भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Memory contention analysis for co-executing workloads
Mini Project: Jan 2023-May 2023

Lokesh Surana

ES20BTECH11017
Engineering Science
Indian Institute of Technology Hyderabad

Supervisor

Dr. Rajesh Kedia

Department of Computer Science and Engineering
Indian Institute of Technology Hyderabad

Sunday 7th May, 2023

Abstract

This project investigates the impact of multiple processes accessing the same memory on the runtime of each process on the PYNQ Z2 board. We begin by providing a detailed overview of FPGA, PYNQ Z2, and PS/PL components. We then run the BNN (Binary Neural Network) code and several compute/memory-intensive processes from the Polybench repository individually and in pairs, recording their runtimes. Polybench is a collection of standard benchmarks that include both compute-intensive and memory-intensive processes.

Our results indicate that memory-intensive processes exhibit significant spikes in runtime when run in pairs, confirming the existence of memory conflicts. In contrast, the hardware components of BNN, specifically the inference and classification rates, remained consistent across different processes. However, we found that the software component of BNN was affected by the presence of other processes, with its runtime increasing as more processes were run. Our findings are important for practitioners, as they highlight the importance of carefully selecting the processes to be run simultaneously and considering the number of times each process will be run, to optimize overall runtime and minimize conflicts.

Overall, this project contributes to our understanding of the impact of memory conflicts on PYNQ Z2 board performance and provides valuable insights for researchers and practitioners in the field of FPGA-based computing.

1 PYNQ Z2 Board by Xilinx

The PYNQ Z2 board is a powerful development board designed by Xilinx for advanced applications using Xilinx's Zynq chip. It combines a Xilinx Zynq SoC (System on Chip) with an ARM processor and programmable logic in a single device.

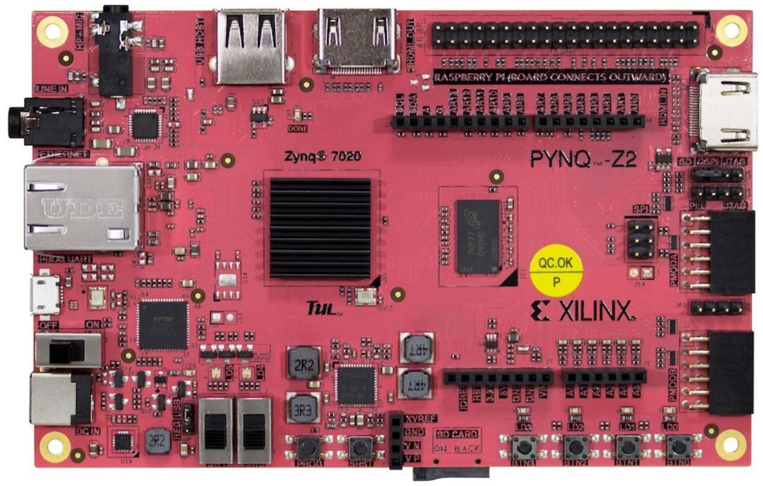


Figure 1: PYNQ-Z2 Board by xilinx

1.1 Features and Specifications

Some key features and specifications of the PYNQ Z2 board are:

1. Xilinx Zynq XC7Z020-1CLG400C SoC
2. 512 MB DDR3 RAM, 256 Mb Quad-SPI Flash
3. Gigabit Ethernet, HDMI input/output
4. 2 x Pmod, 1 x Arduino Shield, 40-pin GPIO
5. Python programming with Jupyter Notebook interface

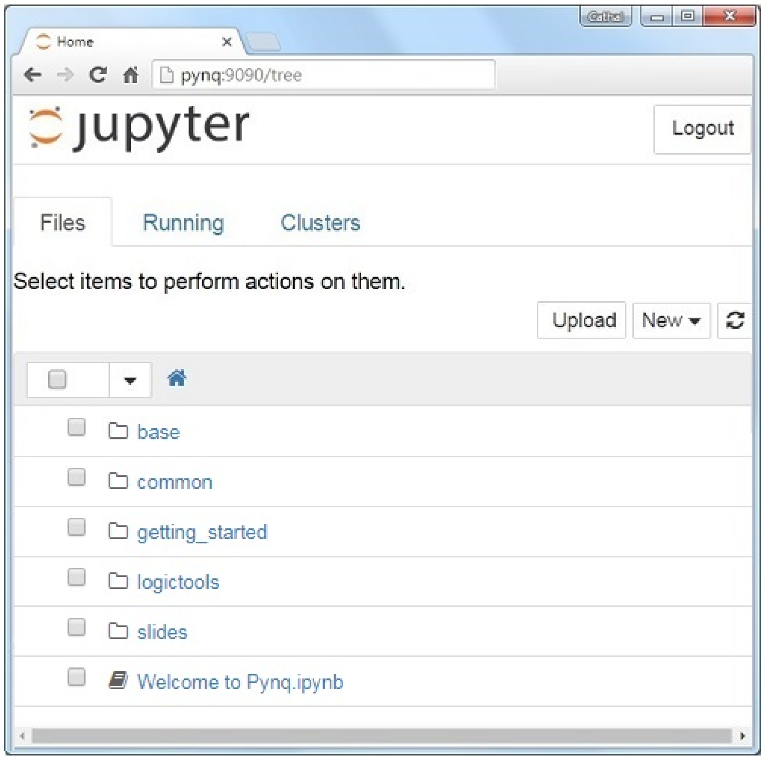
1.2 PS-PL Architecture

The Zynq-7000 All Programmable SoC consists of two main components: the Processing System (PS) and the Programmable Logic (PL). The PS is built around two ARM Cortex-A9 cores and contains a variety of on-chip peripherals, such as memory controllers, USB controllers, Ethernet MACs, and GPIOs. The PL is a highly flexible, programmable fabric that can be configured to implement custom digital logic circuits. The PL communicates with the PS through a high-speed AXI bus, which allows for efficient data transfer between the two domains.
In addition to the PS and PL, the Zynq SoC also contains several other important components, such as the High-Performance Data Mover (HPDM), which provides high-speed DMA transfers between the PS and PL, and the System Management Unit (SMU), which handles power management, clocking, and system reset functions.

1.3 Jupyter Notebook Interface

The PYNQ Z2 board provides a Jupyter Notebook interface, a web-based interactive computing environment that allows users to create and share documents containing live code, equations, visualizations, and narrative text. The Jupyter Notebook interface allows for easy programming in Python and is designed to simplify the development of advanced applications.

Users can interact with the board using a web browser on any device connected to the same network as the PYNQ Z2 board. The Jupyter Notebook interface provides a user-friendly environment for exploring the capabilities of the board and prototyping new applications.



Select items to perform actions on them.

base
common
getting_started
logictools
slides
Welcome to Pynq.ipynb

Figure 2: Jupyter notebook interface for board

2 Setup

2.1 Operating System and Interface

The PYNQ Z2 board comes with the PYNQ operating system, which is a custom Linux distribution built specifically for Xilinx Zynq devices. The PYNQ OS is based on the Ubuntu 18.04 LTS distribution and is optimized for the ARM Cortex-A9 processors in the Zynq SoC.
As mentioned above PYNQ Z2 provides a Jupyter Notebook interface. The Jupyter Notebook interface allows users to interact with the board using a web browser, making it easy to write, test and debug code on the board.

2.2 BNN (Binary Neural Network)

BNN (Binarized Neural Network) is a neural network architecture that uses binary values for network weights and activations, thus making the network more efficient in terms of memory usage and computation time. The BNN-PYNQ repository is an implementation of BNN for the PYNQ platform, which uses a Xilinx FPGA to accelerate the network computations. The repository provides a set of Jupyter Notebooks that demonstrate the use of BNN-PYNQ for image classification tasks using the CIFAR-10 dataset. The notebooks include both software-based and hardware-accelerated versions of the BNN algorithm, allowing for performance comparison and evaluation.

The BNN-PYNQ repository also provides a library of hardware-accelerated functions for convolution, pooling, and fully connected layers, allowing for efficient implementation of BNN in FPGA hardware. Additionally, the repository provides a tool for quantizing pre-trained neural network models to binary weights and activations, allowing for easy deployment of BNN on the PYNQ platform.

The repository can be found here: `https://github.com/Xilinx/BNN-PYNQ`

2.3 Polybench

Polybench is a collection of kernels and solvers for a variety of numerical algorithms that are widely used in scientific computing. It provides a standard benchmark suite to evaluate the performance of computer systems for these algorithms.

The repository can be found here: `https://github.com/Meinersbur/polybench`

2.4 Connecting and Setting Up the Board

To connect to the PYNQ Z2 board, the board needs to be connected to a network via Ethernet. The board's USB-serial connector can then be connected to a computer, and a terminal program, such as **PuTTY**, can be used to connect to the board. The IP address can be found by typing the command "ifconfig" in the terminal.

To set up the PYNQ Z2 board for the project, we used the pre-installed **PYNQ-Z2 v2.5 image**, which was downloaded from the official PYNQ website. The image was then flashed onto an SD card using the **balenaEtcher** tool. The SD card was then inserted into the PYNQ Z2 board.

Note: The PYNQ-Z2 board is currently available in version 2.7, but the BNN/PYNQ repository has been archived and was last configured for version 2.5. Since we wanted to use the notebooks available in that repository to check for memory contention, we opted to use version 2.5 of PYNQ-Z2.

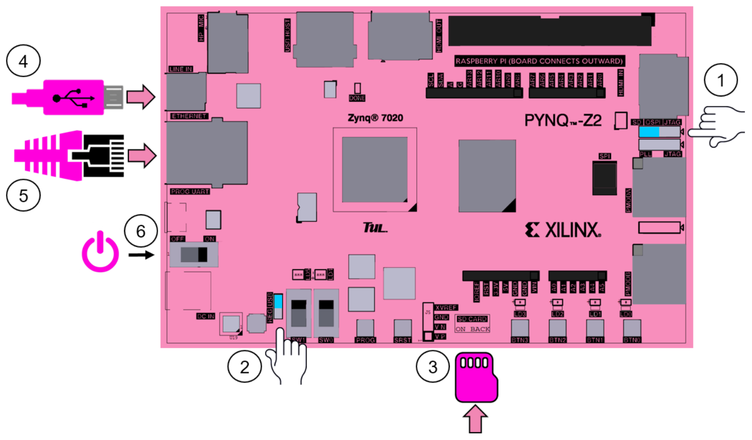


Figure 3: Board setup

2.5 Accessing Board and Jupyter Notebook Interface

To access the Jupyter Notebook interface on the PYNQ Z2 board, open a web browser on a computer connected to the same network as the board and enter the IP address assigned to the board followed by ":9090".

For example, if the IP address of the board is 192.168.1.10, enter `http://192.168.1.10:9090` in the web browser. This will open the Jupyter Notebook interface, where Python code can be written and executed. The Jupyter Notebook interface allows users to interactively write, test, and debug Python code, making it an ideal interface for developing and testing machine learning models on the PYNQ Z2 board.

3 Runtime observations

3.1 Polybench algorithms

The algorithms in Polybench can be grouped into several categories:

1. Linear Algebra Kernels: These are kernels that perform operations on matrices and vectors, such as matrix multiplication (gemm), vector multiplication and matrix addition (gemver), scalar, vector and matrix multiplication (gesummv), symmetric matrix multiplication (symm), triangular matrix multiplication (trmm), and triangular solver (trisolv).
2. Solvers: These are kernels that solve linear systems, such as Cholesky decomposition (cholesky), LU decomposition (lu), LU decomposition followed by forward and backward substitution (ludcmp), Toeplitz system solver (durbin), Alternating Direction Implicit solver (adi), and Dynamic programming algorithm for sequence alignment (nussinov).
3. Stencils: These are kernels that perform computations on a 2D or 3D grid, such as 2D and 3D Jacobi stencil computation (jacobi-1D and jacobi-2D), 2D Seidel stencil computation (seidel), and 3D Heat equation over data domain (head-3d).
4. Miscellaneous: These are kernels that do not fit into any of the above categories, such as matrix transpose and vector multiplication (atax), BiCG sub-kernel of BiCGStab linear solver (bicg), covariance computation (covariance), correlation computation (correlation), edge detection filter (deriche), 2D finite different time domain kernel (fdtd-2d), Gram-Schmidt decomposition (gramschmidt), and symmetric rank-k operations (syrk) and symmetric rank-2k operations (syr2k).

3.2 Methodology to measure runtime of Polybench algorithms

We measured the execution time of each algorithm by running their respective executables separately, without initiating any other processes ourselves, while allowing for system processes that may have been running at the time.

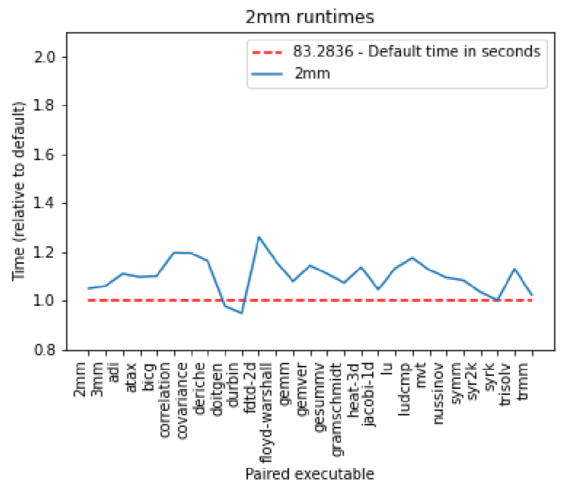
1. The runtimes for the benchmark programs vary widely, ranging from less than a millisecond to over 10 minutes.
2. Programs that involve matrix multiplication operations tend to have longer runtimes than those that involve simpler operations, such as vector addition or multiplication.
3. To measure the pairwise runtimes of each algorithm pair, we adjusted the size of datasets (N) used in algorithms that required long execution times, such as lu and ludcmp. Specifically, we reduced N from 2000 to 1500 for lu and ludcmp, and for floyd-warshall, we adjusted the dataset size by reducing N from 2800 to 2000. This allowed us to ensure that each algorithm could be executed within a reasonable timeframe of a few minutes, even when run individually.
4. The python script used to measure runtimes compiled all the algorithms with the -O3 compiler optimization and saved them in a single directory. The code for python script can be found here: `https://github.com/Lokesh8203/ES3305/blob/main/run.py`

5. We encountered an issue with the script, where each instance of code was only being run once. This led to an inappropriate methodology for drawing conclusions, as some algorithms were given more processing time than others. For example, if the "lu" algorithm took 5 minutes to complete and was paired with the "atax" algorithm, which stopped after 0.5 seconds, the "lu" algorithm would continue to run for the rest of the allotted time as if there were no other processes to compete with. As a result, we had to modify our methodology to ensure fair comparisons between algorithms.
6. We created a new Python script that ran two executables a total of m and n times, respectively. This approach ensured that the comparison of m times the runtime of executable one with n times the runtime of executable two was valid and comparable. Here is the code for that python script `https://github.com/Lokesh8203/ES3305/blob/main/run_iterations.py`
7. The number of executions for each pair of executables, m and n, were passed as arguments to the script. These values were calculated using the known individual runtimes, and the criterion was set such that each algorithm should have completed at least two iterations. The notebook containing the code for calculating and saving the values of m and n in a text file can be accessed through this link `https://github.com/Lokesh8203/ES3305/blob/main/iteration_count.ipynb`. The text file with values for m and n for each pair of executable can be found here `https://github.com/Lokesh8203/ES3305/blob/main/pairs.txt`

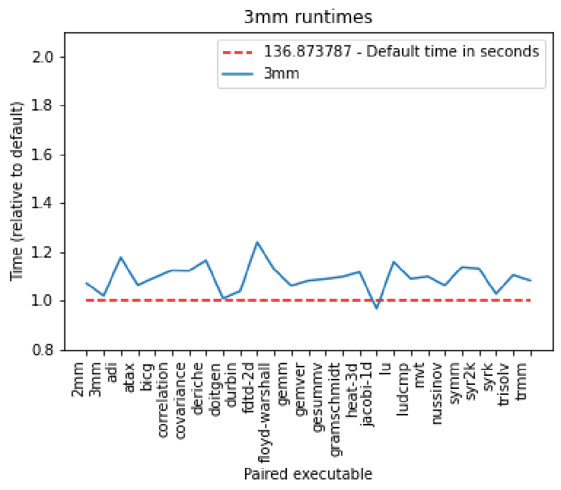
3.3 Plot for runtime for each algorithm

We have generated plots of the runtime of each algorithm using a line chart. The red line in each plot represents the default runtime of the algorithm when run individually, which has been normalized to one. The plotted points represent the relative runtime of the algorithm when run with a specific process, which is represented on the x-axis of the plot.

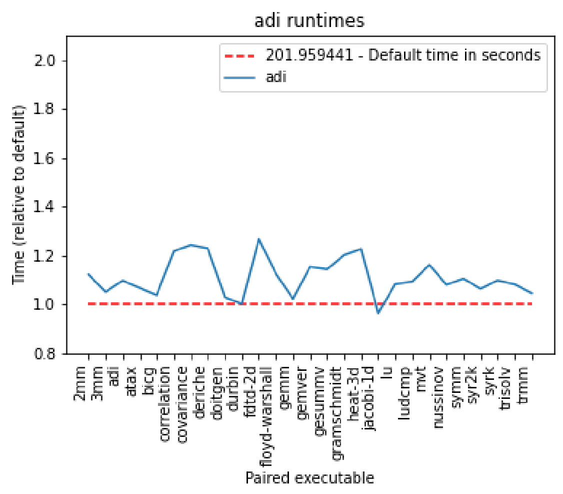
To extract the runtime data and generate the plots, we used a Jupyter Notebook, available here `https://github.com/Lokesh8203/ES3305/blob/main/time%20trends%20(Iterations).ipynb`



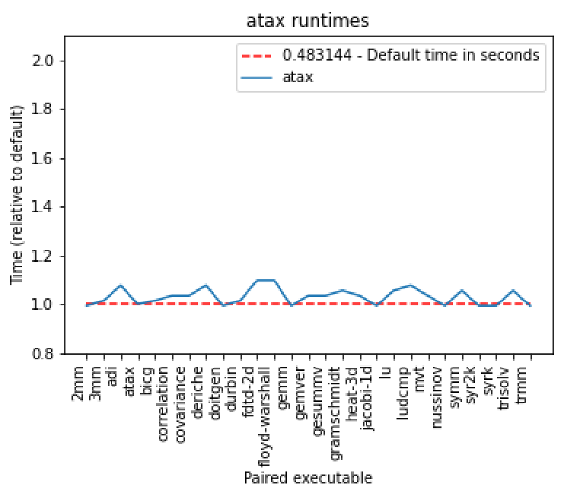
2mm



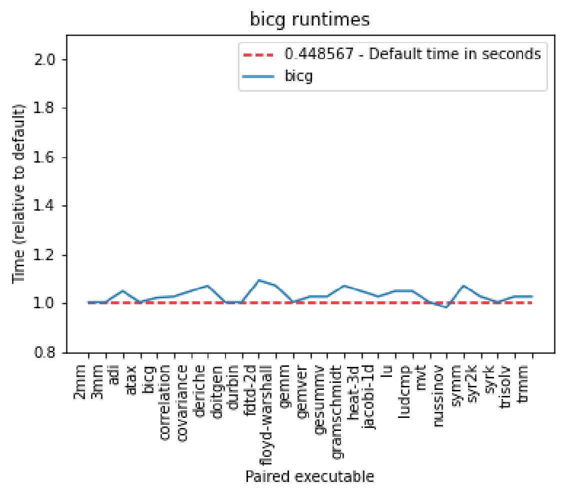
3mm



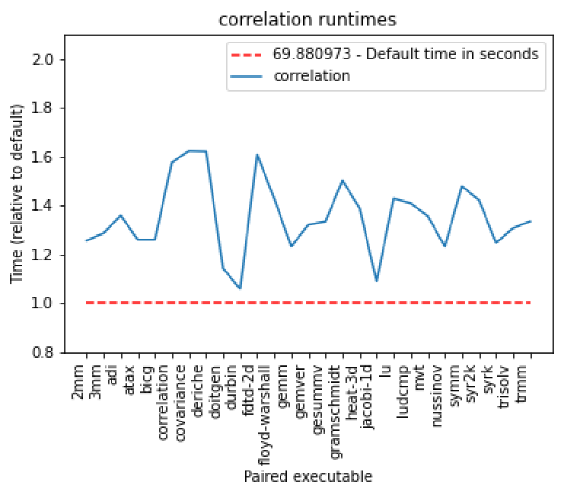
adi



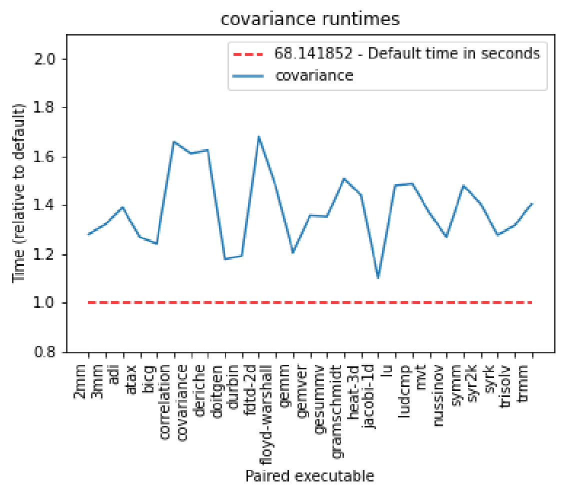
atax



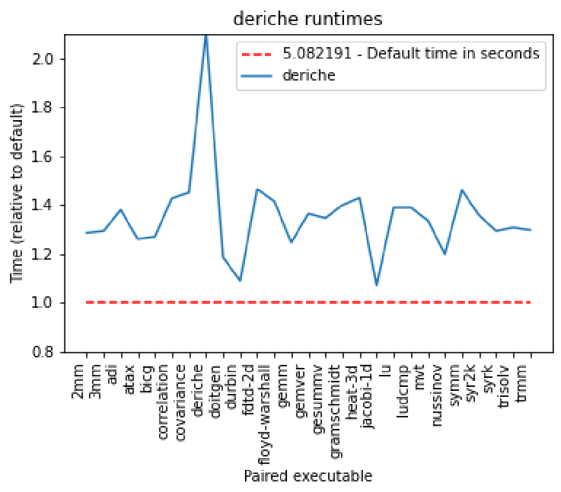
bicg



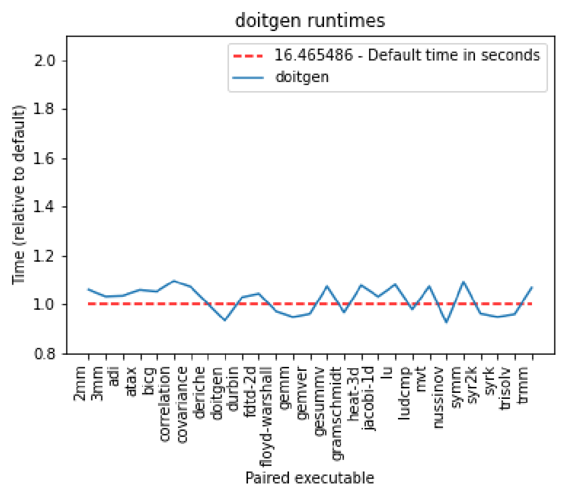
correlation



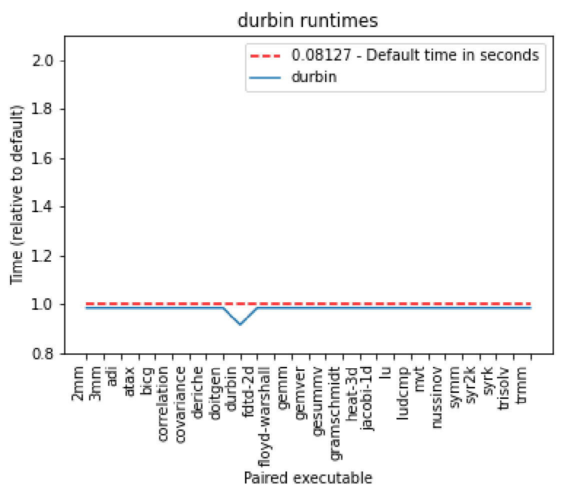
covariance



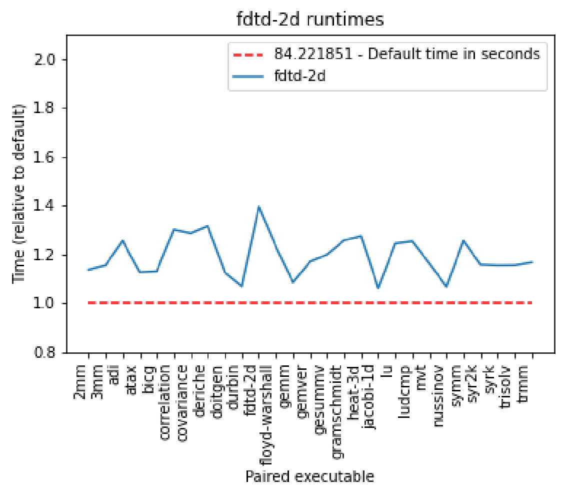
deriche



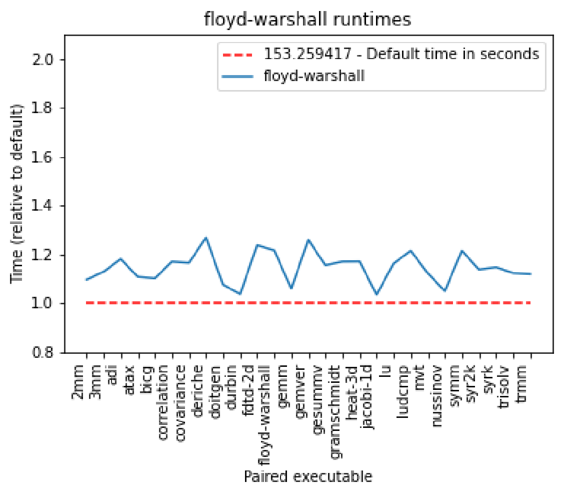
doitgen



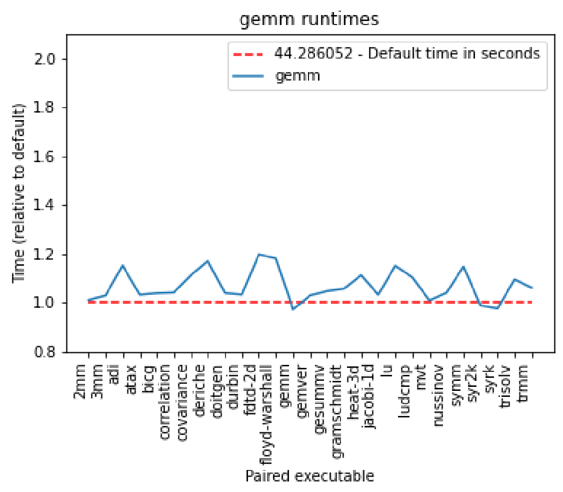
durbin



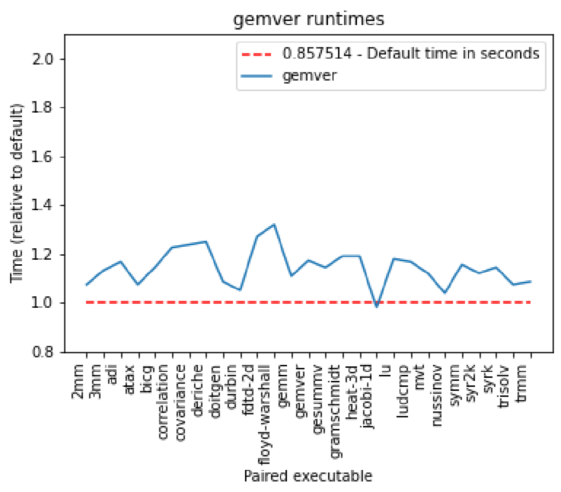
fdtd-2d



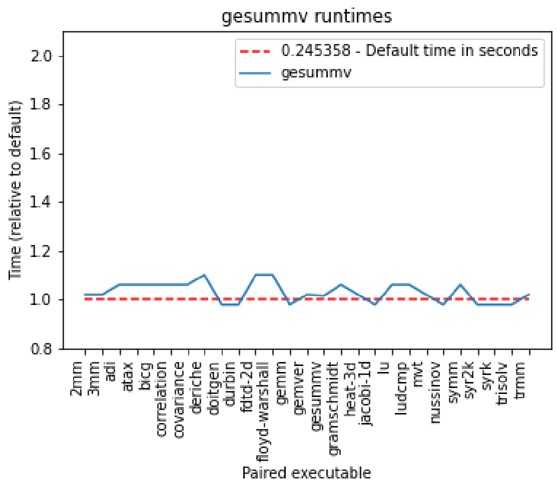
floyd-warshall



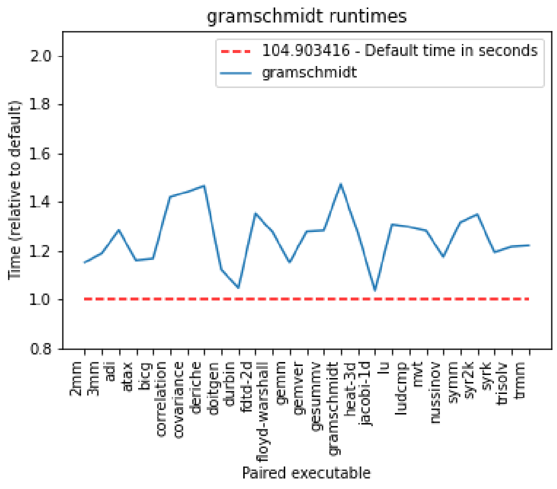
gemm



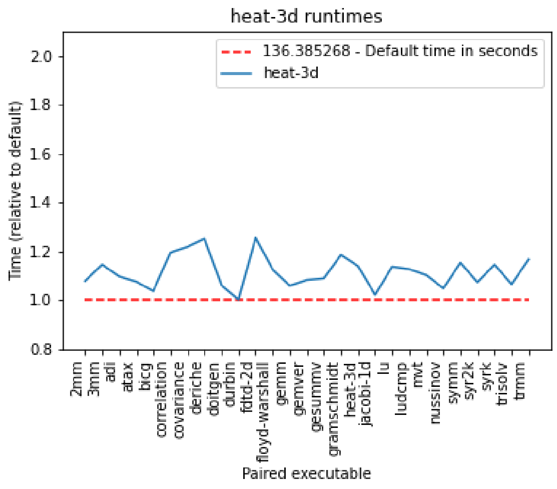
gemver



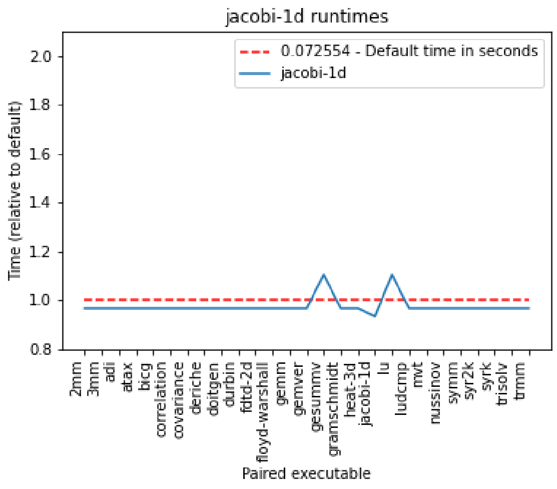
gesummv



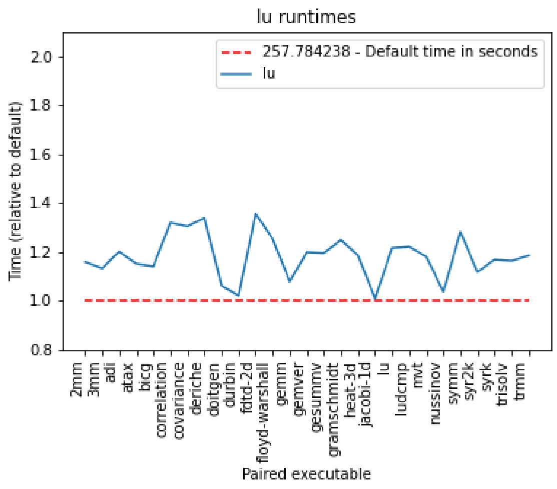
gramschmidt



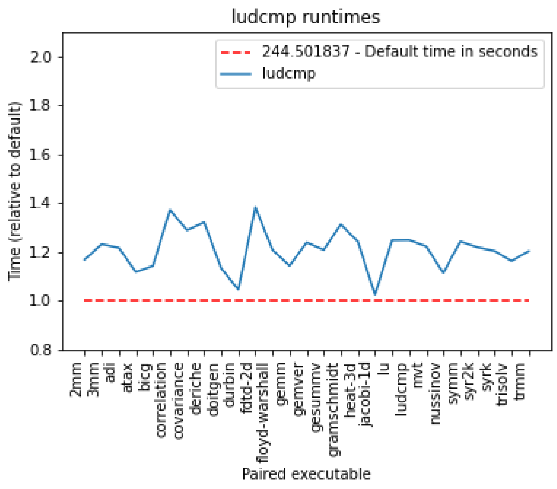
heat-3d



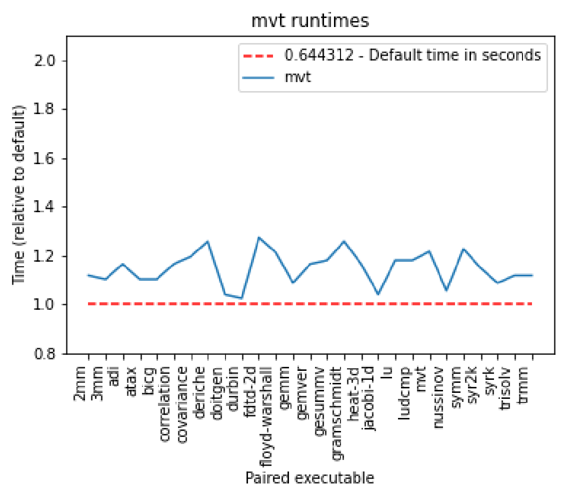
jacobi-1d



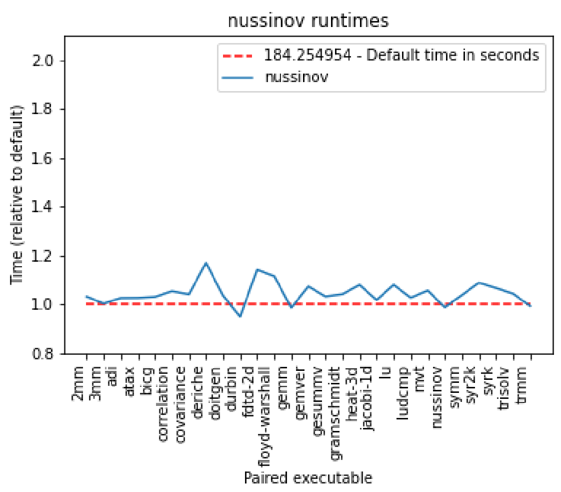
lu



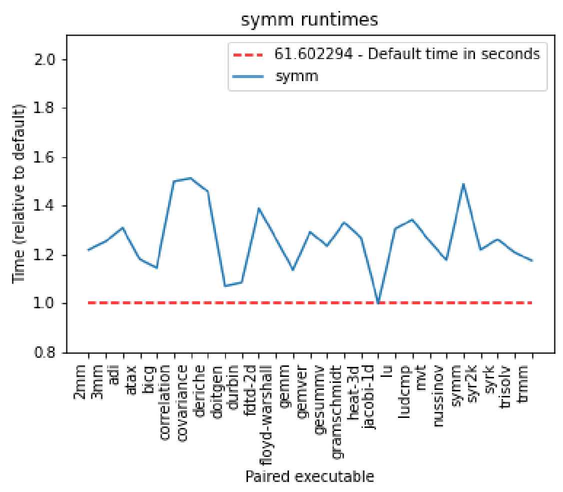
ludcmp



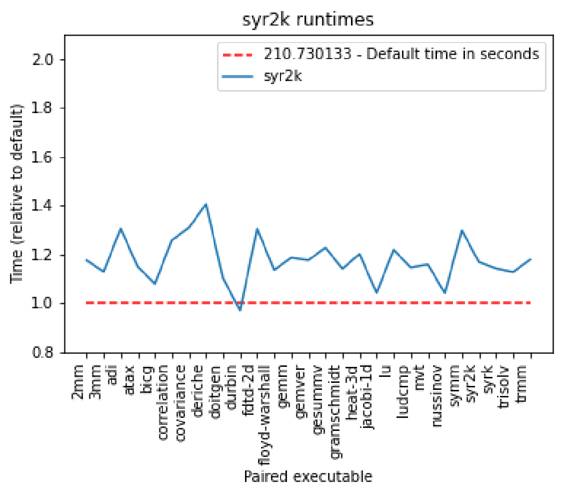
mvt



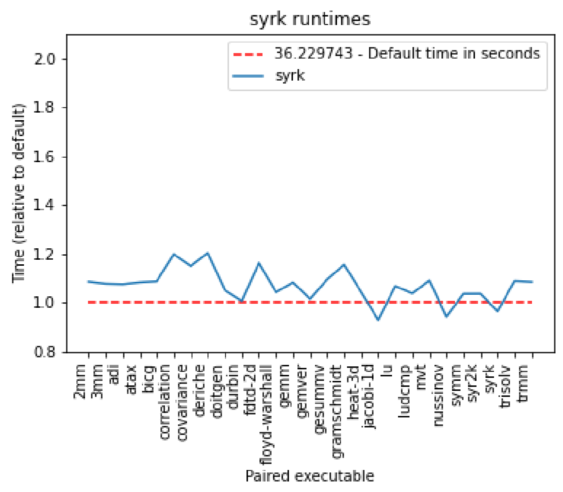
nussinov



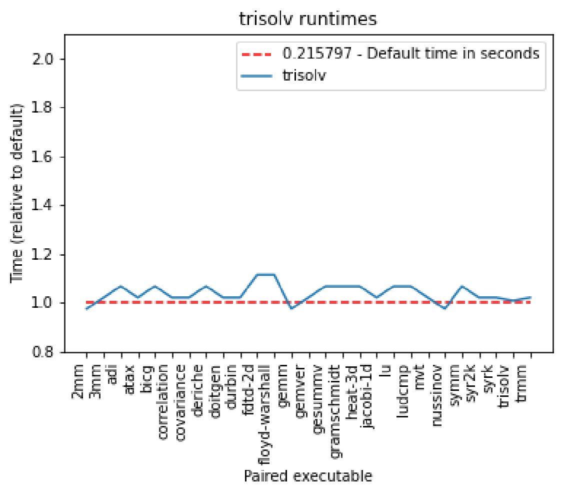
symm



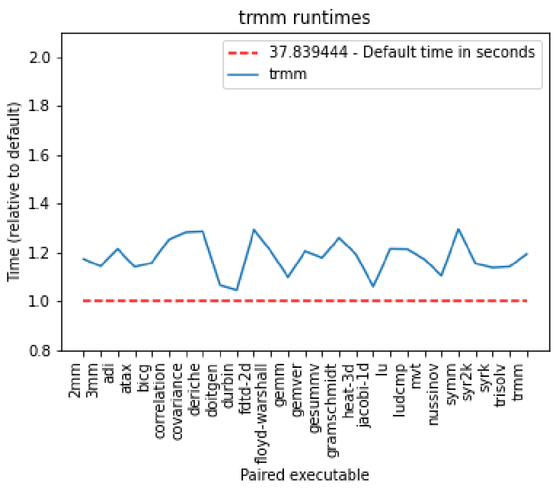
syr2k



syrk



trisolv



trmm

1. 10 Processes having lowest default runtime:
   (a) jacobi-1d: 0.072554 seconds
   (b) durbin: 0.08127 seconds
   (c) trisolv: 0.215797 seconds
   (d) gesummv: 0.245358 seconds
   (e) bicg: 0.448567 seconds
   (f) atax: 0.483144 seconds
   (g) mvt: 0.644312 seconds
   (h) gemver: 0.857514 seconds
   (i) deriche: 5.082191 seconds
   (j) doitgen: 16.465486 seconds
2. 10 Processes having highest default runtime:
   (a) lu: 257.784238 seconds
   (b) ludcmp: 244.501837 seconds
   (c) adi: 201.959441 seconds
   (d) nussinov: 184.254954 seconds
   (e) floyd-warshall: 153.259417 seconds
   (f) 3mm: 136.873787 seconds
   (g) heat-3d: 136.385268 seconds
   (h) gramschmidt: 104.903416 seconds
   (i) fdtd-2d: 84.221851 seconds
   (j) 2mm: 83.2836 seconds

3. During the analysis of the runtime plots, it was observed that for some processes, such as jacobi-1d, durbin, trisolv, gesummv, doitgen, nussinov, syrk, and trisolv, the time taken was sometimes below the default time. One possible explanation for this is that when running a process in the terminal, the time taken is reported as real, user, and sys. However, when running the same process multiple times, the effect of sys time gets averaged out, resulting in a smaller time than the default case. This was particularly evident for the aforementioned processes, which had the lowest default runtime and were highly affected by the sys time during a single run.

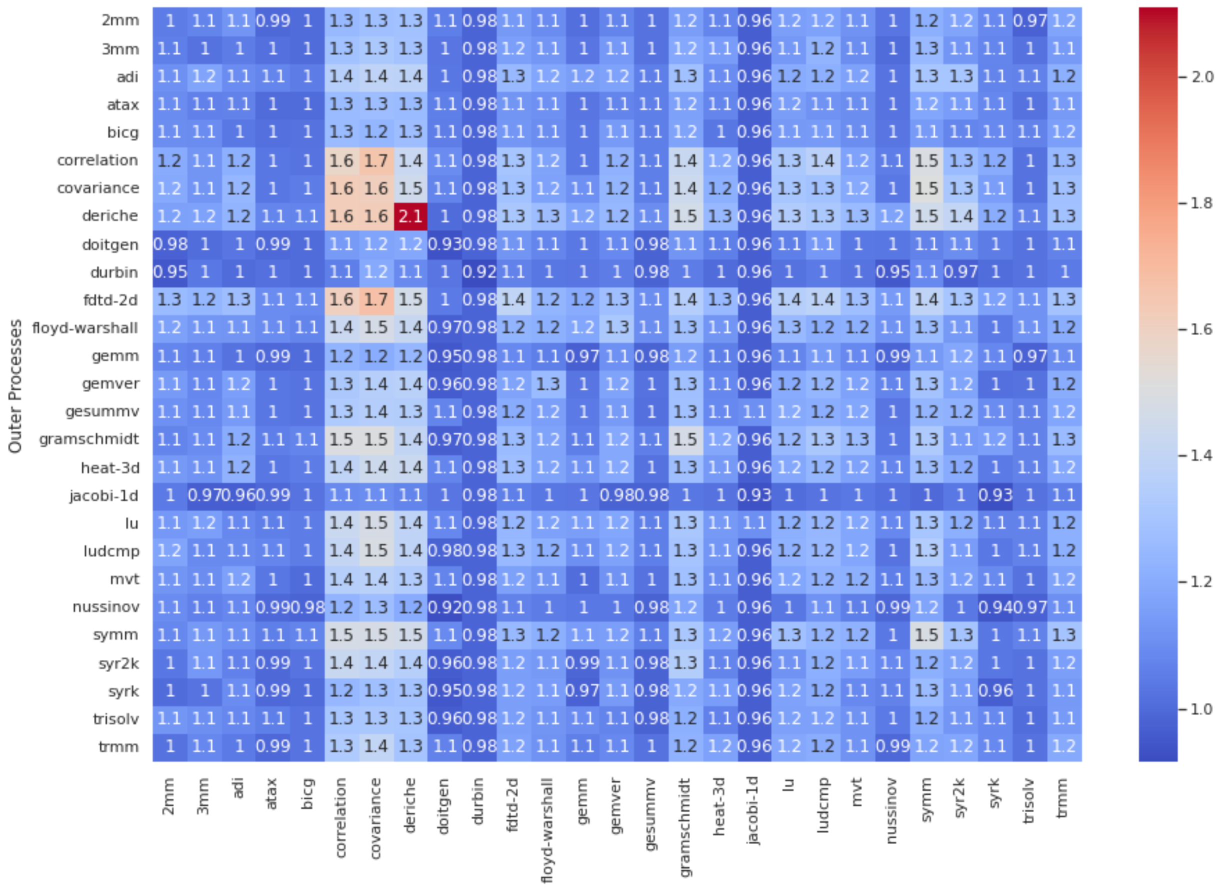


Figure 18: Heatmap of normalized runtimes (Time relative to default runtimes)

A few observations from this heatmap are:

1. The processes, namely correlation, covariance, deriche, fdtd-2d, gramschmidt, and symm, are found to be more sensitive to the presence of other processes compared to the remaining processes in the experiment. This is evident from the plots showing the relative time taken by each process when run with a specific number of processes, where these processes showed a significant increase in runtime compared to their default runtime.

2. The processes, namely doitgen, durbin, gemm, jacobi-1d, gramschmidt, and symm, are found to be least sensitive to the presence of other processes compared to the remaining processes in the experiment.

3.4 BNN notebooks

1. BNN-PYNQ repository provides implementations of neural networks for binary classification tasks using both hardware and software approaches.
2. The hardware implementation uses FPGA hardware acceleration to speed up the inference process, while the software implementation runs on the ARM Cortex-A9 processor on the PYNQ-Z2 board.
3. The provided notebooks showcase how to launch the BNN implementations in both hardware and software modes. The hardware section of each notebook demonstrates how to use the FPGA to accelerate the neural network's inference process, while the software section shows how to run the same neural network on the board's CPU.

3.5 BNN runtime

We attempted to run multiple instances of selected processes from Polybench in parallel using Jupyter Notebooks, and we observed the following results:

1. We observed that the classification rate and inference rate reported in the 'Launching BNN in Hardware' section remained constant across all possible permutations, while the rates decreased in the 'Launching BNN in Software' section when running multiple processes in parallel.
2. The reduction in rates appeared to be primarily dependent on the sharing of CPU cores among multiple processes rather than memory usage. As it was observed that the reduction in rates occurred uniformly across all types of processes and was only dependent on the number of processes rather than the type of processes.

ACKNOWLEDGMENT

I would like to express my sincere gratitude to the Department of Computer Science and Engineering, Indian Institute of Technology Hyderabad, for providing me with the opportunity to undertake this project.

I would also like to extend my heartfelt thanks to Professor Rajesh Kedia for his invaluable guidance and support throughout this project.

I would like to acknowledge the Xilinx BNN-PYNQ and Polybench repositories for providing the necessary code and benchmarks for my project. I am also grateful for the use of the Xilinx PYNQ Z2 board provided by the Dept. of CSE, IITH, which allowed me to run and test the code effectively.

REFERENCES

1. PYNQ Documentation by Xilinx
2. Polybench: The polyhedral benchmark suite by Meinersbur
3. BNN-PYNQ: Binary Neural Network on PYNQ