NTNU – Trondheim
Norwegian University of
Science and Technology

Mining Bitcoins using a Heterogeneous Computer Architecture

Torbjørn Langland
Kristian Klomsten Skordal

Master of Science in Computer Science
Submission date: June 2015
Supervisor: Donn Morrison, IDI
Co-supervisor: Yaman Umuroglu, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

Recent years have seen the emergence of a new class of currencies, called cryptocurrencies. These currencies use cryptography to provide security and peer-to-peer networking to provide a decentralized system. Bitcoin is the most popular of these currencies. It uses a two-pass SHA-256 hash at its core. Producing new bitcoins is done through a process referred to as "mining", which involves a brute-force search for a hash with a specific value. This process requires large amounts of computing power.

Current-generation hardware for bitcoin mining includes highly-optimized ASIC chips which provide huge amounts of performance. However, designers of such chips are having problems with delivering enough power and cooling to the chips. To alleviate this problem, this thesis looks at the possibilities of using heterogeneous computing to reduce power consumption and produce a more energy-efficient mining solution.

A SHA-256 accelerator and a DMA module is developed and integrated into a tile for the Single-ISA Heterogeneous MAny-core Computer, SHMAC, and a system with multiple cores is used to exploit the thread-level parallelism provided by the platform. The system is tested using a benchmark to find out what performance and energy efficiency can be expected when using the system for bitcoin mining.

The results show a maximum performance of 175,7 kH/s when running the benchmark application on 14 cores using the SHA-256 accelerator and the DMA module. The best energy efficiency was obtained when running on 14 cores *without* the DMA enabled, at 163,2 kH/J. The results does not compare well to specialized FPGA-based bitcoin miners, but demonstrates the SHMAC platform's large degree of thread-level parallelism which can be better exploited in other applications.

Sammendrag

De siste årene har en ny type valuta dukket opp, kalt "cryptocurrencies" eller digitale valutaer. Disse valutaene bruker kryptografi til å tilby sikkerhet og peer-to-peer-nettverk til å tilby et desentralisert system. Bitcoin er den mest populære av dem. Bitcoin er bygd rundt en dobbel SHA-256 hash. Å lage nye bitcoins blir gjort via en prosess som kalles for "mining", og involverer et brute-force søk etter en hash med en bestemt verdi. Dette krever store mengder med datakraft.

Den nåværende genersasjonen av hardware for bitcoin mining består av høyt optimaliserte ASIC-brikker som gir store mengder ytelse. Likevel opplever designerne av slike brikker problemer med kjøling og strømtilførsel. For å bøte på dette problemet ser vi på mulighetene for å bruke heterogene arkitekturer for å redusere strømforbruk og lage en mer energieffektiv løsning.

En SHA-256 akselerator og en DMA-modul blir utviklet og integrert i en tile for SHMAC, the Single-ISA Heterogeneous MAny-core Computer, og et system med flere kjerner blir brukt for å utnytte trådnivåparallelliteten som platformen tilbyr. Systemet blir testet med et benchmark-program for å finne ut hvilken ytelse og energieffektivitet som kan forventes når man bruker systemet til bitcoin-mining.

Resultatene viser en maksimal ytelse på 175,5 kH/s når man kjører benchmark-programmet på 14 kjerner med SHA-256 akseleratoren og DMA-modulen. Den beste energieffektiviteten ble observert når programmet ble kjørt på 14 kjerner med DMA-modulen avskrudd, på 163,2 kH/J. Resultatene kommer ikke godt ut når man sammenlikner med spesialiserte FPGA-baserte bitcoin-minere, men demonstrerer likevel hvordan SHMAC-platformens høye grad av trådnivåparallelitet kan utnyttes av andre programmer.

Acknowledgments

We would like to thank Donn Morrison and Yaman Umuroğlu for their work as supervisors on this project, taking time to assist us with all our problems.

In addition we would like to thank Asbjørn Djupdal for assistance in setting up our Versatile Express box.

Contents

Chapter 1

Introduction

In recent years, digital currencies know as "cryptocurrencies" have become popular and most popular of these is bitcoin. The term "cryptocurrency" stems from the fact that a cryptographic hash function is at the core of the algorithms involved. For bitcoin, the SHA-256 hash algorithm is used in a two-pass configuration producing a double SHA-256 hash [17].

Producing new bitcoins is done through a process called mining, described in Section 2.1.1. Mining requires large amounts of computing power, and has led to a quick development in the hardware used in the process, from regular CPUs and graphics processors, to Field-Programmable Gate Arrays (FPGA) and highly specialized Application Specific Integrated Circuits (ASIC). This evolution of mining hardware is described in Section 2.3. Bitcoin mining requires a large degree of thread-level parallelism, because the SHA-256 algorithm does not provide any opportunities to exploit parallelism when calculating a single hash; instead, running several separate SHA-256 computations in parallel provides a possibility for exploiting parallelism to increase the number of SHA-256 computations that can be run in a specified period of time.

Using bitcoin mining as an application, this project develops a SHA-256 accelerator for the Single-ISA Heterogeneous MAny-core Computer (SHMAC), described in section 2.5, which providess a high degree of thread-level parallelism. To achieve higher throughput for the SHA-256 accelerator and the possibility of higher energy efficiency overall, a DMA is also developed. The two modules are integrated into a general-purpose CPU tile, and an estimation of the bitcoin mining efficiency is made using a benchmark application measuring the performance of the system, by measuring how many hashes can be calculated each second and how much power is used when doing the calculations.

1.1 Original Assignment Text

This project aims to develop a bitcoin mining accelerator that will ultimately be used in the single-ISA, many-core, heterogeneous computing platform SHMAC. Bitcoin mining is, at its core, a SHA-256 hashing problem, so part of the assignment will be to keep the interface generic enough such that other cryptographic algorithms can be readily developed.

This part of the project will focus on building an end-to-end and fully functional hardware and software system that is able to fully participate as a miner in the network (that is, mine bitcoins and communicate with peers).

The energy efficiency of the hardware implementation should be evaluated against that of a general-purpose CPU and should constitute a major aspect of the report.

1.2 Comments on the Assignment Text

Recent hardware developments within the field of bitcoin mining, most important of which is the introduction of dedicated ASIC mining chips described in Section 2.3.3, have led to bitcoin mining using CPUs, GPUs and many FPGA-based designs being considered unprofitable and wasteful in terms of the energy expended.

In such an environment, it is doubtful that an FPGA-based SHMAC implementation can compete with ASIC designs. The assignment then basically boils down to: is bitcoin mining possible using SHMAC? Does the mining process benefit from heterogenity, and what performance and energy efficiency can be expected when using SHMAC as a bitcoin miner? How does it compare to existing FPGA-based bitcoin miners.

Chapter 2

Background

In order to understand how and why heterogeneous systems can be applied to the problem of bitcoin mining, it is first important to obtain an understanding of how the bitcoin currency works and what heterogeneous computing is.

2.1 The Bitcoin Currency

Bitcoin, often abbreviated BTC or ₿ , is a digital currency, using a peer-to-peer network to provide a decentralized currency, not relying on banks or financial institutions to process transactions or maintain accounts. An account simply consists of a cryptographic keypair, which is used to sign and verify transactions, and as such anyone can create as many accounts as they wish. The bitcoin project provides a variety of software for interacting with the bitcoin network and managing accounts.

At the core of the bitcoin system is the block chain, a distributed, linked list consisting of blocks which contains the transactions that have been executed on the network since the previous block was generated.

A block is only valid if the arithmetic value of the double SHA-256 hash of its header is below a certain target value. Finding a block that has such a header is computing intensive because it has to be done through a brute-force search, and this prevents the network from being flooded with new blocks. The target value is decided by the network "difficulty" and is set to such a value that, on average, six new blocks are generated per hour.

To get people to participate in creating new blocks, a reward is offered to whomever manages to create a block. This reward also ensures that more money is added into circulation. The size of the reward decreases over time until a predetermined number of bitcoins have entered circulation. The reward is currently 25 bitcoins, each currently worth about \$224 USD[1], and halves every 210 000 blocks. The total number of bitcoins that is to be generated is 21 000 000 bitcoins. [17]

[1]According to `http://blockchain.info`, which tracks various statistics about the bitcoin system

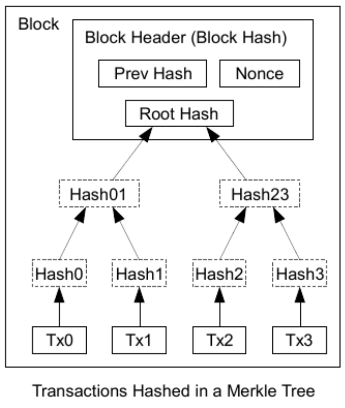


Figure 2.1: Merkle tree in the block header [17]

2.1.1 Mining Bitcoins

The process of creating a new block for the bitcoin blockchain is often referred to as *mining*, refering to the fact that a lot of energy may need to be expended in order to find a valid block.

The process begins with the creation of a transaction that transfers the reward for generating the block into the account of the miner. This transaction is often called the coinbase or generation transaction. All transactions transmitted to the bitcoin network since the last block was generated are gathered and the hash of each transaction is inserted into a merkle tree, which is a tree where where every interior node is labelled with the hash value of its children, see Figure 2.1.

The root of the merkle tree is inserted into the header for the new block together with the hash of the previous block and various other fields specified by the standard. If the hash of the header is below the target value, the block is successfully mined and transmitted to the network. If the hash does not satisfy the demands of the network, a 32-bit field in the block header, called the *nonce*, can be changed to produce a new hash. In addition, transactions can be excluded in order to produce a different merkle root for the header, effectively providing a much larger search space.

If more than one new block is pushed to the network at the same time, the block chain diverges. This situtation is resolved when the next block is generated; the longest block chain is then accepted as the canonical block chain by the rest of the network. [17]

2.1.2 Pooled Bitcoin Mining

Because an increasing amount of computing power is being used to mine bitcoins, the difficulty of finding a block has increased to such a level that no single bitcoin miner can hope to find a block on his or her own any longer. Figure 2.2 illustrates how the difficulty level have risen on the bitcoin network since the beginning. The hashrate of systems used in bitcoin mining is measured in double SHA-256 hashes per second, abbreviated H/s. The current total hashrate is about 396,5 EH/s[2] according to the statistics available from `blockchain.info`. This means that the amount of work needed to find a single block is, on average, 237,9 ZH[3]. Using, for instance, a GPU-based bitcoin miner with a hashrate of 1 GH/s, it would take, on average, $237,9 \cdot 10^{14}$ seconds, or 7 543 759 years to find a block.

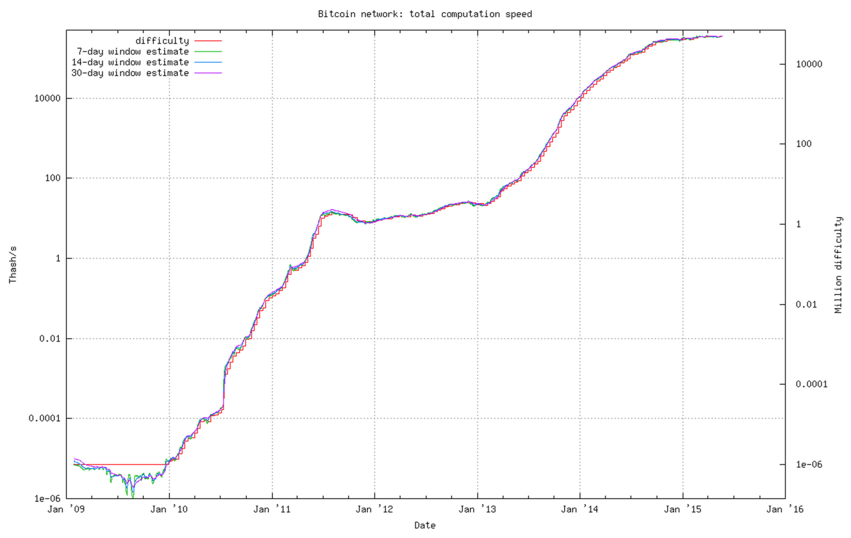


Figure 2.2: The bitcoin mining difficulty, from the beginning until today [1].

To overcome this problem, mining pools were invented. A bitcoin mining pool is a service that distributes work between miners. The work that is distributed has a lower target value than does the bitcoin network itself, meaning that each participating miner can find valid blocks faster. Although most of the blocks submitted to a pool does not fulfill the network requirements, sometimes a block fulfills both the pool's difficulty requirements *as well as* the network requirements; in that case the new block is submitted to the bitcoin network and all the miners who submitted work towards finding the block is rewarded according to their contribution.

As such, pooled mining lets anyone with weaker hardware participate in bitcoin mining in exchange for a smaller reward. However, using hardware such as regular CPUs and GPUs is still considered unprofitable because of these processors' relative performance compared to specialized bitcoin mining ASICs, as discussed in Section 2.3.

2.2 The SHA-256 Hash Algorithm

The SHA-256 hash algorithm is defined in FIPS180-4 [19] and is a member of the SHA-2 family of hash functions. Like all hash functions, it works by accepting an arbitrary amount of input data and producing a constant-length output. The function is one-way, that is, given a hash, it is impossible to determine the input

[2] $396,5 \cdot 10^{18}$ H/s
[3] $237,9 \cdot 10^{21}$ hashes

data used to produce the hash. However, hash functions are deterministic, in that the same input data always produces the same output data.

The SHA-256 algorithm works on blocks of 512 bits. Each of these blocks are expanded into a block of 64 32-bit long words using a message expansion function. Each of these words are used as input to one iteration of the SHA-256 compression algorithm, which uses simple bitwise functions such as shifts, rotations, logic functions and unsigned addition to create a 256 bit output value. Each round of the compression function is dependent on the output data from the previous round in addition to the current input word from the expanded message block and a constant. After 64 iterations of the compression function, the result is added to the intermediate hash values from any previous blocks, or, if there are no previous blocks, the initial hash value, producing a finished hash.

A more detailed description of the algorithm can be found in Appendix A. In the bitcoin system, a double SHA-256 algorithm is used. To obtain a double hash, the output of a previous SHA-256 calculation is simply used as input data to the hashing algorithm. [19]

2.3 Evolution of Bitcoin Mining Hardware

Because of the bitcoin currency's popularity and the competitiveness involved in mining it, there exists many different accelerators to improve the performance and energy efficiency of mining systems. During the currency's history, hardware for bitcoin mining has evolved from regular, general-purpose CPUs to highly specialized ASIC-based systems.

The bitcoin blockchain was started on January 3rd, 2009. At this point, all mining was done using CPU mining. The official bitcoin network client supports mining and was used for this purpose. The fastest CPU miner, a high-end, overclocked Core i7 990x eventually reached 33 MH/s using SIMD extensions in order to improve performance. [5]

2.3.1 GPU Mining

The shift to GPU mining started in July 2010, when the first OpenCL miner was written and used in a private mining setup. In September that year, the first open source GPU miner, which was based on Cuda, was released after the author was paid 10 000 bitcoins, worth about $600 USD at the time, by Jeff Garzik, one of the core bitcoin developers. An open source OpenCL-based miner was released shortly after. Using GPUs for bitcoin mining allows miners to take advantage of the heterogenity offered by common desktop computers to accelerate bitcoin mining. [5]

OpenCL miners typically compute the double SHA-256 hash using a fully unrolled implementation of the compression loop. Multiple hash calculations are run simultaneously exploiting the parallelism offered by GPUs. Many miners tweaked various parameters of their hardware, such as the voltage and clock frequencies of both the video RAM and the GPU core in order to get higher throughput and thus reduce the cost of running each GPU, in order to obtain greater profits.

In order to increase profits, miners started to place multiple GPUs on the same motherboard, providing higher hashrates while still using readily accessible hardware. However, for each GPU added comes an additional cost in wasted resources, such as the additional motherboards needed to run the GPUs, which includes mostly unused hard-drives, RAM and processors. Additional challenges included providing enough power and cooling to the GPUs. The power consumption on each GPU was on average more than 200 W, making the use of rigs comparable with data centers, and the most successful bitcoin mining operations tended to relocate to warehouse space with larger volumes of air for cooling, and cheap industrial power prices. [24]

2.3.2 FPGA Mining

FPGA miners appeared in June 2011, providing better energy efficiency compared to GPUs [24]. Early designs were mostly based on the Spartan 6 FPGAs from Xilinx, and could provide a performance of between 200 MH/s and 220 MH/s per chip, about the same as contemporary GPUs [6], but consuming as little as a fifth of the power used by their GPU counterparts [24]. A representative example of such an FPGA design is the BTCMiner, an open-source bitcoin miner firmware for ZTex' FPGA boards, which can obtain up to 215 MH/s with a power usage of about 9,8 W, giving 21 MH/J [4].

FPGAs provided great advantages to the speed of bitwise functions, which are important for the SHA-256 algorithm. Popular open-source designs were created so that they could be used by different kinds of FPGAs. They consisted of a SHA-256 module which had a configurable depth pipeline, each consisting of the necessary hardware for computing a specific number of rounds of the SHA-256 compression function. Completely unrolling the algorithm created a pipeline of 64 stages, with a throughput of 1 hash per cycle, but it was also possible to specify lesser unroll factors which resulted in fewer pipeline stages and reduced throughput, taking several cycles to perform a hash, but saving registers and logic resources in the FPGAs. The pipelines were duplicated to improve performance.

Power consumption became much higher than typical for FPGAs, with the pipelined design giving an extremely high internal activity factor in the FPGA. Development boards could not provice enough power and heat dissipation for sustained usage, and development of custom boards that focused on providing enough power and cooling with minimal unused resources, became popular. [24]

2.3.3 ASIC Miners

Neither GPUs nor FPGAs can compare to ASICs, specialized chips that started to appear on the network in the beginning of 2013 [7]. It was the need for more high-performance and power efficient bitcoin miners caused implementors of bitcoin miners to turn to ASIC solutions. The first ASIC solutions were crowdfuded projects, raising money over the internet or through preorders from future users.

One of the first ASIC designs were Butterfly Labs' ASIC miners. Having experience developing FPGA based mining solutions, the company created an ASIC chip with 16 double SHA256 modules, the equivalent of 16 FPGA-based miners using a 65 nm process. The end-product ended up being delayed when the chips ended up consuming more power than expected and had to be underclocked in order to be able to cool the chips properly.

ASICMINER was another of the earliest designs, including only one SHA256 hash unit on a chip, replicating a single FPGA-based design at a much higher frequency, with better energy efficiency and a cheaper price. [24]

The Goldstrike 1

Another ASIC design is the Goldstrike 1 architecture, noteworthy because its architecture is described in an article in IEEE Micro by Javed Barkatullah and Timo Hanke [3]. Because of the competitive nature of bitcoin mining, it is not common for designers of ASIC solutions to reveal too many details about the architecture of their chips and few scientific papers exist on the topic.

The Goldstrike is an ASIC-based bitcoin mining solution capable of reaching 2 TH/s. To reach the performance requirements, a bitcoin mining core was created that was able to perform at 125 GH/s. Four of these were combined in one package, with four of these packages then combined on a circuit board to produce the target performance.

The Goldstrike cores consist of an architecture with 120 hash engines running in parallel. A hash engine searches through all possible values of the 32-bit nonce field in the bitcoin header (see Section 2.1.1) in order to find a valid hash. The hash engines use a pipelined double-SHA-256 implementation, with each round of the hash calculation unrolled into a pipeline.

The finished ASIC chips, each with four bitcoin mining cores, were measured to provide 504 GH/s while using 500 W of power. This gives an energy efficiency of about 1 GH/J. Interestingly, the architecture provided challenges with regards to to powering and cooling the chips. Because the architecture uses all available computing power for bitcoin mining all the time while the chips are in use, no parts of the chips can be turned off to save power. [3]

2.4 The Problem of Dark Silicon

Moore's law states that transistor density on integrated chips will continue to double every two years. This has turned out to match reality pretty well, and in addition, native transistor speeds are increasing with a factor of 1.4. According to the principle of Dennard scaling, which predicts that the power density of transistors, that is the voltage and current used to operate the transistor, would scale down with the size of the transistors, this would not be a problem; however, the energy efficiency of transistors is only improving by a factor of 1.4.

Under a constant power-budget, there is a shortfall of a factor of 2 in the energy budget, and this utilization potential of a chip is falling exponentially by 2 times for each generation. If the power limitation were to be based on the current generation, then designs would be 93.75 % dark in eight years. This gives rise to the term “dark silicon”, where a chip must either be underclocked or parts of it turned off in order to stay within a defined power budget, giving rise to “dark” areas of the chip. This is especially true for chips where the cooling solutions are no longer efficient enough to remove the generated heat from a fully powered chip.

In an attempt to work around this problem, the CPU industry moved to using multicore processors around 2005. Since the gains from improving instruction level parallelism were diminishing, the industry focused more on thread level parallelism and throughput through multiprocessing [10]. However, adding multiple cores does not circumvent the problem in the long run. Multicore chips will not scale as transistors shrink, and the fraction of a chip that can be filled with cores running at full frequency is dropping exponentially with each processor generation. Large fractions of the chip will be left dark — either switched off for a long time, or significantly underclocked. [25]

The case of bitcoin mining may be close to the worst case for dark silicon optimizations, far beyond multicore CPUs or GPUs. Due to bitcoin mining's high processing requirements, the logic needed to run the algorithm cannot be turned off without losing performance. Butterfly Labs ran into this problem with their 65 nm chip, and had to scale back the performance to reduce power consumtion and manage to cool the chip properly. While bitcoin mining began as a "race to ASIC", energy costs now determine which ASICs are the most profitable and therefore energy efficiency is a problem that must be addressed. [24]

2.4.1 Approaches to the Problem of Dark Silicon

To work around the problem of dark silicon, several approaches have been suggested. In [25], Taylor have listed up three particular approaches: shrinking the chips, dimming the chips, and specializing the chips. In addition, there is always the possibility that future technology will be a "deus ex machina" and solve the problem in an unexpected fashion.

Shrinking the Chips

Instead of having dark silicon on the chip, one can simply shrink the chip itself. All chips may be shrunk to a certain small extent, but the only chips that truly can gain from shrinking only, will be those on which dark silicon is only a waste, and cannot be used to further enhance the product. Specalizing into these types of shrunk chips may, however, turn out to be less profitable business, as further generations of Moore's law adds little benefit. They are not exponentially cheaper either, given that mask cost, design cost and I/O pad areas does not scale equally, and power density will rise exponentially, increasing the chip temperature.

Dimming the Chips

The term "dim silicon" refers to the use of general purpose logic that typically employs heavy underclocking or infrequent use. As such, large amounts of otherwise dark silicon area is put to productive use while meeting the power budget. While the fraction of dark transistors on chips increases exponentially, the silicon area becomes exponentially cheaper as a resource, relative to the power and energy consumption. This gives the opportunity to spend area to buy energy efficiency. Dark silicon areas can be populated with logic that is used only part of the time.

Lately, more elegant methods are emerging. Among the dim silicon techniques are dynamically varying the frequency with the number of cores being used, scaling up the amount of cache logic, employing near threshold voltage (NTV) designs, and redesigning architectures to accommodate bursts that temporarily allow the power budget to be exceeded, such as Intel's Turbo Boost technology. [25]

Specialization

Instead of using general purpose logic, this approach focuses on specialized use of logic for selected tasks. Within a set of processors, each are specialized for a subset of tasks, increasing energy efficiency or performance compared to a general purpose processor, for those particular tasks. An application is executed on the processor which is deemed the most efficient for the task. Cores not in use are power and clock gated so as to not waste energy unnecessarily.

Challenges with these systems are expected as well, one of them being the so-called "Tower of Babel" crisis, where the notion of general-purpose computation becomes fragmented, and the clear lines of communication between programmers and software and the underlying hardware is no more. There are several cases of overspecialization problems between accelerators, where many of them cannot be used for closely related classes of computation. For instance, CUDA for NVidia GPUs cannot be used for similar architectures, such as AMD GPUs. Specialized hardware also risk becomming obsolete when standards are revised. [25]

Deus Ex Machina

The term "deus ex machina" comes from literature and theater, in which the protagonists seem increasingly doomed until the very last moment, when something completely unexpected comes out of nowhere to save the day. In the case of dark silicon, a deus ex machina would be a major breakthrough in semiconductor device technology. The required breakthrough would have to be very fundamental, making it possible to build circuits out of devices other than MOSFETs[4]. There are physical limits to what can be done with, for instance, the leakage from MOFSET transistors, and transistors made of other materials may go beyond these limits. New transistors must also be able to compete with MOFSETs in performance. Tunnel field-effect transistors (TFET) and nano-electromechanical system switches (NEMS) are examples of inventions that may hint to order-of-magnitude improvements to the leakage problem, although they still fall short in performance. [25]

2.4.2 Benefits of Heterogeneous Architectures

Specialized heterogenenous architectures offers various possibilities for improved performance and energy efficiency; this could be in the system's ability to adapt to various applications or even external conditions such as power or temperature conditions [11, 13, 12]. This has been explored in [11], [13] and [12]. By combining different processors from the Alpha Family, Kumar *et al* has proven that

[4]Metal–Oxide–Semiconductor Field-Effect Transistor

heterogenous architecture does excel in energy efficiency and performance, compared to a homogeneous system. In a simluation the EV4 (Alpha 21064), EV5 (Alpha 21164), EV6 (Alpha 21264) and a single-threaded version of EV8 (Alpha 21464) were combined into a heterogeneous multiprocessor. The SPEC2000 benchmark were run with one application on one core at a time, with the others powered down. The simulation showed that 32 % of the energy were saved with performance loss of only 2.6 % relative to the EV8 core, when using best static scheduling, and using dynamic core switching with simple heuristics further improved the results. [11]

In another simulation where system performance were the main goal, a combination of EV5 and EV6 onto a 100 mm^2 chip were simulated, where the size of one EV6 equals the size of four EV5. The tradeoff was between greater single-threaded performance of the more complex EV6 versus greater thread-level parallelism of adding more EV5 cores. The best homogeneous designs could fit either four EV6 cores or 20 EV5 cores on the chip, and for the simulation, three EV6 and five EV5 were chosen to form a heterogeneous multi-core processor. Instead of one single application, the simulation tested the best global assignments when running multiple threads, from a small number of the SPEC2000 benchmarks. Threads that exploited the EV6 better would be assigned to the more complex cores. Compared to a homogeneous multi-prosessor system with four EV6, the simulation showed that the heterogeneous system performed up to 37 % better with an average 26 % improvement over the configuration, considering 1-20 threads. Compared to 20 EV5 cores, the performance was up to 2.3 times better, and averaged 23 % better over that same range. Using dynamic heuristics for core assignment further increased the performance. Additionally, the heterogeneous system was tested for its tolerance against increasing job queues, and simulation showed that a homogeneous system with four EV6 became saturated at a significantly earlier point than the heterogeneous system, showing that the latter has higher tolerance to larger amounts of wor, because of the increased parallelism. [13]

[12] takes a closer look at what can be considered a good heterogeneous design, in making the heterogeneous multiprocessor from scratch instead of using pre-existing core designs. Using pre-existing cores presents lower flexibility in choices, while best heterogeneous designs are composed of specialized core architectures. The study in [12] came to the following conclusions:

- The most efficient heterogeneous multiprocessors were not constructed from cores that make good general-purpose uniprocessor cores, nor cores that would appear in good homogeneous multicure architectures.
- Each core should be individually tuned for a class of applications with common characteristics.
- And performance advantages of heterogeneous multiprocessors also hold for completely homogeneous workloads (where the task has no differences). In those cases, the diversity across different workloads are exploited.

The more constrained the area or power budget, the more benefit the heterogeneous designs provide, as homogeneous systems were able to run to the fullest on generous budgets only. As future designs become more aggressive, with more cores added to the chip, the results underline the impact heterogeneous design will have for the future.

2.5 The Single-ISA Heterogeneous MAny-core Computer

The Single-ISA Heterogeneous MAny-core Computer (SHMAC) is an architecture designed for investigating heterogeneous systems at all abstraction levels, as illustrated in Figure 2.3. It implements a tile-based architecture with a mesh interconnect. All processor tiles implement the same ARM ISA and the same memory model, in order to achieve a common programming model [9]. A high-level illustration of SHMAC can be seen in Figure 2.4.

Because the programming model is kept constant, the underlying implementation of the system can change. This means that the same application can be run on SHMAC instances with different tile layouts or on SHMAC instances implemented on different physical media, such as in FPGAs or ASICs. Currently, SHMAC is only run on FPGAs.

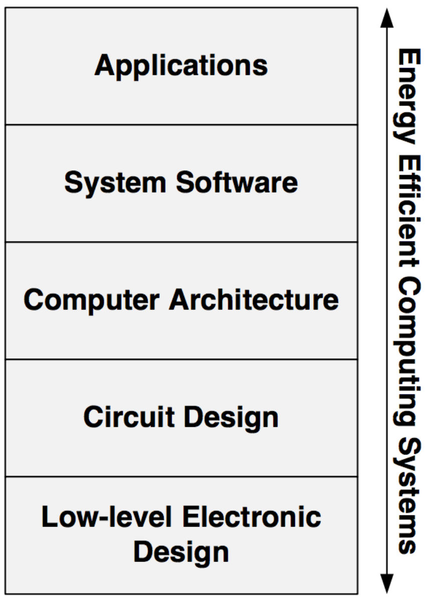


Figure 2.3: Levels of abstraction in computing systems [9].

2.5.1 Versatile Express

The platform used to run SHMAC is a Versatile Express from ARM. Versatile Express is a family of development platforms, made for providing an environment for prototyping future System-on-Chip (SoC) designs. A Versatile Express system typically consists of a μATX motherboard, to which various daughterboards can be added. In some cases, a daughterboard can also allow for additional daughterboards to be inserted to it, as illustrated in Figure 2.5. Commonly used are the CoreTile Express and LogicTile Express boards. CoreTile Express contains the CPU cores that runs the system. The version used in this project contains four Cortex A9-processors, running Linary Linux

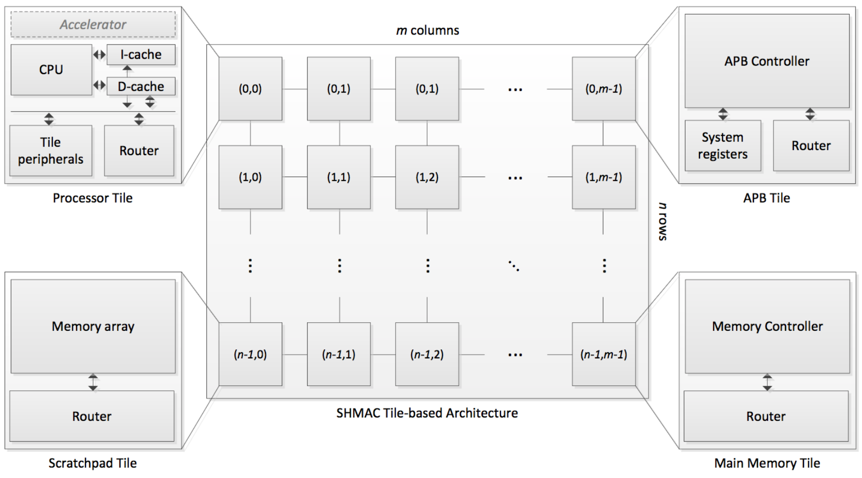


Figure 2.4: High-Level architecture of SHMAC [9].

13.12, with Linux kernel version 3.14.25. Only one core is used by the Linux host system.

The LogicTile Express contains one or more FPGAs, the number depending on the version, suitable for prototyping SoC designs. Our project uses the LogicTile Express 20MG, containing a Virtex-7 XC7V2000T FPGA by Xilinx, and a 4 GB DDR3 memory chip. The architectural relationship between the CoreTile Express and the LogicTile Express used in our system can be seen in Figure 2.6, where both tiles are plugged into the same motherboard. [14, 15]

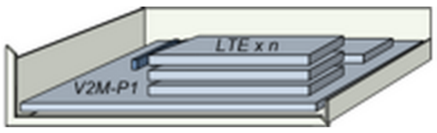


Figure 2.5: Illustration of Versatile Express build [14].

2.5.2 SHMAC Architecture

SHMAC is a tile-based architecture, with the processing elements laid out in a rectangular grid with neighbour-to-neighbour connections, using a mesh interconnect with XY-routing.

Several tile types are supported. The main ones are:

Processor Tile Currently two versions are supported: the open source ARM Amber CPU, and an improved version called Turbo Amber. The original Amber core provides support for the ARMv2a instruction set [22], while the Turbo Amber provides support for the ARMv4t instruction set, giving the core the ability to also execute instructions from the 16-bit Thumb instruction set. Several performance optimizations are also added to the core [2]. An overview of the processor tile can be seen in Figure 2.7.

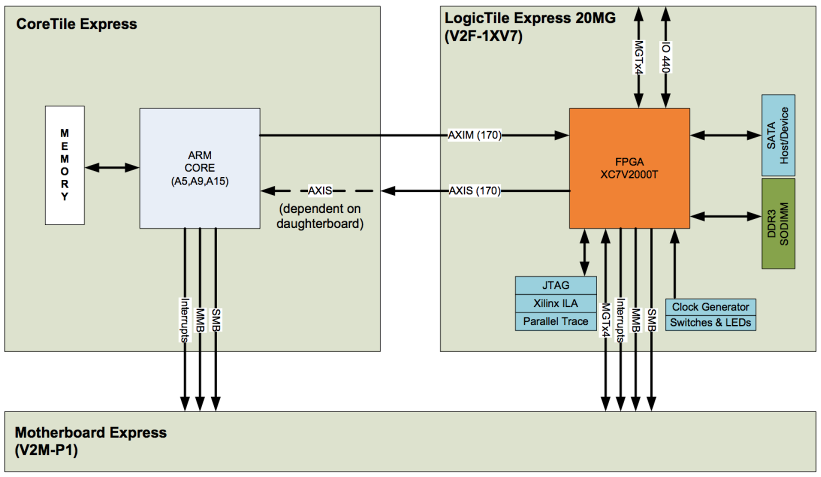


Figure 2.6: Architectural relationship between CoreTile Express and LogicTile Express [15].

Scratchpad Tile — A memory tile providing a small amount of memory for use by software. This memory is provided by the on-chip block RAM resources of the FPGA [9].

DDR Memory Tile — Memory controller tile that gives SHMAC access to off-chip DDR memory.

APB Interface (I/O tile) — This tile implements the Advanced Peripherals Bus (APB) slave which gives the host processor on the Versatile Express board access to SHMAC's memory space for use when programming the memories. [9]

Dummy Tile — Empty tile which only contains router functionality. Can be used, for instance, as a filler tile. [9]

2.5.3 SHMAC Memory Map

Figure 2.8 shows how the memory is mapped in the ARM-based SHMAC. All processor tiles share the same address space with only the tile register space being private for each tile. This memory area contains information about the tile itself, such as its coordinates, the CPU ID number and other useful data. In addition the tile register space contains the memory-mapped peripherals for each tile, such as timers and the interrupt controller. Any custom peripheral, such as the DMA and the hashing accelerator developed in the project is mapped into this address space. The system registers are used for communication with the host system.

2.5.4 SHMAC Interconnection Network

SHMAC utilizes a 2D mesh-based interconnection network to connect all the tiles, which is used to transport data packets of 128 bits length. Store-and-

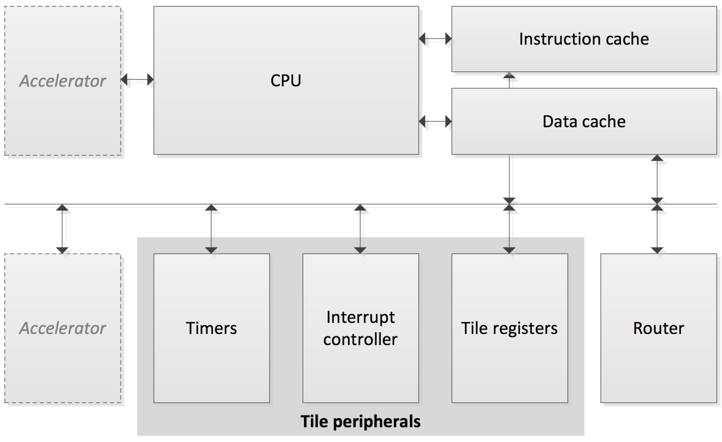


Figure 2.7: SHMAC processor tile [9].

forward switching with on/off flow control is used, and the XY-algorithm is used for deciding the route of each data packet. Each tile consists of a router with five ports, one for each of its neighbours and the local connection. When multiple packets are inbound from different directions, a round-robin scheme is used to arbitrate between which packet to route through, and packets from the local tile has equal priority to those arriving from the other directions. [9]

In the current implementation, it takes 3 cycles for a data packet to transfer from one tile to the next. The architecture of the router can be seen in Figure 2.9.

2.5.5 Wishbone Bus

Wishbone is a bus architecture developed by OpenCores to create a common interface for use between IP cores, especially in open source designs. It supports single-word transfers as well as burst transfers. A pipelined transfer mode is also provided which allows multiple requests to be sent from a module without the module having to wait for an acknowledge.

Wishbone is used as internal bus on the CPU tiles of SHMAC. Burst or pipelined transfers are not currently supported, neither internally on the tiles, or by the interconnect network, which can limit performance.

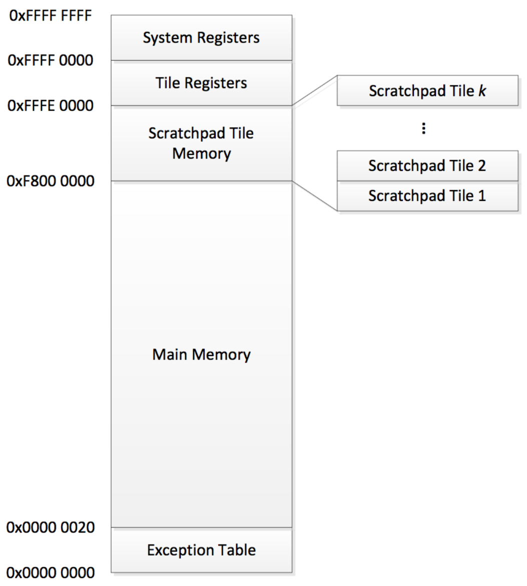


Figure 2.8: Memory map of ARM-based SHMAC, as seen in [9].

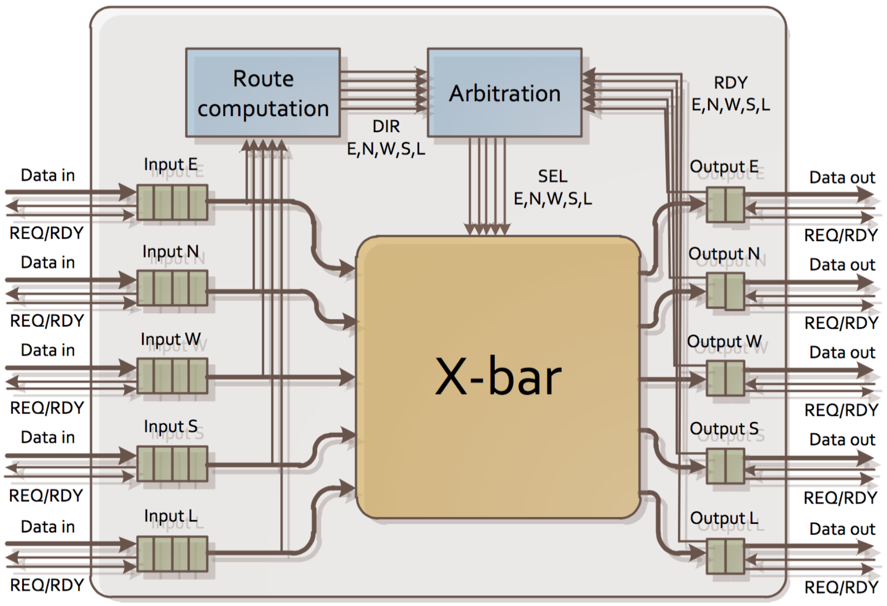


Figure 2.9: Architecture of the SHMAC router [9].

Chapter 3

Architecture

In order to develop a bitcoin mining system, a SHA-256 accelerator was developed. In addition, it was decided to design and integrate a DMA module in order to improve the throughput of the system. The new modules were then added to a CPU tile. Since it is not possible to parallelize a single run of the SHA-256 algorithm, multiple, independent tiles in a grid are used to run multiple SHA-256 computations at the same time.

3.1 Accelerated Hashing Tile

In order to test the effects of accelerating SHA-256 hashing, a new tile containing a hashing accelerator and a DMA was developed for SHMAC. A high-level overview of the new tile can be seen in Figure 3.1.

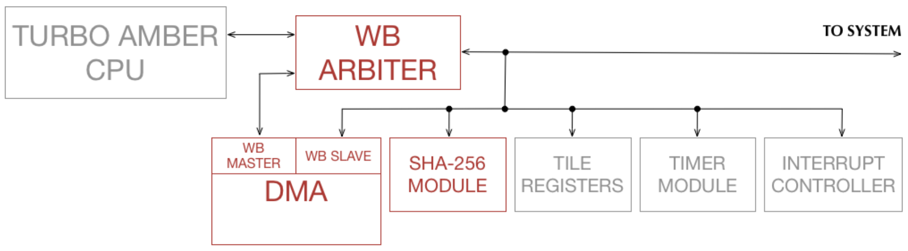


Figure 3.1: SHMAC tile with SHA-256 accelerator and DMA. Added components are highlighted in red.

The new tile is derived from the Turbo Amber tile, which contains a Turbo Amber CPU and peripherals such as an interrupt controller and timer modules, connected together with a wishbone bus. The SHA-256 accelerator and the DMA's slave interface and master interface is added to this bus.

The tile also needs an arbiter to arbitrate between the DMA master and the CPU on the wishbone bus. For this purpose, the reference arbiter from the Wishbone Public Domain Library for VHDL was adapted for use. This is a round-robin arbiter, with its function illustrated in Figure 3.2.

Round-robin arbiters work well in data acquisition systems where data is collected and placed into memory, since peripherals must often store data to

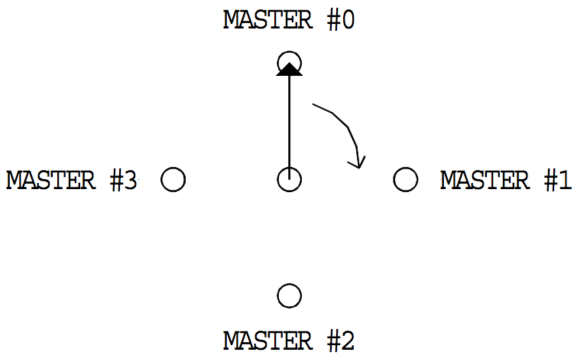


Figure 3.2: Wishbone round-robin arbiter [20].

memory. The choice of this arbiter is because using an already established wishbone arbiter saves time for this project as opposed to desiging a new one, which may end up less efficient if done poorly.

3.1.1 SHA-256 Hashing Module

The hashing module made for this project is a simple implementation of the algorithm described in Appendix A. It uses 65 cycles to compute the hash of its input data, running one iteration of the SHA-256 compression function every cycle except cycle 65, which is used to form new intermediate hash values from the results of the compression function. The algorithm is specified in big-endian format in [19], including necessary constants, so the module was designed to do the calculation in big-endian to reduce the possibility of errors. A high-level overview of the module is available in figure 3.3.

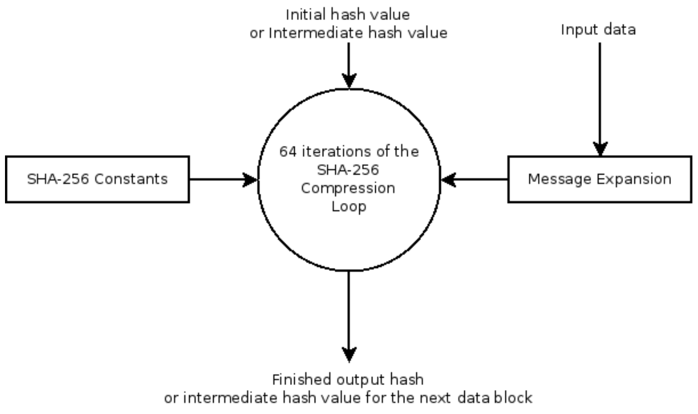

Initial hash value
or Intermediate hash value
Input data
SHA-256 Constants
64 iterations of the SHA-256 Compression Loop
Message Expansion
Finished output hash
or intermediate hash value for the next data block

Figure 3.3: High-level overview of the SHA-256 accelerator architecture.

In order for the module to remain generic, so that it can also be used in cryptography, an optimization specifically for bitcoin mining has been omitted; this means that the module does not support doing the two-pass hashing required by the bitcoin protocol, instead relying on software to set up correct input data for the second pass. It also relies on software to do the neccessary padding of the input data as required by the SHA-256 algorithm.

Even for generic SHA-256 hashing, several optimizations are possible. The SHA-256 algorithm includes 64 32-bit constants, one for each round of the compression function. These can be stored in a block RAM memory to save some logic resources [16]. However, this occupies valuable block ram resources that, in SHMAC, is used both by CPUs and scratchpad memory tiles. Thus, using block RAMs for optimization would place limits on how many tiles can be included in a SHMAC design; indeed, it was observed when synthesizing large designs with many cores that the FPGA would run out of block RAM before any other resources according to the synthesis logs.

Another optimization, that has already been mentioned in Section 2.3.2, is pipelining. This can increase the throughput to as much as one hash per cycle, but will require a large amount of additional logic because of the amount of data required by each iteration of the SHA-256 compression function. The state required would include 2048 bits of storage for the expanded message block in addition to the two sets of 8 32-bit words needed for state and the intermediate hash value respectively.

The module is controlled by the processor using a memory-mapped interface. This allows the use of a DMA to offload data transfer between memory and the hashing module. The memory-mapped interface provides registers for 512 bits of input and 256 bits of output data, in addition to control and status registers. This interface can also be used for accelerators of other cryptographic hash functions which processes input data of the same length and returns a hash of 256 bits or less, such as RIPEMD-160 or RIPEMD-256 [8] or the still popular MD5 algorithm [21]. With some work, the interface could be made even more generic in order to support algorithms with other input and output sizes; a possibility would be to eliminate the input and output registers completely in favour of using a DMA built into the module to move data of arbitrary sizes into and out of the module.

Another, alternative interface to the module that was considered was using the co-processor interface of the CPU to communicate with the module. The ARM instruction set supports up to 16 coprocessors, which can be communicated with using the `mrc` and `mcr` instructions. In such a scheme, the DMA could either transfer data to and from the accelerator directly with the coprocessor instructions, or by including a DMA in the hashing accelerator to transfer data to and from the accelerator. This would, however, preclude the DMA from being used as a general data transfer module, providing unnecessary overhead in terms of possibly unused logic.

3.1.2 DMA Module

The DMA module was designed for the purpose of investigating whether separate data transfers with DMA could improve throughput and energy efficiency of the hashing process, and also gain energy savings by freeing up the on-tile CPU for other work or sleeping. An overview of the module can be seen in figure 3.4.

The DMA module consists of the DMA logic itself, in addition to a wishbone slave interface for configuration and a wishbone master interface for transfer-

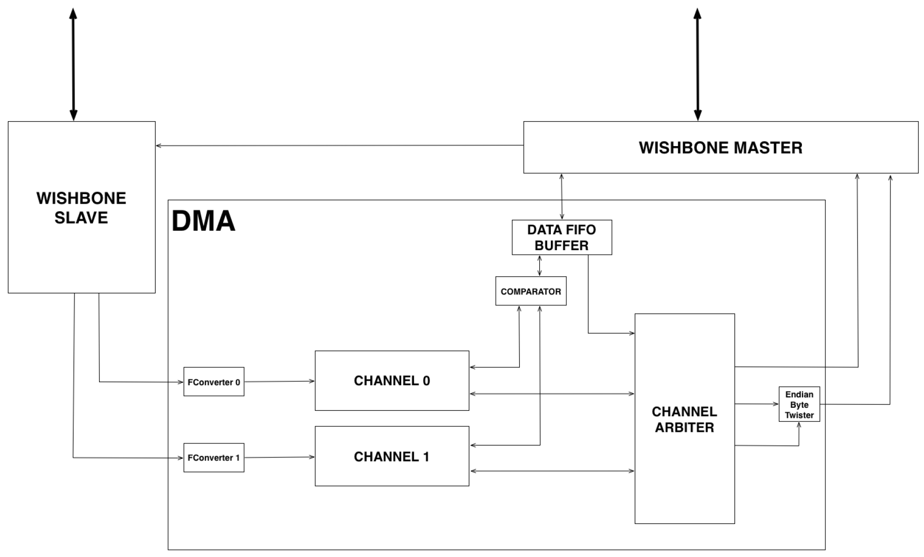


Figure 3.4: DMA overview, including wishbone interfaces.

ring data. It transfers words of 32-bit data using single-cycle wishbone transfers. In addition, the DMA supports swapping the endianness of the data it copies. This improves performance when used with the SHA-256 accelerator, because the results from the accelerator are in big-endian and must be converted to little-endian to give the correct results when used by the software running on the processor. Converting endianness in software adds several additional instructions per hash, which are now avoided. If endian swapping is active, the data bytes are swapped combinatorcially before being passed on to the wishbone master.

The wishbone slave consists of three registers for each DMA channel, used for base source address, base destination address, and details of the transfer. When request is activated, the selected channel receives data from the slave, and executes the transfer. An arbiter arbitrates between the channels if both are active. Every single command, either load or store, are passed from the channels to the wishbone master, where they are executed. Loaded data is passed on to the corresponding channel, and a channel informs the wishbone master when it is finished, so that the slave interface is informed when the final transfer is done executing. The corresponding request detail register is modified, and an interrupt request is sent to the interrupt controller.

The current DMA module only supports transfers of single 32-bit words, while the interconnect used between tiles in SHMAC supports 128-bit blocks. This means four transfers are done on the network for each 128-bit blocks, when only one entire block transfer is needed. The reason for not expanding to 128-bit blocks was due to compatibility issues with the SHA-256 module. We were concerned that expanding would force blocks to have an alignment of 128-bits, which would make the DMA harder to use from a software perspective and add additional difficulties in transferring data to and from the registers of the hashing accelerator. Otherwise they would have had to be changed so that they would be correctly aligned.

3.2 System architecture

The CPU tile with the integrated SHA-256 accelerator and DMA module was placed in a grid with other similar CPU tiles in order to exploit the thread-level parallelism offered by the SHMAC architecture. The test designs were synthesized using Xilinx' Vivado software suite, version 2013.4, and uploaded to the Versatile Express machine.

3.2.1 Initial 5x4 Grid Architecture

Initially, a 20 tile setup was used on SHMAC. The following tiles were included in the design:

- 16 CPU tiles with on-tile DMA module and SHA-256 accelerator
- 2 scratchpad tiles
- 1 DRAM tile
- 1 I/O tile

The layout is illustrated in Figure 3.5. The I/O tile is placed to the left of the first processor in the system, as only the first processor tile is used for communicating with the host system. This gives the first processor a "dedicated" connection to the I/O tile, preventing data that is sent to the host from interferring with data transfers needed for the hashing benchmark application. In addition, the rectangular grid was chosen to ensure that all cores are placed as close as possible to the memory tiles to reduce the latency of memory transfers.

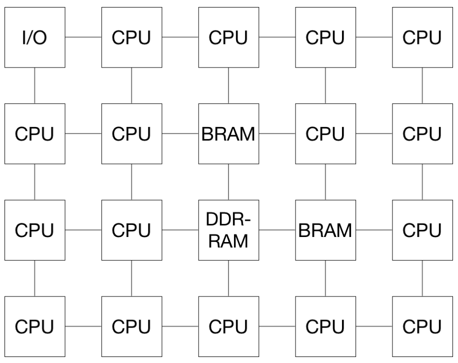


Figure 3.5: Test setup, using BRAM tiles as scratchpad and DRAM as main memory.

3.2.2 Alternative 15x2 Grid Architecture

As testing uncovered a possible hardware bug in the implementation of the scratchpad tiles, discussed in Section 4.3.1, a second design had to be created in order to measure the scaling of the performance and energy efficiency when using accelerators.

The second design places all processor tiles in a single row. This causes all traffic to the memory tiles, placed on the row below, to come from above, which bypasses the scratchpad bug. The design contains 14 CPU tiles and is illustrated in Figure 3.6. The current implementation of SHMAC does not allow more than 15 tiles on each row due to only using 4 bits to represent each grid coordinate.

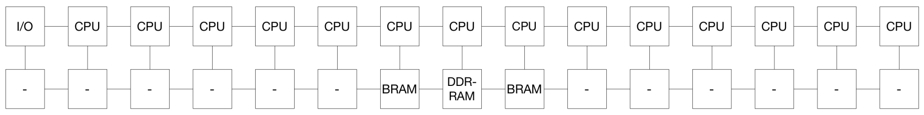


Figure 3.6: Alternative test setup.

Chapter 4

Evaluation

This chapter presents an evaluation and discussion of the performance and energy efficiency of the proposed bitcoin mining system.

4.1 Measuring Mining Performance

In order to estimate the bitcoin mining performance of the system, a benchmark is used to measure the number of double hashes that can be performed each second. This is done by repeatedly double-hashing a block of data and recording the number of hashes achieved per second per core. By adding more cores over time, it is possible to see how the performance scales as more cores are added. Recording the number of hashes per second per core also makes it possible to see how the performance of each core is affected by the traffic on the network-on-chip generated by the other cores in the network. Cores not active are put into a no-operation loop.

Using the architecture setup described in Section 3.2, tests where run using the following method:

1. Hashing using software only, not using the SHA-256 hashing module or DMA module. All work is done by the on-tile processor.

2. Hashing using only the hashing accelerator. The processor controls and copies data to and from the hashing module, and does the endian conversion in software.

3. Hashing using the hashing accelerator and DMA. The processor controls each module, but the DMA handles data transfer, as well as endian conversion.

Interrupts are used by the hashing module to signal when it is finished working. However, the software does not support interrupt from DMA, and it has to be polled while copying data to and from the module, which may have a minor influence on the results.

The benchmark software was compiled using the GCC compiler, version 4.8.2, compiled for the arm-none-eabi target triplet. To generate a binary file for the SHMAC, utilities from GNU Binutils version 2.24, also compiled for the arm-none-eabi target triplet, were used.

4.2 Estimating Power Usage

Power usage for the application is determined by measuring the wall-power of the box that SHMAC is running on both when idle and when running the test applications. The power usage of the application is then determined using the following formula, where P is the power in watts:

$$P_{application} = P_{running} - P_{idle}$$

Using the result of the power measurement, the energy efficiency can be obtained as the number of hashes per second per watt, H/s/W. As watt is defined as joule per second, J/s, the unit can be rewritten to hashes per joule, H/J.

To obtain the power usage of SHMAC, a Yokogawa WT210 power meter was used to measure the power drawn by the Versatile Express box from the power socket in the wall while idle and when running the bitcoin application. This method provides some uncertainty in the result of the measurements, as the Versatile Express also runs a Linux host system. It was not possible to obtain power measurements directly from the FPGA running SHMAC.

4.3 Performance Results

To establish a baseline for the performance gains obtained when using the accelerators, a measurement of the performance when using software hashing was first obtained. The performance is measured in the number of double hashes per second and abbreviated H/s, as is the convention for bitcoin mining systems.

4.3.1 Initial Results

Using software-only hashing produced a best result of 21 261 H/s when running with only 4 active processors. As can be seen in the plot in Figure 4.1, it can be observed that adding more processors after the fourth produces no noticeable additional performance gain. The reason for this is that all cores makes frequent accesses to DRAM, when running the software algorithm, which causes the DRAM tile to quickly become congested. The reason for these frequent accesses is probably due to the use of many variables in the code as well as the effect of stack usage. Since the Turbo Amber core uses a write-through cache, all memory writes ends up going to DRAM immediately which causes additional congestion.

Another interesting effect to note in the results is how the XY routing affects the performance of each tile. The more tiles that tries to access main memory through a tile's router, the less performance that particular tile has; network congestion does, in other words, have a great impact on individual tile performance. This was especially obvious for processor tile 10, which is located on the end of row 3 with no processors on either its left or right sides. This tile showed notably better performance than any other processor tiles in the grid. This is because its location means that no data to or from other tiles have to pass through its router, which gives it more time to process its own requests. In addition, this tile can access all memory tiles without having to send or receive

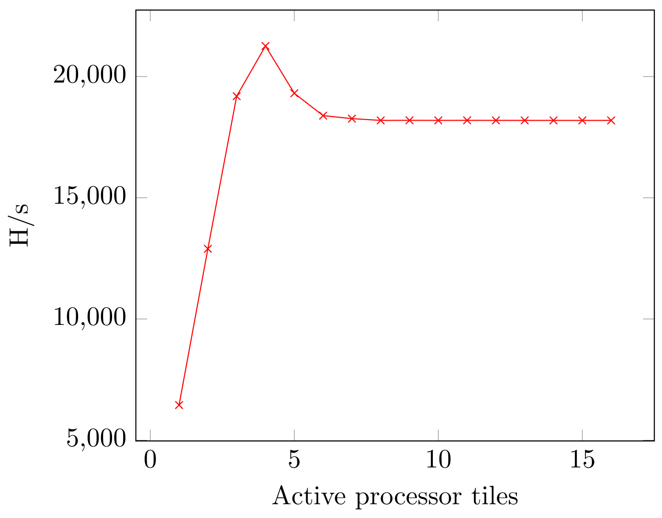


Figure 4.1: Software hashing performance

data through any other processor tiles. The individual performance of each processor is noted in Appendix B, Table B.1.

Figure 4.2 shows the results when using the SHA-256 accelerator, without and with the DMA module enabled, respectively. Looking at the case where only one core is running, one can see that the performance when using the accelerator and DMA is about 2,8 times faster than using the software version, at 18 230 H/s in hardware and 6 450 H/s in software. Adding more processors gives an even higher gain due to the near linear scaling of the performance when using the hardware accelerators.

Only up to four processors where used for this test, as adding in processors from the second row of processors when scaling up caused the application to crash. An attempt at starting processors from the top and bottom row at the same time also led to a crash. It was discovered that this was likely because of an bug in the implementation of the scratchpad memory tile used. An attempt was made at integrating fixes to the scratchpad bug from an experimental SHMAC branch, but failed due to differences in the code. Because of this bug, it is not possible to predict how many cores can be active and hashing using hardware acceleration at the same time before reaching the memory congestion limit, nor is it possible to see if the DMA has any significant effect on the performance. It was decided to attempt to work around the scratchpad bug with a new design, in order to better measure how the performance and energy efficiency of hardware hashing scales and what effect using a DMA will have when more tiles are active.

4.3.2 Results from the Alternative Design

In order to obtain better results with regards to scaling, a new test design was created which works around the scratchpad tile bug and places 14 CPU cores on the same row in the grid. The design is described in Section 3.2. The performance obtained using software-only hashing is plotted in Figure 4.3 and mirrors the results obtained from the original design, in that adding more cores after the

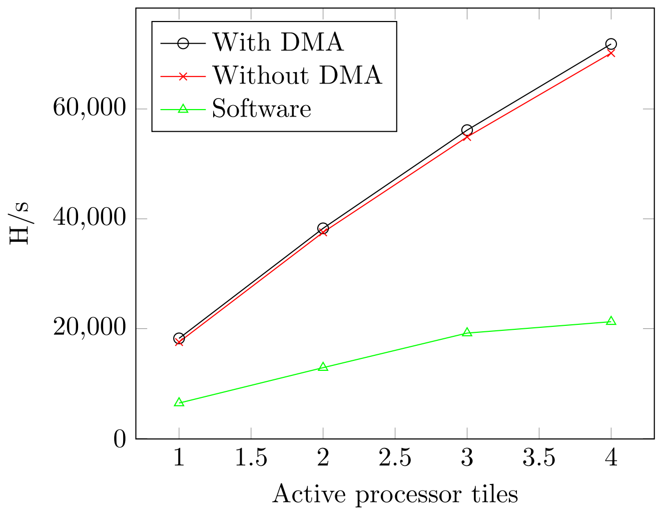


Figure 4.2: Hardware hashing performance, with software performance included for comparison

fourth does not provide any additional performance. The greatest performance obtained using software hashing in the alternative design was 21 364 H/s with 4 cores active.

The performance results obtained using the hardware modules are plotted in Figure 4.4. From these results, it can be observed that the performance continues to scale almost linearly up until around the tenth core is added. At this point, memory congestion is starting to affect the scaling, causing it to flatten out. The best performance was obtained when running with all cores active, at 175 711 H/s with DMA enabled or 179 526 H/s with the DMA disabled. Even though this case did not show any advantage of using the DMA, using the DMA for data transfer *does* provide a small and consistent performance increase when using less than 12 cores.

The reason for so little improvement is likely due to difference in traffic on the interconnect network. When the CPU is loading data into the hashing accelerator, it first loads data from main memory into its data cache. This is done using 128-bit block transfers, transferring an entire cache line at a time from the memory. The DMA only use 32-bit transfers, causing four times more traffic in the tile network when loading data, as every 128-bit block is accessed four times. When it comes to storing, the caches use write-through as strategy, and for every 32-bit word updated in a 128-bit block, the cache will write back to the memory. In this case, both the CPU and the DMA will cause the same traffic for each store operation. With the CPU's load traffic being 25% compared to using the DMA, and the store traffic 100%, using the CPU for data transfers should cause an average of 67.5% data traffic, compared to using the DMA module.

Still, a slight performance increase is observed while using the DMA module, before reaching 12 tiles. It is likely due to the DMA module relieving the CPU of the endian conversion, as this is done instantly during DMA transfer, if enabled. Without the DMA, the CPU must do this over several cycles. The CPU

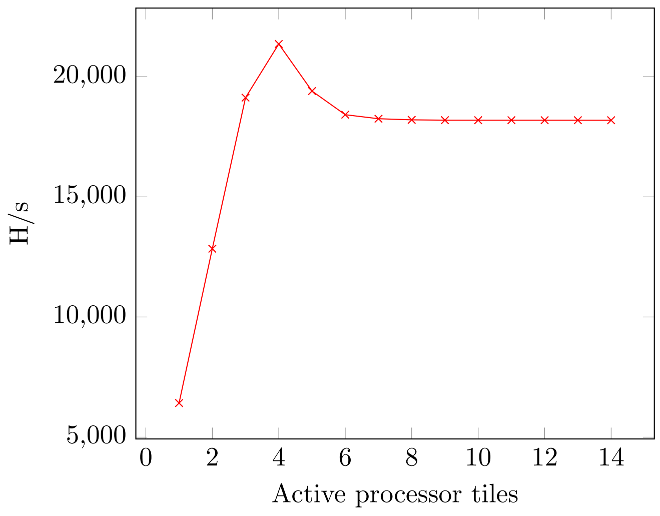


Figure 4.3: Software hashing performance for the alternative architecture design

likely also has more overhead when copying the data to or from the accelerator compared to the DMA, although it is hard to predict how much the DMA improves on this when the CPU is supported by its cache, that loads a full 128-bit cache line per load from memory.

4.4 Power and energy efficiency

The energy efficiency was calculated by measuring the power usage of the application, as described in Section 4.2, and looking at how many double hashes the system does. This provides a number of double hashes per second per watt.

Obtaining the power measurements provided a difficult task, as the idle power of the Versatile Express box seemed to be steadily rising over time. This can be caused by multiple factors, such as the host system working, changes in temperature due to the benchmark application running or environmental factors. Because of this, the idle power was measured between each individual test run to make sure the power usage of the application was measured as correctly as possible. Because of the problems encountered with the initial test design, only the power results from the alternative design is discussed here. However, both designs show the same trends, and the results from the initial design can be seen in the appendix, Section B.2.1.

Figure 4.5 shows the measured power usage when running with software hashing, accelerated hashing without DMA and accelerated hashing with DMA. For software-only hashing, it is interesting to note that less power is used when adding more cores. With 1–3 cores active, the power usage for the application was measured to be between 2,3 W and 2,5 W, while running at 5 cores and more only used between 0,7 W and 0,8 W. One reason for this may be that as more tiles are added, longer periods of stalling is needed for each core to wait for data from the main memory. When cores are inactive, they are waiting in a nop-operation loop, a software loop consisting of a NOP operation. Executing

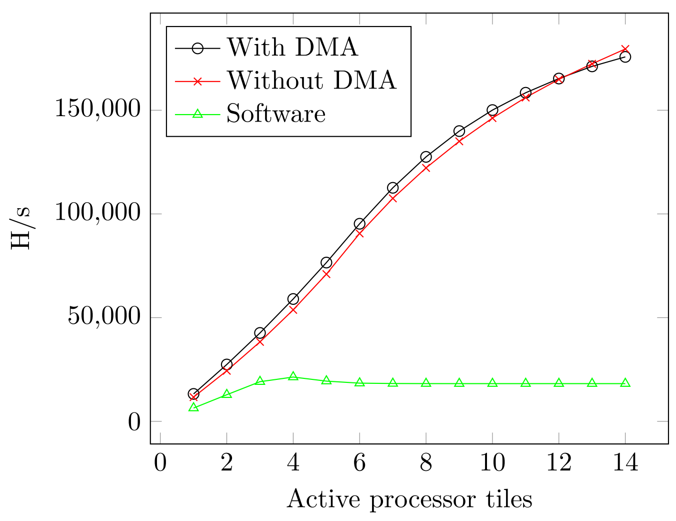


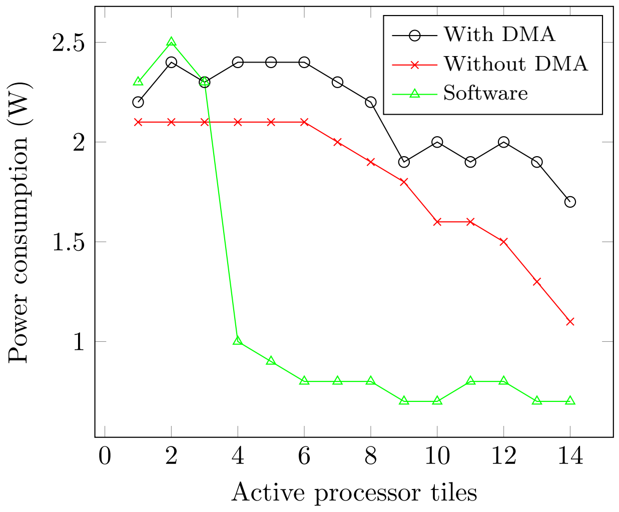
Figure 4.4: Hardware hashing performance for the alternative architecture with software performance included for comparison.

this loop causes the processor to decode and execute two instructions, the no-operation instruction and the branch instruction. However, when stalling, the processor does not decode or execute new instructions, which may cause less transistor switching to happen internally in the processor, thus reducing power wasted from the switching. This could be a good indicator that the Turbo Amber processor could benefit from a form of sleep or wait-for-interrupt instruction, which is not currently supported by the processor [22], which will allow it to stall as long as it has no work to do.

The same trend can be seen when using hardware, although the drop in power usage is not as pronounced as in the software-only case. This is likely because it takes longer for the traffic to the memories to be large enough to cause longer periods of stalling in the processor and DMA module.

The energy efficiency is shown in Figure 4.6. It is evident that using a hardware accelerated hashing module provides large gains in energy efficiency, which becomes especially clear when adding more than 4 cores, because of the much higher performance the accelerators can provide with little overhead in energy use. At 14 active tiles, hardware acceleration without DMA provides over 6 times better energy efficiency than software hashing, at 163 205,5 H/J versus 25 985,5 H/J. It is interesting to note that the energy efficiency when using the DMA module becomes notably worse compared to when it is disabled, when the interconnect network starts to become congested, around the point where 10 cores or more are active.

The numbers obtained here shows that our accelerated hashing tile cannot compete with other FPGA-based bitcoin hashing systems, such as BTCMiner, mentioned in Section 2.3.2, which gets a result of 21 MH/J, 128 times better than our 163,3 kH/J. There are several reasons for this, mainly the lack of pipelining in the SHA-256 accelerator, which slows throughput by 65 times compared to a fully unrolled SHA-256 pipeline, and a memory system that is not optimized for the high throughput required for bitcoin mining.



htb

Figure 4.5: Measured power consumption using software and hardware accelerators.

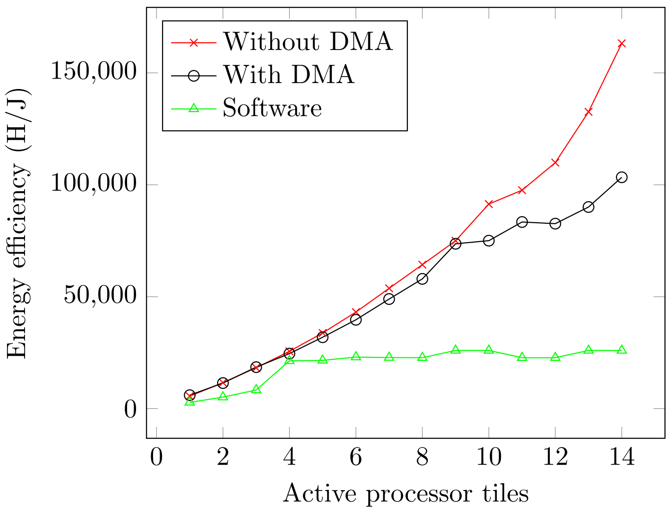


Figure 4.6: Energy efficiency

Chapter 5

Conclusion

The results obtained with the benchmark shows that using the SHA-256 accelerator and DMA module described in this report, it is possible to get a bitcoin mining performance of 175,7 kH/s when running the hashing benchmark on 14 cores simultaneously while using the DMA and a performance of 179,5 kH/s when not using the DMA. The best energy efficiency is obtained when running the benchmark on 14 cores *without* the DMA enabled, at 163,2 kH/J.

Although the performance and energy efficiency obtained in this study cannot compete with other bitcoin mining solutions implemented on FPGAs, providing 128 times lower energy efficiency than a dedicated bitcoin mining system, it still shows that SHMAC is an excellent platform for exploiting thread-level parallelism, and using our hardware accelerators provided a near-linear performance scaling over 14 CPU cores while retaining good energy efficiency.

With respect to bitcoin mining, the days of FPGAs are gone, and bitcoin mining is now the domain of highly optimized ASICs. However, although SHMAC cannot be used for bitcoin mining, the high degree of thread-level parallelism provided by the platform can be well exploited in other application areas benefitting from high levels of parallelism.

5.1 Future Work

It is possible to improve the results obtained in this thesis further by adding additional optimizations to the accelerators.

5.1.1 Enhancing the SHA-256 Module

The SHA-256 module can obtain performance increases from pipelining. Although this uses more resources on the FPGA chip as well as requiring additional memory transfers from a CPU or a DMA to keep it fed with data, it can increase performance and energy efficiency depending on the application.

5.1.2 Enhancing the DMA Module

The current DMA Module only supports transfers of single 32-bit words, while the interconnect used in SHMAC supports 128-bit words. This means four transfers are done on the network when only one is needed. Exploiting the unused bandwidth could provide a speedup of as much as 75 % and reduce the workload on the DMA significantly. While expanding the DMA transfer width, it must still be made compatible with the other modules on the tile, including the SHA-256 accelerator. This likely requires a solution where individual 32-bit words may be selected and written individually.

The DMA could also be expanded to include support for wishbone burst transfers. This can allow quicker data transfer, as overhead is reduced, and potentially improving throughput.

Bibliography

[1] Bitcoin charts. https://blockchain.info/charts. Accessed May 7th, 2015.

[2] Anders Tvetmarken Akre and Sebastian Bøe. Turbo amber - a high-performance processor core for SHMAC. June 2014.

[3] J. Barkatullah and T. Hanke. Goldstrike 1: Cointerra's first-generation cryptocurrency mining processor for bitcoin. *Micro, IEEE*, 35(2):68–76, Mar 2015.

[4] The Bitcoin Community. Btcminer. https://en.bitcoin.it/wiki/BTCMiner. Accessed June 6th, 2015.

[5] The Bitcoin Community. History. https://en.bitcoin.it/wiki/History. Accessed May 20th, 2015.

[6] The Bitcoin Community. Non-specialized hardware comparison. https://en.bitcoin.it/wiki/Non-specialized_hardware_comparison. Accessed May 7th, 2015.

[7] The Bitcoin Community. When was the first ASIC miner released? https://bitcointalk.org/index.php?topic=382895.0. Accessed May 20th, 2015.

[8] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. RIPEMD-160: A strengthened version of RIPEMD. In Dieter Gollmann, editor, *Fast Software Encryption*, volume 1039 of *Lecture Notes in Computer Science*, pages 71–82. Springer Berlin Heidelberg, 1996.

[9] EECS. Single-ISA heterogeneous many-core computer (SHMAC) project plan, March 2014.

[10] John L. Hennesy and David A. Patterson. *Computer Architecture, A Quantitative Approach.* Elsevier, Inc, 5th edition edition, 2012.

[11] R. Kumar, K.I. Farkas, N.P. Jouppi, P. Ranganathan, and D.M. Tullsen. Single-ISA heterogeneous multi-core architectures: the potential for processor power reduction. In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pages 81–92, Dec 2003.

[12] R. Kumar, D.M. Tullsen, and N.P Jouppi. Core architecture optimization for heterogenous chip multiprocessors, September 2006.

[13] R. Kumar, D.M. Tullsen, P. Ranganathan, N.P. Jouppi, and K.I. Farkas. Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. In *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*, pages 64–75, June 2004.

[14] ARM Limited. ARM versatile express product family.

[15] ARM Limited. LogicTile express 20MG datasheet.

[16] R.P. McEvoy, F.M. Crowe, C.C. Murphy, and W.P. Marnane. Optimisation of the SHA-2 family of hash functions on FPGAs. In *Emerging VLSI Technologies and Architectures, 2006. IEEE Computer Society Annual Symposium on*, March 2006.

[17] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `https://bitcoin.org/bitcoin.pdf`, October 2008. Accessed December 7th, 2014.

[18] National Institute of Standards and Technology. Descriptions of SHA-256, SHA-384, and SHA-512.

[19] National Institute of Standards and Technology. Secure hash standard (SHS). *Federal Information Processing Standards Publication*, March 2012.

[20] Wade D. Peterson. *Technical Manual: WISHBONE Public Domain Library for VHDL*. OpenCores, October 2001.

[21] Ronald Rivest. The MD5 message-digest algorithm. RFC 1321, RFC Editor, April 1992.

[22] Conor Santifort. "amber open source project". March 2015.

[23] Kristian Klomsten Skordal and Torbjørn Langland. Evaluating possibilities for bitcoin mining with SHMAC. December 2014.

[24] Michael Bedford Taylor. Bitcoin and the age of bespoke silicon. In *Proceedings of the 2013 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, CASES '13, pages 16:1–16:10, Piscataway, NJ, USA, 2013. IEEE Press.

[25] Michael Bedford Taylor. A landscape of the new dark silicon design regime. *IEEE MICRO*, September 2013.

Appendix A

The SHA-256 Algorithm

The SHA-256 algorithm is a member of a set of algorithms referred to as the SHA-2 standard. These are described in [19] and consists of algorithms for producing hashes with lengths of 224, 256, 384 and 512 bits. The algorithms use simple operations, limited to shifts, rotates, xor, and unsigned additions, common single-cycle operations for general purpose CPUs, in addition to a lookup-table of constants. This allows for high-speed implementations in both software and hardware. The different SHA-2 algorithms differ in how and with what parameters the various operations are invoked.

SHA-256 is the algorithm used in cryptocoin mining. It operates on blocks of 512 bits and keeps a 256 bits long intermediate hash value as state. Bitcoin uses a double pass SHA-256 hash, which first calculates the hash of a block of the data to be hashed and then hashes the hash of the first pass.

Before the first block is processed, the initial hash value is set to a predefined value. The entire message that is to be hashed is then padded by adding a 1 bit to the end of the message and then appending zeroes until the length of the final block is 448 bits. Then the length of the entire message, without padding, is added as a 64-bit big-endian integer to the end of the block.

Then, each input block is split into a 64 times 32-bit long expanded message block, where each 32-bit word W_j is defined according to the formula

$$W_j = \begin{cases} M_j & j \in [0, 15] \\ \sigma_1(W_{j-2}) + W_{j-7} + \sigma_0(W - j - 15) + W_{j-15} & j \in [16, 63] \end{cases}$$

where M_j is the jth word of the input message block and the functions σ_0 and σ_1 are defined as

$$\sigma_0 = R^7(x) \oplus R^{18}(x) \oplus S^3(x)$$
$$\sigma_1 = R^{17}(x) \oplus R^{19}(x) \oplus S^{10}(x)$$

where the operator R^n means right rotation by n bits and S^n means right shift by n bits [1].

[1]Curiously, [18] defines the operator R as shift and S as rotate. We use the more intuitive definitions.

The Compression Function

The compression function is the core of the SHA-256 algorithm. It uses a look-up table of 64 constants, K_j, and the following functions when calculating the new intermediate hash values:

$$\begin{aligned}
Ch(x, y, z) &= (x \wedge y) \oplus (\neg x \wedge z) \\
Maj(x, y, z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \\
\Sigma_0(x) &= R^2(x) \oplus R^{13}(x) \oplus R^{22}(x) \\
\Sigma_1(x) &= R^6(x) \oplus R^{11}(x) \oplus R^{25}(x)
\end{aligned}$$

Before starting the iterations with the compression function, the intermediate hash values from the previous message block are assigned to the variables a–h.

At the beginning of each iteration of the compression function, two temporary values are calculated:

$$\begin{aligned}
T_1 &= h + \Sigma_1(e) + Ch(e, f, g) + K_j + W_j \\
T_2 &= \Sigma_0(a) + Maj(a, b, c)
\end{aligned}$$

The new hash values are then assigned as follows:

$$\begin{aligned}
&h \leftarrow g \\
&g \leftarrow f \\
&f \leftarrow e \\
&e \leftarrow d + T_1 \\
&d \leftarrow c \\
&c \leftarrow b \\
&b \leftarrow a \\
&a \leftarrow T_1 + T_2
\end{aligned}$$

The compression function is run 64 times, once for each word in the extended message block, W_j. Afterwards, the intermediate hash for the message is updated by adding the variables a–h to the corresponding values of the intermediate hash values from the previous message block.

When the final input block has been processed, the final hash is composed by concatenating the intermediate hash values. [18, 19, 23]

Appendix B

Additional Results

This chapter lists up all the details regarding the various measurements done during this thesis. For each selected SHMAC architecture, the performance and power usage is measured for varying numbers of active cores. Every architecture has been measured when running software-only hashing, when running the SHA-256 accelerator without using the DMA module, and then with using the DMA module.

B.1 Performance Measurements

B.1.1 Initial Architecture

The first architecture used for measuring, was the 5 by 4 grid architecture described in section 3.2. Unfortunately, while software hashing worked on all tiles, the application crashed when using the SHA-256 accelerator on more than one row of processors and the design was therefore discarded. The reasons are discussed in section 4.3.1.

The performance obtained when doing software hashing is listed in table B.1.

Tables B.2 and B.3 show the results when using the SHA-256 hashing accelerator, without and with DMA module, respectively. Only four cores were used because of the scratchpad bug mentioned in section 4.3.1.

B.1.2 Alternative Architecture

The results from doing the software hashing for this architecture is listed in table B.4. The CPU tiles nearest the DRAM tile have highest individual performance compared to others present.

Tables B.5 and B.6 show the results when using the SHA-256 accelerator in the alternative architecture, without and with DMA module, respectively. This time, using an architecture that circumvents the cache-bug problem, all available CPUs are used in the tests.

Sum [H/s]	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
6450	6450	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
12895	6446	6449	-	-	-	-	-	-	-	-	-	-	-	-	-	-
19190	6394	6398	6398	-	-	-	-	-	-	-	-	-	-	-	-	-
21261	5312	5316	5316	5317	-	-	-	-	-	-	-	-	-	-	-	-
19310	4085	4029	3191	3207	4798	-	-	-	-	-	-	-	-	-	-	-
18388	3171	3154	2083	2090	3961	3929	-	-	-	-	-	-	-	-	-	-
18265	2354	2349	1474	1474	3119	3120	4375	-	-	-	-	-	-	-	-	-
18191	1964	1970	1143	1142	2993	2993	2993	2993	-	-	-	-	-	-	-	-
18195	1455	1458	801	801	2239	2239	2240	2240	4722	-	-	-	-	-	-	-
18192	1142	1146	630	629	1762	1762	1762	1762	3799	3798	-	-	-	-	-	-
18194	820	823	440	439	1257	1257	1257	1256	3120	3119	4406	-	-	-	-	-
18192	579	582	309	308	888	888	889	889	2572	2572	3887	3829	-	-	-	-
18192	531	534	278	278	811	811	811	812	2417	2417	3745	2374	2373	-	-	-
18191	528	531	278	279	809	809	808	809	2411	2411	3737	1324	1324	2133	-	-
18191	528	531	276	277	807	807	807	806	2406	2406	3730	875	875	1541	1519	-
18191	528	530	276	276	806	805	806	805	2402	2402	3723	827	828	1490	844	843

Table B.1: Software performance, for an increasing number of cores active.

Sum [H/s]	**0**	**1**	**2**	**3**
17557	17557	-	-	-
37539	17528	20011	-	-
54912	17481	19941	17490	-
70179	17400	19891	17411	15477

Table B.2: Performance when using the SHA-256 accelerator only in the initial architecture.

Sum [H/s]	**0**	**1**	**2**	**3**
18230	18230	-	-	-
38267	18189	20078	-	-
56150	18129	19880	18141	-
71840	17973	19577	17984	16306

Table B.3: Performance when using the SHA-256 accelerator and DMA in the initial architecture.

B.2 Power measurements

B.2.1 Energy Efficiency for the Initial Test Setup

Table B.7 shows the measured power usage, along with uncertainty and calculated application power of the initial test architecture. The data is plotted in figure B.1.

Tables B.8 and B.9 shows the individual power usage measured when using the SHA-256 accelerator without and with DMA module, respectively. Uncertainty and calculated application power is included as well. The data are plotted in figures B.2. As can be seen from the tables and figure, there is no significant difference in power usage when running with and without the DMA.

B.2.2 Alternative Architecture

Table B.10 shows the power measurements for the alternative architecture.

Tables B.11 and B.12 shows the individual power usage when using the SHA-256 accelerator without and with DMA module, respectively. Uncertainty and calculated application power are included as well.

B.3 Energy Efficiency

In this chapter, the energy efficiency is presented, calculated from the measured performance and energy usage.

B.3.1 Initial Architecture

The energy efficiency of software hashing in the initial architecture can be seen in table B.13, and the corresponding plot in figure B.3.

Tables B.14 and B.15 shows the energy efficiency of the SHA-256 accelerator, without and with using the DMA module, respectively. The plot for both can be seen in figure B.4, where they are compared to each other.

Sum [H/s]	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
6418	6418	-	-	-	-	-	-	-	-	-	-	-	-	-
12841	6419	6422	-	-	-	-	-	-	-	-	-	-	-	-
19130	6374	6378	6378	-	-	-	-	-	-	-	-	-	-	-
21364	5338	5342	5342	5342	-	-	-	-	-	-	-	-	-	-
19404	3186	3158	3883	4471	4706	-	-	-	-	-	-	-	-	-
18420	1723	1718	2538	3506	4313	4622	-	-	-	-	-	-	-	-
18253	945	944	1549	2410	3472	4304	4629	-	-	-	-	-	-	-
18206	516	519	902	1532	2442	3505	4358	4432	-	-	-	-	-	-
18193	345	348	633	1139	1958	3102	4257	3240	3171	-	-	-	-	-
18192	313	315	582	1057	1834	2997	4231	3017	1969	1877	-	-	-	-
18192	311	312	573	1040	1821	2979	4221	2989	1823	1096	1027	-	-	-
18192	308	310	570	1032	1818	2970	4217	2980	1819	1036	587	545	-	-
18191	307	309	571	1032	1811	2965	4216	2974	1810	1035	565	305	291	-
18190	307	310	570	1030	1809	2962	4211	2971	1809	1034	564	298	158	157

Table B.4: Software performance results for the alternative architecture.

Sum [H/s]	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
11676	11676	-	-	-	-	-	-	-	-	-	-	-	-	-
24394	11664	12730	-	-	-	-	-	-	-	-	-	-	-	-
38329	11642	12699	13988	-	-	-	-	-	-	-	-	-	-	-
53760	11618	12660	13951	15531	-	-	-	-	-	-	-	-	-	-
70983	11563	12600	13872	15463	17485	-	-	-	-	-	-	-	-	-
90590	11503	12533	13800	15371	17384	19999	-	-	-	-	-	-	-	-
107510	11438	12454	13711	15270	17265	19897	17475	-	-	-	-	-	-	-
122186	11366	12372	13621	15163	17136	19779	17339	15410	-	-	-	-	-	-
135009	11295	12299	13527	15046	17001	19663	17195	15261	13722	-	-	-	-	-
146221	11222	12212	13423	14918	16855	19546	17029	15104	13574	12338	-	-	-	-
156133	11140	12123	13320	14794	16722	19431	16852	14931	13418	12200	11202	-	-	-
164860	11068	12038	13225	14675	16585	19319	16653	14741	13234	12041	11056	10225	-	-
172381	10988	11943	13109	14541	16422	19208	16422	14538	13028	11844	10882	10070	9386	-
179526	10944	11891	13045	14461	16329	19123	16306	14347	12839	11669	10721	9925	9253	8673

Table B.5: Performance results when using the SHA-256 accelerator in the alternative architecture.

Sum [H/s]	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
13227	13227	-	-	-	-	-	-	-	-	-	-	-	-	-
27410	13220	14190	-	-	-	-	-	-	-	-	-	-	-	-
42627	13183	14157	15287	-	-	-	-	-	-	-	-	-	-	-
58989	13124	14091	15218	16556	-	-	-	-	-	-	-	-	-	-
76556	13027	13991	15102	16431	18005	-	-	-	-	-	-	-	-	-
95259	12869	13828	14920	16213	17769	19660	-	-	-	-	-	-	-	-
112599	12695	13621	14691	15961	17495	19307	18829	-	-	-	-	-	-	-
127515	12487	13393	14443	15689	17145	18885	18568	16905	-	-	-	-	-	-
139937	12231	13085	14115	15326	16762	18411	18225	16597	15185	-	-	-	-	-
150069	11911	12749	13783	14959	16333	17862	17749	16176	14867	13680	-	-	-	-
158421	11585	12456	13402	14560	15971	17278	17236	15782	14441	13323	12387	-	-	-
165271	11291	12094	13039	14193	15685	16642	16627	15491	14055	12927	12007	11220	-	-
171143	10967	11768	12720	13935	15445	15971	15965	15290	13829	12619	11679	10900	10055	-
175711	10625	11447	12461	13803	15040	15282	15274	14942	13676	12374	11375	10559	9726	9127

Table B.6: Performance results when using the SHA-256 accelerator with DMA in the alternative architecture.

Cores	**Average System Power [W]**	**Uncertainty**	**Application Power [W]**
Idle	34.3	0.4	0.0
1	36.6	0.4	2.3
2	36.7	0.3	2.4
3	36.7	0.4	2.4
4	35.3	0.4	1.0
5	35.0	0.2	0.7
6	35.1	0.3	0.8
7	35.0	0.2	0.7
8	34.9	0.3	0.6
9	34.9	0.2	0.6
10	34.9	0.3	0.6
11	34.9	0.3	0.6
12	34.8	0.2	0.5
13	34.9	0.3	0.6
14	34.9	0.4	0.6
15	35.0	0.4	0.7
16	34.8	0.2	0.5

Table B.7: Measured power usage for software hashing using the initial test architecture.

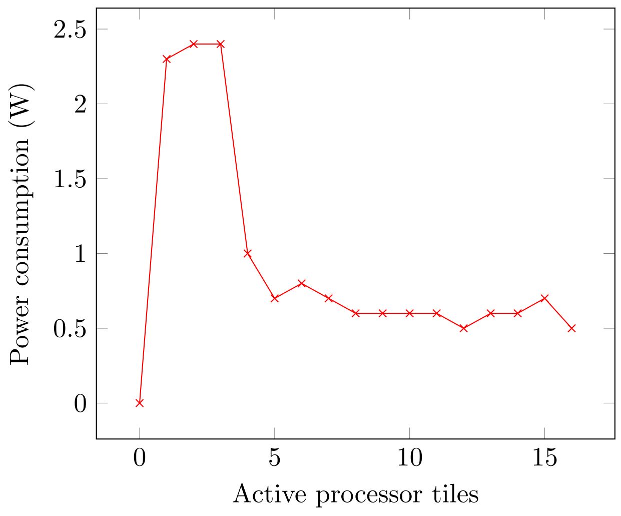


Figure B.1: Measured power consumption for software hashing using the initial test architecture.

Cores	**Average System Power [W]**	**Uncertainty**	**Application Power [W]**
Idle	34.3	0.4	0.0
1	36.5	0.5	2.2
2	36.5	0.5	2.2
3	36.3	0.4	2.0
4	36.4	0.4	2.1

Table B.8: Power usage using the SHA-256 accelerator using the initial test architecture.

Cores	**Average System Power [W]**	**Uncertainty**	**Application Power [W]**
Idle	34.3	0.4	0.0
1	36.4	0.4	2.1
2	36.7	0.3	2.3
3	36.6	0.4	2.2
4	36.6	0.4	2.2

Table B.9: Power usage using the SHA-256 accelerator and DMA using the initial test architecture.

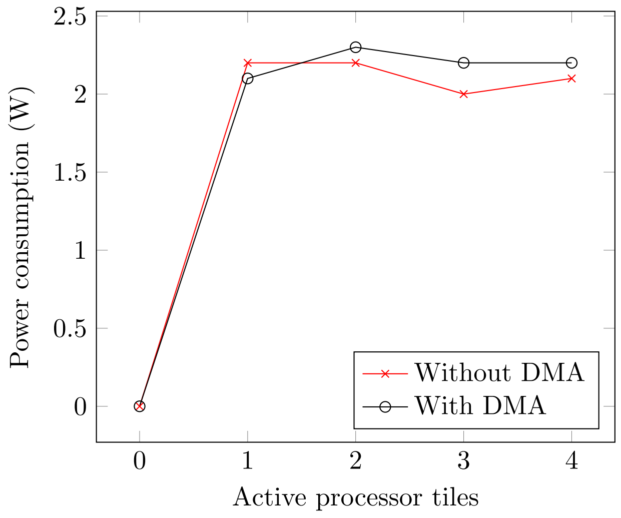


Figure B.2: Calculated power consumption using hardware accelerators

Cores	**Average System Power [W]**	**Uncertainty**	**Application Power [W]**
Idle	33.9	0.3	0.0
1	36.2	0.3	2.3
2	36.4	0.4	2.5
3	36.2	0.4	2.3
4	34.9	0.3	1.0
5	34.8	0.2	0.9
6	34.7	0.3	0.8
7	34.7	0.3	0.8
8	34.7	0.3	0.8
9	34.6	0.4	0.7
10	34.6	0.4	0.7
11	34.7	0.3	0.8
12	34.7	0.3	0.8
13	34.6	0.4	0.7
14	34.6	0.4	0.7

Table B.10: Measured power usage when doing software hashing in the alternative architecture.

Cores	Average System Power [W]	Uncertainty	Application Power [W]
Idle	33.8	0.3	0.0
1	35.9	0.3	2.1
2	35.9	0.3	2.1
3	35.9	0.3	2.1
4	35.9	0.3	2.1
5	35.9	0.3	2.1
6	35.9	0.3	2.1
7	35.8	0.3	2.0
8	35.7	0.4	1.9
9	35.6	0.4	1.8
10	35.4	0.4	1.6
11	35.4	0.3	1.6
12	35.3	0.4	1.5
13	35.1	0.3	1.3
14	34.9	0.3	1.1

Table B.11: Measured power usage when using the SHA-256 accelerator without the DMA in the alternative architecture.

Cores	Average System Power [W]	Uncertainty	Application Power [W]
Idle	33.8	0.3	0.0
1	36.0	0.3	2.2
2	36.2	0.3	2.4
3	36.1	0.4	2.3
4	36.2	0.3	2.4
5	36.2	0.3	2.4
6	36.2	0.3	2.4
7	36.1	0.3	2.3
8	36.0	0.3	2.2
9	35.7	0.3	1.9
10	35.8	0.3	2.0
11	35.7	0.3	1.9
12	35.8	0.2	2.0
13	35.7	0.4	1.9
14	35.5	0.4	1.7

Table B.12: Measured power usage when using the SHA-256 accelerator with DMA in the alternative architecture.

Cores	H/s	W	H/s/W
1	6450	2.3	2804.3
2	12895	2.4	5372.9
3	19190	2.4	7995.8
4	21261	1.0	21261
5	19310	0.7	27585.7
6	18388	0.8	22985
7	18265	0.7	26092.9
8	18191	0.6	30318.3
9	18195	0.6	30325
10	18192	0.6	30320
11	18194	0.6	30323.3
12	18192	0.5	36384
13	18192	0.6	30320
14	18191	0.6	30318.3
15	18191	0.7	25987,1
16	18191	0.5	36382

Table B.13: Energy efficiency of software hashing in the initial architecture.

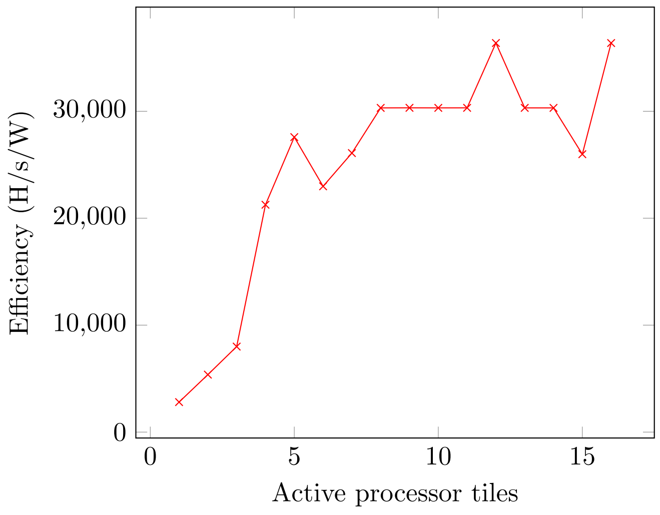


Figure B.3: Energy efficiency of software hashing in the initial architecture.

Cores	H/s	W	H/s/W
1	17557	2.2	7980.4
2	37539	2.2	17063.2
3	54912	2.0	27456
4	70179	2.1	33418.6

Table B.14: Energy efficiency when using the SHA-256 accelerator without DMA in the initial architecture.

Cores	**H/s**	**W**	**H/s/W**
1	18230	2.1	8681
2	38267	2.3	16637.8
3	56150	2.2	25522.7
4	71840	2.2	32654.5

Table B.15: Energy efficiency when using the SHA-256 accelerator with DMA in the initial architecture.

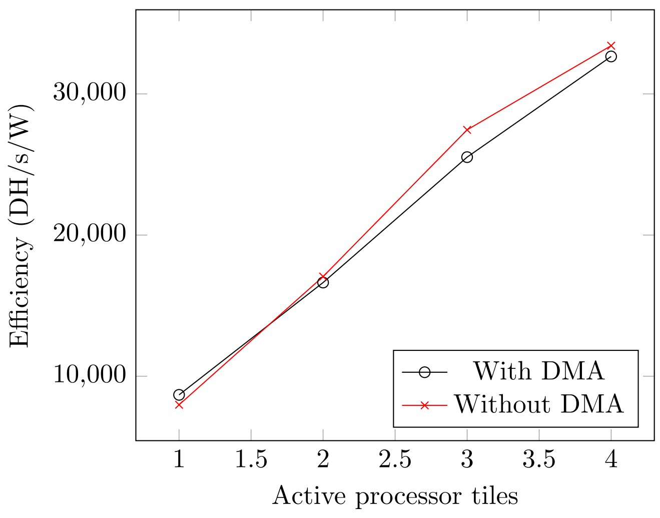


Figure B.4: Energy efficiency when using hardware accelerators in the initial architecture.

Cores	**H/s**	**W**	**H/s/W**
1	6418	2,3	2790.4
2	12841	2,5	5136.4
3	19130	2,3	8317.4
4	21364	1.0	21364
5	19404	0.9	21560
6	18420	0.8	23025
7	18253	0.8	22816.3
8	18206	0.8	22757.5
9	18193	0.7	25990
10	18192	0.7	25988.6
11	18192	0.8	22740
12	18192	0.8	22740
13	18191	0.7	25987.1
14	18190	0.7	25985.7

Table B.16: Energy efficiency when using software hasing only for the alternative architecture.

B.3.2 Alternative Architecture

Table B.16 shows the energy efficiency of doing software hasing in the alternative architecture. The numbers are plotted in figure B.5. While performance is greates with only 4 tiles, the reduction in power usage makes using of more tiles more efficient.

Tables B.17 and B.18 shows the energy efficiency for the SHA-256 accelerator, without and with DMA module respectively.

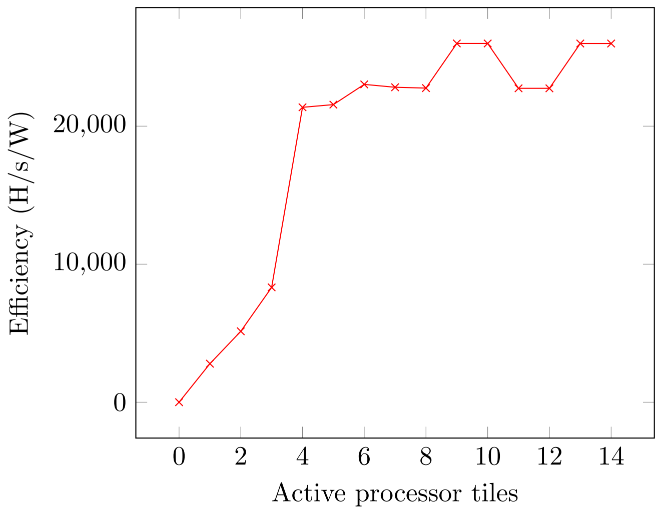


Figure B.5: Energy efficiency when using software hashing only in the alternative architecture.

Cores	DH/s	W	DH/s/W
1	11676	2.1	5560
2	24394	2.1	11616.2
3	38329	2.1	18251.9
4	53760	2.1	25600
5	70983	2.1	33801.4
6	90590	2.1	43138.1
7	107510	2	53755
8	122186	1.9	64308.4
9	135009	1.8	75005
10	146221	1.6	91388.1
11	156133	1.6	97583.1
12	164860	1.5	109906.7
13	172381	1.3	132600.8
14	179526	1.1	163205.5

Table B.17: Energy efficiency using the SHA-256 accelerator for the alternative architecture.

Cores	**DH/s**	**W**	**DH/s/W**
1	13227	2.2	6012.3
2	27410	2.4	11420.8
3	42627	2.3	18533.5
4	58989	2.4	24578.8
5	76556	2,4	31898.3
6	95259	2.4	39691.3
7	112599	2.3	48956.1
8	127515	2.2	57961.4
9	139937	1.9	73651.1
10	150069	2	75034.5
11	158421	1.9	83379.5
12	165271	2	82635.5
13	171143	1.9	90075.3
14	175711	1.7	103359.4

Table B.18: Energy efficiency using the SHA-256 accelerator with DMA for the alternative architecture.