



Phoenix

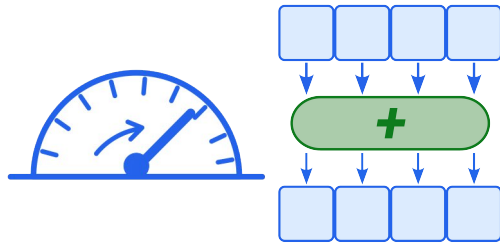
Auto-Vectorisation Reborn

Jonas Sys, Elisa Gonzalez Boix, Christophe Scholliers

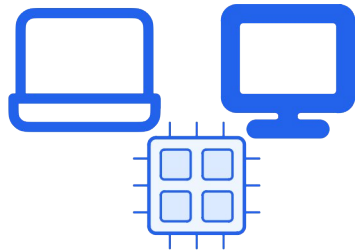


VRIJE
UNIVERSITEIT
BRUSSEL

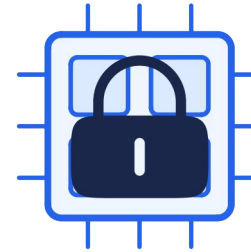
Vectorisation



Single Instruction, Multiple Data
Significant speedups



Available on commodity hardware



Hardware-dependent
Not portable



Complex & error-prone

Vectorisation Approaches

Manual Vectorisation



Guaranteed performance
Complex & error-prone

```
inline void gamma_srgb(std::vector<float> &xs) {
    const auto threshold = _mm512_set1_ps(0.04045f);
    // ... other constants

    for (size_t i = 0; i < xs.size(); i += 16) {
        const auto idx_mask = (1uL << std::min(xs.size() - i, 16)) - 1;
        const auto ld = _mm512_maskz_loadu_ps(idx_mask, xs.data() + i);
        const auto mask = _mm512_mask_cmp_ps_mask(idx_mask, ld, threshold, _CMP_LT_OQ);
        const auto mask_not = idx_mask & (~mask);

        const auto div = _mm512_maskz_mul_ps(mask, ld, gamma_div_rcp);

        auto t_24 = const_fac; // t^0 term
        const auto t = _mm512_maskz_mul_ps(mask_not,
            _mm512_maskz_add_ps(mask_not, ld, x2t_plus), x2t_div_rcp); // t
        const auto t2 = _mm512_maskz_mul_ps(mask_not, t, t); // t^2
        const auto t3 = _mm512_maskz_mul_ps(mask_not, t2, t); // t^3

        t_24 = _mm512_maskz_fmadd_ps(mask_not, linear_fac, t, t_24);
        t_24 = _mm512_maskz_fmadd_ps(mask_not, quadratic_fac, t2, t_24);
        t_24 = _mm512_maskz_fmadd_ps(mask_not, cubic_fac, t3, t_24);

        const auto res = _mm512_mask_blend_ps(mask, t_24, div);
        _mm512_mask_storeu_ps(xs.data() + i, idx_mask, res);
    }
}
```



[1] Free Software Foundation, "Using the GNU Compiler Collection," version 15.1.0, 2026. [Online]. Available: <https://gcc.gnu.org/onlinedocs/>.
[2] C. Lattner and V. Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In International Symposium on Code Generation and Optimization, 2004. CGO 2004., pages 75–86.
[3] OpenJDK Community, "OpenJDK 26: HotSpot Virtual Machine," 2026. [Online]. Available: <https://openjdk.org/>.
[4] Intel Corporation, oneAPI DPC++ Library Guide. Intel, 2022. URL: <https://www.intel.com/content/www/us/en/docs/onedpl/developer-guide/>.
[5] Roland Leißa, Immanuel Haffner, and Sebastian Hack. Sierra: A SIMD extension for C++. In Proceedings of the 2014 Workshop on Programming Models for SIMD/Vector Processing, pages 17–24. ACM.
[6] L. Dagum and R. Menon. Openmp: an industry standard api for shared-memory programming. IEEE Computational Science and Engineering, 5(1):46–55, 1998.
[7] Tao B. Schardl and I-Ting Angelina Lee. OpenCilk: A Modular and Extensible Software Infrastructure for Fast Task-Parallel Code. In Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming, pages 189–203. ACM.
[8] Intel Corporation, Intel Intrinsics Guide — Version 3.6.9. Intel Corporation, 12 2024. URL: <https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html>.
[9] Arm Limited, Neon - Arm®. Arm Corporation, 04 2026. URL: <https://www.arm.com/technologies/neon>.
[10] xtensor-stack and Open-Source Contributors. xsimd (Version 14.1.0) [Source code]. GitHub. URL: <https://github.com/xtensor-stack/xsimd>.
[11] OpenJDK. Project Panama: Interconnecting JVM and native code. URL: <https://openjdk.org/projects/panama/>.
[12] Microsoft Corporation. System.Numerics Namespace | Microsoft Learn. Microsoft Corporation. URL: <https://learn.microsoft.com/en-us/dotnet/api/system.numerics?view=net-10.0>.
[13] Angela Pohl, Biagio Cosenza, Mauricio Alvarez Mesa, Chi Ching Chi, and Ben Juurlink. An evaluation of current SIMD programming models for C++. In Proceedings of the 3rd Workshop on Programming Models for SIMD/Vector Processing, pages 1–8. ACM.

Vectorisation Approaches

Manual Vectorisation



Guaranteed performance
Complex & error-prone

Language Extensions

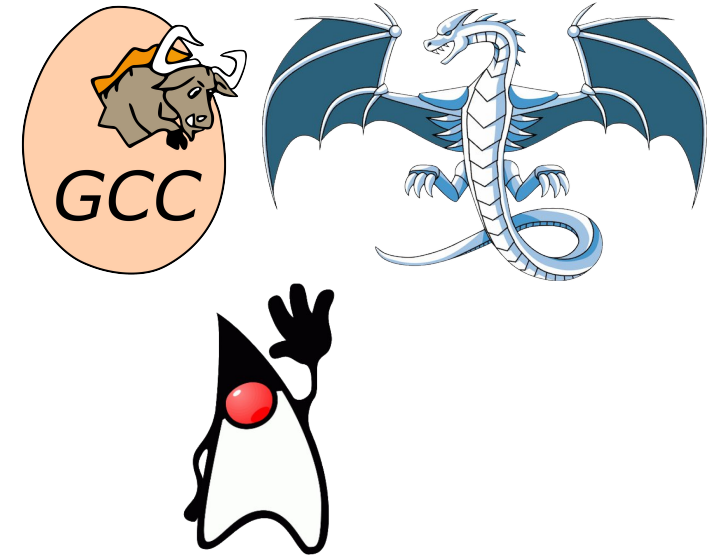


Possible race-conditions



Abstractions guarantee
safe vectorisation

Automatic Vectorisation



Scalar code
Not always possible

[1] Free Software Foundation, "Using the GNU Compiler Collection," version 15.1.0, 2026. [Online]. Available: <https://gcc.gnu.org/onlinedocs/>.
[2] C. Lattner and V. Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In International Symposium on Code Generation and Optimization, 2004. CGO 2004., pages 75–86.
[3] OpenJDK Community, "OpenJDK 26: HotSpot Virtual Machine," 2026. [Online]. Available: <https://openjdk.org/>.
[4] Intel Corporation, oneAPI DPC++ Library Guide. Intel, 2022. URL: <https://www.intel.com/content/www/us/en/docs/onedpl/developer-guide/>.
[5] Roland Leißa, Immanuel Haffner, and Sebastian Hack. Sierra: A SIMD extension for C++. In Proceedings of the 2014 Workshop on Programming Models for SIMD/Vector Processing, pages 17–24. ACM.
[6] L. Dagum and R. Menon. Openmp: an industry standard api for shared-memory programming. IEEE Computational Science and Engineering, 5(1):46–55, 1998.
[7] Tao B. Schardl and I-Ting Angelina Lee. OpenCilk: A Modular and Extensible Software Infrastructure for Fast Task-Parallel Code. In Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming, pages 189–203. ACM.
[8] Intel Corporation, Intel Intrinsics Guide — Version 3.6.9. Intel Corporation, 12 2024. URL: <https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html>.
[9] Arm Limited, Neon - Arm®. Arm Corporation, 04 2026. URL: <https://www.arm.com/technologies/neon>.
[10] xtensor-stack and Open-Source Contributors. xsimd (Version 14.1.0) [Source code]. GitHub. URL: <https://github.com/xtensor-stack/xsimd>.
[11] OpenJDK. Project Panama: Interconnecting JVM and native code. URL: <https://openjdk.org/projects/panama/>.
[12] Microsoft Corporation. System.Numerics Namespace | Microsoft Learn. Microsoft Corporation. URL: <https://learn.microsoft.com/en-us/dotnet/api/system.numerics?view=net-10.0>.
[13] Angela Pohl, Biagio Cosenza, Mauricio Alvarez Mesa, Chi Ching Chi, and Ben Juurlink. An evaluation of current SIMD programming models for C++. In Proceedings of the 3rd Workshop on Programming Models for SIMD/Vector Processing, pages 1–8. ACM.

Phoenix Abstractions

Restricted Vector Loops

#1: Iteration Independence

No non-local modifications

#2: Loop Completion

No break/return

Early-end single iteration: `continue/continue with`

#3: Index-Free Loop Operations

Due to masking

Phoenix Rising

The Phoenix logo, a stylized orange and yellow bird with flames for feathers, positioned over the code editor.

```
1 float32 x; float32 y; }
2
3 distance(const point &p1, const point &p2) {
4     float32 x1 = p1.x; float32 x2 = p2.x;
5     float32 y1 = p1.y; float32 y2 = p2.y;
6     float32 dx = (x2 - x1); float32 dy = (y2 - y1);
7     return sqrt(dx * dx + dy * dy);
8 }
9
10 point add(const point &p1, const point &p2) {
11     return new point(p1.x + p2.x, p1.y + p2.y);
12 }
13
14 point divide(const point &p, float32 div) {
15     if(div == 0.0f) return new point(0.0f, 0.0f); // edge case
16     return new point(p.x / div, p.y / div);
17 }
18
19 @Export
20 void kmeans(const vec<point> &points, const vec<point> &ctr, vec<point> &new_centers, float32 &new_centers_size) {
21     view<pair<point, float32>> new_centers(ctr.size());
22     project p in points into new_centers {
23         pair<float32, uint32> closest;
24         @Vectorize
25         accumulate (ctr, 1) in ctrs with new pair=> {
26             float32 dist = distance(p, ctr);
27             if(dist < closest.first) {
28                 continue with new pair=>float32, uint32>(dist, 1);
29             }
30             continue;
31         }
32         continue[closest.second] with {current} => new pair=>point, float32>(add(current.first, p), current.second + 1.0f);
33     }
34
35     project x in new_centers into ctrs_out {
36         continue with divide(x.first, c.second);
37     }
38 }
39 }
```

Views & Restricted Vector Loops

Restricted Vector Loops

#1: Iteration Independence

#2: Loop Completion

#3: Index-Free Loop Operations

```
foreach (x, idx) in view {}  
    (modify in-place)  
mask (x, idx) in view {}  
    (filter in-place)  
project (x, idx) in view into output {}  
    (modify out-of-place)  
merge (x1, ..., xn, idx) in (view1, ..., viewn) into output {}  
    (combine multiple views)  
accumulate (x, idx) in view with initial into var {}  
    (reduce to single value)
```

continue

mask out *current* output element

continue with e

write *current* output element

continue[idx] with $(x) \rightarrow e$

update *arbitrary* output element
executes scalar (correctness)

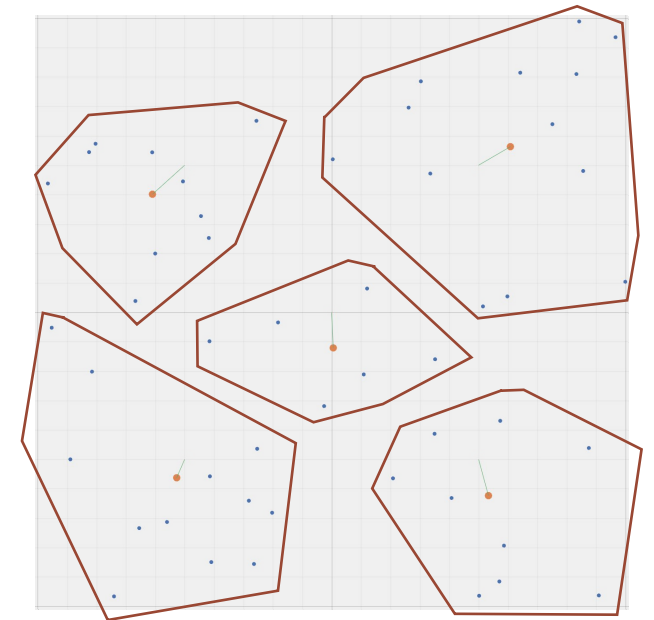
KMeans

```
function kmeans(points, centers) is
    origin = (0, 0)
    new_centers = [(origin, 0) for each center in centers]

    for each point in points:
        c_idx = argmin distance(point, center) in centers
        new_centers[c_idx].point += point
        new_centers[c_idx].count += 1

    centers_out = []
    for each (sum, count) in new_centers:
        centers_out += sum / count

    return centers_out
```



Phoenix in Flight

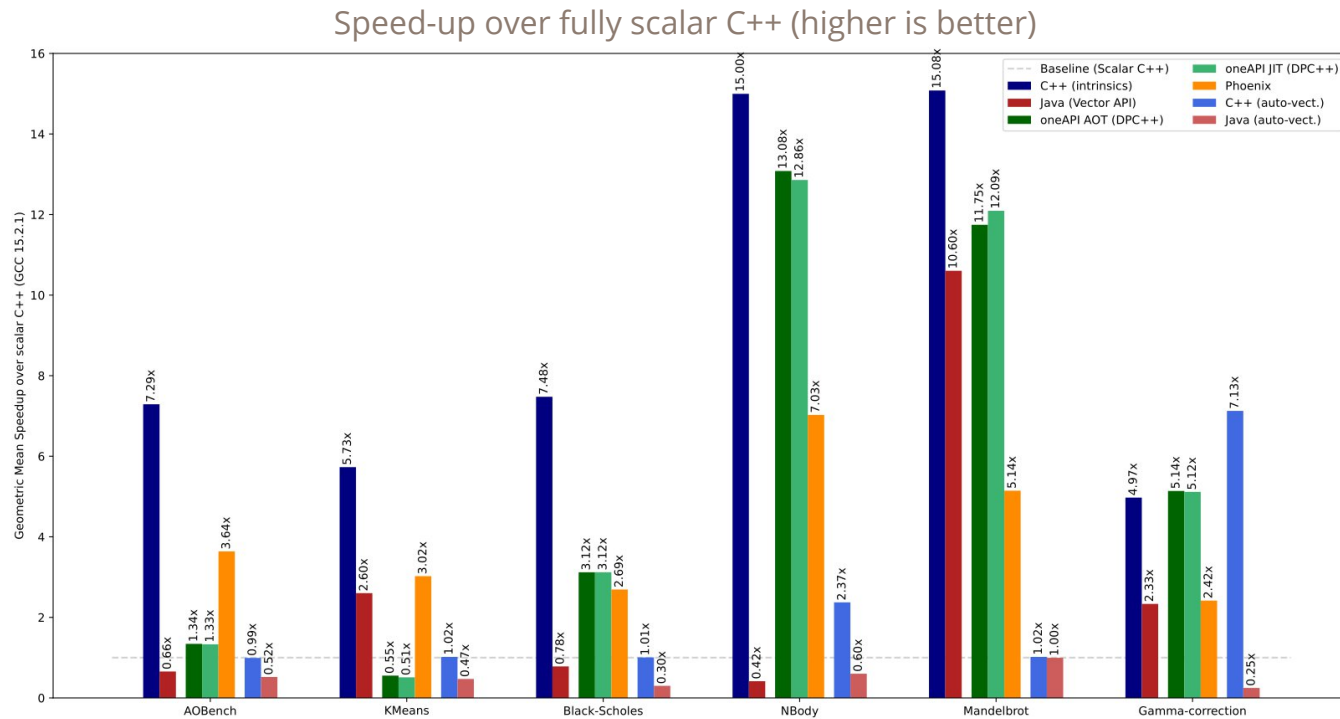


```
Float32 x; Float32 y; }  
point add(const point &p1, const point &p2) {  
    Float32 x1 = p1.x; Float32 x2 = p2.x;  
    Float32 y1 = p1.y; Float32 y2 = p2.y;  
    Float32 dx = (x2 - x1); Float32 dy = (y2 - y1);  
    return sqrt(dx * dx + dy * dy);  
}  
point add(const point &p1, const point &p2) {  
    return new point(p1.x + p2.x, p1.y + p2.y);  
}  
point divide(const point &p, Float32 div) {  
    if(div == 0.0f) return new point(0.0f, 0.0f); // edge case  
    return new point(p.x / div, p.y / div);  
}  
@Export  
void kmeans(const vec<point> &points, const vec<point> &ctr, vec<point> &new_centers, Float32 &new_centers_size) {  
    project p in points into new_centers {  
        pair<Float32, uint32> closest;  
        @Vectorize  
        accumulate (ctr, 1) in ctrs with new pair=<Float32, uint32> {  
            Float32 dist = distance(p, ctr);  
            if(dist < closest.first) {  
                continue with new pair=<Float32, uint32>(dist, 1);  
            }  
            continue;  
        }  
        continue[closest.second] with {current} -> new pair=<point, Float32>(add(current.first, p), current.second + 1.0f);  
    }  
    project x in new_centers into ctrs_out {  
        continue with divide(x.first, c.second);  
    }  
}
```

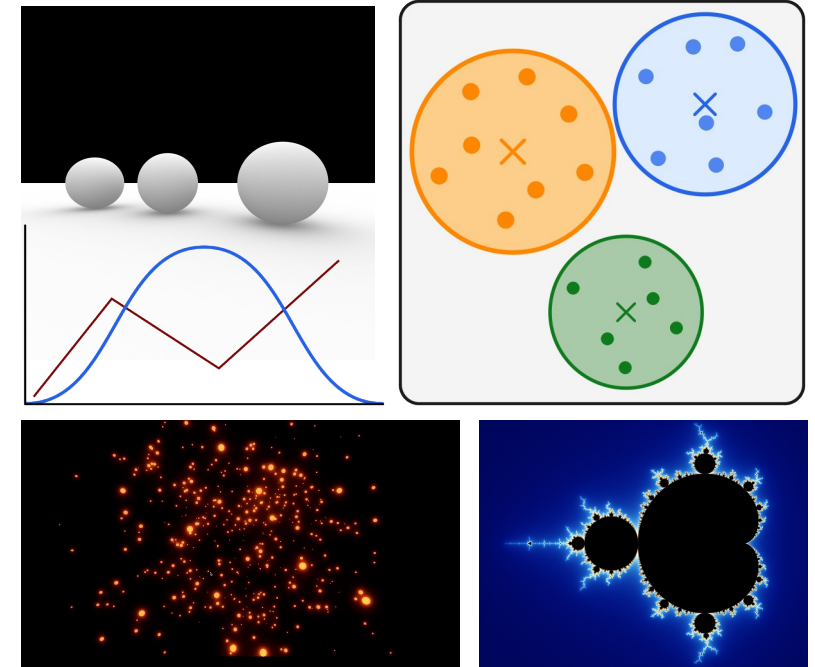


Evaluation

Execution Speed



Expressiveness



Conclusion



Restricted vector loops



Efficient vectorisation



Jonas Sys
jonas.sys@ugent.be