Optimization of the N-body Simulation on Intel's Architectures Based on AVX-512 Instruction Set

Enzo Rucci[*a], *Ezequiel Moreno*[b], *Adrián Pousa*[a], *and Franco Chichizola*[a]

[a]*III-LIDI, Facultad de Inforámtica, UNLP - CeAs CICPBA. La Plata (1900), Bs As, Argentina*
[b]*Facultad de Informática, UNLP. La Plata (1900), Bs As, Argentina*

[*] *Corresponding author: erucci@lidi.info.unlp.edu.ar*

ABSTRACT

The N-body simulations have become a powerful tool to test the gravitational interaction among particles, ranging from a few bodies to complete galaxies. Even though N-body has already been optimized on many parallel platforms, there are hardly any studies which take advantage of the latest Intel architectures based on AVX-512 instruction set. This SIMD set was initially supported by Intel's Xeon Phi Knights Landing (KNL) manycore processors launched at 2016. Recently, it has been included in Intel's general-purpose processors too, starting at the Skylake (SKL) server microarchitecture and now in its successor Cascade Lake (CKL). This paper optimizes the all-pairs N-body simulation on both current Intel platforms supporting AVX-512 extensions: a Xeon Phi KNL node and a server equipped with a dual CKL processor. On the basis of a naive implementation, it is shown how the parallel implementation (can) reach, through different optimization techniques, 2355 and 2449 GFLOPS on the Xeon Phi KNL and the Xeon CKL platforms, respectively.

KEYWORDS

N-body ; AVX-512 ; Xeon Phi ; Knights Landing ; Xeon Platinum ; Skylake ; Cascade Lake

1 INTRODUCTION

Nowadays, the scientific community is experimenting with a new revolution on parallel processor technologies in the road to the Exascale. The novelties and enhancements not only involve hardware technologies but also changes in parallel programming models [6]. Beyond that, one of the most important challenges that still remains is how to perform large-scale simulations in a reasonable time using affordable computer systems. In that sense, a deep knowledge of hardware and software optimization is required in these cases to fulfill that purpose.

Physics is one of the areas affected by the above-mentioned challenges due to an increasing number of simulation-based applications demanding High-Performance Computing (HPC) to meet time requirements. One of these applications is the N-body simulation, which approximates a in numeric manner the evolution of a system of bodies where each body interacts with the rest of them [1].

The best-known use of this simulation corresponds to Astrophysics, where each body represents a galaxy or a single star that experiment attraction due to the gravitational force. The N-body simulations have become a powerful tool to test the gravitational interaction among particles, ranging from a few bodies to complete galaxies. However, they are also used in other areas. For example, for protein folding in computational biology [3] or for global illumination in computer graphics [4].

The final authenticated version is available online at https://doi.org/10.1007/978-3-030-48325-8_3

There are different methods to compute the N-body simulation [12]. The simplest way is known as *all-pairs* (or *direct*) and involves evaluating all interactions among all pairs of bodies. It is a brute force method with high computational cost ($O(n^2)$). Due to its complexity, the direct version is only used when the number of bodies is small or moderate at most. When the number of bodies is large, this version is still employed to calculate the interactions between near bodies but combined with a distinct strategy to distant bodies. This is the approach used by advanced methods, like the Barnes-Hut (*O(n log n)*) or the Fast Multipole Method ($O(n)$). Therefore, optimizing the direct version also benefits the other alternatives that makes use of it.

This paper optimizes the all-pairs N-body simulation on Intel's architectures based on the latest AVX-512 instruction set, which allows the exploitation of 512-bit vectorial instructions. This SIMD set was initially supported by Intel's Xeon Phi Knights Landing (KNL) manycore processors but has been recently included in Intel's general-purpose processors too (starting at the Skylake server microarchitecture). The contributions of this work can be seen as an extension of [8], where the optimization process only considered the Xeon Phi KNL architecture. By incorporating Intel's general-purpose processors to the study, this work is able to offer insights regarding both Intel's architectures based on the AVX-512 instructions. Additionally, the power-performance perspective is also addressed.

The rest of the paper is organized as follows. Section 2 introduces the basic concepts of the N-body simulation. Section 3 briefly introduces Intel's architectures based on AVX-512 instruction set. The optimized implementation is described in Section 4. In Section 5, performance results are presented and finally, in Section 6, conclusions and some ideas for future research are summarized.

2 N-BODY SIMULATION

The problem consists in simulating the evolution of a system composed by N bodies during a time-lapse. Given the mass and the initial state (speed and position) for each body, the motion of the system is simulated through discrete instants of time. In each of them, every body experiences an acceleration that arises from the gravitational attraction of the rest, which affects its state.

The simulation basis is supported by Newtonian mechanics [9]. The simulation is performed in 3 spatial dimensions and the gravitational attraction between two bodies C_1 and C_2 is computed using Newton's law of universal gravitation:

$$F = \frac{G \times m_1 \times m_2}{r^2} \qquad \textbf{Equation 1.}$$

where F corresponds to the *magnitude* of the gravitational force between bodies, G corresponds to the gravitational constant [1], m_1 corresponds to the body mass of C_1, m_2 corresponds to the body mass of C_2, and r corresponds to the Euclidean distance [2] between C_1 and C_2.

When N is greater than 2, the gravitational force on a body corresponds to the sum of all gravitational forces exerted by the remaining $N-1$ bodies. The force of attraction leads each body to accelerate and move, according to the Newton's second law, which is given by the following equation:

$$F = m \times a \qquad \textbf{Equation 2.}$$

where F is the force vector, calculated using the magnitude obtained from the equation of gravitation and the direction

[1] Equivalent to 6.674×10^{11}

[2] The Euclidean distance is given by $\sqrt{((x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2)}$, where (x_1, y_1, z_1) is the position of $C1$ and (x_2, y_2, z_2) is the position of C_2.

and sense of the vector going from the affected body to the body exerting the attraction.

Regarding the above equation, it is clear that the acceleration of a body can be calculated by dividing the total force by its mass. During a small time interval dt, the acceleration a_i of the body C_i is approximately constant, so the change in velocity is approximately:

$$dv_i = a_i dt \qquad \text{Equation 3.}$$

The change in body position is the integral of its speed and acceleration over the dt time interval, which is approximately:

$$dp_i = v_i dt + \frac{a_i}{2}dt^2 = (v_i + \frac{dv_i}{2})dt \qquad \text{Equation 4.}$$

This formula uses an integration scheme called Leapfrog [13], in which one half of the position change is because of the old speed while the other half is because of the new speed.

3 INTEL'S ARCHITECTURES BASED ON AVX-512 INSTRUCTION SET

AVX-512 is a set of new 512-bit SIMD x86 instructions presented by Intel in 2013. It was initially supported by Xeon Phi KNL manycore processors but has been recently included in the general-purpose sector too (starting at the Skylake-X microarchitecture). A single AVX-512 instruction can pack eight double-precision multiply-add operations (16 FLOPS) or 16 single-precision multiply-add operations (32 FLOPS).

3.1 *Intel Knights Landing*

KNL is the second generation of the Intel Xeon Phi family and the first one capable of operating as a standalone processor. The KNL architecture features up to 72 cores based on the Intel's Atom microarchitecture, which are organized in *Tiles*. A Tile includes 2 cores and is interconnected with the rest of them by a 2D mesh. Additionally, each Tile features 2 Vector Processing Units (VPUs) and a shared L2 cache of 1 MB. Besides supporting the AVX-512 SIMD set, these VPUs are also compatible with prior vector sets such as SSE*x* and AVX*x* [7].

The 2D mesh can be configured in three different cluster operating modes: 1) *All-to-All*; 2) *Quadrant/Hemisphere*; and 3) *SNC-4/SNC-2*. The main difference between these modes is whether the cores will access in UMA or NUMA manner to a particular memory.

The KNL architecture also features a 3D-stacked DRAM known as Multi-Channel DRAM (MCDRAM). This special memory offers 3 operating modes: 1) *Cache*, where the MCDRAM works as an L3 cache; 2) *Flat*, where the MCDRAM has a physical addressable space offering the highest bandwidth and lowest latency; and 3) *Hybrid*, where this memory is divided in two parts: one part in *Cache mode* and one in *Flat mode*. From a performance point of view, Flat mode can achieve better results. However, programmer intervention may be required, as opposed to the Cache mode [2].

3.2 *Intel Cascade Lake*

Cascade Lake (CKL) is the latest microarchitecture presented by Intel for general-purpose processors. This microarchitecture is an optimization of the previous Skylake (SKL), which was the first to support the AVX-512 SIMD set in the general-purpose segment. As the SKL case, Intel distinguishes between two CKL processor versions: *client* and *server*, being the latter considerably larger than the former.

CKL server processors present up to 56 cores, which share a last level cache of (up to) 80 Mb. In a similar way to the

KNL case, CKL server cores are organized in Tiles that are interconnected by a 2D mesh. Additionally, these cores also feature AVX-512 extensions and support previous vector instructions such as SSE*x* and AVX*x*.

4 OPTIMIZATION OF THE N-BODY SIMULATION

This section describes the optimization techniques that were applied to the parallel implementation for both target architectures.

4.1 *Naive Implementation*

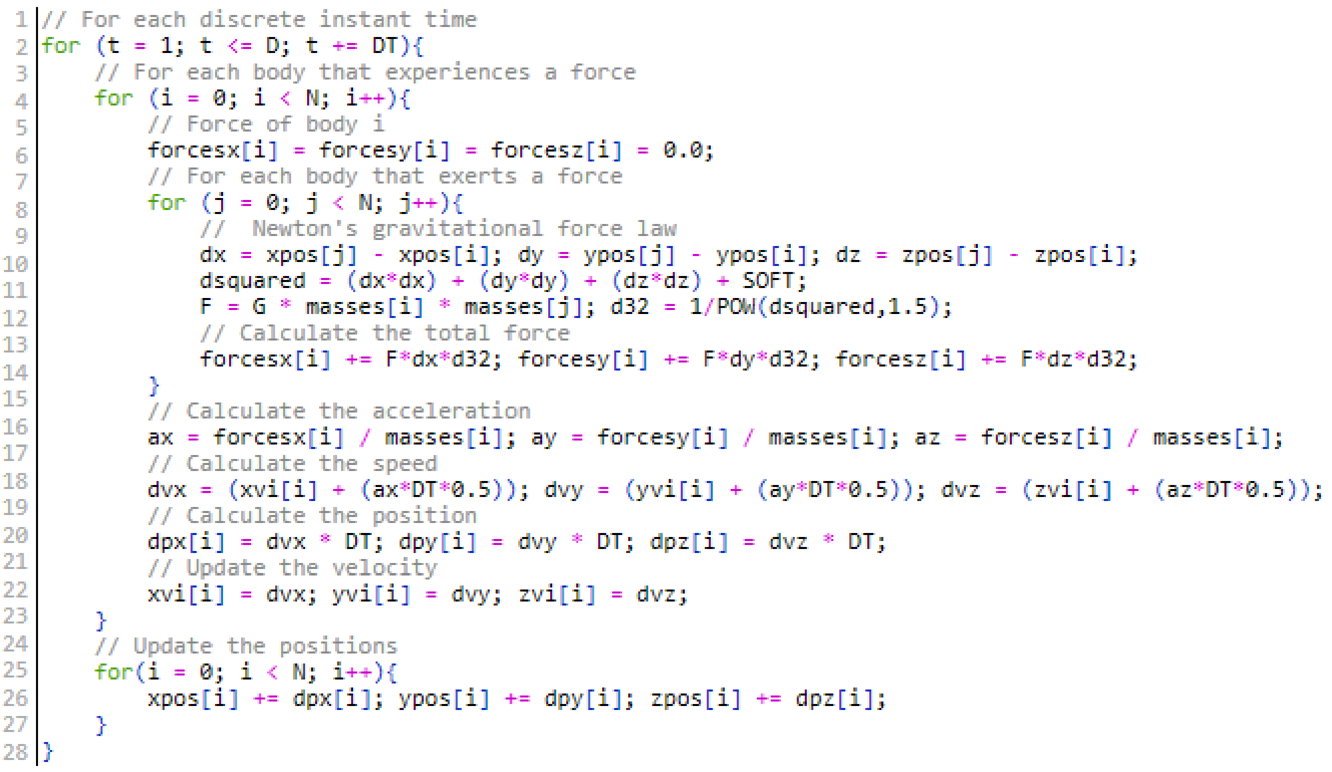
Initially, a naive implementation was developed, which served as a baseline in order to assess the improvements introduced by the subsequent optimization techniques. The pseudo-code of the naive implementation is shown in Fig. 1.

4.2 *Multi-threading*

The first optimization consists in introducing thread-level parallelism through OpenMP directives. The loops on lines 4 and 25 are parallelized by inserting *parallel for* directives. In this way, the dependencies of the problem are guaranteed since one body cannot move until the rest have finished calculating their interactions. And they cannot advance either to the next step until the others have completed the current step. Finally, the *static* option for the *schedule* clause permits an equal distribution of bodies among the threads, achieving a minimum cost load balance.

4.3 *Vectorization*

The ICC optimization report makes possible to identify loops that are automatically vectorized. Given its feedback, it was possible to know that the compiler detects false dependencies in some operations, disabling the generation of SIMD instructions. Consequently, in order to guarantee the vectorization of operations, a guided approach was implemented through the use of the OpenMP 4.0 *simd* directive. In particular, the vectorized loops are those of lines 8 and 25, the latter being combined with the *parallel for* directive (as mentioned in the previous section). Finally, to avoid the overhead of potential misaligned memory accesses, the data was aligned to 64-bytes and the *aligned* clause was added to the



```
 1 // For each discrete instant time
 2 for (t = 1; t <= D; t += DT){
 3     // For each body that experiences a force
 4     for (i = 0; i < N; i++){
 5         // Force of body i
 6         forcesx[i] = forcesy[i] = forcesz[i] = 0.0;
 7         // For each body that exerts a force
 8         for (j = 0; j < N; j++){
 9             //  Newton's gravitational force law
10             dx = xpos[j] - xpos[i]; dy = ypos[j] - ypos[i]; dz = zpos[j] - zpos[i];
11             dsquared = (dx*dx) + (dy*dy) + (dz*dz) + SOFT;
12             F = G * masses[i] * masses[j]; d32 = 1/POW(dsquared,1.5);
13             // Calculate the total force
14             forcesx[i] += F*dx*d32; forcesy[i] += F*dy*d32; forcesz[i] += F*dz*d32;
15         }
16         // Calculate the acceleration
17         ax = forcesx[i] / masses[i]; ay = forcesy[i] / masses[i]; az = forcesz[i] / masses[i];
18         // Calculate the speed
19         dvx = (xvi[i] + (ax*DT*0.5)); dvy = (yvi[i] + (ay*DT*0.5)); dvz = (zvi[i] + (az*DT*0.5));
20         // Calculate the position
21         dpx[i] = dvx * DT; dpy[i] = dvy * DT; dpz[i] = dvz * DT;
22         // Update the velocity
23         xvi[i] = dvx; yvi[i] = dvy; zvi[i] = dvz;
24     }
25     // Update the positions
26     for(i = 0; i < N; i++){
27         xpos[i] += dpx[i]; ypos[i] += dpy[i]; zpos[i] += dpz[i];
28     }
29 }
```

Figure 1. Pseudo-code of the naive implementation.

simd directives.

4.4 *Block Processing*

Block processing is implemented to exploit data locality. The pseudo-code of the parallel block implementation is shown in Fig. 2. Regarding the naive implementation, the *i*-loop (line 4) is divided and placed inside the *j*-loop (line 8). Consequently, one loop is replaced by two others: a loop that iterates over all blocks (line 5) and an inner loop that iterates over the bodies of each block (line 12). This minimizes the traffic between cache and main memory by increasing the number of times that each data is used in the innermost loop.

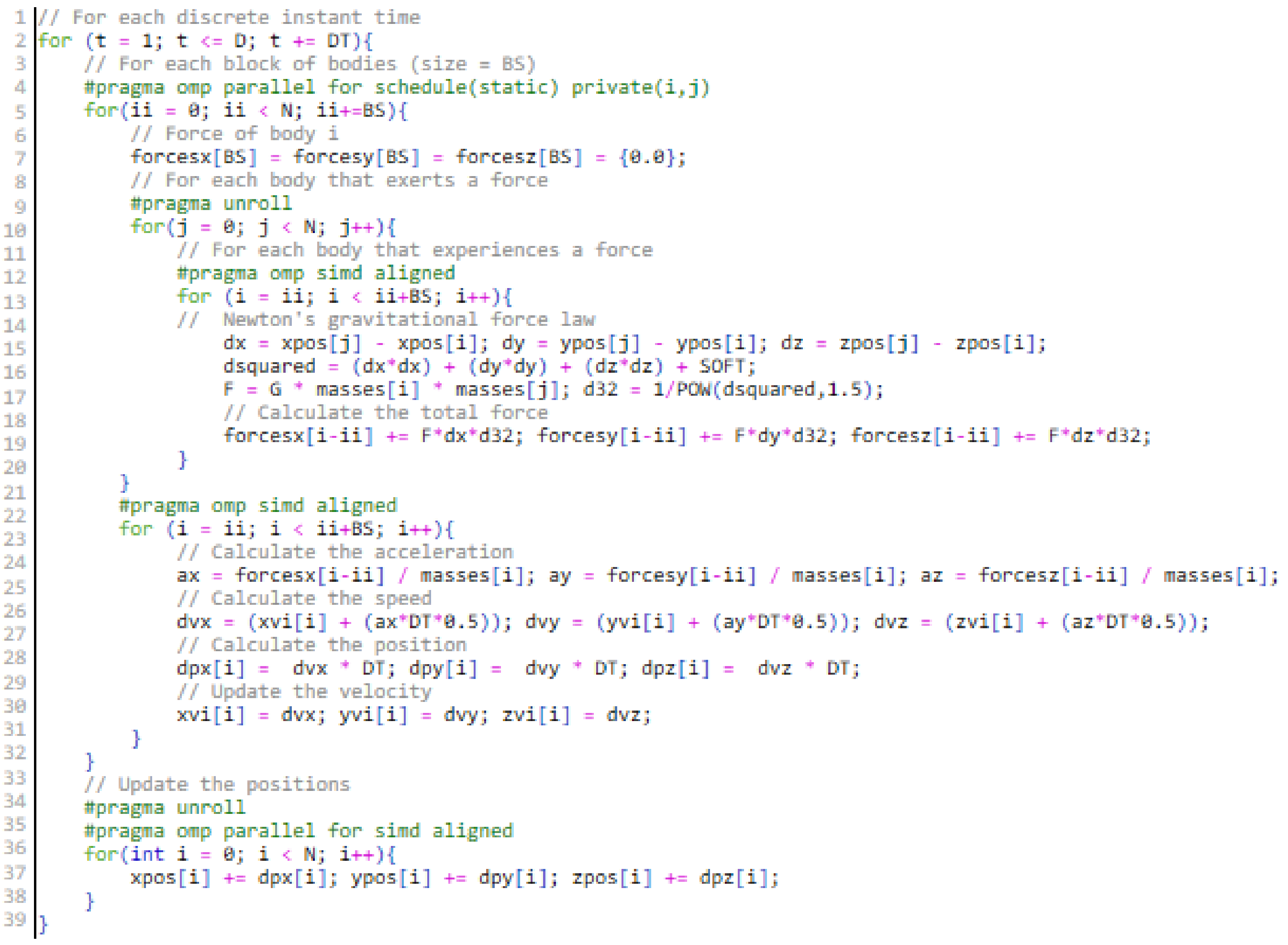
4.5 *Loop Unrolling*

Loop unrolling is another optimization technique that can improve the performance of a program. In particular, it was found beneficial to completely unroll the innermost loop of the implementation presented in Fig. 2 (line 9), in addition to the one that will update the body positions later (line 35).

5 EXPERIMENTAL RESULTS

All tests were carried out using the platforms described in Table 1 [3]. To generate explicit AVX2 instructions, the flag `-xAVX2` was used in both servers. In similar manner, the flags `-xMIC-AVX512` and `-xCORE-AVX512 -qopt-zmm-usage=high`

[3]The Xeon Phi server was used in All-to-all cluster mode and the MCDRAM memory was set to Flat mode. Because DDR memory modules present different sizes, it was impossible to configure the processor in a different mode.



```
 1 // For each discrete instant time
 2 for (t = 1; t <= D; t += DT){
 3     // For each block of bodies (size = BS)
 4     #pragma omp parallel for schedule(static) private(i,j)
 5     for(ii = 0; ii < N; ii+=BS){
 6         // Force of body i
 7         forcesx[BS] = forcesy[BS] = forcesz[BS] = {0.0};
 8         // For each body that exerts a force
 9         #pragma unroll
10         for(j = 0; j < N; j++){
11             // For each body that experiences a force
12             #pragma omp simd aligned
13             for (i = ii; i < ii+BS; i++){
14             //  Newton's gravitational force law
15                 dx = xpos[j] - xpos[i]; dy = ypos[j] - ypos[i]; dz = zpos[j] - zpos[i];
16                 dsquared = (dx*dx) + (dy*dy) + (dz*dz) + SOFT;
17                 F = G * masses[i] * masses[j]; d32 = 1/POW(dsquared,1.5);
18                 // Calculate the total force
19                 forcesx[i-ii] += F*dx*d32; forcesy[i-ii] += F*dy*d32; forcesz[i-ii] += F*dz*d32;
20             }
21        }
22        #pragma omp simd aligned
23        for (i = ii; i < ii+BS; i++){
24             // Calculate the acceleration
25             ax = forcesx[i-ii] / masses[i]; ay = forcesy[i-ii] / masses[i]; az = forcesz[i-ii] / masses[i];
26             // Calculate the speed
27             dvx = (xvi[i] + (ax*DT*0.5)); dvy = (yvi[i] + (ay*DT*0.5)); dvz = (zvi[i] + (az*DT*0.5));
28             // Calculate the position
29             dpx[i] =  dvx * DT; dpy[i] =  dvy * DT; dpz[i] =  dvz * DT;
30             // Update the velocity
31             xvi[i] = dvx; yvi[i] = dvy; zvi[i] = dvz;
32         }
33     }
34     // Update the positions
35     #pragma unroll
36     #pragma omp parallel for simd aligned
37     for(int i = 0; i < N; i++){
38         xpos[i] += dpx[i]; ypos[i] += dpy[i]; zpos[i] += dpz[i];
39     }
40 }
```

Figure 2. Pseudo-code of the optimized parallel implementation.

Table 1. Experimental platforms used in the tests.

Platform	Processor	Memory	OS + Compiler
Xeon Phi KNL	Intel Xeon Phi 7250 64-core 1.30GHz (4 hw threads per core)	16GB HBW + 192GB DDR4	Ubuntu 16.04.3 LTS + ICC (v19.0.0.117)
Xeon Platinum CKL	2×Intel Xeon Platinum 8276 28-core 2.20Ghz (2 hw threads per core)	256GB DDR4	Ubuntu 18.04.3 LTS + ICC (v19.1.0.166)

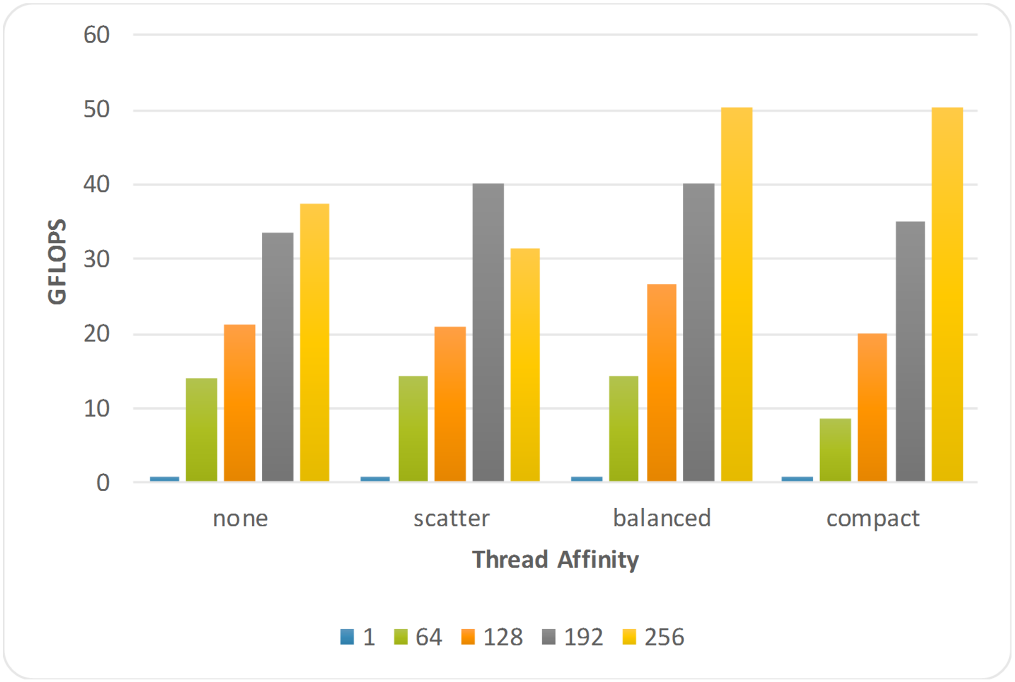


Figure 3. Performance for the different affinity types when N=65536 and the number of threads varies on the Xeon Phi 7230.

were employed to exploit AVX-512 extensions in the Xeon Phi and in the Xeon Platinum platforms, respectively. Also, the flag `-fp-model fast=2` was enabled to accelerate floating-point operations. Regarding the Xeon Phi server, the *numactl* utility was used to take advantage of its MCDRAM memory (no source code modification is required). Finally, different workloads were tested in both platforms: N = {65536, 131072, 262144, 524288} [4] .

5.1 *Performance Results on the Intel Xeon Phi 7230*

The metric GFLOPS (billion floating-point operations per second) was selected as performance metric and is calculated by using the formula $GFLOPS = \frac{20 \times N^2 \times I}{t \times 10^9}$, where N is the number of bodies, I is the number of simulation steps, t is the runtime in seconds, and 20 represents the amount of floating-point operations required for each interaction [5].

Figure 3 shows the performance for the different affinity types when *N=65536* and the number of threads varies. It can be observed that enabling multi-threading significantly improves the performance, especially when all hardware threads are active (except with *scatter* affinity). Regarding affinity, it can be seen that it is advisable to select one of the available strategies instead of delegating the distribution to the operating system (*none*). Unlike *scatter*, *balanced* and *compact* guarantee the proximity among OpenMP threads with consecutive identifiers, minimizing in this way the data communication that each thread requires [6].

[4]The number of simulation steps was set to $I = 100$.
[5]A widely accepted convention in the available literature for this problem.
[6]Since all processor cores are in the same package, *balanced* and *compact* produce the same distribution when using all hardware threads

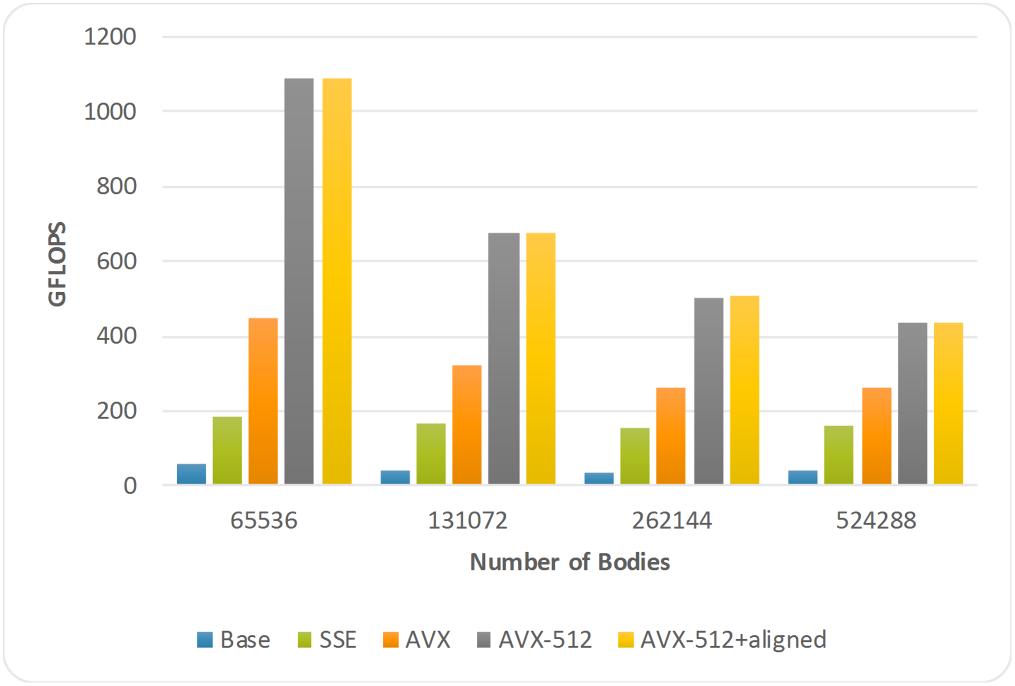


Figure 4. Performance for the different SIMD sets used when the number of bodies varies on the Xeon Phi 7230.

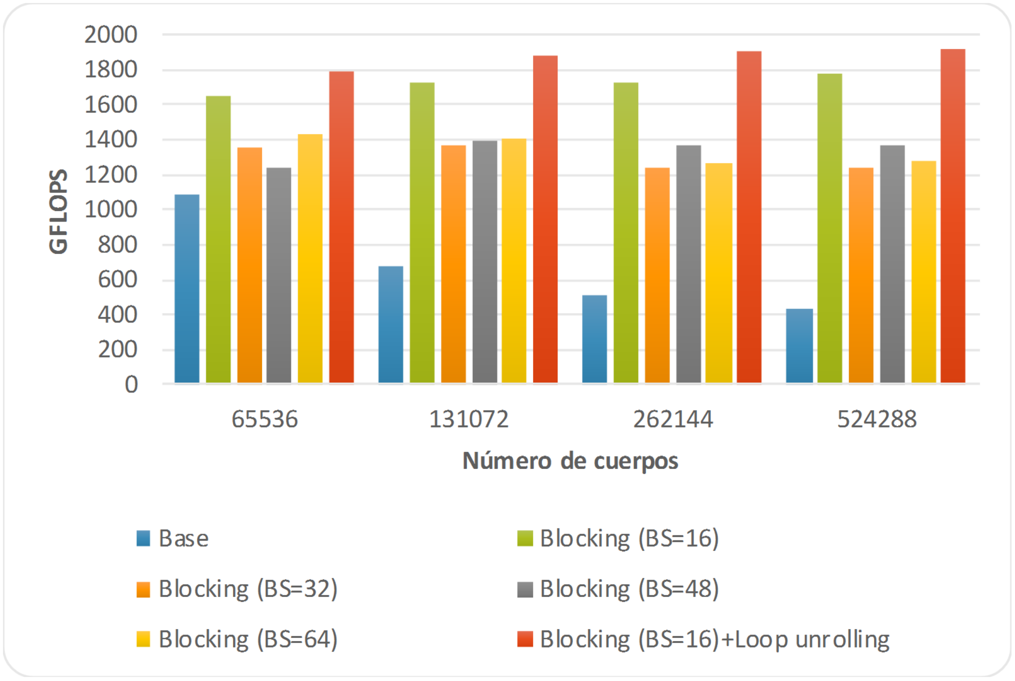


Figure 5. Performance for the block processing and loop unrolling techniques when the number of bodies varies on the Xeon Phi 7230.

As it was mentioned in Section 4.3, the compiler detects false dependencies in that loop and it is not able to generate SIMD binary code by itself. Fig. 4 presents the performance for the different SIMD sets used varying the number of bodies. By adding the corresponding *simd* constructs, the compiler generates SSE4.1 instructions, reaching a 3.9× speedup. Re-compiling the code including the `-xAVX2` and `-xMIC-AVX512` flags force the compiler to generate AVX2 and AVX-512 SIMD instructions, respectively. AVX2 extensions accelerated the previous version by a factor of 7.4× while AVX-512 instructions achieved a speedup of 15.1×. Therefore, it is clear that this application benefits from wider vectorial instructions. Additionally, no performance improvement was noted from memory alignment.

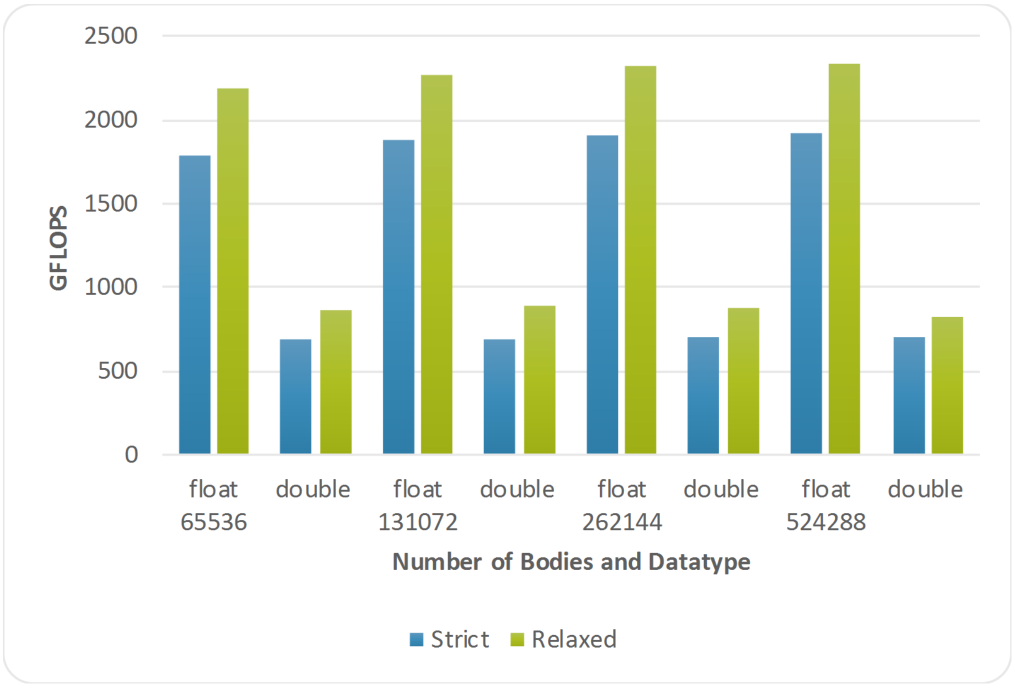


Figure 6. Performance for precision relaxation when both the dataytpe and the number of bodies varies on the Xeon Phi 7230.

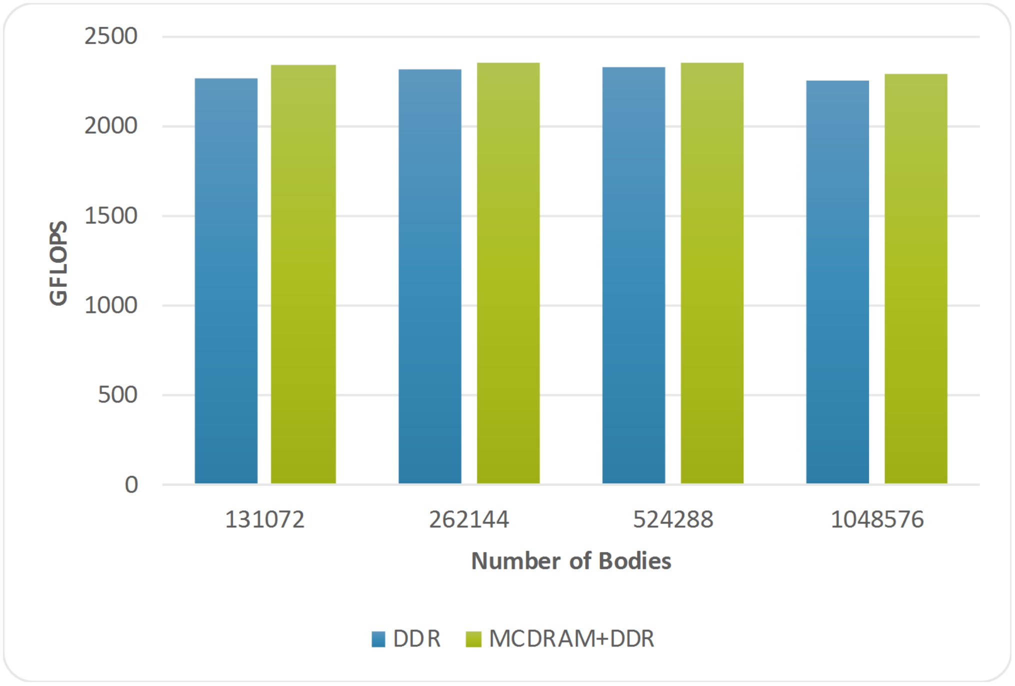


Figure 7. Performance for MCDRAM exploitation when the number of bodies varies on the Xeon Phi 7230.

As it can be seen in Fig. 5, the block processing technique significantly improves performance and its benefit enlarges as the number of bodies increases. In particular, this technique yields an average speedup of $2.9\times$ and a peak of $4.1\times$ (BS = 16). Besides, if the loop unrolling is applied, this technique leads to a 9% performance improvement.

Fig. 6 presents the performance for precision relaxation when both the datatype and the number of bodies varies. Performance increases 22% when using the compiler optimization to relax precision. On the contrary, performance drops by approximately 60% when doubling numeric precision (double datatype).

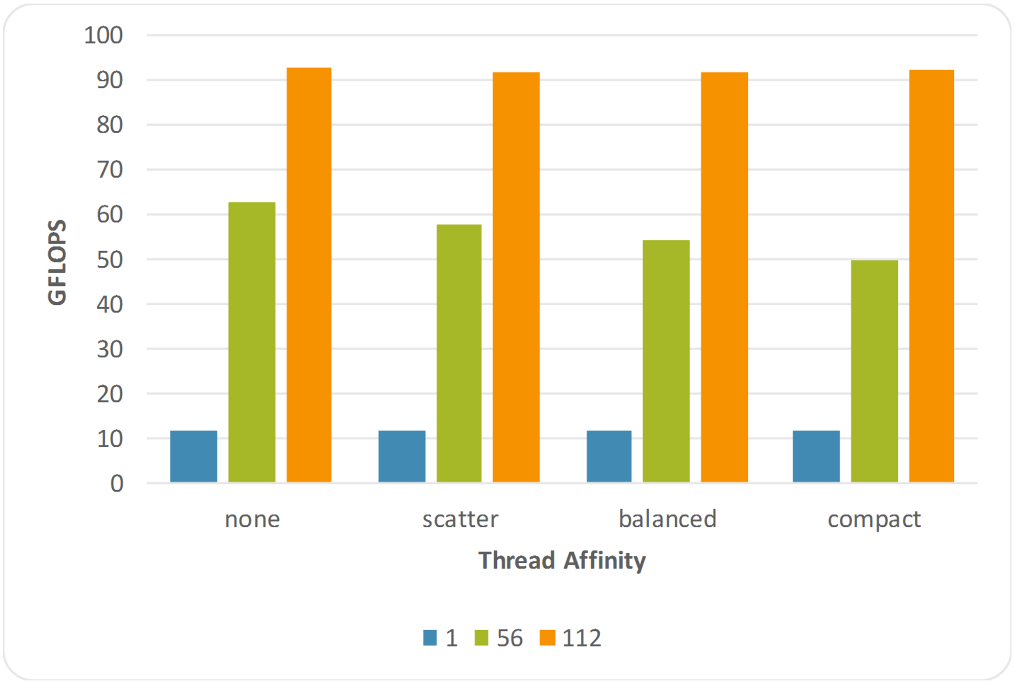


Figure 8. Performance for the different affinity types used when $N = 65536$ and the number of threads varies on the Xeon Platinum 8276.

Finally, from Fig. 7 it can be seen that the performance remains (almost) constant as the number of bodies increases, obtaining a maximum of 2355 GFLOPS. Also, the MCDRAM exploitation produces a free, small performance improvement (2%). This behavior is similar to the one observed in [10] and is related to the fact that memory latency is more influential than bandwidth in this application [7].

5.2 *Performance Results on the Intel Xeon Platinum 8276*

As in the previous subsection, the performance metric selected is GFLOPS. Fig. 8 shows the performance for the different affinity types used when N=*65536* and the number of threads varies. A similar behavior was observed in the Xeon Phi case: multi-threading improves performance and the best results are obtained when all hardware threads are active. However, two small differences can be mentioned. The first one is that the naive implementation reaches higher GFLOPS in this platform. As the Xeon Platinum CKL processor is designed as a general-purpose device, serial codes perform better than in Xeon Phi architecture. The second difference is that thread affinity does not play an influential role in this platform when hyper-threading is enabled.

Fig. 9 presents the performance for the different SIMD sets used when the number of bodies varies. As in the Xeon Phi architecture, higher GFLOPS are obtained when using wider vectorial instructions. Average speedups of 3.7×, 7.2× and 10.5× are achieved when using SSE, AVX and AVX-512 instructions, respectively. In addition, memory alignment does not affect performance.

The performance for the block processing and loop unrolling techniques when the number of bodies varies is illustrated in Fig. 10. The blocking version (BS = 16) runs 2.1× faster than the non-blocking one. The improvement factor is smaller than in the Xeon Phi case due to the larger cache subsystem of the CKL microarchitecture. Still, it represents a key factor to improve performance in this implementation. Moreover, 2% of additional GFLOPS are obtained through loop unrolling.

Lastly, Fig. 11 shows the performance for precision relaxation when both the dataytpe and the number of bodies varies.

[7]The latencies of DDR4 and MCDRAM memories are similar

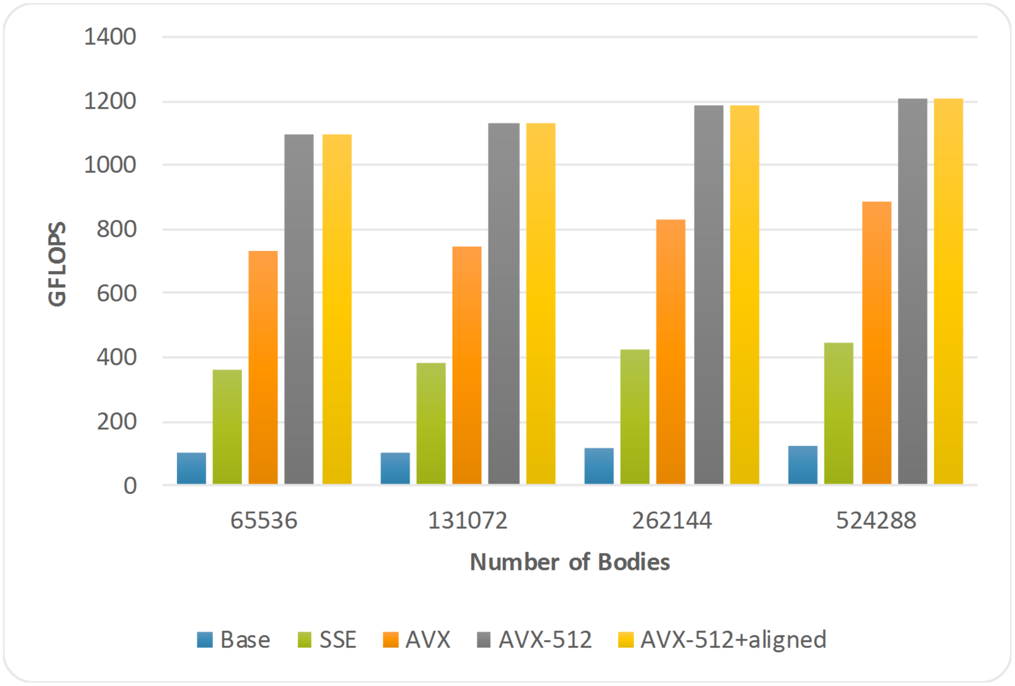


Figure 9. Performance for the different SIMD sets used when the number of bodies varies on the Xeon Platinum 8276.

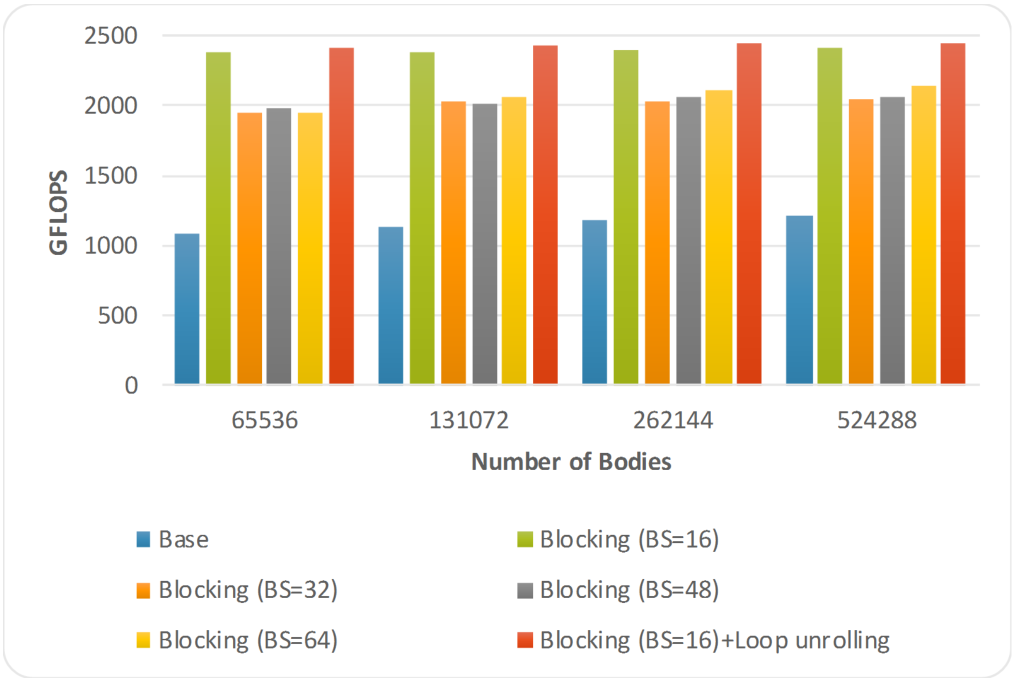


Figure 10. Performance for the block processing and loop unrolling techniques when the number of bodies varies on the Xeon Platinum 8276.

As opposed to the Xeon Phi case, no performance improvement is obtained in relaxed mode. By analyzing the assembly code, it was observed that the compiler generated the same binary code for both versions in this architecture (with or without floating-point optimization). Furthermore, doubling the numerical precision reduces performance by approximately 78%, a larger penalty compared to the Xeon Phi architecture. Considering the increase in the number of bodies, the performance remains (almost) constant, obtaining a peak of 2449 GFLOPS.

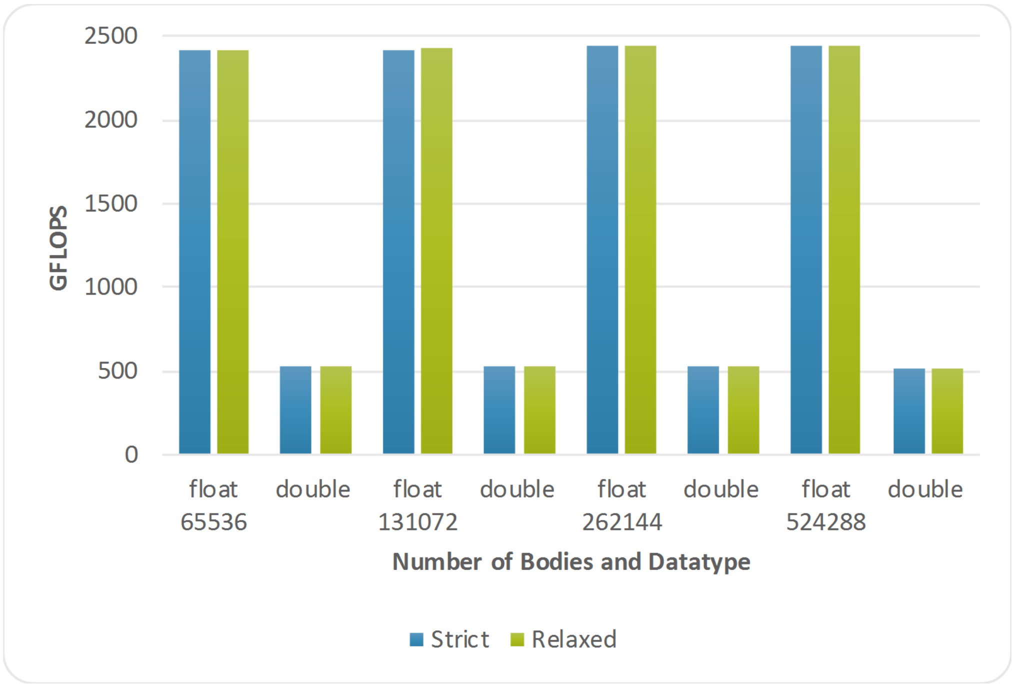


Figure 11. Performance for precision relaxation when both the dataytpe and the number of bodies varies on the Xeon Platinum 8276.

5.3 *Performance and Power Efficiency Comparison between Platforms*

Currently, in the HPC community not only performance matters but also energy efficiency does. Table 2 presents power efficiency ratios considering the GFLOPS peak performance and the Thermal Design Power (TDP) of each platform. Considering the single-precision case (SP), while Xeon Platinum CKL server reaches more GFLOPS than the Xeon Phi platform, the associated high power consumption results in lower GFLOPS/Watt quotient. On the contrary, the Xeon Phi platform becomes the best option in both aspects in the double-precision scenario (DP).

Table 2. Performance and power efficiency comparison between platforms.

Platform	GFLOPS (peak)		TDP (Watt)	GFLOPS/Watt	
	SP	DP		SP	DP
Xeon Phi KNL	2355	883	215	10.95	4.1
Xeon Platinum CKL	2449	532	2×165	7.42	1.61

6 RELATED WORKS

The acceleration of the N-body simulation has been widely studied in the literature. However, few works considered the Xeon Phi architecture, using mostly the first generation of this family (KNC) [5, 11, 12]. Regarding KNL, the work of Vladimirov and Asai [10] can be mentioned, which has some similarities and differences with the present investigation. As in this paper, the authors study the parallelization of the all-pairs version of the simulation. They show how it is possible to improve the performance through different optimizations, although the final implementation reaches a higher peak performance (2875 GFLOPS). Unlike this study, the work prioritizes the optimizations for KNL, simplifying some calculations of the simulation, such as employing a simpler integration scheme (requiring fewer operations) and just computing a single simulation step. By using the same performance metric [8], they overestimate the FLOPS obtained. In relation to the performance analysis, several additional aspects were considered in this paper, such as the number of threads and their affinity, the different SIMD instruction sets, the tuning of the block size, the impact of doubling numerical precision, and the variation in the number of bodies.

[8]They also assume that 20 floating point operations are performed per interaction but the real number is lower because of the simpler integration scheme (fewer operations are computed).

With respect to Intel's general-purpose processors based on AVX-512 instructions, to the best of the authors' knowledge, this is the first work studying the all-pairs N-body optimization on these processors.

Finally, from the energy efficiency point of view, Zecena et al. [14] evaluated the performance and energy efficiency of different N-body codes on CPUs and GPUs. In this work, they showed that GPU-based implementations can boost the performance and energy efficiency by orders of magnitude compared to CPUs. No power-performance study considering Intel's architectures based on AVX-512 instructions was found.

7 CONCLUSIONS AND FUTURE WORK

This paper focused on the optimization of the all-pairs N-body simulation on Intel's architectures based on the AVX-512 instruction set. On the basis of a naive implementation, it was shown how the parallel implementation reached, through different optimization techniques, 2355 and 2449 GFLOPS on the Xeon Phi and the Xeon Platinum platforms, respectively. Among the main conclusions of this research, it can be mentioned that:

- In general, a higher number of threads bettered performance. Regarding affinity, significant differences were found among the different options in the Xeon Phi platform, so this is a factor that should not be omitted when running a parallel application.
- Vectorization represented a fundamental factor to improve performance. In that sense, it was possible to achieve almost linear speedups with regard to the number of simultaneous operations of each SIMD set, at a low programming cost.
- The exploitation of the data locality through block processing was another key aspect to obtain high-performance. The impact of this optimization was larger in the Xeon Phi platform due to its smaller cache subsystem.
- Accuracy is usually an important aspect of N-body simulations. Double precision reduces performance to less than the half, so it must be used only when required.
- The MCDRAM memory usage did not provide large performance improvements in this application. However, no source code modification was required to reach it.
- From a power efficiency perspective, the Xeon Phi platform resulted as the best option in both single and double precision computations.

As future work, two possible research lines can be mentioned:

- Implementing advanced methods for this simulation considering the results obtained with all-pairs version.
- Extending the power-performance comparison in order to include GPUs since they are the dominant accelerators today.

REFERENCES

[1] Andrews, G.R.: Foundations of Multithreaded, Parallel, and Distributed Programming. Addison-Wesley (2000)

[2] Codreanu, V., Rodríguez, J., Saastad, O.W.: Best Practice Guide - Knights Landing (2017), `http://www.prace-ri.eu/IMG/pdf/Best-Practice-Guide-Knights-Landing.pdf`

[3] Freddolino, P.L., Harrison, C.B., Liu, Y., Schulten, K.: Challenges in protein-folding simulations. Nature Physics **6**(10), 751–758 (2010). https://doi.org/10.1038/nphys1713, `https://doi.org/10.1038/nphys1713`

[4] Goradia, R.: Global illumination for point models. Fourth Annual Progress Seminar, 2008 (2008), `https://tinyurl.com/y5nqel39`

[5] Lange, B., Fortin, P.: Parallel dual tree traversal on multi-core and many-core architectures for astrophysical n-body simulations. In: Silva, F., Dutra, I., Santos Costa, V. (eds.) Euro-Par 2014 Parallel Processing. pp. 716–727. Springer International Publishing, Cham (2014)

[6] Prat, R., Colombet, L., Namyst, R.: Combining task-based parallelism and adaptive mesh refinement techniques in molecular dynamics simulations. In: Proceedings of the 47th International Conference on Parallel Processing. ICPP 2018, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3225058.3225085, `https://doi.org/10.1145/3225058.3225085`

[7] Reinders, J., Jeffers, J., Sodani, A.: Intel Xeon Phi Processor High Performance Programming Knights Landing Edition. Morgan Kaufmann Publishers Inc., Boston, MA, USA (2016)

[8] Rucci, E., Moreno, E., Camilo, M., Pousa, A., Chichizola, F.: Simulación de N Cuerpos Computacionales sobre Intel Xeon Phi KNL. In: Actas del XXV Congreso Argentino de Ciencias de la Computación (CACIC 2019). pp. 194–204 (2019)

[9] Tipler, P.: Physics for Scientists and Engineers: Mechanics, Oscillations and Waves, Thermodynamics. Freeman and Co (2004)

[10] Vladimirov, A., Asai, R.: N-body simulation. In: Intel Xeon Phi Processor High Performance Programming Knights Landing Edition. Morgan-Kaufmann (2016)

[11] Vladimirov, A., Karpusenko, V.: Test-driving intel xeon phi coprocessors with a basic n-body simulation. Tech. rep., Stanford University and Colfax International (3 2013), `https://tinyurl.com/y5vtj34a`

[12] Yokota, R., Abduljabbar, M.: N-body methods. In: High Performance Parallelism Pearls - Multicore and Many-core Programming Approaches, chap. 10, pp. 175–183. Morgan-Kaufmann, 1 edn. (2015)

[13] Young, P.: The leapfrog method and other âĂIJsymplecticâĂİ algorithms for integrating newton's laws of motion. Tech. rep., Physics Department, University of California, USA (4 2014), `https://young.physics.ucsc.edu/115/leapfrog.pdf`

[14] Zecena, I., Burtscher, M., Jin, T., Zong, Z.: Evaluating the performance and energy efficiency of n-body codes on multi-core cpus and gpus. In: 2013 IEEE 32nd International Performance Computing and Communications Conference (IPCCC). pp. 1–8 (Dec 2013). https://doi.org/10.1109/PCCC.2013.6742789