CSE4014 - High Performance Computing
Winter Semester 2021

Project Review 1

Project Title - **Comparison between Huffman coding, arithmetic coding, ANS and an implementation of a parallel compression backup tool**

Submitted to - Dr. Vimala Devi K
Slot - B2+TB2

By -
18BCE0139, Ananya Ganesh, ananya.ganesh2018@vitstudent.ac.in
18BCE0270, Vishaal Selvaraj, vishaal.selvaraj2018@vitstudent.ac.in
18BCE0857, Eesha Shetty, eesha.shetty2018@vitstudent.ac.in

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

1. Abstract -

Most compression softwares like 7-Zip, WinRAR, WinZip etc. run only on the CPU. This is because data dependencies are usually high in compression operations which makes them less parallelizable. However, we can compress data on the GPU by decomposing the data into smaller parts and then compressing it on each GPU core. This can give a significant speedup in compression operations. The project leverages this speedup to create compressed data backups as efficiently as possible. The underlying compression algorithm uses the ANS encoder (Asymmetric Numeral Systems) which combines the speed of huffman coding and the compression efficiency of arithmetic coding.

This project can be divided into two parts, one being the comparison between the compression algorithms like Huffman coding, arithmetic coding and ANS; and the other being the implementation of a parallel compression backup tool.

Our project also includes support for scheduled backups to periodically backup your data to a local hard disk or to a FTP server mounted to your system.

2. Problem Statement and Objectives -

This project aims to create compressed backups as efficiently as possible. In order to support different GPU architectures, OpenCL was chosen as the platform for GPGPU computation. Some of the core objectives of this project are -

1. Allow the user to create/restore compressed backups efficiently.
2. Allow the user to select the hardware on which the data is processed.
3. Allow the user to set how many jobs (in this case work-items) need to be run parallely.
4. Provide the user with a functional CLI (Command Line Interface) to accomplish these tasks.
5. Create schedules to make sure data backups are done on a periodic basis.

3. Literature Survey -

3.1. A Performance Comparison of CUDA and OpenCL

Author - Kamran Karimi, Neil G. Dickson, Firas Hamze
Date of Publication - 2011
Publisher - arXiv preprint arXiv:1005.2581

Problem and Objectives -
CUDA and OpenCL offer two different interfaces for programming GPUs. OpenCL is an open standard that can be used to program CPUs, GPUs, and other devices from different vendors, while CUDA is specific to NVIDIA GPUs. Although OpenCL promises a portable language for GPU programming, its generality may entail a performance penalty.
This paper compares the performance of CUDA and OpenCL using complex, near-identical kernels. It shows that when using NVIDIA compiler tools, converting a CUDA kernel to an OpenCL kernel involves minimal modifications.

Proposed Methodology -
This paper uses Adiabatic QUantum Algorithms (AQUA) which is a Monte Carlo simulation of a quantum spin system written in C++.
The method tested CUDA and OpenCL versions of the application on an NVIDIA GeForce GTX-260. The application goes through the following steps during its run: (1) Setup the GPU (includes GPU detection, compiling the kernel for OpenCL, etc.) (2) Read the input, (3) copy data to the GPU, (4) Run the kernel on the GPU, (5) copy data back to the host, (6) process the returned data using the CPU and output the results.

Limitations -
The conclusions of the comparison are - CUDA performed better when transferring data to and from the GPU. We did not see any considerable change in OpenCL's relative data transfer performance as more data was transferred. CUDA's kernel execution was also consistently faster than OpenCL, despite the two implementations running nearly identical code.
CUDA seems to be a better choice for applications where achieving as high a performance as possible is important. Otherwise the choice between CUDA and OpenCL can be made by considering factors such as prior familiarity with either system, or available development tools for the target GPU hardware.

3.2. Energy-efficient primary/backup scheduling techniques for heterogeneous multicore systems

Author - Abhishek Roy; Hakan Aydin; Dakai Zhu
Date of Publication - October 2017
Publisher - IEEE

Problem and Objectives -
This paper considers energy-efficient and fault-tolerant scheduling of real-time tasks on heterogeneous multicore systems. Each task consists of a main copy and a backup copy which are scheduled on different cores, for fault tolerance purposes.
They identify and address two dimensions of the problem, i.e., partitioning tasks and determining processor voltage/frequency levels to minimize energy consumption. The experimental results show that the proposed algorithms' performance levels are close to that of an ideal solution with optimal (but computationally prohibitive) partitioning and frequency assignment components.

Proposed Methodology -
The paper specifies general principles and algorithms that guide to the solution.
It follows a canonical execution order, in which a given core all primary tasks are started as soon as possible,whereas backup tasks are delayed as much as possible subject to the deadline constraints, and executed at the maximum frequency if needed.
A related framework is the so-called energy-aware standby-sparing technique, in which one of the cores is designated for the primary tasks and the other one for the backup tasks exclusively. However in the paper, the framework for maximum flexibility they allow scheduling the primary and backup copies on both cores, when possible – for that reason, they call the framework mixed primary backup (MPB) assignment. They also propose schemes such as task partitioning and speed (frequency) assignment.

Limitations -

For the optimal scheme, they could only calculate energy consumptions for up to 17 tasks due to its prohibitive computational complexity.

3.3. Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding

Author - *Jarek Duda*
Date of Publication - January, 2014
Publisher - arXiv preprint arXiv:1311.2540

Problem and Objectives -
The modern data compression is mainly based on two approaches to entropy coding: Huffman (HC) and arithmetic/range coding (AC). The former is much faster, but approximates probabilities with powers of 2, usually leading to relatively low compression rates. The latter uses nearly exact probabilities - easily approaching theoretical compression rate limit (Shannon entropy), but at the cost of much larger computational cost.
This article also introduces and discusses many other variants of this new entropy coding approach, which can provide direct alternatives for standard AC, for large alphabet range coding, or for approximated quasi arithmetic coding.

Proposed Methodology -
Asymmetric numeral systems (ANS) is a new approach to accurate entropy coding, which allows to end this tradeoff between speed and rate: the recent implementation provides about 50% faster decoding than HC for 256 size alphabet,with compression rate similar to that provided by AC. This advantage is due to being simpler than AC: using a single natural number as the state, instead of two to represent a range.

Limitations -
1. tANS requires building and storing coding tables: about 1-16kB for 256 size alpha-bet,
2. The decoding is in the opposite direction to encoding, which requires storing the final state and may be an inconvenience, especially while adaptive applications.

3.4. Fractal video compression in OpenCL: An evaluation of CPUs, GPUs, and FPGAs as acceleration platforms

Author - Doris Chen; Deshanand Singh
Date of Publication - January 2013
Publisher - IEEE, 2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC)

Problem and Objectives -

Fractal compression is an efficient technique for image and video encoding that uses the concept of self-referential codes. This paper presents a real-time implementation of a fractal compression algorithm in OpenCL. It shows how the algorithm can be efficiently implemented in OpenCL and optimized for multi-CPUs, GPUs, and FPGAs.

Proposed Methodology -

Our OpenCL implementation of fractal encoding performs the following steps:
• Read image or video data from a file and process into frame buffers
• Process a single image frame
• Create a codebook
• SAD-based search algorithm to find the best matching entries and generation of fractal codes

Limitations -

• The challenge of this programming model is Amdahl's Law, where the application speedup is limited by the serial portion of the application.
• In some applications, acceleration using GPUs was considered; however the work is limited to approximately 1 frame per second and is nowhere near real-time.

3.5. Acceleration of ultrasonic data compression using OpenCL on GPU

Author - Boyang Wang; Pramod Govindan; Thomas Gonnot; Jafar Saniie
Date of Publication - 2015
Publisher - IEEE
Problem and Objectives -

Ultrasonic imaging applications require real-time acquisition and processing of huge volumes of data. Subsequently, reducing the data storage becomes an essential requirement for applications involving ultrasound technology. Compression of the acquired data significantly reduces the storage and also helps in faster data processing. Furthermore, computationally efficient data compression enables real-time data transfer to remote locations for expert analysis.

Proposed Methodology -

• Ultrasonic data compression is performed by utilizing the signal compaction properties of Discrete Wavelet Transform (DWT).
• The input signal samples are decomposed by filtering into lowpass and highpass sub-bands. These sub-bands are further decomposed to get narrower subbands which generate smaller size transform coefficients.
• The sub-bands with very low energy will not carry any required information. Therefore, they can be eliminated to achieve signal compression. The compressed signal can be synthesized to reconstruct the original signal by using inverse DWT.

Limitations -

There aren't any limitations specified in this paper. This study demonstrates an efficient implementation of ultrasonic compression algorithm realized on GeForce GT 750M NVIDIA GPU using OpenCL to accelerate the computational performance. Compression is achieved by using the signal compaction properties of DWT. OpenCL program executed on NVIDIA GT 750M GPU completes the compression of 33 MBytes into 6.6 MBytes (80% compression) in 0.289 seconds, whereas the C++ program implemented on Intel (R) Core ™ i7-4500U CPU requires 1.236 seconds for performing the same operation.

3.6. Optimizing Parallel Reduction on OpenCL FPGA Platform – A Case Study of Frequent Pattern Compression

Author - *Zheming Jin; Hal Finkel*
Date of Publication - 2018
Publisher - IEEE
Problem and Objectives -

Field-programmable gate arrays (FPGAs) are becoming a promising heterogeneous computing component in high-performance computing. To facilitate the usage of FPGAs for developers and researchers, high-level synthesis tools are pushing the FPGA-based design abstraction from the register transfer level to high-level language design flow using OpenCL/C/C++.

Inspired by the reduction operation in frequent pattern compression, the author transforms the function into an OpenCL kernel, and describes the optimizations of the kernel on an Arria10-based FPGA platform as a case study.

Proposed Methodology -

• The original kernel is part of an open-source library for the study of practical data compression algorithms for on-chip caches. Frequent pattern compression is based on the observation that a majority of words fall under one of a few compressible patterns. The scheme defines a set of patterns and then uses them to encode application words with fewer bits of data. As shown in the kernel, the function accumulates the total length of the encoded word values by matching each word with the predefined patterns. The matching of the words is independent of each other.

• To evaluate the effect of coding style upon the kernel performance on an FPGA, we propose four versions of the baseline OpenCL kernel. Each kernel is a single work-item kernel. The input word “values” and the output length of “compressible” are accessed from the global memory address space specified by the __global qualifier. The restrict keyword is added for each global address space. Inserting the keyword in pointer argument prevents the compiler from creating unnecessary memory dependencies between non conflict memory load and store operations.

Limitations -

There aren’t any limitations specified in this paper. FPGAs are becoming a promising heterogeneous computing component in high-performance computing. The HLS tools are pushing the FPGA-based design abstraction by allowing users, who have little experience in FPGA development, to transform a program written in high-level language to a hardware implementation. Currently, there are few studies on the atomic functions in the OpenCL-based

design flow on the FPGA. In this paper, we evaluate and optimize parallel reduction using frequent matching compression as a case study on an Arria10-based FPGA platform.

4. Proposed Alternate Methodology -

ANS entropy coders require dynamic memory allocation for encoding data. However, dynamic memory allocation in OpenCL kernels requires a custom heap implementation which is inefficient. Our proposed method runs the ANS entropy coder on fixed-size memory. The residues (uncompressed data) is compressed separately on the CPU. If the compression ratio is greater than 1, there are no residues.

The proposed solution uses a zero-order context model (frequency distribution) for modelling symbol probabilities. The frequency table is built on the GPU using data decomposition and the sub-tables are merged to form one global frequency table. This frequency table is then supplied to the ANS encoder for entropy coding.

Finally, after the entropy coding phase, the compressed data, residue indices, metadata and the frequency table are written to a file.

In order to give granular control over the performance, the user is allowed to select the hardware on which the operation is run, set memory limits, set data size limits, set maximum work-items etc. For backups, all files in the source directory are recursively processed and stored in the target directory taking the user-defined parameters into account.

The project can also be run as a background service to perform scheduled backups. In the case of UNIX/Linux environments, the user can mount a FTP server to the filesystem and use the backup tool to backup data over the network. Since the backups are compressed, this also saves a lot of bandwidth that is wasted due to data redundancy.

References -

1. Karimi, Kamran, Neil G. Dickson, and Firas Hamze. "A performance comparison of CUDA and OpenCL." *arXiv preprint arXiv:1005.2581* (2010).
2. A. Roy, H. Aydin and D. Zhu, "Energy-efficient primary/backup scheduling techniques for heterogeneous multicore systems," 2017 Eighth International Green and Sustainable Computing Conference (IGSC), Orlando, FL, USA, 2017, pp. 1-8, doi: 10.1109/IGCC.2017.8323569.
3. Duda, Jarek. "Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding." *arXiv preprint arXiv:1311.2540* (2013).
4. D. Chen and D. Singh, "Fractal video compression in OpenCL: An evaluation of CPUs, GPUs, and FPGAs as acceleration platforms," 2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC), Yokohama, Japan, 2013, pp. 297-304, doi: 10.1109/ASPDAC.2013.6509612.
5. B. Wang, P. Govindan, T. Gonnot and J. Saniie, "Acceleration of ultrasonic data compression using OpenCL on GPU," 2015 IEEE International Conference on Electro/Information Technology (EIT), Dekalb, IL, USA, 2015, pp. 305-309, doi: 10.1109/EIT.2015.7293358.
6. Z. Jin and H. Finkel, "Optimizing Parallel Reduction on OpenCL FPGA Platform – A Case Study of Frequent Pattern Compression," 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Vancouver, BC, 2018, pp. 27-35, doi: 10.1109/IPDPSW.2018.00015.