University of Delaware
Department of Electrical and Computer Engineering
Computer Architecture and Parallel Systems Laboratory

Energy efficient tiling on a Many-Core Architecture

Elkin Garcia
Daniel Orozco
Guang R. Gao

CAPSL Technical Memo 102
October, 2010

Contents

1 Introduction **1**

2 Energy Consumption Model on a Many-Core Architecture **2**
2.1 The IBM Cyclops-64 Architecture 2
2.2 Energy Consumption model for Cyclops-64 4

3 Tiling Techniques for Energy Efficient Applications **5**
3.1 Matrix Multiplication 6
3.2 Finite Difference Time Domain 7

4 Experimental Evaluation **9**
4.1 Evaluation of the Energy Consumption Model 9
4.2 Evaluation of the Energy Efficient Tiling 10

5 Related Work **11**

6 Conclusions and Future Work **12**

List of Figures

1 C64 Architecture details 3
2 FDTD 1D Tiling for minimizing energy consumption 8
3 E_d vs. Γ with 150M Operations per Processor 10
4 Energy consumption (Static Es, Dynamic Ed and Total Et) vs Predicted model P and Measured M using different tilings for MM and FDTD 12

List of Tables

1 E_d consumed by memory operations for MM 8
2 E_d consumed by memory operations for FDTD 9
3 Energy Coefficients e and R^2 11

Abstract

Energy efficiency and power consumption have become an imperative requirement in Computer Architecture. The rising multi-core and many-core era has been motivated by the increasing demand of high performance computations restricted to a feasible power requirement. How to model the energy consumption of many-core architectures in order to propose techniques for the design of energy efficient applications is a topic of high interest in the community.

In this paper, we develop an energy consumption model for many-core architectures with software-managed memory hierarchy and we propose a general methodology for designing tiling techniques for energy efficient applications. The energy consumption model developed and the methodology proposed have the following characteristics: (1) The energy consumption model depends of the number and type of instructions executed and the total execution time of the application. (2) This model is scalable with the number of hardware thread units and considers stalls produced by data dependencies or arbitration of shared resources. (3) The methodology proposed is based on an optimization problem that produces optimal tiling and sequence of traversing tiles minimizing the energy consumed and parametrized by the sizes of each level in the memory hierarchy. (4) We show two different techniques for solving the optimization problem for two different applications: Matrix Multiplication (MM) and Finite Difference Time Domain (FDTD). Our experimental evaluation on a real IBM Cyclops-64 chip (C64) proves the accuracy of our energy consumption model and shows that the techniques proposed reduce the total energy consumption and also increase the power efficiency.

1 Introduction

The rapid progress of technology has made possible the integration of large number of processing cores on a single chip. As a consequence, parallel computing design has turned of special interest to the scientific community. Indeed, many-core and multi-core architectures have risen as the solution to most of the issues facing the field of high-performance computing. Energy efficiency and power consumption have become an imperative requirement, the design of new generation of exa-scale supercomputers is restricted to feasible power requirements [1, 2].

Integration of processors on a chip becomes challenging at different levels. From the point of view of semiconductor manufacturing process, new technologies and materials are needed for increasing the number of transistors per area. The integration of hundreds of processors on a single chip under area constraints and the significant increase on leakage current requires the redesign of traditional uniprocessor architectures with deep pipelines, complex branch prediction hardware and a cache-based memory hierarchy.

Particularly, traditional parallel programming methodologies have been focusing on improving performance and they assume cache-based parallel systems exploiting temporal locality. However, the data location and replacement in the cache is controlled by hardware making difficult a fine control and wasting energy [3, 4]. As a result, innovative architectures have arisen; one, unique on its type, is the IBM Cyclops-64 (C64) many-core-on-a-chip system. C64 contains 160 hardware Thread Units (TU) and it has a software-managed memory hierarchy where the data movement between different levels of the hierarchy is managed by the programmer. It saves the die area of hardware cache controllers and over-sized caches. Although this might complicate programming at their current stage, these systems provide more flexibility and opportunities to improve not only performance but also energy efficiency.

Several studies focusing on increasing the performance of a broad range of applications have been done on this architecture (e.g. Matrix Multiplication, LU decomposition, Fast Fourier Transform, etc) [5–8], but none of these techniques has directly considered the energy efficiency as a goal. Despite of that, some of them have provided evidence of the power efficiency of C64 [5, 9].

In this paper, we develop an energy consumption model for many-core architectures with software-managed memory hierarchy. The energy consumption model depends of the number and type of instructions executed and the total execution time of the application. We use the C64 many-core architecture to illustrate that our model is scalable with the number of hardware thread units and it considers stalls produced by data dependencies or arbitration of shared resources.

We also propose a general methodology for designing tiling techniques for energy efficient applications. The methodology proposed is based on an optimization problem that produces optimal tiling and sequence of traversing tiles minimizing the energy consumed and parameterized by the sizes of each level in the memory hierarchy. We show two different techniques for solving the optimization problem for two different applications: Matrix Multiplication (MM) and Finite

Difference Time Domain (FDTD). Our experimental evaluation uses a real IBM Cyclops-64 chip (C64) that proves the accuracy of our energy consumption model and shows that the techniques proposed reduce the total energy consumption and also increase the power efficiency.

The rest of this paper is organized as follows. In Section 2, we describe the C64 architecture and explain our energy consumption model. In Section 3, we analyze and propose solutions to the problem of designing tiling techniques for energy efficiency. In Section 4, we present the experimental evaluation of our energy consumption model and the tiling techniques proposed. Section 5 shows a review of related work. Finally, we conclude and present future work in Section 6.

2 Energy Consumption Model on a Many-Core Architecture

In this section we will propose a model for energy consumption on general purpose many-core architectures with software-managed memory hierarchy. Given our special interest on scalability, C64 seems the only one that has more than one hundred hardware threads and it has already been built. First, we will show a general review of the characteristics of C64 on section 2.1, we will emphasize the ones that concern to power consumption. Second, we will explain our energy consumption model for C64 on section 2.2.

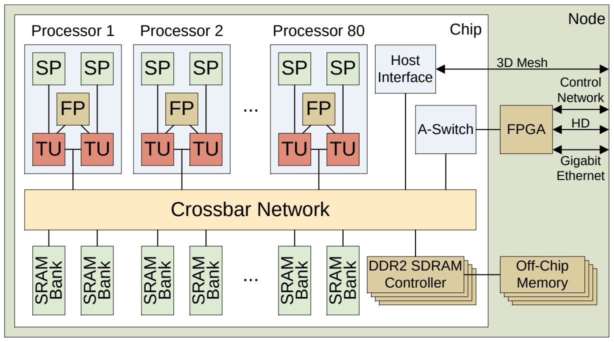
2.1 The IBM Cyclops-64 Architecture

Cyclops-64 (C64) is an innovative architecture developed by IBM, designed to serve as a dedicated petaflop computing engine for running high performance applications. A C64 chip is an 80-processor many-core-on-a-chip design, as can be seen in Figure 1a. Each processor is equipped with two thread units (TUs), one 64-bit floating point unit (FP) and two on-chip memory banks of 30kB each. It can issue one double precision floating point "Multiply and Add" instruction per cycle, for a total performance of 80 GFLOPS per chip when running at 500MHz.

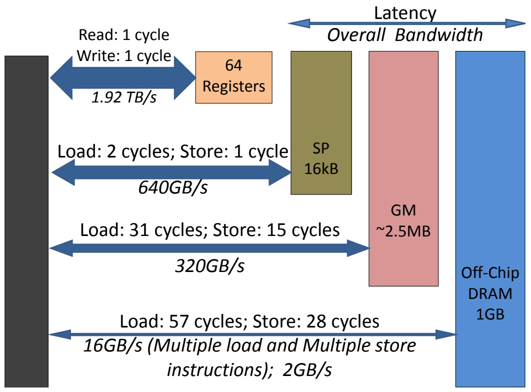
A processing node consist of a C64 chip using a $1.2V$ regulated power supply, external off-chip memory (DRAM) connected to a $1.8V$ regulated power supply and a small amount of external interface logic. A C64 chip has a 96-port crossbar network with bandwidth of 384GB/s that connects all TUs and on-chip memory banks [10].

A C64 chip has an explicit three-level memory hierarchy (scratchpad memory, on-chip memory (SRAM), off-chip memory (DRAM)), 16 instruction caches of 32kB each (not shown in the figure) and no data cache. The scratchpad memory (SP) is a configured portion of each on-chip SRAM bank which can be accessed with very low latency and energy by the TU it belongs to. The remaining sections of all on-chip SRAM banks consist of the on-chip global memory (GM), which is uniformly addressable from all TUs. As a summary, Figure 1b reflects the current size, latency (when there is no contention) and bandwidth of each level of the memory hierarchy.

Execution on a C64 chip is non-preemptive and there is no hardware virtual memory manager. The former means that the C64 micro-kernel will not interrupt the execution of a user application



(a) C64 Chip Architecture


Latency
Overall Bandwidth
Read: 1 cycle
Write: 1 cycle
1.92 TB/s
64 Registers
Load: 2 cycles; Store: 1 cycle
640GB/s
SP 16kB
Load: 31 cycles; Store: 15 cycles
320GB/s
GM ~2.5MB
Load: 57 cycles; Store: 28 cycles
16GB/s (Multiple load and Multiple store instructions); 2GB/s
Off-Chip DRAM 1GB

(b) Memory Hierarchy of C64

Figure 1: C64 Architecture details

unless an exception occurs. The latter means the three-level memory hierarchy of the C64 chip is visible to the programmer. In addition, the C64 instruction set architecture incorporates efficient support for thread level execution, hardware barriers and atomic in-memory operations.

Because C64 is a general purpose many-core architecture it has not been designed for energy efficiency and it does not have special features for saving power. For example, it is not possible to turn off cores not used or to slow down the clock rate of a set of cores or for the whole chip.

Despite the fact that the C64 Instruction Set Architecture (ISA) does not include any additional instructions that help reduce energy consumption we can group the instructions according to the hardware units they use and the complexity of the operation (reflected indirectly on the execution time if there is not contention). Furthermore, we can use these groups to build our energy consumption model. According with that, the taxonomy proposed for the ISA is:

- Logical Operations: And, or, etc.
- Integer Arithmetic Operations:
  - Simple: Add, sub.
  - Medium: Multiply.
- Floating Point Operations:
  - Simple: Add, sub.
  - Medium: Multiply, multiply and add.
- Memory Operations:
  - On Registers: Move, load immediate.
  - On SPM: load, store.

  - On SRAM: load, store.
  - On DRAM: load, store.

Some instructions not mentioned here. For example, branches can be included in the logical operations category, given the hardware resources and amount of work they require.

2.2 Energy Consumption model for Cyclops-64

Our energy consumption model has two main components. The first one is called *static energy* E_s, it comes from the leakage currents and other units that work continuously such as the clock. This component is a function of time t.

The second one is called *dynamic energy* E_d, it is the energy consumed by each functional unit in the execution of some instruction without the leakage component. It is related with the power consumption of transistors on registers and logic during switching, also called dynamic power.

Based on that, given a program Λ with K instructions I_j, the energy consumed can be expressed by:

$$E_T(\Lambda) = E_s(t) + \sum_{j=1}^{K} E_d\left(I_j\right) \tag{1}$$

Clearly, the model can be detailed even more because the power dissipated by leakage current is constant (given the absence of mechanism for reducing voltage or turning off functional units in C64) and also other units are always working at the same frequency (given the absence for changing this parameter). In other words E_s is linear with time.

In a similar way, instructions that use the same resources doing a similar amount of work, like the hierarchy explained on section 2.1, consume the same amount of energy. This linearity helps us to express our energy consumption model by:

$$E_T(\Lambda) = e_0 \cdot t + \sum_{i=1}^{M} e_i \cdot N\left(C_i\right) \tag{2}$$

Where e_0 is the static power dissipated, and e_i for $i = 1, \ldots, M$ is the energy consumed by one instruction of class C_i. The function $N(\cdot)$ counts the number of instructions in the program Λ that belong to a given class. This class can have only one instruction (e.g. when the kind of processing and the functional units that it uses are unique like integer multiplication) or multiple instructions (e.g. when they are similar in terms of amount of work and use the same resources like all the logical operations)

This model also considers the case of shared resources and overlapping, extremely important on many-core. First, each instruction represents the use of some resources for some task and it would take similar time. In a scenario of contention (e.g. the crossbar network for accessing memory), the amount of work made by the functional units will be the same but the time will increase. This will be reflected on the increase in the term that correspond to static energy. In a similar way, in the same processor multiple units can work in parallel (e.g. Floating Point Unit and Integer Unit) taking less time to complete the tasks compared with the sequential execution, as a result the term for static energy will decrease but the dynamic energy will remain similar. Even more important, for a chip with more than a hundred of processors, the dynamic energy terms reflect the energy per instruction regardless of whether it was executed in parallel with others or serially.

In Addition, it is natural to think than some instructions (or group of them) consumes more energy than others, some cases are:

- An operation that requires more computations than another of the same type. (e.g. integer multiplication vs. integer addition).
- An operation that uses a more complex hardware than another one. (e.g. floating point addition vs integer addition, on-chip memory operations vs integer operations).
- An operation that uses off-chip resources compared with one that only uses on-chip resources (e.g. load from DRAM vs load from SRAM).

3 Tiling Techniques for Energy Efficient Applications

In this section we will analyze the problem of designing tiling techniques for energy efficiency. Although instruction scheduling techniques are able to hide latency of operations, this kind of techniques are not useful here because *dynamic energy* E_d can not be hidden. We propose to find a feasible tiling that minimizes the total energy cost by minimizing the energy contribution of the most energy hungry instructions.

The optimization problem proposed is based on two facts: (1) Memory operations on off-chip memory are the most expensive in terms of energy, followed by on-chip memory operations. (2) There is not a dependency between different latencies for the same operation (e.g. contention of memory operations) and the dynamic energy it consumes. These two facts will be proved on section 4.1.

Our objective is to find the tiling T described by its parameters L and the sequence of traversing tiles S that minimize the consumed Dynamic Energy E_d on Γ processors by the subset of most energy hungry instructions I_E subject to the data stored D_H at each level H of the memory hierarchy cannot exceed the maximum memory size available $\mathrm{Mem}_{H\,\max}$ and the tiling allows parallel computation without communication between tiles. According to our model described on eq. (2), this Dynamic Energy E_d for a problem Λ is function of the number of instructions $N(\Lambda, I_j)$ with $I_j \in I_E$ and its energy coefficients e_j. This can be expressed as the optimization problem:

$$\begin{aligned} \min_{T(L,S)} \quad & E_d\left(I_E\right) = \sum_{I_j \in I_E} \left(e_j \cdot N\left(I_j\right)\right) \\ s.t. \quad & D_H\left(\Lambda, \Gamma, T\right) \leq \mathrm{Mem}_{H\max} \\ & T \text{ is parallel} \end{aligned} \tag{3}$$

Given the fact that memory operations are the most energy hungry instructions on most architectures and particularly on the C64 many-core architecture. The particular optimization problem using the Load LD and Store ST instructions for *off-chip* memory (DRAM) and *on-chip* memory (SRAM) is:

$$\begin{aligned} \min_{T(L,S)} \quad & e_1 N(LD_{\text{dram}}) + e_2 N(ST_{\text{dram}}) + e_3 N(LD_{\text{sram}}) + e_4 N(ST_{\text{sram}}) \\ s.t. \quad & D_H\left(\Lambda, \Gamma, T\right) \leq \mathrm{Mem}_{H\max} \\ & T \text{ is parallel} \end{aligned} \tag{4}$$

Where $N(LD)$ and $N(ST)$ are also function of Λ, Γ, T.

The optimization problem described by 3 and 4 cannot be easily solved. Even more, there is not guarantee of analytical solution. The following subsections will show two approaches for solving these kind of optimization problems for two kind of applications: Matrix Multiplication (MM) and Finite Difference Time Domain (FDTD).

3.1 Matrix Multiplication

Despite Matrix Multiplication (MM) algorithms have been studied extensively, the many-core architecture design space has not yet been explored in detail. MM is extremely important on scientific applications that use linear algebra. Our target operation is the multiplication of dense square matrices $A \times B = C$, each of size $m \times m$ using algorithms of running time $O(m^3)$. We will focus on matrices that fit in on-chip memory SRAM and the memory operations will be load and store from SRAM to registers. For this case, the optimization problem on 4 becomes:

$$\begin{aligned} \min_{T(L,S)} \quad & e_3 N(LD_{\text{sram}}) + e_4 N(ST_{\text{sram}}) \\ s.t. \quad & R\left(\Lambda, \Gamma, T\right) \leq R_{\max} \\ & T \text{ is parallel} \end{aligned} \tag{5}$$

An optimal partition for a load-balanced distribution between processors P assumes blocks $C' \in C$ of size $n \times n$ $\left(n = \frac{m}{\sqrt{\Gamma}}\right)$. Each block is subdivided in tiles $C'_{i,j} \in C'$ of size $L_2 \times L_2$. Based on the data dependencies, the required blocks $A' \in A$ and $B' \in B$ of sizes $n \times m$ and $m \times n$ are subdivided in tiles $A'_{i,j} \in A'$ and $B'_{i,j} \in B'$ of sizes $L_2 \times L_1$ and $L_1 \times L_2$ respectively. Each tile can be calculate using $C'_{i,j} = \sum_{k=1}^{m/L_2} A'_{i,k} \cdot B'_{k,j}$.

The number of loads and stores can be calculated analytically for each one of the 6 alternatives for traversing tiles that can be summarize on two sequences S_1, S_2. The specific optimization problem now becomes:

$$\begin{aligned} \min_{\substack{L\in\{L_1,L_2\},\\ S\in\{S_1,S_2\}}} \quad & f(m,\Gamma,L,S) = \begin{cases} \frac{2e_3}{L_2} m^3 + e_4 m^2 & \text{if } S = S_1 \\ \left(\frac{e_3+e_4}{L_1} + \frac{e_3}{L_2}\right) m^3 + e_3\left(\sqrt{\Gamma}-1\right) m^2 & \text{if } S = S_2 \end{cases} \\ \text{s.t.} \quad & 2L_1L_2 + L_2^2 \le R_{\max}, \quad L_1, L_2 \in \mathbb{Z}^+ \end{aligned} \tag{6}$$

Analyzing the piecewise function f, it can be easily shown that S_1 sequence has an smaller objective function than S_2 under the conditions $\frac{e_4}{e_3} \le \sqrt{\Gamma} - 1$ and $\frac{L_2}{L_1} \ge \frac{e_3}{e_3+e_4}$. The first one is easily satisfied if Γ is big enough, the second one can be satisfied when $L_2 \ge L_1$ and it can be verified with the solution.

We will solve the integer optimization problem using the branch and bound technique. Since f only depends on L_2, we minimize the function f by maximizing L_2. Given the constraint, L_2 is maximized by minimizing L_1. Thus $L_1 = 1$, we solve the optimum L_2 in the boundary of the constraint and round off it. The solution of Eq. (6) is:

$$L_1 = 1,\ L_2 = \left\lfloor \sqrt{1 + R_{\max}} - 1 \right\rfloor \tag{7}$$

The solution satisfies the constraints and also proves the hypothesis $L_2 \ge L_1$, finishing the branch and bound process. This result is not completely accurate, since we assumed that there are not remainders when we divide the matrices into blocks and subdivide the blocks in tiles. Despite this fact, they can be used as a good estimate.

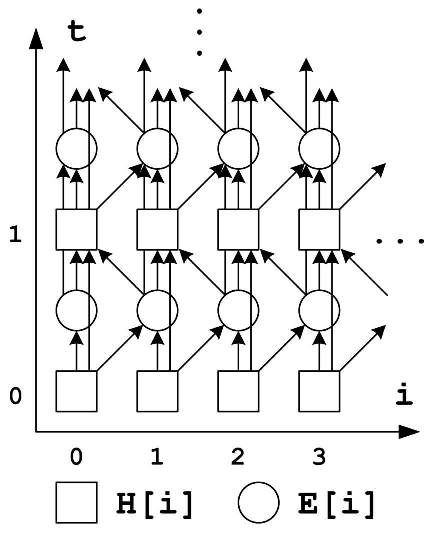
For comparison purposes, C64 has 63 registers and we need to keep one register for the stack pointer, pointers to A, B, C matrices, m and stride parameters, then $R_{\max} = 63 - 6 = 57$ and the solution of Eq. (7) is $L_1 = 1$ and $L_2 = 6$. Table 1 summarizes the results in terms of dynamic energy consumed by LDs and STs for the tiling proposed and other 2 options that fully utilizes the registers and have been used in practical algorithms: inner product of vectors ($L_1 = 28$ and $L_2 = 1$) and square tiles ($L_1 = L_2 = 4$). As a consequence of using sequence S_1, the dynamic energy of STs is equal in all tiling strategies. As expected, the tiling proposed consumes minimum energy: approximately 6 times less than the inner product tiling and 1.5 times less than the square tiling.

3.2 Finite Difference Time Domain

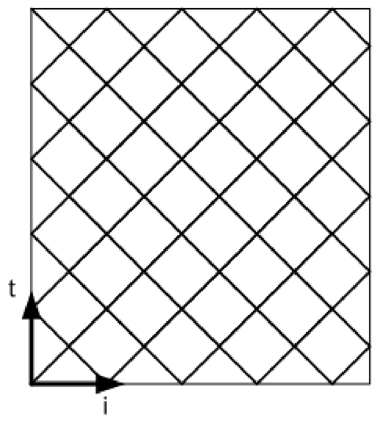
The Finite Difference Time Domain (FDTD) [11] technique is a common algorithm to simulate the propagation of electromagnetic waves through direct solution of Maxwell's Equations. FDTD

Table 1: E_d consumed by memory operations for MM

Memory Operations	Inner Product	Square	Optimal
Loads	$2e_3m^3$	$\frac{e_3}{2}m^3$	$\frac{e_3}{3}m^3$
Stores	e_4m^2	e_4m^2	e_4m^2



(a) DDG for FDTD 1D



(b) Diamond Tiling

Figure 2: FDTD 1D Tiling for minimizing energy consumption

was chosen to illustrate the techniques presented here since it is easy to understand, it is widely used, and it can be easily written for multiple dimensions. Specifically, we will study FDTD in one dimension i of size m and q time steps. The data is read directly from off-chip memory with tiles on on-chip memory. For this case, the optimization problem on eq. 4 becomes:

$$\begin{aligned} \min_{T(L,S)} \quad & e_1N(LD_{\text{dram}}) + e_2N(ST_{\text{dram}}) \\ s.t. \quad & \text{Mem}_{\text{sram}}(\Lambda, \Gamma, T) \leq \text{Mem}_{\text{max}} \\ & T \text{ is parallel} \end{aligned} \tag{8}$$

The solution of this problem is based on the analysis of its Data Dependency Graph (DDG) that can be detailed on Figure 2a. Our solution is inspired by [12] where they find the tiling that maximize the data reuse. Because the number of useful computations can not be decreased by the tiling. For a FDTD problem of size fixed size, maximize the data reused is equivalent to minimize the number of memory operations $N(LD_{\text{dram}}) + N(ST_{\text{dram}})$. In addition, given the regularity of the DDG, a tiling that saves energy will not load extra data for doing extra computations. It means that the number of loads and stores will be the same. In that order the ideas, the *diamond tiling* showed on Figure 2b solves the optimization problem given by eq. 8

Table 2 summarizes the results in terms of dynamic energy consumed by LDs and STs for the tiling proposed and other 3 well-known techniques [13]. The unit for the tile size L is the node $E[i], H[i]$. Clearly, Diamond tiling for FDTD has the smallest coefficients.

Table 2: E_d consumed by memory operations for FDTD

Memory Operations	Naive	Split	Overlapped	Diamond
Loads	$e_1 qm$	$\frac{9e_1}{2L} qm$	$\frac{9e_1}{L} qm$	$\frac{2e_1}{L} qm$
Stores	$e_2 qm$	$\frac{9e_2}{2L} qm$	$\frac{3e_2}{L} qm$	$\frac{2e_2}{L} qm$

4 Experimental Evaluation

This section describes the experimental evaluation of the proposed energy consumption model given in section 2.2 and the tiling techniques for energy efficiency analyzed in section 3.

4.1 Evaluation of the Energy Consumption Model

The energy coefficients e_i where obtained using measurements of current and voltage from the power supplies in a real chip. The instantaneous power $P[t]$ at time t can be calculated using $P[t] = v_1[t] \cdot i_1[t] + v_2[t] \cdot i_2[t]$, the average power $\bar{P}$ is estimated by the mean of several samples of $P[t]$ and the total energy consumed is $E_T = \bar{P} \cdot t$.

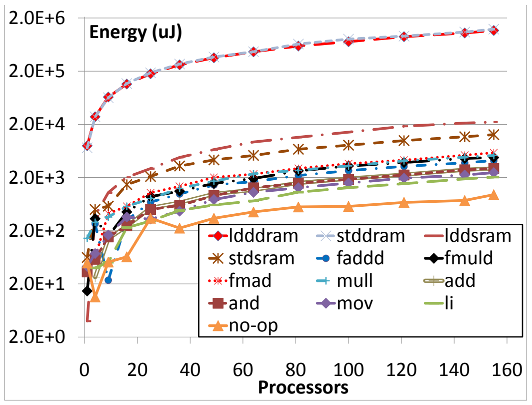
A test bed for the ISA of C64 was created for the estimation of the energy coefficients e_i of (2). The test bed include multiple programs, each one with a known number of instructions for a subset of the ISA. The estimation of $e_0 = 63.11W$ was straight forward calculated only measuring the consumption of the system on standby. Notice that while e_0 is estimated in Watts, e_i for $i > 0$ is estimated in Joules/Instruction.

The dynamic energy E_d for a program Λ running in parallel on Γ processors with a fixed number of instructions of class I_j per processor can be estimated by eq. 9

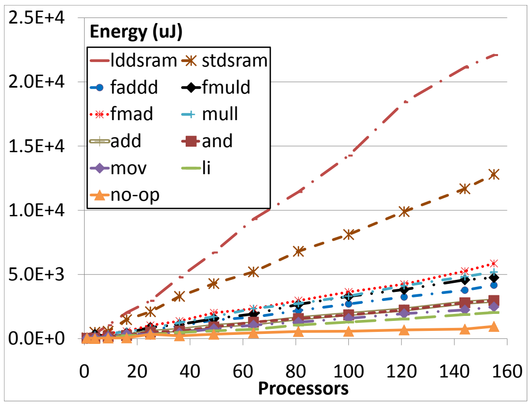
$$E_d(\Lambda, I_j, \Gamma) = \left(\bar{P} - e_0\right) \cdot t \tag{9}$$

The results for a representative subset of the ISA are shown on Figure 3. As shown on Figure 3a, load and store on DRAM (ldddram, stddram) are the most energy hungry, followed by load and store on SRAM (lddsram, stdsram), the difference of energy consumption between DRAM and SRAM operations is almost 2 orders of magnitude. Figure 3b proves the linearity of energy consumption with Γ. It details that after memory operations, floating point operations (fmaddd, fmuld and fmad) and difficult integer operations (mull) consumes similar energy. Integer, logical and register movement operations (add, and, mov, li) are on the bottom of the list. The instruction that consumes less is no-op as expected.

The remainder energy coefficients e can be extrapolated using a linear regression from the E_d estimated for each instruction. We used a model with intercept at origin given the assumption



(a) Overall comparison of selected ISA



(b) Comparison for On-chip Mem. Op., FPU Op. and Integer/Logical Op.

Figure 3: E_d vs. Γ with 150M Operations per Processor

that no dynamic energy is consumed on standby. The resultant coefficients e for a subset of the ISA is shown on Table 3. The table also includes the coefficients of determination R^2 for measuring the variability between the data and the model proposed. As expected, a linear approximation with the number of processors models accurately E_d, its coefficients R^2 are really close to 1, it corroborates that there is not dependency between the latency of the operation and the dynamic energy consumed. Some additional aspects to highlight are: (1) Instead DRAM operations consume similar energy, a load from SRAM consumes almost twice the energy of an store to SRAM. (2) Despite the floating point fused-multiply-add (fmad) consumes a little bit more energy than a simple floating point multiply (fmuld) or floating point add (faddd), notice that one fmad executes a multiply and an addition. At the end, an fmad saves around 63% of energy compare with separates fmuld and faddd. (3) Integer and floating point multiplication cost similar, the same is true for logical and simple integer operations. The last two observations confirms the high correlation between the energy consumption of an instruction and the related hardware and functional units the instruction requires.

4.2 Evaluation of the Energy Efficient Tiling

We will use the two applications explained before (MM and FDTD) for showing the advantages of the tilings that solve the optimization problems of section 3. First, we will compare the estimated energy consumption using the coefficients of section 4.1 with the measured energy based on voltage and current on the real chip. Second, we will compare energy consumption of the tiling proposed with other well known tiling techniques.

For MM we use a matrix size that fits on SRAM, we compare our approach *(OptT)* with the register tiling based on dot product *(DPT)*. Both methods uses assembly for taking advantage of the complete register file. For FDTD, the tile size is the maximum possible that fits on

Table 3: Energy Coefficients e and R^2

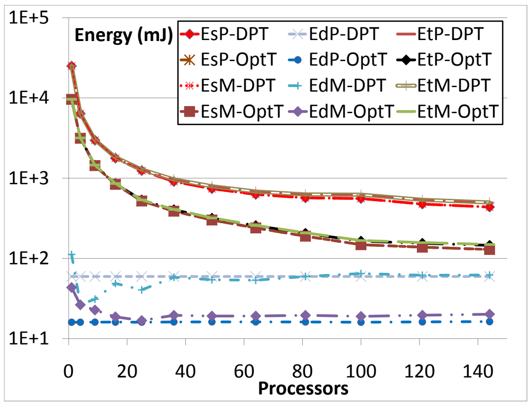
Instruction	**e[pJ/Operation]**	R^2
ldddram	48924.10	0.999
stddram	51488.99	0.998
lddsram	964.65	0.997
stdsram	548.31	0.999
fmad	245.27	0.997
faddd	178.30	0.995
fmuld	210.15	0.996
mull	225.43	0.998
add	127.65	0.998
and	126.69	0.998
mov	105.48	0.996
li	86.01	0.997
no-op	39.66	0.936

SPM, we compare our diamond tiling *(DmT)* with 3 well-known techniques: A rectangular tiling (naive) *(NT)*, the overlapped tiling *(OT)* that uses redundant computations in order to tile time and space dimensions and split tiling *(ST)* that uses multiple shapes for fully partitioning the iteration space [13].

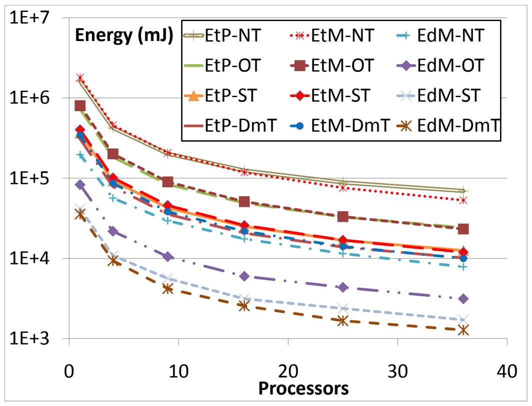
Figure 4a compares the energy consumption measured with the energy predicted by our model for the MM application. We can see how the predictions are highly close to the measured value for the dynamic and static components. The average error of our model for E_d and E_T is 26.6% and 0.82% respectively. We also noticed how the tiling proposed decreases substantially the dynamic and total energy consumption in 56.52% and 61.21% on average. An interesting result that can be extrapolated from the measurements of performance and power is that the power efficiency [MFLOPS/W] increases between 2.62 and 4.13 times for this test example. For the FDTD application, figure 4b shows the effectiveness of diamond tiling for decreasing the total and dynamic energy with respects to the other tiling techniques. The total average energy reduction was 81.26%, 57.27% and 15.69% compared with split tiling, overlapped tiling and naive tiling respectively. Also our energy consumption model is accurate to the real behavior of the application, the average error is 7.3% for E_T.

5 Related Work

Energy consumption on traditional architectures has been extensively studied [14]. Most of the research has focused on systems with caches [15]. Accurate but highly complex models and techniques for reducing energy consumption has been proposed for uniprocessor architectures. They uses precise information about the hardware and are based on elaborated instruction scheduling [14,16]. As a consequence the extrapolation to many-core architectures is highly difficult and not scalable with the number of hardware threads. Energy efficiency on multiprocessors has been focused on the hardware design, including hardware features like power saving off-chip memory



(a) MM with $m = 300$



(b) FDTD with $m = 100k$ and $q = 500$

Figure 4: Energy consumption (Static Es, Dynamic Ed and Total Et) vs Predicted model P and Measured M using different tilings for MM and FDTD

or dynamic voltage selection [17].

Methodologies and techniques for increasing performance on many-core architectures with software-managed memory hierarchy have been a promising topic of research [5–8]. Some of them have shown empirical evidence about increasing the power efficiency [5, 9].

6 Conclusions and Future Work

In this paper, we develop an energy consumption model for many-core architectures with software-managed memory hierarchy. We validate the accuracy of this model with the C64 many-core architecture and we show the model depends of the number and type of instructions executed and the total execution time of the application. An advantage is that this model is scalable with the number of hardware thread units and consider stalls produced by data dependencies or arbitration of shared resources.

We also propose a general methodology for designing tiling techniques for energy efficient applications. The methodology proposed is based on an optimization problem that produces optimal tiling and sequence of traversing tiles minimizing the energy consumed and parametrized by the sizes of each level in the memory hierarchy. We also show two different techniques for solving the optimization problem for two different applications: Matrix Multiplication (MM) and Finite Difference Time Domain (FDTD). Our experimental evaluation shows that the techniques proposed reduce the total energy consumption effectively, decreasing the static and dynamic component. The average energy saving for MM is 61.21%, this energy saving is 81.26% for FDTD compared with the naive tiling.

Future work includes to extend the model and methodology proposed to other algorithms (e.g. Linpack) and study the impact of dynamic scheduling techniques in the energy consumption.

We also are interested on the relation between optimum tiling for increasing performance and optimum tiling for energy efficiency.

References

[1] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, and K. Yelick, “Exascale computing study: Technology challenges in achieving exascale systems,” 2008. [Online]. Available: www.cse.nd.edu/Reports/2008TR-2008-13.pdf

[2] J. Torrellas, “Architectures for extreme-scale computing,” *Computer*, vol. 42, no. 11, pp. 28 –35, nov. 2009.

[3] D. Callahan and A. Porterfield, “Data cache performance of supercomputer applications,” in *Supercomputing ’90: Proceedings of the 1990 ACM/IEEE conference on Supercomputing*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1990, pp. 564–572.

[4] M. Kondo, H. Okawara, H. Nakamura, T. Boku, and S. Sakai, “Scima: a novel processor architecture for high performance computing,” in *High Performance Computing in the Asia-Pacific Region, 2000. Proceedings. The Fourth International Conference/Exhibition on*, vol. 1, 2000, pp. 355–360 vol.1.

[5] E. Garcia, I. E. Venetis, R. Khan, and G. Gao, “Optimized Dense Matrix Multiplication on a Many-Core Architecture,” in *Proceedings of the Sixteenth International Conference on Parallel Computing (Euro-Par 2010), Part II*, ser. Lecture Notes in Computer Science, vol. 6272. Ischia, Italy: Springer, 2010, pp. 316–327.

[6] L. Chen, Z. Hu, J. Lin, and G. R. Gao, “Optimizing the Fast Fourier Transform on a Multi-core Architecture,” in *IEEE 2007 International Parallel and Distributed Processing Symposium (IPDPS ’07)*, Mar. 2007, pp. 1–8.

[7] Z. Hu, J. del Cuvillo, W. Zhu, and G. R. Gao, “Optimization of Dense Matrix Multiplication on IBM Cyclops-64: Challenges and Experiences,” in *12th International European Conference on Parallel Processing (Euro-Par 2006)*, Dresden, Germany, Aug. 2006, pp. 134–144.

[8] I. E. Venetis and G. R. Gao, “Mapping the LU Decomposition on a Many-Core Architecture: Challenges and Solutions,” in *Proceedings of the 6th ACM Conference on Computing Frontiers (CF ’09)*, Ischia, Italy, May 2009, pp. 71–80.

[9] E. Garcia, R. Khan, K. Livingston, I. E. Venetis, and G. Gao, “Dynamic percolation - mapping dense matrix multiplication on a many-core architecture,” *CAPSL Technical Memo 98*, June 2010. [Online]. Available: ftp://ftp.capsl.udel.edu/pub/doc/memos/memo098.pdf

[10] M. Denneau and H. S. Warren Jr., “64-bit Cyclops: Principles of Operation,” IBM Watson Research Center, Yorktown Heights, NY, Tech. Rep., April 2005.

[11] K. Yee, “Numerical solution of inital boundary value problems involving maxwell’s equations in isotropic media,” *Antennas and Propagation, IEEE Transactions on*, vol. 14, no. 3, pp. 302–307, May 1966.

[12] D. Orozco, E. Garcia, and G. Gao, “Locality optimization of stencil applications using data dependency graphs,” 2010.

[13] S. Krishnamoorthy, M. Baskaran, U. Bondhugula, J. Ramanujam, A. Rountev, and P. Sadayappan, “Effective automatic parallelization of stencil computations,” *SIGPLAN Not.*, vol. 42, no. 6, pp. 235–244, 2007.

[14] F. Yao, A. Demers, and S. Shenker, “A scheduling model for reduced cpu energy,” oct. 1995, pp. 374 –382.

[15] H. Hanson, M. Hrishikesh, V. Agarwal, S. Keckler, and D. Burger, “Static energy reduction techniques for microprocessor caches,” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 11, no. 3, pp. 303 – 313, jun. 2003.

[16] S. Lee, A. Ermedahl, and S. L. Min, “An accurate instruction-level energy consumption model for embedded risc processors,” in *LCTES ’01: Proceedings of the ACM SIGPLAN workshop on Languages, compilers and tools for embedded systems.* New York, NY, USA: ACM, 2001, pp. 1–10.

[17] A. Andrei, P. Eles, Z. Peng, M. Schmitz, and B. Hashimi, “Energy optimization of multiprocessor systems on chip by voltage selection,” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 15, no. 3, pp. 262 –275, mar. 2007.