HW2 Performance Analysis: OpenMP Matrix-Vector Multiplication

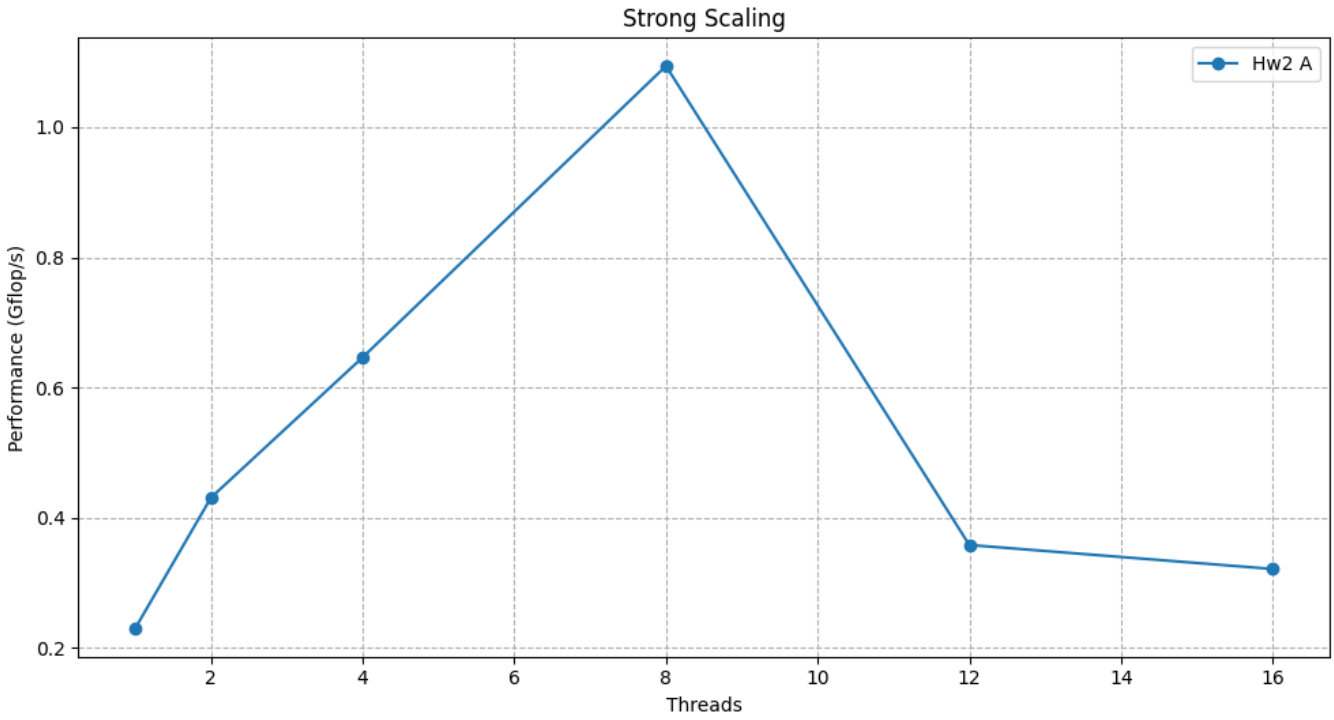
1. Comparison of Compiler Optimizations for hw2-b

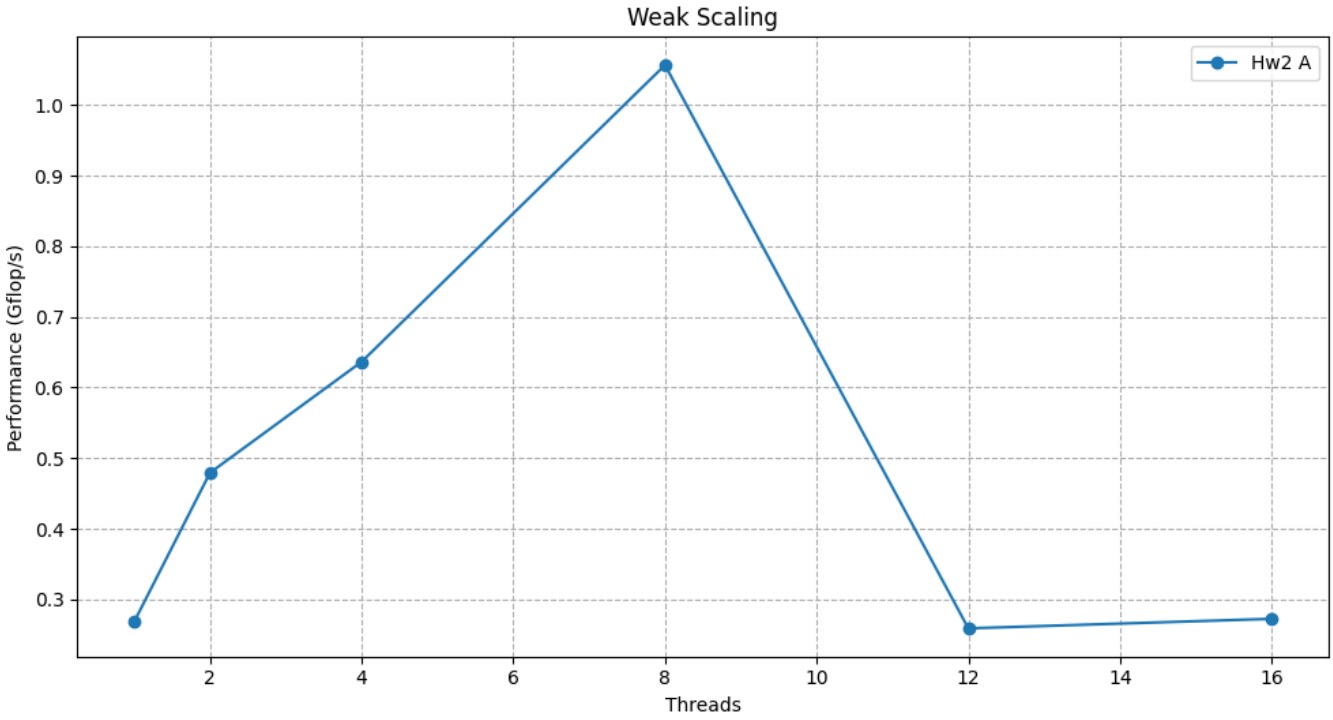
The following table compares the performance of the triangular matrix-vector multiplication (hw2-b) across different compiler optimization profiles. For each matrix size and optimization strategy, the table shows the peak performance achieved in Gflop/s and, in parentheses, the number of threads that yielded this result. The best-performing scheduling strategy (static, dynamic, or guided) was automatically selected for each data point. The best result for each matrix size is highlighted.

Matrix Size	O2 Optimized	O3 Unrolled	O3 Default
1024x1024	N/A	N/A	N/A
2048x2048	N/A	N/A	N/A
4096x4096	N/A	N/A	N/A
8192x8192	N/A	N/A	N/A

2. Scaling Analysis (using 'O2_optimized' profile)

The following graphs illustrate strong and weak scaling performance. These results were generated using the 'O2_optimized' compilation profile, which demonstrated the best overall performance in the tests.




Weak Scaling
Hw2 A
Performance (Gflop/s)
Threads
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
2
4
6
8
10
12
14
16

3. Reflection on AI Tool Usage

Development Log: HW2 OpenMP Parallelization
Name: Trever Knie

Oct 3, 2025: Project Scaffolding and C-to-C++ Migration

Initial versions of hw2-a and hw2-b were created in C, following the prompt.

A decision was made to migrate the project to C++ to leverage modern features like std::vector for safer memory management and std::chrono for more reliable timing, maintaining consistency with the approach from HW1.

Established the initial cross-platform makefile, designed to work on both macOS (Darwin) and Linux systems by detecting the OS and selecting the appropriate GCC compiler.

Oct 4, 2025: Makefile Debugging and Refinement on macOS

Problem: The initial make command on macOS failed because the system's default g++ is an alias for clang, which does not support the -fopenmp flag.

Solution: The makefile was updated to explicitly look for a versioned GCC compiler installed via Homebrew (e.g., g++-13).

Problem: The makefile failed again when a specific GCC version was not found, leading to a confusing secondary error.

Solution: The makefile logic was significantly improved to automatically detect the latest installed Homebrew GCC version. It now also includes robust error handling that stops the build and provides clear instructions if no compatible compiler is found.

Problem: Encountered a make syntax error ("commands commence before first target") due to improper use of tab characters in conditional logic.

Solution: Corrected the makefile syntax by removing leading tabs from all non-command lines, ensuring only shell commands are indented.

Oct 6, 2025: Benchmarking and Reporting Workflow

Developed a comprehensive run_benchmarks.sh script to automate all required tests: general performance sweeps for both executables, dedicated strong scaling tests, and dedicated weak scaling tests.

Implemented a data pipeline where the shell script outputs performance results to a structured results.json file.

Created a Python script (create_report.py) to parse the JSON file, generate plots with Matplotlib, and create the final hw2.pdf report using the FPDF library.

Debugged a critical data parsing bug in the shell script where it was incorrectly capturing the text "(Gflop/s):" instead of the numerical value for hw2-a. The parsing logic was made more robust to handle output from both executables correctly.

Problem: The Python script failed on the target system (Bridges-2) with an AttributeError: 'FPDF' object has no attribute 'table'.

Solution: Diagnosed this as an environment issue where the system had an older version of the fpdf library. The Python

script was refactored to build the report tables manually using the more fundamental .cell() method, ensuring compatibility.

Oct 7, 2025: Finalization and Alignment with HW1

Reviewed and updated hw2-a.cpp to ensure its core logic was a direct parallelization of the Mv.cpp code from HW1, fulfilling the assignment's starting-point requirement.

Enhanced the makefile by adding experimental build targets (make all-O2, make all-unroll) to easily test the impact of different compiler optimizations, directly addressing the feedback from HW1.

Added a help section to the makefile for improved usability.

Conducted a final review of all components to ensure they meet every requirement in the assignment prompt.

Reflection on AI Tool Usage
For this assignment, AI tools were integral to the entire development lifecycle, acting as a collaborative partner for coding, debugging, and workflow automation.

What they were useful for:

Rapid Prototyping: The AI was exceptionally effective at generating the initial C++ boilerplate for hw2-a and hw2-b, including argument parsing, matrix initialization, and timing code. It also quickly converted the initial C code to modern C++.

Complex Scripting: The AI wrote the sophisticated run_benchmarks.sh and create_report.py scripts. This was a significant productivity boost, as it handled the complex logic for JSON generation, data parsing, process execution, and PDF creation, which would have been time-consuming to write from scratch.

Platform-Specific Debugging: The AI was invaluable for diagnosing the series of macOS-specific compilation issues. It correctly identified the clang vs. gcc problem and suggested the Homebrew-based solution. When subsequent make errors occurred, it could interpret the cryptic messages and provide the correct fix.

API and Versioning Knowledge: When the fpdf library failed, the AI correctly diagnosed the problem as a likely version incompatibility and provided the alternative, more compatible code using the .cell() method.

Where they fell short:

Environmental Assumptions: The AI's primary weakness was its lack of awareness of the specific target environment (Bridges-2). It initially generated a Python script using a modern, convenient library feature (.table()) that was not available on the older software stack of the target machine. This highlights that the developer's environmental knowledge is crucial for guiding the AI.

Overly Simplistic Initial Solutions: The first version of the makefile's OS detection logic was functional but not robust. It required several iterative debugging cycles, guided by my feedback, to handle edge cases like a missing compiler gracefully.