

An Efficient SIMD Implementation of Pseudo-Verlet Lists for Neighbour Interactions in Particle-based Codes



www.swiftsim.com

James S. Willis | Matthieu Schaller, Pedro Gonnet | Institute for Computational Cosmology, Durham University
Contact: james.s.willis@durham.ac.uk

Abstract

We present a SIMD implementation of the pseudo-Verlet list neighbour search algorithm^[1]. We have applied our strategy to the SPH cosmological code SWIFT^[2]. We report an overall speed-up over the scalar version of 2.21x, 2.41x and 3.65x for AVX, AVX2 and AVX-512 instruction sets.

Computing Particle Interactions

The most time-consuming part of particle-based simulations^[3,4] is calculating short-range pair-wise interactions. This involves computing distances between particle pairs and checking if they lie within a cut-off radius, h , of each other. A standard approach for efficient neighbour finding is to use a mesh of cells with edge lengths that match the particle cut-off radius. Particles are then interacted within pairs of neighbouring cells (see Fig. 1). However, this is still a suboptimal solution as the majority of particles in neighbouring cells are out of range. We improve this strategy using the pseudo-Verlet list algorithm illustrated in Fig. 2

Data Layout Strategy

SIMD performance is highly dependent of the input data's memory layout. Therefore, we use a *software cache* that stores the data we require contiguously. This improves memory access when populating SIMD vectors.

We create the cache using a structure-of-arrays data layout for the sections of the code that contain SIMD instructions. We filter the data and only store what is loaded into the SIMD vectors. Although creating and filling the cache incurs an additional computational cost, the improved memory access patterns more than make up for this overhead.

SIMD Implementation

The overall vectorisation strategy involves picking one particle, $p_i^{(a)}$, from cell (a) and interacting it with a vector of particles, $p_j^{(b)}$, from cell (b) that are in range. The contributions from $p_j^{(b)}$ that are not within range are masked out.

We also use a *particle cache* that contains the subset of particle properties that we need for an interaction. The particles are cached in order along the cell-pair axis (see Fig. 2), using the sorted list of particle indices from the Verlet list. We only store the particles that are in range.

When implemented in the SWIFT cosmological code using the AVX, AVX2 and AVX-512 instruction sets, this algorithm reached speed-ups of 2.21x, 2.41x and 3.65x when compared to a scalar version (see Fig. 3).

We gain a greater performance improvement than a naive N^2 implementation that was sped-up by an ideal factor of 8 with SIMD instructions. This demonstrates the importance of using clever algorithms and investing the effort to modify them for improved SIMD vectorisation.

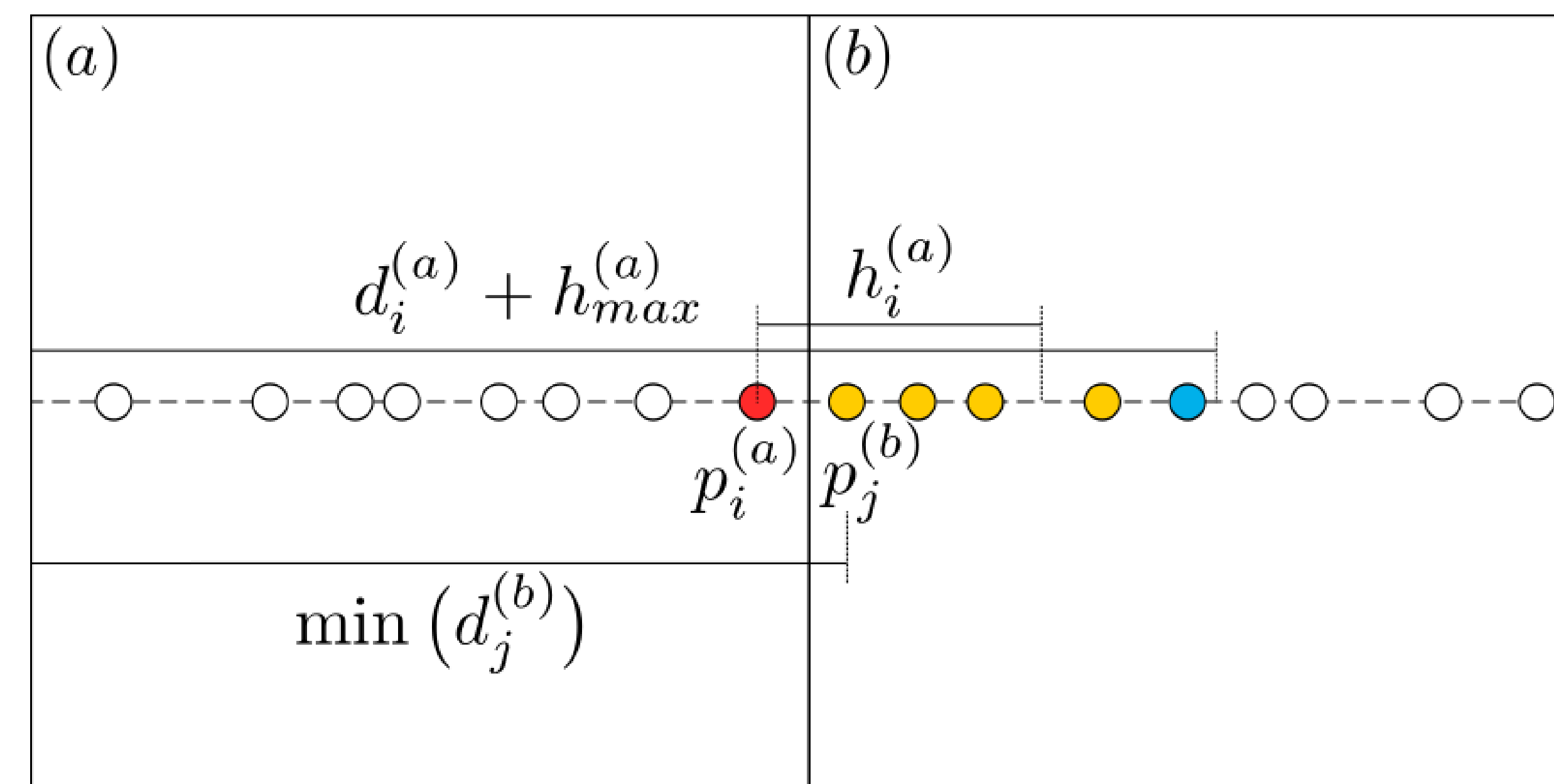


Figure 2: Cell-pair interactions using a sorted pseudo-Verlet list. 1) Project the particles onto the 'cell-pair axis' that joins the centres of the two cells, then sort them along that axis. 2) Calculate the separation between particles $p_i^{(a)}$ in the left-hand cell (a) and particles $p_j^{(b)}$ in the right hand cell (b) that lie within h_i , the cut-off radius of $p_i^{(a)}$, on the sorted axis. 3) If any particles $p_j^{(b)}$ satisfy the criterion $\|p_i^{(a)} - p_j^{(b)}\|_2 < h_i^{(a)}$, then compute the interaction.

```
for (int i = 0; i < count_a; i++)
    for (int j = 0; j < count_b; j++)
        r = particle_dist(parts_a[i], parts_b[j]);
        if (r < h)
            // Compute interaction.
```

Code 1: Naive implementation, $\mathcal{O}(N^2)$. Simple iteration over neighbours.

```
for (int i = 0; i < count_a; i++)
    for (int j = 0; j < count_b; j++)
        if (&& dist_b[j] < dist_a[i] + h; k++)
            r = particle_dist(parts_a[index_a[i]], parts_b[index_b[j]]);
            if (r < h)
                // Compute interaction.
```

Code 2: Pseudo-Verlet list iteration over neighbours.

```
int first_a = count_a;
while (first_a > 0 && dist_a[index_a[first_a - 1]] + h_max_a > dist_b[0])
    first_a--;
```

Code 3: Find the left-most particle in cell a that interacts with any particle in cell b .

```
int temp = 0;
while (temp <= count_b &&
      (dist_a[first_a] + parts_a[index_a[first_a]].h > dist_b[temp])) temp++;
max_index_a[first_a] = temp;
for (i = first_a + 1; i < count_a; i++)
    temp = max_index_a[i - 1];
    while (temp <= count_b &&
          (dist_a[i] + parts_a[index_a[i]].h > dist_b[temp])) temp++;
    max_index_a[i] = temp;
```

Code 4: For each particle in cell a , find the index (max_index_a) of the farthest possible interacting particle in cell b .

```
for (int i = first_a; i < count_a; i++)
    for (int j = 0; j <= max_index_a[i]; j += SIMD_LENGTH)
        simd_r = particle_dist_simd(i, j);
        mask = (simd_r < SIMD(parts_a[index_a[i]].h));
        if (ANY(mask))
            particle_interact_simd(i, j, mask);
```

Code 5: Use the calculated loop bounds to find the particle neighbours and perform the interaction.

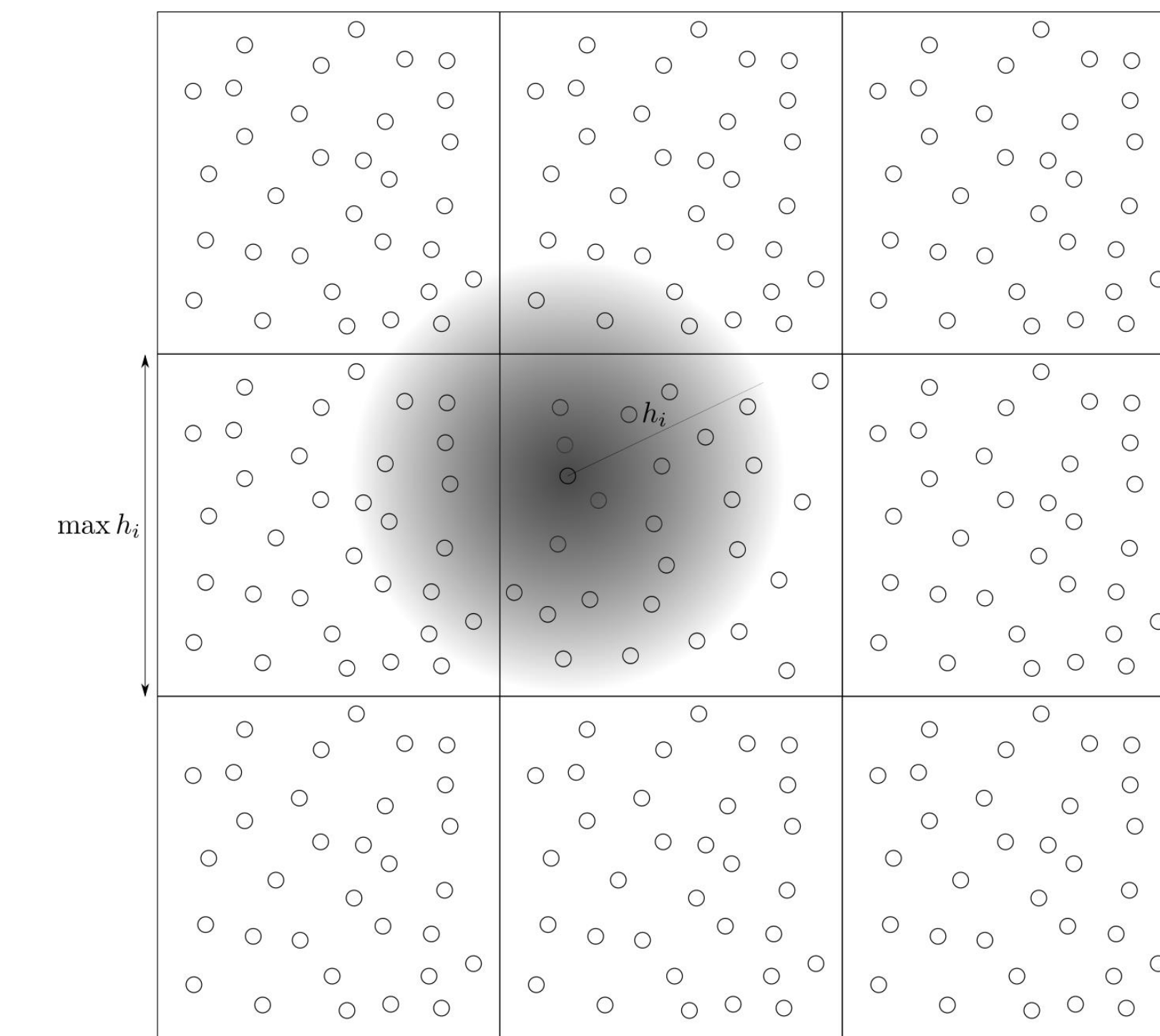


Figure 1: Computing the interactions of a set of N particles using smoothed particle hydrodynamics. The domain is broken up into a mesh of cells of edge length $\max h_i$. Particle neighbours can be found by searching neighbouring cells using h_i as the search radius. However, a large fraction of particles in neighbouring cells will be out of range. This fraction varies depending on the orientation of the two neighbouring cells.

Our pseudo-Verlet list method fixes this problem by only searching for neighbours that lie within a given cut-off radius of the cell edge.

Performance Results

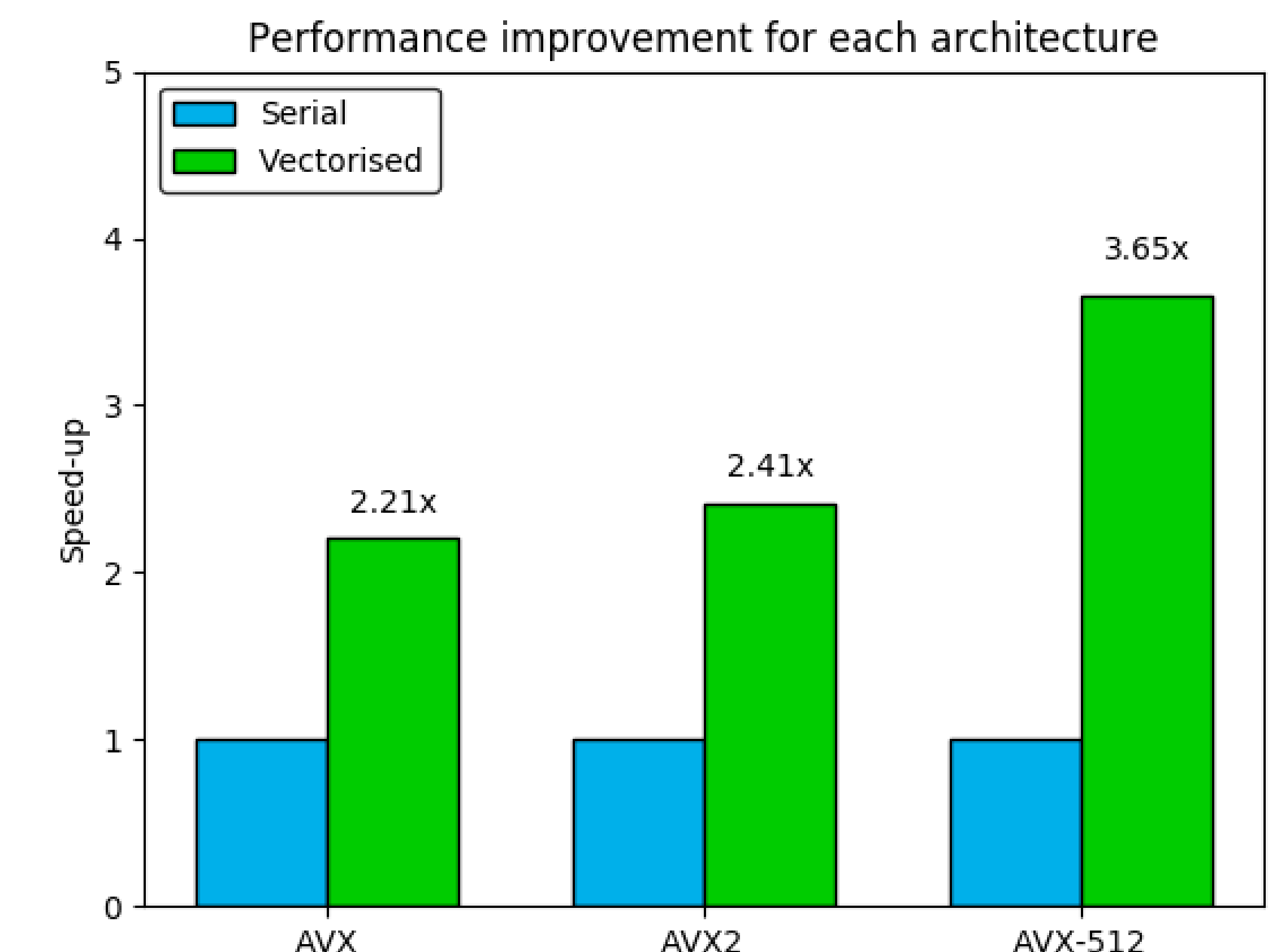


Figure 3: Speed-up performance of the SIMD pseudo-Verlet list algorithm when compared to the scalar version. Results for AVX, AVX2 and AVX-512 instruction sets are shown. The difference between the AVX and AVX2 results (both using 256-bit long vectors) is due to the use of Fused Multiply-Add instructions in the AVX2 set. Similarly, the speed-up for the AVX-512 instruction set is greater due to the increased vector length (512-bit) and advanced masking instructions.

Table 1: Median times and corresponding speed-ups of the SIMD pseudo-Verlet list algorithm implemented using the AVX, AVX2 and AVX-512 instruction sets.

Machine Name	CFLAGS	Scalar Time [ms] (-no-vec -no-simd)	Vectorised Time [ms]	Speed-up
COSMA-5 (Sandybridge)	-xAVX	1.213	0.548	2.21x
Hamilton (Broadwell)	-xCORE-AVX2	0.805	0.335	2.41x
COSMA-7 (Skylake)	-xCORE-AVX512	0.980	0.269	3.65x

[1] P. Gonnet, "Pseudo-verlet lists: a new, compact neighbour list representation," Molecular Simulation, vol. 39, no. 9, pp. 721–727, 2013.

[2] M. Schaller, P. Gonnet, A. B. G. Chalk, and P. W. Draper, "Swift: Using task-based parallelism, fully asynchronous communication, and graph partition-based domain decomposition for strong scaling on more than 100,000 cores, PASC '16, (New York, NY, USA), pp. 2:1–2:10, ACM, 2016.

[3] Steve Plimpton. 1995. Fast Parallel Algorithms for Short-Range Molecular Dynamics. J. Comput. Phys. 117, 1 (1995), 1 – 19. <https://doi.org/10.1006/jcph.1995.1039>

[4] Mark James Abraham, Teemu Murtola, Roland Schulz, Szilárd Páll, Jeremy C. Smith, Berk Hess, and Erik Lindahl. 2015. GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. SoftwareX 1-2 (2015), 19 – 25. <https://doi.org/10.1016/j.softx.2015.06.001>