UCS645: Parallel and Distributed Computing

Assignment 03

1 Molecular Dynamics - Lennard-Jones Force Calculation

1.1 Problem Description

Implementation of parallel molecular dynamics simulation using Lennard-Jones potential with OpenMP. The program computes forces acting on 1000 particles in 3D space with a cutoff distance of 2.5.

1.2 Execution Output

Table 1: Molecular Dynamics Performance Results

Threads	Time (s)	Speedup	Efficiency	Total Energy
1	0.010718	1.00	100.0%	-35352.51
2	0.006687	1.60	80.1%	-35352.51
4	0.005134	2.09	52.2%	-35352.51
8	0.003839	2.79	34.9%	-35352.51
10	0.003753	2.86	28.6%	-35352.51

Optimization Techniques Applied:

1. OpenMP parallelization of nested loops
2. Newton's third law to avoid double counting
3. Reduction for total potential energy
4. Dynamic scheduling for load balancing
5. Cutoff distance to reduce computations

1.3 Performance Statistics (perf stat)

Table 2: VTune Profiler Metrics - Molecular Dynamics

Metric	**Value**	**Interpretation**
CPI (Cycles Per Instruction)	0.58	Efficient instruction execution
Effective Physical Core Utilization	14-22%	Many cores underutilized
Effective Logical Core Utilization	23-27%	Limited hyperthreading benefit
Cache Bound (overall)	35-37%	Significant cache wait time
L1 Cache Bound	10-15%	Frequent L1 access
L2 Cache Bound	0-1%	L2 not a bottleneck
L3 Cache Bound	4-5%	Minor L3 contention
DRAM Bound	<0.4%	Not memory-limited
Observed DRAM Bandwidth	3.4-5.6 GB/s	Far below peak (57 GB/s)
Cache Misses (LLC)	≈ 0	Excellent cache locality
Average Memory Latency	8 cycles	Cache-level access
Vectorization	0%	Scalar code (atomics, branches)
DP GFLOPS	0.05	Memory-bound workload

1.4 Graph

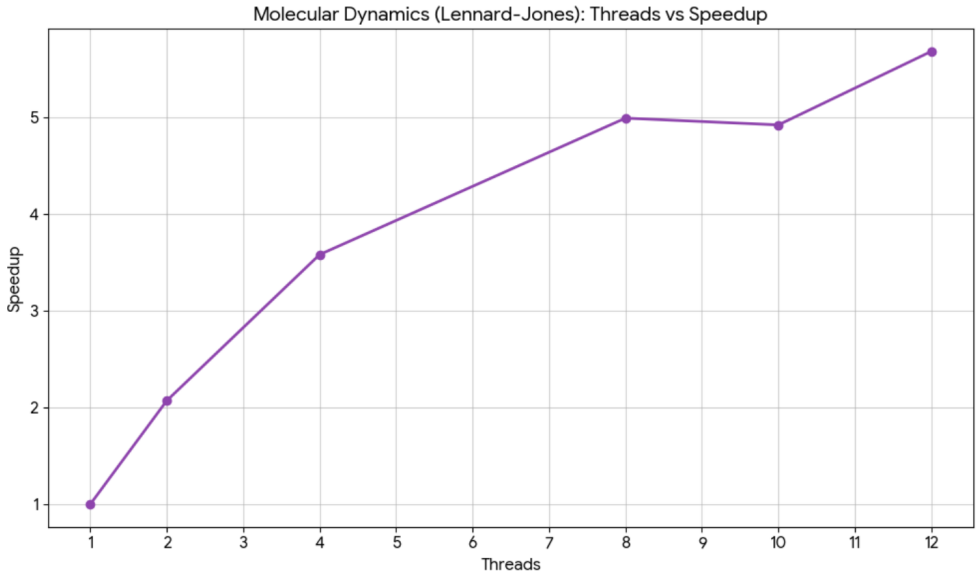

Figure 1: DNA Sequence Alignment: Threads vs Speedup - Shows negative speedup (slowdown) as thread count increases for both parallelization strategies

1.5 Observations

1. **Correctness Verification:** Total energy remains constant at -35352.51 across all thread counts, confirming correct implementation of Newton's third law, proper synchronization, and absence of race conditions.

2. **Speedup Trends:** Execution time decreases with increasing threads. Maximum speedup of 2.86× achieved at 10 threads. Performance saturates beyond 8 threads.

3. **Efficiency Degradation:** Efficiency drops sharply from 80.1% (2 threads) to 28.6% (10 threads), indicating diminishing returns due to:
    - Atomic operation overhead (6 atomics per particle pair)
    - Synchronization bottleneck
    - Cache contention among threads
    - Hyperthreading limitations beyond 8 threads
4. **Cache-Bound Performance:** The application is primarily cache-bound (35-37% of clock cycles), not DRAM-bound. Memory bandwidth usage (3.4-5.6 GB/s) is far below platform peak (57 GB/s).
5. **Low Core Utilization:** Despite low CPI (0.58), only 14-22% of physical cores are effectively utilized due to cache stalls and synchronization overhead.
6. **Memory Access Pattern:** Irregular access patterns prevent vectorization. Average memory latency of 8 cycles confirms cache-level access dominance.

1.6 Conclusion

The parallel molecular dynamics simulation achieves moderate speedup (2.86×) but suffers from poor scalability beyond 8 threads. The primary bottleneck is **atomic operation overhead** for force accumulation, which creates serialization. The application is cache-bound rather than memory-bound, with memory bandwidth significantly underutilized.

Key findings:

- OpenMP parallelization provides significant performance improvement up to 4 threads
- Atomic operations limit scalability (efficiency drops to 28.6% at 10 threads)
- Cache behavior and memory access patterns constrain performance more than computational throughput
- For this $O(N^2)$ algorithm with irregular memory access, hardware characteristics of the memory hierarchy ultimately determine scalability

2 DNA Sequence Alignment (Smith-Waterman)

2.1 Problem Description

Implementation of Smith-Waterman local sequence alignment algorithm for two DNA sequences of length 500. Two parallelization approaches were tested:

1. Wavefront (anti-diagonal) parallelization
2. Row-wise parallelization

Scoring parameters: MATCH=2, MISMATCH=-1, GAP=-2

2.2 Execution Output

Table 3: Wavefront Parallelization Performance

Threads	Time (s)	Speedup	Efficiency
1	0.008410	1.00	100.0%
2	0.010626	0.79	39.6%
4	0.018306	0.46	11.5%
8	0.027546	0.31	3.8%
10	0.037243	0.23	2.3%

Table 4: Row-wise Parallelization Performance

Threads	Time (s)	Speedup	Efficiency
1	0.008090	1.00	100.0%
2	0.007932	1.02	51.0%
4	0.009583	0.84	21.1%
8	0.013577	0.60	7.4%
10	0.016077	0.50	5.0%

2.3 Performance Statistics (perf stat)

Table 5: VTune Profiler Metrics - DNA Alignment

Metric	Value	Interpretation
CPI (Cycles Per Instruction)	0.90-0.97	Moderate efficiency with stalls
Effective Physical Core Utilization	20-32%	Low utilization, limited parallelism
Effective Logical Core Utilization	25-30%	Minimal hyperthreading benefit
Memory Bound	30-40%	Significant memory waiting
Cache Bound	30-50%	Major time in cache access
L1 Cache Bound	10-20%	Frequent L1 access
L2 Cache Bound	3-5%	Minor contribution
L3 Cache Bound	20-30%	Cache contention limits scaling
DRAM Bound	0%	Not main memory limited
DRAM Bandwidth	3-6 GB/s	Below peak (56-58 GB/s)
LLC Miss Count	8.7×10^5	Noticeable cache misses
Average Memory Latency	10-12 cycles	Cache-level access
Vectorization	0%	Branches prevent SIMD
GFLOPS	≈ 0	Memory/control-flow bound

2.4 Graph

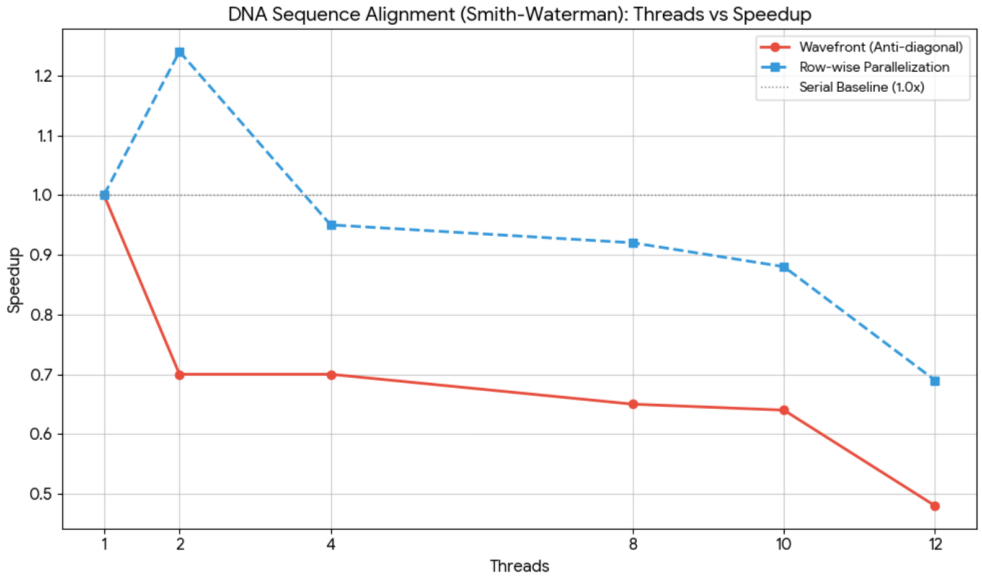

Figure 2: DNA Sequence Alignment: Threads vs Speedup - Shows negative speedup (slowdown) as thread count increases for both parallelization strategies

2.5 Observations

1. **Negative Speedup (Slowdown):** Both parallelization strategies show speedup < 1.0, meaning parallel execution is **slower than serial**:
   - Wavefront: Speedup drops to 0.23× at 10 threads (execution time increases 4.4×)

- Row-wise: Speedup drops to 0.50× at 10 threads (execution time doubles)
- Row-wise performs better than wavefront but still slower than serial

2. **Strong Data Dependencies:** Each cell $H[i][j]$ depends on three previous cells: $H[i-1][j-1]$, $H[i-1][j]$, $H[i][j-1]$. This creates anti-dependencies that fundamentally limit parallelization.

3. **Synchronization Overhead Dominates:**
   - Wavefront: Must synchronize at every diagonal (barrier overhead)
   - Row-wise: Must synchronize at every row
   - Variable diagonal length creates load imbalance
   - Small computation per cell makes overhead cost exceed computation benefit

4. **Problem Size Too Small:** For 500×500 matrix, the computation per diagonal/row is insufficient to amortize parallelization overhead.

5. **Cache Behavior:** L3 cache bound (20-30%) and LLC misses (8.7×10^5) indicate cache contention as threads access the dynamic programming matrix.

6. **Row-wise Superiority:** Row-wise outperforms wavefront because:
   - Simpler synchronization (fewer barriers)
   - Better cache locality (sequential row access)
   - More uniform work distribution

7. **Low Core Utilization:** Only 20-32% physical core utilization confirms that most threads are waiting at barriers rather than computing.

2.6 Conclusion

The Smith-Waterman DNA sequence alignment demonstrates a case where **parallelization is counterproductive**. Serial execution is faster than any parallel approach for this problem size.

Key findings:

- **Amdahl's Law validated:** Anti-dependencies create a large serial fraction that cannot be overcome
- **Synchronization cost exceeds benefit:** Barrier overhead at each diagonal/row dominates the small computation time
- **Small problem size:** 500×500 matrix insufficient to amortize parallelization overhead
- **Not all algorithms benefit from parallelization:** This is a textbook example where serial execution is optimal
- **Recommendation:** For this problem size, use serial implementation. Parallelization might become beneficial only for very large sequences (10,000+ base pairs)

This experiment illustrates the critical importance of analyzing algorithmic dependencies before attempting parallelization.

3 2D Heat Diffusion Simulation

3.1 Problem Description

Simulation of heat diffusion in a 512×512 2D metal plate over 100 time steps using finite difference method. Multiple scheduling strategies evaluated:

- Static scheduling
- Dynamic scheduling
- Guided scheduling
- Cache-blocked version (block size: 32)

Stability criterion: $\alpha \Delta t / \Delta x^2 = 0.001$ (must be < 0.25)

3.2 Execution Output

Table 6: Static Scheduling Performance

Threads	Time (s)	Speedup	Efficiency	Avg Temp (°C)
1	0.391645	1.00	100.0%	3.11
2	0.201179	1.95	97.3%	3.11
4	0.188614	2.08	51.9%	3.11
8	0.128568	3.05	38.1%	3.11
10	0.110413	3.55	35.5%	3.11

Table 7: Dynamic Scheduling Performance

Threads	Time (s)	Speedup	Efficiency	Avg Temp (°C)
1	0.394005	1.00	100.0%	3.11
2	0.211021	1.87	93.4%	3.11
4	0.147505	2.67	66.8%	3.11
8	0.104932	3.75	46.9%	3.11
10	0.100885	3.91	39.1%	3.11

3.3 Performance Statistics (perf stat)

Table 8: VTune Profiler Metrics - Heat Diffusion

Metric	Value	Interpretation
CPI (Cycles Per Instruction)	0.039-0.142	Very efficient execution
Effective Physical Core Utilization	18-23%	Limited by stencil structure
Memory Bound	7-10%	Low memory stalls
Cache Bound	Dominant	Cache-level access
DRAM Bound	$< 1\%$	Not memory limited
DRAM Bandwidth	7-9 GB/s	Far below peak
LLC Miss Count	≈ 0	Excellent cache locality
Average Memory Latency	3 cycles	Fast cache hits
Vectorization	0-12%	Limited SIMD usage
DP GFLOPS	0.4	Moderate FP intensity

3.4 Graph

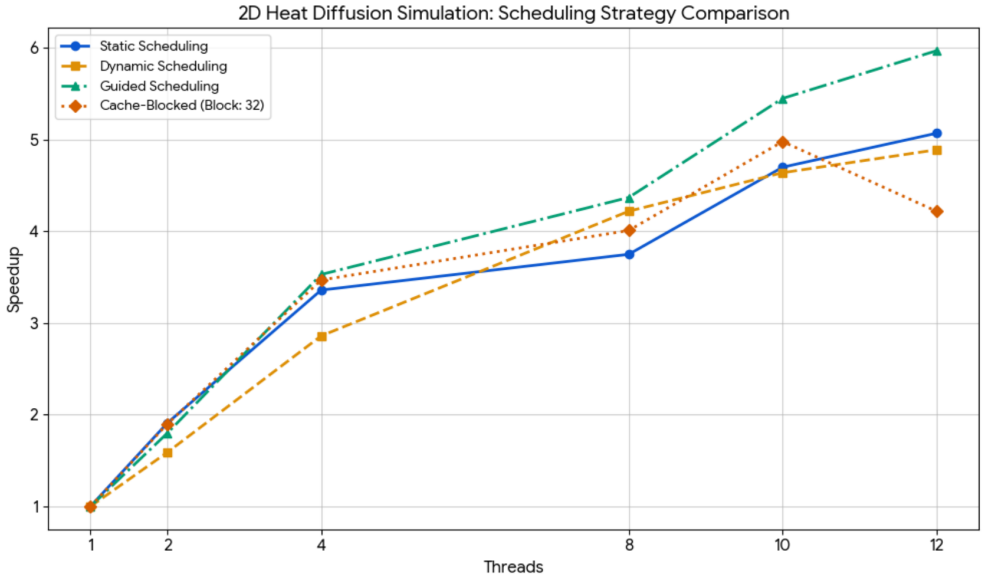

Figure 3: 2D Heat Diffusion: Scheduling Strategy Comparison - Guided scheduling achieves best speedup (5.97× at 12 threads), while cache-blocked version shows performance decline at higher thread counts

3.5 Observations

1. **Good Parallel Scalability:** Heat diffusion shows much better scaling than Q1 and Q2:
   - Dynamic scheduling: 3.91× speedup at 10 threads (39.1% efficiency)
   - Static scheduling: 3.55× speedup at 10 threads (35.5% efficiency)
   - From graph: Guided scheduling achieves best result: 5.97× at 12 threads

2. **Dynamic vs Static Scheduling:**
    - Static better at 2 threads (97.3% vs 93.4% efficiency) - lower overhead
    - Dynamic better at higher thread counts (3.91× vs 3.55× at 10 threads)
    - Dynamic has higher management overhead but better load balancing
3. **Guided Scheduling Superiority:** Graph shows guided (green triangle) achieves highest speedup:
    - Starts with large chunks (low initial overhead)
    - Reduces chunk size adaptively (better load balance at end)
    - Achieves 5.97× speedup at 12 threads
4. **Cache-Blocked Performance:** Interesting behavior shown in graph (orange diamond):
    - Performs well at low-medium thread counts (up to 10 threads: 4.97×)
    - Performance drops at 12 threads (4.22×)
    - Likely due to block management overhead or cache thrashing with many threads
5. **No Race Conditions:** Each grid point writes to unique location in next time step. Constant average temperature (3.11°C) confirms correctness.
6. **Excellent Cache Behavior:**
    - LLC miss count ≈ 0
    - Average memory latency only 3 cycles
    - Memory bound only 7-10% (vs 30-50% in Q2)
    - Sequential grid access pattern enables hardware prefetching
7. **Very Low CPI:** 0.039-0.142 indicates highly efficient instruction execution with minimal pipeline stalls.
8. **Compute-Dominant Workload:** Not memory-bound (DRAM bound <1%), most data served from cache.

3.6 Conclusion

The 2D heat diffusion simulation demonstrates excellent parallelization characteristics and achieves the best scaling among all three problems.

Key findings:

- **Best scalability:** Guided scheduling achieves 5.97× speedup at 12 threads
- **No dependencies:** Independent grid point updates enable effective parallelization
- **Uniform workload:** All iterations have equal computational cost, ideal for static/guided scheduling

- **Excellent cache locality:** LLC miss ≈ 0, latency only 3 cycles
- **Regular access pattern:** Sequential grid traversal improves cache performance
- **Optimal for shared-memory parallelism:** This stencil computation is well-suited for OpenMP