# MuCoSim Seminar WS 2024/25

### Student Info


- Name:Ashwin Varkey
- E-Mail: ashwin.varkey@fau.de
- Matriculation number:23115420


## Phase 1 : Analysis of MD-Bench Verletlist on AMD Milan

### Application Info

- Code: MD-Bench
- URL: https://github.com/RRZE-HPC/MD-Bench

MD-Bench is a performance-oriented prototyping harness for state-of-the-art Molecular Dynamics (MD) algorithms.It is a toolbox for the performance engineering of short-range force calculation kernels and is developed by the Erlangen National High Performance Computing Center (NHR@FAU)

### Project Description

To perform MD-Bench analysis for solid copper on AMD Milan using a specific compiler optimized for the Verlet list. Test various SIMD vector lengths, and compare the performance of full vs. half neighbor lists, as well as single vs. double precision.

### Testsystem / Software Environment

- Host/Clustername: Milan1
- CPU name: AMD EPYC 7543 32-Core Processor ( Zen3 architecture )
- CPU clock: 2.79 GHz (3.8 GHz Turbo)
- Memory capacity:251.56 GB
- Number of cores per node: 64
- Compiler: gcc/11.2.0 (gcc)
- Tools:likwid/5.3.0 (for performance profiling and analysis)

### How to build software

Begin by editing `config.mk`, which contains various configurations that we need:

- TOOLCHAIN: GCC
- ISA: x86
- SIMD: NONE, SSE, AVX2
- OPT_SCHEME: Verlet List (VL)
- ENABLE_LIKWID: True
- DATA_TYPE: SP and DP
- DATA_LAYOUT: AOS
- ENABLE_OMP_SIMD: True

Once you have configured `config.mk`, run `make` in the terminal to initiate the build process.

- **Allocating node on testcluster**:

```
salloc -t 03:00:00 --exclusive -w milan1 -c 64 -p work -C hwperf
```

### Results and Analysis

#### Runtime Profile

This analysis aims to identify bottlenecks in the program and highlight functions that might requiere a deeper analysis.

| Region | Time [s] (FN) | Time [s] [HN] | % Overall (FN) | SSE time [s] | AVX time [s] |
|---|---|---|---|---|---|
| Force (SP) | 14.28 | 8.76 | 83.31 | 5.90 | 4.18 |
| Reneighbour | 2.86 | 2.05 | 16.69 | 2.93 | 2.85 |
| Force (DP) | 14.28 | 8.91 | 81.07 | 5.89 | 4.18 |
| Reneighbour | 2.95 | 2.11 | 18.93 | 2.91 | 2.87 |

Table 1: Runtime profile for MDBench-VL-ICC-DP for different configurations

#### Key Observations:

- The main bottleneck in your program seems to be the Force (computeForceLJFullNeigh()) region is compute bounded
- Reneighbour region (buildNeighborCPU()) is memory bound
- The Half Neighbor (HN) strategy improves the scalar case

#### Instruction Count Analysis

This analysis quantifies an application by the number of executed instructions, aiding in evaluating SIMD vectorization efficiency and comparing implementation variants and algorithmic variations.

#### Approach 1

- Use **RETIRED_SSE_AVX_FLOPS_ALL** to calculate total arithmetic operations.
- Use **RETIRED_MMX_FP_INSTR_SSE** to measure FLOPs performed using SSE instructions (arthimetic SIMD).

```
likwid-perfctr -C S0:1 -g RETIRED_MMX_FP_INSTR_SSE:PMC0 -m ./MDBench-VL-GCC-X86-DP
```

**Issues**

- RETIRED_MMX_FP_INSTR_SSE includes all SSE instructions, not just arithmetic ones, which skews analysis by counting non-arithmetic operations like SHUFPS, BLENDPS, and MOVAPS, and it cannot differentiate between AVX and SSE FLOPs.

**Approach 2**

- Since Zen 3 lacks accurate events for counting arithmetic instructions by SIMD width and scalar, values were sourced from Zen 4 (AMD Genoa), which provides better events like RETIRED_PACKED_FP_OPS to track packed floating-point operations (ADD, SUB, MUL, DIV, FMA) by SIMD variant.

```
likwid-perfctr -C 0 -m -g RETIRED_PACKED_FP_OPS_FP256_ADD:PMC0
(RETIRED_FP_OPS_BY_TYPE_SCALAR_ADD:PMC0 for scalar)
RETIRED_PACKED_FP_OPS_FP256_SUB:PMC1,
RETIRED_PACKED_FP_OPS_FP256_MUL:PMC2,
RETIRED_PACKED_FP_OPS_FP256_DIV:PMC3  ./MDBench-VL-GCC-X86-AVX2-DP
```

The metrics with * in prefix are from AMD Genoa

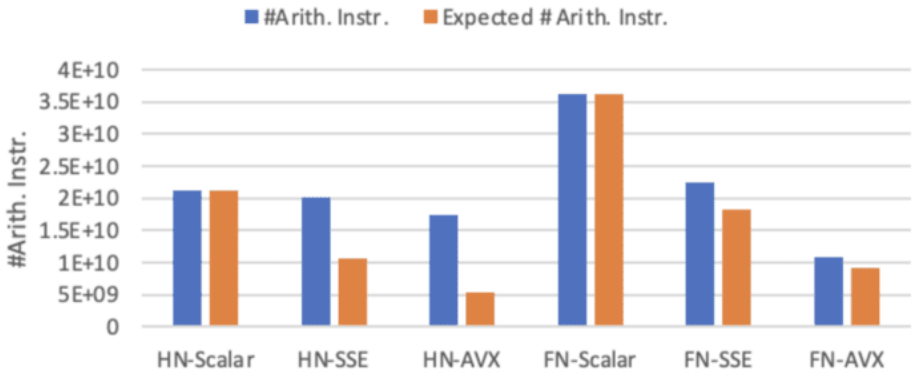
| SP | FN novec | FN SSE | FN AVX |
|---|---|---|---|
| Instructions total | 73458550000 | 41804010000 | 28181150000 |
| *Instructions arith | 36283151000 | 13152431974 | 6761919900 |
| *Vectorization ratio [%] | 0.00 | 98.34 | 93.19 |
| *Percentage Arithmetic [%] | 49.39 | 31.46 | 23.99 |
| Runtime | 13.72 | 4.33 | 3.37 |
| CPI | 0.69 | 0.38 | 0.44 |
| Perf [MFlops/s] | 2643.31 | 11988.41 | 15332.21 |

| DP | FN novec | FN SSE | FN AVX |
|---|---|---|---|
| Instructions total | 73432270000 | 55602820000 | 32516670000 |
| *Instructions arith | 36283320000 | 22360608150 | 10934725320 |
| *Vectorization ratio [%] | 0.00 | 97.35 | 95.79 |
| *Percentage Arithmetic [%] | 49.41 | 40.21 | 33.63 |
| Runtime | 13.86 | 5.90 | 4.18 |
| CPI | 0.70 | 0.39 | 0.48 |
| Perf [MFlops/s] | 2617.62 | 7481.53 | 10642.43 |

Table 2: Instruction Count Analysis for full neighbour verletlist comparing single precision vs double precision

**Inferences - SIMD mode, Neighbourlist and Precision**

- With SIMD,the CPI lowers and total instr./arithmetic counts come down.
- Increasing SIMD reduces runtime (3× lesser in FN-SSE, 4× in FN-AVX).
- AVX underutilizes vectorization as YMM registers handle data movement (vmovups), while arithmetic still relies on XMM (vmulss).
- Half neighbor shows no time reduction due to sparse datasets, while full neighbor optimizes SIMD by reducing arithmetic instructions.




Graph 1: Arithmetic Instructions count for verletlist SIMD vectorization

**Performance Analysis**

```
likwid-perfctr -C S0:1 -g L2/L3/MEM1/CACHE -m ./MDBench-VL-GCC-X86-DP
```

| | HN-SP | HN-DP | FN-SP | FN-DP |
|---|---|---|---|---|
| L2 [MBytes/s] | 2429.01 | 3058.21 | 1517.02 | 1826.01 |
| L2 [GB] | 20.02 | 26.61 | 20.42 | 25.31 |
| L3 [MBytes/s] | 1845.81 | 2015.72 | 1115.92 | 1265.52 |
| L3 [GB] | 16.12 | 17.62 | 15.42 | 17.62 |
| MEM [GB] | 8.61 | 10.61 | 8.91 | 10.91 |
| data cache misses (e4) | 29520 | 14508 | 17841 | 12671 |

Table 3: Performance Analysis for verletlist for different configurations

**Inferences**

- Data reuse in L2 (10 GB in memory, 20 GB in L2) occurs twice, with AMD Milan's victim cache writing back only the modified data.
- SP has more cache misses than DP, limiting performance, while the kernel is not memory-bound with bandwidths (0.7-2 GB/s) far below AMD Milan's single-core capacity.

## Phase 2 : Energy efficiency and performance analysis verletlist vs clusterpair on Intel Sapphire Rapids

### Project Description

Analyze energy and performance of MD-Bench on Intel Sapphire Rapids, comparing verletlist vs. clusterpair for solid copper with one compiler, varying SIMD lengths, precision, and neighbor list configurations.

### Testsystem / Software Environment

- Host/Clustername: saprap2
- CPU name: Intel® Xeon® Platinum 8470 processor (Sapphire Rapids)
- Base Frequency: 2.00 GHz (Turbo 3.80 GHz)
- Memory capacity: 503 GB
- Number of cores per node: 52 cores and 2 threads per core (SMT active)
- Interconnect: PCIe®
- Compiler: intel/2023.2.1 (ICC)
- LIKWID Tools: likwid/5.4.0
- Slurm: slurm/23.02.7

### How to build software

Edit config.mk to set compilation and optimization options and run "make" command on terminal. For this case:

- TOOLCHAIN: ICC
- ISA: x86
- SIMD: NONE, SSE, AVX, AVX512
- OPT_SCHEME: Verlet List (VL) and Cluster Pair (CP)
- ENABLE_LIKWID: True
- ENABLE_OPENMP: True
- ENABLE_MPI: True
- DATA_TYPE: SP and DP
- DATA_LAYOUT: AOS
- SORT_ATOMS: True

### Benchmarks

#### Study 1 : Frequency study with SIMD

- Analyze performance on SPR by varying frequencies with Slurm (srun) and likwid-setFrequencies. Reset with likwid-setFrequencies -reset before release.

```
srun --cpu-freq=<freq in KHz>-< freq in KHz >:performance
likwid-perfctr -C 0 -g FLOPS_DP -m ./MDBench-VL-ICC-X86-DP
```

Set vs. Actual Frequency (GHz): 1.6 → 1.596, 3.0 → 2.993, 3.8 → 3.788, 4.0 → 3.788, 4.2 → 3.789

Performance (MFlops/s) @ 1.6 GHz → 3.8 GHz:

FN: SSE 2090.1 → 4980.0, AVX 3755.1 → 8978.7, AVX512 7891.5 → 18993.4
HN: SSE 1227.8 → 2923.2, AVX 1344.4 → 3205.6, AVX512 5784.9 → 14230.4

- Turbo mode is capped at 3.8 GHz, even when higher frequencies (e.g., 4.0 GHz, 4.2 GHz) are set.
- Higher CPU frequency improves performance by executing more instructions per second.

### Study 2 : AOS vs SOA

- The decision between AOS and SOA depends on workload characteristics, memory access patterns, and the hardware's SIMD or cache efficiency. In SIMD scenarios, both AOS and SOA exhibit similar performance. Test results indicate that hybrid data layouts could be more effective.

### Study 3: Sorting

- Sorting operations have little effect on performance, while switching to single precision reduces the instruction count but provides only a slight boost in FLOPS.

### Study 4 : Clusterpair vs. Verletlist Across Varying Problem Sizes

| Metrics ( "Force" region ) | 16*16*16 VL | 24*24*24 VL | 32*32*32 VL | 48*48*48 VL | 64*64*64 VL |
|---|---|---|---|---|---|
| Total Instr. | 2,09,78,21,000 | 4,10,34,43,000 | 16,82,42,60,000 | 56,96,70,10,000 | 1,34,55,45,00,000 |
| Arith. Instr. | 1,03,82,06,000 | 2,03,15,77,000 | 8,33,73,49,000 | 28,22,11,80,000 | 66,66,64,70,000 |
| Performance [Mflops/s] | 19,598 | 19,407 | 19,207 | 18,717 | 18,478 |
| Time [s] | 0.46 | 0.87 | 3.52 | 12.12 | 28.94 |
| Useful data volume [GB] | 8.12 | 15.91 | 65.58 | 220.99 | 522.98 |
| % Arith. | 49 | 50 | 50 | 50 | 50 |
| **Metrics ( "Force" region )** | **16*16*16 CP** | **24*24*24 CP** | **32*32*32 CP** | **48*48*48 CP** | **64*64*64 CP** |
| Total Instr. | 3,35,15,61,000 | 11,42,68,40,000 | 26,70,39,10,000 | 90,03,27,30,000 | 2,12,64,74,00,000 |
| Arith. Instr. | 2,70,90,09,000 | 9,24,59,73,000 | 21,59,76,30,000 | 72,82,91,00,000 | 1,72,00,22,00,000 |
| Performance [Mflops/s] | 54,161 | 54,527 | 54,831 | 54,868 | 54,599 |
| Time [s] | 0.44 | 1.4 | 3.19 | 10.66 | 25.26 |
| Useful data volume [GB] | 0.96 | 3.26 | 7.63 | 25.73 | 60.8 |
| % Arith. | 81 | 81 | 81 | 81 | 81 |

Table 4: Instruction Metrics for MDBench-VL-ICC-X86-AVX512-DP at Frequency 3.8 GHz

- The total number of executed instructions increases significantly with problem size. Clusterpair demonstrates better computational efficiency, with 81% of its instructions being arithmetic, whereas VL is constrained by non-computational operations.

**Results and Analysis**

**Runtime Profile**

| Region | Time [s] (FN) | Time [s] [HN] | % Overall (FN) | SSE Time [s] | AVX-512 Time [s] |
|---|---|---|---|---|---|
| Force (SP) | 14.70 | 10.09 | 80.37 | 10.98 | 4.36 |
| Reneighbour | 3.59 | 2.62 | 20.61 | 3.48 | 3.71 |
| Force (DP) | 14.90 | 10.36 | 80.45 | 10.7 | 4.52 |
| Reneighbour | 3.62 | 2.67 | 20.49 | 3.57 | 3.69 |

Table 5: Runtime Profile for MDBench-VL-ICC-X86-DP (FN vs HN) at Frequency 3 GHz

Increasing SIMD width reduces runtime in the force region (81-83% of total runtime) but does not enhance performance in the reneighboring region.

**Instruction Count Analysis**

```
srun --cpu-freq=3000000-3000000:performance
likwid-perfctr -C 0 FLOPS_DP -m ./MDBench-CP-ICC-X86-AVX512-DP
```

**Verletlist**

| SP | FN novec | FN SSE | FN AVX | FN AVX512 |
|---|---|---|---|---|
| Instructions total [10e3] | 86,037,460 | 55,933,420 | 47,291,520 | 9,881,790 |
| Instructions arith. [10e3] | 37,738,310 | 13,881,946 | 7,969,306 | 4,148,089 |
| Vectorization ratio [%] | 0.0 | 94.71 | 72.71 | 98.11 |
| Percentage Arithmetic [%] | 43.92 | 24.82 | 16.93 | 42.01 |
| runtime [s] | 14.73 | 6.23 | 5.53 | 3.73 |
| cpi [cyc/instr] | 0.51 | 0.43 | 0.32 | 1.11 |

| DP | FN novec | FN SSE | FN AVX | FN AVX512 |
|---|---|---|---|---|
| Instructions total [10e3] | 86,037,500 | 51,775,360 | 49,708,750 | 16,824,260 |
| Instructions arith. [10e3] | 37,738,310 | 21,724,264 | 13,953,280 | 8,416,387 |
| Vectorization ratio [%] | 0.0 | 98.51 | 87.6 | 99.11 |
| Percentage Arithmetic [%] | 43.91 | 41.92 | 28.13 | 50.13 |
| runtime [s] | 14.91 | 10.92 | 6.37 | 4.30 |
| cpi [cyc/instr.] | 0.51 | 0.35 | 0.38 | 0.76 |

Table 6: Instruction Count Analysis for Full Neighbour verletlist at Frequency 3.0 GHz

**Inferences**

- Instruction counts (both arithmetic and total) and runtime decrease with longer SIMD vectorization lengths in both single-precision (SP) and double-precision (DP) configurations.
- The higher CPI for AVX-512 indicates that, while it delivers better performance, gather instructions introduce additional latency and overhead.
- Switching from DP to SP slightly improves runtime, with a 1.5-2x reduction in instruction/arithmetic counts, as expected.
- Arithmetic instructions decrease with SIMD, but total instructions remain unchanged in AVX due to overhead, causing a fallback to SSE.

**Performance Analysis - Clusterpair vs Verletlist**

```
srun --cpu-freq=3000000-3000000:performance
likwid-perfctr -C 0 -g MEM/L2/L3 -m ./MDBench-VL-ICC-X86-AVX512-DP
```

| Force Region | HN-DP-VL | FN-DP-VL | HN-DP-CP | FN-DP-CP |
|---|---|---|---|---|
| L2 [GB] | 19.48 | 23.10 | 16.60 | 14.93 |
| L3 [GB] | 28.68 | 30.12 | 11.13 | 9.48 |
| MEM [GB] | 0.69 | 0.48 | 0.27 | 0.33 |
| PERF [MFLOP/S] | 11228.84 | 15485.67 | 40616.95 | 48226.10 |
| Data Volume [GB] | 35.34 | 65.58 | 4.85 | 7.63 |

Table 7: Performance Analysis comparing verletlist and clusterpair MDBench-ICC-AVX512-DP at Frequency 3.0 GHz

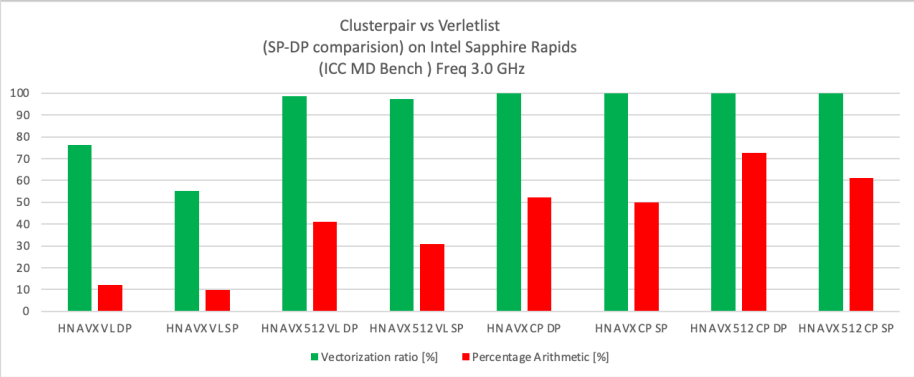
- Both optimizations reduce memory access through L2/L3 cache reuse, with Clusterpair outperforming Verletlist in performance and efficiency due to its Array-of-Struct-of-Arrays layout.
- Clusterpair uses lesser data volume compared to verletlist

**Cluster Pair vs. Verlet List: Neighbourlist and Precision**

| DP ("Force" region) | HN AVX | HN AVX512 | FN AVX | FN AVX512 |
|---|---|---|---|---|
| Instructions total [10e3] | 38,896,330 | 18,736,860 | 62,482,960 | 26,703,910 |
| Instructions arith [10e3] | 20,334,402 | 13,598,404 | 32,593,012 | 21,617,719 |
| runtime [s] | 4.71 | 2.72 | 7.41 | 3.63 |
| cpi [cy/inst] | 0.35 | 0.42 | 0.35 | 0.39 |

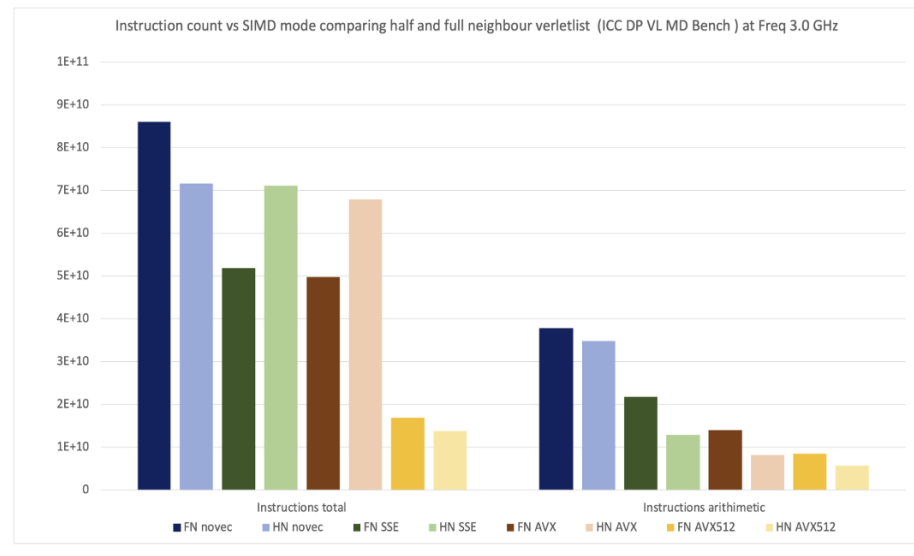
Table 8: Instruction Count Analysis by SIMD Mode for clusterpair at Frequency 3.0 GHz

- The Clusterpair approach maximizes SIMD utilization, achieving 100% vectorization, a higher arithmetic percentage, and lower runtime than Verletlist in both single and double precision.




Graph 2: Comparison of vectorization and arithmetic percentage between Clusterpair and Verletlist

- In SIMD, the half-neighbor list doesn't reduce instructions or runtime for Verletlist, whereas in Clusterpair, it significantly lowers these metrics.




Graph 3: Comparison of instruction counts between half and full-neighbor lists for Clusterpair vs. Verletlist

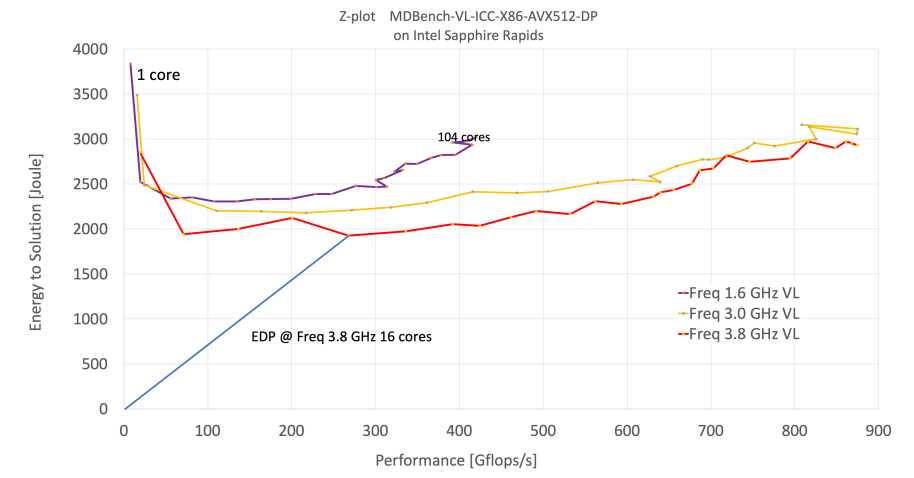
### Energy analysis -Verletlist

The Z-plot shows the energy-to-solution versus performance (inversely related to runtime), with a third variable (e.g., processor frequency or core count) represented along the line, highlighting efficiency across different configurations.

Energy-to-Solution is the total energy consumed during a run, measured by the LIKWID-perfctr ENERGY group, which sums PKG and DRAM Energy/Power from socket-specific counters, read using a single hardware thread per socket.

The Energy-Delay Product (EDP) is used to find a balance between low energy consumption and good performance.

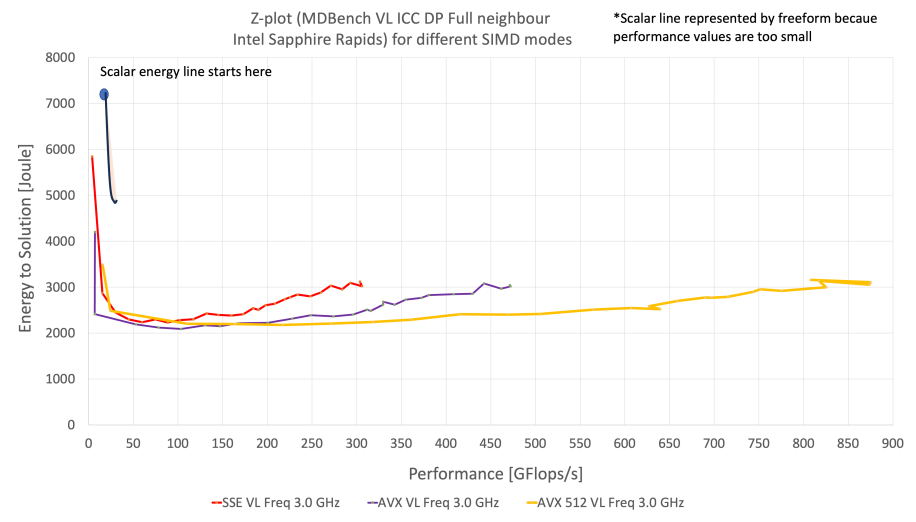
```
srun -n 1 -c 208 --cpu-freq=3800000-3800000:performance
likwid-perfctr -c S0:0@S1:0 -g ENERGY
likwid-pin -c N:0-103  ./MDBench-VL-ICC-X86-AVX512-DP

srun -n 1 -c 208 --cpu-freq=2400000-2400000:performance
likwid-perfctr -m -C N:1-103 -g FLOPS_DP ./MDBench-VL-ICC-X86-AVX512-DP
```




Graph 4: Z-plot for varying frequencies in verletlist with AVX 512 SIMD vectorization.

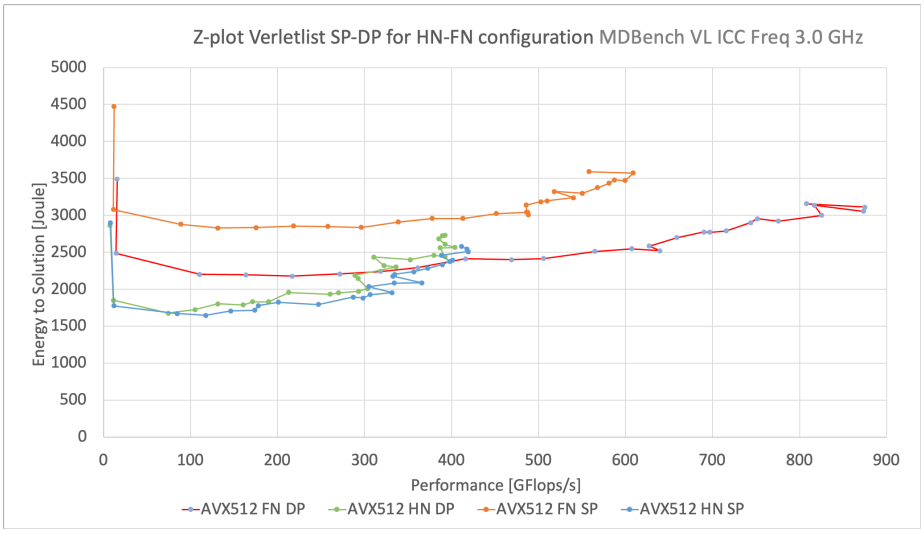
### Key Observations

- The energy-delay product (EDP) decreases as frequency increases at a fixed core count, reaching its lowest value at 16 cores and 3.8 GHz (blue line).
- The power cap is set to 508.45 W at 3.0 GHz, with the PKG domain power cap configured to 700 W.
- SIMD modes excel in energy efficiency, with AVX-512 providing the best performance (~850 GFlops/s) and the lowest energy-to-solution.



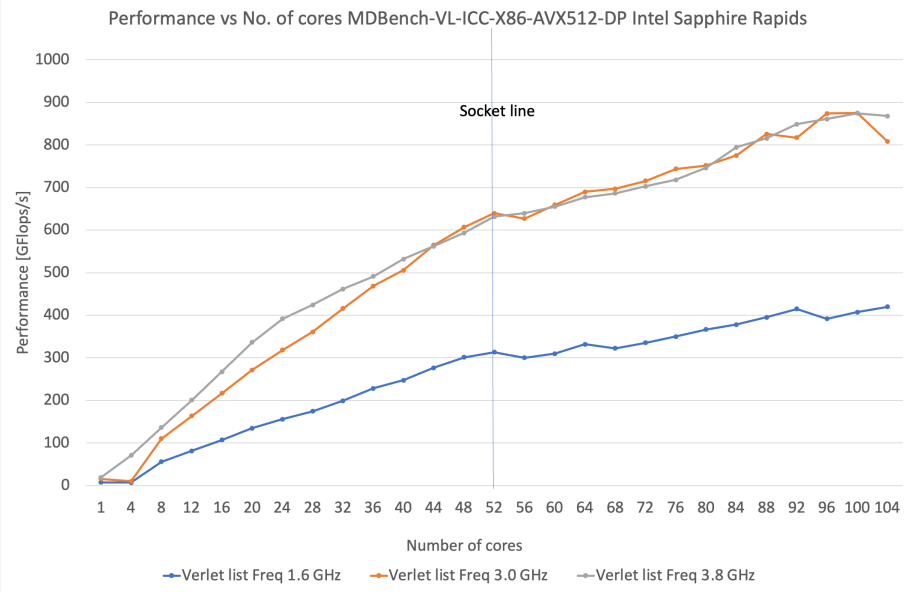

Graph 5: Z-plot for Verletlist by SIMD modes

- The half-neighbor list reduces energy-to-solution but does not match the performance of the full-neighbor configuration, leading to an unbalanced solution.




Graph 6: Z-plot comparing verletlist for Half Neighbor vs Full Neighborlist

### Scalability Studies - Verletlist vs Clusterpair

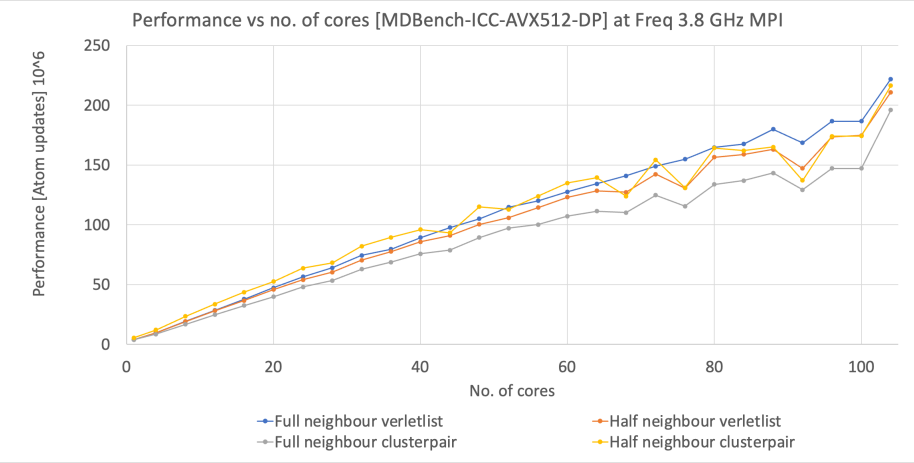



Graph 7: Performance scaling with core count for verletlist

Clusterpair struggles with OpenMP scaling due to uneven workload distribution, prompting the use of MPI for studying performance scaling.

### Performance Scalability using MPI

- Verletlist scales linearly with core count due to balanced workload distribution, while clusterpair maintains linear scaling upto 40 cores.

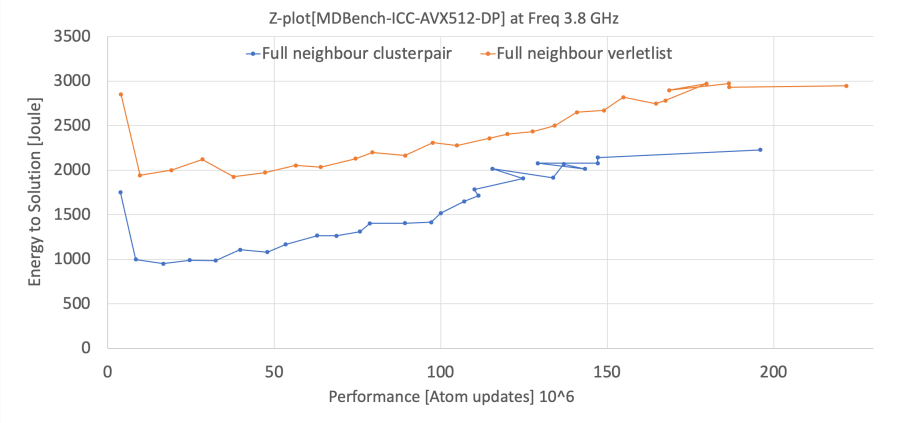



Graph 8: Performance scaling with core count for verletlist vs. clusterpair

### Energy Analysis using MPI

```
srun --cpu-freq=3800000-3800000:performance  -n 104
--mpi=pmi2 ./MDBench-CP-ICC-X86-AVX512-DP -half 1 -method 0 -bal 0
```

- The Z-plot shows that clusterpair offers a lower energy-to-solution ratio with high performance, making it preferable, even though verletlist achieves better core scaling and higher performance gains at the expense of higher energy-to-solution values.




Graph 9: Z-plot comparing clusterpair and verletlist for AVX 512 SIMD vectorization

**Workload distribution:** The arithmetic instruction and data volume highlight the trade-off: in Clusterpair, memory access is more efficient, requiring fewer instructions due to its SIMD-optimized format.

Clusterpair shows workload imbalance as core scaling increases, with a wider gap in arithmetic instructions, while Verletlist maintains a <1% difference up to 104 cores, explaining its better scalability.

| | CP DP FN AVX 512 Freq 3.8 GHz | | | | | | | VL DP FN AVX 512 Freq 3.8 GHz | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Cores | Min Arith. Instr. | Max Arith. Instr. | Useful data volume (GB) | Arith % | Diff | Diff % = (Diff*100)/ Min | | Min Arith. Instr. | Max Arith. Instr. | Useful data volume (GB) | Arith % | Diff | Diff % = (Diff*100)/ Min |
| 1 | 21597630000 | - | 7.63 | 80.88 | | | | 8337349000 | - | 65.58 | 49.56 | | |
| 8 | 2680383000 | 2707205000 | 7.52 | 80.36 | 2,68,22,000 | 1 | | 1041284000 | 1043019000 | 9.49 | 48.81 | 17,35,000 | 0 |
| 16 | 1330044000 | 1360373000 | 7.28 | 79.51 | 3,03,29,000 | 2 | | 520410700 | 521914800 | 5.48 | 48.40 | 15,04,100 | 0 |
| 32 | 661258900 | 693454000 | 6.51 | 77.47 | 3,21,95,100 | 5 | | 259951800 | 261001400 | 3.48 | 47.05 | 10,49,600 | 0 |
| 80 | 261616700 | 278122600 | 6.07 | 47.14 | 1,65,05,900 | 6 | | 105142000 | 105980600 | 2.70 | 35.85 | 8,38,600 | 1 |
| 100 | 211360000 | 224906600 | 5.60 | 37.92 | 1,35,46,600 | 6 | | 83832830 | 84687930 | 2.50 | 33.27 | 8,55,100 | 1 |
| 104 | 202985500 | 216267000 | 5.50 | 49.17 | 1,32,81,500 | 7 | | 80601190 | 81346010 | 2.62 | 31.82 | 7,44,820 | 1 |

Table 9: Workload analysis comparing verletlist and clusterpair at Frequency 3.8 GHz

**Summary**

- Clusterpair's efficiency in memory and SIMD storage reduces instructions, but its 3x higher computations compared to verletlist limits time savings.
- Verletlist is constrained by data gathering and SIMD setup, while clusterpair boosts efficiency by eliminating gathers, though this results in a higher arithmetic instruction count.
- The critical difference lies in instruction throughput. The verletlist algorithm is bound by instruction throughput, with performance constrained by data gathering and SIMD register setup.
- Clusterpair outperforms verletlist in performance across all configurations while using less memory, thanks to improved locality with its Array-of-Struct-of-Arrays layout.
- Clusterpair fully utilizes SIMD, achieving 100% vectorization, a higher arithmetic percentage, and lower runtime than verletlist across all configurations.
- Performance improves with core count and frequency, with verletlist scaling linearly, while clusterpair faces OpenMP scaling issues due to uneven workload distribution, necessitating an MPI study where it scales linearly up to 40 cores.
- The Z-plot shows clusterpair's lower energy-to-solution, making it preferable despite verletlist's better core scaling, while AVX-512 excels in efficiency, providing the highest performance with the lowest energy-to-solution.