Mei Zhang
Dr. Megan Thomas
Seminar in Computer Science [CS-4960]
October 24, 2023

Reducing Power Consumption with Loop Tiling Methods

0. Introduction

With the recent rise of Generative AI (Taylor C., 2023), the increasing demand for training deep learning models (Eke D., 2023) on large-scale datasets introduces the issue of computationally-heavy tasks increasing the carbon emissions left behind by data centers (Shao et al., 2022). To solve this problem, we look into power-reducing methods of computing.

One solution to reducing power consumption is implementing a compiler optimization known as loop tiling, a method that can speed up the performance of multiply and accumulate (MAC) operations, often in convoluted neural network (CNN) computations, by optimizing data locality (an efficient scope of data given an operation) to increase cache hits; effectively reducing computation time in data centers (Huang et al., 2021).

1. Abstract

This paper will first provide a brief background in data center machine idling, the proportional power usage (computationally intensive processes will proportionally consume the same percentage of power) of machine learning computations in data centers (Weihl et al., 2011), and the effectiveness of loop tiling methods in reducing power consumption (Ni et al., 2018). Furthermore, this paper will discuss the simple algorithmic logic behind loop tiling (Kelefouras et al., 2022), its memory optimization in multidimensional matrices, and the variables (ie: tile size selection (Zhao et al., 2018) and layer padding size (Alwani et al., 2016)) that enable it to perform successfully. Lastly, this paper will discuss the research done by Yuanhui Ni et al. (2018) assessing the effectiveness of integrating loop tiling in specialized neuromorphic hardware; examining their initial observation of RRAM (Resistive Random-Access Memory,

hardware using resistance as a measure for binary representation) crossbar (architecture in RRAM for efficient computations) based CNN, experimental setup, and data evaluation.

2. Data Center Energy Consumption

Research done by Avita Katal to survey software approaches for reducing energy consumption (2022), predicts data center energy consumption will increase from 200 TWh in 2016 to 2767 TWh by 2030, also directly increasing CO_2 emissions. To put the numbers into perspective, using the 2020 annual household site consumption, provided by the U.S. Energy Information Administration (EIA) (2023), 9.481×10^{15} Btu (British thermal units) of energy was consumed by U.S. households in 2020. In comparison, 200 TWh and 2767 TWh of power equates to 6.824×10^{14} and 9.441×10^{15} Btu respectively, which is 7.2% to 99.6% of the U.S. annual energy consumption in 2020 (EIA, 2023). In 2010, data centers and IT infrastructures have begun to take measures in response to the increasing cost of energy, eliciting researchers to develop new ways to reduce unnecessary machine usage in data centers, with some of the solutions as simple as turning the unused machines off (Berral et al., 2010). The machines that remain on are often never idle. Rather they operate at varying degrees of power usage depending on the task(s) being processed; proportional computing (Weihl et al., 2011). The concept of using the machines less, or at least effectively, brings the concept of reducing computational time, or improving computational speeds.

3. Loop Tiling & Optimizations

3.1. A Power Consumption Solution

Loop tiling is one solution that reduces data center energy consumption of CNN computations by 61.7% in comparison to a conventional compilation method known as one-to-one mapping (Ni et al., 2018). The effectiveness of loop tiling methods in comparison to data center machined-focused solutions are not immediately comparable due to the power-usage efficiency unit (PUE) (Isazadeh et al. 2023). Compilation methods are at the discretion of the end-user, and are niche to computing convolutional tasks.

3.2 Loop Tiling Algorithm

The objective of a loop tiling algorithm is to reduce the number of memory access misses, and inversely, gain memory access hits; allowing the computation to access data from cache (Kelefouras et al., 2022) such as L1 and L2 cache (Prett B., 2016). To accomplish this, multiple nested for-loops are set up to iterate matrix-matrix multiplication (MMM) over a user-defined tile size, which should be set to a number that coincides with the memory capabilities of the cache, which also vary depending on the machine (Zhao et al., 2018).

3.2.1 Loop Tiling Functional Overview

In Figure 1b, Kelefouras (2018) writes a simple loop tiling algorithm to conduct MMM using Matrix A and Matrix B. The final output of this process will be Matrix C of size ($M \times N$) where M – not shown in Figure 1a – is the column size of Matrix A (shown in Figure 1a as *i loop*) and N is the row size Matrix B (shown in Figure 1a as *j loop*).

In lines 1 and 2 of Figure 1b, the nested for loops serve the purpose of dividing an empty Matrix C into a matrix of ($Tile_i \times Tile_j$) tiles where *Tile* is the tile size defined by the user. The *Tile* value will later be used to determine the range of computations allowed for loops 4-6. The line 3, defines a shared dimension between both Matrix A and B, *kk*, which will remain constant throughout its loop cycle. The purpose of this variable is to define the range for the sixth loop defined later. In lines 4 and 5, the for loops iterate over a given tile of ($M \times N$) tiles in Matrix C, and enable the loop in line 6 to compute MMM from *k* to *kk* for all of *i* and *j*.

3.2.2 Loop Tiling Example Walkthrough

Starting Tile: Tile *number* of tile at [*ii* to *ii*+*Tilei*][*jj* to *jj*+*Tilej*]: Tile 0 [0][0].

Input: Matrix A: Tile 0 at [0][0] of size [7][9] and Matrix B: Tile 0 at [0][0] of size [9][8].

Start of MMM: Tile 0: A[0][0] × B[0][0].

1. Complete iteration of (*k* to *kk*+*Tilek*) for one (*j* to *jj*+*Tilej*): A[0][9] × B[9][0].
2. Complete iteration of (*j* to *jj*+*Tilej*) for one (*i* to *ii*+*Tilei*): A[0][9] × B[9][8].
3. Complete iteration of (*i* to *ii*+*Tilei*) for tile (*k* to *kk*+*Tilek*): A[7][9] × B[9][8].
4. For each iteration from *k* to *kk*+*Tilek*, add the result of (A[*i*][*k*] × B[*k*][*j*]) to C[*i*][*j*].

End of MMM: A[7][9] × B[9][8].

Output: Matrix C: Tile 0 at [0][0] of size [7][8].

Increment Tile: Tile 0 [0][0] → Tile 1 [0][1] … Tile 3 [0][3] → Tile 4 [1][0] … Tile 16 [3][3].

Continue Computations: Repeat steps 1-4 of MMM for each tile increment.

Once all tiles have finalized their computations, a computed Matrix C of size [ii][jj] is returned.

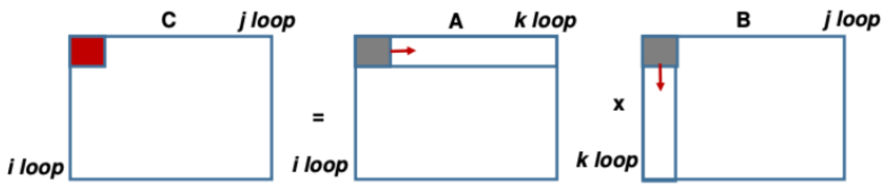


Figure 1a. (Kelefouras et al., 2022)

```
for (ii = 0; ii < N; ii += Tilei)
    for (jj = 0; jj < N; jj += Tilej)
        for (kk = 0; kk < N; kk += Tilek)
            for (i = ii; i < ii+Tilei; i++)
                for (j = jj; j < jj+Tilej; j++)
                    for (k = kk; k < kk+Tilek; k++)
                        C[i][j] += A[i][k] × B[k][j]
```

Figure 1b. (Kelefouras et al., 2022)

3.3. Tile Size-Selection

Selecting the appropriate tile size is a crucial part of optimizing loop tiling algorithms. The tile size defines the amount of data processed in each iteration. If the tile size is too small, it may underutilize the memory size of the cache. If the tile size is too large, it could exceed available memory space from the cache, and cause excessive data movement. The ideal tile size is typically defined by the user through trial-and-error, and should maximize data reuse and minimize overhead memory access with hardware and memory constraints in mind (Zhao et al., 2018).

Another important thing to consider in selecting tile sizes is the static nature of user-defined tile sizes. A study done by Jiacheng Zhao and team ran a test using three different tile sizes, outer-level size (*OB,* or outer bound) $\in$ {448, 616, 840}. In Figure 2, a normalized graph with 840 as the base comparison, shows the count of cache misses of a tile size for each workload. The lowest miss count between all three tile sizes at a given workload indicates the most optimal tile size. The analysis between the three tile sizes: tile size 448 shows to be optimal for workloads: 2, 3, and 8-10; tile size 616 shows to be optimal for workloads: 1, 6, and 7; and tile size 840 shows to be optimal for workload 4 and 5. With this information, we can infer the need for dynamic tile sizing solutions to optimize cache memory access and increase run time speeds (Zhao et al., 2018).

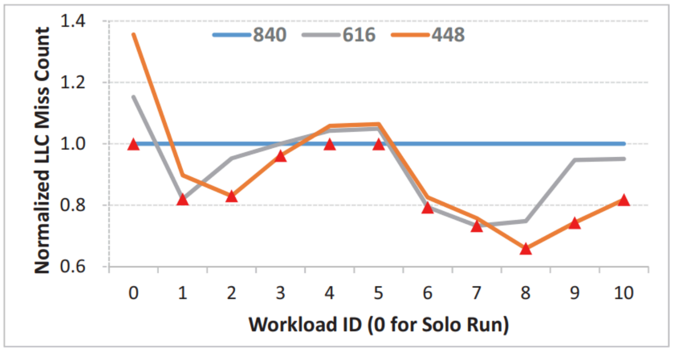
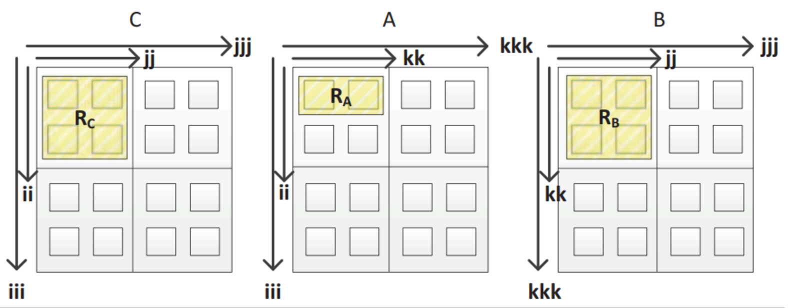


Figure 2. (Zhao et al., 2018)

As a solution for manual user-defined tile sizes, Jiacheng Zhao and team introduces a peer-aware tile size selection (TSS) model. By inserting a cache-hit and miss monitoring system for a tiled section of a matrix, the system can identify the frequency of data access in the cache and tailor an optimized tile size for the following tile. In Figure 3, this resizing of reuse patterns can be seen from a tiled general matrix multiplication (GEMM). GEMM has two level-tiling and includes both inner-level tiling (*IB*, or inner bound; used for private cache) and outer-level tiling (*OB*, used for public cache). Jiacheng and the team's objective was to select the best size for the outer-level tiling. And through a series of arithmetic computations, the tile size, *OB*, can be computed (Zhao et al., 2018).



Reuse Pattern	Reuse Size (n)	Reuse Distance (rd)	Reuse Iteration	Reuse Count (rc)
R_A	OB*IB	2*OB*IB+IB*IB	jj	$\frac{N}{IB} * \frac{N}{IB} * \frac{N}{OB}$
R_B	OB*OB	2*OB*IB+OB*OB	ii	$\frac{N}{IB} * \frac{N}{OB} * \frac{N}{OB}$
R_C	OB*OB	3*OB*OB	kkk	$\frac{N}{OB} * \frac{N}{OB} * \frac{N}{OB}$

Figure 3. (Zhao et al., 2018)

In a programming context, we first identify the cache pressure as the shared cache miss count (LLC), which is later referred to when a request to get the required tile size is made. To compute the tile size, three parameters must be passed: the cache pressure, the layer model, and an annotation objective declaring what optimization is being requested (Zhao et al., 2018).

```
void gemm(int N)
{ int i, j, k;
        p = get_instant_cache_pressure();
        OB = get_required_ts(p, mode, objective);
        for (iii = 0; iii < N; iii += OB)
                for (jjj = 0; jjj < N; jjj += OB)
                        for (kkk = 0; kkk < N; kkk += OB)
                                …
}
```

Figure 4. (Zhao et al., 2018)

This model requires the hardware support cache partitioning to run different processes using only a section of the cache, and problems also arise in regard to the volatile behavior latency-sensitive processes. This model also requires the cache partitioning-support to run different processes using only a section of the cache, and problems also arise in regard to the volatile behavior latency-sensitive processes (Zhao et al., 2018).

3.4. Padding Layer & Data Locality

To improve data locality, we can implement a padding layer to add extra elements (layer padding) around the input data of convolution layers, which will allow for more efficient data access patterns during convolution. Manoj Alwani and team, from Stony Brook University, found that by adding an extra 362KB of on-chip storage reduced the need to transfer data off-chip, resulting in a 95% increase and reduced the size of total transfers from 77MB to 3.6MB (Alwani et al., 2016).

4. Loop Tiling-Integrated Hardware

4.1. RRAM Background

While loop tiling methods often utilize cache as a component of fast memory access, Yuanhui Ni notes upcoming metal oxide resistive switching random access memory (RRAM, hardware specialized to enable binary representation through resistance states) allows multiply and accumulate (MAC) operations data to be stored with low power consumption. Through this discovery, a new approach in energy-focused loop tiling techniques ignited for RRAM crossbar-based computing systems (RCSs) (Ni et al., 2018).

4.2 Experiment Objective

The objective of Yuanhui Ni's experiment is to measure their initial observation and assess whether it can meet limitations in power budgets. The initial observation being: memory controlled by the FPGA (Field Programmable Gate Array, or customizable processing units for niche tasks) can be shared between adjacent convolution layers; later introduced as the peripheral circuit unit (PeriCU)-reuse scheme, which enables memory access bypassing where convolution layers can retrieve readily available data from its dependency layer, and improves the energy efficiency RCSs. And should the convolution layers not align adjacently (tiled in different crossbar logic planes or CLP), then a safe starting time is given to delay the convolution layer's start time (Ni et al., 2018).

4.2. Experiment Setup

Prefacing the research experiment setup, note that the hardware used for Yuanhui's experiments were specialized to compute neuromorphic-specific tasks (Yoon et al., 2023); in this context, CNN computations.

Yuanhui's research utilizes RRAM-based crossbar arrays, RRAM devices arranged in crossbar architecture or simply hardware arranged in matrix-fashion, for the CNN accelerator. The peripheral circuits are designed with parameters such as a resistance range (500Ω - 500kΩ) to define binary representation, an amplifier (AMP) to manage the electric flow for read-write operations, an analog-to-device/device-to-analog converter (ADC/DAC) to interface data read-write data transfers, and integrated circuit (IC) technology using a 65nm node (Imran et al., 2023). An off-chip DDR3 DRAM is used as the main memory with an energy budget of 70pJ/bit and bandwidth of 12.8GB/s. (Ni et al., 2018).

4.3 Comparison Assessment

For reliable comparison between loop tiling and non-loop tiling methods, the experiment results are compared to conventional non-loop tiling methods to examine the power efficiency of loop tiling on the RRAM-based CNN accelerator. Validation will later be conducted along the controlled variable experiments using the same setup with, but without loop tiling. The independent variables are as follows: the loop tiling algorithm, layer assignment, and scheduling for convolutional layers in the CNN accelerator design (Ni et al., 2018).

4.4 Results

In Figure 5, the results of normalized energy efficiency can be seen. The consistent variable, *Conventional,* performs slightly better than the Naive Multi-CLP (non-loop tiling multi-CLP). Though, both in comparison to the Ene-Tiled Multi-CLP (loop tiling integrated multi-CLP), the Ene-Tiled Multi-CLP's improves power reduction by an average of 61.7% to the alternatives due to its energy-driven performance in rescheduling convolution layers (Ni et al., 2018).

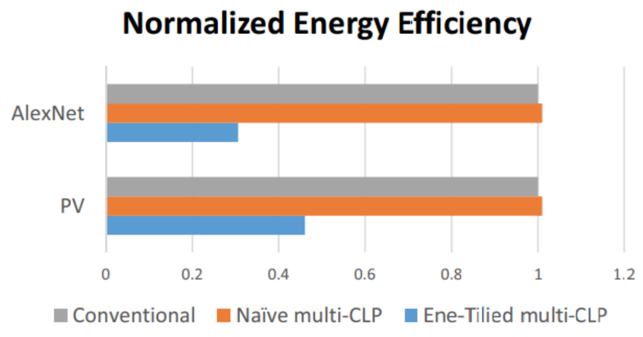


Figure 5. (Yuanhui Ni et al., 2018)

In Figure 6, normalized average power consumption results can be seen, which displays a rather contradicting outcome for Naive Multi-CLP – after acknowledging its energy efficiency – as the Naive Multi-CLP and Ene-Tiled Multi-CLP result in distinguishable power reduction rates of 82.2% and 91.4%, respectively, in comparison to the *Conventional* approach (Ni et al., 2018).

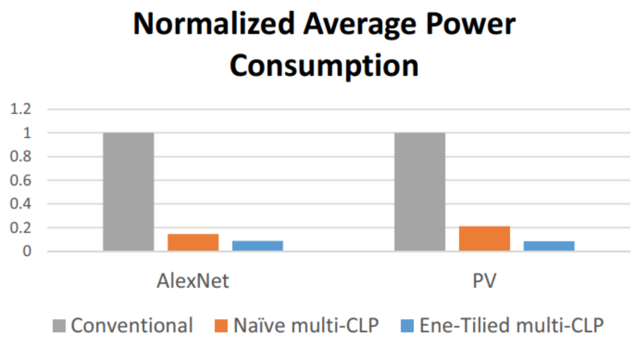


Figure 6. (Yuanhui Ni et al., 2018)

4.5 Performance Tradeoffs

Loop tiling offers significant advantages in terms of energy efficiency and power reduction, but it is important to consider the performance tradeoffs. The additional overhead computation brought by loop tiling can impact execution time. And while loop tiling is highly effective in reducing power consumption, it slightly increases execution time compared to non-loop tiling compilers. Though, the reduction in tile size to enable cache usage can mediate and potentially outweigh the cons of this dilemma. This performance tradeoff requires critical analysis of prioritizing performance over power budgets (Ni et al., 2018).

5. Conclusion

Loop tiling methods help reduce data center power consumption by optimizing the data locality of MAC operations in MMM. By utilizing nested for loops to iterate over the 2-dimensional space of a matrix, we iterate over the output matrix in user-defined tile-sized sections where further iterations occur to write in the results of the tile's MAC operations. To further optimize loop tiling data locality, the proposed peer-aware TSS can be implemented to present an adaptive behavior towards selecting tile sizes and leveraging hit frequency as the basis for resizing. As further improvements in improving data locality, by allocating extra memory to convolution layers, we can accommodate for the boundary effect found in data gradients during convolution. Specialized hardware such as RCS can be used to store MAC operations with extremely low power. RCS hardware also allows for loop tiling integration, which may require the addition of safe starting time to enable consistent adjacency between convolution layers to memory access bypass their data. Through this implementation, the average power consumption is reduced a further 9.2% against non-loop tiling methods. Despite adding additional overhead computation, loop tiling methods effectively leverages data locality to efficiently access memory with power budgets, benefiting upstream systems, and leading to a more eco-friendly approach to AI development.

5. Personal Reflection

It was highly interesting to learn about loop tiling and how it leverages tiling CNN processes into smaller tiles to optimize the use of cache's fast data access and retrieval (Kelefouras et al., 2022). Through this energy efficient compiling technique for CNN computations, data centers can enter states of low power much earlier (Weihl et al., 2011).

Some personal questions remain unsolved: the flexibility of specialized hardware to accommodate for convolution layers of various sizes. Would an RRAM crossbar based CNN with a fixed matrix size work with other sizes as well? If not, would a compiler revision of loop tiling to resize a matrix to fit RRAM crossbar based CNN fix the issue? If not, how efficient would it be to use fixed-size processing units for CNN? Last and most importantly, would an adjustment in hardware to be flexible for various matrix sizes be energy efficient? Some of these questions are answered by Tianshi Chen et al. (2014) on their research regarding building high-throughput accelerators for large-scale CNN. Though, their research solution does not provide scope of RRAM crossbar based CNN hardware, which is the hardware piece I remain interested in.

The core research of my paper, done by Yuanhui Ni et al. (2018), was the highlight of my interests as it discusses the topic of hardware units with extremely low energy consumption. Seeing the parallels of both software and hardware optimizations in regard to loop tiling struck my curiosity, especially towards hardware. Though, learning about hardware peaks my interest more than learning the software concepts, and I would love to further explore specialized hardware in the realm of machine learning computations.

References

Barrel J., Goiri Í., Nou R., et al. (2010) Towards energy-aware scheduling in data centers using machine learning. e-Energy '10: Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking pp. 215-224. https://doi.org/10.1145/1791314.1791349

Chen T., Du Z., Sun N., et al. (2014) DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning. ACM SIGARCH Computer Architecture News *42*(1) pp. 269-284. https://doi.org/10.1145/2654822.2541967

Eke D. (2023) ChatGPT and the rise of generative AI: Threat to academic integrity? Journal of Responsible Technology (Vol. 13). https://doi.org/10.1016/j.jrt.2023.100060

Energy Information Administration (EIA) (2023). Summary annual household site consumption and expenditures in the United States—totals and intensities, 2020. 2020 RECS Survey Data. https://www.eia.gov/consumption/residential/data/2020/c&e/pdf/ce1.1.pdf

Imran M., Almeida F., Basso A., et al. (2023). High-speed SABER key encapsulation mechanism in 65nm CMOS. Journal of Cryptographic Engineering *13*(3). https://doi.org/10.1007/s13389-023-00316-2

Isazadeh A., Ziviani D., Claridge D. (2023). Global trends, performance metrics, and energy reduction measures in datacom facilities. Renewable and Sustainable Energy Reviews (Vol. 174). https://doi.org/10.1016/j.rser.2023.113149

Katal A., Dahiya S., Choudhury T. (2023). Energy efficiency in cloud computing data centers: a survey on software technologies. Cluster Computing *26*(6) pp. 1845–1875. https://doi.org/10.1007/s10586-022-03713-0

Kelefouras V., Djemame K., Keramidas G., et al. (2022). An Analytical Model for Loop Tiling Transformation. Embedded Computer Systems: Architectures, Modeling, and Simulation pp. 95-107. SAMOS 2021. Lecture Notes in Computer Science, vol 13227. https://doi.org/10.1007/978-3-031-04580-6_7

Ni Yuanhui Ni., Qiu K., Chen W., et al. (2018). Low power driven loop tiling for RRAM crossbar-based CNN. Proceedings of the 33rd Annual ACM Symposium on Applied Computing pp. 375-380. https://doi.org/10.1145/3167132.3167174

Prett B. (2016) Efficient use of Tiling. Intel. https://www.intel.com/content/www/us/en/developer/articles/technical/efficient-use-of-tiling.html

Renganarayanan L., Kim D., Strout M., et al. (2012). Parameterized loop tiling. ACM Transactions on Programming Languages and Systems *34*(1) pp. 1-41. https://doi.org/10.1145/2160910.2160912

Shao X., Zhang Z., Song P., et al. (2022) A review of energy efficiency evaluation metrics for data centers. Energy and Buildings (Vol. 271). https://doi.org/10.1016/j.enbuild.2022.112308

Taylor C. (2023) ChatGPT creator OpenAI is reportedly earning $80 million a month—and its sales could be edging high enough to plug its $540 million loss from last year. Fortune. https://fortune.com/2023/08/30/chatgpt-creator-openai-earnings-80-million-a-month-1-billion-annual-revenue-540-million-loss-sam-altman/

Weihl B., Teetzel E., Clidaras J., et al. (2011). Sustainable data centers. XRDS: Crossroads, The ACM Magazine for Students *17*(4) pp. 8-12. https://doi.org/10.1145/1961678.1961679

Yoon et al., (2023). A review on device requirements of resistive random access memory (RRAM)-based neuromorphic computing. APL Materials *11*(9). https://doi.org/10.1063/5.0149393

Zhao et al., (2018). Revisiting Loop Tiling for Datacenters. ICS '18: Proceedings of the 2018 International Conference on Supercomputing pp. 328-340. https://doi.org/10.1145/3205289.3205306