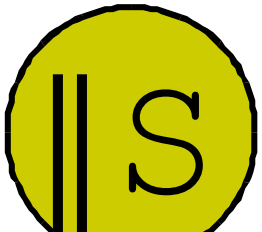
Vector Matrix Multiplication using Parallel Solution Technology

26 *January 2025*

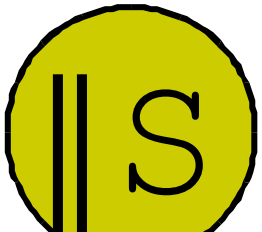
By Parallel Solutions

yehuda.singer@llp.co.il
+972-52-2306-311



Problem Definition

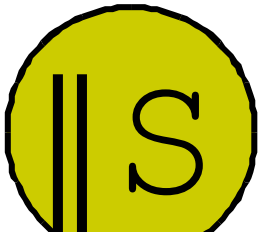
- Multiplication of two large matrices *A* and *B* to produce matrix *C*.
- Every matrix has a minimum size of 1024X1024 elements.
- We want to use vector instructions instead of scalar instructions.
- We use AVX512 *Intel* instructions.



Definitions

- **A** is an $m \times n$ matrix
- **B** is an $n \times p$ *matrix*

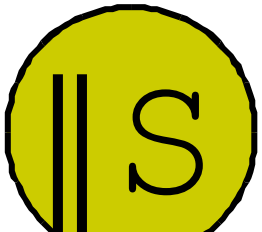
- The result matrix *C is* $m \times p$ *matrix.*



What is a vector?

- ***Vectors*** are used to store multiple values of the same type using a single variable, instead of declaring separate variables for each value.
- Vector used for arithmetic holds values of type:

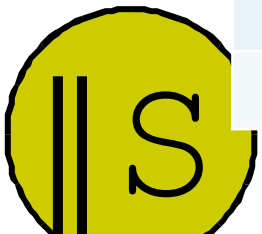
1. Fixed integer*: short*, int, *long* and *long long*.
2. Floating point of various precision: float single precision, float double precision.



Storing Vectors in memory

- Every vector is stored in a series of continued memory slots.
- Each slot holds one item of the vector.
- This fits the memory organization as linear.
- Given a 100 item vector, we get:

Memory slot address	Vector item number
1000	0
1001	1
…	…
1099	99



Matrix organization

- Matrices can be represented by a series of vectors.
- Each vector represents a row.

Given a *m* × *n matrix.*

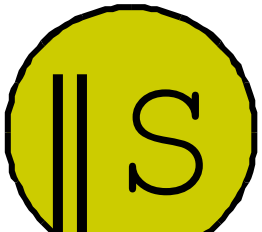
We have *m* vectors. Each vector holds *n* items.

Problem: the computer memory is a one-dimensional array of memory slots. How to store a matrix?

Matrix representation in Memory

- Given a 120X100 matrix.
- We have 120 rows.
- Each row holds 100 items.
- Each row is stored in a vector.

Memory Slot address	Vector Item Number	Items in vector
1000	Vector #0	0..99
1100	Vector#1	0..99
….		
120100	Vector# 119	0..99



Accessing an item of a matrix

- We want to access the item a_{ik} of the matrix.
- There are two steps:

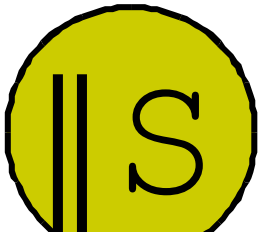
1.To access row *i*, the address is 1000+100**i*. Clearly

2. To access item k the address is (1000+100**i)+k.*

I selects a row

Memory Slot address	Vector Item Number	Items in vector K designating an item of the vector
1000	Vector #0	0..99
1100	Vector#1	0..99
….		
120100	Vector# 119	0..99

k vector item from the selected row



Matrix multiplication

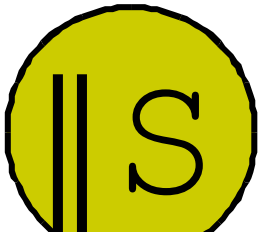
- Every element in matrix *C*,

$c_{ij} = \sum_{k=0}^{n} a_{ik} * b_{kj}$

$i = 1, \ldots, m$ and $j = 1, \ldots, p$.

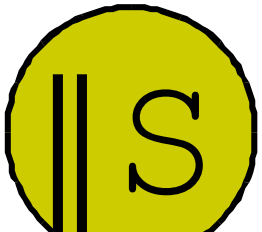
If we want to use vectors:

1. Every row of Matrix *A* is a vector.
2. We want that the columns of matrix B will be vectors. ***Problem***: But the items of a column of matrix *B*, are spread over all the rows.



Solution

- We use ***transposed*** matrix from linear algebra.
- In linear algebra, the transpose of a matrix is an operator which flips a matrix over its diagonal; that is, it switches the row and column indices of the matrix M by producing another matrix, often denoted by M^T



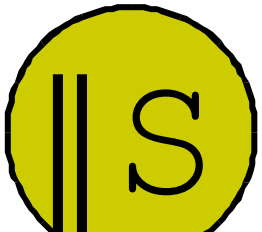
Transposing a matrix

- **Original** matrix B

1	**4**
2	5
3	6

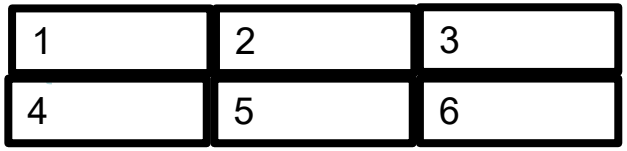
- Transposed **Matrix B**

1	2	3
4	5	6

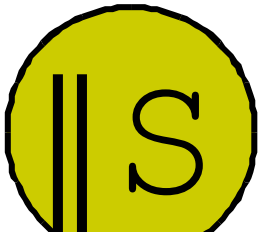


Transposed Matrix Properties

- Transposed matrix B



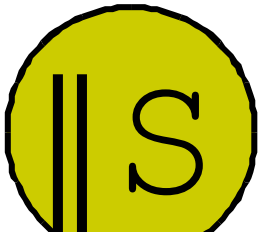
1. Now the transposed matrix B has the columns as rows of the original matrix.
2. Each row of matrix B is a vector.



Multiplying matrices, A & B

- We take the matrix A and treat it as a set of vectors. Each row is a vector.
- We take the matrix B and transpose it. The result is that every Row in matrix B is a vector.

Each element in the generated matrix C is a vector. The multiplication of One vector of matrix A and One vector of matrix B.



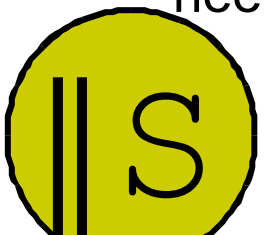
Memory organization

- Each vector is a continued memory area.
- When reading a vector, all its contents is brought to the cache.
- The matrix multiplication is:

  Produce a vector C by multiplying a row from A by all the vectors of B.

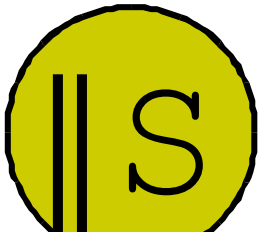
- Notes:

1. Multiplication is using iteration.
2. In each iteration only one row of A is brought to the cache.
3. Matrix B can reside completely in the cache.
4. The elements of matrix C generated each iteration are not needed to be in the cache.



Parallelism

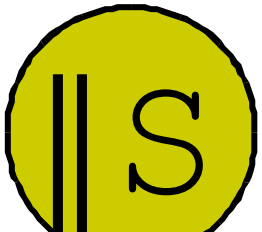
- Each multiplication of a row from A and the vectors of B is a set of operations which are independent.
- Hence, these operations can be performed in parallel.



Cache memory operation

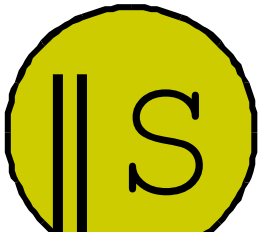
- The cache memory holds at any time;

1. One row of matrix A.
2. Columns of the original Matrix B which are now rows of the transposed matrix.

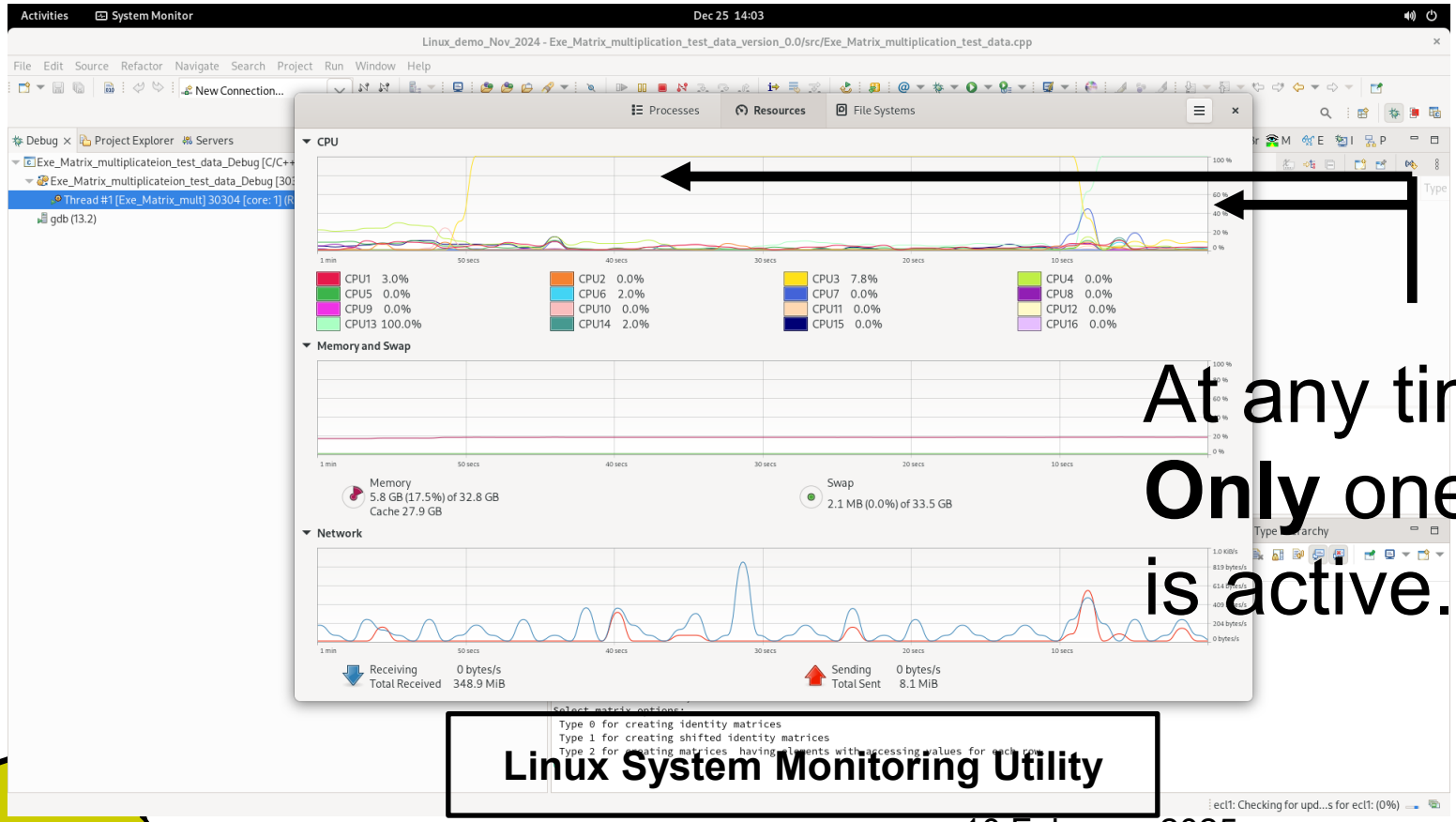


Software & Hardware Configuration

- Computer: Intel *i9*, 16 cores 80x86
- Memory: 32GB
- Operating System: SUSE Linux
- Development tool: Eclipse as an IDE .
- Compiler, Linker & Debugger: GNU tools.
- Language: C++ Version 17.



Sequential Running on a 16 core Processor



At any time, **Only** one core is active.

Linux System Monitoring Utility

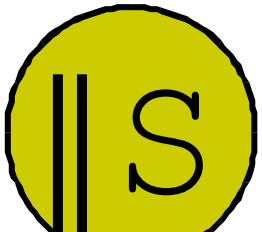
Float Matrix Multiplication using scalar instructions on 16 core machine

Matrix Size	Sequential Program Execution time (seconds)	Parallel Program Execution time (seconds)	Acceleration Ratio
1024X1024	2.029150000	0.051397	39.47993074
2048X2048	32.999700000	3.082414	10.70579747
4096X4096	539.000000000	26.080434	20.666834

Remarks:

1. For 1024X1024 matrices we get execution time which is applicable **to Real-Time AI**.

.

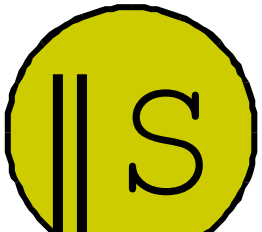


Float Matrix Multiplication with Vector AVX512 instructions

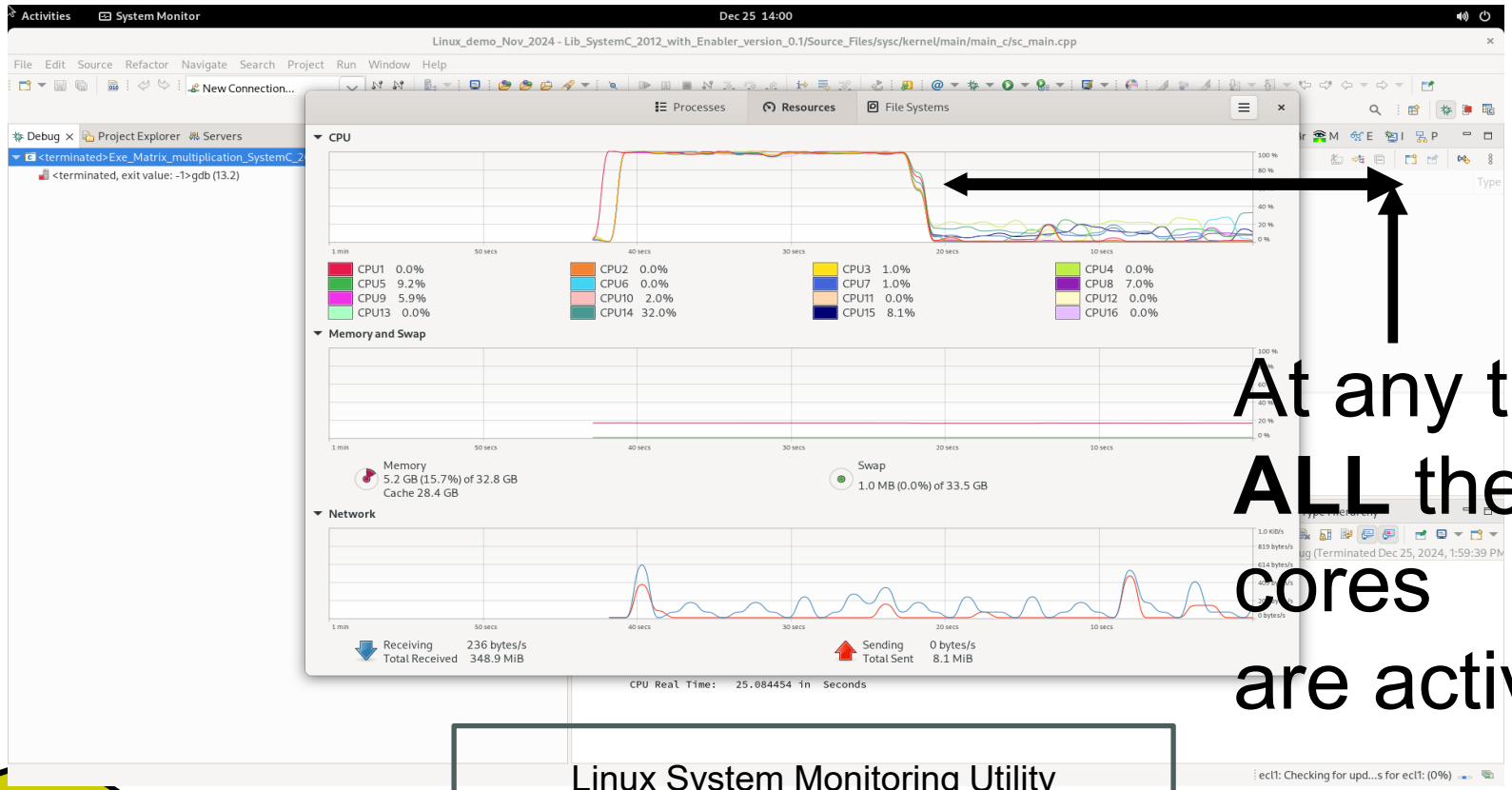
Matrix Size	Sequential Program Execution time (seconds)	Parallel Program Execution time (seconds)	Acceleration Ratio
1024X1024	0.023584390	0.012966	1.818941077
2048X2048	1.981760000	0.090297	21.94713003
4096X4096	14.041200000	6.0554832	2.318757981

Remarks:

1. In spite of smaller acceleration ratio, we get a much faster execution time.
2. For 4096X4096 matrix multiplication, we use the main memory, which slows the process.



Parallel Running on a 16 cores processor

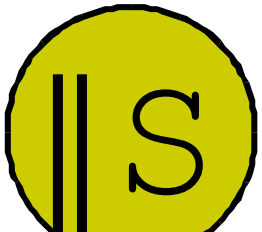


At any time, **ALL** the cores are active.

Linux System Monitoring Utility

Summary

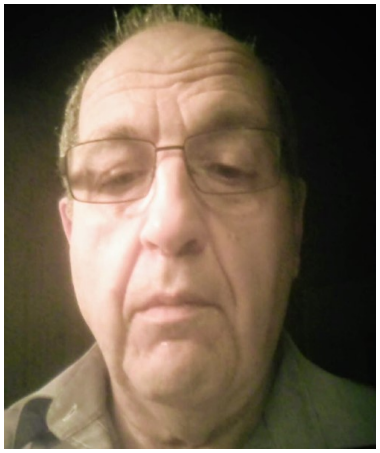
- Our technology provides the acceleration ratio of matrix multiplication with results far better than the number of cores.
- We got far better acceleration ratio than the number of cores.



Parallel Solutions: Leading Team

- **Dr. Yehuda Singer**

Ph.D. in computer science, specialized in computer architecture***, Real-Time*** and high performance. Over 42 years of experience in the development of embedded systems and FPGAs in various multi-disciplinary applications. Dr. Singer was the leader of the Computer Studies of the extension of Derby university in Israel. Dr. Singer holds an M.Sc. and Ph.D. degrees in computer science from Weizmann Institute and Bar-Ilan university respectively.



- **Dr. Joshua Gur**

Joshua - a physicist with vast experience in multidisciplinary system design, Dr. JG has Been the chief Display & Video Engineer for the past 30 years at one security plant of the Israeli Aircraft Industries. He has over 45 years of extensive experience in video systems, electro-optics and multidisciplinary system design and holds 9 patents. Dr. Gur holds a B.Sc.in Physics and Mathematics, M. Sc in Electro-optics from the Hebrew University of Jerusalem, and a Ph.D. in optics from Rochester University of NY USA.