

Performance Characterization and Optimization for Intel Xeon Phi Coprocessor

Cheng Zhang^{1(✉)}, Li Liu², Ruizhe Li¹, and Guangwen Yang^{1,2(✉)}

¹ Department of Computer Science and Technology,
Tsinghua University, Beijing 100084, China
{zhang-cheng09,lrz04}@mails.tsinghua.edu.cn

² Center for Earth System Science, Tsinghua University, Beijing 100084, China
{liuli-cess,ygw}@mail.tsinghua.edu.cn

Abstract. The Intel Xeon Phi is a many-core accelerator which focuses on the high performance applications. To characterize the performance of the Intel Xeon Phi, a system of dual 8-core Intel Xeon E5-2670 processors is employed as a control platform, and a subset of the PARSEC benchmark suite is selected as the benchmark applications. The first evaluation in this paper shows that the applications on the Intel Xeon Phi is averagely 2.06x slower than on the dual Intel Xeon E5-2670. The further detailed performance characterization quantifies the performance impact of various architecture parameters on the Intel Xeon Phi. To set an example for how to improve the architecture of the Intel Xeon Phi for better performance, the hardware optimization with an additional set of vector processing units is discussed and a simple emulator is developed accordingly. The evaluation results show that this optimization can provide an average speedup of 1.10.

Keywords: Intel Xeon Phi · Parallel architecture · Performance characterization · Architecture parameters · Hardware optimization

1 Introduction

Limited by power consumption and chip cooling, many-core accelerators, for example GPGPU, have gained popularity in recent years. As the architecture of the accelerators always differs from modern CPUs in many aspects, users have to pay much more efforts to rewrite the applications with other programming models like CUDA or OpenCL, which decreases the popularity of the accelerators.

To achieve better programmability and usability than GPGPU, the Intel Corporation has developed a new many-core processor product, the Intel Xeon Phi ‘Knights Corner’ Coprocessor (KNC), based on the Intel Many Integrated Core (MIC) architecture. As its main architecture is similar to modern multi-core CPUs, the KNC supports almost all standard program models, such as OpenMP, MPI and POSIX threads (Pthreads). Generally, the CPU applications can directly run on the KNC with a few or even no changes.

At the same time, there are some specific architecture designs in the KNC. For example, the KNC has 61 cores, each of which is an in-order dual-issue core with two levels of cache, supports 4 hardware threads and has a set of 512-bit wide vector processing units (VPU). These specific architecture designs make the KNC achieve higher peak performance, and result in different performance characteristics of applications, compared to CPUs.

The KNC has attracted a lot of research interests ever since it became available. Most of previous researches related to the KNC focus on optimizing applications according to the specific architectures of the KNC [1–10]. Although the KNC provides much higher peak performance than modern CPUs, the CPU applications rarely achieve performance improvement or even obtain much poorer performance when being directly run on the KNC. To achieve better application performance on the KNC than the CPUs, users have to manually optimize the program heavily according to the specific architecture features of the KNC. This dramatically harms the highlighted advantages of the KNC in programmability and usability. In other words, the KNC also suffers from the conflict between programmability and performance. On the other hand, the KNC mainly targets to the kind of applications that can scale well over one hundred threads and efficiently use the VPU [11]. If the KNC can achieve similar or even better performance than CPUs for more kinds of applications, it will be more attractive for usage.

We therefore use a subset of the parallel benchmark suite PARSEC [12, 13], which are from different application domains and with different performance characteristics (VPU utilization, memory accessing, parallel scalability and so on), as the benchmark applications for this work. To provide valuable references for further improving the architecture of the KNC, we first quantify the performance characteristics of the benchmark applications with the same code on the KNC and CPU. We use dual 8-core Intel Xeon E5-2670 processors (As the code-name of Intel Xeon E5-2670 is “Sandy Bridge”, we call it as SNB in the following context; each SNB core is out-of-order execution, supports 2 hyper threads and has a set of 256-bit VPU), as a counterpart to the KNC.

Although the peak performance of the KNC is 3.20x higher than the dual SNB, the average performance of these benchmark applications on the KNC is 2.06x smaller than the dual SNB. To reveal why, we quantify the performance characteristics in two steps. In the first step, we analyze the single-core performance of the benchmark applications on the two systems. The single-core performance achieved on the KNC is 3.76x smaller than the dual SNB, although the peak performance of a KNC core is only 1.19x smaller than a SNB core. The causes for this result are quantified as follows:

- When vectorization is disabled, the number of micro operations executed on the KNC is 1.11x more than the dual SNB
- When vectorization is disabled, the architecture of 4 hardware threads on each KNC core significantly improves the instruction level parallelism (ILP) of the applications with a speedup of 2.21. The micro operations per cycle (MPC)

on the KNC is only 1.07, which is much smaller than 1.58, the MPC on the dual SNB. The causes for this result include:

- The usage of the two pipelines (U pipeline and V pipeline) on a KNC core is imbalanced. About 68.5 % instructions are executed by the U pipeline.
 - The branch misprediction rate on the KNC is 15.8 %, which is much larger than 2.6 %, the branch misprediction rate on the dual SNB. The overhead of branch mispredictions covers 4.2 % execution time on the KNC.
 - The average latencies of the memory hierarchies on the KNC are longer than the dual SNB. Meanwhile, the KNC does not have L3 cache to hide the latencies of memory accesses. On the KNC, accessing the memory hierarchies costs about 21.3 % execution time. But on the dual SNB, it costs about 19.3 % execution time.
- Although the VPU of the KNC is 2x wider than the dual SNB, the speedup achieved through vectorization on the KNC is only 1.17x higher.

In the second step, we analyze the scaling performance of the benchmark applications on the two systems when using multiple or even many cores, referencing the single-core performance on each system as the baseline. The overall speedup on the KNC is only 1.97x higher, although the KNC has 61 cores and the dual SNB has 16 cores:

- When 8 cores are used, almost the same average parallel speedup 6.36 and 6.52 is achieved on the KNC and SNB.
- When the number of cores is increased from 8 to 16, the average parallel speedup achieved on the KNC and dual SNB is 1.71 and 1.36 respectively. The lower parallel speedup on the dual SNB is because the cache-coherent Non-Uniform Memory Access architecture (ccNUMA) [14,15] introduces remote memory access with much longer latency, which results in about 21.8 % performance loss.
- Some applications cannot scale well to so many cores on the KNC. When achieving the best application performance on the KNC, the number of cores used by the benchmark applications is about 40 on average, which wastes about 1/3 cores.

The above results indicate that the application performance on the KNC can be prospectively improved through balancing the pipeline usage, improving the branch prediction architecture and improving the cache architecture. To investigate how to balance the pipeline usage, we developed a simple emulator that can well simulate the pipeline usage on the KNC with an average accuracy 98.7 %. Using this emulator, an additional set of VPU for the V pipeline is added. With this optimization, the KNC can gain a speedup of 1.104 on average.

The remainder of the paper is organized as follows. Section 2 presents the related work of Intel Xeon Phi. Section 3 describes the experiment settings in this paper. Section 4 quantifies the architecture parameters resulting in the performance differences of the two systems. Section 5 details an emulator and hardware optimizations for balancing the pipeline usage on the KNC. Section 6 discusses and concludes this work.

2 Related Work

Intel Xeon Phi is a new many-core product, which has a similar architecture with multi-core CPUs. It supports almost all standard program models such as OpenMP, MPI and POSIX threads. Previous studies have shown that transporting scientific applications and kernels to Intel Xeon Phi is relatively straightforward [16, 17]. So better programmability and usability is the highlighted advantage of Intel Xeon Phi, comparing with other accelerators.

Since Intel Xeon Phi is released, a lot of researchers pay much attention on it. Existing works can be classified into two categories. The first category evaluates the basic performance parameters of the KNC using micro-benchmarks. Schmidl D. et al. showed that the maximum memory bandwidth achieved on the KNC is about 150GB/s, which is far below the theoretical memory bandwidth 352GB/s [1]. Saini S. et al. showed that the average latencies of the memory hierarchies on the KNC are longer than on the dual SNB. The average latencies of accessing L1, L2, L3 cache and main memory on the dual SNB are 4, 12, 39 and 210 cycles respectively. The average latencies of accessing L1, L2 cache and main memory on the KNC are 3, 24 and 310 cycles respectively [18]. Cramer T. et al. showed that the KNC does not introduce more overheads for OpenMP synchronization, although it has so many cores [5]. These works also demonstrated that when CPU applications directly run on the KNC without manual optimizations, the applications cannot effectively utilize the computing capability of the KNC.

The second category focuses on optimizing applications according to the architecture of the KNC [2–4, 7–9]. Williams S. et al. showed that without optimization, the dual SNB outperforms the KNC by 1.45x for 3D geometric multi-grid [3]. With a series of optimizations including communication-avoiding, SIMD and prefetching, the application gains about 3.8x and 1.5x improvement on the KNC and the dual SNB respectively. Finally, the KNC outperforms the dual SNB by 1.75x. Similar results can be found in [2], where a set of financial analytics benchmarks is optimized. Taking the Brownian bridge algorithm in the benchmarks as an example, with simple optimizations using `pragma simd`, `omp` and `unroll`, the KNC is 25 % slower than the dual SNB. With further optimizations including SIMD across paths, reducing bandwidth usage and etc., the KNC performs 2x faster than the dual SNB.

Previous works demonstrated that the application performance of the KNC over the dual SNB highly depends on deep manual optimizations which introduce significant code changes. The KNC almost cannot outperform the dual SNB when without deep manual optimizations or only with simple optimizations. This situation can make the KNC less attractive. Improving the hardware architecture is another important approach to boosting the application performance without codes changes. To improve the hardware architecture of the KNC, this paper quantified the impact of various architecture parameters on the application performance. The results revealed the potential of different hardware optimizations. Moreover, we investigated the hardware optimizations to balance the pipeline usage of the KNC.

3 Analysis Settings

In this work, the dual SNB is used as a counterpart of the KNC. This section will briefly introduce their architecture. For the purpose of performance characterization, several applications from the PARSEC [12, 13] are selected as benchmarks.

3.1 Intel Xeon Phi

Figure 1 shows the micro-architecture of the Intel Xeon Phi. To deliver high computing capability, the KNC integrates 61 cores and each core has a set of 512-bit wide VPU. So the KNC can achieve 1063.84 GFLOPS (double-precision).

Each KNC core is an in-order x86 processor core, based on a modified Pentium processor design. Its instruction set is similar to Pentium, however, it does not support MMX, SSE or AVX instructions. It is dual-issued with 2 different pipelines (U and V pipeline). The U pipeline can execute all the instructions, while the V pipeline can only execute a proportion of instructions. To hide the latency exposed by the in-order execution, each core supports four hardware threads. Its instruction decoder is a two-cycle unit. A KNC core therefore cannot execute the instructions from the same hardware thread cycle by cycle. To reduce the size of the die and power consumption, the KNC core lacks state-of-the-art branch prediction units.

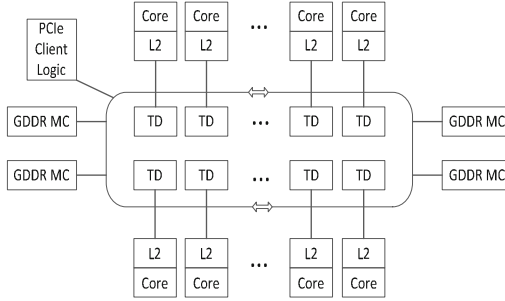


Fig. 1. Micro-architecture of Intel Xeon Phi.

3.2 Dual Intel Xeon E5-2670

For comparison, we use dual Intel Xeon E5-2670 processors as the representative of CPUs. Figure 2 shows its micro-architecture. Each SNB integrates 8 core and each core has a set of 256-bit wide VPU. So the dual SNB can achieve 332.8 GFLOPS (double precision).

Each SNB core is an out-of-order x86 processor core. It supports full x86 instruction set, MMX, SSE and AVX instruction set. The SNB core is quad-issued, i.e. 4 instructions can be issued in a cycle. Different from the KNC, the instruction decoding on the SNB only takes one cycle. To increase the performance of the SNB core, it has state-of-the-art branch prediction units.

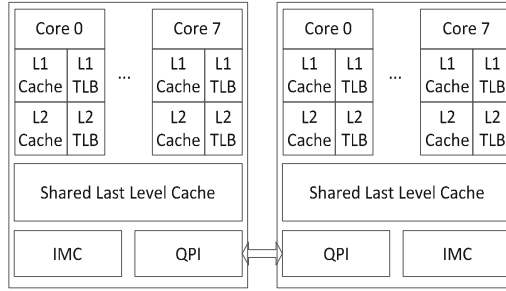


Fig. 2. Micro-architecture of dual Intel Xeon E5-2670.

3.3 PARSEC Benchmarks

PARSEC is one of the most important multi-core and multi-processor benchmark suites, which has been widely used for application-driven researches and performance measurement of real machines. As a benchmark suite, PARSEC has the following five necessary characteristics [12, 13]:

- Multithreaded Applications. All applications in PARSEC are multithreaded using OpenMP, POSIX threads or Intel Threading Building Blocks (TBB).
- Emerging Workloads. New emerging applications that require higher computing capability are included by this suite.
- Diversity. PARSEC contains applications in different domains (Financial Analysis, engineering, data mining, animation, etc.) with different parallel programming models (OpenMP, Pthreads, TBB) and parallel granularities.
- State-of-the-Art Techniques. The applications in the benchmark suite implement the latest techniques (e.g., simulated annealing, deduplication, ray tracing, etc.).
- Supporting Research. This suite provides an infrastructure for research, which allows researchers to instrument and simulate the applications effectively.

In this work, a subset of PARSEC benchmark suite is selected, including blackscholes, canneal, dedup, fluidanimate, streamcluster and swaptions. Table 1 shows the characteristics of these selected 6 applications. We believe they have sufficient diversity to quantify the impact of various architecture parameters on the application performance.

Throughout this work, we use the Intel compiler version 14.0.2 with optimization level O3 to compile the benchmark applications on both the KNC and dual SNB. In our experiment, Intel Xeon Phi is used as a standalone SMP on a single chip. In this mode, the code is compiled on the dual SNB using the flags -mmic. The binary can be executed on the KNC only.

4 Performance Characterization

To motivate the detailed performance characterization, we first evaluate the overall application performance on the KNC and dual SNB. In Table 2, the left

Table 1. Key characteristics of the benchmark applications.

Application	Application domain	Parallelization granularity	Cache requirement	Data usage	
				Sharing	Exchange
blackscholes	Financial analysis	Coarse	2 MB	Low	Low
canneal	Engineering	Fine	2 GB	High	High
dedup	Enterprise storage	Medium	2 GB	High	High
fluidanimate	Animation	Fine	128 MB	Low	Medium
streamcluster	Data mining	Medium	256 MB	Low	Medium
swaptions	Financial analysis	coarse	512 KB	Low	Low

half shows the best application performance where the KNC may not be fully utilized. The right half shows the application performance for maximum usage of the cores and hardware threads on the KNC. These results reveal that the KNC is largely underutilized. The best performance of the applications on the KNC is 2.1x poorer than the dual SNB on average, although the peak performance of the KNC is 3.2x higher than the dual SNB. According to the architecture difference between the KNC and dual SNB, we conduct two major steps for detailed analysis: single-core performance analysis and scaling performance analysis with multiple or even many cores.

4.1 Single-Core Performance Analysis

Table 3 shows the execution time of the applications when using only one core on the KNC and dual SNB respectively. Although the peak performance of a KNC core is only 1.19x smaller than a SNB core, the single-core performance on

Table 2. The application performance on the two systems. Optimal means the best application performance on the two systems. Maximum means the application performance with maximum usage of the cores and hardware threads on the KNC. The thread number of fluidanimate must be 2^n . We therefore use 128 threads to run it on the KNC for maximum. The thread number of swaptions is less than 128. We therefore use 60 cores and 120 threads to run it for maximum.

Application	Optimal				Maximum	
	KNC		Dual SNB		KNC	
	Time(s)	Cores/threads	Time(s)	Cores/threads	Time(s)	Cores/threads
blackscholes	1.35	60/240	2.25	16/16	1.35	60/240
canneal	19.60	60/120	16.28	16/16	40.68	60/240
dedup	26.48	16/32	9.08	16/16	109.67	60/240
fluidanimate	100.54	32/64	56.93	16/16	131.18	60/128
streamcluster	114.93	16/32	40.90	16/16	1398.93	60/240
swaptions	41.45	60/120	13.46	16/16	41.45	60/120

the KNC is averagely 3.76x smaller than on the dual SNB. This result is because of less effective vectorization and lower ILP on the KNC.

According to Table 3, the number of micro operations on the two systems is different, and the micro operations on the KNC are 1.11x more than the dual SNB on average. This is due to the difference of instruction sets on the two systems.

Both the KNC and dual SNB provide VPU for performance acceleration. However, only two applications, i.e., blackscholes and streamcluster, can benefit from vectorization (please refer to the speedup of vectorization in Table 3). This is because there are almost no manual optimizations for vectorization and most operations cannot be automatically vectorized by the compiler in the other 4 applications. The VPU of a KNC core is 2x wider than a SNB core, while the vectorization speedup of blackscholes on the KNC is 2.5x higher than the SNB. This is because the math functions log and exp, which are frequently called in blackscholes, obtain vectorization speedup of 9.3 and 2.1 on the KNC and dual SNB respectively. The much larger vectorization speedup on the KNC is because the VPU on the KNC supports single-precision transcendental instructions for exp and log [19]. Regarding streamcluster, vectorization achieves a speedup of 1.9x on the dual SNB while almost no speedup on the KNC. This is because the most significant data structure in streamcluster is written into an array of structures (AOS), which harms the vectorization speedup obtained on the KNC [2]. On average, vectorization speedup achieved on the KNC core is only 1.17x higher than on the SNB core.

Although the KNC provides four hardware threads to exploit ILP, most of the applications, i.e., blackscholes, dedup, fluidanimate and swaptions, obtain much poorer ILP on the KNC (please refer to the micro operations per cycle in Table 3). On average, the MPC on the KNC is only 1.07, much smaller than 1.58, the MPC on the dual SNB. To reveal why much lower ILP is achieved on the KNC, we further investigate the branch prediction performance and the memory access performance on the two systems, and the pipeline usage on the KNC.

Branch Prediction Performance. Table 4 shows the branch prediction performance on the KNC and dual SNB respectively. The number of branch instructions executed is almost the same on the two systems. However, the KNC achieves much poorer branch prediction performance, with much more branch mispredictions. This is because the KNC cores lack state-of-the-art branch prediction units. The overhead of branch mispredictions on the KNC is low, due to the in-order execution and short pipelines [20]. According to our evaluation, the average overhead of a branch misprediction on the KNC is about 4 cycles. Almost 4.2% execution time on the KNC is wasted by branch mispredictions, which further limits the MPC no more than 1.92 on average.

Memory Access Performance. To measure the overhead introduced by memory accessing, we first obtained the number of cache misses using the VTune.

Table 3. Performance parameters of the applications when only using one core on the two systems. The number of micro operations and micro operations per cycle is collected by VTune.

Application	Execution time(s) (enable vectorization)		Vectorization speedup		Number of micro operations (disable vectorization)		Micro operations per cycle (disable vectorization)	
	KNC (4 threads)	SNB (1 thread)	KNC	SNB	KNC	SNB	KNC	SNB
blackscholes	35.12	26.88	10	4	3.96e11	4.87e11	1.05	1.76
canneal	498.92	154.13	1	1	1.94e11	1.30e11	0.334	0.272
dedup	134.24	35.30	1	1	1.48e11	1.57e11	1.15	1.94
fluidanimate	1699.19	449.02	1	1	1.99e12	1.95e12	1.11	1.85
streamcluster	1006.54	270.33	1.02	1.91	1.68e12	1.63e12	1.50	1.31
swaptions	1403.05	210.12	1	1	1.93e12	1.39e12	1.25	2.33

Table 4. Performance of branch prediction, when vectorization is disabled.

Application	Branch instructions		Misprediction PKI		Overhead(%)
	KNC	SNB	KNC	SNB	KNC
blackscholes	2.71e10	2.75e10	11.50	0.60	3.53
canneal	1.31e10	1.28e10	14.02	2.99	1.16
dedup	1.32e10	1.44e10	15.11	6.05	4.40
fluidanimate	3.00e11	2.92e11	39.51	7.26	12.4
streamcluster	1.20e11	1.20e11	2.64	1.25	1.19
swaptions	1.42e11	1.36e11	9.12	1.26	2.80

Then we calculated the overhead according to the access latencies from Saini S. et al. [18], as shown in Table 5. The overhead of memory accessing is different on the two systems for each application. This is because the architecture of memory hierarchies is different. On average, about 21.3 % and 19.3 % of execution time is spent on accessing memory hierarchies on the KNC and dual SNB respectively. This overhead further limits the MPC no more than 1.50 on average.

Instruction Pipeline Usage. Table 6 shows the pipeline usage on the KNC. Most instructions are performed by the U pipeline. This is because the in-order execution and the function layout of the pipelines decreases the chances of simultaneously issuing multiple instructions. The U pipeline can execute all the instructions, while the V pipeline can execute only a part of instructions. When there is only one instruction able to be issued, it must be operated on the U pipeline. When there are two independent instructions, they can be issued and operated on the two pipelines simultaneously only when they are pairable (there are specific pairing rules to check whether the second instruction can pair up with the first instruction). So the number of micro operations executed by the

Table 5. Performance of cache hierarchies accessing, when vectorization is disabled.

Application	L1 misses PKI		L2 misses PKI		L3 misses PKI	Overhead (%)	
	KNC	SNB	KNC	SNB	SNB	KNC	SNB
blackscholes	0.09	0.08	0.013	0.014	0.013	0.22	0.39
canneal	18.7	24.6	14.3	17	15.2	77.02	64.81
dedup	10.7	11.7	0.07	0.09	0.018	24.48	21.02
fluidanimate	2.20	2.51	0.31	0.36	0.23	5.48	6.71
streamcluster	18.57	22	0.92	1.09	1.04	10.53	14.45
swaptions	8.3	7.8	0.0003	0.0004	0.00004	10	8.12

Table 6. Proportion of instructions executed by the two pipelines of the KNC core, when vectorization is disabled.

Application	Proportion of instructions executed (%)	
	U pipeline	V pipeline
blackscholes	83.3	16.7
canneal	67.2	32.8
dedup	65.8	34.2
fluidanimate	75.1	24.9
streamcluster	54.4	45.6
swaptions	65.5	34.5

V pipeline is much less than the number of micro operations executed by the U pipeline. The imbalanced usage of the U and V pipelines limits the MPC no more than 1.11 on average.

The second column in Table 7 lists out the calculated MPC on the KNC when only considering the overhead introduced by the imbalance of pipeline usage, branch mispredictions and memory accessing, which is close to the real MPC in the third column. This result demonstrates that these three factors are main causes to the smaller MPC on the KNC.

4.2 Scaling Performance Analysis with Multiple or Many Cores

In this analysis, we measured the parallel speedup of the benchmark applications on the KNC and dual SNB. The single-core performance is referenced as the baseline for calculating the parallel speedup. In other words, the parallel speedup of using only one core is 1. Table 2 shows the overall parallel speedup when fully utilizing the KNC and dual SNB respectively. On average, the overall parallel speedup on the KNC is only 1.97x higher than the dual SNB, although the number of cores on the KNC is 3.75x more than the dual SNB. According to the architecture of the KNC and dual SNB, we selected three levels to scale core number, i.e., 8, 16, and 60, for further analysis.

Table 7. Micro operations per cycle calculated and real micro operations per cycle measured on the KNC, when vectorization is disabled. When calculate micro operations per cycle, we only consider the overhead introduced by the imbalance of pipeline usage, branch misprediction and memory accessing.

Application	Micro operations per cycle	
	Calculated	Measured
blackscholes	1.16	1.05
canneal	0.338	0.334
dedup	1.10	1.15
fluidanimate	1.10	1.11
streamcluster	1.63	1.50
swaptions	1.34	1.25

For the dual SNB, the 8 cores on only one SNB processor are used in this test, to avoid the overhead introduced by the Non-Uniform Memory Access architecture. As shown in Fig. 3, similar 8-core speedup is achieved on the two systems.

As shown in Fig. 4, most applications, i.e., canneal, dedup, fluidanimate and streamcluster, obtain a lower 16-core speedup on the dual SNB. This is because the ccNUMA on the dual SNB enlarges the memory access latency. As shown in Table 1, these applications have high-level of data sharing or data exchange among threads. That is the reason why the memory access latency enlarged by the ccNUMA dramatically decreases the 16-core speedup on the dual SNB.

The above analysis shows that the KNC can make applications achieve similar or even higher parallel speedup than the dual SNB, when using the same number of cores. However, when scaling the number of cores on the KNC from 16 to 60, the applications only obtain a 1.12x speedup on average (Table 2). One important cause for this result is that the overhead introduced from load imbalance, synchronization and contention for shared resources rapidly increases with the increment of the thread number.

Regarding the highest parallel speedup, all applications except blackscholes cannot fully utilize the hardware threads and cores on the KNC. On average, the applications can effectively utilize only 40 cores and 100 threads, and obtain a 1.64x speedup when scaling the core number from 16.

5 Architecture Optimizations

The performance results in the last section show that the imbalanced usage of the U and V pipelines on the KNC dramatically limits the ILP of applications. As the imbalanced pipeline usage is due to the in-order execution and the function layout of the two pipelines, we investigated the potential benefits of hardware optimizations, i.e., improving the function layout of the pipelines, for balancing the usage of the pipelines.

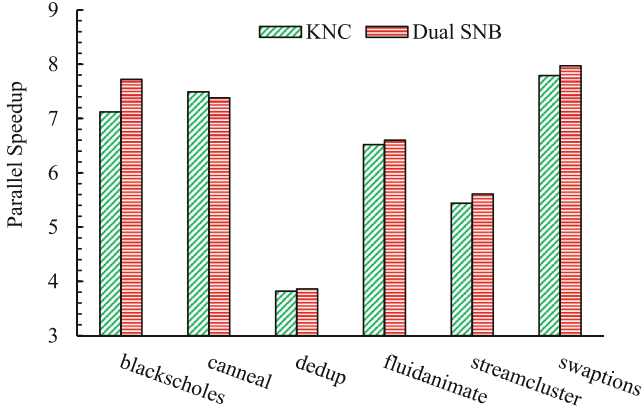


Fig. 3. Parallel speedup of 8 cores.

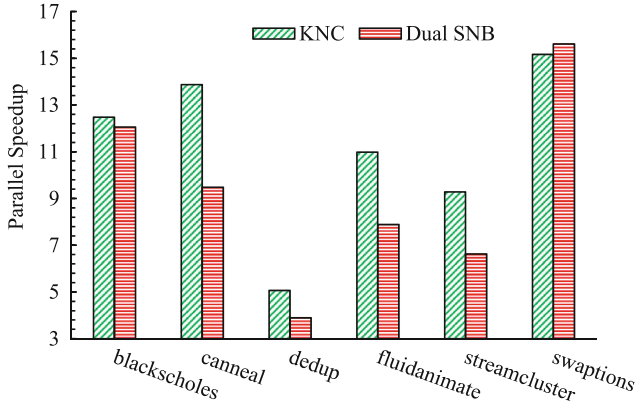


Fig. 4. Parallel speedup of 16 cores.

For evaluating the potential benefits, a simulator is required to simulate the pipeline usage on the KNC. However, there are no simulators for the KNC. Moreover, the simulators for the general multi-core processors focus on modeling the cores, memory system, network and other components, without simulating the pipeline usage, which is our concern. So the existing simulators are not suitable for our simulation.

In the following of this section, we first introduce an emulator that can accurately simulate the usage of the pipelines and next discuss several hardware optimizations for balancing the usage of the pipelines.

5.1 An Emulator for the Balance of Pipeline Usage

In order to simulate the impact of improving the function layout of the pipelines, the instructions that can be executed by the U and V pipelines respectively

Algorithm 1. Selecting instructions to issue

Require: Unissued instruction buffer *Unissue_insts*, Executing instruction buffer *Executing_insts***Ensure:** Instruction *instU* for the U pipeline, Instruction *instV* for the V pipeline

- 1: Unset *instU* and *instV*
 - 2: Set *inst* to be the first instruction in *Unissue_insts*
 - 3: **if** *inst* does not depend on any unretired instruction in *Executing_insts* **then**
 - 4: Set *instU* to be *inst* and delete *instU* from *Unissue_insts*
 - 5: Set *inst* to be the first instruction in *Unissue_insts*
 - 6: **if** *inst* can pair with *instU* and *inst* does not depend on any unretired instruction in *Executing_insts* **then**
 - 7: Set *instV* to be *inst* and delete *instV* from *Unissue_insts*
 - 8: **end if**
 - 9: **end if**
-

can be changed. As the emulator only focuses on the usage of the U and V pipelines, the events that can stall the two pipelines at the same time, e.g., branch mispredictions, cache misses, TLB (translation lookaside Buffer) misses, etc., are not considered and simulated, and the latency of each instruction is directly from the corresponding documents of Intel Corporation [21, 22]. Although the emulator is simply implemented, it can accurately simulate the usage of the pipelines, as shown in Sect. 5.1.2.

Description. Figure 5 shows the architecture of the emulator. The 4 hardware threads on a KNC core are simulated simultaneously. Following the architecture of the KNC core, the instructions issued at one cycle must be from the same hardware thread and the 4 hardware threads are scheduled in round-robin fashion. As a result, the hardware threads are independent to each other in the simulation and the order of issuing instructions among the hardware threads does not impact the balance of pipeline usage.

To trace the instructions executed by the KNC, a tool PIN is used. PIN provides a set of APIs which enable arbitrary codes to be inserted into the binary executable of the application to collect the execution information without modifying the source codes [23]. It provides scalable support for multithreaded applications [24] on heterogeneous environments. For a processor with multiple hardware threads, it can distinguish the instructions for each hardware thread.

As a KNC core can decode 2 instructions each cycle, at most 2 instructions can be executed at one cycle. Algorithm 1 shows how to select the instructions to be issued. If the first unissued instruction depend on any unretired instructions, no instructions are issued at this cycle. Otherwise, it is issued to the U pipeline. Meanwhile, if the second unissued instruction can pair with the first instruction and does not depend on any unretired instructions, it is issued to the V pipeline.

The KNC designs a set of pairing rules to determine whether two instructions can be issued simultaneously to the two pipelines. Each instruction has a pairing mode in four categories (NP, PU, PV and UV) [21]. NP means the

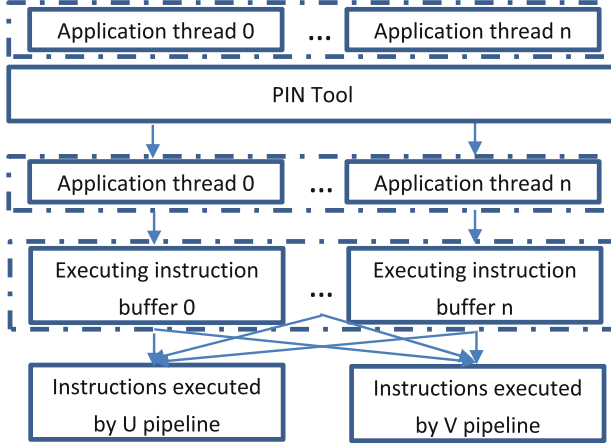


Fig. 5. Architecture of the emulator.

instruction cannot execute with any other instruction concurrently. PU means the instruction can only be issued to the U pipeline. PV means the instruction can only be issued to the V pipeline when there is another instruction to be issued simultaneously. UV means the instruction can be issued to the U or V pipeline. The change of the function layout of the pipelines can be easily achieved through modifying the pairing mode of instructions.

Evaluation. To evaluate the emulator, we built a case to simulate the balance of pipeline usage on the KNC, where the hardware optimization is disabled. Formula 1 defines the maximum instructions per cycle (MIPC), which is the upper bound of ILP determined by the imbalance of pipeline usage, where P_U is the proportion of instructions executed by the U pipeline and P_V is the proportion of instructions executed by the V pipeline. Table 8 shows the MIPC measured by the VTune and simulated by the emulator respectively. The last column in Table 8 shows the error rate of the emulator when the MIPC from the VTune is treated as the reference. The error rate ranges from 0.88 % to 2.18 % and is 1.29 % on average, which demonstrates that our emulator can accurately simulate the balance of pipeline usage.

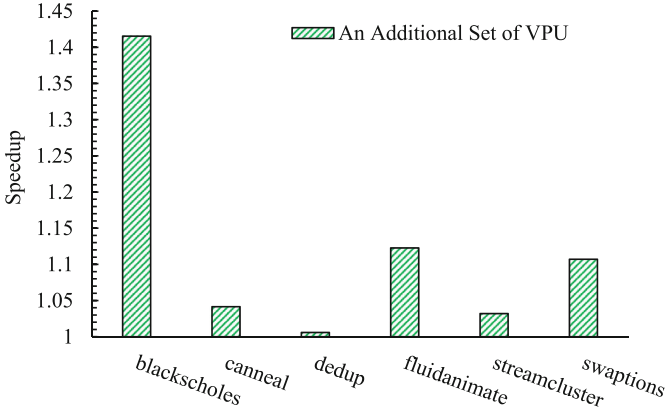
$$MIPC = \frac{1}{\max(P_U, P_V)} \quad (1)$$

5.2 Hardware Optimizations for Better Balance of Pipeline Usage

The two pipelines on a KNC core are different in functionality. Most vector instructions can be executed on the U pipeline only. A straightforward hardware optimization is to make the two pipelines have the same functionality, which

Table 8. MIPC measured by VTune and simulated by our emulator (vectorization is enabled).

Application	MIPC		Error rate of the emulator (%)
	VTune	Emulator	
blackscholes	1.140	1.130	0.88
canneal	1.487	1.504	1.14
dedup	1.520	1.544	1.58
fluidanimate	1.331	1.360	2.18
streamcluster	1.514	1.528	0.92
swaptions	1.527	1.543	1.05

**Fig. 6.** Speedup of MIPC from this hardware optimization in our emulator. The baseline is the MIPC on the KNC measured by the VTune.

requires an additional set of 512-bit VPU for the V pipeline. This optimization is easily achieved by the emulator through modifying the pairing mode of the corresponding instructions from PU to UV. For other instructions with paring mode NP, PV or UV, their paring mode keeps the same. Figure 6 shows the speedup of MIPC achieved by an additional set of VPU for the V pipeline. With an additional set of VPU for the V pipeline, the balance of the pipeline usage is improved, especially for the applications blackscholes, fluidanimate and streamcluster. On average, the KNC achieves a speedup 1.121.

Figure 7 shows the performance improvement achieved from the hardware optimization, considering the overhead of branch mispredictions and memory accessing (reference to Sect. 4.1). With an additional set of VPU for the V pipeline, the KNC gains an average speedup of 1.104. So we can demonstrate that the hardware optimization can improve the real performance of the KNC effectively.

The energy per instruction (EPI) of the vector instructions with register operands is 2x of the scalar instructions and EPI of the vector instructions with other operands is almost same as the scalar instructions [25]. That is to say, the SIMD execution is energy-efficient to boost peak performance [26]. So we believe that an additional set of 512-bit VPU for V pipeline is practical.

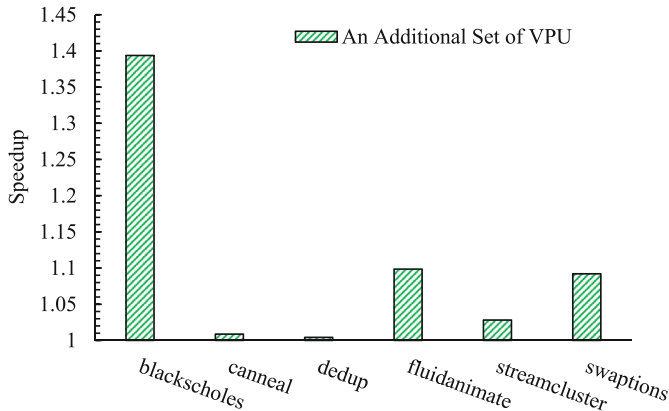


Fig. 7. Performance improvement from the hardware optimization in our emulator, when considering the overhead of branch mispredictions and memory accessing. The baseline is the performance on the KNC measured by the VTune.

6 Discussion and Conclusion

This paper aims to characterize and optimize real performance of the KNC. With the comparison between the KNC and dual SNB using real applications from the PARSEC benchmark suite, the impact of various architecture parameters on the KNC performance is quantified. To investigate the hardware optimizations for better real performance of the KNC, a simple emulator for the balance of pipeline usage is developed and several hardware optimizations are designed and evaluated. The evaluation result shows that the proposed hardware optimizations, e.g., an additional set of VPU for the V pipeline, can improve the real performance of the KNC.

If the overhead of an additional set of VPU is not affordable, one possible solution for an additional set of VPU is to split the 512-bit VPU into two identical 256-bit VPU. This solution may not improve the application performance of blackscholes, which can fully utilize the 512-bit VPU, but can prospectively improve other applications.

Acknowledgments. This work is supported in part by the Natural Science Foundation of China (no. 41275098), the National Grand Fundamental Research 973 Program of China (no. 2013CB956603), and the Tsinghua University Initiative Scientific Research Program (no. 20131089356).

References

1. Schmidl, D., Cramer, T., Wienke, S., Terboven, C., Muller, M.S.: Assessing the performance of Openmp programs on the Intel Xeon Phi. In: Euro-Par 2013 Parallel Processing, pp. 547–558 (2013)
2. Semelyanskiy, M., Sewall, J., Kalamkar, D.D., Satish, N., Dubey, P., Astafiev, N., Burylov, I., Nikolaev, A., Mайдanov, S., Li, S., Kulkarni, S., Finan, C.H.: Analysis and optimization of financial analytics benchmark on modern multi- and many-core ia-based architectures. In: SC Companion: High Performance Computing, Networking, Storage and Analysis (2012)
3. Williams, S., Kalamkar, D.D., Singh, A., Deshpande, A.M., Van Straalen, B., Smelyanskiy, M., Almgren, A., Dubey, P., Shalf, J., Oliker, L.: Optimization of geometric multigrid for emerging multi-and manycore processors. In: Conference on High Performance Computing, Networking, Storage and Analysis (2012)
4. Park, J., Tang, P.T.P., Smelyanskiy, M., Kim, D., Benson, T.: Efficient backprojection-based synthetic aperture radar computation with many-core processors. In: Conference on High Performance Computing, Networking, Storage and Analysis (2012)
5. Cramer, T., Schmidl, D., Klemm, M., an Mey, D.: Openmp programming on Intel Xeon Phi coprocessors: an early performance comparison. In: Proceedings of the Many-core Applications Research Community (MARC) Symposium at RWTH Aachen University, pp. 38–44 (2012)
6. Liu, X., Smelyanskiy, M., Chow, E., Dubey, P.: Efficient sparse matrix-vector multiplication on X86-based many-core processors. In: Proceedings of the 27th International ACM Conference on International Conference on Supercomputing (ICS), Eugene, Oregon, USA (2013)
7. Pennycook, S.J., Hughes, C.J., Smelyanskiy, M., Jarvis, S.A.: Exploring Simd for molecular dynamics, using Intel Xeon processors and Intel Xeon Phi coprocessors. In: 2013 IEEE 27th International Symposium on Parallel & Distributed Processing (IPDPS). Boston, MA, USA (2013)
8. Saule, E., Kaya, K., Catalyurek, U.V.: Performance evaluation of sparse matrix multiplication kernels on Intel Xeon Phi. In: Parallel Processing and Applied Mathematics (2013)
9. Gao, T., Lu, Y., Zhang, B., Suo, G.: Using the intel many integrated core to accelerate graph traversal. *Int. J. High Perform. Comput. Appl.* **28**(3), 255–266 (2014)
10. Ravi, N., Yang, Y., Bao, T., Chakradhar, S.: Semi-automatic restructuring of offloadable tasks for many-core accelerators. In: Conference on High Performance Computing, Networking, Storage and Analysis (SC). Denver, USA (2013)
11. Reinders, J.: An overview of programming for Intel Xeon processors and Intel Xeon Phi coprocessors. Intel (2012)

12. Bienia, C., Kumar, S., Singh, J.P., Li, K.: The parsec benchmark suite: characterization and architectural implications. In: Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, pp. 72–81 (2008)
13. Bienia, C., Li, K.: Parsec 2.0: a new benchmark suite for chip-multiprocessors. In: Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation (2009)
14. Molka, D., Hackenberg, D., Schone, R., Mller, M.S.: Memory performance and cache coherency effects on an Intel Nehalem multiprocessor system. In: Proceedings of the 18th International Conference on Parallel Architectures and Compilation Techniques (PACT), pp. 261–270 (2009)
15. Iyer, R., Bhuyan, L.N.: Switch cache: a framework for improving the remote memory access latency of CC-NUMA multiprocessors. In: Proceedings of 5th International Symposium on High-Performance Computer Architecture (1999)
16. Koesterke, L., Boisseau, J., Cazes, J., Milfeld, K., Stanzione, D.: Early experiences with the intel many integrated cores accelerated computing technology. In: Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery. Salt Lake City, Utah, USA (2011)
17. Schulz, K.W., Ulerich, R., Malaya, N., Bauman, P.T., Stogner, R.H., Simmons, C.: Early experiences porting scientific applications to the many integrated core (Mic) platform. In: TACC-Intel Highly Parallel Computing Symposium. Austin, TX (2012)
18. Saini, S., Jin, H., Jespersen, D., Feng, H., Djomehri, J., Arasin, W., Hood, R., Mehrotra, P., Biswas, R.: An early performance evaluation of many integrated core architecture based SGI rackable computing system. In: Conference on High Performance Computing, Networking, Storage and Analysis (2013)
19. Rahman, R.: Intel Xeon Phi Coprocessor Architecture and Tools: the Guide for Application Developers (Experts Voice in Microprocessors). Springer, Berlin (2013)
20. Thiagarajan, S.U., Congdon, C., Naik, S., Nguyen, L.Q.: Intel Xeon Phi Coprocessor Developer’s Quick Start Guide”. <https://software.intel.com/enus/articles/intel-xeon-phi-coprocessor-developers-quick-start-guide>
21. Pentium Processor Family Developers Manual Volume 3: Architecture and Programming Manual. vol. 3, no. 241430 (1995)
22. Fang, J., Varbanescu, A.L., Sips, H., Zhang, L., Che, Y., Xu, C.: An Empirical Study of Intel Xeon Phi. arXiv preprint (2013)
23. Luk, C.K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V.J., Hazelwood, K.: Pin: building customized program analysis tools with dynamic instrumentation. In: ACM SIGPLAN Symposium on Programming Language Design & Implementation (PLDI). Chicago, Illinois, USA (2005)
24. Hazelwood, K., Lueck, G., Cohn, R.: Scalable support for multithreaded applications on dynamic binary instrumentation systems. In: Proceedings of the 2009 International Symposium on Memory Management (ISMM), Dublin, Ireland (2009)
25. Shao, Y.S., Brooks, D.: Energy characterization and instructionlevel energy model of Intels Xeon Phi processor. In: 2013 IEEE International Symposium on Low Power Electronics and Design (2013)
26. Czechowski, K., Lee, V.M., Grochowski, E., Ronen, R., Singhal, R., Vuduc, R., Dubey, P.: Improving the energy efficiency of big cores. In: Proceedings of the 41st Annual International Symposium on Computer Architecture (2014)

Algorithms and Architectures for Parallel Processing
15th International Conference, ICA3PP 2015,
Zhangjiajie, China, November 18-20, 2015,
Proceedings, Part I
Wang, G.; Zomaya, A.; Martinez Perez, G.; Li, K. (Eds.)
2015, LI, 793 p. 380 illus. in color., Softcover
ISBN: 978-3-319-27118-7