POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

On the Design of Multi-directional Systolic Arrays for Band and Generic Matrix-Matrix Multiplications

Laurea Magistrale in Electronics Engineering - Ingegneria Elettronica

Author: Leonel Gouveia Ergin

Advisor: Prof. Christian Pilato

Co-advisor: Stephanie Soldavini

Academic year: 2021-2022

1. Introduction

Increasing computing performance can no longer be achieved just with the miniaturization of transistors [3]. Due to Post-Dennardian scaling, the power densities in circuits are increasing each generation, forcing larger and larger portions of the circuit to be shut off (*Dark Silicon*) [3]. One way to address the dark silicon problem is to employ specialized co-processors (*accelerators*) [3]. The reliance of increasingly popular AI applications on fast matrix multiplications encouraged us to study systolic [1] co-processors.

1.1. Contributions

We consider systems with fixed arrays of processing elements, achieving different operations by rerouting the connections between them. As a working prototype, we have implemented an array capable of two distinct matrix operations: generic and band.
We have also implemented a complete workflow allowing us to stitch RTL kernels into HLS systems. Our working system can be used as a reference example to understand how to interface custom RTL kernels with complete Xilinx Systems using RTL-HLS hybrid design.

2. Background

2.1. Systolicism

Systolic systems are composed of regular, locally interconnected arrays of processing elements. Their principle is to fetch data rhythmically and let it ripple through the processing elements to achieve a computational result, as shown in Figure 1. A systolic system develops an algorithm in space rather than time.

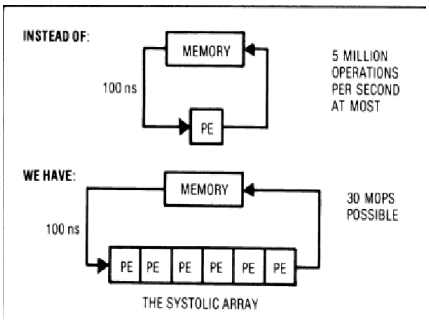

Figure 1: Basic Systolic principle, from [1]

Many common problems have been studied using systolic approaches [1]. Recently, systolic architectures are regaining a lot of attention due to the increasing popularity of AI applications, which exploit matrix multiplications intensively.

$$\overset{q}{\left\{\right.}\overbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 & \dots & 0 \\ a_{21} & a_{22} & a_{23} & a_{24} & 0 & \dots \\ 0 & a_{32} & a_{33} & a_{34} & a_{35} & 0 \\ \vdots & 0 & a_{43} & a_{44} & a_{45} & \\ & \vdots & & \ddots & \ddots & \ddots \\ 0 & 0 & & & & \end{bmatrix}}^{p} \cdot \overset{p}{\left\{\right.}\overbrace{\begin{bmatrix} b_{11} & b_{12} & 0 & \dots & & 0 \\ b_{21} & b_{22} & b_{23} & 0 & \dots & 0 \\ b_{31} & b_{32} & b_{33} & b_{34} & 0 & \\ 0 & b_{42} & b_{43} & b_{44} & b_{45} & \\ \vdots & \vdots & & \ddots & \ddots & \ddots \\ 0 & 0 & & & & \end{bmatrix}}^{q} = \overset{w}{\left\{\right.}\overbrace{\begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} & \dots & & 0 \\ c_{21} & c_{22} & c_{23} & c_{24} & c_{25} & \dots & 0 \\ c_{31} & c_{32} & c_{33} & c_{34} & c_{35} & c_{36} & \dots & 0 \\ c_{41} & c_{42} & c_{43} & c_{44} & c_{45} & c_{46} & c_{47} & \\ \vdots & \vdots & & \ddots & \ddots & \ddots & & \\ 0 & 0 & & & & & & \end{bmatrix}}^{w}$$

Figure 2: Band matrices and their multiplication. Here $p = 3$, $q = 2$ and $w = 4$

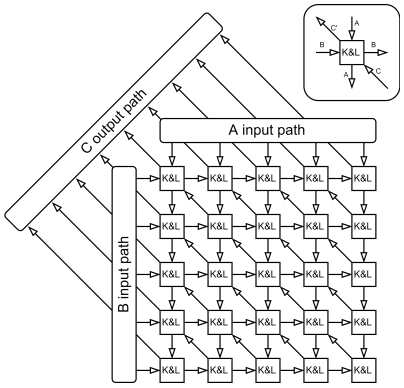

Figure 3: K&L systolic array for BMMM

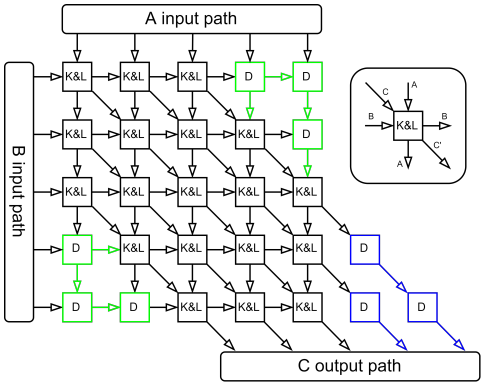

Figure 4: Systolic array for GMMM

2.2. Systolic Array Design for Band Matrix-Matrix Multiplication (BMMM)

2.2.1 Kung and Leiserson Processing Element (KPLE)

The KLPE features three inputs A, B and C, and three outputs A, B, and C'. The C' output is defined by the operation $C' = C + A \times B$. All outputs are registered.

2.2.2 Band Matrices

The multiplication of two band matrices A (with width p and height q) and B (with width q and height p) result in a matrix C which has both width and height $w = p + q - 1$. We call w the band width. This operation is shown in Figure 2.

2.2.3 Systolic Array for BMMM

By interconnecting KLPE's, we build a multiplier introduced by Kung and Leiserson [2]. This array can pipeline the multiplication of any size of matrices, as long as their w is smaller or equal than the lateral size of the array. In Figure 3, we can see the array.

2.3. Systolic Array Design for Generic Matrix-Matrix Multiplication (GMMM)

Figure 4 displays an array which can achieve GMMM. It is comprised of KLPE's and delay blocks, labeled D in the diagram. In order to multiply two generic matrices of size N, we need an array with $(2N - 1)^2$ processing elements.

3. Parametric Multi-directional Systolic Kernel

Based on the similarities between the BMMM and GMMM systems, we have implemented a Unified Matrix-Matrix Multiplier (UMMM). The diagonal paths in the array can be rerouted, allowing it to behave either like the BMMM or the GMMM array. It reuses the same KLPE's for both operations, as shown in Figure 5.

3.1. GMMM Peripherals

Figure 6 shows the sequence of data dispatching to/from the GMMM kernel. Each matrix is sent at the rate of one line per cycle. The peripherals must steer this data to the correct inputs. The opposite steering happens at the output.

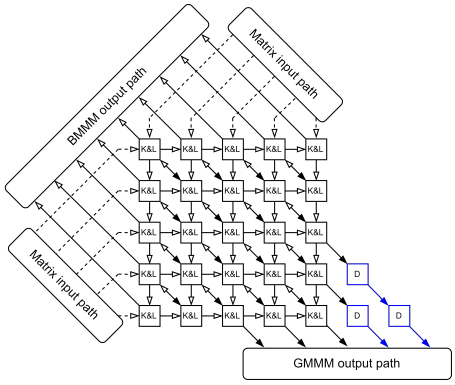

Figure 5: Systolic array core for UMMM

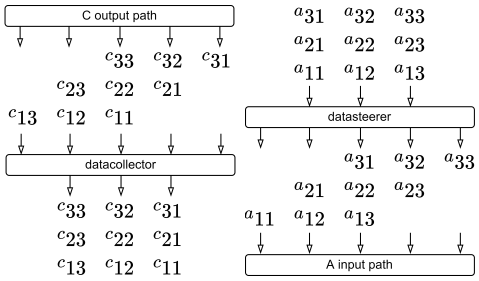

Figure 6: GMMM peripherals

3.2. BMMM Peripherals

The band input peripheral device takes in rows in the custom format and dispatches the elements according to the pattern seen in Figure 7. Our solution employs internal buffers to access multiple rows of data simultaneously. The same operation happens in reverse for the output.

3.3. UMMM Kernel

A final overview of the UMMM kernel can be seen in Figure 8. The `opmode` signal controls the array configuration.

4. RTL Kernel Integration

4.1. HLS Wrapper

The HLS wrapper around our kernel deals with memory transfers to and from the Host computer. Its organization is shown in Figure 9.

4.2. Read and Write Functions

Our reading hardware must read data from the 256-bit High Bandwidth Memory (HBM) transfers (called data chunks) and reconstruct vectors of data. The writing hardware uses the same structure in reverse. The algorithm is displayed in Figure 10.

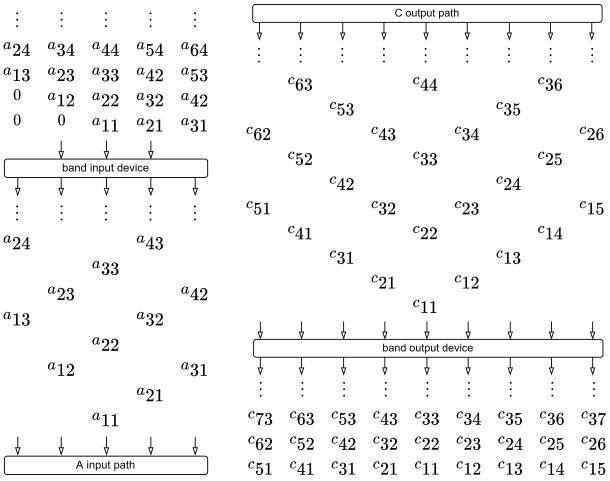

Figure 7: BMMM peripherals

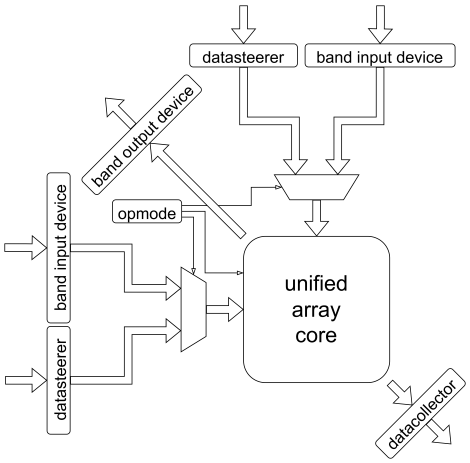

Figure 8: Unified array

4.3. Equivalent HLS Implementation

Our HLS implementation is designed to fit in place of the RTL. Both the GMMM and the BMMM section are structured into a triple-for-loop implementation of the matrix multiplication. We attempted to play both fields as fairly as possible, optimising both RTL and HLS to the extents of our capabilities.

5. Experiments and Results

5.1. First Comparison

This test is our first comparison of equally capable kernels. The results feature in Table 1. All the metrics are similar except for the DPS's. The execution times are very similar.
From the architecture alone, we expected our RTL to be much faster than our HLS in both GMMM and BMMM. As we can see, this is not the case. In addition, each GMMM operation

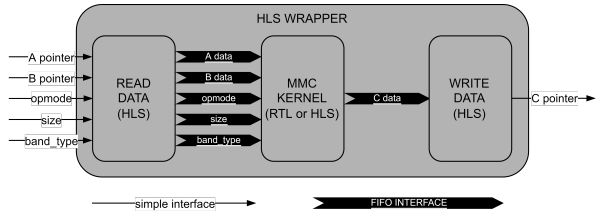

Figure 9: HLS wrapper

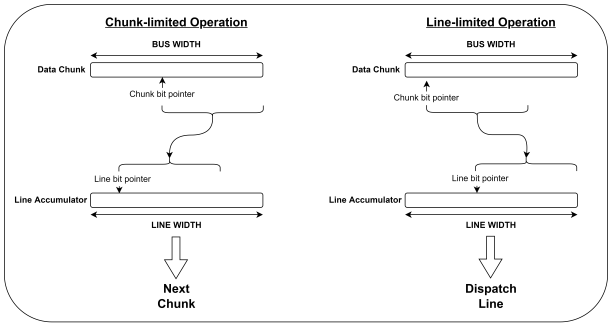

Figure 10: Read Data Algorithm

on the RTL kernel should take 16 cycles. At 148.6MHz, this equates to 108ns per GMMM. This is 100x faster than what we measured.

5.2. Kernel Running Rates

We have implemented a stall counter within our RTL kernel's GMMM section. We approach a running rate of 1.36%. From these results, we expect that this operation could run 73.5x faster. For the BMMM, we have found that it approaches a running rate of 3.81%. We could expect an increase of performance of 26.25x, with appropriate memory hardware.

5.3. RTL GMMM Analysis with Custom Memory Management

We have implemented bespoke memory hardware for the GMMM section of a kernel with parameters `MAT_SIZE` = 16 and `DATA_WIDTH` = 8. We ensured a 1 line per cycle data delivery rate. the results of our experiment are in Figure 11:

- Until 1000 operations, the operating time is dominated by the launch of the kernel.
- At high amounts of streamed operations, we observe a performance gap of 2.6x between our measured time and the theoretical time.

Figure 12 compares the former kernel with its HLS counterpart:

- If the target application only runs occasional GMMM operations, implementing a systolic GMMM array compared to a HLS version is a waste of area (see Table 2).
- For many streamed GMMM operations, our RTL kernel performs 6.63x faster than our HLS kernel.

In Table 2, we see that theoretically, the HLS is 20x slower than the RTL. By compounding the 6.63x real-world gap with the 2.6x gap between the RTL and its theoretical time and by adapting for clock frequency disparities, we obtain 19.95x. This shows us that the HLS is running at its full potential but the RTL is being throttled because data requirements are too demanding for our FPGA.

Table 1: Comparison of area and performance metrics. The GMMM time is averaged over 1000 streamed operations. The time for BMMM is measured for one operation of size 1000.

`MAT_SIZE`=16 `DATA_WIDTH`=8	LUT (%)	REG (%)	DSP (%)	RAM (%)	Clock Frequency (MHz)	Time GMMM (μs)	Time BMMM (ms)
RTL	11.92	8.70	10.96	11.41	148.6	10.43	15.20
HLS	11.75	7.81	0.39	12.20	287.3	9.26	15.48
RTL/HLS ratio	1.014	1.114	28.10	0.935	0.517	1.126	0.982

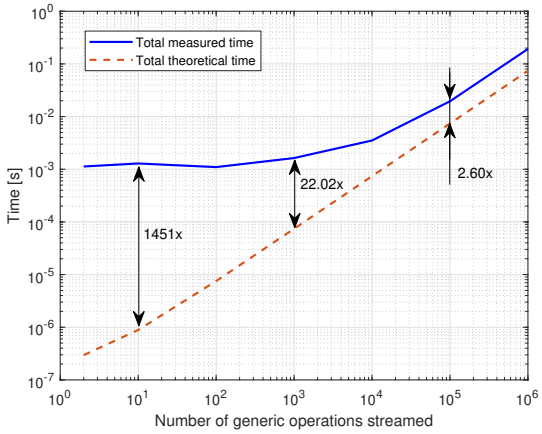

Figure 11: Time measurement for a RTL kernel using custom memory management.

Table 2: Area comparison for optimised kernels with custom memory management hardware.

`MAT_SIZE` =16 `DATA_WIDTH` =8	LUT (%)	REG (%)	DSP (%)	RAM (%)	FREQ (MHz)	Compute Cycles
HLS	11.00	7.62	0.33	11.46	250.0	20
RTL	11.58	8.53	10.83	13.05	216.4	1
RTL/HLS ratio	1.053	1.119	32.818	1.139	0.866	1/20

5.4. Final Kernels

The results of the area utilisation for a family of kernels using our fully-parametric data manage-

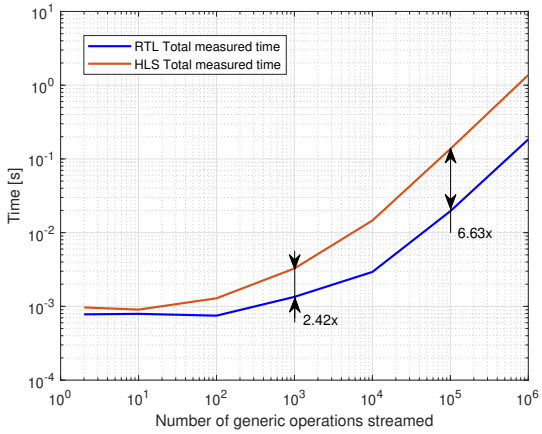

Figure 12: Time comparison (RTL vs. HLS) for GMMM with custom memory management.

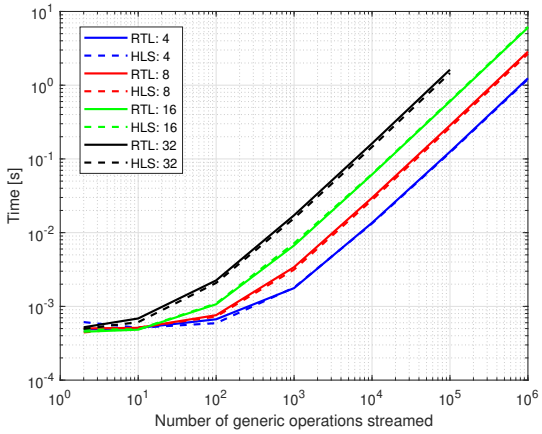

Figure 13: Timing comparison between RTL and HLS for GMMM.

ment hardware feature in Table 3.

Table 3: Area report for our final kernels. Every kernel was compiled with `DATA_WIDTH`= 8.

	MAT_SIZE	FREQ (MHz)	LUT (%)	REG (%)	DSP (%)	RAM (%)
RTL	4	242	10.32	7.45	0.75	11.41
	8	214	10.98	7.69	2.70	11.41
	16	203	12.17	8.55	10.86	11.76
	32	153	15.14	11.81	44.19	12.80
HLS	4	251	10.47	7.48	0.79	11.46
	8	226	10.97	7.54	0.28	11.46
	16	199	11.83	7.63	0.37	11.81
	32	174	12.55	7.78	0.54	13.37
$\frac{\text{RTL}}{\text{HLS}}$ ratio	4	0.97	0.99	1.00	0.95	1.00
	8	0.95	1.00	1.02	9.64	1.00
	16	1.02	1.03	1.12	29.35	1.00
	32	0.88	1.21	1.52	81.83	0.96

5.4.1 GMMM Analysis

In Table 4, we see that both the HLS and RTL kernels are limited by the writing hardware. Figure 13 shows the timing comparison. All the comparable kernels take the same amount of time to complete. In Figure 14, we see our timing predictions compared to the achieved times.

Table 4: Latency report for the GMMM operation, in number of cycles per line. The slowest kernels in the chain are highlighted.

GMMM MAT_SIZE	RTL Read	RTL Compute*	RTL Write	HLS Read	HLS Compute*	HLS Write
4	2	1	**76**	3	3.00	**76**
8	2	1	**75**	3	13.00	**75**
16	2	1	**75**	2	20.00	**75**
32	3	1	**76**	3	36.53	**76**

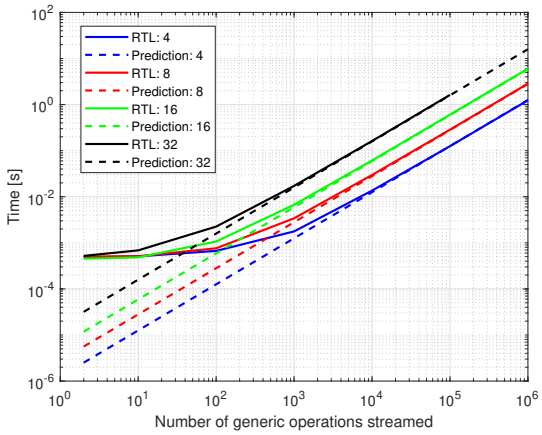

Figure 14: Timing results for our RTL kernels running GMMM and the associated theoretical predictions.

5.4.2 BMMM Analysis

In Table 5, we can see that the write function is still the bottleneck for our RTL implementation. However, the HLS is limited by the computation kernel. In Figure 15, we see that despite using non-optimal data management hardware, our RTL kernels are still faster than our HLS kernels. Our timing predictions for the BMMM kernels can be found on Figures 16 and 17. They are very accurate, except for the smallest RTL kernel, whose prediction grossly overestimates the operating time.

6. Conclusions

We created a systolic RTL kernel for unifying generic and band matrix-matrix multiplications. Our kernel contains 30x more DSP with

Table 5: Latency report for the BMMM operation, in number of cycles per line. The slowest kernels in the chain are highlighted.

BMMM	RTL			HLS		
`MAT_SIZE`	Read	Compute*	Write	Read	Compute	Write
4	2	3	**76**	2	**666**	76
8	2	3	**76**	2	**1591**	76
16	3	3	**150**	3	**3823**	150
32	3	3	**298**	2	**9824**	298

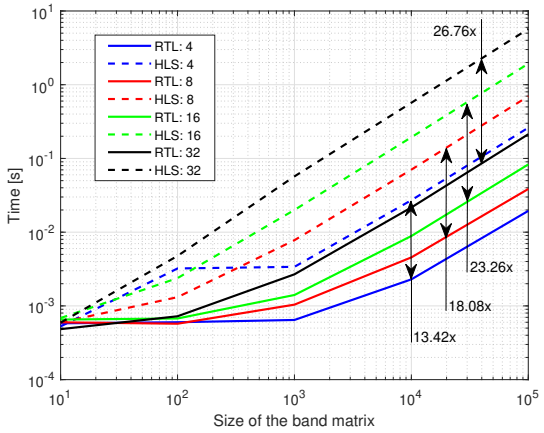

Figure 15: Timing results for our RTL and HLS kernels running BMMMM, using fully-parametric memory management.

a theoretical performance improvement of 20x for streamed GMMM operations and 610x for streamed BMMM operations. However, our design has an inexplicable 2.6× performance gap between our bottleneck-free kernels and their theoretical maximum speed. This requires further investigation of the underlying hardware.
For the hybrid RTL-HLS workflow, we will add foolproofing for the setup of high-level parameters in multiple files to avoid mismatching HLS peripherals with RTL blackboxes.

7. Acknowledgements

I'd like to thank my friends and family for their support. I would also like to thank my advisor, Prof. Christian Pilato, for always asking the right questions and single-handedly elevating the quality of my work. Finally, the most special thanks have to go to Stephanie Soldavini, who throughout this thesis has played the role of advisor, assistant, helper, consoler, consultant, Linux-guru, deadlock resetter, documentation magician and last but certainly not least, friend.

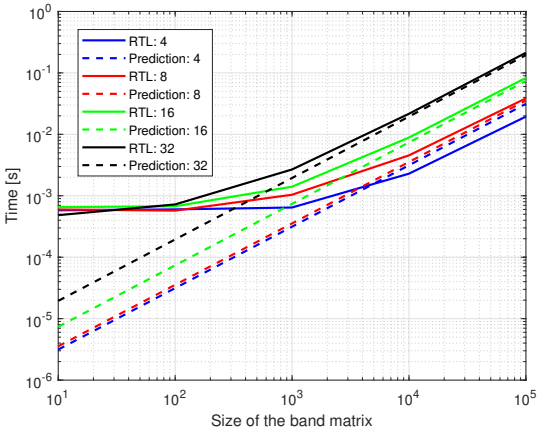

Figure 16: Timing results for our RTL kernels running BMMM and the associated theoretical predictions.

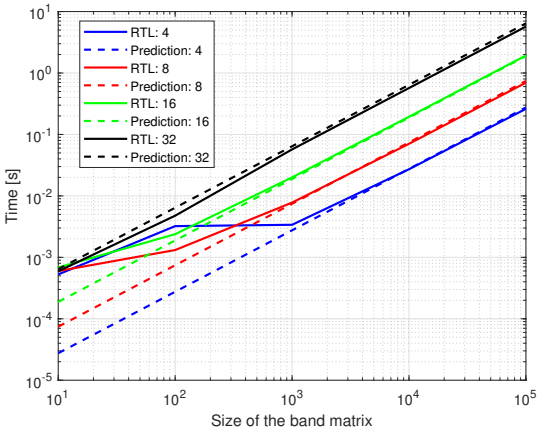

Figure 17: Timing results for our HLS kernels running BMMM and the associated theoretical predictions.

References

[1] Hsiang-Tsung Kung. Why systolic architectures? *Computer*, 15(01):37–46, 1982.

[2] HT Kung and Charles E Leiserson. Systolic arrays (for vlsi). In *Sparse Matrix Proceedings 1978*, volume 1, pages 256–282, 1979.

[3] Michael B. Taylor. Is dark silicon useful? harnessing the four horsemen of the coming dark silicon apocalypse. In *ACM/EDAC/IEEE DAC*, pages 1131–1136, 2012.