

Assignment 2

Gaurav Kukreja
Evangelos Drossos

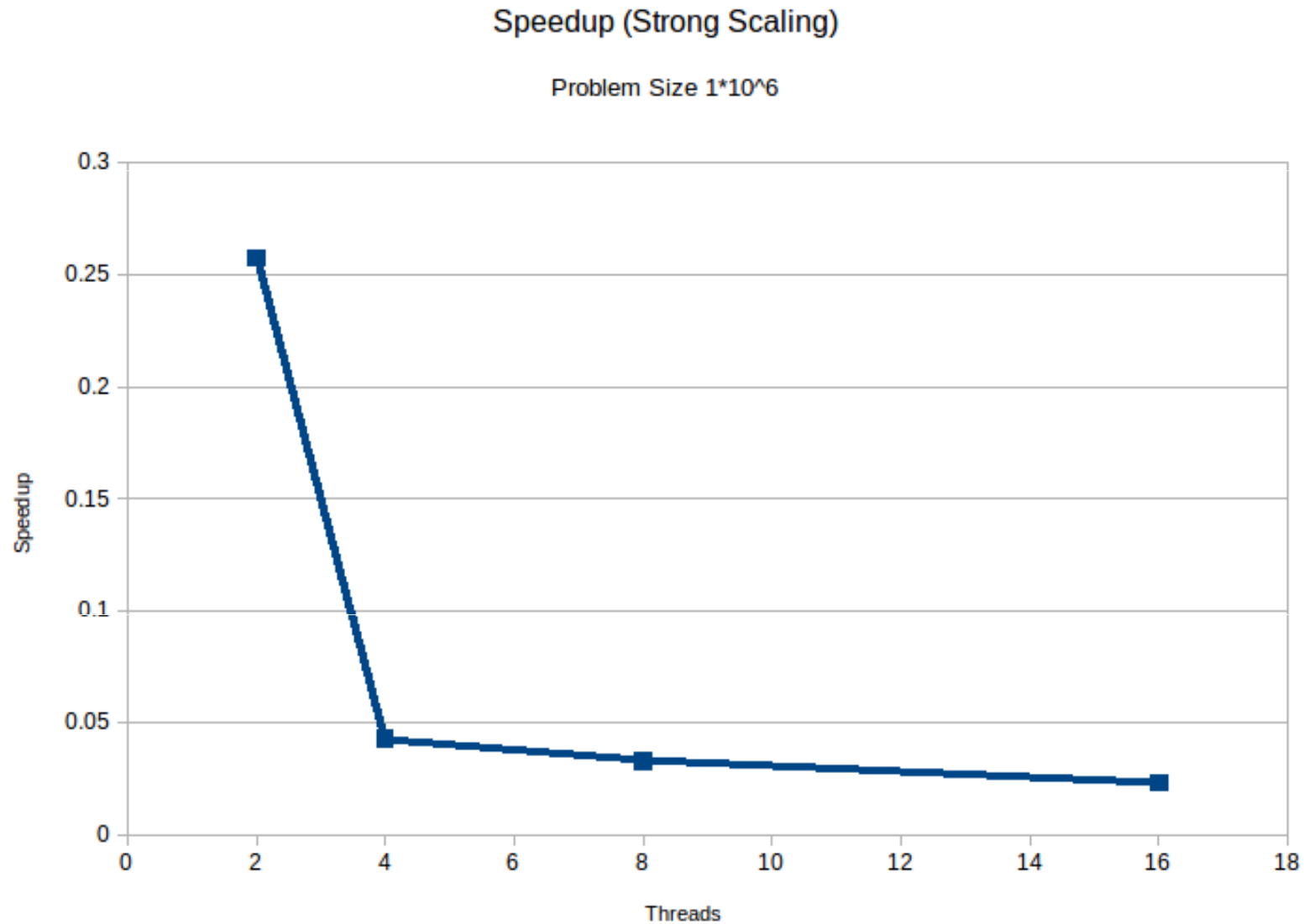
Exercise 5 – Pi Calculation

- Pi Value = 3.141593
- Problem Size tried from $1 \cdot 10^8$ to $8 \cdot 10^8$.
- Important Observation from OpenMP Parallelization
 - Reduction provides near linear speedup
 - Critical is much worse than even Serial Implementation

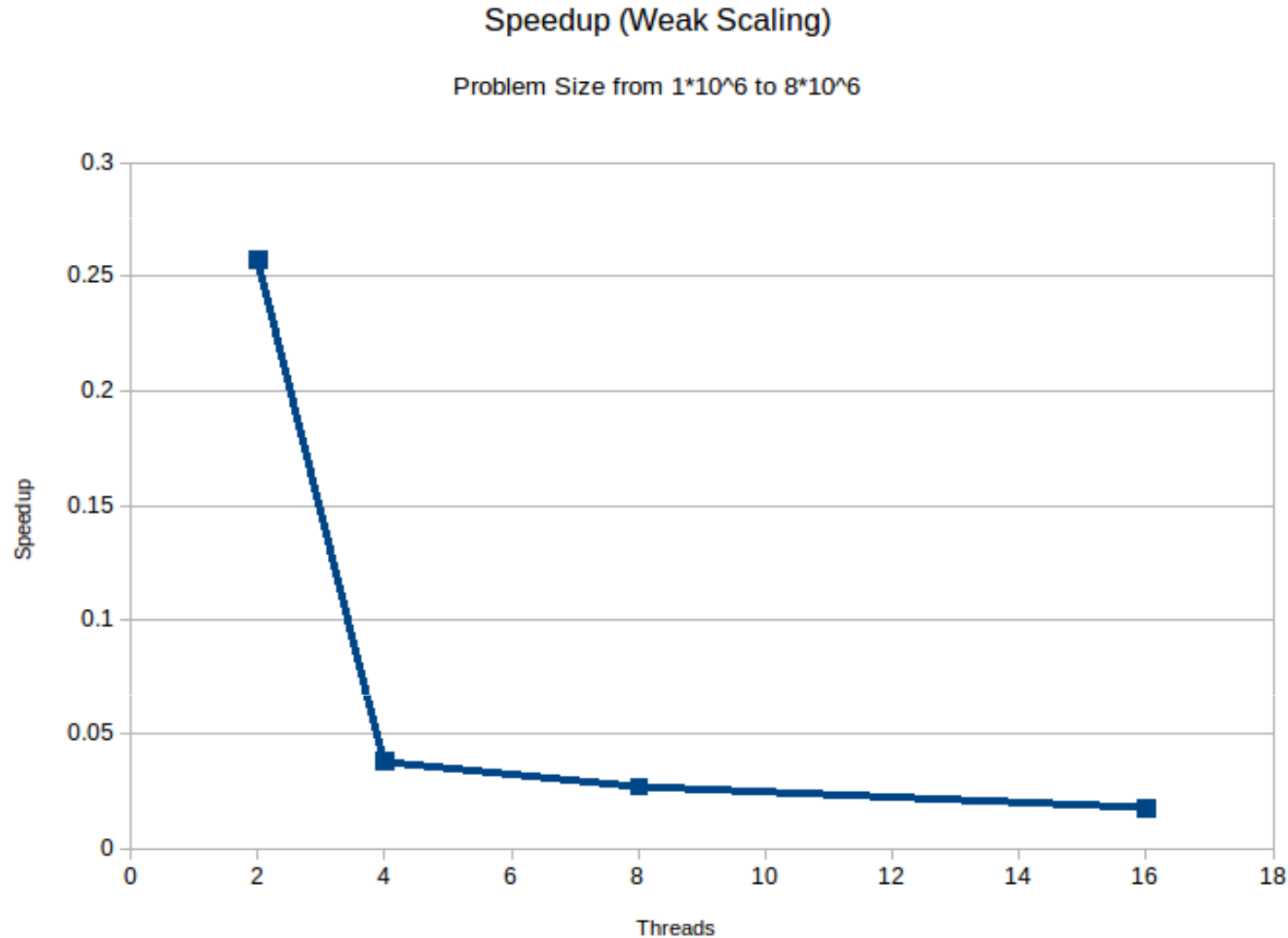
Analysis of result using *critical*

- Using *critical* primitive for combining the values calculated by threads, is disastrous.
- Leads to serialization of code.
- Much worse performance than even Serial Implementation, because threads wait for each other. The synchronization is an overhead.

Analysis of result using *critical*



Analysis of result using *critical*

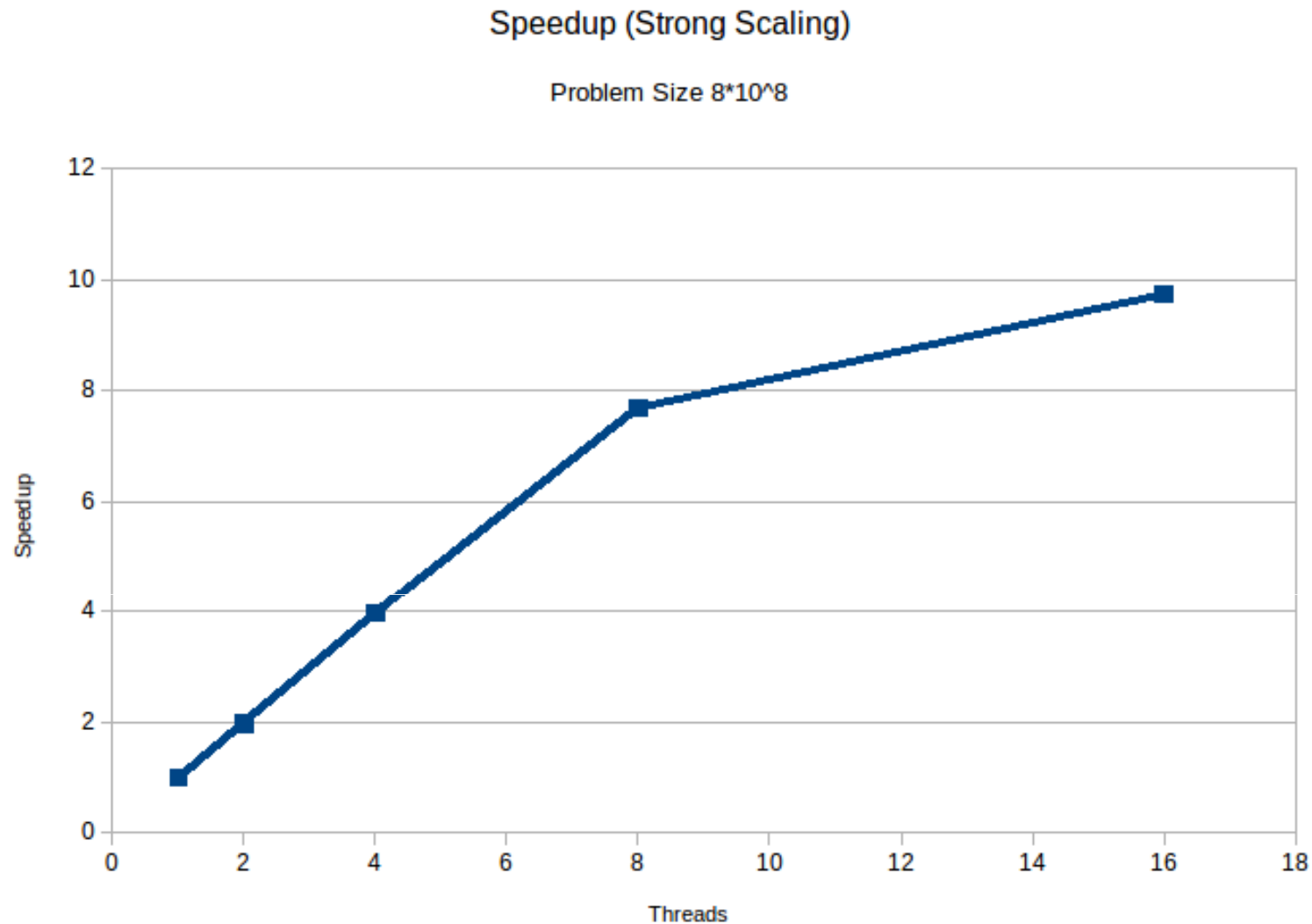


Weak Scaling : The problem size doubles from $1 \cdot 10^6$ to $8 \cdot 10^6$ while number of threads double from 2 to 8.

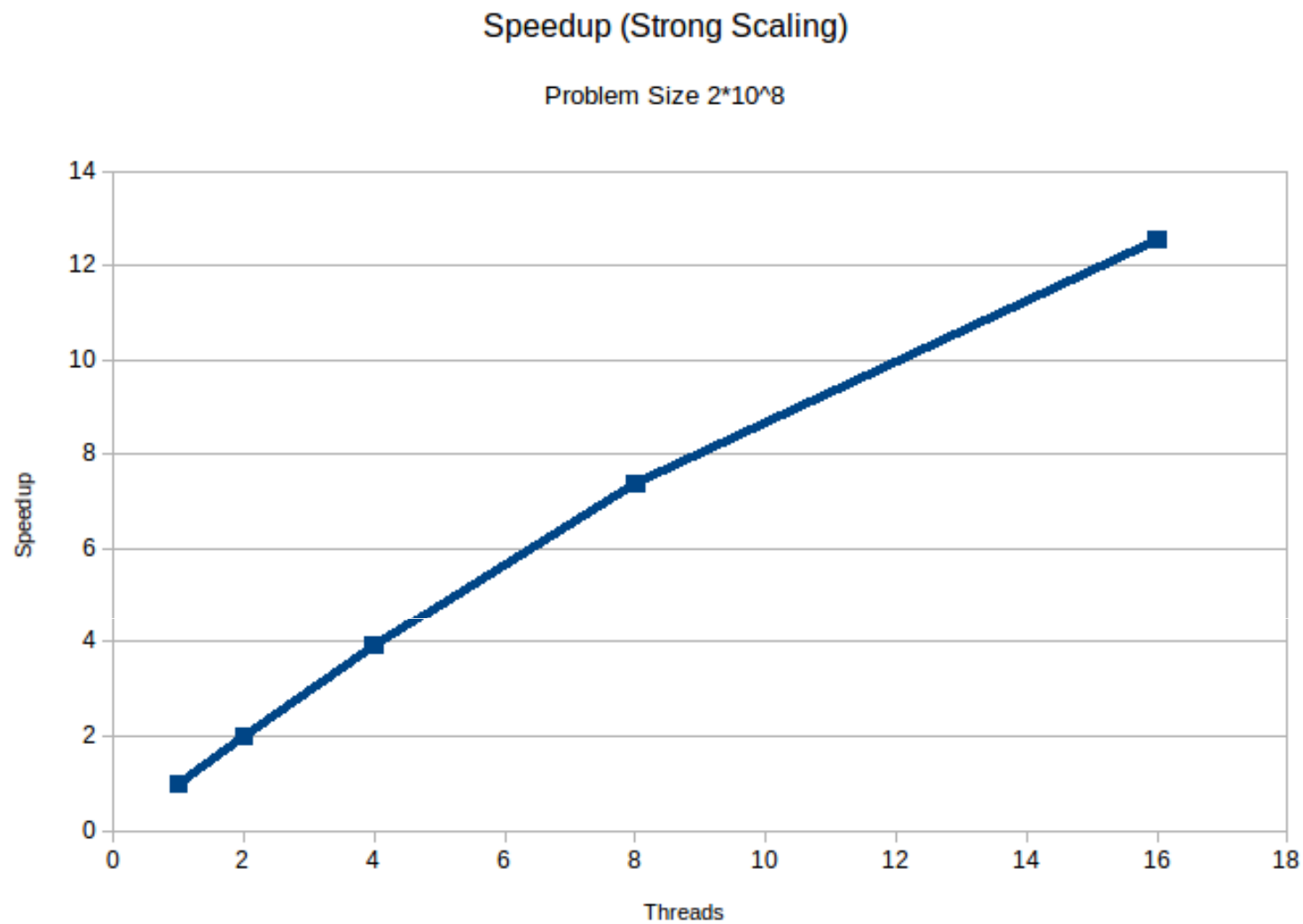
Analysis of result using *reduction*

- Using *reduction* for combining the results obtained by threads is optimal.
- Compiler creates private copies of variable, and combines them after each thread executes the loop.
- Near Linear Speedup achieved.

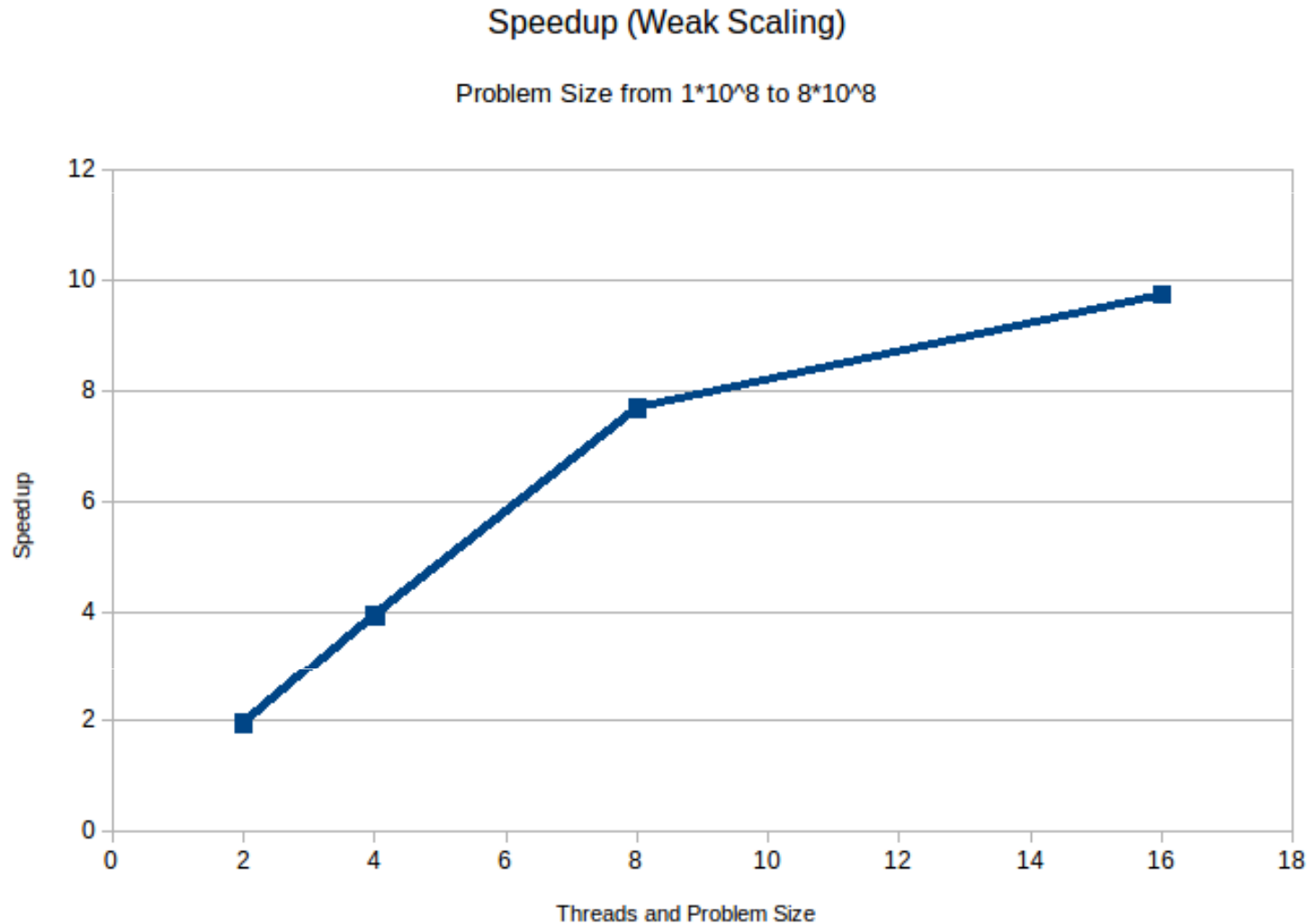
Analysis of result using *reduction*



Analysis of result using *reduction*



Analysis of result using *reduction*



Weak Scaling : The problem size doubles from $1 \cdot 10^8$ to $8 \cdot 10^8$ while number of threads double from 2 to 8.

Exercise 6 – Matrix Multiplication

Optimizations Performed

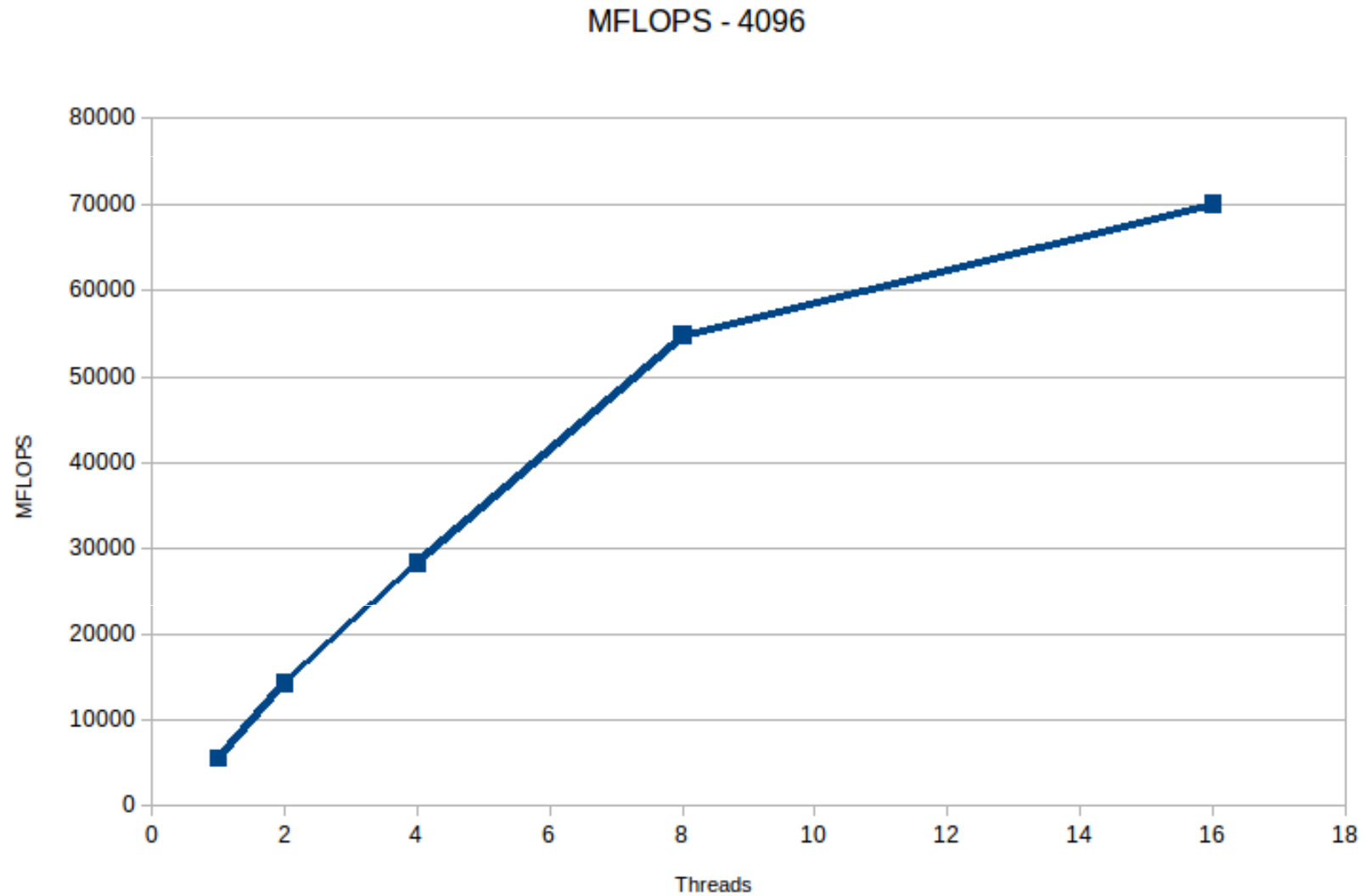
- Vector Intrinsics

- Improved the performance using Vector Intrinsics by calculating 2x8 matrix, in one loop iteration, using 12 vector registers.
- Performance increased from ~4.8 GFLOPs to ~7.9 GFLOPs

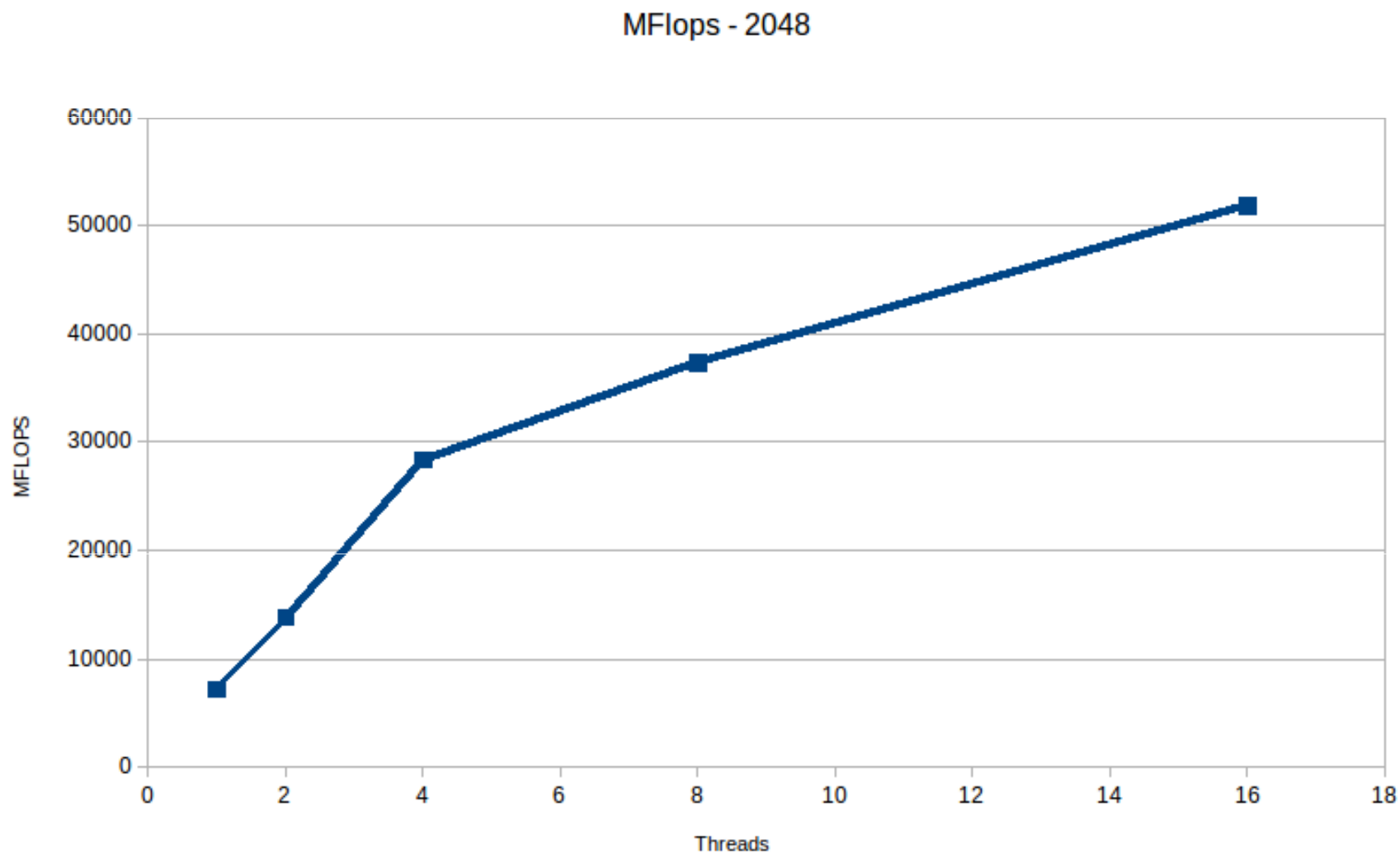
- OpenMP Parallelization

- Parallelized at block level

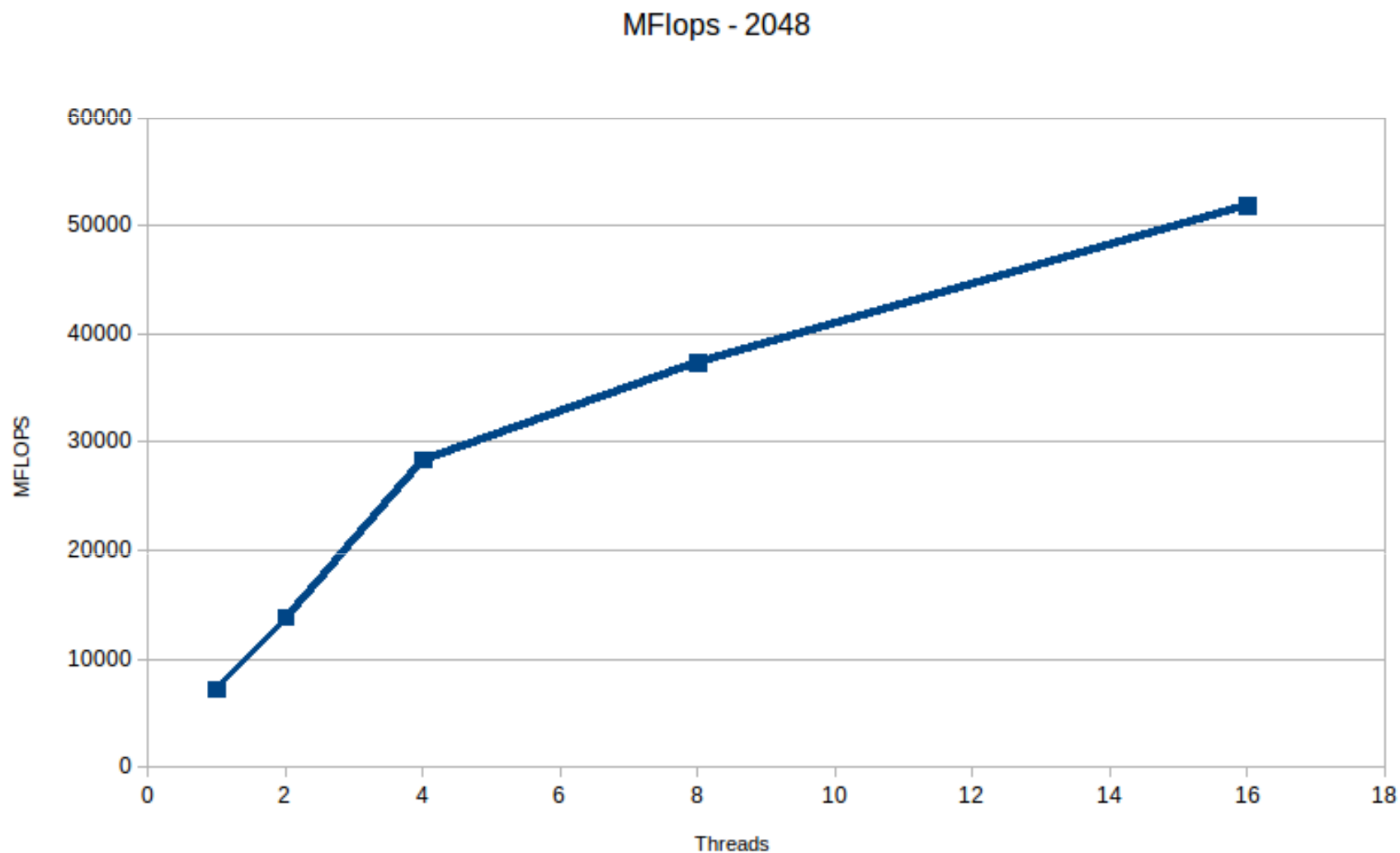
Matrix Multiplication - Speedup



Matrix Multiplication - Speedup



Matrix Multiplication - Speedup



Exercise 7 - Quicksort

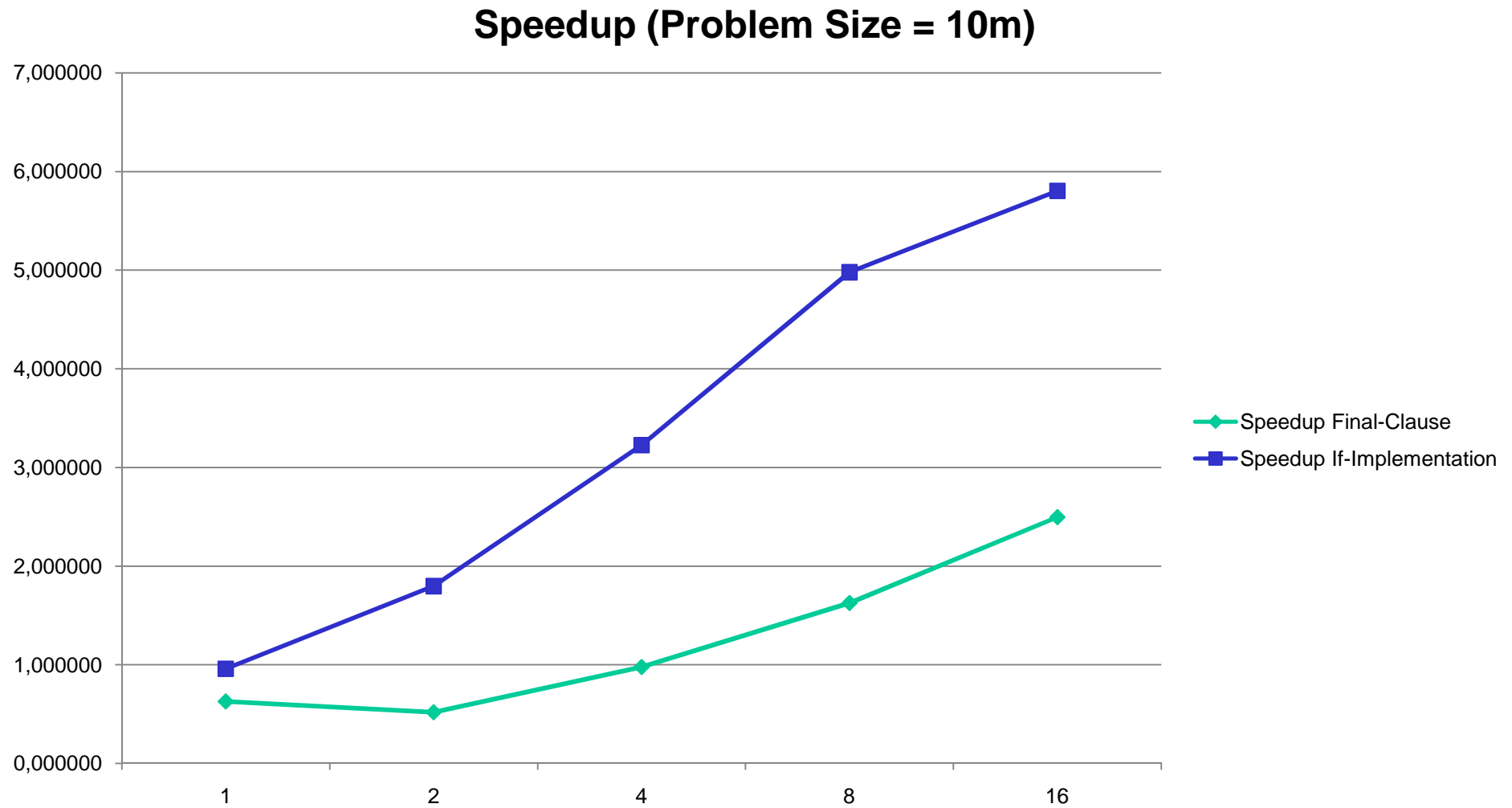
OpenMP tasks used

final clause stops parallelization of recursion when
 $\text{cur_array_length} < (\text{init_length} / \text{max_num_threads} * \text{Factor})$
(final_quicksort.c)

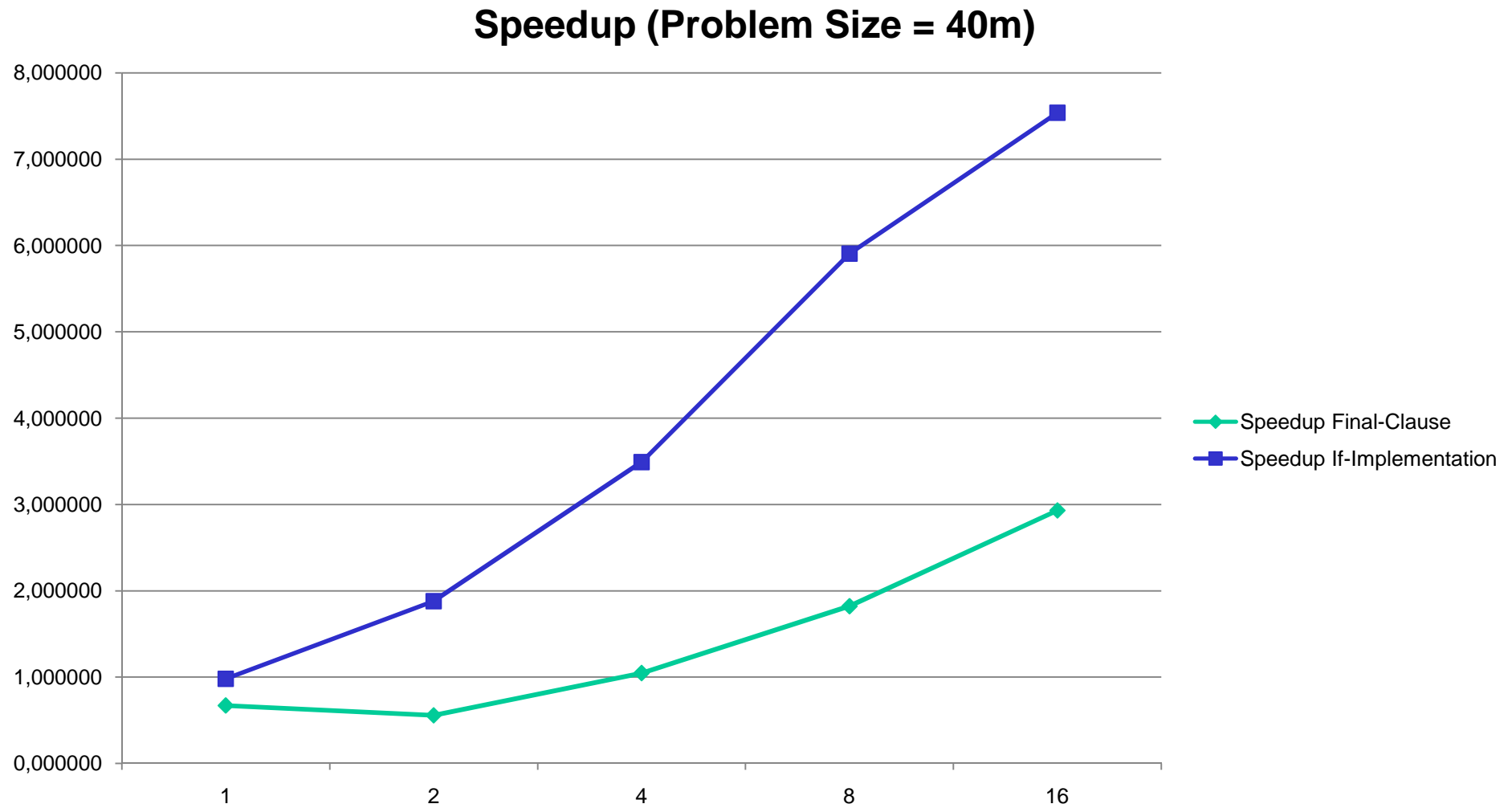
Code implemented using if-else construct in place of final

-> results significantly better for all problem sizes and # of threads tested

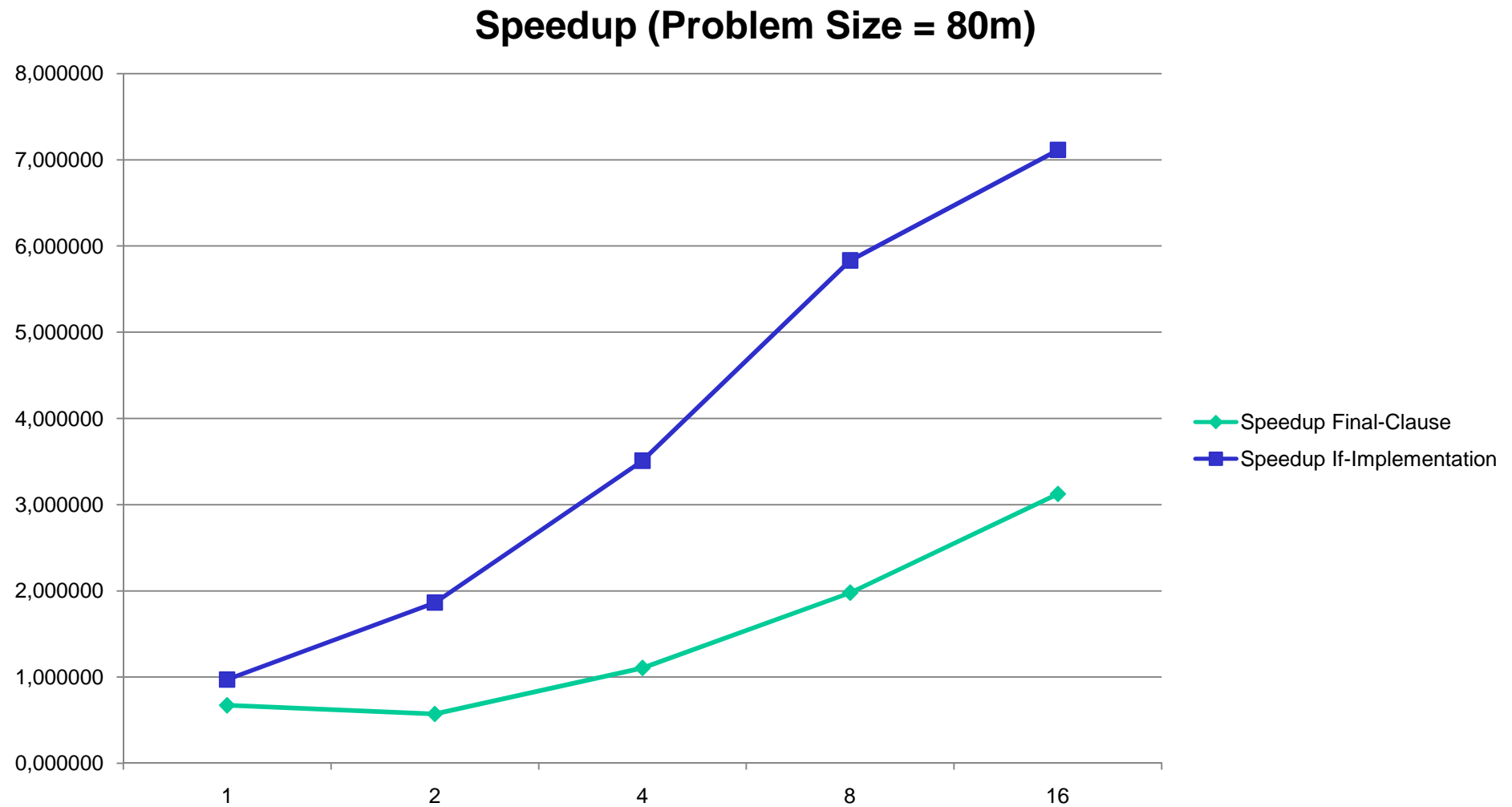
Quicksort – Strong Scaling 1



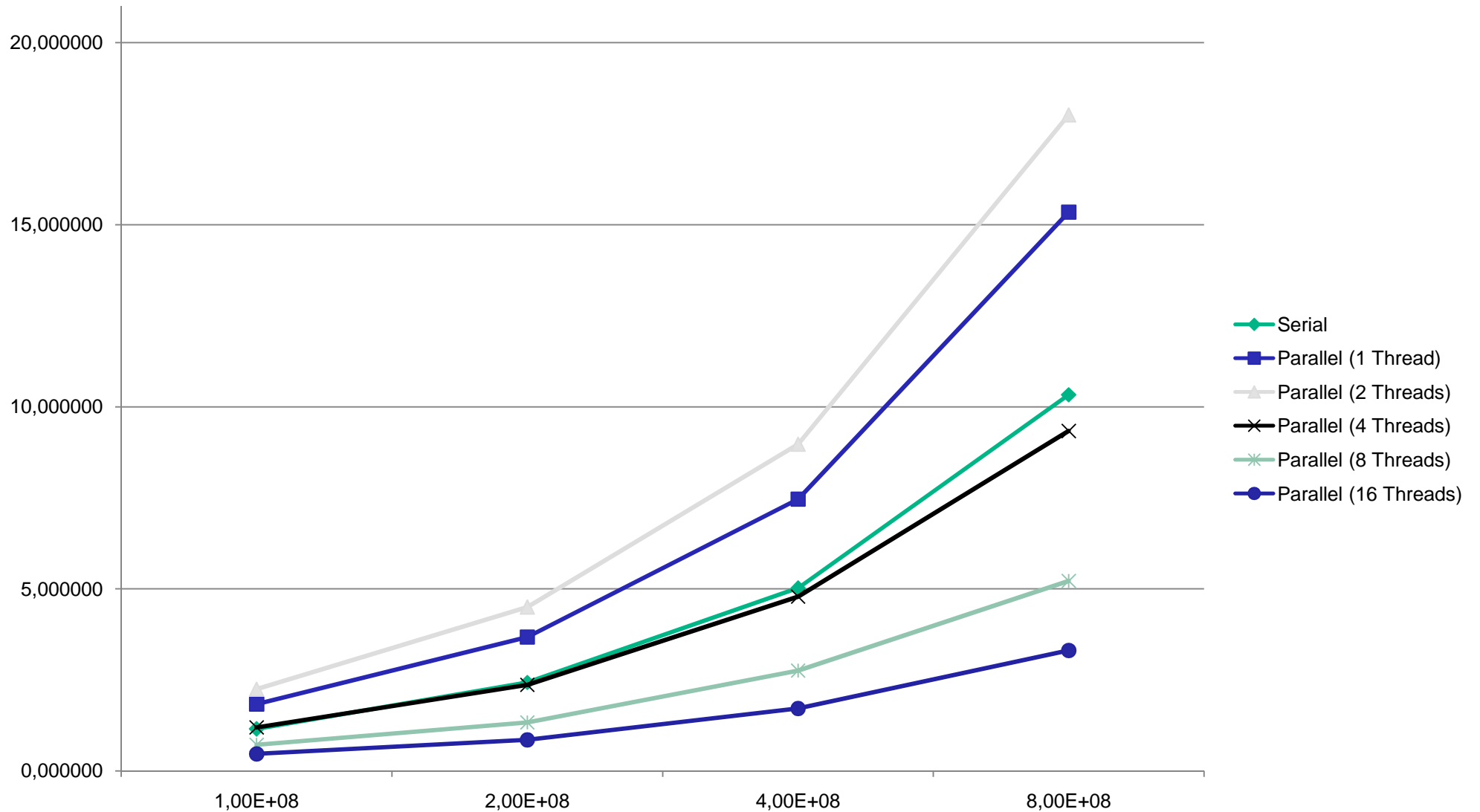
Quicksort – Strong Scaling 2



Quicksort – Strong Scaling 3



Quicksort - Elapsed Time to Different Problem Sizes (*final*)



Quicksort - Elapsed Time to Different Problem Sizes (*if-else*)

