Studying the Impact of CPU and Memory Controller Frequencies on Power Consumption of the Jetson TX1

Hazem A. Abdelhafez, Matei Ripeanu
{hazem,matei}@ece.ubc.ca

***Abstract*—Nowadays, heterogeneous unified memory architecture platforms are becoming increasingly common. These platforms incorporate several co-processors on a single chip with a shared physical memory. The use cases for such platforms can vary dramatically. On the one hand, they can be used in the context of Edge computing, which cannot tolerate high latency and has strict energy/power constraints. On the other hand, motivated by their growing computing capabilities, and their energy-efficiency, many have considered replacing traditional bulky servers with these platforms to deliver the same computing power but with lower energy budget. This study is an exploratory step to understand the trade-off between power consumption, message latency, and throughput on a low-power heterogeneous platform for data stream processing workloads by characterizing several common computing kernels found in computer vision algorithms. Our preliminary experiments on NVIDIA Jetson TX1 show that it is possible reduce power consumption by up to 12%.**

I. Introduction

Recently, heterogeneous computing has become the leading approach to raise the performance envelope and reduce the energy consumption of computing systems. Energy consumption not only poses a huge threat to our environment, but also contributes significantly to the cost of any computing-based service. For example, in cloud computing environments, power draw represents $\sim$15% of the Total Cost of Ownership (TCO) [1], while cooling and other hardware equipment accounts for the rest. Moreover, for each 1 W of power drawn by the computing devices, 0.5 to 1 W is drawn by the cooling system [2], which indirectly increases the contribution of the computing equipment in the TCO. In 2016, a study showed that data-centers in the USA consume approximately 70,000GWh which represents 1.8% of the total electric power produced in the USA and is projected to increase by 4% by 2020 [3].

Additionally, the growing number of IoT devices and their various use cases represent a challenge to existing cloud computing paradigm. On the one hand, resource consolidation and flexible scaling features have made cloud computing the backbone of the majority of large-scale computing environments, but, on the other hand, it is not well-suited for latency-sensitive applications. Moreover, the large number of connected devices led to higher bandwidth cost and congestion, with additional privacy and security concerns from the users' perspective. These shortcomings led to the emergence of a new computing paradigm known as Fog or Edge computing [4]. In this approach, instead of sending data to the cloud to process it and return the results to the user, IoT devices directly run computation algorithms on the Edge device or on a geographically co-located cluster-like resources (sometimes dubbed cloudlets or elements of Fog computing). In all these scenarios energy efficiency is a major concern.

Solutions at various levels of the computing stack have been proposed to address the energy-efficiency of the computing infrastructure. For instance, the adoption of dedicated accelerators or highly-parallel architectures has increased drastically especially heterogeneous platforms that incorporate several co-processors. GPUs' massively-parallel architecture allows them to be more energy efficient for throughput oriented problems. Field Programmable Gate Arrays (FPGAs) and Application-specific Integrated Circuits (ASICs) have also witnessed wide adoption (e.g. Microsoft Catapult [5], Google Tensor Processing Unit (TPU) [6]).

In data-center environments, resource consolidation and scheduling algorithms that aim at reducing the consumed power has also been proposed [7], [8], [9], others developed algorithms that predict the future workload and turn-off or on the servers to reduce power consumed during low load cycles [10]. Moreover, there are efforts that focus on the energy complexity of the computational algorithms [11] with the goal of understanding how the algorithms behaviour affect energy consumption and subsequently making them more energy efficient.

Recently, heterogeneous unified-memory platforms that incorporate low-power CPUs for running the operating system and low-load tasks and GPUs for high-load tasks have emerged. An example of such platforms is the NVIDIA Jetson embedded boards family. These platforms target Edge computing scenarios (e.g. self-driving cars, drones, robots, etc.). However, the compute capabilities offered by their heterogeneous resources made them potential candidates to replace bulky servers in data-center environments as well, mainly because of their much lower idle base power compared to traditional servers.

Objective of this paper: Stream data processing model is common for a wide range of applications (e.g. multimedia stream processing, dynamic graph algorithms, inference task in interactive AI applications, and IoT applications.) This study aims to answer two questions:

- In the context of streaming applications, can we save power by tuning the CPU and memory controller to low

This work is supported in part by the Institute for Computing, Information and Cognitive Systems (ICICS) at UBC.

frequencies and at the same time maintain the system's stability?
- Are the default power management capabilities on Jetson TX1 able to deliver best performance-per-watt? if not, then how much more energy can we save?

Contributions: We show that on a low-power heterogeneous platform such as the Jetson TX1, we can tune the system to save from 2% to 12% of the consumed power by operating on lower frequencies of the CPU and the memory controller while increasing the average response time to a certain limit that still fulfills the QoS metrics (message processing rate) of a stream-based application.

This paper is organized as follows: Section II presents background information about data stream systems and energy-efficiency in a computing environment along with an illustration of the main features of the Jetson platform. Section III, details our methodology and describe the system components for our experiments and how they interact together. Section IV, presents and discusses the results obtained and main lessons learned from the experiments. Section V, presents previous work done in the context of energy-efficiency of data stream processing. Finally, Section VI, summarizes the key findings and limitations.

II. Background

A. Data Stream Systems

In the context of this paper, a data stream system is a system in which data is generated by a single or multiple producers as small batches of data (in the order of bytes to kilo-bytes) and consumed by a single or multiple consumers. A producer can be: physical sensors, such as weather, medical, surveillance sensors, or IoT devices, such as smart appliances, traffic cameras, wearables, etc. [12], [13].

A consumer in such a system can be any processing device that incorporates one or more processors that can read, interpret and transform the incoming data batches from the producer to output a specific final or intermediate result that can be further processed.

A consumer in a data stream processing system typically applies a compute kernel on each batch of data or on a time window of several batches. Batches of data in a stream system can also be referred to as messages, records or elements. The compute kernel can be the same for all incoming messages or different pipe-lined kernels that execute different tasks on the messages. In some cases, it may be required that a data stream system maintain a state the depends on the previous messages computations, in this case the state data should ideally be persisted in memory for fast access or on disk if memory is full.

To evaluate the performance of a data stream processing system usually message processing time (the time spent by a user defined compute kernel analyzing the message) and maximum attainable throughput (the maximum number of messages processed per second) are the most important metrics. For instance, in a video streaming application, it is essential that a consumer system sustains a minimum processing rate expressed as $Frames/Sec$ or else some frames are dropped resulting in a degraded user experience.

Generally, data stream processing systems can be deployed on several computing nodes where the nodes are connected together via a network interface or deployed on edge devices closer to the data producers to save network bandwidth.

B. Energy Efficiency in Data Stream Context

Power is the amount of energy, expressed in Joules (J), consumed per time unit (one second) and is measured in Watts (W). Power consumption in a computing system can be calculated as the sum of dynamic power and static power.

For a processing unit, static power is the power consumed by its hardware circuitry in the idle state, and it represents the baseline for the total power consumption. Dynamic power is the power consumed by applications utilizing the processing unit and it depends on the application behaviour and the available hardware resources. It is also referred to as switching power and it results from:

- Capacitor charging and discharging in the electronic circuits due to data flow changing the states of the transistors between ON/OFF states.
- Leakage current due to short circuit during the transition phases from 0 to 1 and from 1 to 0.

Dynamic power can be calculated as follows:

$$P = \alpha C V^2 f \tag{1}$$

where $\alpha \epsilon [0, 1]$ is the activity factor and it represents the average number of times the CMOS transistors switch between the 1 and 0 states per clock cycle, in other words, it is application workload dependent. C is the capacitance and it is the physical property of the circuit components, V is the supply voltage, f is the clock frequency that drives the transistors. The supply voltage is proportional to the operating frequency, this is to guarantee stability of the system and its ability to maintain the sought operating frequency [14].

Most operating systems include a functionality to control the operating voltage and frequency of the various devices. This functionality is referred to as Dynamic Voltage Frequency Scaling (DVFS) [15]. DVFS is a hybrid between Dynamic Voltage Scaling (DVS) and Dynamic Frequency Scaling (DFS). In DVFS, the operating system kernel scales the processor's operating frequency up or down based on the processor utilization level.

In the context of a data stream processing system, the number of messages processed per Watt is often used to compare the energy efficiency of different stream processing systems.

C. The Jetson Platform

In 2014, NVIDIA introduced the TK1 board as the first low-power embedded board of the Jetson platform. Followed by the TX1 in 2016, TX2 in 2017 and AGX Xavier in 2018. The four boards share two interesting features that made the Jetson platform suitable for our study. First, each board

TABLE I: NVIDIA Jetson TX1 board characteristics.

CPU	64-bit ARM Cortex-A57
Architecture	ARMv8-a
L1/L2 cache Sizes	32KB/2MB
Core Frequency Range	102 MHz to 1.734 GHz
Peak Theoretical FLOPs	55.48 GFLOPs
DRAM	4GB LPDDR4 RAM, 2 Channels
Data Bus Width	64 bit
Controller Frequency Range	40.8 MHz to 1600 MHz
Peak Theoretical Bandwidth	25.6 GB/Sec

has an integrated ARM processor, a powerful NVIDIA GPU compared to the on-board ARM CPU, and a shared global memory. Second, each of those integrated components have a wide range of operating frequencies with more than 10 values to select from for the CPU, GPU and the memory controller, besides, it is possible to activate and deactivate CPU cores. These two features allow us to study the performance and the power consumption trade-offs on these boards thoroughly compared to the traditional computing devices that have a narrow range of operating points for its CPUs only.

In this study we use the TX1 board as host for a data stream processing application but we exclude the GPU. Table I lists the main features of the TX1 board CPU and memory.

III. Methodology

Our goal is to quantify the power/performance trade-off when tuning the CPU and memory controller frequency the Jetson TX1 board in a stream data processing application.

A. Experimental Setup

Our system comprises two compute nodes, a power logging kit and a network router for connectivity. One of the nodes acts as a producer that generates messages according to specific criteria as we will discuss later in this section. It is also responsible of reading the measured power values from the power logging kit.

The second node is a Jetson TX1 board that is responsible of processing the messages sent by the producer. It applies a specified compute kernel on each message individually. We only consider a stateless multi-threaded processing model. In this model a thread processes each message independently from previous messages and from other threads. While this might not be the case for all data stream processing applications, it helps simplify our study and focus more on the trade-off between performance and power consumption instead of managing data inter-dependency or maintaining state in each thread.

B. Load Generation

The producer generates messages as double-precision byte arrays of different sizes for each experiment iteration. It sends those messages over a UDP socket to the consumer. We use ZMQ [16], which is a distributed messaging library used in several big-data systems to facilitate communication among the distributed system components. It supports broker/broker-less based messaging paradigm. In our experiments we used

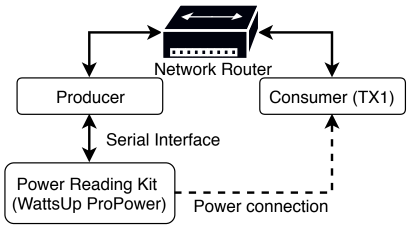


Fig. 1: Overview of system components and connections

the ZMQ library for a broker-less based communication style between the two nodes.

The message generation in the producer is a Poisson process where the inter-arrival time between the messages is drawn from an exponential distribution. We generate loads with different throughput values that represent the rate parameter in the Poisson process.

The message sizes we selected are 128 and 1024 elements, each element is an 8 bytes double value. We chose those values to represent fit-in-cache and non-fit-in-cache cases. For the offered load, we chose 240 and 960 messages per second. These values represent low and high loads above which many hardware settings cause the consumer to drop messages and below which the base power consumption prevails against dynamic power.

C. Connectivity

The producer and the consumer are connected through a local network device with a link bandwidth of 1 Gbps. The power logging kit is connected to the producer node through a serial interface to read power measurement data. The high-level system architecture is illustrated in Fig. 1.

D. Workload Kernels

The consumer is a multi-threaded application that applies a set of linear algebra routines (using the OpenBLAS library and optimized for the TX1 armv8-a processor architecture) on the incoming messages. Given an input vector X and an input/output vector Y each routine works as follows:

SAXPY: scales vector X by a scalar factor A and add it to vector Y where $Y = AX + Y$.

DOT Product: dot product of vector X by vector Y.

Euclidean Distance: calculates the Euclidean distance between two vectors X and Y such that $Y = \sqrt{\sum_{i=0}^{N-1} Y_i^2}$.

Matrix Multiplication: applies the DGEMM routine which, given input matrices A and B, outputs matrix C such that $C = \alpha AB + \beta C$, where α and β are scalar values.

We choose those BLAS routines as representative workload kernels because they are widely used in several application domains such as computer vision applications, video analysis, data stream clustering/classification algorithms, dynamic graph analysis and machine learning.

E. Measurements Infrastructure for Power and Time

1) Power Measurement: The earlier versions of the Jetson TX1 don't include internal power sensors, hence, we use an external power measurement tool. We used a commercial

WattsUp PowerPro kit [17] to measure the power consumed by the TX1 board during the experiments. The WattsUp kit has a serial interface that we used to read the power measurements with a sampling frequency of one sample/second. It has an accuracy of +/- 3% above 10 watts and +/- 5% below 10 watts. We separate the power logging from the TX1 to avoid any interference with the experiments thus, we connect the WattsUp kit to the TK1 that runs the producer too. Although there are more accurate solutions to measure the power consumption, such as using the INA219 integrated circuit [18], these solutions require building a custom circuit to interface with this IC family. This approach is more accurate but time consuming.

2) Timing Measurement and Synchronization: One of the main challenges in the measurement setup is how to synchronize the power readings, triggering the producer to start sending messages and the consumer statistics. Our goal is to make sure that the read power corresponds to the interval during which the TX1 is doing actual work on the incoming messages from the producer, and to avoid interference from auxiliary computations (allocating memory for timing measurements, launching threads for synchronization between the producer and the consumer, initializing ZMQ socket and receiver buffer) before and after (calculating statistics at the end of each iteration, freeing allocated memory, destroying ZMQ context and killing all active threads) the experiments as much as possible.

To avoid system times discrepancies, we use monotonic clock which is independent of the system's wall time and is not affected when changing the CPU's operating frequency. It also has a resolution in the nanoseconds range. To quantify the overhead of querying the clock, we ran 100K queries and we found that the average time for each query is $\sim 72ns$. This small overhead guarantees that our measurement resolution is in the sub-microsecond range.

To synchronize the statistics collected by the consumer with the power readings, we output the timestamp with mu accuracy for each power reading from the WattsUp kit, and the timestamps at which the producer started/finished sending messages to the consumer. When the consumer executable runs, it prepares the data containers needed for statistics collection, creates the requested number of threads, initializes all relevant data to the experiment and finally sends a *start_sending* request to the producer that triggers sending messages from the producer side. At this moment, the producer saves the timestamp as START_TIME and keeps running until the consumer issues *stop_sending* request at STOP_TIME.

Moreover, the producer host and the TX1 board are connected over a local network. For 1K packets of size 64Bytes, the percentage of packet loss is 0%, the min/average/max/standard deviation of the Round-Trip Time (RTT) are 1.27/1.77/2.29/0.19 ms respectively. Therefore, statistically, the time it takes for the consumer to notify the producer to start issuing messages is less than 2.5ms. This means that the timestamps recorded by the producer represent the actual experiment start time with an accuracy of +/- 1.77ms. To make sure that the power readings represent the power consumed by the TX1 during the actual processing of the messages, we truncate any power reading with time-stamp before START_TIME and after STOP_TIME.

F. Performance and Power Consumption Metrics

The main Key Performance Indicators we are interested in are: maximum sustainable consumer throughput (messages/second) after which the consumer starts to drop messages, average message processing time (ms) and power consumption in (Watts). In our system, we don't allow message drop by the consumer due to overload and hence we only report the results of the iterations in which the maximum sustainable consumer throughput is at least equal to the offered throughput.

We calculate the consumer throughput for a total number of processed messages n as follows:

$$R = \frac{n}{P_n - T_1} \quad msgs/second$$

where P_n is the time at which message n was processed and T_1 is the time at which the first message was received. We calculate the average message processing time as:

$$T_p = \frac{\sum_{i=1}^{n}(P_i - T_i)}{n} \quad seconds$$

where T_i is the time at which message i was received and P_i is the time at which message i was processed. We need T_p in order to quantify the impact of the different hardware settings on the raw performance.

G. Additional Experimental Details

Since our goal is to evaluate the impact of different frequency settings and number of active cores for the ARM CPU and the frequency settings for the Memory Controller on the energy-efficiency achievable by the system, we set the GPU to the lowest frequency, disable HDMI ports and shutdown any GUI-based service before we run our experiments. To control the operating settings of the Jetson TX1, we edit the kernel configuration through the exposed virtual file-system directory $sysfs$ to specify the operating frequencies and active CPU cores before each experiment run.

IV. Experimental Results and Discussion

In this section we answer the two motivating questions for this study: 1) Can we save power by lowering the operating settings of the hardware and how much savings can we achieve?, 2) Which power control policy achieves the lowest power consumption while maintaining system stability?. To answer these questions we introduce four main operational modes:

Brute-Force Search: In this mode, we run the producer and then launch the consumer in a loop where in each iteration we change the number of active CPU cores, the cores frequency and the memory controller frequency. The goal is to find the best settings in terms of power consumption.

DVFS: In this mode, instead of manually selecting the operating settings for the TX1, we let the Linux Dynamic Voltage Frequency Scaling (DVFS) system, which is set to the Interactive policy by default, control the CPU and memory controller frequencies dynamically during runtime. On the TX1, the rail voltage that supplies the processor's core circuitry depends on the operating frequency. However, the mapping from frequency to voltage is fixed, therefore in our experiments we can only set the operating frequency of the processor but not the rail voltage. Additionally, the TX1 provide a software agent (ACTMON) [19] which monitors the hardware activity of all board devices including the processors and the memory. This central agent provides statistics about the utilization level of the hardware components to the DVFS agent. The DVFS then scales the CPU and GPU frequencies and rail voltage according to the load level, and it scales the memory controller frequency based on the memory access cycles that represent the amount of traffic to and from the memory.

Maximum Settings: In this mode we set the board to the maximum operating settings where all the CPU cores are active, the CPU and the memory controller frequencies are set to the maximum values for the best performance. However, we keep the GPU at the lowest frequency possible.

Race-to-Finish: This mode requires that once a message is received, we set the board to the maximum settings until the message processing is done. Then, we set the board to the DVFS Interactive mode to save consumed power until next message is received and repeat the whole cycle again.

In our experiments we have variable workload characteristics expressed in terms of message size and offered throughput. We also have several operational settings including the number of active CPU cores, CPU frequency and memory controller frequency.

A. Experiment-1

We start by evaluating the power consumption of the first three operational modes. In Fig. 2, we show the average power consumed on the Y-axis and the four BLAS routines sorted on the X-axis. Each sub-figure represents a specific workload in terms of producer throughput and message size in bytes, and the number of threads. In tables II and III, we show the corresponding average message processing times in μs for each operational mode. We also report results of 3 threads only which is the maximum number of threads we use in our experiments since the ARM CPU has only 4 cores. We leave one core for OS kernel and ZMQ receiver thread. We don't report the other thread count results due to space limitation.

Finding-1: There is still room for saving power by extending the message processing time while still maintaining application stability (where the consumer throughput is at least equal to the producer throughput). Comparing the Brute-Force Search mode against the DVFS mode we found operational settings that reduce the consumed power from 2% and up to 12%. By comparing the Brute-Force Search approach against the Maximum Settings mode, we found operational settings that can save from 15% to 25% of power consumption.

Our explanation for why we found settings that save power compared to the DVFS is that DVFS is known to be sensitive to hardware resources utilization levels [20]. This drives its selection of high operating frequencies for the CPU and the memory controller to fulfill the compute needs of the workloads. However, the DVFS software agent doesn't take into consideration that message processing time could be more relaxed and thus it is possible to operate at lower frequencies to save power. In the Brute-Force Search mode, higher energy-efficiency was achieved by extending the message processing time by up to 3 times as shown in Tables II and III.

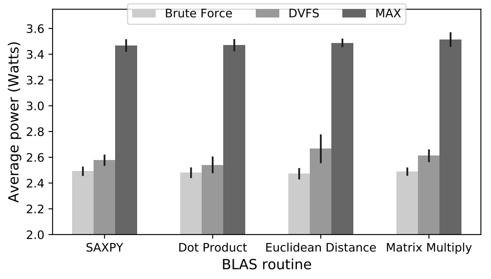
Compared to the Maximum settings approach, the Brute-Force Search mode showed that by operating at lower frequencies we can definitely save power but comparing the average message processing delay we see that the Maximum Settings mode achieves up to 18 times better than Brute-Force Search and 6 times better than DVFS.

The main takeaway from this finding is that we need to consider the data stream application Quality of Service (QoS) metrics while tuning the hardware settings to save consumed power. While the Brute-Force Search mode in this context is meant to prove this point, we believe it is impractical for real application deployments, hence, we plan to work on building a scheduler that dynamically selects the best hardware settings during run-time to save consumed power while maintaining the required workload performance constraints.

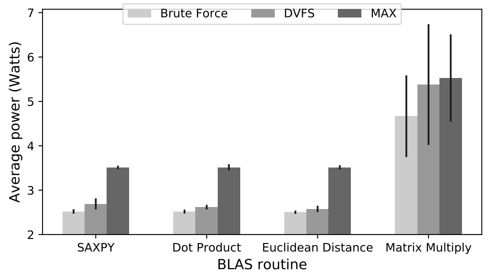
B. Experiment-2

To compare the Race-to-Finish (RTF) approach to the other operational modes we needed a single point of control to set the hardware settings to the designated values before and after each message processing hence we set the number of worker threads to one. The idea is to have one worker thread that reads messages from the receiver buffer, then sets the OpenBLAS internal number of threads to the value we actually want to test. This worker sets the TX1 hardware settings to the maximum before processing each message and once the BLAS routine is done, it sets the hardware back to the DVFS mode instead of the minimum settings. This is done to make sure that the rest of the tasks such as fetching messages from the reader queue, networking functions and other OS kernel processes are not deprived the needed resources to complete their tasks in a reasonable time. In Fig. 3, we show the average consumed power of the four compute modes including the RTF mode.

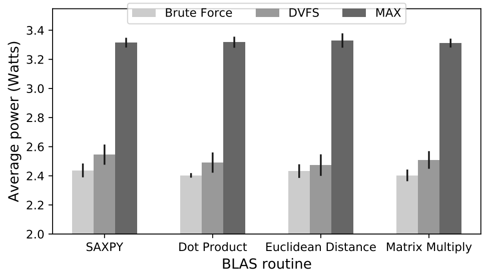
Finding-2: As illustrated in Fig. 3, the RTF results are significantly worse than both the DVFS and the Brute-Force Search results. For instance, the Brute-Force Search mode was able to tune the system to consume less power compared to RTF by 27% to 30%. This finding aligns with a previous study on multimedia application [21]. We also found that the overhead of tuning the CPU and memory controller frequencies before and after each message is significant. This is due to the way we set the frequencies through writing to exposed file system interface. We quantified the overhead for this tuning and on the one hand, we found that the overhead of setting the platform operations settings to the maximum



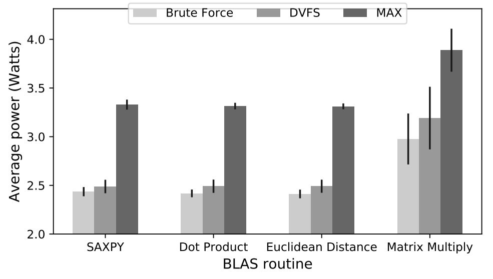
(a) Offered load of 960 msgs/sec, message size is 1KB, 3 threads



(b) Offered load of 960 msgs/sec, message size is 8KB, 3 threads



(c) Offered load of 240 msgs/sec, message size is 1KB, 1 thread



(d) Offered load of 240 msgs/sec, message size is 8KB, 1 thread

Fig. 2: Average power consumption for various workloads

TABLE II: Average message processing time in μs for R = 240 Msgs/sec and 1 Thread.

BLAS Routines	Msg size = 1024 Bytes			Msg size = 8192 Bytes		
	DVFS	Maximum Settings	Brute-force Search	DVFS	Maximum Settings	Brute-force Search
SAXPY	9.50	0.57	4.18	23.61	1.40	4.62
DOT	7.45	0.54	4.26	25.96	1.61	34.02
Euclidean Distance	16.81	0.98	7.91	60.87	3.70	36.50
Matrix Multiply	353.82	21.07	490.64	2350.86	1622.83	3961.44

TABLE III: Average message processing time in μs for R = 960 Msgs/sec and 3 Threads.

BLAS Routines	Msg size = 1024 Bytes			Msg size = 8192 Bytes		
	DVFS	Maximum Settings	Brute-force Search	DVFS	Maximum Settings	Brute-force Search
SAXPY	4.55	0.49	4.14	8.75	1.31	12.14
DOT	3.58	0.44	4.44	10.29	1.53	14.02
Euclidean Distance	6.43	0.89	13.67	24.22	3.62	32.02
Matrix Multiply	176.76	21.93	477.82	2026.42	1973.94	3168.23

is around $\sim 0.16ms$. The cost for setting the hardware to maximum and then to minimum operational settings before and after the kernel respectively is $\sim 9ms$. the cost for setting the hardware to maximum then to the default governor but keeping the memory controller at minimum is $\sim 3ms$. On the other hand, the cost for setting the hardware settings to maximum then DVFS but keeping the memory controller at max is $\sim 0.3ms$. There is a lower-level interface for controlling the frequencies of the CPU and the memory controller by directly manipulating hardware registers. In the future, we plan to carry on more thorough analysis for the RTF approach using this lower-level interface to fairly evaluate this approach compared to the others.

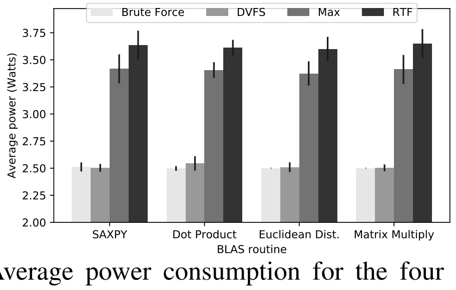


Fig. 3: Average power consumption for the four computing modes

C. Experiment-3

To understand the impact of the memory frequency on the power consumption, we plot the average consumed power for

TABLE IV: Impact of memory controller on power and performance. CPU frequency is set to max. We compare the power and message processing time for each kernel for two cases: lowest and highest memory controller frequencies. R is the incoming message rate, and S is the message size.

BLAS Routines		R=240 Msgs/sec		R=960 Msgs/sec	
		S=1KB	S=8KB	S=1KB	S=8KB
SAXPY	Power increase %	28.53	28.05	23.77	23.13
	Time reduction %	9.24	38.56	11.57	28.66
Euclidean Distance	Power increase %	27.97	26.86	24.36	22.29
	Time reduction %	13.26	12.29	3.88	16.68

TABLE V: Ratio of base (Idle) power contribution in the overall power consumption for low workload of message rate ($R = 30Msgs/sec$) and message size of ($1K$).

	SAXPY	DOT	Euclidean Distance	Matrix Multiply
DVFS Power	2.45	2.43	2.44	2.46
Base Power %	96%	97%	96%	95%

our workloads against several memory controller frequencies in Fig. 4.

Finding-3: Figure 4 shows the memory controller frequency impact on power consumption. Table IV shows that increasing the memory controller frequency from 40.8MHz to 1.6GHz, increases the power consumption, but the message processing time reduction is not proportional to the power increase percentage except for two cases in the SAXPY kernel with message size of 8K.

D. Experiment-4

To quantify the energy savings under low offered load, we conduct an experiment for a message rate of $R = 30Msgs/sec$ and message size of $1K$.

Finding-4: We found that because of the low utilization factor in the dynamic power consumption in Equation (1), the static power is significant. In table V we show the total consumed power in the DVFS mode with one worker thread, and the percentage of the base (idle) power to this total power. We measure the base power of the board when the board is idle, however, for fair comparison we turn off the HDMI port and turn on the fan to have the same settings for all experiments. We found the average base power to be $2.36W$. This means that at low workload cycles the room for saving power consumption is limited since we already operate close to the base power limits which depend on the physical characteristics of the hardware.

V. Related Work

There are several studies on energy efficiency and performance trade-offs of data stream applications. Many of these studies either focus on a narrow data stream application category [21], [22] or focus on evaluating the stream processing frameworks themselves without any evaluation of the underlying hardware settings impact [23].

Dayarathna et al. [23] analyze several stream processing frameworks such as Apache Storm, and Apache Spark. Their study indicates that data communication between different nodes in a cluster contributes significantly to the power consumption of the stream processing framework and that building energy-efficient stream processing applications shouldn't be only driven by CPU usage levels but also the communication patters between nodes. However, the study did not investigate how tuning the hardware settings affects the energy consumption of these frameworks.

Stokke et. al. [21] study the factors that contribute to the power consumption on the Tegra TK1 board, the use case in this study is multimedia applications. A key finding in this study is that Race-to-finish is not the best approach to save power consumption under specific performance constraints.

De Matteis and Mencagli [22] present a latency-aware method to control the CPU frequency and the number of active CPU cores to achieve the best energy-efficiency in an elastic data stream processing application.

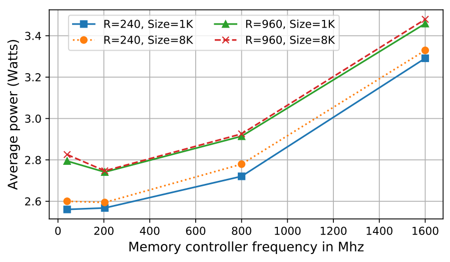
There are many previous research efforts that study various workload-aware DVFS-based strategies to reduce the energy consumption of computing resources [24] [25]; We believe that the Jetson platform brings unique features (e.g. wide range of operating points) that require further exploration to understand its energy-performance trade-offs. Our work in this context is an exploration of the energy-efficiency of the Jetson TX1.

VI. Summary and Limitations

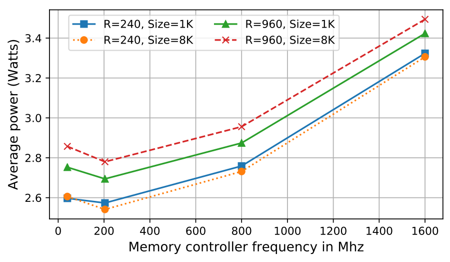
Summary. We study the trade-off between performance and power consumption in the context of a data stream application on the Jetson TX1. Our findings show that there is still room for improvement in power consumption by up to 12% by tuning the platform operational settings (number of active cores, CPU frequency and memory controller frequency) at the cost of increasing the average message processing time. However, it is possible to maintain application stability where the consumer is able to maintain an adequate processing throughput to avoid dropping messages or overflowing the network buffers. We also found that the Race-to-finish approach for streaming applications is not the best power saving strategy which aligns with previous findings in a similar application context of multimedia stream processing. Moreover, our experiments show that the memory controller frequency has a significant impact on the overall power consumption compared to the CPU frequency settings. Finally, the room for power saving at low-load application phases is small because the dynamic power consumption becomes minor compared to the static power on the Jetson TX1.

Limitations. Although we position this study as an exploratory one, there are several limitations that we plan to address in the future to gain better insight. We summarize these limitations as follows:

Computation model: We focus only on stateless stream-based computations. We believe that a stateful stream-based computation is more challenging and covers more interesting use cases such as dynamic graph algorithms and recurrent neural networks.



(a) SAXPY kernel with 3 worker threads



(b) Euclidean distance kernel with 3 worker threads

Fig. 4: Impact of memory controller impact on power consumption. CPU frequency is 1.734 GHz with 3 out of 4 active cores.

Heterogeneous computing: We only study the impact of frequency tuning of the CPU cores and the memory controller. However, the main computation engine on the Jetson platform is the GPU which is responsible for most of the performance delivered by the platform. Hence, incorporating the GPU and CPU in a heterogeneous use case where the streamed workload is distributed between them is more challenging and opens more opportunities for energy savings. Additionally, incorporating the GPU requires data sharing with the CPU, thus, allowing us to explore the unified memory capabilities on the Jetson platform.

Workload: We study the performance-power trade-off for a set of low-level linear algebra kernels, incorporating real world applications allows us to better understand the practical limitations via a thorough evaluation on real life data sets.

References

[1] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, “The cost of a cloud: research problems in data center networks,” *ACM SIGCOMM computer communication review*, vol. 39, no. 1, pp. 68–73, 2008.

[2] S. Mittal, “Power management techniques for data centers: A survey,” *arXiv preprint arXiv:1404.6681*, 2014.

[3] A. Shehabi, S. Smith, D. Sartor, R. Brown, M. Herrlin, J. Koomey, E. Masanet, N. Horner, I. Azevedo, and W. Lintner. (2016) United states data center energy usage report. [Online]. Available: cloudfront.escholarship.org/dist/prd/content/qt84p772fc/qt84p772fc.pdf

[4] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[5] A. Putnam, A. Caulfield, E. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, E. Peterson, A. Smith, J. Thong, P. Y. Xiao, D. Burger, J. Larus, G. P. Gopal, and S. Pope, “A reconfigurable fabric for accelerating large-scale datacenter services,” in *Proceeding of the 41st Annual International Symposium on Computer Architecture (ISCA)*. IEEE Press, June 2014, pp. 13–24. [Online]. Available: https://www.microsoft.com/en-us/research/publication/a-reconfigurable-fabric-for-accelerating-large-scale-datacenter-services/

[6] Google Tensor Processing Unit. Accessed: Feb, 2019. [Online]. Available: https://cloud.google.com/tpu/

[7] R. Buyya, A. Beloglazov, and J. H. Abawajy, “Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges,” *CoRR*, vol. abs/1006.0308, 2010. [Online]. Available: http://arxiv.org/abs/1006.0308

[8] B. Anton and B. Rajkumar, “Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers,” *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.1867

[9] C.-M. Wu, R.-S. Chang, and H.-Y. Chan, “A green energy-efficient scheduling algorithm using the dvfs technique for cloud datacenters,” *Future Generation Computer Systems*, vol. 37, pp. 141 – 147, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X13001234

[10] T. V. T. Duy, Y. Sato, and Y. Inoguchi, “Performance evaluation of a green scheduling algorithm for energy savings in cloud computing,” in *2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW)*, April 2010, pp. 1–8.

[11] E. D. Demaine, J. Lynch, G. J. Mirano, and N. Tyagi, “Energy-efficient algorithms,” *CoRR*, vol. abs/1605.08448, 2016. [Online]. Available: http://arxiv.org/abs/1605.08448

[12] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, “Models and issues in data stream systems,” in *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS ’02. New York, NY, USA: ACM, 2002, pp. 1–16. [Online]. Available: http://doi.acm.org/10.1145/543613.543615

[13] L. Golab and M. T. Özsu, “Issues in data stream management,” *SIGMOD Rec.*, vol. 32, no. 2, pp. 5–14, Jun. 2003. [Online]. Available: http://doi.acm.org/10.1145/776985.776986

[14] N. S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan, “Leakage current: Moore’s law meets static power,” *Computer*, vol. 36, no. 12, pp. 68–75, Dec 2003.

[15] M. Weiser, B. Welch, A. Demers, and S. Shenker, “Scheduling for reduced cpu energy,” in *Mobile Computing*. Springer, 1994, pp. 449–471.

[16] ZeroMQ. Accessed: 22.04.2018. [Online]. Available: www.zeromq.org

[17] WattsUp PowerPro Kit. Accessed: 23.04.2018. [Online]. Available: www.powermeterstore.ca/p1206/watts_up_pro.php

[18] T. Instruments. (2019) Bidirectional current/power monitor with i2c interface. [Online]. Available: http://www.ti.com/lit/ds/symlink/ina219.pdf

[19] NVIDIA, “NVIDIA Tegra X1 Mobile Processor,” NVIDIA Coporation, Tech. Rep., 12 2016.

[20] K. R. Stokke, H. K. Stensland, C. Griwodz, and P. Halvorsen, “Load balancing of multimedia workloads for energy efficiency on the tegra k1 multicore architecture,” in *Proceedings of the 8th ACM on Multimedia Systems Conference*. ACM, 2017, pp. 124–135.

[21] K. R. Stokke, H. K. Stensland, P. Halvorsen, and C. Griwodz, “Why race-to-finish is energy-inefficient for continuous multimedia workloads,” in *Embedded Multicore/Many-core Systems-on-Chip (MCSoC), 2015 IEEE 9th International Symposium on*. IEEE, 2015, pp. 57–64.

[22] T. De Matteis and G. Mencagli, “Keep calm and react with foresight: Strategies for low-latency and energy-efficient elastic data stream processing,” *ACM SIGPLAN Notices*, vol. 51, no. 8, p. 13, 2016.

[23] M. Dayarathna, Y. Li, Y. Wen, and R. Fan, “Energy consumption analysis of data stream processing: a benchmarking approach,” *Software: Practice and Experience*, vol. 47, no. 10, pp. 1443–1462, 2017.

[24] K. R. Basireddy, E. W. Wachter, B. M. Al-Hashimi, and G. Merrett, “Workload-aware runtime energy management for hpc systems,” in *2018 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 2018, pp. 292–299.

[25] H. Cai, Q. Cao, F. Sheng, M. Zhang, C. Qi, J. Yao, and C. Xie, “Montgolfier: Latency-aware power management system for heterogeneous servers,” in *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2016, pp. 1–8.