Improving the Performance of Proof of Work-Based Bitcoin Mining Using CUDA

Seid Mehammed
Department of Computer Science
Woldia University, Woldia, Ethiopia

Dagmawi Lemma
Department of Computer Science
Addis Abeba University, Addis Abeba, Ethiopia

Abstract:- The most dominant block chain consensus algorithm is Proof of Work (POW). It is an algorithm, which scales up the bitcoin transaction well globally, by competition a cryptographic hash function. This process is named mining. POW-based bitcoin mining is a well-known problem of computational and memory-intensive algorithms.

On the other hand, the high-threaded CUDA architecture has become with enhanced performance for a various range of computation and memory-intensive applications. Thus, feature of massive number of software threads with low overhead context switch provides high computational throughput and hides the memory access latencies. However, it is not effective enough for all applications because of two challenges that directly affect performance such as scheduling new threads and the overhead to startfresh kernels on the CUDA. Existing work tried to model performance of POW-based mining from various aspects. However, no model considers all of these factors came together at the same time.

The main contribution of the articlecombination of the POW-based bitcoin-mining algorithm with a focus on the higher-level analysis of algorithm performance and lower-level details about runtime configuration (thread per block) and scheduling on CUDA.

The results indicate that the model can be effectively use on various optimization techniques. It's able to get a performance, which is almost 4 times when compared to baseline serial algorithm of POW-based mining implementation.

Keywords:- *POW, CUDA, Bitcoin-Mining, Blockchain, Thread, Thread-block.*

1.

I. INTRODUCTION

Cryptocurrency is a digital asset designed to work as a medium of exchange and it uses strong cryptography to control the creation of additional unit, secure business transactions, and verify the transfer of asset [1, 2, 3]. Bitcoin is one of the cryptocurrencies, which are decentralized electronic payment platforms that particularly work in a peer-to-peer (P2P) network.

Bitcoin has attracted substantial interest in recent years from the overall public. It was the first established cryptocurrency, with the first trade in 2009 [4].

The success of bitcoin is the result of its features such as Security, Privacy, as users manage their data, Immutability, and Transparency, due to distributed ledger in a distributed cryptocurrency rather than all of the traditional services provided by the third party[5].

Other cryptocurrencies such as Litecoin, and Ethereum, use the Proof-of-Work (POW)-based mining algorithm. The POW consensus protocol gets its success to be powered by a blockchain to control as opposed to centralized digital currency such as Master Card, PayPal, and Visa [2].

Mining is the process of verifying all transactions on the bitcoin blockchain in a P2P network, and adecentralizedsecurity mechanism for P2P digital currency[6]. A blockchain is an electronic ledger of transactions, consisting of a series of blocks of transactions, each block containing a block body and block-header, within a trusted network infrastructure[6, 7].

Each miner (node) calculates a hash over the block header to find a nonce value. For example, when the miner wants to add a new block to the blockchain, all data of the bitcoin block-header are first to perform double hashing (SHA-256) with a concatenated nonce (32-bit length). After hashing check if the result is less than the targets, then add the block into the blockchain and share for everyone in the bitcoin community (node) acknowledges the new block. If it is not less than the target, then the nonce is changing by a random number generator and this keeps iterating 4 billion times sequential until finally, the requirements are met. This technique is called POW.

Before the invention of the Internet, the banking system was based on manual work. With the advent of the Internet the banking, the system is online systems. The identity may include customer name, account number, recipient's details, mobile number, and any identity card number. The details provide by customer to perform transaction can be acquire by hacker even when there is security in place and their accounts can be hacked. Hence, online banking has become a major security concern even though there are many advantages.

To overcome this drawback, the idea of the digital currency known as cryptocurrency was introduced with bitcoin in 2009 [1]. Cryptocurrency uses strong cryptographic algorithms to monitor decentralize and secure cryptocurrency transactions.

Cryptocurrency is a decentralized system based on blockchain as against the current banking system, which is based on a central banking system using centralized digital currency. Through POW-based mining, all financial crypto currency transactions are public.

One of the major challenges to the existing algorithm of POW-based mining is highly computational and memory-intensive algorithm. Due to that, the mining process takes a lot of time for block creation. As a result, a process for one block creation takes a time of average 10 minutes, transaction throughput of around 3–10 TPS, and the transaction confirmation takes time [1, 6, 8].

Compared to a traditional payment systems, it supports small number of transactions per second, for example, PayPal and Visa can process up to several thousand transactions per second[9, 10].

Thus, this algorithm is not fit for large networks that require vast numbers of transaction processes[11]. However, the POW-based bitcoin network as a decentralized payment platform, by itself cannot support the global market anytime soon[12]. Therefore, it needs to further enhance the performance of POW-based mining algorithms.High-performance implementation of POW-based mining algorithms is very useful in constructing fast and secure virtual payment system[9].

Existing improvement of POW-based mining by using special hardware FPGA and ASIC implementations were introduce. However, FPGA or ASIC mining compromises the democratization and decentralization of the bitcoin-network and its expensive hardware cost.

On the other hand, CUDA enables developers to speed up performance for computing and memory-intensive applications, but not effective enough for all applications because of two challenge that directly affect performance. First, there is no opportunity for the scheduler to schedule new threads when possible. Second, there is overhead to start new kernel on the CUDA.

Researchers designseveral performance models that detention memory hierarchy for several types of high-threaded CUDA architecture such as, global memory, shared memory, and many cores, to enhance the performance of many applications. However, here is narrow literature study this relation on high threaded CUDA architecture with POW-based mining algorithm[13].

Therefore, no highly threaded CUDA architecture permits an unlimited number of threads, and different Thread_Per_Block counts result in a POW-based bitcoin mining of diversified performance.

The research tries to answer the following questions

What are the current statesof the art performance of POW-based mining algorithms and CUDA?

- How does the difference of thread and thread-block counts effect the scheduling and the real performance of bitcoin mining?
- How can the actual performance be predict for POW-based mining on runtime configurations?

Therefore, we focus on the bridging of the POW-based mining algorithm and high-threaded CUDA platform the tradeoffs concerning general optimal performance.

Study over-all problem of algorithm first to understand, analyze, and finally optimize the algorithm performance on high-threaded CUDA platform and programmers to develop a solution with limited drawbacks and high performance. This is a wide problem, not all type of application obvious similar performance pattern on high-threaded CUDA platform.

From the theoretical side, we can effectively use the number of Instruction Level Parallelism (ILP), Data Level Parallelism (DLP), and Thread Level Parallelism (TLP) to measure runtime assuming instructions take constant time.

For the problem side, need to identify input parameters and variable parameters in POW-based mining algorithm. For architecture side, need to identify fixed memory-sizes and configuration parameters (threads, thread-blocks, and warp) for CUDA platform.

A. *Related work*

Several research directions are propose to enhance the performance of POW-based bitcoin mining in a public blockchain. In this paper, we have categorized related work done so far based on the performance enhancement approach: unrolling loop for SHA-256, pipeline, and another different approach, etc.

a) Unrolling loop of hash SHA-256

We summarize the work, which is, conducted on the well-known unrolling loop to improve the performance of SHA-256.

According to [14, 15] use unrolled architectures to reduce the number of clock cycles required to perform SHA-256 hash computation by implementing multiple rounds of SHA-256 compression function using combinational logic on POW-based bitcoin scenario. This architecture helps improves throughput by optimizing info dependencies involve within the message compression function. As a trade-off, unrolling the SHA-256 architecture comes at the value of a decrease within clock-frequency and rise within the area complexity.

In this article, a new structure using the above-mentioned two optimization methods is design for the SHA-256 algorithm in bitcoin-mining scenario, which improves its computational performance.

b) Pipelining

The goal of pipelining is to optimize the critical path. Pipelined SHA-256 architectures [16, 17, 18] use registers to break the long path of the computation of the working variable A within the message compression function. Pipelining is not as easy to realize because the SHA-256 compression function is design difficult. Useful with the small number of processors, but does not scale up to fit with more processors. One reason pipeline chain does not attain sufficient length.

c) Other Different Approach

L. Dadda et al. [18] proposed an optimization method using the carry-save adder; This approach increases the throughput but this architecture requires additional control circuitry for additional register.

The authors of [16] used combination of techniques such as carry_save_adders and pipeline to increase the performance of SHA-256. Unrolled techniques and pipelines increase throughput of SHA-256. Next, constant inputs and difficulty requirements in the process of bitcoin mining is used to reduce number of cycles per SHA-256 operation.

We propose an optimization strategy to accelerate the performance memory_intensive POW-algorithm.

Courtois et al. [14] explore mining optimizations from an algorithmic perspective to summarized the optimization methods for the SHA-256 algorithm in bitcoin mining applications, but they demand higher area consumption, which increases the mining cost.

d) CUDA performance

Yudanov et al. observed that the serialization effect of thread divergence hurts the performance of CUDA applications noticeably [19]. The authors track down and resolved the source of branch divergence in a simulation of neural networks application and achieved 9X, speedup compare to a baseline CPU implementation.

Baghsorkhi et al. propose an analytical model to predict the performance of a CUDA kernel executing on a generic CUDA architecture based on its compute-to-memory-access ratio, coalesced memory accesses, thread divergence ratio, and shared memory usage of each thread block [20].

These four criteria are sufficient to determine the performance of a CUDA application when its memory footprint is smaller than CUDA memory size.

Liu et al. [21] define a general performance model that predicts the performance of a bio-sequence database scanning application fair precisely.

More recently, Kim et al. [22] design tool to estimation CUDA-memory performance by assembling performance critical constraints.

II. SUMMARY

There are no works on the performance of POW-based mining that integrates with the CUDA platform.

Therefore, to improve the performance of POW-based bitcoin mining using CUDA, there is a need to design new data access techniques that consider multiple constraints. According to the literature, our solution is intend to use better memory access, ILP, and TLP for better performance we proposed for POW-based bitcoin mining using CUDA.

A. The Proposed Solution

a) Overview

Performance on highly threaded CUDA architectures depends on the correctness of the essential algorithm, the effect of the memory sub-system on performance, and the effectiveness of scheduling.

A program runs efficiently only when it launches a large number of threads while not incurring too much memory traffic. Interactions betweenparameters, as well as impact of the algorithm performance, are often, not sound understood.

In this paper, we utilized performance prediction on many-threaded CUDA architecture to improve an integrated framework merging both analyzing algorithm efficiency and calculating the achieving running time based on number of parallelism, occupancy, and latency hiding. We consider memory difficulty determine by the number of memory data transfers from slow-memory to fast-memory as a critical performance parameter.

At the similar time, seeking large thread count per Thread_Block does not always offer high occupancy on Streaming multiprocessors(SM) and cannot assure good performance.

Compared to corresponding multi-threaded CPU baseline implementations, POW-based bitcoin mining executes up to 4 times faster.

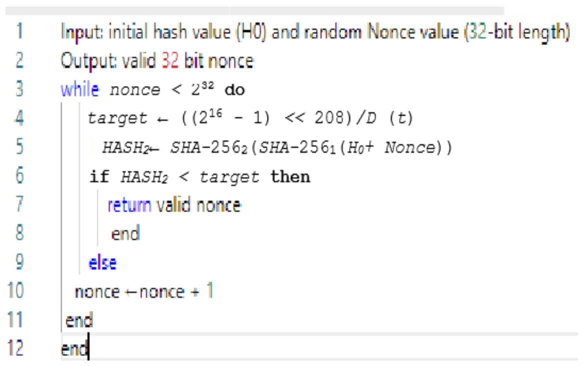
b) Proposed solution Architecture

Our model is useful in several aspects:-

- It can recognize the performance constraint of a POW-based mining algorithm, and decide the algorithm is more likeperformance bound by memory access or by computation.
- Kernel executions launch a grid of thread blocks, each of which consists of several threads. Problems are decomposed, processed on the two-level thread hierarchy, by specifying the grid-size (number of thread-BlocksPerKernel) and block size (number of Threads_Per_Block) for better scheduling frequent access nonce data.
- It helps detect performance enhancement opportunities of two dimension, scheduling, and algorithm design.
- High occupancy,present CUDA scheduler greedily dispatches thread_blocks depending on the resource usage of every thread_block.

In this section, we describe and present architecture of our proposed system on a generic CUDA architecture. In section, Coalescence within a data_block is discuss. In section , we describe Improving Data Transfer Performance using ILP and TLP.

In this section, we formalizestructure of POW-based mining in context of the bitcoin cryptocurrency.



```
1   Input: initial hash value (H0) and random Nonce value (32-bit length)
2   Output: valid 32 bit nonce
3   while $nonce < 2^{32}$ do
4       $target \leftarrow ((2^{16} - 1) << 208)/D\ (t)$
5       $HASH_2 \leftarrow SHA\text{-}256_2(SHA\text{-}256_1(H_0 + Nonce))$
6       if $HASH_2 < target$ then
7           return valid nonce
8           end
9       else
10      nonce ← nonce + 1
11  end
12  end
```

Algorithm 1: POW-based mining algorithm

In Algorithm 1, the input of the $Hash_2$ is the initial hash value of H_0 with concatenating nonce 1024-bit message. The 1024-bit message split into two 512-bit message; then SHA-256_0 calculates a value of the first 512-bit message of bitcoin-header, and SHA-256_1 computes a hash value of the final H_2 512-bit message.

Due to H_2 requirement, 256-bit hash output from SHA-256_1 must be compressinto the final 256-bit hash by using SHA-256_2. In the bitcoin mining process, the final H_2 256-bit hash output from SHA-256_2 is compared to the target value.

This process is repeating 2^{32} times until the hash of SHA-256_2not meets the target requirement. Therefore, the 512-bits of data input precompute by SHA-256_0 function does not the change frequently because it does not include the 32-bit nonce attribute.

On the contrary, the 512bits of data input to SHA-256_1 are updating frequently because of the changing value of the nonce attribute. Whenever the output of SHA-256_1 changes, SHA-256_2 also, needs recomputed and access their values up to 2^{32} times.

As a result, memory-intensive POW-based mining algorithm depends on global_memory bandwidth and latency. Reducing the computing overhead of the $hash_2$ function, while maintaining the necessary data dependencies, the compute overhead of the $hash_2$function (Line 5, on Algorithm 1) should be, minimized.

In this paper, we proposed block data access framework (DAFW) on CUDA platform to improve the performance of POW-based bitcoin mining. Furthermore, we focus on improving POW-based bitcoin mining algorithm performance using CUDA platform by presenting two techniques.

- Using memory access pattern and
- On Thread-level , andInstruction-level parallelism(TLPs and ILPs)

To help performance tasks, we discuss Block Data Access frameworks (DAFW) on the CUDA platform.

Besides, categories of Block Data Access Frameworks (DAFW) are Warp_by_Warp and Block_by_Block. The underlying framework code forming this framework has already been testing, works correctly, and achieves high throughput.

To support our Block DAFW in the context of POW-based mining algorithm, we implement CUDA kernels by using C++ outline generic programming and better performance. Block DAFW provided testingon both ILP and TLP via parameters ‹nWork, nWarpss›.

The **nWork** parameter permits to experimentation with ILP by changing the number of data element per thread in our case 32-bit nonce data per thread.

nWarps parameter permits us to experiment with TLP by changing the number of threads per thread-block.

- Block DAFW

The Block DAFW is based on simple data transfer kernel. For ILP, we use software pipelining on multiple work-items (32-bit nonce data) per-thread per data-block the amount of work per thread is specify via a work_per_thread(nWork) parameter.

For TLP, we support vast thread parallelism via dynamic CTA layout for fixed 1D block_size and 2D grid.

By revolving the Block DAFW into C++, outlinekernels a program-writer canexperimentation with diverse configuration (Work_Per_Thread, CTA layout) byalteringoutlineconstraints. With Block DAFW, data is divide into m set size blocks and organize into 2D and 1D grid; depending on the number of thread and blocks need to wholly cover all data using Layout 1D function.

- First, choose afixed_sizethread_block, fixed-size Work_Per_Thread (nWork), and work out the corresponding fixed-size data_block.
- Second partition data array into m fixed-size data blocks fully covers data-range [0, n).
- Third, launch one thread_block_Per_data-block.
- Forth inside each kernel, map each thread-block onto its matchingdata_block
- Fifth each thread within thread_block process its assigned work (transforms ‹copies› input into output)

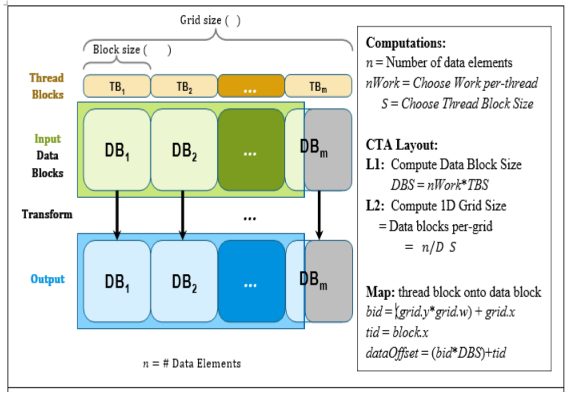


Fig.1: Proposed system architecture

As showed in Fig.1, to make Block DAFW more efficient, we choose fixed thread_block_size (TBS) $=nWarps*WarpSize$, and fixed amount of Work_Per_Thread(nWork), typically in the range (1...4). From it, the fixed data block size (DBS) can be computed as DBS=TBS*nWork. Once, data_block_size is known, a number of data-blocks (m) need to cover the entire data set can be computed as $n\ /DBS$.

One_to_one mappings between thread and data_block makes it relatively easy to compute where the matching data_block starts as block Offs= (bid *DBS). For 1D grid, the blockID(bid) can be computed as bid_s=blockIdx.x and for a 2D grid, *bid* = (*blockIdx.y*gridDim.x*) + *blockIdx.x*.

To get higher performance with Block DAFW, we use two main techniques.

Coalescence within data block and amortizing cost across multiple work item.

- Coalescence with the data block

DAFW mainly deals with the well-organizedmapping of thread-blocks onto data-blocks.

We use two DAFW techniques such as Warp_by_Warp, and Block_by_Block that provision coalescence for high throughput.

Block_by_Block DAFW is straightforward the CUDA provisions coalescence allocating one Thread_Per_Data element consecutively and then striding (strides=TBS) to next row of data within data_block as needed.

Warp_by_Warp DAFW supports coalescence assigns each threads to warps its fixed_size sub chunk of work within data_block and moves each thread_warp to its starting offset, and then strides (strides=WarpSizes) through the sub chunk of work warp by warp.

As we show in Fig. 2 the left and right columns represent Block_by_Block, and Warp_by_Warp DAFW respectively.

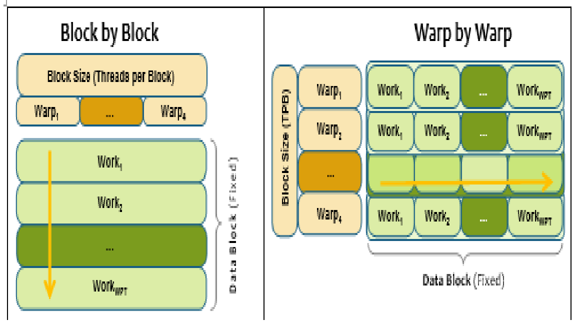


Fig. 2: Data Access Frameworks

As shown in Fig. 2, we focus the differences between Block_by_Block and Warp_by_Warp DAFW. For fixed_sizethread_block of size(TBS), each thread is responsible for processing and data transfer exactly nWork_work-items (nonce value) which totally covers DBS data elements in current data block.

Block_by_Block memory access pattern stride through data_block cooperatively by all threads in thread_block. In general, Block_by_Block DAFW a vertical access pattern on a 2D block, where the number of rows=nWork and the length of each data rows=TBS.

Each thread within thread_block is assigned a single column and then strides to the next assigned row of data elements (stride=TBS) in block_by_block manner.

Warp_by_Warp DAF Wallocates a fixed-size data sub-chunk of work to each thread_warp, with Work_Per Warpwork out as WPW =n Works*Warp Sizes.

In general, Warp_by_Warp DAFW a horizontal access pattern on 2D block layout, where a number of data rows is equal to number of warps_per thread-block work out as nRows =TBS/Warp Sizes, and length of each data row is equal to WPW. For each thread_warp is allocate one data row and strides through its row one data_warp at a time (strides=WarpSizes =32).

Specific experiment generating these traces used two warps(TBS=64) and 6 work-item Per Thread (nWorks=6) per data-block. Each warp is represent by 8 small plus sign per warp.

Block_by_Block DAFW (middle_row, left_panel), ping-pong as two warps stride through their respective six data element; Warp by-Warp BAFW (middle row, right panel) has more serialtrace layout for each warp assign sub-chunk of work.

We suggest Warp_by_Warp DAFW use for two reasons:

- Data access pattern is somewhatmore localized, which can outcome in uncertain growingin I/O throughput 1-3% faster over Block_by_Block DAFW despite additional setup and indexing overhead.
- Each warp proceeds on its assign sub-chunk of work freeof any other warp in thread_block. With Block_by_Block DAFW, each warp wait for all warps to complete transferring data before useful processing work can proceed.

- Improving Data Transfer Performance using ILP and TLP

The existing CUDA data transfer kernel had poor performance.

- ➤ **Growing TLP**:- Increasing the Thread_Per_Block
- ➤ We discuss using our Block DAFW for data transfer primitive.
- ➤ **Growing ILP**:- Increasing the work done per-thread
- ➤ **Loop Unrolling:** -Classic programming technique to handle multiple work-items in CPU serial code.

Each work-item (nonce value) requires registers to trace execution state; we adviseunrolling data in small batches of 2-8 work items to avoid exceeding number of registers and spilling into local memory. Processing more work-itemPerThread is the CUDA kernel equivalent of loop unrolling.

Automatic Loop Unrolling: - The CUDA compiler supports automatic Loop unrolling example(as we shown in Algorithm line occurring before source line #7 for an example) #pragma unroll 4 directive (in lighter-grey) around a looping structure requests CUDA to automatically unroll the wrap code (k=4) times.

```
    // Copy assigned work items (nWork)
    #pragma unroll 4
1:  for (i=0; i<nWork; ++i)
2:    wOff = (i*TBS)+dOff; // Work Item Offset
      // Range check [start, stop]
3:    inRange = (start ≤ wOff) & (wOff ≤ stop);
4:    if (inRange)
5:      D[wOff] = S[wOff]; // Copy Work Item from input to output
6:    end if
7:  end for
```

Algorithm 2: Automatic loop unrolling

Manual Loop Unrolling: Main idea here instructions for ithwork_itemstall, like instructions from other k-1 work-items can be schedule as replacement to hide stall and keep each processing core busy do useful work.

More independent work items increase register pressure limits occupancy so, we experiment with manual unrolling (upto 16 work-items) in-group (2, 4, 8, and16).

As we show in Algorithm 1, the lighter grey if (nWork >=?) { … } Wrappersare elide away at compile time by CUDA compiler. If (nWorks>=) {…} wrappers are resolve at compile time. Unroll code is harder to read, and understand. Besides, up.to k× as many generated instructions also, use k× as many registers. Growing ILP is not only way to hide stall.

TLP techniques work better Growing TLP: - Another way to hide stall uses TLP to recycle instructions from other independent and concurrent warps of execution. Fermi supports upto 48 warps (1538threads) per SM and Kepler supports upto 64 warps (2048threads) per SM. In this section, we show how increasing number of threads per ThreadBlock can growth throughput Upto 218GB/s (2.6× faster than existing data transfer.

```
// Process work items [1-4]
  // Get work offsets
 1: if (nWork≥1) { w1 = (0*TBS)+dOff; }
 2: if (nWork≥2) { w2 = (1*TBS)+dOff; }
 3: if (nWork≥3) { w3 = (2*TBS)+dOff; }
 4:  if (nWork≥4) { w4 = (3*TBS)+dOff; }
  // Range check [start, stop]
 5: if (nWork≥1) { t1 = (start ≤ w1) & (w1 ≤ stop); }
 6: if (nWork≥2) { t2 = (start ≤ w2) & (w2 ≤ stop); }
 7: if (nWork≥3) { t3 = (start ≤ w3) & (w3 ≤ stop); }
 8:  if (nWork≥4) { t4 = (start ≤ w4) & (w4 ≤ stop); }
  // Load data
 9:  if (nWork≥1) { if (t1) { v1 = D[w1]; } }
10:  if (nWork≥2) { if (t2) { v2 = D[w2]; } }
11:  if (nWork≥3) { if (t3) { v3 = D[w3]; } }
12:  if (nWork≥4) { if (t4) { v4 = D[w4]; } }
  // Store data
13:  if (nWork≥1) { if (t1) { S[w1] = v1; } }
14:  if (nWork≥2) { if (t2) { S[w2] = v2; } }
15:  if (nWork≥3) { if (t3) { S[w3] = v3; } }
16:  if (nWork≥4) { if (t4) { S[w4] = v4; } }
```

Algorithm 1: Manual loop unrolling

One cause data transfer in existing CUDA poor throughput codes naively not take benefit of vast parallelism via TLP available on CUDA platform.

Data transfer only launch with CTA outline of only 32 threads (TBS=32) achieving an occupancy of 16.6% (8/48) and 25% (16/64.0) on CUDA version 580 and Titan respectively.

Programmer can directly specify number of threads_per_block as part of CTA.

- Conclusion

Use both ILP and TLP for better performance

For better ILP To:-

- ➢ Support multiple work_item_per_thread (n Work=4).
- ➢ Reduce register pressure, batch work-items in groups of four.
- ➢ Suggest manual over automatic loop unroll.
- ➢ 2-4 work-item per thread are a good starting point.

For better TLP

- ➢ For block size, 128 threads is a good starting point (TBS =128)
- ➢ Growing ILP result up.to 1.2 times faster performance
- ➢ Prefer growing TLP over growing ILP
- ➢ Growing TLP result upto 2.95 times faster performance

Approaches of orthogonal, growth both TLP and ILP for best performance

- ➢ Setup pointers once per thread as an alternative of once per work-item
- ➢ Register faster than shared memory, which is faster than global_memory, which is faster than CPU RAM.
- ➢ Understand CUDAMemory for better performance

For better performance, cluster similar memory access together in a small batch of k work-item (k loads followed by k stores).

A. Experiment and Evaluation Result

a) Introduction

Mainly natures of bitcoin block-header data collect by generating existing bitcoin client tool and the parallelism results of the experiment are discuss.

b) Experimental Procedure

The following subsections discuss the activities and steps to judge POW-based bitcoin mining using CUDA.

- Data Collection

The experimentation of POW-Based Bitcoin Mining begins with generating bitcoin block-header from different bitcoin client browser. We have collected from block to take bitcoin block-header of data for experimentation.

- Tools and Programming Languages

Our baseline hardware infrastructure consists of 3.8 GHz Intel Xeon with 8 hardware threads and 10MB of combined L2/L3 cache, connected to 16GB of quad-channel memory clock at 1850MHz. All CUDA kernels were executing on a GTX 680 CUDA with 1,536 computing cores, each running at 1020MHz, and 2GB of CUDA memory.

All CUDA-based applications were implement using CUDA and GPGPU driver installed on 64-bit Ubuntu 18 Linux with kernel 5.

c) Evaluation result

To measure the performance of our implementation and compare it against on baseline implementations we use the following speed-up formula. Assume n size of Nonce value, and metric is measure by counting the number of clock cycle it takes for all n threads.

Finally, the number of hash per second is calculate as n/s.

Num Hashes = GDIMX*i

Where Hash rate new is the hash rate of the optimized POW-based bitcoin mining on CUDA implementation, while Hash rate existing is un-optimized.

III. EXECUTION OPTIMIZATIONS EFFECTS ON POW-BASED MINING ALGORITHM

To measure impact of optimization describe on the performance of our CUDA implementation, we have set up the following experiments:-

- **Baseline**: - This carrying out, which does not use any optimization at all moreover, all variable are stored in Global_Memory and distributionof memory is doing automatically.
- **Constant memory**: -Variables that do not change during execution, such as SHA-256 constants, are now store in constant_memory. Other variable is still stored in local memory.

SHA-256 makes use of a 64-word constant array k that is derived from fractional-portions of first 64 prime[23]. A word from k is reference with stride-1 in each round of SHA-256 compression function, which is compute twice for each thread. Previously, these constants were stored in local memory as an array unique to each thread that leads to too much unnecessarily long memory accessed. Switch constant-memory to store k result in single largest optimization for CUDA algorithm shooting hash-rate up.to 1.5 times improvement. After this optimization, gains have only been incremental.

- **Shared_memory** with bank conflicts: -Changing variables such as the input buffer and the resulting hashes are now stored in the fast-shared memory.
- **Shared_memory without bank conflicts:-** The distribution of shared_memory is no bank_conflicts occur. With stride access pattern,

every thread can access its bank such that warp serializes are keep to a minimum, which increases performance.

- **Optimized SHA-256:-** Depending on nonce length, only the necessary calculations are performed
- **Constant without bank_conflicts and shared_memory:-** Now changing and non-changing variables preserved in shared and constant_memory respectively.
- **Optimize double SHA-256 with shared and constant_memory:-** Number VI, optimized version of SHA-256 Compression_Function is use as well.

To make fair comparison, the configuration parameters are equal. The number of thread blocks is set 128 and the number of threads per block is 64. As we show in Figure 4.1, performance increase per optimization in black combine optimizations are show in gray. The figure shows us that shared_memory and SHA-256 compression function optimization achieve speed up of three and two times the baseline respectively.

Finally, Fig. 3:shows that storing variables in the shared memory. Therefore, bank conflicts and warp serialize should be avoided.

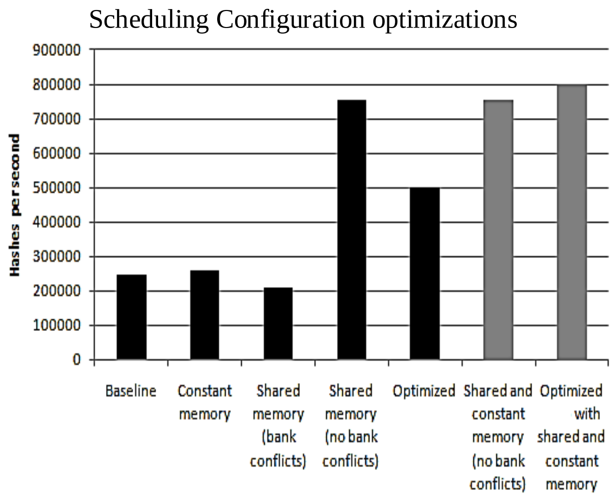


Fig. 3: Execute on CUDA platform.

As we show, Fig. 3:the influence of the block size on the performance while the grid size is set 224. With the optimal implementation and the Threads_Per_Block set to 128, to launch an exhaustive computation with our maximum performance of approximately 800000 hashes per second.

Block sizes up to 224 used well but did not achieve more performance than 128 threads per block. With block sizes higher than 32 kernel refused to run.

Influence of number of threads_per_block on the performance of our graph shows a behavior depending on the number of threads per-block configured; we get stair-like graphs. Multiple warp size(32) and half warp size(16) result in more optimal performance.

The shape of curves represent of speedup achievable relative to fixed serial running time.

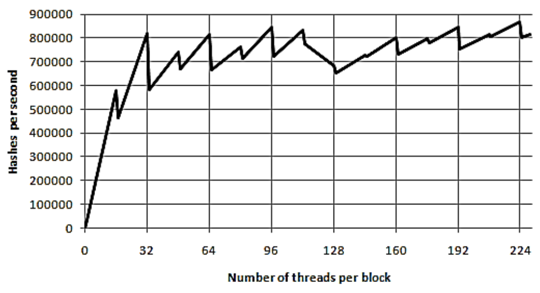


Fig. 0: Number of threads per-block

IV. CONCLUSION

Our work has presented performance analysis for POW-based mining Algorithm Bridge both together with highly threaded CUDA architecture.

High-level theoretical analysis performance of POW-based bitcoin mining algorithm, particularly studying to memory difficulty and compute such that whether the algorithm is memory-bound or compute-bound in standings of how fit the memory latencies are hidden by huge number of threads.

We also develop a framework for mapping them by block DAFW scheduling (‹nWarps, nWork (nonce value) ›) mechanisms. Combining memory complexity and computation is critical performance measurement. These results compare to show that our model is to-

- Classify performance restriction of a POW-based mining algorithm
- Discover and decrease the design and configuration space for tuning kernel execution on CUDA. A kernel execution launches grid of thread blocks, each of which consists of several threads. Problems are decompose, process on two-level thread hierarchy, by identifying the number of thread-Blocks Per Kernel, and a number of Threads_Per_Block for better scheduling frequent access nonce data.
- Identify performance enhancement opportunities of two dimension, scheduling, and POW-based mining algorithm design for any application. Suboptimal configuration of kernel launch can delay the performance.
- High occupancy, present CUDA scheduler dispatches thread-blocks in greedy way depending on resource usage of every thread block.

Our work only works on offline data stored generates bitcoin block-headers data using JSON RPC provided by the bitcoind. Complete experiments testing with the Bitcoin network will future work.

Therefore, we recommend that the proposed approaches can be test in a real Bitcoin-network such a way that network is speed is taken into account.

REFERENCES

[1.] S. Nakamoto, "*Bitcoin: A peer-to-peer electronic cash system, 2008. bitcoin. org/bitcoin. pdf*", Accessed on, pp. 11-18, 2018.

[2.] A. Back, "*Hashcash-a denial of service counter-measure*", sunsite.icm.edu.pl, 2002.

[3.] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "*Bitcoin-ng: A scalable blockchain protocol*", 2016, pp. 45-59.

[4.] S. J. A. o. Nakamoto, "*Bitcoin: A peer-to-peer electronic cash system, 2008. bitcoin. org/bitcoin. pdf*", pp. 11-18, Oct, 2018.

[5.] A. Anjum, M. Sporny, and A. Sill, "*Blockchain standards for compliance and trust*", IEEE Cloud Computing, vol. 4, no. 4, pp. 84-90 2017.

[6.] A. M. Antonopoulos. "*Mastering Bitcoin: Programming the open blockchain".* " O'Reilly Media, Inc.", 2017.

[7.] S. Nakamoto and A. Bitcoin, "*A peer-to-peer electronic cash system*", Bitcoin.–URL: https://bitcoin. org/bitcoin. pdf, 2008.

[8.] B. S. Reddy and G. V. V. Sharma, "*Optimal Transaction Throughput in Proof-of-Work Based Blockchain Networks*", 2019, vol. 28, p. 6.

[9.] I. G. Varsha, J. N. Babu, and G. J. Puneeth, "*Survey on Blockchain: Backbone of Cryptocurrency*", academia, 2020.

[10.] Y. Sompolinsky and A. Zohar, "*Secure high-rate transaction processing in bitcoin*", 2015: Springer, pp. 507-527. ,Y. Sompolinsky and A. Zohar, "*Secure high-rate transaction processing in bitcoin*", in International Conference on Financial Cryptography and Data Security, 2015: Springer, pp. 507-527.

[11.] M. Salimitari and M. Chatterjee, "*A survey on consensus protocols in blockchain for iot networks*", arXiv preprint arXiv:1809.05613, 2018,S. J. Alsunaidi and F. A. Alhaidari, "*A survey of consensus algorithms for blockchain technology*", 2019: IEEE, pp. 1-6

[12.] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "*On the security and performance of proof of work blockchains*", 2016, pp. 3-16. ,K. Croman *et al.*, "*On scaling decentralized blockchains*", 2016: Springer, pp. 106-125.

[13.] L. Marziale, G. G. Richard Iii, and V. Roussev, "*Massive threading: Using GPUs to increase the performance of digital forensics tools*", digital investigation, vol. 4, pp. 73-81, 2007.

[14.] N. T. Courtois, M. Grajek, and R. Naik, "*Optimizing sha256 in bitcoin mining*", 2014: Springer, pp. 131-144.

[15.] Y. K. Lee, H. Chan, and I. Verbauwhede, "*Iteration bound analysis and throughput optimum architecture of SHA-256 (384,512) for hardware implementations*", in International Workshop on Information Security Applications, 2007: Springer, pp. 102-114.

[16.] M. Macchetti and L. Dadda, "*Quasi-pipelined hash circuits*", in 17th IEEE Symposium on Computer Arithmetic (ARITH'05), 2005: IEEE, pp. 222-229.

[17.] A. Satoh and T. J. I. Inoue, "*ASIC-hardware-focused comparison for hash functions MD5, RIPEMD-160, and SHS*", Elsevier, vol. 40, no. 1, pp. 3-10, 2007.

[18.] L. Dadda, M. Macchetti, and J. Owen, "*An ASIC design for a high speed implementation of the hash function SHA-256 (384, 512)*", in Proceedings of the 14th ACM Great Lakes symposium on VLSI, 2004, pp. 421-425.

[19.] D. Yudanov, M. Shaaban, R. Melton, and L. Reznik, "*GPU-based simulation of spiking neural networks with real-time performance & high accuracy*", in The 2010 international joint conference on neural networks (IJCNN), 2010: IEEE, pp. 1-8.

[20.] D. A. Alcantara *et al.*, "*Real-time parallel hashing on the GPU*," in ACM SIGGRAPH Asia 2009 papers, pp. 1-9, 2009.

[21.] W. Liu, W. Muller-Wittig, and B. Schmidt, "*Performance predictions for general-purpose computation on GPUs*", in 2007 International Conference on Parallel Processing (ICPP 2007), 2007: IEEE, pp. 50-50.

[22.] Y. Kim and A. Shrivastava, "*Cumapz: a tool to analyze memory access patterns in cuda*", in 2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC), 2011: IEEE, pp. 128-133.

[23.] I. Ahmad and A. S. Das, "*Hardware implementation analysis of SHA-256 and SHA-512 algorithms on FPGAs*", Computers & Electrical Engineering, vol. 31, no. 6, pp. 345-360 2005.