UCS645: Parallel and Distributed Computing
Assignment 03

1. Molecular Dynamics - Lennard-Jones Force Calculation

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.

Execution Output

Table 1: Molecular Dynamics Performance Results

```
Molecular Dynamics: Lennard-Jones Force Calculation
Number of particles: 1000
Cutoff distance: 2.5

Threads   Time (s)       Speedup        Efficiency     Total Energy
================================================================================
1         0.006824       1.00           x100.0          %35352.51
2         0.008944       0.76           x38.1           %35352.51
4         0.004590       1.49           x37.2           %35352.51
8         0.004633       1.47           x18.4           %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
```

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

Observations

5. **Correctness Validation:** Total energy stays fixed at −35352.51 for every thread configuration, verifying correct Newton's third law handling, proper synchronization, and no race conditions.

6. **Speedup Behavior:** Runtime reduces as thread count increases, with peak speedup of 2.86× at 10 threads. Gains level off after about 8 threads.

7. **Efficiency Drop:** Parallel efficiency declines significantly from 80.1% (2 threads) to 28.6% (10 threads), showing diminishing returns caused by:

- Atomic update overhead (six atomic operations per particle pair)
- Synchronization hot spots
- Inter-thread cache contention
- Limits from hyperthreading past 8 threads

8. **Cache-Limited Execution:** The workload is mainly cache-bound (about 35–37% of cycles) rather than DRAM-bound. Observed memory bandwidth (3.4–5.6 GB/s) is well below the system peak of 57 GB/s.

9. **Underused Cores:** Even with a low CPI of 0.58, only roughly 14–22% of physical cores are effectively active because of cache stalls and synchronization costs.

Conclusion

The parallel molecular dynamics simulation delivers a moderate speedup (2.86×) but shows limited scalability beyond 8 threads. The main bottleneck arises from atomic operation overhead during force accumulation, which introduces serialization. The program is primarily cache-bound rather than memory-bound, with available memory bandwidth largely unused.

Key findings:

- OpenMP parallelization yields clear performance gains up to 4 threads
- Scaling stagnates beyond 4–8 threads with minimal additional speedup
- Parallel overhead and synchronization reduce efficiency at higher thread countsCache behavior and memory access patterns constrain performance more than com- putational throughput
- Cutoff distance and Newton's third law optimizations reduce total computation but do not fully solve scaling limits

2. DNA Sequence Alignment (Smith-Waterman)

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

Execution Output

Table 3: Wavefront Parallelization Performance

```
DNA Sequence Alignment (Smith-Waterman Algorithm)
Sequence 1 length: 500
Sequence 2 length: 500
Scoring: MATCH=2, MISMATCH=-1, GAP=-2

=== Wavefront Parallelization (Anti-diagonal) ===
Threads   Time (s)       Speedup        Efficiency
----------------------------------------------------------
1         0.005199       1.00           x100.0%
2         0.023178       0.22           x11.2%
4         0.029100       0.18           x4.5%
8         0.056159       0.09           x1.2%
```

Table 4: Row-wise Parallelization Performance

```
=== Row-wise Parallelization (Simpler, but limited) ===
Threads   Time (s)       Speedup        Efficiency
----------------------------------------------------
1         0.000785       1.00           x100.0%
2         0.006234       0.13           x6.3%
4         0.011041       0.07           x1.8%
8         0.023062       0.03           x0.4%

Algorithm Analysis:
1. Dynamic Programming with O(mn) time complexity
2. Anti-dependencies limit parallelization
3. Wavefront method: process anti-diagonals in parallel
4. Each cell depends only on: H[i-1][j-1], H[i-1][j], H[i][j-1]
```

Observations

3. **Negative Speedup (Slowdown):** Both parallel approaches produce speedup values below 1.0, meaning parallel runs are slower than the serial version.
    - Wavefront: speedup falls to about 0.09× at 8 threads (runtime increases by ~11×).
    - Row-wise: speedup drops to about 0.03× at 8 threads (runtime increases by ~29×).
    - Row-wise is faster than wavefront at 1 thread, but degrades more sharply as threads increase.

2. **Strong Data Dependencies:** Each DP cell H[i][j] depends on H[i−1][j−1], H[i−1][j], and H[i][j−1]. These dependency chains restrict how much work can run concurrently and cap achievable parallelism.
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:** With a 500×500 matrix, the work per diagonal or row is limited, so parallel overhead outweighs compute savings
5. **Cache Behavior:** Shared access to the DP matrix increases cache pressure and coherence traffic, further reducing multi-thread efficiency.
6. **Low Core Utilization:** Poor speedup and efficiency indicate many threads spend time waiting at synchronization points instead of performing useful computation.

Conclusion

Key findings:

- Amdahl's Law is evident: dependency constraints create a large unavoidable serial portion
- Synchronization overhead dominates due to frequent barriers per row or diagonal
- Parallel execution is not always beneficial; algorithm structure and granularity matter
- Parallelization may only help for much larger matrices where computation per synchronization step is significantly higher

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

3. 2D Heat Diffusion Simulation

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

Execution Output

Table 5: Static and Dynamic Scheduling Performance

```
=== static Scheduling ===
Threads   Time (s)       Speedup        Efficiency     Avg Temp
----------------------------------------------------------------------
1         0.059372       1.00           x100.0          %3.11°C
2         0.028959       2.05           x102.5          %3.11°C
4         0.018027       3.29           x82.3           %3.11°C
8         0.020061       2.96           x37.0           %3.11°C

=== dynamic Scheduling ===
Threads   Time (s)       Speedup        Efficiency     Avg Temp
----------------------------------------------------------------------
1         0.105271       1.00           x100.0          %3.11°C
2         0.096672       1.09           x54.4           %3.11°C
4         0.098207       1.07           x26.8           %3.11°C
8         0.119374       0.88           x11.0           %3.11°C
```

Table 6: Guided Scheduling and Cache Block

```
=== guided Scheduling ===
Threads   Time (s)       Speedup        Efficiency     Avg Temp
-------------------------------------------------------------------
1         0.055135       1.00           x100.0          %3.11°C
2         0.029245       1.89           x94.3           %3.11°C
4         0.017415       3.17           x79.1           %3.11°C
8         0.019585       2.82           x35.2           %3.11°C

=== Cache-Blocked Version (Block size: 32) ===
Threads   Time (s)       Speedup        Efficiency
-------------------------------------------------------
1         0.024623       1.00           x100.0%
2         0.013980       1.76           x88.1%
4         0.009513       2.59           x64.7%
8         0.011689       2.11           x26.3%
```

Observations

Good Parallel Scalability: The heat diffusion simulation exhibits strong parallel scalability compared to Q1 and Q2. Execution time decreases significantly as the number of threads increases up to 4 threads, with performance tapering off at higher thread counts due to parallel overheads

Dynamic vs Static Scheduling:

- Static scheduling performs better at lower thread counts because of minimal scheduling overhead. It achieves higher efficiency at 2 threads and delivers the best speedup among the basic scheduling strategies.

- Dynamic scheduling shows poor scalability in this workload. The frequent task redistribution introduces significant overhead, resulting in limited speedup and even performance degradation at higher thread counts.

Guided Scheduling Superiority: Guided scheduling outperforms dynamic scheduling and remains competitive with static scheduling. By assigning larger chunks initially and progressively reducing chunk sizes, it balances workload effectively while keeping overhead lower than dynamic scheduling. This results in consistently better performance than dynamic scheduling across all thread counts.

Cache-Blocked Performance: The cache-blocked implementation improves single-thread execution time substantially, indicating better cache utilization. However, scalability is limited as thread count increases. Performance improves up to moderate thread counts but declines at higher threads, likely due to block management overhead and cache contention among threads

Very Low CPI: The observed CPI values remain extremely low, indicating efficient instruction execution with minimal pipeline stalls

Conclusion

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

Key findings:

- Static scheduling delivers the best overall performance for this uniform workload.
- Guided scheduling provides a good balance between load distribution and overhead, outperforming dynamic scheduling.
- Dynamic scheduling is inefficient due to high runtime overhead.
- Cache blocking improves single-thread performance but limits scalability at higher thread counts.
- Independent grid updates and regular memory access patterns result in excellent cache locality.
- The stencil-based computation is compute-dominant and well-suited for OpenMP parallelization.