ALGORITHM ANALYSIS:

A COMPARATIVE STUDY OF BAM, SAM, AND SAMK

Lee Place
Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County

Hafsa Chaudry
Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County

Heather DeVal
Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County

Overview

Introduction	• Define the problem • Define the solution
Methods	• Discuss testing conditions • Define the algorithms implemented
Discussion	• Discuss reasoning behind implementation choices • Interpret and synthesize findings
Results	• Regression Analysis
Conclusion	• What we learned

Problem

- Complex operations tend to require a high usage of time and space
  - Such as Matrix Multiplication!
- Practical scenarios require quick implementations!

Solution

- Find the matrix multiplication algorithm that uses time and space most efficiently.
- Implementations of Matrix Multiplication:
  - *BAM*: Basic Matrix Multiplication Algorithm
  - *SAM*: Strassen’s Algorithm
  - *SAMk*: Strassen’s Algorithm with small problem cutoff k

Methods

- How?
  - Examine the results of the BAM, SAM, SAMk algorithm on matrices of various sizes.
    - Which algorithm effectively uses space? Time? Both?
  - Use the results to determine the optimal cutoff value for k
  - Observe where n reaches a crossover value when SAMk overtakes BAM

- Language: Python
- System: UMBC GL Server
- Time and Space:
  - Time:
    - Python function “python –m timeit” returns the running time of a code section
  - Space:
    - malloc
- Input Sizes Tested: 2 through 1028
- Number of Trials:

Matrix Multiplication Algorithms: BAM

- Basic Summation: $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{ki}$
  - Runtime: $\theta(n^3)$
- $BAM(A[\,][n], B\ [n][\,])$

```
    $C[\,][\,] = C(n, n)$
    $for\ x\ =\ 0\ to\ n\ -\ 1$
            $for\ y\ =\ 0\ to\ n - 1$
            $total\ =\ 0$
            $\text{for w}\ =\ \text{0 to z} - 1$
                    $\text{total}\ =\ \text{total}\ +\ A_{wi} \times B_{wi}$
            $C_{ij}\ =\ sum$
    $return\ C$
```

Matrix Multiplication Algorithms: SAM

- Developed by Volker Strassen in 1969
- Reduced matrix multiplication run time from $O(n^3)$ to $O(n^{\log 7})$
- Conditional on the basis that both matrices A and B to be multiplied together are square matrices.
  - In our case, the input was square matrices so we did not have to pad them with zeros

Matrix Multiplication Algorithms: SAMk

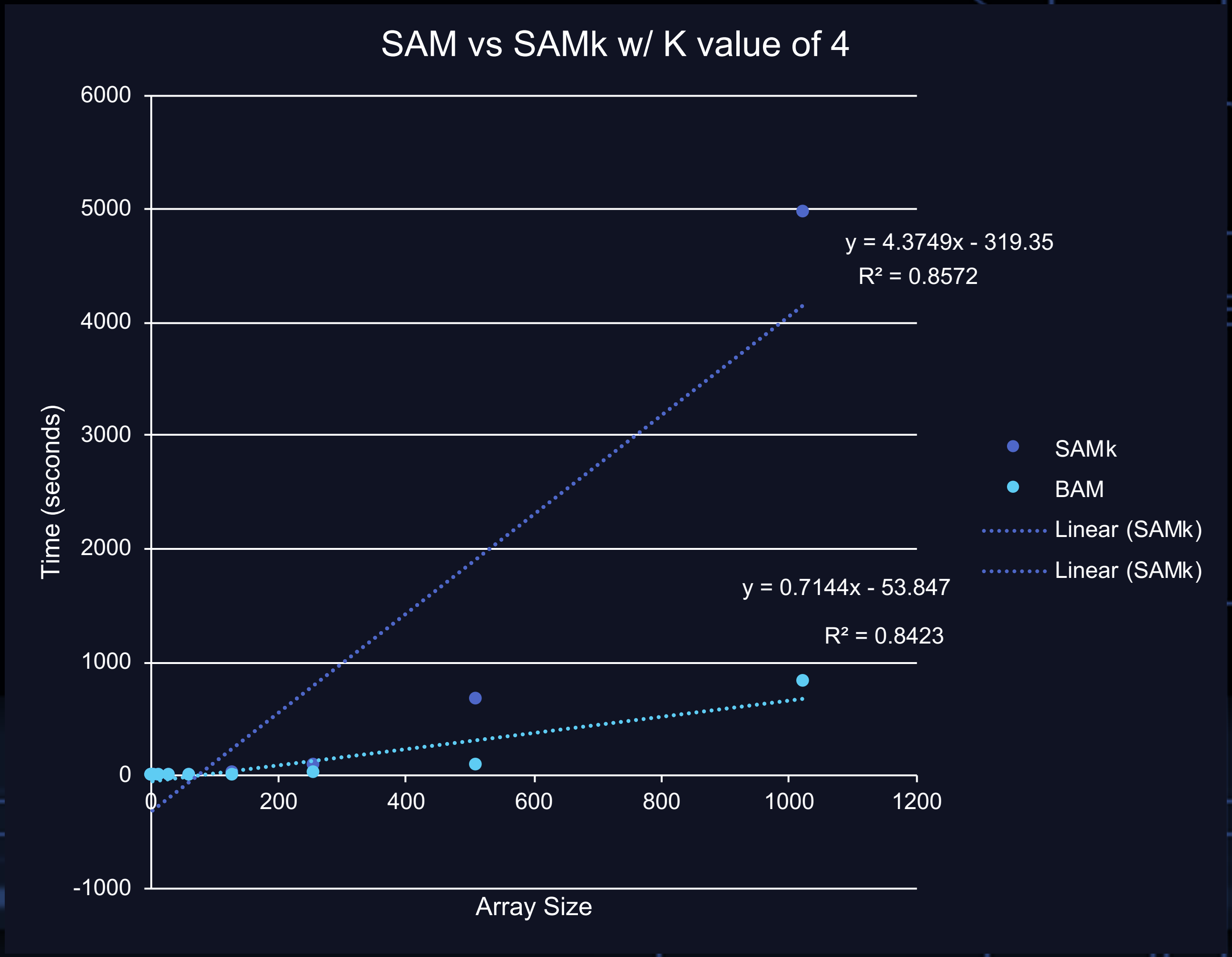
- Modified version of Strassen’s original algorithm
- An improvement over SAM
  - Algorithm solves all of its subproblems less than the small problem cutoff k using the BAM algorithm implementation.
  - k is a parameter passed into the function
- Optimal value of k was found through regression analysis

Memory Management

- How we handle memory:
  - Memory is handled by python and stored in blocks
  - Care was taken to minimize copying data within reason
  - The numpy library was used for several operations, numpy optimizes data types to in general use less memory than a simple array.
- Issues we encountered:
  - We ran into some issues with the data types used in a traditional python lists as the BAM algorithm was producing lists with values larger than what was allowed. Simple casting fixed this issue.

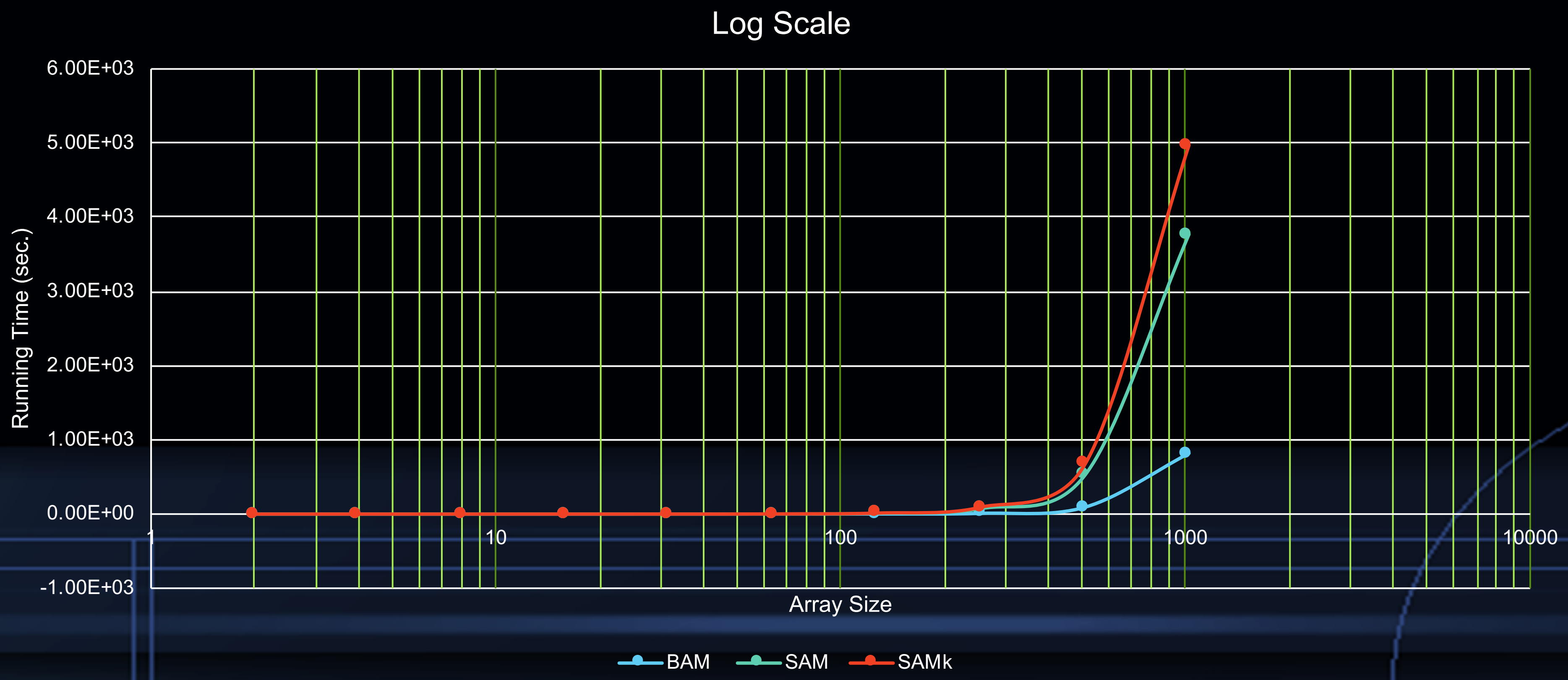
Results

- Optimal Value of k: 4
  - Running time determined:
- For what input size N_0 does the running time of SAMk cross the running time of BAM?


SAM vs SAMk w/ K value of 4

Data Interpretation

- What were our findings?



Regression Analysis

- Constant factors in the theoretical running time of SAMk that best match our data
- Resulting running time:

Conclusion