LEBANESE AMERICAN UNIVERSITY

KT2C: K-way Thermal Chip Clustering

By

Ahmad Al Kawam

A thesis
Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Engineering

School of Engineering
June 2014

LAU
الجامعة اللبنانية الأميركية
Lebanese American University

Lebanese American University

School of Engineering ; Byblos Campus

THESIS APPROVAL FORM

Student Name: Ahmad Al Kawam I.D. #: 201105416

Thesis Title : KT2C: Kanway Thermal Chip Clustering

Program: Computer Engineering Mse.

Department: Electrical and Compter

School: Engineering

The undersigned certify that they have examined the final electronic copy of this thesis and approved it in Partial Fulfillment of the requirements for the degree of:

M. S. in the major of Computer Engineering

Thesis Advisor's Name IYAD OUNAIS Signature Signatures Redacted Date 11/6/14

Committee Member's Name Zahi Nakad Signature Signatures Redacted ate 11 June 2014

Committee Member's Name DANI TANNIR Signature Signatures Redacted Date 11/6/2014

THESIS COPYRIGHT RELEASE FORM

LEBANESE AMERICAN UNIVERSITY NON-EXCLUSIVE DISTRIBUTION LICENSE

By signing and submitting this license, you (the author(s) or copyright owner) grants to Lebanese American University (LAU) the non-exclusive right to reproduce, translate (as defined below), and/or distribute your submission (including the abstract) worldwide in print and electronic format and in any medium, including but not limited to audio or video. You agree that LAU may, without changing the content, translate the submission to any medium or format for the purpose of preservation. You also agree that LAU may keep more than one copy of this submission for purposes of security, backup and preservation. You represent that the submission is your original work, and that you have the right to grant the rights contained in this license. You also represent that your submission does not, to the best of your knowledge, infringe upon anyone's copyright. If the submission contains material for which you do not hold copyright, you represent that you have obtained the unrestricted permission of the copyright owner to grant LAU the rights required by this license, and that such third-party owned material is clearly identified and acknowledged within the text or content of the submission. IF THE SUBMISSION IS BASED UPON WORK THAT HAS BEEN SPONSORED OR SUPPORTED BY AN AGENCY OR ORGANIZATION OTHER THAN LAU, YOU REPRESENT THAT YOU HAVE FULFILLED ANY RIGHT OF REVIEW OR OTHER OBLIGATIONS REQUIRED BY SUCH CONTRACT OR AGREEMENT. LAU will clearly identify your name(s) as the author(s) or owner(s) of the submission, and will not make any alteration, other than as allowed by this license, to your submission.

Name: Ahmad Al Kawam

Signature: Ahmad Al Kawam

Date: 23/5/2014

PLAGIARISM POLICY COMPLIANCE STATEMENT

I certify that:

- I have read and understood LAU's Plagiarism Policy.
- I understand that failure to comply with this Policy can lead to academic and disciplinary actions against me.
- This work is substantially my own, and to the extent that any part of this work is not my own I have indicated that by acknowledging its sources.

Name: Ahmad Al Kawam

Signature: Ahmad Al Kawam

Date: 23/5/2014

ACKNOWLEDGMENT

This project would not have been possible without the support of many people.

I would like to express my sincere gratitude towards my advisor, Dr. Iyad Ouaiss, for his never-ending guidance and support.

I am also deeply grateful to Dr. Zahi Nakad, and Dr. Dani Tannir, for being on my thesis committee.

And finally, many thanks to my parents and friends who endured this long process with me, always offering support and love.

KT2C: K-way Thermal Chip Clustering

Ahmad Al Kawam

ABSTRACT

We introduce the K-way Thermal Chip Clustering (KT2C) algorithm; a VLSI chip partitioning algorithm that is used to reduce the chip's temperature and prevent the formation of hotspots. The clustering algorithm is integrated in the design flow between the stages of binding and floor-planning. KT2C uses power and area information from a component library to identify the components that will most probably result in hotspots. These components are labeled as thermal centers which then form independent clusters. The rest of the components are distributed among these clusters with the goal of reducing the center's temperature. The algorithm then iterates attempting to produce a uniform temperature distribution among clusters while reducing wire-length. Upon convergence, the components undergo two stages of floor-planning. At the intra-cluster floor-planning stage, the positions of the components inside each cluster are determined, whereas in the inter-cluster floor-planning stage the position of each cluster block on the chip is determined. The temperature profile of the resulting chip is calculated using the thermal simulator "Hotspot" and is compared to the temperature profile of a design achieved using a regular design flow. The results show significant reduction in both average and peak temperatures at the cost of a small increase in area. Furthermore, when optimizing for wire-length, KT2C produced a remarkable reduction in total wire-length which surpassed the reduction achieved by the non-clustering solution. Finally, a parallel design of the KT2C design flow was proposed and the conducted timing analysis showed that large speedups could be achieved if parallelism was implemented.

Keywords: Thermal-aware VLSI Design, VLSI Partitioning, Cluster Analysis, Thermal-aware VLSI Clustering, Chip Hotspot Minimization, Chip Temperature Reduction.

TABLE OF CONTENTS

Chapter **Page**

I- Introduction 1

II- Literature Review 7

III- Related Topics 18

3.1 High Level Synthesis 18

3.2 A GPU-Parallel Bee Colony Optimization for Design Space Exploration in High-Level Synthesis 30

3.3 CMOS Power Calculation 39

3.4 Clustering Analysis 45

3.5 Floor-planning 48

3.6 Thermal Modeling 52

IV- K-way Thermal Chip Clustering Algorithm 59

4.1 Overview of the new design flow 59

4.2 Algorithm Description 61

4.3 Design Implementation 66

4.4 Parallelization Study 69

4.5 Component Library Synthesis 74

V- Final Results 78

VI- Conclusion 84

Bibliography 86

Chapter One

Introduction

VLSI chip design has witnessed several tremendous changes in recent years with the increase in the demand for computational speed and capacity. Feature size has dropped down into the sub-100nm level; chip capacity has exceeded the billions of transistors limit, and Gigahertz frequencies have become a commonplace. Computing platforms are composed of multi-processors with each microprocessor incorporating several cores. Recent developments have further increased the integration density by the introduction of System on Chip (SoC) systems which, in addition to cores include intellectual property (IP) modules on the same chip. On the transistor level, area and delay constraints has geared the VLSI design community towards three dimensional ICs leading to a rapid increase in integration density.

This increase in integration density had a substantial effect on the power density of VLSI chips causing it to soar at a remarkable rate causing power density to double about every three years. This rate is expected to increase in the following technology generations due to high operating frequencies and tiny feature sizes (The International Technology Roadmap for Semiconductors, 2001). The power density has reached 50W/cm2 for the 100nm technology and is estimated at 100W/cm^2 for technologies below 50nm (The International Technology Roadmap for Semiconductors, 2003). Since power consumed by the chip is entirely dissipated

as heat, this exponential increase in power consumption has caused a phenomenal increase in temperature (Skadron, et al., 2003). Operating temperatures of current chips has exceeded 100°C with intra-chip temperature differentials ranging between 10 °C and 20 °C (Tsai & Kang, 2000) as shown in Table 1.

Table 1 Tech generation, chip size, and maximum power according to year (Tsai & Kang, 2000)

Year	1999	2001	2003	2006	2009	2012
Tech Generation (nm)	180	150	130	100	70	50
Chip Size (mm2)	340	385	430	520	620	750
Maximum Power (W)	90	110	130	160	170	175

This increase in operating temperatures and intra-chip temperature fluctuations has very harmful consequences on the chip. More than half of the failures in 2-D ICs are attributed to thermal effects in the die (Gurrum, Joshi, King, Ramakrishna, & Gall, 2008). This is attributed to the exponential degradation of the electro migration (EM) lifetime with linear increase in temperature. The impact of temperature on reliability can be viewed in the Arrhenius equation where the mean time for failure (MTF) is exponentially decreased by T, the operating temperature (Huang, et al., 2004) as illustrated in Figure 1. Moreover, temperature has a strong impact on timing. As temperature increases, transistor speed decreases due to the degradation of carrier mobility leading to a 4% drop in current drive capability for every 10 °C. As for interconnects, the resistivity of copper is significantly larger at high temperatures. This resistivity is 47% larger at 120°C that it is at 20 °C. This increase is reflected on an increase in Elmore (interconnect) delay, which experiences a 5% increase for every 10 °C. Finally, high temperature is a major contributor to the increase in leakage power which could be orders of magnitude larger than in normal operation (Tsai & Kang, 2000) (Huang, et al., 2004).

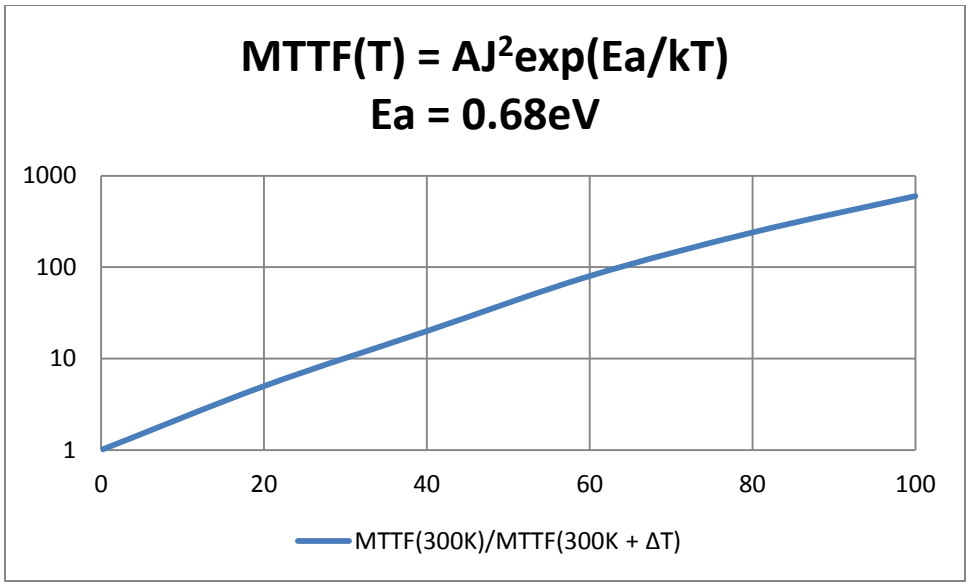

Figure 1 MTTF as a function of temperature (Tsai & Kang, 2000)

In order to counterpart the severe effects of high temperature, several methods have been developed and integrated into recent IC technologies. The first obvious strategy was to remove the dissipated heat through cooling techniques such as using heat sinks, fans, and even liquid cooling for high performance systems. As technologies became more power hungry, more effective techniques were needed. Power minimization techniques were used, such as reducing chip and package capacitance. This was achieved by using processes with partially or fully depleted wells and using special interconnect substrates such as Multi-Chip Modules (MTM) (Pedram & Vaishnav, 1997). Several dynamic thermal management (DTM) techniques were also developed. These techniques use a run-time feedback from the integrated circuit in order to manage temperature controlling factors such as operating voltage and frequency. For example, in the Pentium4 processor, Intel implemented a real-time temperature management technique that senses the processor's temperature and applies a technique called clock gating to reduce the temperature when needed (Gunther, Binns, Carmean, & Hall, 2001). This clock gating

technique disables the clocking signal when a temperature threshold is exceeded and enables it when the temperature reaches the desired value. Other DTM techniques use voltage and frequency scaling in order to decrease the dynamic power of the chip which highly depends on operating voltage and frequency. Other techniques build on the above by forming voltage and frequency islands in the chip in order to be able to drop the dynamic power, and hence the temperature, in the desired area only (Martonosi & Brooks, 2001) (Cao, Krusius, Korhonen, & Fisher, 1998) (Huang, Renau, Yoo, & Torellas, 2000). These techniques, although effective, slow down the operation of the processor and might cause it to exceed its computational deadlines (Sankaranarayanan, Velusamy, Stan, L, & Skadron, 2005).

In addition to runtime techniques, several works in the literature propose thermal optimization techniques during design time. Since the chip's operation is highly dependent on the decisions made in the design process, it is advantageous to embed thermal-awareness in the design process. In addition to the total temperature, the chip's temperature is reliant on its power distribution. The main power sources in the chip are its cellular blocks; this made Cell-Placement the natural starting point for temperature aware design. Placement is considered a low-level stage in the design flow which made it possible to drive a feedback from the thermal simulator to guide the cell placement with minimal runtime increase (Tsai & Kang, 2000).

The development of compact thermal models (CTMs) enabled taking thermal aware design into higher levels of abstraction. This is due to their ability to provide a fast and relatively accurate estimate of the chip's temperature (Skadron, et al., 2003). Thermal aware floor-planning is used mainly to optimize the floor-plan of microprocessors by modifying the positions of heat

sensitive modules such as memory, and heat intensive modules such as arithmetic logic units (ALUs) (Sankaranarayanan, Velusamy, Stan, L, & Skadron, 2005). Furthermore, different techniques for thermal-aware binding were proposed which use thermal feedback to distribute the operations among different functional units and registers with an effort to reduce switching activities and to provide an even power and thermal distribution leading to the minimization of hotspots (Mukherjee, Memik, & Memik, Peak temperature control and leakage reduction during binding, 2005). Some techniques integrate thermal awareness with stages as high as resource allocation and operation scheduling in order to minimize chip temperature.

In addition to thermal simulation, total power and its distribution could be used as a good indicator for temperature due to the strong link between the two (Tsai & Kang, 2000). Though, not all power management techniques have an effect on temperature. Power aware algorithms targeting temperature reduction are designed using different techniques than those used to prolong battery lifetime and regulate peak power (Skadron, et al., 2003). Irregularity in power dissipation causes the formation of regions whose temperature elevates faster that total chip temperature. The main goal behind temperature aware power management algorithms is to eliminate, or minimize the formation of these "Hotspots" by aiming for a uniform power distribution across the chip.

In this thesis, we introduce the K-way Thermal Chip Clustering (KT2C) algorithm, a VLSI chip partitioning algorithm used to reduce chip temperature and prevent the formation of hotspots. Partitioning has been used in the past three decades as a "divide and conquer" approach in designing circuits. Floor-planning tools use different partitioning techniques such as hierarchical

partitioning to identify highly interconnected components in order to speed up floor-planning while minimizing wire-length. Clustering on the other hand is a technique in machine learning applied on large datasets in order to group data points together into clusters according to their similarity. Our clustering algorithm is integrated in the design flow between the stages of binding and floor-planning. KT2C uses the power and area data of a component library to identify the components that will most probably form hotspots. These components are labeled as thermal centers and for separate clusters. The rest of the components are then distributed amongst these clusters with the goal of reducing the thermal temperature. The algorithm then iterates attempting to produce a uniform temperature distribution among clusters while reducing wire-length.

In the next chapter the literature is surveyed for works targeting thermal chip reduction at different levels. The improvement due to applying each technique is then reported.

Chapter Two

Literature Review

Thermal optimization in VLSI has been targeted using approaches from different levels in the design flow. Since power distribution is a main contributor to determining chip temperature, cell placement and floor-planning have been extensively studied for thermal optimization goals. Furthermore, because these stages are close to the physical design level, temperature estimations are more accurate and including thermal costs into their objective functions is much simpler. One of the early attempts at thermally optimized placement is the technique presented in (Chu & Wong, 1998). The authors introduce Matrix Synthesis Placement (MSP) in which they formulate placement as matrix synthesis problem: The chip is divided into a grid with each cell representing an element in the matrix. The temperature level in each cell is represented by a nonnegative number in the relative matrix element. MSP dictates that the sum of the elements in a sub matrix of the general matrix should not exceed a certain limit. The sub matrix with the highest sum represents the hottest area in the chip. The authors proposed three techniques to place the cells while satisfying the condition of MSP. Their algorithm was tested on the MCNC benchmarks and the results show an average of 35.6% decrease in the overall heat distribution in the chip while producing an average of 0.6% increase in wiring compared to a traditional thermal unaware placement algorithm. Another well referenced article in the literature is the work presented in (Tsai & Kang, 2000) in which the authors propose a thermal model they integrated into a standard cell placement tool and a macro cell

placement tool. For the cell placement tool, the temperature constraint is used to compute the corresponding power distribution constraint. For each move, the tool updates the power distribution, checks if the constraints are met, and includes the power distribution in cost evaluation function. As for the macro cell placement tool, the temperature model is used to update the chip profile after each move. The temperature is then included in the cost evaluation function. The algorithm was evaluated on popular standard cell and macro cell benchmarks included in the MCNC benchmarks. The macro cell placement tool produced a maximum 29.44°C temperature decrease. And the standard cell placement tool produced a 6.3°C temperature decrease compared to the non-thermal placement tool. The tools produced an area overhead less than 5% and a runtime overhead reaching 45% for the standard cell placer and 167% for the macro cell placement tool. Finally, in (Hung, et al., 2005) the authors introduce a genetic algorithm (GA) based thermal aware floor-planning algorithm. The algorithm starts with a set of chromosomes; each encoding the orientation of each block and the polish expression of the slicing tree. The chromosomes then undergo GA operations of crossover, mutation, and selection. The solutions are then fed to an area calculator and to the thermal simulator Hotspot to calculate the fitness of each chromosome. The fitness calculation function incorporates both area and temperature. The experiments were carried out on the MCNC benchmarks and a face detection circuit. For the MCNC benchmarks the average temperature decreased by 1°C to 5°C. The peak temperature decrease varied between 1°C and 24°C. The algorithm produced a 0.2 average dead area ratio.

With the introduction of 3D ICs, and since they experience serious thermal issues due to their high power density, several algorithms have been proposed to improve their placement. The work in (Cong, Wei, & Zhang, 2004) proposes a thermally driven floor-planning technique for 3D cells. The authors introduce new moves to floor-planning in order to exploit placement in the Z direction. The authors also integrate three thermal models into the design flow in-order to evaluate the temperature of each solution. The three models are composed of an accurate model based on compact resistive networks, a fast model based on closed form equations, and a hybrid between the two. These models are used to measure the chip temperature of each solution by dividing the 3D chip into a grid of temperature measurement points. The temperature is then counted as part of the cost function during solution evaluation. To speed up the algorithm, temperature calculation is only performed after performing significant moves such as swapping two blocks. The experiments were carried out on the MCNC benchmarks and the GSRC benchmarks. The thermal guided floor-planner introduced a 66% reduction in average chip temperature on average while introducing a 21% area overhead on average. The authors of (Yan, Zhou, & Hong, Efficient thermal-aware placement approach integrated with 3D DCT placement algorithm, 2008) present a thermal force directed placement algorithm for 3D standard cell circuits. The algorithm starts with a random placement after which the initial thermal forces are calculated. The algorithm then iterates by updating the power distribution, calculating the new temperature using a thermal model, and calculating the new positions according to the new forces. The algorithm was tested on MCNC benchmarks and the IBM PLACE benchmarks. The algorithm produced a 1.3% reduction in average temperature, a 12% reduction in maximum temperature, and a 17% reduction in the average thermal gradient. The

algorithm also produced a 5.5% increase in total wire length. Meanwhile, in (Yan, Zhou, & Hong, Thermal aware placement in 3D ICs using quadratic uniformity modeling approach, 2009) the authors formulated the thermal placement problem of 3D ICs as a quadratic uniformity model using discrete cosine transform. The model takes into consideration both wire-length and thermal density. The authors introduce a global placement algorithm, a layering algorithm, and a local placement algorithm. In global placement, initial solutions are first given, and then a solution cost factor called DTCOST is computed. The coefficients of the quadratic formulations of DTCOST are then updated and then solved to get the new cell locations. In the layer assignment algorithm, the cells are assigned to the layer near their z position and then adjusted to remove uneven area distribution. The detailed placement algorithm is used to remove any overlaps and to optimize wire-length. To compute the thermal density, the authors use two methods, one that uses the power in each cell to compute the relative thermal density, and in the other the users use the power density as an indicator for thermal change. The algorithms were tested on the IBM place benchmarks. The experiments show a 2.8% to 3.6% decrease in average temperature and a 14% to 16% decrease in peak temperature. The method also imposed a 5%-7% increase in wire-length.

Various other placement techniques included partitioning into their algorithm in order to group similar components together. Most of these techniques performed partitioning heuristics which try different cuts with an objective of minimizing temperature. For instance, in (Chen & Sapatnekar, 2003) the authors introduce a thermally aware partitioning based placement algorithm based on FM bi-partitioning algorithm. The authors also introduce a fast thermal

model based on dividing the chip into thermal grids and computing the thermal conductivity matrix. The algorithm performs hierarchical bi-partitioning by moving standard cells across partitions performing the move with the maximum gain while complying with thermal and other constraints. The algorithm was tested on the MCNC benchmarks. The algorithm decreased the maximum temperature by as much as 56.2°C with an average of 12.96°C. The algorithm decreased the overall temperature by 0.09°C on average. Another example of including a partitioning stage in placement is presented in (Goplen & Sapatnekar, 2007) where the authors introduce a thermal aware placement algorithm for 3D ICs. The algorithm is composed of three stages: global placement, coarse legalization and detailed legalization. During global placement, the algorithm performs a recursive bi-partitioning algorithm which divides the cell among partitions trying to minimize a cost function. The cost function takes into account wire-length, number of vias, and the temperature of all components. The coarse and detailed legalizations perform swap moves and shift moves for spreading and removing overlaps. The temperature is calculated after each move using a grid thermal resistance model. The algorithm was tested on the IBM PLACE-1 benchmarks. The tests showed an improvement of 7.7% in the cost function with 65 times longer execution time than a regular placement algorithm. Furthermore, (Schafer & Kim, 2008) introduces a thermal guided partitioning algorithm to improve the results of a thermal aware floor-planner. The algorithm divides an initial floor-plan into grids then uses the thermal model "HotSpot" to calculate the temperature of each grid. Isothermal logic partitioning is then implemented which first groups the grids having temperatures belonging to the same interval into clusters. Then the partitioning technique performs horizontal and vertical cuts on the hottest grid in the clusters having a

temperature that exceeds a preset constraint. The resulted partitions are then fed into a thermal aware floor-planner which manipulates each partition to reduce the peak and overall temperature while keeping the time degradation minimal. The algorithm was tested on the ISCAS85 benchmarks. The algorithm resulted in a maximum temperature reduction between 6.27% and 8.85% for small circuits and between 2.4% and 6.46% for larger circuits with improvement reaching up to 13.99°C with a maximum timing degradation of 10%.

Using a different approach the authors of (Liu, Nannarelli, Calimera, Macii, & Poncino, 2010) propose a post placement technique to eliminate hotspots and produce uniform thermal profiles. The algorithm takes a placed net-list as an input and performs a thermal analysis on the chip using an RC based model. The algorithm then proceeds by performing one of two options: Empty Row Insertion, or using the Hotspot Wrapper. The Empty row insertion inserts an entire empty row in the hot region thus spreading the power density over a larger area decreasing temperature. The other method, Hotspot Wrapper, inserts empty cells in the hotspot region thus the area of only the hotspot which leads to a decreased temperature. The algorithm was tested on a synthesized benchmark composed of about 12000 cells where the hotspot size and position was controlled by the authors. The algorithms yielded up to a 30% improvement in hotspot temperature but with a 35% area overhead for the empty row insertion and a 40% overhead for the hotspot wrapper.

With the introduction of compact thermal models, thermal simulation became computationally efficient and was, therefore, included in higher stages in the design flow. Thermal awareness was mainly integrated in the Binding stage of high level synthesis. Different bindings produce

different switching activities and thus different dynamic powers for components of the same type. Several techniques have been proposed that use binding to even the power distribution among the chip components. For instance, the authors of (Mukherjee, Memik, & Memik, Peak temperature control and leakage reduction during binding, 2005) implement a thermal optimization and hotspot minimization binding algorithm. This algorithm iterates, going through all operations, redistributing them among resources, either by swapping operations or inserting them in other resources in order to decrease the temperature of the resource. They used Hotspot to simulate the steady state temperatures of the resources. The authors used Mediabench benchmarks extracted using SUIF and Machine-SUIF. They obtained a maximum reduction of 19.86°C. Overall they have obtained an average reduction of 12.2°C across all benchmarks. In another example, in (Liu, Bian, & Zhou, 2007) the authors present an ILP formulation of HLS in which they incorporate two constrains: max cycle power and max module power into the binding formulation. Assuring that these constraints are met reduces the formation of hotspots and reduces the overall temperature. The authors used DSP applications from the Mediabench benchmarks, the maximum average temperature improvement was 20.831 and the average improvement was 6.187. As for peak temperatures, the maximum was 19.21 and the average was 13.89. Finally, in (Kim & Lim, 2006) the authors present a thermally aware binding algorithm to minimize peak temperature and eliminate hotspots through formulating binding as a network flow problem. The algorithm uses floor-plan data to improve the quality of its results. In the absence of floor-plan data, the algorithm tries to minimize switching activities. The algorithm starts with an initial binding, identifies the resources with high switching activities and constructs a network of all the possible bindings for its bound

operations. The algorithm then searches for the binding that minimizes the switching activity. The algorithm then performs floor-planning, calculates the temperature of the resources, and repeats the optimization process using temperature instead of switching activity. The authors tested their algorithm on benchmarks from Mediabench. They also compared their results to those obtained using two competing power aware and temperature aware binding algorithms. The algorithm outperformed both algorithms on most benchmarks. On average, the algorithm reduced the temperature by 10.1°C and 11.8°C degrees less compared to the other two algorithms respectively.

Some works explored the possibility of including thermal awareness in resource allocation. The authors in (Yu, Zhou, & Bian, Peak Temperature Control in Thermal-aware Behavioral Synthesis through Allocating the Number of Resources, 2009) improve on the rebinding algorithm proposed by Seda et al. by adding resource reallocation. The authors also introduce a factor "V" which is used instead of temperature as a criterion in Seda's algorithm. V depends on the resource's temperature and its power density. The algorithm rebinds operation from the hottest resources to the coolest of the same type. The algorithm then checks if the difference in V between the hottest resource and the coolest resource of a different type is greater than a certain threshold. If so, the algorithm increases the number of the hot resource and decreases the number of the coolest resource, while meeting area constraints. The authors tested the algorithm on mediabench benchmarks and common DSP benchmarks. The algorithm reduced the peak temperature by as high as 27.9°C, and 11.1°C on average. It also reduced the total power by 13.0% on average. The algorithm introduces some area and latency overhead

reaching 5.7% increase in area and two cycles in latency compared to the technique proposed in (Mukherjee & Memik, An integrated approach to thermal management in high-level synthesis, 2006). Furthermore, in (Yu, Zhou, & Bian, Exploiting thermal-area tradeoffs in high-level synthesis through resources number selection, 2008), the authors implement an algorithm that chooses the optimal number of two types of ALUs, such that the difference in power density between the two types of ALUs is minimal. This way the whole chip will have almost the same power density. The authors also introduce a thermal aware scheduling and binding algorithm that minimizes switching activity.

The literature provides several papers in which high level synthesis and floor-planning were integrated together to produce more desirable results. The work in (Mukherjee & Memik, An integrated approach to thermal management in high-level synthesis, 2006) presented a complete thermal aware high level synthesis algorithm which performs regular HLS, then uses "HotFloorplan" to produce a temperature aware floor-plan. The authors generate the power trace file of the circuit and use hotspot to produce the stead state temperatures of the component. The algorithm then iterates for all operations, moving operations from the hottest component to the coolest component using a simulated annealing strategy. The algorithm also rebinds operations if a rebinding is found to be advantageous. If rebinding is not possible, the algorithm alters the operation's schedule until the conflict is resolved. The algorithm iterates until no substantial improvement occurs or until a termination condition is met. The authors tested the algorithm on DFGs generated using task graphs for free TGFF. The max temperature reduction on the hottest resource was 12.94 °C, and 7.95°C on average. The temperatures of all

components increased by 0.87°C on average due to an increase in their switching activities. In (Krishnan & Katkoori, 2010) the authors implement a two-step simulated annealing based HLS algorithm. In each cost function evaluation, the algorithm performs list based scheduling, allocation, binding, and then floor-planning. In the first high temperature step of the SA, the algorithm optimizes for area, latency, and power. In this stage the algorithm performs standard moves for HLS and Floor-planning, including rescheduling, rebinding, and standard floor-planning moves. In the second low temperature stage, the algorithm optimizes for temperature, power, and area. The tasks performed at this stage are changing the task priority within the same mobility range, rebinding operations and variables, in addition to standard floor-planning moves. Thermal aware floor-planning occurs by sorting the components according to temperature, and limiting the swap of a high temperature component to a one with low temperature. The authors tested the algorithm on the Mediabench benchmarks. The algorithm presented an average peak temp improvement reaching 15.5°C with a peak area overhead of 12.6%. Furthermore, in (Zhenyu, Yang, Wang, Dick, & Shang, 2006) the authors propose a thermal aware incremental floor-planning high level synthesis algorithm. The algorithm starts with an initial schedule, allocation, and binding. Then, simulations are made to extract area, latency, and power data from the design. The Algorithm then applies slack distribution, voltage clustering, and voltage islands aware floor-planning to construct an initial solution for the design. The algorithm then iterates applying rescheduling, incremental binding by resource sharing/splitting, and floor-planning to improve the design taking into consideration area, delay, power, and temperature. In each solution the algorithm iterates between binding and floor-planning to further optimize the design. The solution then performs

thermal aware floor-planning moves to optimize the design to reducing peak temperatures and hotspot formations. The authors propose heuristics to solve each optimization problem and the solutions are evaluated using a multi-objective cost function. The authors test their algorithm on popular DSP benchmarks. When optimizing for peak temperature, the proposed algorithm reduces the peak temperature by 12.5°C on average. This optimization produced an area overhead on most cases reaching 42.4%.

After surveying the literature for works similar to KT2C, in the next chapter we discuss the topics related to the algorithm that have been used in the KT2C optimized design flow.

Chapter Three

Related Topics

3.1 High Level Synthesis

The complexity of today's VLSI circuit design has made it crucial to take the design to higher levels of abstraction. Developing computer-aided design (CAD) techniques that interpret abstract descriptions of the design and transform them to physical or logic level implementations is one of the most effective methods to handle this complexity.

High level Synthesis is the automated process of transforming a behavioral description of a design to the structural design that implements that behavior (Gajski, Dutt, Wu, & Lin, 1992). The behavioral level is often referred to as the algorithmic level since it models the system as a set of inputs and outputs mapped onto operations. The behavioral level describes how these operations relate to each other in terms of precedence and causality. The structural description on the other hand often represents the circuit as a set of components and their corresponding netlists. High level synthesis takes the algorithmic level description of the intended chip and produces a Register-Transfer Level (RTL) description. The RTL level description could then be used in lower stages of the design flow such as Floor-planning or Placement. The high level synthesis operation consists of the generation of a datapath and a controller. Datapath generation is a combination of different NP-hard optimization problems which fit into three

interdependent tasks: (1) Scheduling, which is the process of assigning timeslots to the different operations while insuring not to violate any dependencies. (2) Allocation determines the number of functional, storage, and interconnect units needed, taking into consideration that some resources could be shared by different operations. Finally, (3) binding maps the operations onto their actual hardware units which will eventually produce a netlist of the connections in the chip.

High-level Synthesis explores the design space for possible mappings from the behavioral description to the structural implementation of the design. This search often returns several implementations for the same behavior. These solutions are usually refined by applying different constraints on the solutions. Several types of constraints could be applied to tune the result towards the specific needs of the designer. Typical constraints include resource, area, and timing constraints. Resource constraints guide the search towards solutions that use no more than a specified number of recourses of each type. For example, one might want to limit the number of multipliers to be used in the design due to their large area and power consumption. Area constraints on the other hand force the high level synthesis engine to explore designs that do not exceed a specified area. Area is usually estimated by summing the areas of all the components used since information about the placement of each component is not yet available. Another more accurate, yet more computationally expensive, estimate of area could be provided by integrating a logic synthesis engine into the high level synthesis solution quality evaluation phase. This approach is seldom used because it becomes harder for HLS to explore a large number of solutions in a short amount of time. Moreover, a very popular

type of constraints which is usually used alongside resource and area constraints is the Timing constraints. This kind of constraint works on limiting the execution time of the resulting design to a certain number of control steps.

The quality of the solution is usually evaluated against how well it meets its design goals. If two solutions meet their design constraints, their quality is assessed by the level at which each design minimizes its area and timing. The number of resource and area usually closely related and are often treated as one term which is inversely proportional to timing. That is, designs that have good quality in terms of area, generally have a bad timing quality. The reason for this inverse proportionality is that area minimization usually favors using fewer resources which requires reusing the same resource by different operations. This strategy usually extends the operation several control steps and thus degrading timing. Trying to minimize the control steps on the other hand leads to a higher level of parallelism in the circuit which requires more resources and thus larger area. High level synthesis involves finding feasible trade-offs between these two design goals.

In addition to the typical design goals, it is advantageous to include other design goals, such as power and temperature, into the solution evaluation. Different mappings from behavioral to structural design result in different switching activities among components causing different power distributions. Applying power constraints has recently become a popular choice especially that clock frequencies are becoming higher and feature size is becoming smaller. Temperature is also becoming an attractive factor that is being included in the design goals.

This has been made possible by the development of fast thermal models and is made necessary by dangerous effects high temperature is causing to VLSI chips.

3.1.1 High Level Synthesis Stages

High level synthesis inputs a behavioral description of the design in the form of a Data-Flow Graph (DFG) composed of multiple nodes and edges. The DFG is a directional graph in which each node represents an operation and each edge represents a precedence link between the nodes. The inputs of the circuit are placed in input nodes at the top of the graph and flow down through operation nodes which then converge into output nodes at the bottom of the graph. The longest path between the input nodes and the output nodes, i.e. taking the largest amount of control steps, is called the critical path. An example of a DFG is presented in Figure 2; the nodes 1 through 11 are operation nodes, node 0 is an input node, and node n is an output node. The directed edges between nodes 1 and 2 on one hand and node 3 on the other imply that nodes 1 and 2 are predecessors of node 3 and hence operations 1 and 2 must finish executing before operation 3 can start. Also we can deduce from this DFG that the path 0-1-3-4-5-n is the critical path since it passes through the largest number of operations, assuming all operations take one control step to execute. The DFG is parsed into the High-level Synthesis engine and the nodes undergo the first HLS stage of Scheduling.

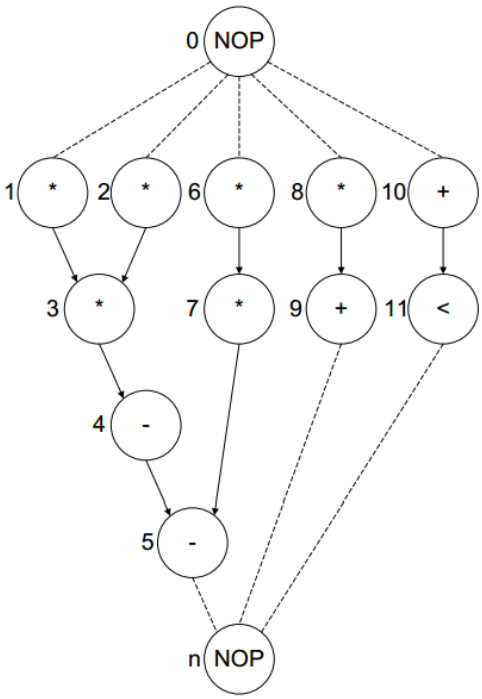

Figure 2 Data-Flow Graph (DFG)

3.1.2 Scheduling

Scheduling is the stage in which operation nodes are allocated to the specific control steps in which they start their execution. Scheduling primarily takes into consideration the precedence links between operations. Many scheduling techniques have been proposed in order to minimize different factors such as the number of control steps, referred to as latency, and the number of resources used. Changes in the schedule can cause fundamental changes in the structural design outputted by the HLS engine. This is because scheduling has a significant effect on the number of resources. Furthermore, scheduling plays a role in the mapping of operations into their actual functional units and thus affecting factors such as resource usability and the switching activity between resources.

The literature provides several scheduling algorithms including a variety of deterministic one-shot algorithms and iterative heuristic algorithms to accomplish that purpose. One of the most popular scheduling algorithms is the As Soon As Possible (ASAP) scheduling algorithm. As the name suggests, this scheduling algorithm assigns operations to control steps as soon as their predecessors have been scheduled. Therefore, this algorithm always outputs the minimum number of control steps a DFG needs. The pseudo-code of the ASAP algorithm is presented in Figure 3 and an illustration is provided in Figure 4.

```
for each node vi ∈ V do
        if Pred_vi = ø then
                Ei = 1;
                V = V - {vi};
        else
                Ei = 0;
        elseif
endfor
while V ≠ ø do
        for each node vi ∈ V do
                if ALL_NODED_SCHED (Pred_vi E) then
                        Ei = MAX(Pred_vi E) + 1;
                        V = V – {vi};
                endif
        endfor
endwhile
```

Figure 3 ASAP scheduling algorithm (Gajski, Dutt, Wu, & Lin, 1992)

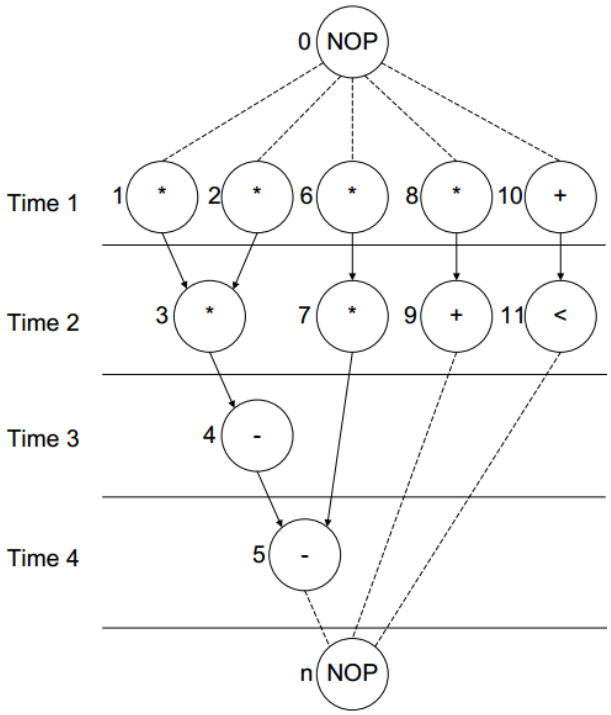

Figure 4 ASAP-scheduled DFG

Another popular scheduling algorithm is the As-Late-As-Possible algorithm (ASAP), summarized in Figure 5, which when given an upper bound on the number of control steps, schedules the nodes from the end backwards. In other words, the algorithm starts by scheduling the operation nodes with no successors at the upper bound of control steps and then proceeds by scheduling the nodes that have all their successors scheduled. This is illustrated in Figure 6 where the upper bound is considered to be 4 control steps.

```
for each node vi ∈ V do
        if Succ_vi = ø then
                Li = T;
                V = V - {vi};
        else
                Li = 0;
        elseif
endfor
while V ≠ ø do
        for each node vi ∈ V do
                if ALL_NODED_SCHED (Succ_vi E) then
                        Ei = MAX(Succ_vi E) + 1;
                        V = V – {vi};
                endif
        endfor
endwhile
```

Figure 5 ALAP scheduling algorithm (Gajski, Dutt, Wu, & Lin, 1992)

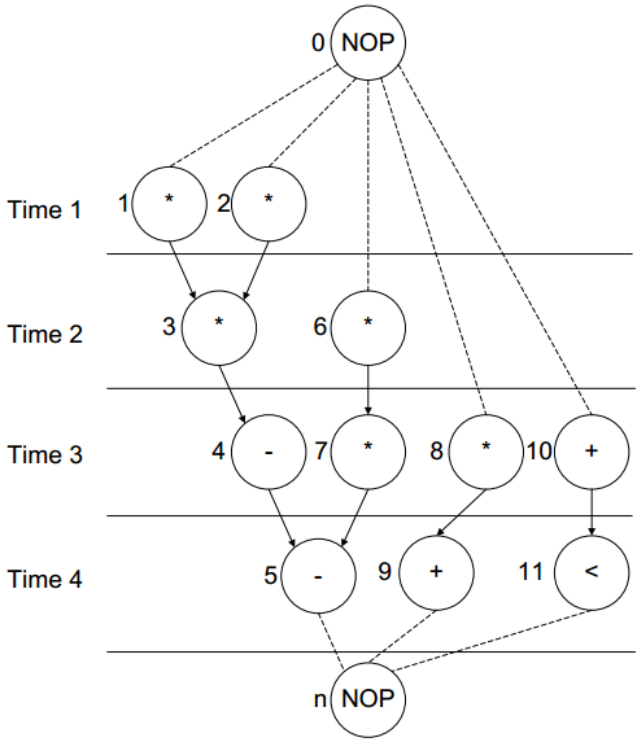

Figure 6 ALAP scheduling DFG

3.1.2.1 List Scheduling

```
INSERT_READY_OPS (V, Plist_t1, Plist_t2, …, Plist_tm);
Cstep =0;
while (Plist_t1 ≠ ø) or … or (Plist_tm V ≠ ø) do
        Cstep = Cstep + 1;
        for k = 1 to m do
                for f_unit = 1 to N_k do
                        if PList_tk ≠ ø then
                                SCHEDULE_OP(S_current, FIRST(PList_tk),Cstep) ;
                                PList_tk = DELETE( PList_tk, FIRST(PList_tk));
                        endif
                endfor
        endfor
        INSERT_READY_OPS (V, Plist_t1, Plist_t2, …, Plist_tm);
endwhile
```

Figure 7 LIST scheduling algorithm (Gajski, Dutt, Wu, & Lin, 1992)

The List scheduling algorithm is considered to be a very effective algorithm which can produce an optimized schedule while taking constraints into account. This algorithm can be used with two types of constrained problems; it can either minimize the number of resources for a given latency constraint, or it can minimize the number of control steps taken by the design given a resource constraint.

As explained in Figure 7, the List algorithm uses both ASAP and ALAP scheduling to calculate the "mobility" of each operational node. The algorithm then schedules the nodes whose predecessors have been scheduled while giving priority to the nodes with the least mobility. In resource constrained problems, as illustrated in Figure 8, the algorithm schedules the node with the least mobility as long as there are resources of the relevant type available. If not, the algorithm adds a control step and repeats the same procedure until all nodes are scheduled.

LIST can also be used to optimize for the number of resources under latency constraints. In such problems, the algorithm initially starts with an empty resource bag and only increments the number of a certain resource if a node's mobility is zero at a certain control step and there are no available resources for it to be scheduled as illustrated in Figure 9.

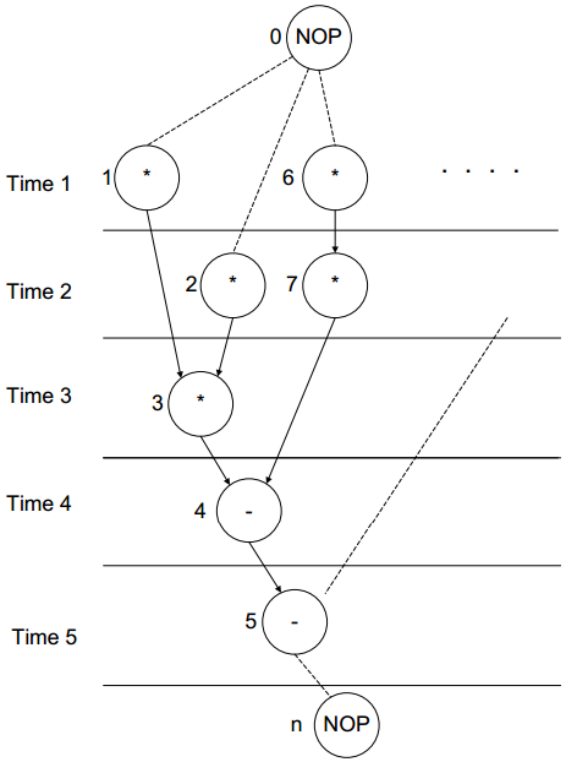

Figure 8 Resource-constrained LIST scheduling (2 Multipliers, 1 Subtractor)

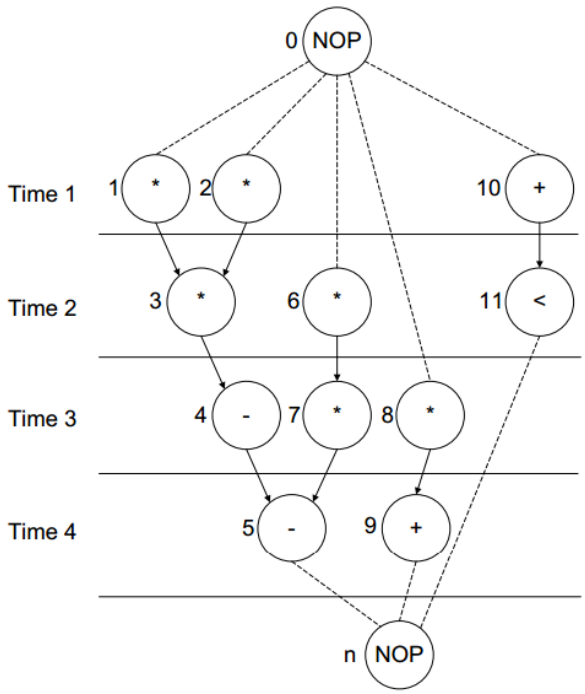

Figure 9 Latency-constrained LIST scheduling (Latency = 4 time steps)

3.1.3 Binding and Allocation

Binding and allocation is the second stage in high level synthesis. Allocation aims at determining the optimal number of resources to be used while binding aims to finding the best mapping between scheduled operations into their relevant structural components. In some cases, the number of resources is determined during scheduling as in the case of latency constrained List scheduling. In other cases, iterative methods are used to find the optimal number of resources satisfying certain design goals. In most cases, the minimum number of resources is determined during the binding task using resource sharing algorithms. An example of popular resource sharing binding algorithm is the Left-Edge algorithm.

3.1.3.1 Left-Edge algorithm

As summarized in Figure 10, the Left-Edge algorithm deals with one resource type at a time. For each resource type, the algorithm sorts the relevant nodes according to their start time. The algorithm starts with one resource and binds to it the operation with the earliest start time. The algorithm then searches for the operation that starts directly after the first operation that finished execution and binds it to the resource. The algorithm keeps adding operations until there are no more operations that start execution after the end of the last operation. If there are still any operations that are not bound to a resource, the algorithm adds a resource and repeats the same behavior. Left-Edge algorithm could be used to bind operations to their relevant functional unit, their data to registers, and their inter-connections to appropriate busses and multiplexer in the same fashion.

```
LEFT_EDGE(f)
        Sort elements of I in a list L in ascending order of I;
        C = 0;
        While (some interval has not been colored)  do{
                S = ø;
                r= ø;
                While (exists s Є L such that I > r) do{
                        s = First element in the list L with I>r;
                        S = S U {s};
                        r=r_s
                        Delete s from L;
                }
                c=c+1;
                Label elements of S with color c;
        }
```

Figure 10 Left-Edge algorithm (Gajski, Dutt, Wu, & Lin, 1992)

3.2 A GPU-Parallel Bee Colony Optimization for Design Space Exploration in High-Level Synthesis

In this section I discuss the research I did under the supervision of Dr. Ouaiss prior to the work discussed in this thesis. The techniques used in this research contributed to the design of KT2C. The probabilistic strategy used in this work for exploring the solution space was embedded in the cluster initialization stage of KT2C. Furthermore, the parallel nature of the algorithm presented in this work and the advantages witnessed by this parallelism was the motivation for designing KT2C as a parallel algorithm. This work was concluded in a manuscript submitted as a journal publication.

This section presents a parallel bee colony optimization algorithm to perform an effective and efficient design space exploration for high-level synthesis. Design space exploration for HLS is a search problem that aims at finding the best solutions that meet design objectives. The design space exploration of today's complex VLSI circuits is an NP-hard problem which is best solved using heuristics. The implemented algorithm uses a large number of bees in which each bee performs the HLS tasks of scheduling, allocation, and module selection. The bees then share and manipulate their solutions in order to minimize the area and the overall latency of the targeted digital circuit. Due to the large number of bees and the considerable effort each bee has to undertake, the algorithm has been modified to run on a CUDA capable graphics processing unit making use of its massively parallel architecture. The bee colony optimization algorithm was tested on well-known benchmarks and compared to different effective HLS techniques.

3.2.1 Bee Colony Optimization

The Bee Colony Optimization (BCO) algorithm (Teodorovic, Lucic, Markovic, & Dell' Orco, 2006) divides the problem into multiple stages, each stage consisting of two phases: forward pass and backward pass. The algorithm starts with each artificial bee having a partial solution to the problem. Then the algorithm alternates between forward and backward passes to eventually yield complete solutions. The best complete solution is then saved and the algorithm iterates again until a stopping criterion is met.

In the forward pass, each bee evaluates all the possible constructive moves and makes one or more moves. The bee's choice favors the attractive moves that would improve the quality of the solution, but still giving a chance for less attractive moves to be chosen.

After the forward pass, the bees undergo a backward pass in which all the bees return to the hive. In the hive, each bee goes through a decision process where it decides whether to continue exploring its solution space, abandon its solution and follow another bee, or recruit new bees to continue exploring its solution space more effectively. This divides the bees into three groups: the explorers, the followers, and the recruiters. Usually, bees with good solutions choose to continue exploring their solution space whereas bees with bad solutions abandon them.

3.2.2 High-Level Synthesis BCO Algorithm

The implemented BCO algorithm aims to find the optimal schedule and number of functional units to be allocated to a design problem and is summarized in Figure 11.

HLS Bee Colony Optimization

Initialization:

Determine the number of bees B, the number of iterations I, and the number of stages ST.

Find the ASAP, ALAP, and the mobility-based List Schedules.

Set i = 1. Until i = I, repeat the following steps:

Set the mobility-based list priority as the base priority for all bees.

Initialize allocated FUs randomly.

Set j = 1. Until j = ST, repeat the following steps:

Forward pass:

- Each bee evaluates all its possible nodes and calculates node's probability to be chosen, P.
- Each bee Chooses one node using roulette wheel and adds it to its partial priority list.

Backward pass:

- Send all bees back to the hive and sort Bees according to their partial solutions' fitness.
- The top C percent of bees continue to explore their solution space. The rest of the bees become uncommitted followers
- Each uncommitted follower chooses one of the explorers to follow.
- With probability R, the follower perturbs its new number of allocated FUs.
- Set j: =j + 1.

Update best solution.

Set, i: = i + 1.

Return best solution.

Figure 11 HLS Bee Colony Optimization (HLS-BCO)

Initially, the ASAP and ALAP schedules are found and the mobility of each node is then calculated. A priority list based on mobility is generated and the priority lists of all bees are set equal to it. As for the number of allocated functional units, it is randomly initialized for each bee. In resource constrained problems, the maximum number of allocated functional units would be equal to the constraints.

As explained in the section 3.2.1, the BCO algorithm undergoes a number of iterations. Each iteration consists of a number of stages which, in this implementation, is equal to the number of nodes in the priority list. In each stage, the artificial bees execute a forward pass. Each bee

evaluates all the possible nodes that could be placed in the corresponding element in the priority list then chooses one node. A node's probability of being chosen by the bee is proportional to the resulting solution's fitness relative to the total fitness of all solutions. This probability is calculated using the Eq. 1.

$$Pi = \frac{fitness(i)}{\sum_{n=0}^{N} fitness(n)}$$

Where,

$$fitness(i) = \frac{1}{cost\ of\ solution\ (i)}$$

$$N = number\ of\ feasible\ nodes$$

(1)

After the forward pass, the bees go through a backward pass. In this phase, the bees are sorted according to their fitness. Then, the top P percent of bees with the highest fitness values continue to explore their solution space while the rest become uncommitted followers.

Finally, the uncommitted followers choose recruiters from amongst the explorer bees. The explorer bees with better fitness values have a higher probability to be chosen by the uncommitted followers and thus should have more followers to explore their search space.

To explore the design space more effectively we introduce a perturbation factor R to the algorithm. With a probability of R the followers will deviate from the explorer they selected by perturbing the number of allocated functional units for that solution. The follower will have the same solution of the selected explorer with a probability of 1 - R. This perturbation allows the bees to explore the neighborhood of the good solutions found.

The bees repeat alternating between forward and backward passes for all the elements in their priority list. The best solution found by the bees is saved and the algorithm iterates again until a termination condition is encountered. At the beginning of each iteration, the priority lists are reset to the initial mobility-based priority list and new values for the number of allocated functional units for each bee are randomly generated.

3.2.3 GPU Acceleration

There are two factors that make the proposed BCO very attractive for parallel implementation: (1) the autonomous and distributed nature of bees, and (2) the large number of bees that should be used in order to obtain good design space exploration. The parallelism was implemented using a CUDA-enabled Graphics Processing Unit which, due to its architecture, appears suitable for this algorithm. CUDA-enabled GPUs are characterized with their parallel throughput and their ability to execute many concurrent threads. This is made possible by the GPU's unique single instruction multiple data (SIMD) multicore architecture shown in Figure 12. A CUDA GPU is composed of a number of Stream Multiprocessors (SMs) sharing a global memory. Each SM is made up of many small CUDA cores which share a memory that is smaller, but faster, than the global memory. The CUDA cores are not as powerful or as fast as CPU cores but due to their large number and parallel function they are capable of producing large speedups.

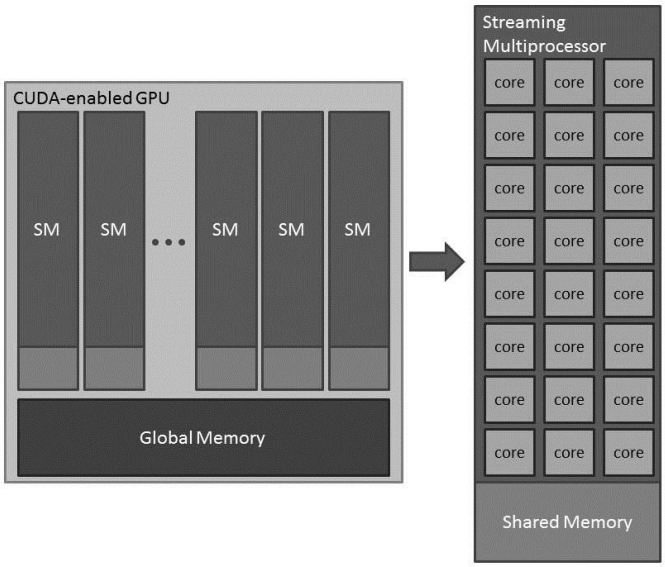

Figure 12 CUDA-enabled GPU architecture

After conducting a timing analysis of the algorithm, the computational bottlenecks were identified and accelerated using the CUDA capable GPU. The timing analysis showed that the algorithm spends most of its time in the forward pass. This is justified since the forward pass contains a large number of cost evaluations that must be carried out for all bees. For that reason, all the functions carried out in the forward pass were converted into GPU functions (a.k.a. kernels). Each kernel is called with a number of threads equal to the number of bees. The operations carried out by each bee in the parallel forward pass are similar to the operations carried out in the sequential forward pass except that multiple bees are executing simultaneously. The maximum number of concurrent bees depends on the capabilities of the used GPU. The modified forward pass used in the parallel implementation of the algorithm is shown in the flowchart of Figure 13.

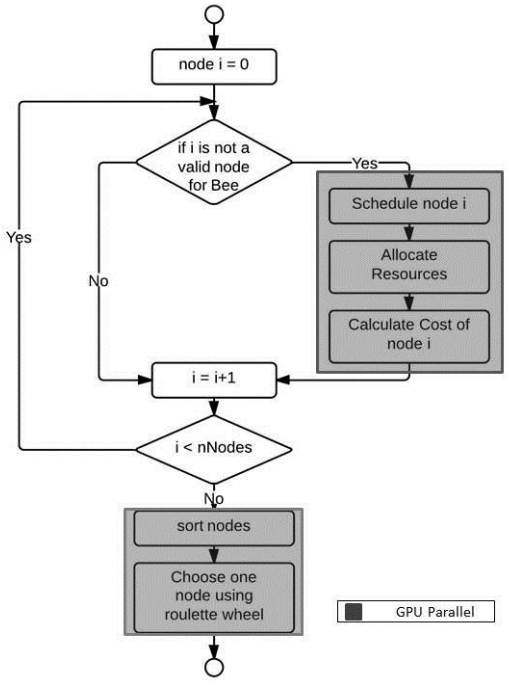

Figure 13 GPU-parallel BCO

3.2.4 Experimental Results

The proposed BCO algorithm was implemented in C++ and tested on an Intel Xeon(R) CPU E5606 running at 2.13GHz. To evaluate the effectiveness of the algorithm, it was tested on a number of benchmark problems drawn from the literature. The algorithm was also compared to several popular high level synthesis techniques: ASAP, ALAP, List, and FDS. Furthermore, and in order to highlight the advantage of using BCO for design space exploration, the genetic algorithm presented in (Krishnan S. K., 2006) was implemented. All the techniques were assessed on two things: the number of time steps for the datapath (Latency) and the total area including the areas of the allocated functional, storage, and interconnect units. Each time step is assumed to be 20 ns. The CMOS module library used consists of ALUs, multipliers, registers and multiplexers.

In the experiments in which BCO was compared to the algorithms mentioned above, the algorithm ran sequentially with the number of bees is set to 128 bees, the explorer bees are selected from the top C = 20% of the bee population, the follower perturbation rate R is set to 0.4 and the number of iterations N to 10. The weights alpha and beta are used to set both terms of the cost function to a proportional scale. Alpha and beta were set with different values for each benchmark problem. The algorithms were tested on eight benchmark problems having different number of nodes and complexity to ensure a fair evaluation.

The parameters for the genetic algorithm were extracted from (Krishnan S. K., 2006) and were as follows: The algorithm ran with a fixed population of 100 chromosomes. The crossover probability was set to 0.90 and the mutation probability was set to 0.20. The runs were stopped after 100 generations.

Tables 2 summarizes the area costs obtained by the HLS BCO algorithm on the different benchmarks compared to the ASAP, ALAP, List, FDS, and GA techniques under latency constraints. Since BCO and GA are stochastic algorithms whose results might vary from run to run, the recorded results are the average of 10 independent runs. The standard deviation of the cost was also reported for these two algorithms. The format was as follows: Average Area (Standard Deviation). Table 3 summarizes the improvement attained by using our algorithm compared to the above mentioned techniques.

Table 2 Area Costs under Latency Constraints

		Area (um2)					
Benchmark	**Latency Constraint(ns)**	**BCO**	**ASAP**	**ALAP**	**LIST**	**FDS**	**GA**
diffeq2	120	784(0)	1142	1096	938	938	784(0)
4pDCT	80	1286.4(24.45)	1778	1618	1472	1366	1298(0)
arf	160	2112(0)	3488	3488	2808	2384	2436.4(69.75)
ewf	280	2560(0)	3586	3428	2668	2668	2668(0)
cos2	120	3338(0)	4480	5168	3460	3460	3338(0)
nestor2	120	3702(0)	5516	5290	4428	4012	3672.6(16.81)
DCT2	120	8240(0)	9532	9532	8240	8240	8252.4(39.21)

The GPU parallel BCO algorithm was implemented in CUDA C and ran on an NVIDIA Tesla C2070 GPU installed on the same system used for the sequential tests. The parameters C% and R were set to 0.2 and 0.4 respectively. The number of bees was set to 1024 and thus the GPU kernels were launched with 1024 threads divided into blocks with 128 threads per block. The number of iterations was set to 20 to insure the convergence of the algorithm. The GPU parallel BCO algorithm was compared to the sequential algorithm for execution time. Since the parallelism was located in the forward passes, two sets of results were compared: the total forward pass time, and the total runtime of the algorithm. The results are summarized in Table 4. In addition, it is worth mentioning that both sequential and parallel algorithms produced equal results with the respect to area and latency.

Table 3 Area Improvement when using HLS-BCO

		Improvement (%)				
Benchmark	**Latency Constraint(ns)**	**ASAP**	**ALAP**	**LIST**	**FDS**	**GA**
diffeq2	120	31.35	28.47	16.42	16.42	0.00
4pDCT	80	30.26	23.36	15.76	9.22	0.89
arf	160	39.45	39.45	24.79	11.41	13.31
ewf	280	28.61	25.32	4.05	4.05	4.05
cos2	120	25.49	35.41	3.53	3.53	0.00

nestor2	120	32.89	30.02	16.40	7.73	-0.80
DCT2	120	13.55	13.55	0.00	0.00	0.15

Table 4 Parallel vs. sequential runtime

		Total Forward Pass Runtime			**Total Runtime**		
Benchmark	**Number of Nodes**	**Parallel (s)**	**Sequential (s)**	**Speed-up**	**Parallel (s)**	**Sequential (s)**	**Speed-up**
diffeq2	10	0.42	6.03	14.45	1.96	7.90	4.04
4pDCT	15	1.67	20.38	12.19	4.06	22.75	5.60
Arf	28	23.22	214.44	9.24	28.64	220.82	7.71
Ewf	34	59.44	538.00	9.05	67.33	545.49	8.10
cos2	42	117.05	893.50	7.63	127.70	902.85	7.07
nestor2	48	192.23	1413.95	7.36	206.48	1425.51	6.90
DCT2	70	752.50	5329.18	7.08	780.56	5351.66	6.86

3.3 CMOS Power Calculation

Power has gained attention in recent years as a primary target for optimization in VLSI chip design. This is, in part, due to the increasing demand on portable devices having high computation abilities and prolonged battery life. Furthermore, the rapid increase in clock frequencies, decrease in feature size, and increase in chip density has increased the amount of heat dissipated in the circuit due to high power. These factors have steered the attention of VLSI designers towards the different sources of power dissipation in the chip, which could be summarized into two categories: Static Current, and Dynamic Current.

3.3.1 Static Current

The static current component is composed of the summation of two current sources: the Leakage current, and the Stand-by current.

The Leakage current depends on the fabrication technology and is itself composed of two currents. These two current are (1) the reverse bias current formed between the source and drain diffusions and the bulk region, and (2) the sub-threshold current caused by the inversion charge formed at gate voltages that are below threshold. Different technologies have different values of Leakage currents and designers can minimize its effects by choosing the proper technology for their circuit application.

The second component of static current, the Stand-by current, flows from Vdd to ground in a continuous manner. Its effects are usually significant in pseudo-nMOS and nMOS pass transistor logic design styles. The Stand-by current value is negligible in CMOS technologies.

3.3.2 Dynamic Current

Dynamic current is the term that refers to the summation of two types of currents, the Short-Circuit current, and the Capacitance current.

The Short-Circuit current, also called Rush-through current, is caused by the DC current that flows in the path formed between the supply rails during output transitions. The Short-Circuit current depends on the load, the transistor sizes of the gate and the input ramp time. This current increases as the load increases and reaches its maximum value when no load is connected. In order to estimate the value of the Short-Circuit current; different models have been proposed varying in accuracy such as the formulas presented in (Veendrick., 1984) and (Jeppson & Hedenstierna, 1987). Another model is presented in (Turgis, Azemard, & Auvergne., 1995) which estimate the short circuit power dissipation through modeling it as an equivalent virtual capacitance C_{sc}, which they called the Short-Circuit Capacitance.

The Capacitance Current on the other hand, is the most dominant source for power dissipation in VLSI CMOS circuits. This current is formed when the logic values change which results in charging and discharging of the capacitive loads. The Capacitance current power dissipation is given by equation 3:

$$P = 0.5 C_L {V_{dd}}^2 f_{clk} E_{sw} \quad (3)$$

For a certain circuit, this power depends on the physical capacitance of the circuit (C_L), the supply voltage (V_{dd}), and the clock frequency f_{clk}. The Capacitance current power also depends on the switching activity E_{sw} at the input of that circuit. The switching activity is the average number of logic changes at the input of the circuit in one clock cycle. The Capacitance current power is directly proportional to all these factors, with a quadratic dependence on the supply voltage.

Furthermore, if we modeled the Short circuit current as an equivalent capacitance C_{sc}, we can calculate the dynamic power of a CMOS circuit using equation 3 by adding the Short-Circuit capacitance $C_{sc}$to the load capacitance C_L.

3.3.3 Switching Activity Calculation

The switching activity is the average number of logic changes at the input of a CMOS circuit in one clock cycle. There are several methods to estimate switching activity at HLS and RTL levels. One of the efficient methods that calculate switching activity at the high level is introduced in (Bogliolo, Benini, Ricco, & DeMicheli, 1999).

In (Bogliolo, Benini, Ricco, & DeMicheli, 1999) the authors point out that binding operations to different functional units changes the switching activities in the circuit. They introduce a technique that enables the calculation the switching activities resulting from different bindings without the need for multiple simulations.

The proposed technique works as follows:

- Consider a DFG with N operations: Op1, Op2, … , OpN.
- The bit streams at the inputs of each operation are named In1, In2, … , In2 and are composed of the concatenation of the bits of the two operands of each operation.
- The DFG is run with M simulation trace values.
- Consider the N-element vector Neval which stores the number each operation is evaluated during the M simulation runs.
- At each simulation trace, the hamming distance between all the input streams are calculated and added to the "toggle count" N-by-N matrix Tc.
- When the N runs are over, the switching activity due to binding OpX, OpY, and OpZ to the same resource is calculated by equation 4.

$$\text{Switching Activity}(\text{OpX}, \text{OpY}, \text{OpZ}) = \frac{\text{Tc}(\text{InX}, \text{InY}) + \text{Tc}(\text{InY}, \text{InZ}) + \text{Tc}(\text{InZ}, \text{InX})}{\text{BW}(\text{Neval}(\text{OpX}) + \text{Neval}(\text{OpY}) + \text{Neval}(\text{OpZ}) - 1)} \quad (4)$$

Where OpX, OpY, and OpZ are executed in this order and BW is the bit width of the operations' inputs.

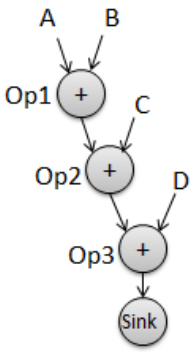

Figure 14 Switching activity example DFG

Op1) n1 = A + B
Op2) n2 = n1 + C
Op3) n3 = n2 + D

Figure 15 Switching activity example operations

To further illustrate the switching activity calculation, we provide the example scenario presented in (Bogliolo, Benini, Ricco, & DeMicheli, 1999). Consider the DFG of Figure 14 which is performing the operations of Figure 15 consisting of three addition operations and whose 2-bit inputs are A, B, C, and D.

This DFG is run for three input sets and the bit streams at the inputs of the three operations, at each input set, are concatenated together and labeled as In1 through In3 as illustrated in Figure 16. We next find the number of evaluations for each operation which is found to be 3 for all operations in this example.

A	B	C	D	In1	In2	In3
2	0	1	1	10 00		
					10 01	
						1101
1	1	3	0	01 01		
					10 11	
						01 00
1	2	1	3	01 10		
					11 01	
						00 11

Figure 16 Switching activity example inputs

	In1	In2	In3
In1	5	7	5
In2	5	3	8
In3	2	4	5

Figure 17 Switching activity example Tc matrix

The results are stored in the matrix Neval. Furthermore, the hamming distance between all input transitions is found and accumulated for all simulation sets in matrix Tc shown in Figure 17. For example, in order to determine Tc(In1, In2), the distance between In1 and In2 in the first simulation set is 1, 3 in the second, and 3 in the third resulting in a total of 7, as shown in Figure 3. For determining Tc(In2, In1) on the other hand, the distance is to be found between In2 of the first simulation set and In1 of the second. This is because In1 has to take place before In2 in each simulation set. The value of Tc(In2,In1) in the last simulation set is found by calculating the distance between In2 of the last simulation set and In1 of the first simulation set.

Finally, the switching activity resulting from joining two operations, for example Op1 and Op2, on the same functional unit is calculated by applying equation 4.

$$\text{Switching Activity}(\text{Op1}, \text{Op2}) = \frac{\text{Tc}(\text{In1}, \text{In2}) + \text{Tc}(\text{In2}, \text{In1})}{\text{BW}(\text{Neval}(\text{Op1}) + \text{Neval}(\text{Op2}) - 1)}$$

$$= \frac{7+5}{4(3+3-1)} = 0.6$$

3.4 Clustering Analysis

Clustering analysis is a technique in machine learning and data mining that aims towards grouping data points into clusters according to a similarity or distance measure. Clustering algorithms organize points that have high similarity, or low distance from each other, in the same cluster while maintaining a high distance between different clusters (Rajaraman, Leskovec, & Ullman, 2011) (Tan, Steinbach, & Kumar, 2006).

In general, clustering is applied to data points that belong to a Euclidean space having vectors of real numbers that are considered coordinates. In this case, the distance between two points is usually the Euclidian distance represented in equation 5. If the point space in non-Euclidean other distance measures are used including Jaccard distance, cosine distance, and Hamming distance.

$$Distance = \sqrt{\sum_i (x_i - y_i)^2} \quad (5)$$

Where x_i and y_i are the coordinates of the two points x and y

Clustering algorithms can be divided into two categories:

1. Hierarchical clustering: in this form of clustering, each data point starts as a cluster of its own. Then, clusters with a close distance are merged together until a termination criterion is met. Some of the popular stopping criteria include stopping at a predefined number of clusters, stopping when an upper bound on intra cluster distance is violated, or when a lower bound on inter cluster distance is met. Changing these criteria results in different ways to cluster the same data, as illustrated in Figure 18.

2. Point Assignment clustering: clustering of this type includes an initial phase where the cluster centers are calculated. After that, the data points are assigned to the cluster which has the smallest distance between the point and the cluster's center.

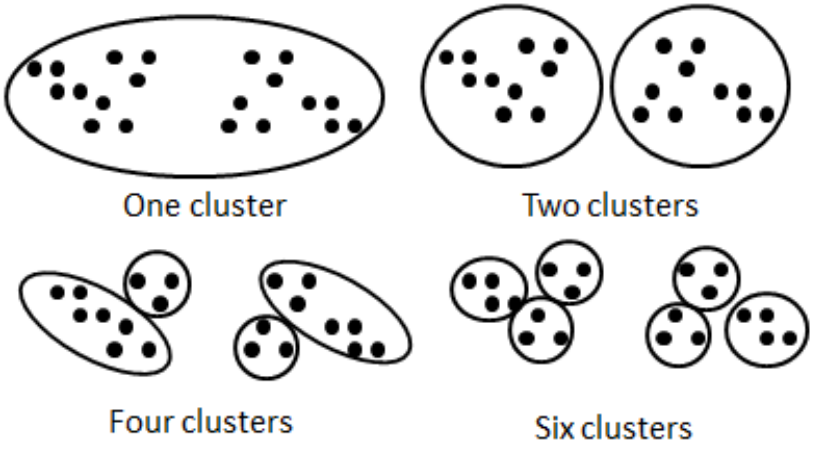

Figure 18 Different ways to cluster data (Tan, Steinbach, & Kumar, 2006)

3.4.1 K-means Algorithm

K-means is a well-known and extensively used clustering algorithm based on point-assignment. The algorithm's pseudo-code is outlined in Figure 19. K-means algorithm needs a Euclidean space and a predefined number of clusters k. According to the basic version of the algorithm, k points are chosen at random to be the centers of the k clusters. The Euclidean distance from each point to the centers of the different clusters is calculated and the point is assigned to the cluster with the closest distance the point has to its center. After all the points are assigned, the new centers are calculated. The new center of a cluster "A" is the mean of all the points belonging to "A". The algorithm iterates until the cluster centers are no longer changing. An illustration of the execution of the k-means algorithm is provided in Figure 20.

Basic K-means algorithm.

1: Select K points as initial centroids.
2: **repeat**
3: Form K clusters by assigning each point to its closest centroid.
4: Recompute the centroid of each cluster.
5: **until** Centroids do not change.

Figure 19 Basic K-means algorithm (Tan, Steinbach, & Kumar, 2006)

Since choosing the initialization values of the k centers significantly affects the output of the clustering algorithm, some techniques have been proposed to determine the best centers of the k clusters. Most of these techniques involve running the k-means algorithm several times and evaluating the squared sum error (SSE). SSE is calculated by summing the distance between each point and the center of cluster it belongs to, and then adding the distance sums of all clusters together as shown in equation 6. The clusters resulting from the run with the least SSE are chosen.

$$SSE = \sum_{i=1}^{K\ clusters} \sum_{i \in C_i} (c_i - x)^2 \quad (6)$$

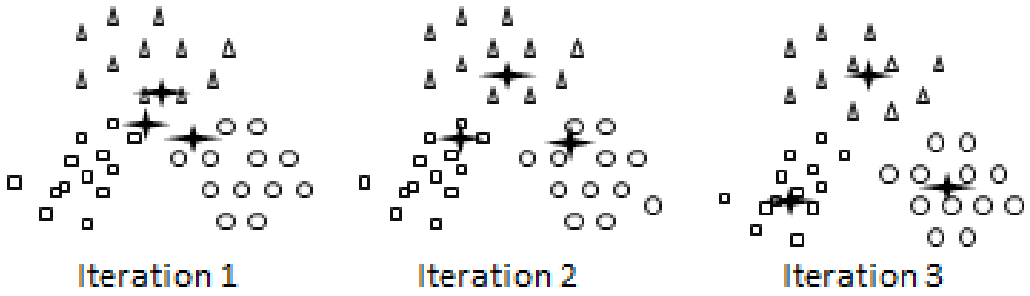

Figure 20 K-means execution (Tan, Steinbach, & Kumar, 2006)

3.5 Floor-planning

The Floor-planning stage in VLSI design entails determining the locations of the different circuit components on the chip. Floor-planning has a significant effect on the design parameters of area, routability and wire length, timing delays, power, and heat dissipation. Floor-planning occurs after binding in the design flow and right before placement. It takes as an input a set of components and a net-list and generates the position of each component on the chip.

The main floor-planning objectives are to reduce the wire length between the different connected components and to reduce the area of the overall design. There are several wire length estimation techniques in the literature, most popular of which are summarized in Figures 21 and 22.

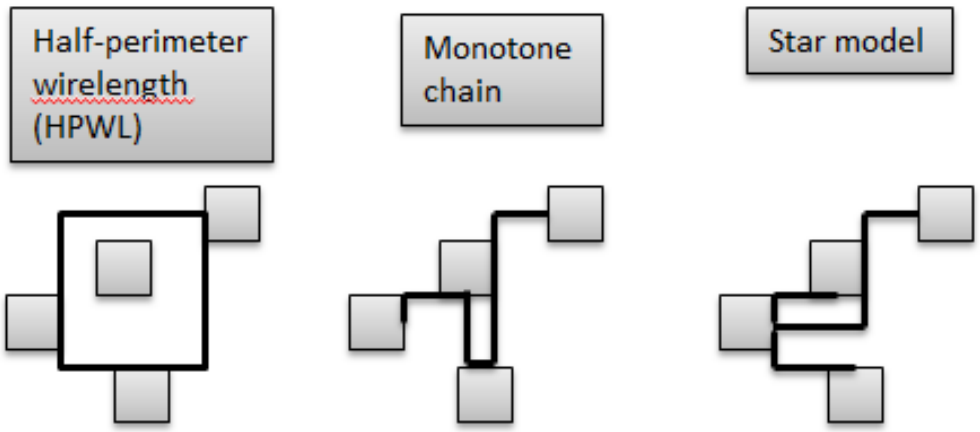

Figure 21 Wire-length estimation techniques (Kahngm, Lienig, Markov, & Hu, 2011)

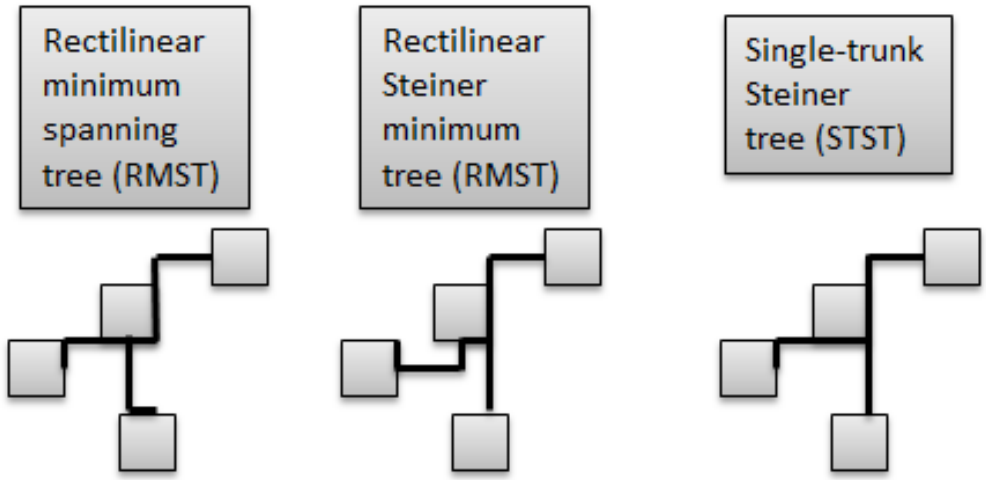

Figure 22 Wire-length estimation techniques (continued...) (Kahngm, Lienig, Markov, & Hu, 2011)

There are three main approaches for achieving an optimized floor-plan:

(1) Partitioning Based floor-planning

(2) Simulated Annealing floor-planning

(3) Analytical floor-planning

In partition based floor-planning, the components are continuously partitioned into smaller groups while minimizing the number of interconnections between groups. By following this method, highly connected components would wind up in the same partition. The floor-planning region is also divided into subareas each with each subarea mapped to a partition of components as illustrated in Figure 23. The partitioning floor-planning algorithm iterates until no further partitioning could be achieved and the partitions are placed in their relevant locations in the floor-pan.

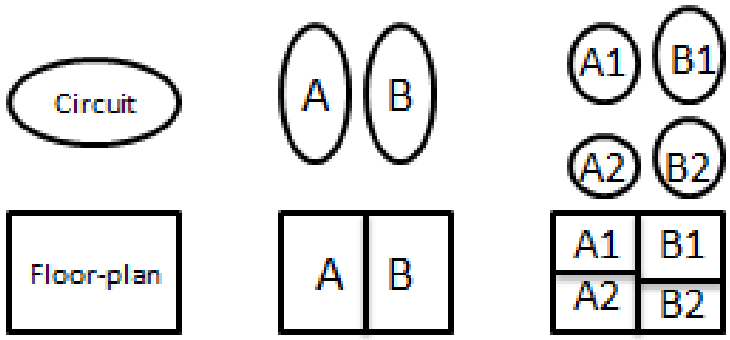

Figure 23 Partitioning-based floor-planning algorithm (Wang, Chang, & Cheng, 2009)

In simulated annealing floor-planning, the floor-planner iteratively tries several floor-plans and then assesses the quality of each floor-plan according to a preset cost evaluation function. In the beginning stage of the annealing process, when the temperature is high, the placer is free to move components across the chip and is allowed to accept low quality solutions hoping to explore more options and reach better designs. As temperature decreases, the placer's freedom in moving components to new locations become more restricted to local areas and the

probability of accepting low quality solutions decreases until eventually the floor-plan converges to an optimized solution. The common moves that placers apply are: moving a component to a new location, swapping two components, and changing the orientation of the component. The pseudo-code of a simulated annealing floor-planning algorithm is outlined in Figure 24.

Simulated Annealing Algorithm for Floorplanning

```
1. Get an initial floorplan S;   S_Best = S;
2. Get an initial temperature T > 0;
3. while not "frozen" do
4.     for i = 1 to k do
5.       Perturb the floorplan to get a neighboring S' from S;
6.       ΔC = cost(S') – cost(S);
7.       if ΔC ≤ 0 then    // down-hill move
8          S = S';
9.       else    // uphill move
10.        S = S' with the probability e^(-ΔC/T);
11.      end if
12.      if cost(S_Best) > cost(S) then
13.        (S_Best) = S;
14.      end if
15.   end for
16.   T = rT;    // reduce temperature
17. end while
18. return S_Best;
```

Figure 24 Simulated-annealing floor-planning (Wang, Chang, & Cheng, 2009)

The third floor-planning approach expresses the design goals and the positions of the components as analytical functions. This transforms the problem into a mathematical formulation whose solution could be found by solving the equations of the problem. One of the most popular algorithms of this approach is force-directed floor-planning which models the floor-planning problem as a set of forces existing between connected components. High connectivity is modeled as a strong force, and as connectivity between components gets lower, the force between these components becomes weaker. Components that exhibit a strong force

between them are pulled towards each other at the algorithm iterates, and therefore components that have a weak force joining them will be pushed farther from each other as illustrated in Figure 25.

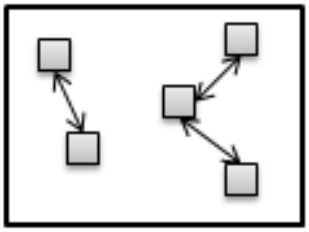

Figure 25 Force-directed floor-planning (Wang, Chang, & Cheng, 2009)

3.6 Thermal Modeling

Due to the significant impact of heat dissipation on modern chips, there has been an increased interest in formulating efficient and accurate thermal models to guide the process of VLSI design. The operating temperature in the chip can be estimated by equation 7. Where Ta is the ambient temperature, P_{total} is the total power consumed by the design, and R_{th} is the equivalent thermal resistance of the circuit. The equation shows a direct link between temperature, Power, and Technology. This portrays that power hungry designs such as high performance circuits will result in higher temperatures. Equation 7 also shows a dependence on the technology used in the fabrication of the chip. This relation has encouraged research into fabrication technologies which have low Thermal Resistance, but these strategies were found to have low cost efficiency. Cooling techniques work on reducing the ambient temperature term of equation 7 to counteract the increase in the second term. Another less accurate

estimate of operating temperature is presented in equation 8. In this equation, the thermal resistance is replaced by the inverse of the die size. This equation illustrates the relation between die size and temperature and explains why modern chips, characterized by their sum 100nm technologies, have seen a significant rise in temperature.

$$T_i = T_a + P_{total} \times R_{th} \quad (7)$$

$$T_i = T_a + \frac{P_{total}}{(10^5 \times die\ size)} \quad (8)$$

For a more accurate temperature calculation, the heat diffusion equation in equation 9 is used.

$$k \,.\, \nabla^2 T + g(x, y, z, t) = 0 \quad (9)$$

Where T is the temperature in °C, k is the thermal conductivity of the material in W/m°C, and g is the power density of the source in W/m^3. This relation illustrates further that the power dependence of temperature is not on the total power consumed but on the power densities across that chip. Thus, different densities results in temperature variations across the chip.

This equation is evaluated in several ways using thermal models, most popular of which are:

- Finite Difference Model (FDM) (Tsai & Kang, 2000)
- Finite Element Model (FEM) (Sabry, Bobtemps, Aubert, & Vahrmann, 1997)
- Compact Thermal Model (CTM) (Skadron, et al., 2003)

3.6.1 Finite Difference Model

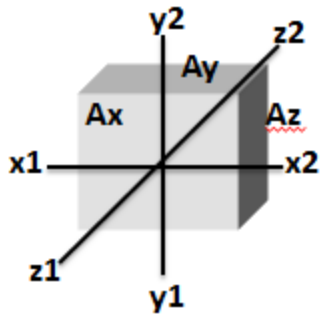

Figure 26 FDM Silicon block dimensions (Tsai & Kang, 2000)

Using FDM, equation 9 is solved as follows; consider a block of silicon with dimensions *dx dy* and *dz* as shown in Figure 26. The finite difference approximation for the spatial derivatives of equation 9 at a node *i* is calculated. This results in equation 10 where t_i is the temperature at node '*i*', t_{x1} and t_{x1} are the temperatures of the neighboring nodes in both x directions. Similarly t_{y1}, t_{y2}, t_{z1}, and t_{z2} are the temperatures of at node '*i*''s neighbors in the y and z directions. Furthermore, A_x is equal to *dy* by *dz*. A_y and A_z are similarly calculated. Equation 10 can be simplified into equation 11.

$$\frac{t_{x1} - t_i}{\delta x/kA_x} + \frac{t_{x2} - t_i}{\delta x/kA_x} + \frac{t_{y1} - t_i}{\delta y/kA_y} + \frac{t_{y2} - t_i}{\delta y/kA_y} + \frac{t_{z1} - t_i}{\delta z/kA_z} + \frac{t_{z2} - t_i}{\delta z/kA_z} + \Delta g(x,y,z,t) = 0 \quad (10)$$

$$(t_{x1} - t_{i)g_{i,x1}} + (t_{x2} - t_i)g_{i,x2} + (t_{y1} - t_i)g_{i,y1} + (t_{y2} - t_i)g_{i,y2}$$

$$+(t_{z1} - t_i)g_{i,z1} + (t_{z2} - t_i)g_{i,z2} = \Delta g(x,y,z,t) \quad (11)$$

where

$$g_{i,x} = \frac{kA_x}{\delta x}, g_{i,y} = \frac{kA_y}{\delta y}, g_{i,z} = \frac{kA_z}{\delta z}$$

At this point, the model takes advantage of the electrical/thermal duality to model temperatures as voltages, delta-g as current, and $g_{i,j}$ as thermal conductance. The model then calculates equation 7 for all nodes and uses Kirchhoff's Current Law to construct the matrix G shown in Figure 27. Where v is the node temperature, I is the power dissipation at each node and $g_{i,j}$ is the thermal conductances between nodes. According to this relation, matrix G is symmetric and positive definite. Local temperatures are then calculated by solving matrix G.

$$\boldsymbol{G}v = \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{n1} & g_{n2} & \cdots & g_{nn} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} i_1 \\ i_2 \\ \vdots \\ i_n \end{bmatrix} = \boldsymbol{i}$$

Figure 27 FDM G Matrix

Calculating nodal temperatures using the finite difference method is computationally expensive and it is usually inefficient to integrate it in temperature aware VLSI design algorithms.

3.6.2 Finite Element Model

The finite element model (FEM) is a numerical method in which the chip is divided into a finite number of geometrical elements. The temperature of each element is then calculated using equation 7. As the number of elements increases, the granularity of temperature calculation increases, and so does the computational complexity.

The elements FEM uses to mesh the circuit could have several three dimensional shapes. One of the best shapes to estimate temperature is the eight-point hexahedral, illustrated in Figure 28.

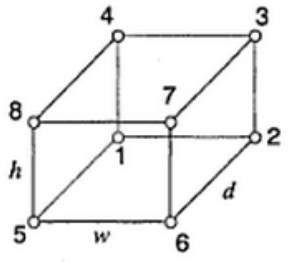

Figure 28 FEM silicon cube (Goplen B. , Advanced placement techniques for future VLSI circuits, 2006)

The temperatures at the nodes of the element are calculated using equation 5. The temperatures inside the element are interpolated based on the nodal temperatures using the tri-linear interpolation function presented in equation 12.

$$T(x,y,z) = \sum_{i=1}^{8} N_i t_i \qquad (12)$$

Where ti is the temperature at node *i* and N_i is the shape function of node *i* which depends on the dimensions of the element. N_i is calculated using equation 13.

$$N_i = (\frac{1}{2} + \frac{2(x_i - x_c)}{w^2}(x - x_c))(\frac{1}{2} + \frac{2(y_i - y_c)}{h^2}(y - y_c))(\frac{1}{2} + \frac{2(z_i - z_c)}{d^2}(z - z_c)) \quad (13)$$

Where x_c , y_c, and z_c are the coordinates of the center of the element and *h*, *w*, and *d* are the element's dimensions illustrated in Figure 24.

3.6.3 Compact Thermal Model (HotSpot)

The compact thermal model HotSpot developed by Skadron et al introduces a computationally efficient method for thermal modeling at higher levels. Instead of calculating the thermal matrix at a nodal level as done in FDM and FEM, Hotspot models the thermal resistances and capacitances between architectural components as shown in Figure 29. The model also takes into account environmental temperature contributors such as the thermal capacitance and resistance of the heat-spreader, heat-sink, and the air conviction current. Similar to FDM and FEM, Hotspot models temperature as voltage, and power dissipation as a current passing through a thermal resistor. Hotspot also calculates the thermal capacitance between the components. The calculation of thermal capacitance is necessary for determining the delay between a change in power and its effect on temperature. The thermal resistance is proportional to the thickness of the material and inversely proportional to its thermal conductivity and area. The thermal capacitance on the other hand is proportional to thickness, area, and the thermal capacitance per unit volume of the material as shown in equations 14

and 15. The temperatures of the chip's components are then determined by solving the resistive-capacitive network illustrated in Figure 29.

$$C = c \,.\, t \,.\, A \tag{14}$$

$$R = \frac{t}{x \,.\, A} \tag{15}$$

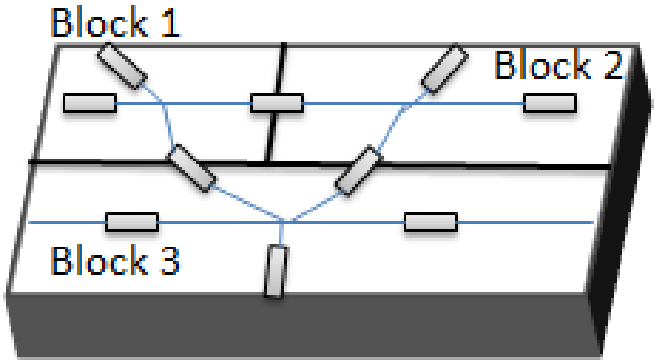

Figure 29 CTM RC network (Skadron, et al., 2003)

After discussing the relevant topics and methods, the KT2C algorithm is introduced in the next chapter. The chapter discusses how the algorithm modifies the regular VLSI design flow. Then, it discusses the design of the algorithm and its implementation.

Chapter Four

K-way Thermal Chip Clustering Algorithm

In this chapter we introduce the K-way Thermal Chip Clustering (KT2C) algorithm, a VLSI partitioning algorithm which uses clustering techniques to identify possible hotspots in the chip and distribute them among independent clusters. The algorithm then clusters the chip components upon these clusters while making an effort at minimizing the temperature of each block. The algorithm then iterates to improve the initial component distribution upon cluster centers by further minimizing temperature variations and reducing wire-length.

4.1 Overview of the new design flow

The K-way Thermal Chip Clustering algorithm has been integrated to the VLSI design flow as a stage between binding and floor-planning, as shown in Figure 30. Similar to the regular design flow, the DFG is inputted to high level synthesis where scheduling, resource allocation, and binding are performed. After binding, the switching activity and the power of each component of the design is calculated by running a trace of simulation values on that circuit. The powers are used by the clustering algorithm, and later on by the thermal model, in estimating the temperature of each component.

After the power simulation stage, the clustering algorithm divides the chip into a number of thermally optimized clusters. This strategy causes floor-planning to occur in two stages: in the

first stage, the positions of the components in each cluster are determined. Then, the floor-planner views the different clusters as blocks and determines their positions as well. And last, the final floor-plan of all the components is calculated taking into account the position and orientation of the block they belong to, and their position and orientation inside that block.

The final floor-plan and the powers of all the components are then inputted into the thermal simulator to calculate the temperature profile of the chip and verify the effectiveness of the clustering algorithm.

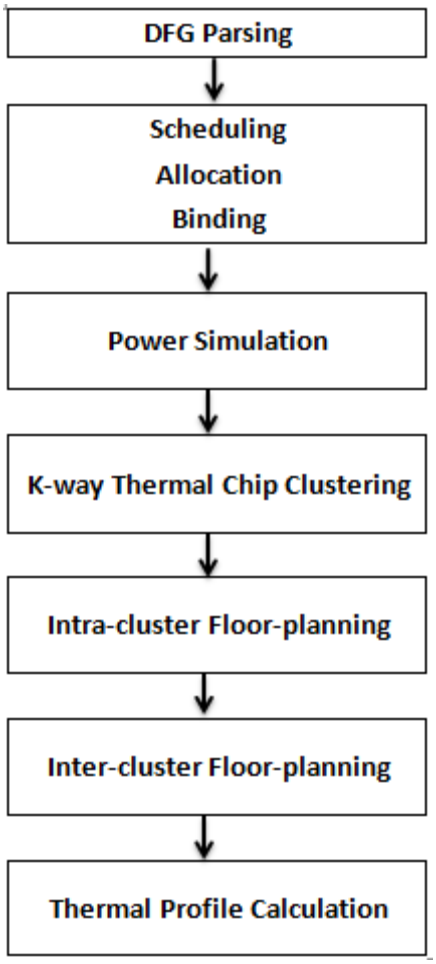

Figure 30 KT2C-integrated design flow

4.2 Algorithm Description

The K-way Thermal Chip Clustering algorithm operates in two phases, the initialization phase, and the iterative phase as shown in the pseudo-code of Figure 31 and described below

K-way Thermal Chip Clustering Algorithm

Initialization Phase:

1. Calculate the power-to-area ratio (PAR) of all components.
2. Identify the maximum and average PAR.
3. Cluster components into two groups, Hot and Cool, according to their distance from the maximum and average PARs
4. Components in the maximum PAR cluster become cluster centers
5. The cool components are distributed among the clusters probabilistically

Iterative Phase:

While (termination criteria not met)

Do

- With probability of 0.5 do displacement operation:
    a. Randomly pick a cluster C and a component P from a different cluster
    b. Evaluate the improvement of moving P to C
    c. If the improvement is positive accept, else roll back the move.
- Else do a swap move:
    a. Pick two components Pi and Pj from two different clusters
    b. Evaluate the improvement of swapping Pi and Pj
    c. If the improvement is positive accept, else roll back the move.

Repeat

6. Output final clusters

Figure 31 KT2C algorithm pseudo-code

Initialization Phase:

In this phase, the algorithm determines the components which will form the centers of the different clusters in the chip. Then the algorithm probabilistically divides the rest of the components upon these centers. The component's areas and powers are inputted into the algorithm and are used to estimate the temperature of each component. KT2C takes advantage

of the electrical-thermal duality explained in section 3.6 which models the thermal characteristics of a chip as an RC network. In this scenario, the initial temperature of each component is calculated as the product of the power dissipation of the component and its thermal resistance, as shown in equation 16. The thermal resistance of the component is given by equation 15. Since the chip thickness and the thermal conductivity of silicon is the same for all components, the algorithm uses the power to area ratio (PAR), given by equation 17, as an indicator for comparing temperature between components. This indictor implies that the higher the power to ratio area is, the higher the temperature would be.

$$Temperature = R_{th} \times Power \tag{16}$$

$$PAR = \frac{Power}{Area} \tag{17}$$

The initialization algorithm determines the maximum and average PARs of all components and then calculates the distance between the components' PARs and these two PARs. The distance between two powers is given by equation 18. If a component's PAR distance is found to be closer to the maximum PAR, the component is labeled as "hot" whereas if it was closer to the average PAR, the component is labeled as "cool". Eventually, the hot components will form the cluster centers, and the cold components will be distributed among these centers in an attempt to cool them.

$$Distance(i, j) = |PAR(i) - PAR(j)| \tag{18}$$

The cool components are assigned to clusters according to a compatibility probability between the cluster's center and the component. The compatibility between two components is proportional to the resultant temperature reduction from placing both components in a single cluster. In order to estimate the steady state temperature, the two components are modeled as two charged capacitors connected to each other. According to section 3.3 the thermal capacitance of a component is proportional to all; the thickness of the material, the thermal conductivity of the material, and the component's area as shown in equation 14. When connecting two charged capacitors the final temperature by the two is governed by equation 19, where T_i and T_j are the initial temperatures of components *i* and *j* respectively. T_i and T_j are calculated for each component using equation 16, and therefore equation 19 could be further simplified into equation 20. Since t and k are the same for all components, the final temperature could be estimated using equation 21.

$$T_{final}(i,j) = \frac{C_i T_i + C_j T_j}{C_i + C_j} \tag{19}$$

$$T_{final}(i,j) = \frac{t}{k} . \frac{P_i + P_j}{A_i + A_j} \tag{20}$$

$$T_{final}(i,j) \sim \frac{P_i + P_j}{A_i + A_j} \tag{21}$$

The compatibility between a component and a center is calculated as the final temperature reduction between the two as shown in equation 22. For each component, the compatibility probability between it and the different cluster centers is calculated as shown in equation 23.

$$Compatibility(i,j) = T_{max}(i,j) - T_{final}(i,j) \quad (22)$$

$$Compatibility\ Probability(i,j) = \frac{Compatibility(i,j)}{\sum_{k=0}^{\#Clusters} Compatibility(i,k)} \quad (23)$$

The cool components are distributed probabilistically between the thermal centers, with each component having the highest probability to be joined with the center it has the highest compatibility with.

Iterative phase:

This phase attempts to improve upon the initial clustering results by lowering the inter-cluster temperature variation and reducing the overall wire-length. In order to estimate the temperature of each cluster, it is modeled as a capacitor network formed by all the capacitances of the cluster's components. The temperature is estimated by equation 24 which is derived in the same way as equation 21.

$$T_{final}(C_i) \sim \frac{\sum_{j \in C_i} P_{j_i}}{\sum_{j \in C_i} A_{j_i}} \quad (24)$$

When the temperatures of all clusters are calculated, the algorithm choses randomly whether to carry out one of two possible operations: a displacement, or a swap. If a displacement operation is chosen, the algorithm picks one cool component and one destination cluster at random. The algorithm then evaluates the improvement due to moving the cool component to the destination cluster. If the improvement was found to be positive, the move is accepted and the component is moved. If a swap operation is chosen, the algorithm chooses two cool

components at random and evaluates the improvement due to swapping both components. If the improvement was found to be positive, the swap is carried out. The algorithm repeats this procedure until convergence is reached.

The improvement is given as the sum of two weighted terms, the inter-cluster temperature improvement, and the inter-cluster connectivity improvement as shown in equation 25.

$$Improvement = \alpha.Temperature\ Improvement + \beta.Connectivity\ Improvement \quad (25)$$

The inter-cluster temperature is calculated as the sum of all the differences between cluster temperatures as shown in equation 26.

$$InterCluster\ Temperature = T_{inter} = \sum_{i=0, j=0 \mid i \neq j}^{\#Clusters} \left| T_{final}(C_i) - T_{final}(C_j) \right| \quad (26)$$

The inter-cluster temperature improvement is then given by equation 27.

$$\text{Temperature Improvement } = \frac{T_{inter}(t) - T_{inter}(t+1)}{T_{inter}(t)} \quad (27)$$

The inter-cluster connectivity on the other hand is given as the number of nets between components in different clusters as shown in equation 28.

$$InterCluster\ Connectivity = Con_{inter} = \sum_{i=0, j=0 \mid i \neq j}^{\#Clusters} connections(i, j) \quad (28)$$

And therefore, the inter-cluster connectivity improvement is given by equation 29.

$$\text{Connectivity Improvement } = \frac{Con_{inter}(t) - Con_{inter}(t+1)}{Con_{inter}(t)} \quad (29)$$

The two terms of equation 25 usually work against each other implying that a positive improvement in connectivity usually results in a negative improvement in temperature. Alpha and beta are weight parameters used to control the relative significance of temperature and connectivity improvements. In all iterations, the centers remain fixed in their clusters because of the large fluctuations in inter-cluster temperature that results from moving them.

After convergence, the components undergo two stages of floor-planning as explained above. In the first stage, the components inside each cluster undergo floor-planning to form larger blocks. These blocks then undergo the second floor-planning stage where their positions are determined. Finally, the position of all components is outputted taking into consideration the position and orientation of the block they belong to, and their position and orientation inside that block. The K-way Thermal Chip Clustering algorithm could be applied with any floor-planning algorithm and does not require any unique floor-planning techniques.

4.3 Design Implementation

The overall design software was implemented in C++. It made use of several integrated open-source software to achieve the desired design flow.

The software parses DFGs secured from the Mediabench benchmark suite (ExPRESS Group, 2014) using a dot-format parser. A summary of the benchmarks used and their sizes is provided in Table 5.

Table 5 Benchmark summary

Benchmark	Number of Nodes	Number of Components
HAL	11	36
Horner Bezier	18	40
Elliptic Wave Filter	34	51
Motion Vectors	32	84
Cosine 1	66	93
Cosine 2	82	102
Feedback Points	53	106
Matrix Multiplication	109	188
BMP Header	106	256
Smooth Triangle	197	346

The software then precedes with the high level synthesis stages. It applies a latency constraint List scheduling algorithm to determine the schedule and number of resources to be used as explained in section 3.1.2. The software then applied a left-edge algorithm to perform functional unit and register binding as shown in section 3.1.3. After that, multiplexers are added where needed and the final net-list of the design is produced.

The components used in the circuit are extracted from a components library we synthesized using an 180nm technology using cadence tools. The steps for the synthesis procedure are described in section 4.5. The components library is composed of a set of Arithmetic Logic Units (ALUs), multipliers, registers, and multiplexers. The component library also describes the dimensions of each component in addition to its average dynamic and static powers, as summarized in Table 7.

The next step is to calculate the switching activities of the components of the circuit which will be used to calculate the dynamic power of each component. This is achieved by applying the technique discussed in section 3.3.3. Each DFG is run with a trace of 10,000 input patterns and

the average switching activity of each component is calculated. The software then makes use of the average dynamic power of each component, indicated by the component library, to calculate the dynamic power specific to that component. The dynamic power is added to the static power of that component to produce its total power. The total power of each component is outputted into a power trace file which will be used later on by the thermal simulator.

The power of each component, alongside with its area and net connections, are fed into the K-way Thermal Chip Clustering algorithm described in section 4.2 which will divide the overall chip into clusters of connected components. These components need to undergo two stages of floor-planning, both at the intra-cluster level and at the inter-cluster level. The floor-planning is performed using the Parquet fixed-outline floor-planner (Markov & Adya, 2003). The Parquet floor-planner takes a list of components and their net-list in bookshelf format. The tool then applies simulated annealing fixed-outline floor-planning to determine the position of the different components optimized for both area and wire-length.

After intra-cluster and inter-cluster floor-planning is performed, the final floor-plan of the chip is outputted into a floor-plan file. This file, along with the power trace file, will be used by the thermal simulator Hotspot (Skadron, et al., 2003) in calculating the thermal profile of the chip. As explained in section 3.6.3, Hotspot uses a compact thermal model to represent the chip as thermal RC network which is then used to calculate the temperature of each component. The thermal profile is used to validate the effectiveness of the KT2C algorithm. The final design flow of the software is illustrated in Figure 32.

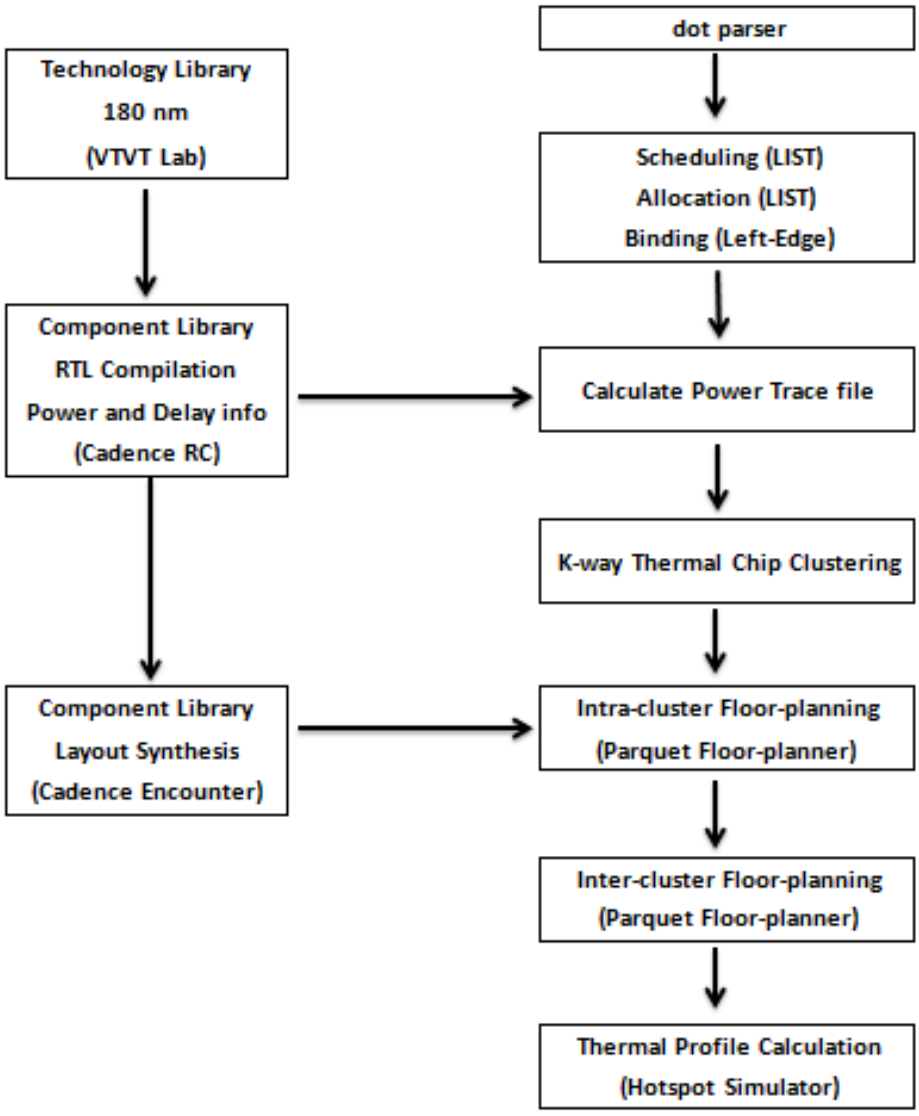

Figure 32 Implementation design flow

4.4 Parallelization Study

One the earliest use of circuit partitioning and VLSI clustering algorithms was to reduce the computational effort carried out by floor-planning and placement tools. This is due to two main reasons:

1. The circuit is divided into independent parts which could undergo the tasks of floor-planning or placement without being affected by what is taking place in other clusters.
2. The number of components in each partition is reduced and therefore the complexity of the floor-planning problem is reduced as well. For example, if the complexity of floor-planning is O(n!), n being the number of components, dividing the circuit into four equal parts would reduce the complexity into four O($\frac{n}{4}$!) tasks which could be running in parallel.

In this section we propose a parallelization scheme for the KT2C design flow through identifying the parts that are suitable for parallelism and we study the possible speedups resulting from implementing this parallel design.

Upon conducting a timing analysis of the KT2C design flows, it was apparent that the computational bottleneck that mainly contributed to the overall runtime was the floor-planning phase. More specifically, the largest runtime was experienced in the intra-clustering floor-planning phase which was taking up to 78.34% of the whole flow's execution time. This result was expected since floor-planning has the highest computational complexity among the other tasks. Moreover, the intra-cluster floor-planning stage has a higher computational demand than the inter-cluster floor-planning phase since the intra-cluster phase has to run several times. These runs of intra-cluster floor-planning seem attractive for parallelism especially due to their independence from each other. According to this parallelization, the new KT2C design flow will look as illustrated in Figure 33.

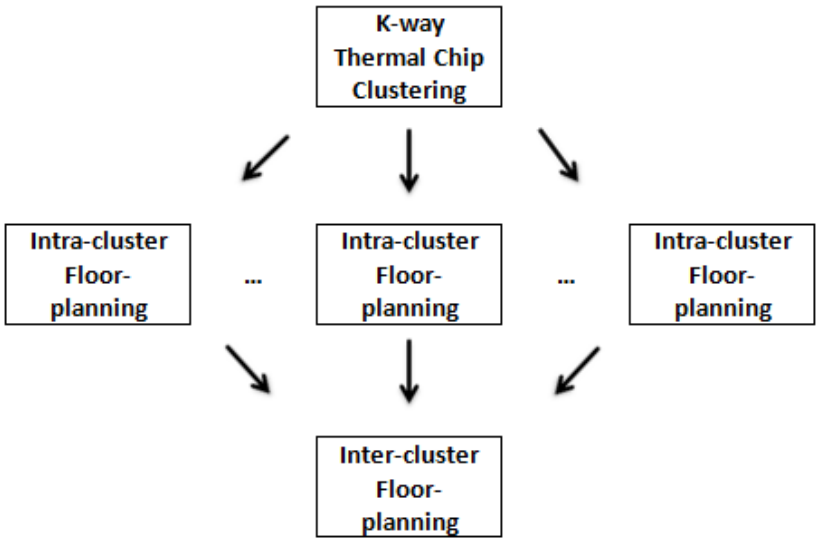

Figure 33 Parallel KT2C design flow

The efficiency of the parallelism of the KT2C design flow depends primarily on the way components are distributed among clusters. The number of components directly correlates with the execution time of the floor-planning task of these components. Therefore, an uneven distribution of components would result in an uneven execution time between intra-cluster floor-planning tasks and thus the parallelism would have a low efficiency. To further explain this idea, consider this example: The clustering algorithm divided the components into four clusters with one cluster having a large share of components while the other clusters had only few components. Consider the overall execution time was 10 minutes and the execution time for the largest cluster alone was 7 minutes while the other clusters had an execution time of around 1 minute. Although the small clusters finished in a very small time, the speedup of the algorithm due to parallelization would be measured according to the largest execution time of 7 minutes causing a lower efficiency compared to another scenario where all clusters have equivalent execution times close to 2 minutes.

In order to evaluate intra-cluster floor-planning's suitability for parallelism, the execution times of the intra-clustering stage over all benchmarks is reported in Table 6. The experiments were conducted on an Intel Core i7 4-core machine running at 2.2 GHz with 8 GB of RAM.

Due to the probabilistic initialization phase, KT2C could have different clustering results on each run and thus different execution times. To account for this, the program was run 10 times with different random number generator seed values. The average execution time of the 10 runs over all benchmarks for the intra-clustering phase is reported in Table 6. Similarly the execution time of the slowest cluster, resulting in the largest execution time is reported. The theoretical speedup is calculated as the total intra-clustering execution time divided by the slowest cluster execution time. The standard deviation over the 10 runs for both total and maximum execution times was around 0.1 seconds. The reported results were calculated through running the algorithm sequentially on one core. The reported speedup is ideally achievable if the number of execution cores is equal to the number of clusters in each benchmark. As shown in Table 6, parallelism is very attractive since theoretical speedups could reach up to 40 times at its best and up to 5.65 times at its worst. Either way, it would be very advantageous to introduce parallelism into the KT2C design flow.

Furthermore, and to verify the algorithm's suitability for parallelization, the program was modified in order to run the intra-clustering phase in parallel using multiple cores by adding OpenMP directives. The execution times using 2, 3, and 4 cores are reported in Table 7. The corresponding speedups are reported in Table 8. As in Table 6, the program was run 10 times with different seed values to account for randomness in the algorithm. The reported results

show that the execution time of the program is significantly reduced as the number of cores is increased which indicates that KT2C is well suited for parallelism. The results show that the speedup does not increase linearly as the number of cores increase. This overhead is due to the formation of bottlenecks in memory and bus access between multiple cores.

Table 6 Intra-clustering execution times in seconds

Benchmark	Intra-Clustering Total Time	Slowest Cluster Time	Speedup
Elliptic Wave Filter	3.1	0.34	9.12
Cosine 1	5.7	0.41	13.90
Cosine 2	6.2	0.43	14.42
Feedback Points	6.5	0.46	14.13
Horner Bezier	2.6	0.46	5.65
Matrix Multiplication	11	0.4	27.50
BMP Header	18	0.67	26.87
Smooth Triangle	21	0.52	40.38
Motion Vectors	4.9	0.26	18.85
HAL	2.2	0.35	6.29

Table 7 Parallel vs. sequential intra-clustering execution times in seconds

Benchmark	Sequential	2 Cores	3 Cores	4 Cores
Elliptic Wave Filter	3.1	1.6	1.2	1
Cosine 1	5.7	2.9	2.2	1.7
Cosine 2	6.2	3.12	2.2	1.9
Feedback Points	6.5	3.3	2.4	2
Horner Bezier	2.6	1.4	1.1	0.92
Matrix Multiplication	11	5.52	4	3.2
BMP Header	18	9.05	6.4	5.2
Smooth Triangle	21	11	7.8	5.9
Motion Vectors	4.9	2.5	1.8	1.5
HAL	2.2	1.2	0.81	0.88

Table 8 Parallel vs. sequential intra-clustering speedups

Benchmark	2 Cores	3 Cores	4 Cores
Elliptic Wave Filter	1.94	2.58	3.1
Cosine 1	1.97	2.59	3.35
Cosine 2	1.99	2.82	3.26
Feedback Points	1.97	2.71	3.25
Horner Bezier	1.86	2.36	2.83
Matrix Multiplication	1.99	2.75	3.44
BMP Header	1.99	2.81	3.46
Smooth Triangle	1.91	2.69	3.56
Motion Vectors	1.96	2.72	3.27
HAL	1.83	2.72	2.5
Average	**1.94**	**2.68**	**3.2**

4.5 Component Library Synthesis

The components library is used to provide area and power estimates to different parts of the design flow, namely, the classification and thermal simulation. The components library is also used to provide the component dimensions and pin connections to the floor-planning stage.

Table 9 Components Information

Component	Area (um)	Delay (ps)	Leakage Power (nW)	Dynamic Power (uW)	Total Power (uW)
Alu	6378	7629	3.43	122.44	122.45
Multiplier	14987	7547	2.47	312.68	312.68
Register	1764	9568	22.63	141.71	141.73
Mux2	588	9353	0.47	19.75	19.75
Mux4	1820	9119	0.74	32.45	32.46
Mux8	4135	8872	1.78	71.17	71.17
Mux16	8113	8477	3.29	115.28	115.28

The component library was synthesized using an 180nm technology standard cell library secured from Virginia Tech (Sulistyo, Perry, & Ha, 2003) (Ha, Sulistyo, & Dong, 2002). The synthesis process was done in two stages; the RTL compilation stage, and the Layout synthesis stage. Cadence tools were used in both stages as described below. The components produced by the synthesis process, along with their details are summarized in Table 9.

4.5.1 RTL Compilation

The cadence Encounter RTL compiler is used to transform a behavioral description of the component into a structural implementation. To illustrate this procedure, consider the case of an 8-bit register whose Verilog code is specified in Figure 34

```
1  module register( in, out, clk, rst);
2  input [7:0] in;
3  input clk, rst;
4  output reg [7:0] out;
5  always @ (posedge clk or negedge rst)
6  begin
7  if(rst == 0)
8          out = 0;
9  else
10         out = in;
11 end
12 endmodule
```

Figure 34 Register behavioral Verilog code

The Encounter RTL compiler takes this behavioral Verilog code, along with the standard cell library specifications in order to produce an RTL design of the code as shown in figure 35 which is then transformed into a structural Verilog code, as shown in figure 36. The software also outputs gate level estimates of the area, power, and timing of the design.

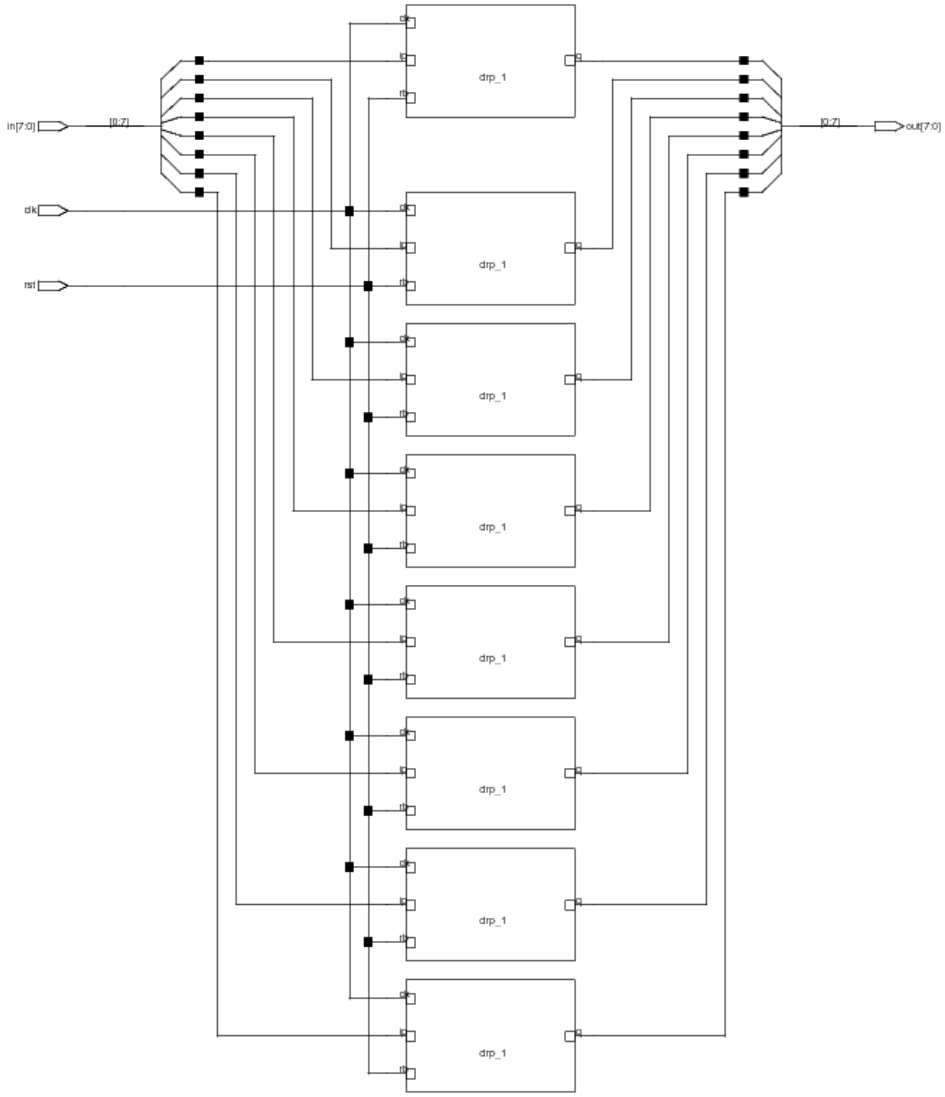

Figure 35 Register RTL design

```
 2 // Generated by Cadence Encounter(R) RTL Compiler RC12.22 - v12.20-s014_1
 3
 4 // Verification Directory fv/register
 5
 6 module register(in, out, clk, rst);
 7   input [7:0] in;
 8   input clk, rst;
 9   output [7:0] out;
10   wire [7:0] in;
11   wire clk, rst;
12   wire [7:0] out;
13   drp_1 \out_reg[7] (.rb (rst), .ck (clk), .ip (in[7]), .q (out[7]));
14   drp_1 \out_reg[6] (.rb (rst), .ck (clk), .ip (in[6]), .q (out[6]));
15   drp_1 \out_reg[4] (.rb (rst), .ck (clk), .ip (in[4]), .q (out[4]));
16   drp_1 \out_reg[0] (.rb (rst), .ck (clk), .ip (in[0]), .q (out[0]));
17   drp_1 \out_reg[3] (.rb (rst), .ck (clk), .ip (in[3]), .q (out[3]));
18   drp_1 \out_reg[5] (.rb (rst), .ck (clk), .ip (in[5]), .q (out[5]));
19   drp_1 \out_reg[2] (.rb (rst), .ck (clk), .ip (in[2]), .q (out[2]));
20   drp_1 \out_reg[1] (.rb (rst), .ck (clk), .ip (in[1]), .q (out[1]));
21 endmodule
```

Figure 36 Register structural Verilog code

4.5.2 Layout Synthesis

The structural Verilog code and the generated timing file are then fed into Cadence Encounter to generate the layout of the intended component. This software applies floor-planning, standard-cell placement, and routing algorithms in producing an optimized layout of the component as shown in figure 37.

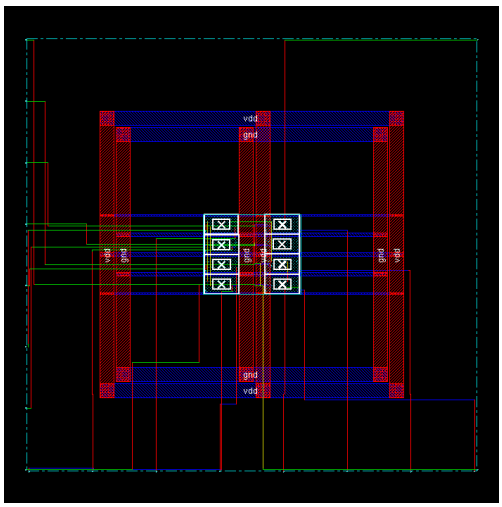

Figure 37 Register layout

Chapter Five

Final Results

The KT2C algorithm has been tested for two scenarios. In the first scenario, the algorithm was run to optimize for temperature in the iterative phase. In the second scenario, the algorithm optimized for wire-length in the iterative phase. This was achieved by setting the values of alpha and beta to one and zero respectively for the first scenario, and to zero and one in the second scenario, respectively.

In order to assess the improvement achieved by the algorithm in both scenarios, a third run was performed where no clustering was applied. The results achieved by the no-clustering run are summarized in Table 10. The results of the temperature optimized run and the wire-length optimized run are summarized in Tables 11 and 13 respectively.

Furthermore, Table 12 reports the improvement achieved by temperature optimization run versus the no clustering run. The results show a reduction in the average chip temperature reaching up to 13.61°C and with an average of 10.87°C throughout all benchmarks. More notably, the temperature reduction algorithm also achieved a large reduction in peak temperature reaching up to 16.5°C and with an average of 13.47°C throughout all benchmarks. This improvement was achieved with an average area increase of 8.67% and an average wire-length increase of 24.77%. The significant decrease in temperature was achieved because of two reasons; first, the hottest components in the chip were spread apart, preventing the

formation of hotspots. The second reason is due to the iterative improvement algorithm which rigorously moves components to achieve uniform temperature dissipation across the chip. Increasing the area alone in the chip does not guarantee temperature reduction. Even with relaxed area constraints, the floor-planner might place two hot components next to each other and thus forming a hotspot. These hotspots will raise the peak temperature and eventually will pull the average temperature of the chip to a higher level. The techniques in the KT2C algorithm insure that hot components are spread away from each other and thus minimizing peak and average temperature. The area increase in KT2C is due to the accumulation of white spaces inside the blocks. In inter-cluster floor-planning, the floor-planning tool views the cluster blocks as rectangular blocks and thus cannot make use of the empty spaces inside the block. The relatively large wire-length increase is due to the fact that the algorithm is moving components between clusters without any account for the increase in connections between blocks. This problem was solved in the wire-length optimization run of the algorithm whose improvement results over the no-clustering run are reported in Table 14.

Table 14 shows a significant improvement in wire-length compared to the no-clustering run. Instead of degrading the wire-length as a cost for temperature optimization, this run gave better wire-length results than the no-clustering run throughout all benchmarks with an average improvement of 25%, reaching up to 44%. This result was possible because the algorithm places highly interconnected components in the same cluster and tries to minimize inter-cluster connections. Eventually, components with high connectivity will be placed near each other and thus significantly minimizing the wire-length. This improvement in wire-length

did not negate the improvement in temperature. The wire-length optimization run had an average temperature improvement with an average of 11.41°C throughout all benchmarks and a maximum temperature improvement with an average of 10.78°C in all benchmarks. These improvements were achieved at a price of an average increase in area of 6.78%.

A comparison between the temperature optimization run and the wire-length optimization run is presented in Table 15. The results show the wire-length run had an average 41% improvement in wire-length compared to the temperature optimization run, with a cost of 6.41°C increase in maximum temperature. The increase in maximum temperature is due to placing connected components with low temperature compatibility next to each other and thus forming hotter regions.

Table 10 No-clustering results

Benchmark	Area (um^2)	Wire Length (um)	Average Temperature(°C)	Max Temperature (°C)
Cosine 1	578419	26.60	171.24	179.86
Cosine 2	576318.6	26.00	178.13	188.51
Elliptic Wave Filter	245138.3	9.81	175.68	182.37
HAL	138430.4	1.79	192.05	197.33
Feedback Points	421044	21.40	192.85	200.56
Horner Bezier	178979.5	3.48	179.78	187.77
Matrix Multiplication	807087	82.20	194.04	203.86
Motion Vectors	298179.2	10.80	204.28	213.12
Smooth Triangle	1345401	252.00	205.26	218.94
BMP Header	1153006	101.00	179.03	190.64

Table 11 Temperature-optimized KT2C results

Benchmark	Area (um^2)	Wire Length (um)	Average Temperature(°C)	Max Temperature (°C)
Cosine 1	613056	32.20	163.91	169.73
Cosine 2	622073.8	33.90	168.26	173.97
Elliptic Wave Filter	268279.5	11.80	164.54	170.30
HAL	154173	2.45	178.43	185.45
Feedback Points	458163.5	25.50	181.41	190.36
Horner Bezier	194685.5	5.13	169.39	175.81
Matrix Multiplication	877443.5	94.60	183.31	191.93
Motion Vectors	326453.5	13.60	191.23	199.65
Smooth Triangle	1461600	294.00	193.17	202.43
BMP Header	1239827	116.00	169.94	181.67

Table 12 Temperature-optimized KT2C improvement over no-clustering

Benchmark	Area Increase (%)	Wire Length Decrease (%)	Average Temperature Reduction (°C)	Max Temperature Reduction (°C)
Cosine 1	5.99	-21.32	7.33	10.13
Cosine 2	7.94	-30.59	9.87	14.54
Elliptic Wave Filter	9.44	-20.50	11.14	12.07
HAL	11.37	-37.12	13.61	11.88
Feedback Points	8.82	-19.07	11.44	10.21
Horner Bezier	8.78	-47.38	10.38	11.96
Matrix Multiplication	8.72	-15.07	10.74	11.93
Motion Vectors	9.48	-25.50	13.06	13.47
Smooth Triangle	8.64	-16.67	12.09	16.50
BMP Header	7.53	-14.46	9.09	8.97
average	8.67	-24.77	10.87	12.17
max	5.99	-14.46	13.61	16.50
min	11.37	-47.38	7.33	8.97

Table 13 Wire-length-optimized KT2C results

Benchmark	Area (um^2)	Wire Length (um)	Average Temperature(°C)	Max Temperature (°C)
Cosine 1	603408.3	18.60	165.48	174.24
Cosine 2	607372	20.70	170.53	179.70
Elliptic Wave Filter	267354	6.87	165.07	171.59
HAL	147596.8	1.71	182.52	189.24
Feedback Points	447629	13.20	184.22	192.05
Horner Bezier	192863.3	2.75	170.57	177.14
Matrix Multiplication	856645.3	79.10	186.02	211.09
Motion Vectors	314515.8	8.35	196.18	211.61
Smooth Triangle	1439884	146.00	195.05	212.68
BMP Header	1264262	56.80	167.62	186.06

Table 14 Wire-length-optimized KT2C improvement over no-clustering

Benchmark	Area Increase (%)	Wire Length Decrease (%)	Average Temperature Reduction (°C)	Max Temperature Reduction (°C)
Cosine 1	4.32	29.82	5.76	5.62
Cosine 2	5.39	20.13	7.60	8.81
Elliptic Wave Filter	9.06	29.91	10.61	10.78
HAL	6.62	4.33	9.52	8.09
Feedback Points	6.31	38.66	8.64	8.52
Horner Bezier	7.76	20.90	9.21	10.63
Matrix Multiplication	6.14	3.87	8.03	-7.23
Motion Vectors	5.48	23.01	8.10	1.51
Smooth Triangle	7.02	42.15	10.20	6.25
BMP Header	9.65	43.98	11.41	4.58
average	6.78	25.68	8.91	5.76
max	4.32	43.98	11.41	10.78
min	9.65	3.87	5.76	-7.23

Table 15 Wire-length-optimized KT2C improvement over temperature-optimized KT2C

Benchmark	Area Decrease (%)	Wire Length Decrease (%)	Average Temperature Reduction (°C)	Max Temperature Reduction (°C)
Cosine 1	1.57	42.15	-1.57	-4.51
Cosine 2	2.36	38.84	-2.27	-5.73
Elliptic Wave Filter	0.34	41.83	-0.53	-1.29
HAL	4.27	30.23	-4.09	-3.79
Feedback Points	2.30	48.48	-2.80	-1.69
Horner Bezier	0.94	46.33	-1.17	-1.34
Matrix Multiplication	2.37	16.46	-2.71	-19.16
Motion Vectors	3.66	38.65	-4.96	-11.96
Smooth Triangle	1.49	50.42	-1.89	-10.25
BMP Header	-1.97	51.05	2.32	-4.39
average	1.73	40.44	-1.97	-6.41
max	4.27	51.05	2.32	-1.29
min	-1.97	16.46	-4.96	-19.16

Chapter Six

Conclusion

We introduced the K-way Thermal Chip Clustering (KT2C) algorithm, a VLSI chip partitioning algorithm used to reduce chip temperature and prevent the formation of hotspots. The clustering algorithm was integrated in the design flow between the stages of binding and floor-planning. KT2C uses the power and area data to identify the hottest components in the chip and then allocates them to different clusters. The cold components are then distributed among these clusters in a probabilistic manner. The algorithm then iterates to further decrease temperature and reduce wire-length. The components undergo two stages of floor-planning; and intra-cluster stage where the positions of the components inside each cluster are determined, and an inter-cluster stage where the positions of each cluster block is determined on the chip. The temperature profile of the resulting chip was calculated using the thermal simulator “Hotspot” and was compared to the temperature profile of a design achieved without thermal clustering. The results show significant reduction in temperature reaching 13.6°C for average temperature and 16.5°C for peak temperature for an average area increase of 6%. Furthermore, with the wire-length-aware version of KT2C not only was wire-length increase prevented, but the algorithm improved total wire-length compared to the no-clustering solution resulting with an average 25.68% improvement in total wire-length over all benchmarks. Finally, the algorithm’s suitability for parallelization was studied. The study

showed that the intra-clustering phase of the algorithm is attractive for parallelization and significant speedups were attained upon running a parallel implementation of the algorithm.

Bibliography

Bogliolo, A., Benini, L., Ricco, B., & DeMicheli, G. (1999). Efficient switching activity computation during high-level synthesis of control-dominated designs. *Proceedings of the International Symposium on Low Power Electronics and Design. Monterey*.

Cao, L., Krusius, J., Korhonen, M., & Fisher, T. (1998). Transient thermal management of portable electronics using heat storage and dynamic power dissipation control. *IEEE Transactions on Components, Packaging, and Manufacturing Technology, 21*(1), 113–123.

Chen, G., & Sapatnekar, S. (2003). Partition-driven standard cell thermal placement. *Proceedings of the International Symposium on Physical Design. Monterey*.

Chu, C. C., & Wong, D. F. (1998). A matrix synthesis approach to thermal placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 17*(11), 1166 - 1174.

Cong, J., Wei, J., & Zhang, Y. (2004). A thermal-driven floorplanning algorithm for 3D ICs. *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design. San Jose*.

ExPRESS Group. (2014). *Benchmarks*. Retrieved from Extensible, Programmable and Reconfigurable Embedded SystemS Group: http://express.ece.ucsb.edu/benchmark/

Gajski, D., Dutt, N., Wu, A., & Lin, S. (1992). *High-level synthesis.* Boston: Kluwer Academic Publishers.

Goplen, B. (2003). Efficient thermal placement of standard cells in 3D ICs using a force directed approach. *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design. San Jose*.

Goplen, B. (2006). *Advanced placement techniques for future VLSI circuits* (Doctoral dissertation, University of Minnesota, Minneapolis). Retrieved from http://www-mount.ece.umn.edu/~sachin/Theses/BrentGoplen.pdf

Goplen, B., & Sapatnekar, S. (2007). Placement of 3D ICs with thermal and interlayer via considerations. *Proceedings of the 44th ACM/IEEE Design Automation Conference. San Diego*.

Gunther, S., Binns, F., Carmean, D. M., & Hall, J. C. (2001). Managing the impact of increasing microprocessor power consumption. *Intel Technology Journal, 5*(1), 1-9.

Gurrum, S. P., Joshi, Y. K., King, W. P., Ramakrishna, K., & Gall, M. (2008). A compact approach to on-chip interconnect heat conduction modeling. *ASME Journal of Electronic Packaging, 130*(3), 1-8.

Ha, J., Sulistyo, B., & Dong, S. (2002). A new characterization method for delay and power dissipation of standard library cells. *VLSI Design, 15*(3), 667-678.

Huang, W., Renau, J., Yoo, S.-M., & Torellas, J. (2000). A framework for dynamic energy efficiency and temperature management. *Proceedings of the 33rd International Symposium on Microarchitecture. Monterey*.

Huang, W., Stan, M. R., Skadron, K., Sankaranarayanan, K., Ghosh, S., & Velusamy, S. (2004). Compact thermal modeling for temperature-aware design. *Proceedings of the 41st Annual Design Automation Conference. New York*.

Hung, W.-l., Xie, Y., Vijaykrishnan, N., Addo-quaye, C., Theocharides, T., & Irwin, M. J. (2005). Thermal-aware floorplanning using genetic algorithms. *Proceedings of the 7th International Symposium on Quality Electronic Design. San Jose*.

International Technology Roadmap for Semiconductors. (2001). ITRS 2001 edition. Retrieved from The International Technology Roadmap for Semiconductors: http://www.itrs.net/Links/2001ITRS/Home.htm

International Technology Roadmap for Semiconductors. (2003). ITRS 2003 edition. Retrieved from The International Technology Roadmap for Semiconductors: http://www.itrs.net/Links/2003ITRS/Home2003.htm

Jeppson, N., & Hedenstierna, K. (1987). CMOS circuit speed and buffer optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 6*(3), 270-281.

Kahngm, A., Lienig, J., Markov, I., & Hu, J. (2011). *VLSI physical design: From graph partitioning to timing closure.* New York: Springer.

Kim, T., & Lim, P. (2006). Thermal-aware high-level synthesis based on network flow method. *Proceedings of the 4th International Conference on Hardware/Software Codesign and System Synthesis. Seoul*.

Krishnan, S. K. (2006). A genetic algorithm for the design space exploration of datapaths during high-level synthesis. *IEEE Transactions on Evolutionary Computation, 10*(3), 213 - 229.

Krishnan, V., & Katkoori, S. (2010). TABS: Temperature-aware layout-driven behavioral synthesis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 18*(12), 1649 - 1659.

Liu, W., Nannarelli, A., Calimera, A., Macii, E., & Poncino, M. (2010). Post-placement temperature reduction techniques. *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition. Dresden*.

Liu, Z., Bian, J., & Zhou, Q. (2007). A Thermal-aware ILP-based algorithm in behavioral synthesis. *Proceedings of the 7th International Conference on Application Specific Integrated Circuits. Guilin*.

Markov, S. N., & Adya, I. L. (2003). Fixed-outline floorplanning : Enabling hierarchical design. *IEEE Transactions on VLSI Systems, 11*(6), 1120-1135.

Martonosi, D., & Brooks, M. (2001). Dynamic thermal management for highperformance microprocessors. *Proceedings of the 7th International Symposium on High-Performance Computer Architecture. Nuevo Leone*.

Mukherjee, R., & Memik, S. O. (2006). An integrated approach to thermal management in high-level synthesis. *IEEE Transactions on VLSI, 14*(11), 1165 - 1174.

Mukherjee, R., Memik, S. O., & Memik, G. (2005). Peak temperature control and leakage reduction during binding. *Proceedings of the International Symposium on Low Power Electronics and Design. San Diego*.

Pedram, M., & Vaishnav, H. (1997). Power optimization in VLSI layout: A Survey. *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology, 15*, 221--232.

Rajaraman, A., Leskovec, J., & Ullman, J. D. (2011). *Mining of massive datasets.* Cambridge: Cambridge University Press.

Sabry, M.-N., Bobtemps, A., Aubert, V., & Vahrmann, R. (1997). Realistic and efficient simulation of electro-thermal effects in VLSI circuits. *IEEE Transactions on VLSI Systems, 5*, 283–289.

Sankaranarayanan, K., Velusamy, S., Stan, M., L, C., & Skadron, K. (2005). A case for thermal-aware floorplanning at the microarchitectural level. *Journal of Instruction-Level Parallelism, 7*, 1-16.

Schafer, B., & Kim, T. (2008). Hotspots elimination and temperature flattening in VLSI circuits. *IEEE Transactions on VLSI Systems, 16*(11), 1475 - 1487.

Skadron, K., Stan, M. R., Huang, W., Velusamy, S., Sankaranarayanan, K., & Tarjan, D. (2003). Temperature-aware microarchitecture. *Proceedings of the 30th Annual International Symposium on Computer Architecture. San Diego*.

Sulistyo, J. B., Perry, J., & Ha, D. S. (2003). *Developing standard cells for TSMC 0.25um technology under MOSIS DEEP rules.* Blacksburg: Department of Electrical and Computer Engineering, Virginia Tech. Retrieved from http://www.vtvt.ece.vt.edu/tutorial/references/tech_report.pdf

Tan, P.-N., Steinbach, M., & Kumar, V. (2006). *Introduction to data mining.* Boston: Addison-Wesley.

Teodorovic, D., Lucic, P., Markovic, G., & Dell' Orco, M. (2006). Bee colony optimization: Principles and applications. *Proceedings of the 8th Seminar on Neural Network Applications in Electrical Engineering. Belgrade*.

Tsai, C.-H., & Kang, S.-M. (2000). Cell-level placement for improving substrate thermal distribution. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems , 19*(2), 253 - 266.

Turgis, S., Azemard, N., & Auvergne., D. (1995). Explicit evaluation of short circuit power dissipation for CMOS logic structures. *Proceedings of the International Symposium on Low Power Design. Dana Point*.

Veendrick., H. J. (1984). Short-circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits. *IEEE Journal of Solid State Circuits, 19*, 468–473.

Wang, L.-T., Chang, Y.-W., & Cheng, K.-T. (2009). *Electronic design automation: Synthesis, verification, and test.* Massachusetts: Morgan Kaufmann.

Yan, H., Zhou, Q., & Hong, X. (2008). Efficient thermal-aware placement approach integrated with 3D DCT placement algorithm. *Proceedings of the 9th International Symposium on Quality Electronic Design. Beijing*.

Yan, H., Zhou, Q., & Hong, X. (2009). Thermal aware placement in 3D ICs using quadratic uniformity modeling approach. *Integration, 42*(2), 175-180.

Yu, J., Zhou, Q., & Bian, J. (2008). Exploiting thermal-area tradeoffs in high-level synthesis through resources number selection. *Proceedings of the 2008 International Conference on Communications, Circuits and Systems. Fujian*.

Yu, J., Zhou, Q., & Bian, J. (2009). Peak temperature control in thermal-aware behavioral synthesis through allocating the number of resources. *Proceedings of the Asia and South Pacific Design Automation Conference. Yokohama*.

Zhenyu, G., Yang, Y., Wang, J., Dick, R., & Shang, L. (2006). TAPHS: Thermal-aware unified physical-level and high-level synthesis. *Proceedings of the Asia and South Pacific Design Automation Conference. Yokohama*.