient, but also more suitable for vectorization. However, cache-blocking techniques change the nature of the benchmark itself. In other words, if researchers are looking for an application with high L1 cache misses, they should consider using the non-clustered version of the benchmark.

The SSE implementations of the blocked benchmark shows a high reduction of the total number of instructions (4.4x), reducing runtime on the ROI by half. However, the increase on L1 data cache miss rate (up to 6.3%) is limiting the speedup provided by the SSE implementation. Our AVX implementation is barely obtaining any speedup or energy reduction when compared to SSE. Cache miss rate is reaching 8.7% for the AVX implementation while instruction count is reduced by a factor of close to 6 when compared with the scalar code, meaning that the AVX version is starving both in data and instructions. The amount of RS related stalls is 72% for AVX, 65% when using Hyperthreading since other resources cause additional problems. L1 data misses stalls are similar for both SSE and AVX, this means that, despite the additional cache misses there are other resources that limit SIMD before the increased cache miss rate can impact performance. A higher number of nodes per cluster may solve this issue and increase the benefits of AVX, but parameter optimization is out of the scope of this paper. For the Cortex-A15 results are slightly different. The instruction count reduction factor is not as good as SSE, but still gives similar speedup. In addition, L1 data cache misses barely change when using NEON, but L2 misses increase by about 4%. This suggests that the NEON implementation is also limited by other resources rather than cache misses.

E. Code/Input Limited Bench. Case Study: Fluidanimate

Finally, Figures 13 and 14 show the performance counter summary for the fluidanimate benchmark. The instruction count is only slightly reduced for this application on all



(a) Sandy Bridge.

(b) Ivy Bridge.

Figure 9: Blackscholes instruction reduction factor, cache miss rate and execution cycle breakdown.

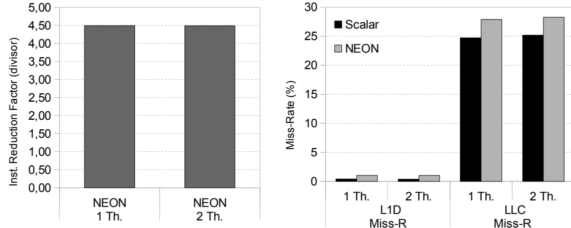


Figure 10: Blackscholes Arndale instruction reduction and L1D miss rate.

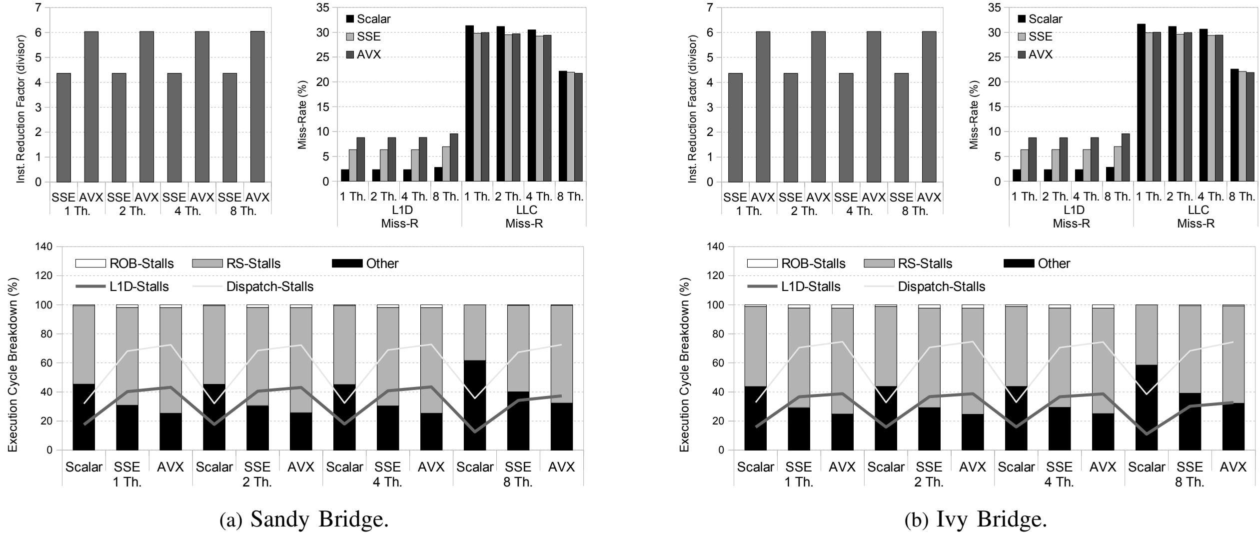
architectures. In consequence, both L1D and LLC misses barely increase. The high amount of divergent branches forces the SSE and NEON implementations to execute many unnecessary instructions that slow down the execution. This is even worse for NEON since it lacks hardware support for some key instructions used by the application.

Fluidanimate divides the evaluation space into a three dimensional grid of particles. For each particle in each cell of each grid it computes forces and densities with adjacent particles in other grids surrounding the particle. In order to use SSE/NEON (AVX) our implementation requires at least 4 (8) particles in each adjacent cell to be used. If not present, the application relies on regular scalar code. Unluckily, PARSEC's inputs have a very small amount of particles per cell (usually 1 or 2), limiting the maximum potential number of SIMD instructions that can be generated. AVX experiences even greater problems since it doubles the width of the SIMD registers, increasing the amount of divergent calculations and finding less chances to generate SIMD instructions. Since the amount of SIMD instructions generated is relatively small the impact of the SIMD implementation on hardware resources is negligible. We strongly believe that a different input (with higher density of particles per cell) would make vectorization show its real potential in terms of performance and energy efficiency for this application.

VI. Related Work

While developing SIMD-enabled applications maximizing resource utilization has been studied and well documented by many researchers [6], automatic vectorization of existing codes is still an open research topic [8]. When the compiler is able to detect code sections to be vectorized results are very similar to those obtained by manually optimizing the application [7]. However, even small code changes can make a huge difference in the way the compiler interprets the code, making the whole process of achieving similar results to manual optimizations very time consuming.

The benchmarking field has also experienced a quick evolution in the past years. Bienia [11] introduces the PARSEC benchmark suite and performs a deep evaluation in terms of performance and resource utilization in his PhD thesis. Ferdnan *et al.* [18] introduces a benchmark suite (CloudSuite) for emerging scale-out workloads, and compares it against state of the art benchmark suites (PARSEC and SPLASH-2). They discuss why architectural changes are necessary to match the requirements of these new workloads. However, none of these approaches analyze effects of vectorization on application energy efficiency. In fact, it could be possible that vectorizing these new workloads would make them more suitable for current architectures. Molka *et al.* [19] discuss weaknesses of the Green500 list with respect to ranking HPC system energy efficiency. They introduce their own benchmark using a parallel workload generator and SIMD support to stress main components in a HPC system. The RODINIA [20] and ALPBench [21] benchmark suites also offer SIMD support. Many authors rely on these benchmark



(a) Sandy Bridge.

(b) Ivy Bridge.

Figure 11: Canneal instruction reduction factor, cache miss rate and execution cycle breakdown.

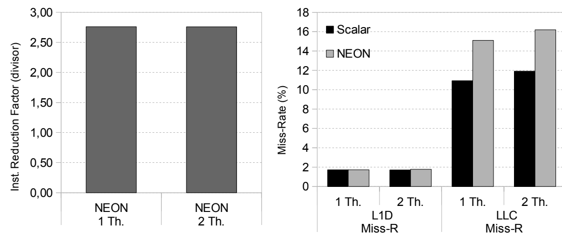


Figure 12: Canneal Arndale inst. reduction and cache miss rate.

suites, but sometimes underestimate the effects that SIMD-enabled applications can have over their contributions. As an example, Totoni *et al.* [22] compare power and performance of several Intel CPUs and GPUs, but do not analyze effects of vectorization on system energy efficiency.

For those applications and kernels used in PARSEC that we managed to vectorize there has also been some evolution in the past few years. Shuo Li from Intel presented a case study of the vectorization of the blackscholes using AVX technology [23]. Ghose *et al.* [24] evaluate different vectorization options for the Streamcluster PARSEC kernel. The vectorization they propose is slightly different from our code (we use SSE3 specific instructions for horizontal adds and AVX). Lotze *et al.* [14] introduce a commercial library for accelerated LIBOR swaption portfolio pricing (equivalent to PARSEC's swaptions application). Intel has also updated their raytracing kernels and packed them in the Embree application (http://embree.github.io/).

VII. Conclusions

Validation of new academic proposals is usually performed against selections of representative applications from different fields of interest known as benchmark suites. However, if benchmarks don't cover the most common architectural features, architects may end up under/over estimating the impact of their contributions. In this paper, we have extended 8 out of 12 benchmarks from the PARSEC benchmark suite with SIMD functionality to assess if SIMD applications should be considered when proposing new architectural modifications in addition to scalar applications. Our vectorization relies on a custom built wrapper and math libraries that will be made available to the research community.

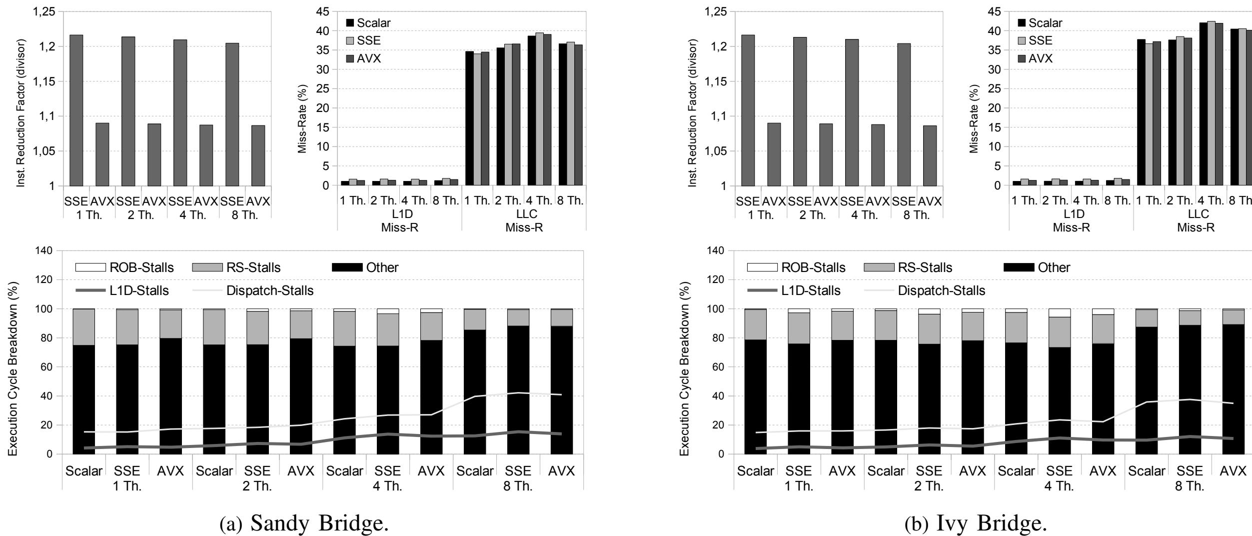
Our evaluation shows that not all applications experience a linear speedup when vectorized, either due to hardware bottlenecks or algorithmic limitations. For vectorization-friendly benchmarks there is a huge impact on both performance and energy efficiency. Intel hardware especially benefits from vectorization as a power saving mechanism since average core power remains almost unaltered when using SIMD instructions. Results also show that SIMD implementations change the architectural trade-offs (especially the requirements to the memory subsystem) of the processor. It is important to keep benchmarks up-to-date with current technologies advancements (e.g., vectorization, heterogeneity, etc) so we can identify design bottlenecks and new hardware requirements.

References

[1] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach, Fifth Edition*. Morgan Kaufmann Publishers Inc., 2012.

[2] R. Dennard *et al.*, "Design of Ion-implanted MOSFET's with Very Small Physical Dimensions," in *Solid-State Circuits, IEEE Journal of*, vol. 9, no. 5, 1974.

[3] K. Skadron *et al.*, "Challenges in Computer Architecture Evaluation," *Computer*, vol. 36, pp. 30–36, 2003.



(a) Sandy Bridge.

(b) Ivy Bridge.

Figure 13: Fluidanimate instruction reduction factor, cache miss rate and execution cycle breakdown.

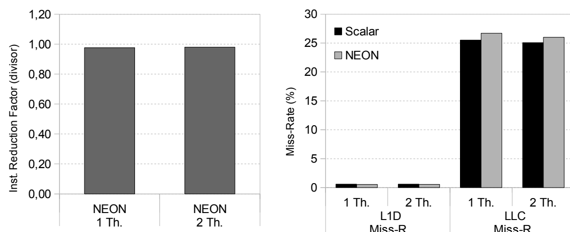


Figure 14: Fluidanimate Arndale instruction reduction and cache miss rate.

[4] W. Feng *et al.*, "OpenCL and the 13 Dwarfs: A Work in Progress," in *Proc. of the 3rd ACM/SPEC Int. Conf. on Performance Engineering*, 2012, pp. 291–294.

[5] S. Borkar and A. A. Chien, "The Future of Microprocessors." ACM, May 2011.

[6] R. Gerber, *The Software Optimization Cookbook*. Intel Press, 2002.

[7] C. Kim *et al.*, "Technical Report: Closing the Ninja Performance Gap through Traditional Programming and Compiler Technology," 2012.

[8] D. Nuzman and R. Henderson, "Multi-platform Auto-vectorization," in *Proceedings of the International Symposium on Code Generation and Optimization*, ser. CGO '06, 2006.

[9] N. Firasta *et al.*, "White Paper: Intel AVX: New Frontiers in Performance Improvements and Energy Efficiency," 2008.

[10] J. M. Cebrian, L. Natvig, and J. C. Meyer, "Improving Energy Efficiency through Parallelization and Vectorization on Intel Core i5 and i7 Processors," in *Proc. of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. IEEE Computer Society, 2012, pp. 675–684.

[11] C. Bienia, "Benchmarking Modern Multiprocessors," Ph.D. dissertation, Princeton University, January 2011.

[12] J. Pommier, "Simple SSE and SSE2 sin, cos, log and exp," 2007. [Online]. Available: http://gruntthepeon.free.fr/ssemath/

[13] H. Lien *et al.*, "Case Studies of Multi-core Energy Efficiency in Task Based Programs," in *ICT-GLOW*, 2012, pp. 44–54.

[14] J. Lotze, P. D. Sutton, and H. Lahlou, "Many-Core Accelerated LIBOR Swaption Portfolio Pricing," in *Proc. of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. IEEE Computer Society, 2012, pp. 1185–1192.

[15] P. J. Mucci *et al.*, "PAPI: A Portable Interface to Hardware Performance Counters," in *Proc. of the Department of Defense HPCMP Users Group Conference*, 1999.

[16] Y. Sazeides *et al.*, "The Danger of Interval-based Power Efficiency Metrics: When Worst is Best," in *In Computer Architecture Letters, Vol 4*, 2005.

[17] J. M. Cebrian and L. Natvig, "Temperature Effects on On-Chip Energy Measurements," in *Proc. of The Third Int. Workshop on Power Measurement and Profiling*, 2013.

[18] M. Ferdman *et al.*, "Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware," in *17th Int. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2012.

[19] D. Molka *et al.*, "Flexible Workload Generation for HPC Cluster Efficiency Benchmarking." Springer Berlin / Heidelberg, 2011.

[20] S. Che *et al.*, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *Proc. of the 2009 IEEE Int. Symp. on Workload Characterizatio*. IEEE, 2009, pp. 44–54.

[21] M. Li *et al.*, "The ALPBench Benchmark Suite," in *In Proc. of the IEEE Int. Symp. on Workload Characterization*, 2005.

[22] E. Totoni *et al.*, "Comparing the Power and Performance of Intel's SCC to State-of-the-art CPUs and GPUs," vol. 0. IEEE Computer Society, 2012, pp. 78–87.

[23] S. Li, "Case Study: Computing Black-Scholes with Intel® Advanced Vector Extensions," 2012. [Online]. Available: http://software.intel.com/en-us/articles/case-study-computing-black-scholes-with-intel\\-advanced-vector-extensions

[24] S. Ghose, S. Srinath, and J. Tse, "Accelerating a PARSEC benchmark using portable subword SIMD," in *CS 5220: Final Project Report. Sch. of Elec. and Comp. Eng., Cornell Eng.*, 2011.