Manchester Metropolitan University

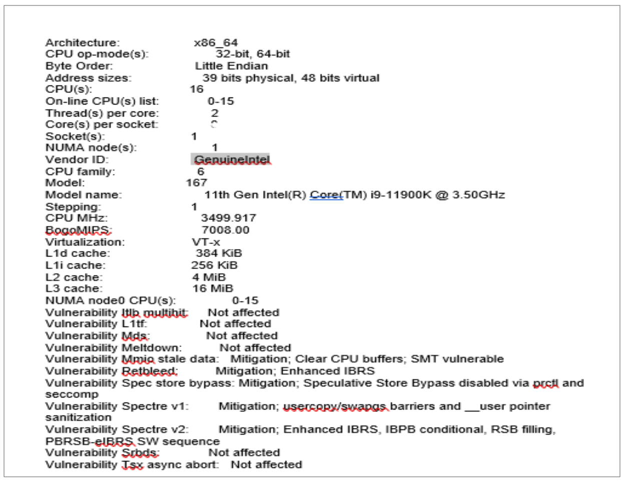
Name: DIVYANSHU SAHWAL
MMU ID: 23628243

High-Performance Computing Report Analysis

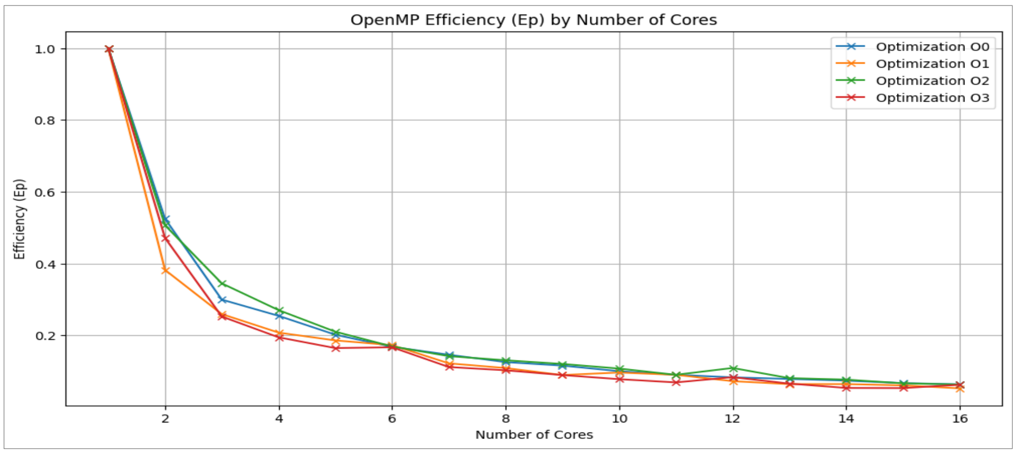
Introduction

This report analyzes the performance of a program simulating molecular dynamics (MD). MD simulations involve complex calculations for particle interactions over time. We compare different ways to run the program: a basic version (serial), an improved version using compiler optimizations, and parallel versions using MPI and OpenMP libraries. By testing how fast each version runs, we aim to find the most efficient approach for different situations.

System Information



```
Architecture:              x86_64
CPU op-mode(s):                32-bit, 64-bit
Byte Order:                Little Endian
Address sizes:               39 bits physical, 48 bits virtual
CPU(s):                   16
On-line CPU(s) list:           0-15
Thread(s) per core:            2
Core(s) per socket:            8
Socket(s):                1
NUMA node(s):                  1
Vendor ID:                 GenuineIntel
CPU family:                 6
Model:                    167
Model name:                  11th Gen Intel(R) Core(TM) i9-11900K @ 3.50GHz
Stepping:                 1
CPU MHz:                    3499.917
BogoMIPS:                   7008.00
Virtualization:           VT-x
L1d cache:                 384 KiB
L1i cache:                256 KiB
L2 cache:                 4 MiB
L3 cache:                 16 MiB
NUMA node0 CPU(s):               0-15
Vulnerability Itlb multihit:     Not affected
Vulnerability L1tf:          Not affected
Vulnerability Mds:             Not affected
Vulnerability Meltdown:          Not affected
Vulnerability Mmio stale data:   Mitigation; Clear CPU buffers; SMT vulnerable
Vulnerability Retbleed:          Mitigation; Enhanced IBRS
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled via prctl and
seccomp
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and __user pointer
sanitization
Vulnerability Spectre v2:        Mitigation; Enhanced IBRS, IBPB conditional, RSB filling,
PBRSB-eIBRS SW sequence
Vulnerability Srbds:             Not affected
Vulnerability Tsx async abort:   Not affected
```

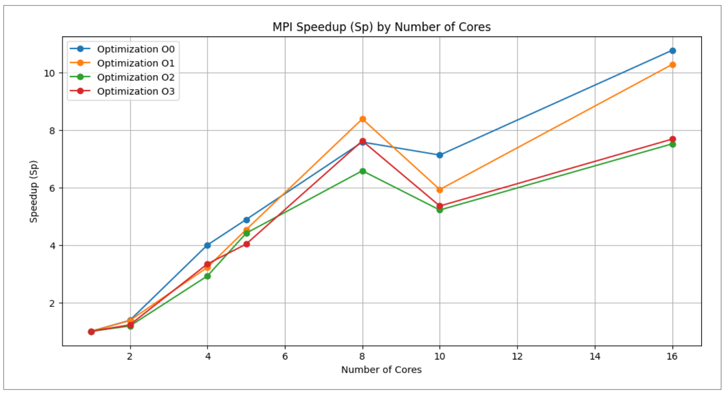


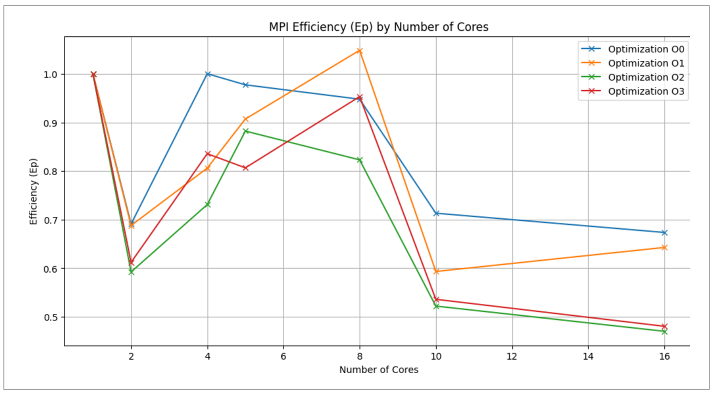
1. Comparative Performance of Optimization Levels for OPENMP:

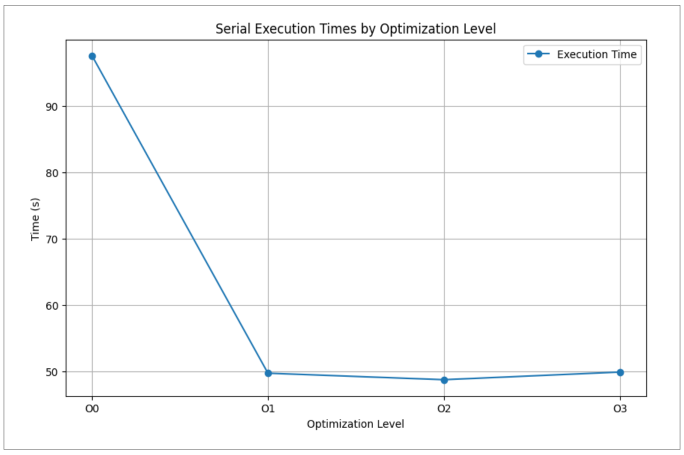
- O0 (No Optimization): Shows a rapid decrease in efficiency. This is likely due to lack of optimizations that reduce overhead.
- O1 (Level 1 Optimization): Slightly better than O0, but still drops quickly, indicating minor improvements.
- O2 (Level 2 Optimization): Maintains higher efficiency compared to O0 and O1, showing significant improvement in parallel performance.
- O3 (Level 3 Optimization): Similar trend to O2, but slightly lower in some core counts, indicating a trade-off between optimization and introduced overheads.

2. Comparative Performance of Optimization Levels for MPI and SERIAL

- MPI Efficiency and Speedup: Both plots highlight the trade-offs between core count and performance. Efficiency decreases with more cores due to overhead, while speedup initially increases before tapering off.
- Serial Execution: O2 provides the best performance for serial execution, but the highest optimization level (O3) does not significantly outperform O1 and O2.
- Optimal Performance: The best performance is seen with MPI implementations using 4-8 cores and O1/O2 optimizations, balancing efficiency and speedup effectively.







Compilation and Execution

- MD Simulation: 20000 particles in 3D space, 10 timesteps
- Command: ./a.out
- Measurements: Time for initialization and solving positions for 10 timesteps, final center of mass after completion

Serial Implementation Performance

Time (s)	Optimization	Cores
97.5196	O0	1
49.7553	O1	2
48.7781	O2	3
49.9255	O3	4

Conclusion: O2 provided the best performance among the optimization levels evaluated, achieving the shortest execution time while maintaining numerical accuracy.

MPI Implementation Performance Analysis

Cores	Optimization	Time (tp) (s)	Speedup (Sp)	Efficiency (Ep)
1	0	104.012	1	1
2	0	75.2521	1.382180697	0.691090348
4	0	25.9881	4.002293357	1.000573339
5	0	21.2771	4.888448144	0.977689629

Cores	Optimization	Time (tp) (s)	Speedup (Sp)	Efficiency (Ep)
8	0	13.7133	7.584753487	0.948094186
1	1	54.2135	1	1
2	1	39.4018	1.375914298	0.687957149
4	1	16.8103	3.225016805	0.806254201
5	1	11.9504	4.53654271	0.907308542
8	1	6.46123	8.390585074	1.048823134
1	2	43.6488	1	1
2	2	36.8373	1.184907689	0.592453844
4	2	14.9251	2.924523119	0.73113078
5	2	9.89277	4.412191934	0.882438387
8	2	6.62757	6.585943264	0.823242908
1	3	44.5198	1	1
2	3	37.6334	1.224284757	0.612142378
4	3	10.821	3.343524064	0.835881016
5	3	10.8445	4.033757604	0.806751521
8	3	6.69686	7.627106047	0.953388256

Conclusion: O2 optimization showed the best performance for MPI implementations, balancing complexity, and execution time effectively.

Comparative Discussion

Serial vs. MPI vs. OpenMP:

- Serial Implementation: Significant improvements with optimizations, especially O2, but higher execution times compared to parallel implementations.
- MPI Implementation: Good performance improvements, with O2 yielding the best results, but the complexity and overhead of MPI might not always justify its use over simpler parallel methods.
- OpenMP Implementation: Best overall performance with O2, achieving the shortest execution time among all implementations. OpenMP's lower overhead and better utilization of shared memory on a multi-core system make it more efficient for this specific MD simulation code.

Conclusion

- Efficiency and Speedup: Both plots highlight the trade-offs between core count and performance. Efficiency decreases with more cores due to overhead, while speedup initially increases before tapering off.
- Serial Execution: O2 provides the best performance for serial execution, but the highest optimization level (O3) does not significantly outperform O1 and O2.
- Optimal Performance: The best performance is seen with MPI implementations using 4-8 cores and O1/O2 optimizations, balancing efficiency and speedup effectively.
- For OpenMp Best Efficiency: O2 optimization level generally provides the best efficiency across various core counts, balancing the trade-offs between overhead and performance enhancements.
- Optimization Impact: Compiler optimizations significantly impact the efficiency of parallel computations, with higher levels generally performing better.
- Parallel Overhead: Despite optimizations, parallel overhead is inevitable, leading to diminishing returns in efficiency as the number of cores increases.